



AFRL-RI-RS-TR-2022-093

PRIVACY COGNIZANT IOT ENVIRONMENT FOR THE BRANDEIS PROGRAM

UNIVERSITY OF CALIFORNIA, IRVINE

JUNE 2021

FINAL TECHNICAL REPORT

APPROVED FOR PUBLIC RELEASE; DISTRIBUTION UNLIMITED

STINFO COPY

**AIR FORCE RESEARCH LABORATORY
INFORMATION DIRECTORATE**

NOTICE AND SIGNATURE PAGE

Using Government drawings, specifications, or other data included in this document for any purpose other than Government procurement does not in any way obligate the U.S. Government. The fact that the Government formulated or supplied the drawings, specifications, or other data does not license the holder or any other person or corporation; or convey any rights or permission to manufacture, use, or sell any patented invention that may relate to them.

This report was cleared for public release by the 88th ABW, Wright-Patterson AFB Public Affairs Office and is available to the general public, including foreign nationals. Copies may be obtained from the Defense Technical Information Center (DTIC) (<http://www.dtic.mil>).

AFRL-RI-RS-TR-2022-093 HAS BEEN REVIEWED AND IS APPROVED FOR PUBLICATION IN ACCORDANCE WITH ASSIGNED DISTRIBUTION STATEMENT.

FOR THE CHIEF ENGINEER:

/ S /

CARL R. THOMAS
Work Unit Manager

/ S /

GREGORY J. HADYNSKI
Assistant Technical Advisor
Computing & Communications Division
Information Directorate

This report is published in the interest of scientific and technical information exchange, and its publication does not constitute the Government's approval or disapproval of its ideas or findings

REPORT DOCUMENTATION PAGE

PLEASE DO NOT RETURN YOUR FORM TO THE ABOVE ORGANIZATION.

| | | | |
|--|--------------------------------|--|---|
| 1. REPORT DATE | 2. REPORT TYPE | 3. DATES COVERED | |
| JUNE 2022 | FINAL TECHNICAL REPORT | START DATE OCTOBER 2015 | END DATE MARCH 2021 |
| 4. TITLE AND SUBTITLE PRIVACY COGNIZANT IOT ENVIRONMENT FOR THE BRANDEIS PROGRAM | | | |
| 5a. CONTRACT NUMBER FA8750-16-2-0021 | 5b. GRANT NUMBER N/A | 5c. PROGRAM ELEMENT NUMBER 62303E | |
| 5d. PROJECT NUMBER | 5e. TASK NUMBER | 5f. WORK UNIT NUMBER R1TY | |
| 6. AUTHOR(S) Sharad Mehrotra, Nalini Venkatasubramanian, Alfred Kobsa, Christopher Davison, Shantanu Sharma, Roberto Yus, Nisha Panwar, Georgios Bouloukakis, Dhurbajyoti Ghosh, Peeyush Gupta, Yiming Lin, Primal Pappachan, Guoxi Wang, Eunjeong Shin, Andrew Chio, Raj Rajgopal, Valerie Guralnik, Phillip Lee | | | |
| 7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES) PRIME University of California, Irvine 141 Innovation Dr. Suite 250 Irvine CA 92617-3213 Sub Honeywell Automation and Control Solutions 1985 Douglas Dr. N. Golden Valley MN 55422 | | | 8. PERFORMING ORGANIZATION REPORT NUMBER |
| 9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES) Air Force Research Laboratory/RITA 525 Brooks Road Rome NY 13441-4505 | | 10. SPONSOR/MONITOR'S ACRONYM(S) AFRL/RI | 11. SPONSOR/MONITOR'S REPORT NUMBER(S) AFRL-RI-RS-TR-2022-093 |
| 12. DISTRIBUTION/AVAILABILITY STATEMENT Approved for Public Release; Distribution Unlimited. PA# AFRL-2022-3057 Date Cleared: 28 June 2022 | | | |
| 13. SUPPLEMENTARY NOTES | | | |
| 14. ABSTRACT The project created an IoT smart space testbed, entitled TIPPERS, with a plug-n-play architecture to infuse diverse privacy enhancing technologies. This includes specification and enforcement of privacy policies, differential privacy, and secure computation using cryptographic approaches including MPC and secret sharing. Research explored novel ways to scaling privacy technologies to large data volume and usage scenario complexities, and to issues that arise in deploying such technologies in the real-world including challenges due to trust. TIPPERS has been transitioned to navy where it was a recipient of the NAVWAR innovations award in 2021 and part of the TRIDENT Warrior Exercises in 2019 and 2020. | | | |
| 15. SUBJECT TERMS Privacy, IoT, Data Management, Differential Privacy, Fine-grained Policy, Access Control, Security, Encrypted Search, Smart Space, Privacy by design, evaluation of privacy enhancing technologies, Attestation | | | |
| 16. SECURITY CLASSIFICATION OF: | | | 17. LIMITATION OF ABSTRACT |
| a. REPORT U | b. ABSTRACT U | c. THIS PAGE U | SAR |
| 19a. NAME OF RESPONSIBLE PERSON CARL R. THOMAS | | | 19b. PHONE NUMBER (Include area code) N/A |

Table of Contents

| | |
|---|------------|
| List of Figures | iii |
| List of Tables | vi |
| 1 Executive Summary | 1 |
| 2 Introduction | 4 |
| 3 Methods, Assumptions, and Procedures | 8 |
| 3.1 Project Tasks | 8 |
| 3.2 Technical Contributions | 10 |
| 3.2.1 Creating a Privacy-Aware Smart Space Infrastructure | 11 |
| 3.2.1.1 Abstracting Interactions with IoT Devices Towards a Semantic Vision of Smart Spaces | 11 |
| 3.2.1.2 Integrating PETs into a Smart Space Platform | 23 |
| 3.2.2 Scaling Encryption and Secret-Sharing Techniques | 29 |
| 3.2.2.1 Partitioned Computations at the Hybrid Cloud | 31 |
| 3.2.2.2 Partitioned Computations at the Public Cloud via PANDA | 37 |
| 3.2.2.3 Scaling Secret-Sharing-based Techniques via OBSCURE | 49 |
| 3.2.3 Scaling Privacy Techniques | 62 |
| 3.2.3.1 Scaling Differential Privacy via One Sided Differential Privacy | 63 |
| 3.2.3.2 Minimally Invasive Monitoring | 66 |
| 3.2.4 Supporting Fine-grained policies in IoT Systems | 69 |
| 3.2.4.1 Scaling Policy Enforcement for IoT Data Management | 71 |
| 3.2.4.2 Preventing Leakages on Access Controlled Protected Data | 77 |
| 3.2.5 Supporting Verification of Data Capture Policies via IOT NOTARY | 83 |
| 3.3 System Testbed | 88 |
| 3.3.1 Sensing Infrastructure | 88 |
| 3.3.2 System Infrastructure | 90 |
| 3.3.3 Applications & Other Software | 102 |
| 3.4 Evaluation and Analysis of PETs | 115 |
| 3.4.1 Evaluation at UC Irvine | 115 |
| 3.4.1.1 Testing Differential Privacy in Window Analytical Tasks | 115 |
| 3.4.1.2 Privacy vs Utility Analysis of Connectivity Data | 119 |
| 3.4.2 Evaluation at Honeywell | 134 |
| 3.4.2.1 Model and Preliminaries | 135 |
| 3.4.2.2 Proposed Attack | 136 |
| 3.4.2.3 Experimental Study | 137 |
| 3.4.2.4 Mitigation Strategy | 138 |
| 3.4.3 Evaluation at US Navy | 139 |
| 3.4.3.1 Trident Warrior 2019 TIPPERS Technology Evaluation | 139 |
| 3.4.3.2 Integration of PETs in TIPPERS | 142 |
| 3.4.3.3 Evaluation of TIPPERS in Preparation of Trident Warrior | 142 |
| 3.4.3.4 PET Evaluation Results from Trident Warrior | 143 |
| 3.4.3.5 Trident Warrior 2019 Privacy Study | 143 |

| | | |
|----------|---|------------|
| 3.5 | Transition Efforts | 146 |
| 3.5.1 | US Navy’s Trident Warrior Exercises | 147 |
| 3.5.1.1 | Trident Warrior 2019 | 147 |
| 3.5.1.2 | Trident Warrior 2020 | 149 |
| 3.5.2 | COVID-19 Mitigation | 159 |
| 4 | Results and Discussion | 161 |
| 5 | Conclusion | 165 |
| 6 | Suggested Further Research | 166 |
| 6.1 | Policy Compliance in Big Data Ecosystems | 166 |
| 6.2 | Understanding the Implications of Data Leakage | 167 |
| 6.3 | Scaling Cryptographic & Secret Sharing based Approaches | 167 |
| 6.4 | Embedding Data Enrichment in Databases | 167 |
| 6.5 | Minimally Invasive Monitoring | 168 |
| 7 | Human Research Protection Official (HRPO) Review | 169 |
| | References | 170 |
| | Appendix A - Published Papers | 180 |
| | Appendix B - Unpublished Papers | 185 |
| | Appendix C - In-progress Papers | 186 |
| | List of Symbols, Abbreviations, and Acronyms | 187 |

List of Figures

| | | |
|----|---|----|
| 1 | TIPPERS Testbed Concept. | 4 |
| 2 | TIPPERS DARPA Brandeis Collaborators | 6 |
| 3 | High-level architecture of SemIoTic. | 12 |
| 4 | Overview of the semic metaontology to support the description of a smartspace. | 13 |
| 5 | Structures generated to handle a User Action: (a) Flattened tree for and (b) Execution plans generated for a | 15 |
| 6 | Snippets of (a) Two domain models based on semic, (b) Code of two wrappers, and (c) Tree generated to handle a sample user action. | 19 |
| 7 | Graphs displayed by the application using SemIoTic. | 20 |
| 8 | Development effort with (w) and without (w/o) SemIoTic. | 21 |
| 9 | Interaction between privacy-aware smart building management system (TIPPERS), IoT Resource Registries (IRR) and IoT Assistants (IoTA). | 25 |
| 10 | Policy related to data collection inside DBH. | 28 |
| 11 | Policy related to a service in the building. | 29 |
| 12 | Privacy settings available. | 29 |
| 13 | Comparing different cryptographic techniques. | 30 |
| 14 | Example relations. | 33 |
| 15 | Example relations with the CPT columns. | 35 |
| 16 | Running times for different sensitivity ratios. | 36 |
| 17 | The CPT column's creation for different sensitivity ratios. | 36 |
| 18 | Comparison of pseudo-sensitive data and sensitivity ratio. | 37 |
| 19 | A relation: <i>Employee</i> | 38 |
| 20 | A sensitive relation: <i>Employee1</i> | 39 |
| 21 | A sensitive relation: <i>Employee2</i> | 39 |
| 22 | A non-sensitive relation: <i>Employee3</i> | 39 |
| 23 | A bipartite graph showing an initial condition sensitive and non-sensitive values before query execution. | 40 |
| 24 | A bipartite graph showing sensitive and non-sensitive bins after query execution, where each sensitive value gets associated with each non-sensitive value. | 46 |
| 25 | Efficiency graph using equation $\eta = \alpha + \rho(SB + NSB)/\gamma$ | 48 |
| 26 | Dataset size. | 48 |
| 27 | Minimally Invasive Monitoring Architecture | 67 |
| 28 | Minimally Invasive Monitoring Architecture | 68 |
| 29 | Policy Evaluation overhead vs. Number of Policies. | 71 |
| 30 | Overview of Sieve. | 72 |
| 31 | Sieve on MySQL and PostgreSQL. | 76 |
| 32 | Scalability comparison. | 76 |
| 33 | Entities in IOT NOTARY. | 84 |
| 34 | Dataflow and computation in the protocol. Trusted parts are shown in shaded box. | 87 |

| | | |
|----|---|-----|
| 35 | The PE-IoT system at UCI OIT. PE-IoT adds privacy compliance to OIT and generates privacy-preserving WiFi connectivity data service providers . . . | 90 |
| 36 | Components of PE-IoT System | 91 |
| 37 | The web interface for users to specify their policies (opt-out) | 92 |
| 38 | Different deployments of the TIPPERS System. | 93 |
| 39 | Server infrastructure. | 93 |
| 40 | TIPPERS authentication GUI. | 94 |
| 41 | Main GUI of the Hub. | 95 |
| 42 | Finding TIPPERS portals in the Hub. | 96 |
| 43 | Most visited portals. | 97 |
| 44 | Screenshot of TPortal showing the apps available for users. | 98 |
| 45 | Screenshots of Tportal. | 99 |
| 46 | Screenshots of TMapper. | 100 |
| 47 | Sample APIs. | 101 |
| 48 | Screenshot of the Concierge app showing the different services offered. | 103 |
| 49 | Screenshot of the Concierge app showing sending a context aware message feature. | 104 |
| 50 | Screenshot of the Self-Awareness App. | 104 |
| 51 | Screenshots of Self-Awareness app. | 105 |
| 52 | Noodle: View Event Interface. | 106 |
| 53 | Noodle: Create Event Interface. | 107 |
| 54 | Screenshot of the Building Analytics app. | 108 |
| 55 | Occupancy adherence app. | 110 |
| 56 | Social distancing app. | 112 |
| 57 | Contact awareness app. | 112 |
| 58 | Exposure alert app. | 113 |
| 59 | Infection propagation app. | 113 |
| 60 | Alerts dashboard app. | 114 |
| 61 | Screenshot of the IoT-DETECTIVE game interface. | 117 |
| 62 | Methodology defined. | 119 |
| 63 | A sample plot: x -axis is the timepoint, y -axis the total number of people localized within each confidence class. | 127 |
| 64 | Prior guessing probabilities for $\delta = 0\%$ (left), $\delta = 50\%$ (middle), $\delta = 90\%$ (right) | 128 |
| 65 | Posterior guesses from noisy Laplace counts for $\delta = 0\%$ (left), $\delta = 50\%$ (middle), $\delta = 90\%$ (right), $\epsilon \in \{0.1, 1.0, 5.0, \infty\}$ (top to bottom) | 128 |
| 66 | Posterior guesses for worst-case DP, $\delta = 90\%$, $\epsilon \in \{0.1, 1.0, 5.0\}$ (left to right) | 129 |
| 67 | Successful guesses from noisy Laplace counts with $\delta = 90\%$ using true distribution (top) and scaled kernel density estimation (bottom), $\epsilon \in \{0.1, 1.0, 5.0\}$ (left to right) | 129 |
| 68 | Comparison of different mechanisms for the external visitor attacker. | 132 |
| 69 | Comparison of different mechanisms for the administrator attacker. | 133 |

| | | |
|----|---|-----|
| 70 | Figure illustrating the privacy implications of the presence data. | 135 |
| 71 | Figure illustrating the proposed attack. | 137 |
| 72 | Figure illustrating the increase of probability of correctly identifying the primary locations given auxiliary information. | 138 |
| 73 | Mobility Center of Excellence room design. | 143 |
| 74 | Results privacy study on Trident Warrior data. | 146 |
| 75 | TIPPERS team with sailors. | 148 |
| 76 | Sailors using TIPPERS mobile devices. | 150 |
| 77 | TIPPERS Mobile Fall Detection application deployed on ship | 152 |
| 78 | Sailors testing Covid-19 mitigation with TIPPERS. | 160 |
| 79 | UCI Computer Science COVID-19 Dashboard. | 162 |
| 80 | UCI Engineering COVID-19 Dashboard. | 162 |
| 81 | CalIT2 COVID-19 Dashboard. | 163 |

List of Tables

| | | |
|----|---|-----|
| 1 | Queries and returned tuples/adversarial view. | 40 |
| 2 | Queries and returned tuples/adversarial view. | 44 |
| 3 | Secret-shares of a vector $\langle 0, 1, 0 \rangle$, created by the DB owner. | 50 |
| 4 | Secret-shares of a vector $\langle 0, 1, 0 \rangle$, created by the user/querier. | 50 |
| 5 | Multiplication of shares and addition of final shares by the servers. | 51 |
| 6 | A relation: Employee | 54 |
| 7 | Two relations obtained from Employee relation. | 55 |
| 8 | An execution of the conjunctive count query. | 56 |
| 9 | An execution of the count query verification. | 59 |
| 10 | An execution of the sum query verification. | 62 |
| 11 | Employee details table. | 78 |
| 12 | Different authorization flows. | 95 |
| 13 | Running times (in seconds) of computing posterior probabilities for Laplace noise. | 129 |
| 14 | Parameters achieving the same utility for different privacy mechanisms. . . . | 131 |

1 Executive Summary

The goal of this project was to create an IoT smart space testbed with a plug-n-play architecture to infuse privacy enhancing technologies in order to explore their effectiveness in protecting users' privacy while still supporting a diverse set of smart space applications. The testbed, entitled TIPPERS, integrated privacy technologies including specification and enforcement of privacy policies, use of differential privacy in both collecting and sharing data, and secure computation using cryptographic approaches including multi-party computation and secret sharing.

In addition to providing a framework to study efficacy of privacy enhancing technologies listed above, TIPPERS provided a fertile tool for exploring novel challenges that arise in at-scale deployment of such technologies into a live real-world environment with data flowing from sensors being used to implement applications of everyday use. Such challenges relate to scaling privacy technologies to large data volumes and usage scenario complexities, or they relate to issues that arise in deploying such technologies, e.g., trust in infrastructure and organizations in adhering to policies.

The specific contributions of the TIPPERS project included:

- Development of a new layered approach to represent information in sensor-driven smart spaces that allows data to be modeled at the raw sensor level, as well as, at the semantically enriched level by dynamically interpreting sensor data as it arrives. Semantic representation makes both the task of specification and reasoning with privacy policies, as well as, application development easier.
- Creation of a novel data management technology entitled TIPPERS system that implements the semantic data model. In addition, TIPPERS embodies the “privacy-by-design” principle by creating a policy-based approach to data management wherein all aspects of data flow – from creation, ingestion, processing, sharing, storage, and retention are controlled by potentially fine-grained policies. TIPPERS also supports plug-n-play mechanisms to integrate a variety of privacy technologies including differential privacy and secure storage.
- Development of an end-to-end testbed at UCI based on the TIPPERS system. The testbed instrumented part of the campus - over 30 buildings - with sensors and sensor processing mechanisms to create a digital representation of activities in the buildings. Such activities include people in the building, their location, ongoing events, attendance by people in the events, etc. In addition, the testbed included a diverse set of end-user applications such as finding colleagues, analysing building usage, analyzing individual's interaction with the building and/or other users, support for contextualised messages etc.
- Integration of Pegasus differential privacy technology into TIPPERS testbed to hide information about individual's short term and longer term behavior.

- Integration of Stealth's Pulsar System and Galois's Jana system into TIPPERS for secure data storage and processing.
- Integration of the privacy assistant and privacy registry technologies developed by CMU into TIPPERS testbed for policy notification and preference selection.
- Development of the SEMIoTIC framework to support semantic abstraction of data in databases in order to specify privacy policies.
- Development of PANDA framework to support partitioned computation over sensitive and non-sensitive data in order to scale secure data management solutions to very large data sets.
- Development of OBSCURE and PRISM frameworks to support efficient evaluation of verifiable aggregation queries and private set operations (intersection and union) using secret sharing.
- Development of CONCEALER and CRYSTAL frameworks to outsource dynamic sensor data and compute both aggregation and SQL queries.
- Development of SIEVE infrastructure to scale database policy enforcement to large number (millions) of fine-grained policies prevalent in large scale smart space applications.
- Formalization of a new challenge of preventing data leakage through inferences when policies are specified at the semantic level of abstraction.
- A new model for one-sided differential privacy (OSDP) that exploits partial sensitivity of data to support increased utility (by allowing more data to be shared) while ensuring strict privacy properties on sensitive data.
- Extension of differential privacy framework to support minimally invasive monitoring that allows analyst to explore data at different levels of invasiveness based on the needs.
- Development of IOT NOTARY framework to verify compliance to organizational data capture policies.
- Development of IOT EXPUNGE framework to verify the data deletion against the user's data retention policies.
- Development of CANOPY algorithms for preventing user privacy in a smart home by exploiting round-robin style dataflow algorithms.
- Development of a PE-IoT framework that allows data streams to be intercepted and for diverse privacy technologies (differential privacy, encryption, anonymization, randomization, etc.) to be applied in sharing data.

- Detailed privacy analysis of different privacy technologies in supporting location privacy at UCI.
- Transition and Deployment of TIPPERS to the US Navy and to other organizations, including other schools such as BSU and to Honeywell Labs.

To report the accomplishments of this project's goals, the remainder of this report is divided into the following sections: Technical Contributions, System Testbed, Evaluation and Analysis of PETs, and Transition Efforts. Following that, two appendices listing the research manuscripts, systems, applications, and demonstrations resultant from this project are provided.

2 Introduction

The objective of this project was to explore a new plug-n-play privacy cognizant end-to-end system architecture and methodology to build IoT applications that serves as an underlying technology enabler/driver for a myriad of smart applications made possible by the emerging sensing and data capture technologies.

In IoT applications, data flows from sensors through data processing pipelines that interpret and enrich sensory data in various ways in order to realize the needs of diverse applications. Live data generated at sensors may be consumed by applications and/or stored for future analysis and processing to implement new functionalities and learn correlations that can help improve embedded applications. Data processing in the IoT context may be performed in on-board sensor devices or at the trusted servers in a local infrastructure, or may be relegated to the possibly untrusted public cloud machines.

We set out to create a one of a kind smart space testbed at UCI that would enable privacy technologies to be evaluated in live everyday context. Our concept behind the testbed, entitled “Testbed for IoT-based Privacy Preserving Pervasive Spaces” (TIPPERS for short) is demonstrated in Figure 1.

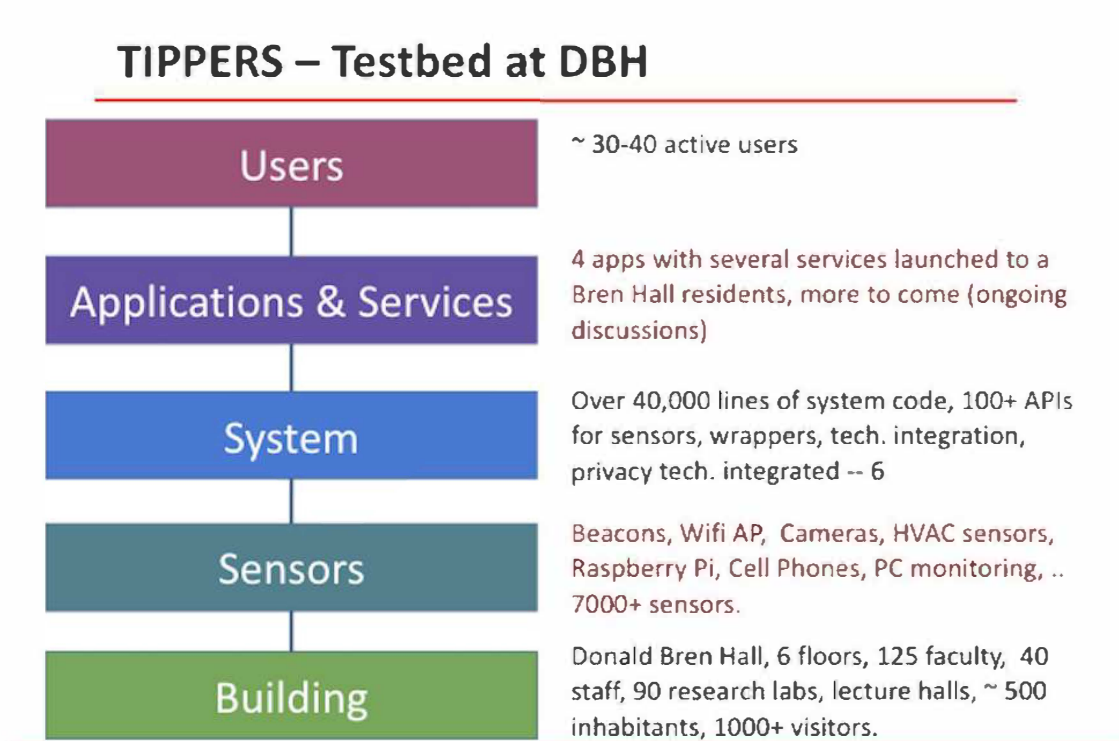


Figure 1: TIPPERS Testbed Concept.

Figure 1 embodies a novel end-to-end testbed design consisting of instrumented spaces (the figure shows Donald Bren Hall, that is one of the buildings instrumented as part of the project) with a variety of sensing devices such as Bluetooth beacons, motion sensors, cameras, acoustic sensors, proximity sensors, and location sensors.

The data flowing out of these sensors is collected and processed in a novel data management technology, the TIPPERS system, that supports plug-n-play mechanisms to embed privacy technologies ranging from encryption and policy-based data processing to differential privacy. The TIPPERS system also provides a semantic abstraction of data that enables building a variety of sensor-based applications as well as making sensor data analysis easier. In particular, the TIPPERS database represents the sensor data flowing into the system as both a stream of sensor data flowing into the system over time, and also at a more semantic level in terms of the evolving state of the physical space and entities that are embedded into the physical space. Sensor data is used to determine location of people over time, ongoing activities/events, and people's participation in the events to create a digital representation of the physical world – i.e, location of individuals over time and their participation in activities such as meetings, classes, special events, etc. Such data is then used to create a variety of applications both at the aggregate /analytic level, as well as, at the individual level. Applications at the aggregate level include analysis of how a building is used, its space utilization, energy usage, etc. At the individual level, TIPPERS supported real-time applications such as finding location of colleagues, sending contextualized messages, enabling individuals to determine how they interacted with others and with the building, etc. With everyday applications such as above in daily use, the TIPPERS testbed provided a fertile ground to study the degree to which current and ongoing privacy technologies address individual's concerns of privacy, whether one can build realistic applications that provide value while ensuring privacy.

The TIPPERS testbed, that initially started with a single building at UCI, the Donald Bren Hall, spread quickly to several buildings all over the campus and now TIPPERS enabled applications are in daily use in about 25 buildings in the campus. In addition, TIPPERS was transitioned to the US Navy and the software has been deployed at Ball State University.

The extended TIPPERS team that included participants from diverse Brandeis funded DARPA partners are shown in the figure entitled TIPPERS Cluster. In particular, TIPPERS integrated differential privacy techniques built by the Duke, Amherst, and Colgate team (PeGaSus/System P). It also integrated secure data management techniques built by Galois entitled Jana, and a system entitled Pulsar built by Stealth technologies. The TIPPERS team collaborated with CMU to integrate the IoT Assistant and the registry technologies that allowed smart spaces to specify their resources as well as policies for data capture and usage. TIPPERS also worked closely with Galois and Cybernetica for determining privacy expectations and leakage analysis of location data in the TIPPERS testbed.

Since TIPPERS testbed built a full stack solution from sensors, data collection protocols, data management, data processing and analytics, as well as, end user applications that were deployed and used by the UCI students, faculty, staff and visitors on a daily basis, it provided a unique opportunity to study privacy challenges in a controlled, yet real-world setting. In particular, it enabled the (extended) team of Brandeis contributors to understand the privacy needs of users, develop, deploy and validate their solutions. It also allowed the TIPPERS team to identify and expose a variety of challenges that become evident only when we apply the technology to build applications for at-scale deployments .

Tippers System

Technologies utilized from other BRANDEIS Program Contributors

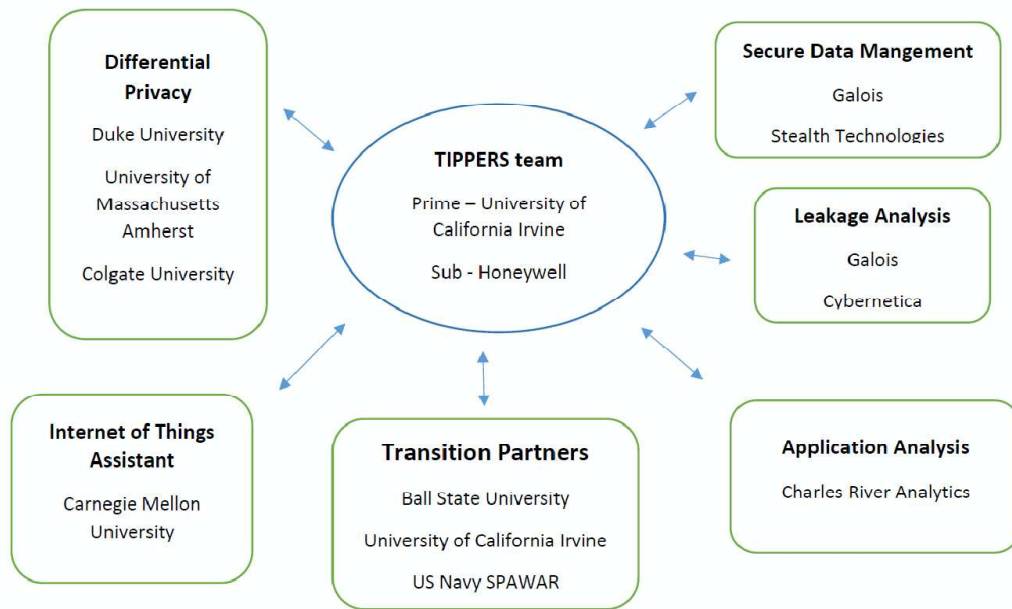


Figure 2: TIPPERS DARPA Brandeis Collaborators

Three novel challenges, that shaped the TIPPERS technical contributions, that emerged from building the testbed were:

- **Semantic Abstraction** – in smart space domain, from the perspective of building applications and/or specifying policies (whether they be data capture, sharing, or retention policies) it is more natural to view data at a higher level of abstraction as compared to simply a collection of sensor values. Semantic level of data empowers analysts/users to specify policies, or performs reasoning and build applications using concepts such as physical spaces, events, people, activities, etc. instead of having to process/interpret/reason with low level sensor data. The system contains enough intelligence to convert/transform lower level sensor data into a semantically enriched representation. Such semantic abstraction, however, introduces new challenges to all aspects of privacy technologies – whether it be policy research, encrypted storage, or differential privacy.
- **Scaling Technology** – privacy enhancing technologies such as encrypted data search, or differential privacy, or policy enforcement have been studied in idealistic situations with limited data, and assumptions of independence that are not valid in the real world setting. Scaling existing PETs to real-world situations where one needs to handle big data and deal with complex data, thus, opens a large number of interesting challenges.
- **Trust and verification** - real-world deployment of privacy technologies, in particular

privacy policies, not only requires systems to scale policy enforcement to the volume and speed of data as it arrives, it also raises concerns of trust amongst the subjects about the data being used by organizations as intended/advertised. Such a requirement emerged directly from the interactions between TIPPERS team and the stakeholders at UCI where the TIPPERS testbed is deployed. One of the salient requirements to emerge was mechanisms for users to verify the actions of the system and the way the system uses the data.

TIPPERS, in addition to providing a platform for other researchers to explore efficacy of their privacy technologies, made technical contributions to address each of the above challenge. To address the challenge of semantic abstraction mechanisms were developed to translate data/queries/policies specified at the higher level of abstraction to the sensor level for implementation. Techniques based on exploiting the fact that in any application context, only part of the data is sensitive, while the other part might not be was used to scale a variety of privacy technologies. For instance, TIPPERS explored a novel concept of partitioned computation where data processing is split between sensitive and non-sensitive data to scale data processing. Likewise, a novel concept of one-sided differential privacy was developed to support strong guarantees similar to differential privacy but only for sensitive data. Differential privacy concept was also extended to build a minimally invasive architecture to support monitoring applications that empowers analyst to request more (invasive) data iteratively after establishing the need for the data. Mechanisms were also developed to verify that the system implements the data capture and retention policies.

The rest of the report is organized as follows. In Section 3.1, we discuss the specific tasks that were part of the agreement and our progress and approach towards each of the tasks. In Section 3.2, we describe our technical contributions to privacy enhancing technologies that address issues related to semantic abstraction, scalability and verification. This is followed by the description of the TIPPERS system and the set of applications built using the system. Section 3.4 focuses on evaluating diverse privacy technologies in the TIPPERS testbed in the terms of location privacy. Section 3.5 describes the transition efforts that we undertook and the appendices contain a bibliography of all the scientific papers that were a by-product of the TIPPERS project.

3 Methods, Assumptions, and Procedures

3.1 Project Tasks

Below is the list of tasks agreed upon by the DARPA contract. A discussion of each task's accomplishment is provided. Research tasks included tasks related to the development and deployment of the TIPPERS research system.

- **Task R1: Architecture design for the research system to support privacy by design principle at different points in system and data workflow**

Approach taken: We designed and implemented a new technology entitled *Privacy Enhanced IOT* (PE-IoT) that serves as a middleware for implementing privacy in IoT data flows. Pe-IoT enabled data flow from sensors/collection of sensors to be intercepted and transformed with appropriate privacy mechanisms ranging from encryption, differential privacy, to obfuscation, de-identification. It also enabled information flow to be subjected to privacy policies.

- **Task R2: Detailed Component design of research system**

Approach taken: Detailed design of the major communication, sensing, acquisition, scheduling, flow management, data management, and analysis components that together formed the RESEARCH system. We followed a spiral design methodology such that basic functionalities were quickly implemented, which then were enhanced over time. Comment: Original plan of using Tridium as a basis of the system design was modified since Tridium framework was not available from Honeywell. Instead the system was designed and built without using Tridium.

- **Task R3: Spiral 1 Implementation & point testing of key components of the research system**

Approach taken: Code development for basic data acquisition, sensor and task scheduling, sensor data management, and metadata extraction. A series of test cases based on driver applications were used to test components developed in Spiral 1. Comment: Original plan of using Tridium was modified since Tridium framework was not available. Instead the system was designed and built without using Tridium.

- **Task R4: Integration of sensor infrastructure and porting current applications to the research system**

Approach taken: research system prototype built and the sensor infrastructure and applications were ported to the system

- **Task R5: Testing and validation of Spiral 1 research system installation**

Approach taken: test cases through driver applications.

- **Task R6: Identification & integration of key TA1&2 technologies into Spiral 1 RESEARCH system Approach taken:** working with DARPA and other TA1 and 2 performers we identified privacy technologies that could be integrated into the driver applications. In particular, we integrated the IRR and IoTA technologies of CMU, Pegasus and System P technologies by Amherst, Duke, and Colgate, the Jana technology by Galois, and Pulsar technology by Stealth.
- **Task R7: Spiral 2 Extensions for RESEARCH system Approach taken:** Techniques for metadata enhancement of raw sensory data in the RESEARCH system, which offer new opportunities to test additional privacy interventions.
 - **SubTask R7.A: Massive scale policy enforcement** We developed a SIEVE system that exploit the extraction of simplified expressions of policies as a pre-filtering process as well as user-defined function technology to discard large amounts of data as soon as possible in the enforcement process and perform more refined checkings for localized smaller subset of the data.
 - **R7.B: Developing models for localization based on IoT sensor data.**
We developed models based on rules that use historical connectivity data and background information (such as the classification of rooms in a building) to determine the localization of individuals at different granularities: building-level, region-level, and room-level.
- **Task R8: Identification & integration of key TA1 and TA2 technologies for integrating into Spiral 2 research system Approach taken::** working with DARPA and other TA1 and 2 performers we identified privacy technologies that can be integrated into the driver applications. These included the IRR and IoTA technologies of CMU, Pegasus and System P technologies by Amherst, Duke, and Colgate, the Jana technology by Galois, and Pulsar technology by Stealth.
 - **R8.A: Approach to exploiting Secure Hardware & Mediating across Secure Systems.** Exploration of (a) exploiting SGX to scale secure data processing, and (b) developing a mediation-based approach that support different types of information with different security requirements across different systems (e.g.. In PULSAR, or Jana, or in Postgres) to takes advantages of what such systems offer without compromising security.
 - **R8.B: Minimally invasive Monitoring concept.** We studied a new architecture that supports minimally invasive monitoring. The key idea is to explore how one can build a system that protects privacy as much as possible, and releases data only when the need arises in the context of a monitoring task.
- **Task R9: Integration of TA4 privacy measurement toolset into Spiral research system Approach taken:** Adaptive interfaces of the research system were used to integrate TA4 privacy measurement toolkit.

- SubTask R9.A: Large scale Privacy Study based on real IoT data Design methodology to analyze location privacy leakage based on data released through different PETs. Use of WiFi connectivity data collected by the TIPPERS system to understand location privacy leakage (e.g., in terms of probability of guessing the location of individuals).
- **Task R10: Refinement of the RESEARCH system prototype for final study**
Approach taken:: Techniques to harden the software base, make it more flexible for further integration and enhancement of TA1 and TA2 techniques
 - SubTask R10.A: Support for Trident Warrior 2019 and 2020 Exercises Analysis of TW 2019 and TW 2020 data collected, improvement of TIPPERS, working with other performers to harden the privacy enhancing technology integration, and working with NIWC to develop additional TIPPERS applications of interest to the US Navy.
 - SubTask R10.B: Scalable, Extensible, & Interoperable System Design by developing a framework for TIPPERS Design of an interoperable layer for the TIPPERS system to enable the definition of new spaces through easy-to-use interfaces and provide interoperability so that content developed for one space (e.g., applications) can be easily incorporated to others without modifications.
 - SubTask R10.C: Extension of UCI Testbed to campus scale Extending the De-identification engine to support opt-in/out technologies. Exploration of differential privacy in the context of WiFi data sharing between campus and service providers. Scalable mechanism to de-identify large amounts of connectivity data in real-time. Hardening of software for de-identification for continuous deployment and possible transition to others. Modifications on TIPPERS engine to support new de-identified data and to facilitate rapid deployment of TIPPERS instances in other campus buildings. Training/information sessions at UCI inform individuals about data capture policies, de-identification engine privacy technologies, and TIPPERS. New IRB protocols for data usage.
 - SubTask R10.D: Attestation & Logging. Design and implementation of tamper proof logging and mechanism to attest policy enforcement by appropriately logging policies and sensor data.

3.2 Technical Contributions

In this section, we describe the technical contributions that resulted from the TIPPERS project. The technical contributions are organized along four major directions.

- Challenges in Creating a Privacy Aware Smart Space Infrastructure.
- Mechanisms / techniques to scale cryptographic solutions for secure data processing to real-world complexity of the TIPPERS testbed.

- Mechanisms to adapt and scale privacy technologies to real-world complexity of the TIPPERS testbed.
- Mechanisms to support fine-grained policies in TIPPERS.

Below, we discuss our contributions along each of the three technical areas discussed above.

3.2.1 Creating a Privacy-Aware Smart Space Infrastructure

We designed a framework that enables the integration of PETs in a smart space. One of the fundamental tasks of such framework is the abstraction of the interaction with IoT devices. The goal is to provide a more semantically meaningful vision of the smart space. This facilitates the integration of PETs in two ways: 1) It simplifies the usage or user-defined privacy policies, which can be defined in terms of high-level concepts and then translated into low-level actions on the raw-data/devices; 2) It enables the integration of technologies such as Differential Privacy or Encryption in different parts of the data flow.

In this section, we first describe the development of a layer in our infrastructure that deals with the challenge of abstracting IoT devices and sensor data. Then, we describe a model to integrate user-defined privacy policies in the previous smart space infrastructure.

3.2.1.1 Abstracting Interactions with IoT Devices Towards a Semantic Vision of Smart Spaces

This section describes a middleware framework for IoT smart spaces, SemIoTic, that provides application developers and end-users with the semantic domain-relevant view of the smart space, hiding the complexity of having to deal with/understand lower-level information generated by sensors and actuators. SemIoTic uses a metamodel, based on the popular SOSA/SSN ontology with some extensions, to represent relationships between the low-level IoT devices' world (i.e., devices, observations) and semantic concepts (i.e., users and spaces and their observable attributes). It supports a language using which users can express their action requirements (i.e., requests for sensor data, commands for actuators, and privacy preferences) in terms of user-friendly high-level concepts. We present an ontology-based algorithmic approach to translate user-defined actions into sensor/actuators commands. Finally, our end-to-end approach includes a cross-layer solution to provide interoperability with diverse IoT devices and their data exchange protocols.

High-Level Architecture

There are three main stakeholders involved in the SemIoTic ecosystem: Administrators of a smartspace who describe it into the system (e.g., what types of sensors are deployed, what information do they collect, etc.); Developers who utilize SemIoTic to develop applications that interact with the smartspace; Users of the applications who express through them actions they want to perform in the space. SemIoTic handles such *user-defined actions* and manages the IoT devices defined in the smartspace to perform them. User-defined actions

are of three types: 1) Requests for dynamic or static information about the space (e.g., to obtain the current location of a person or to monitor the occupancy of a specific room every five minutes for the next two hours). 2) Commands related to such entities (e.g., to switch on the AC if the occupancy of the room is above its capacity). 3) Privacy preferences/policies regarding the handling of information (e.g., to deny the capture of any information that can lead to determining the location of a person). The architecture of SemIoTic to handle such interaction is based on three main components (see Fig. 3):

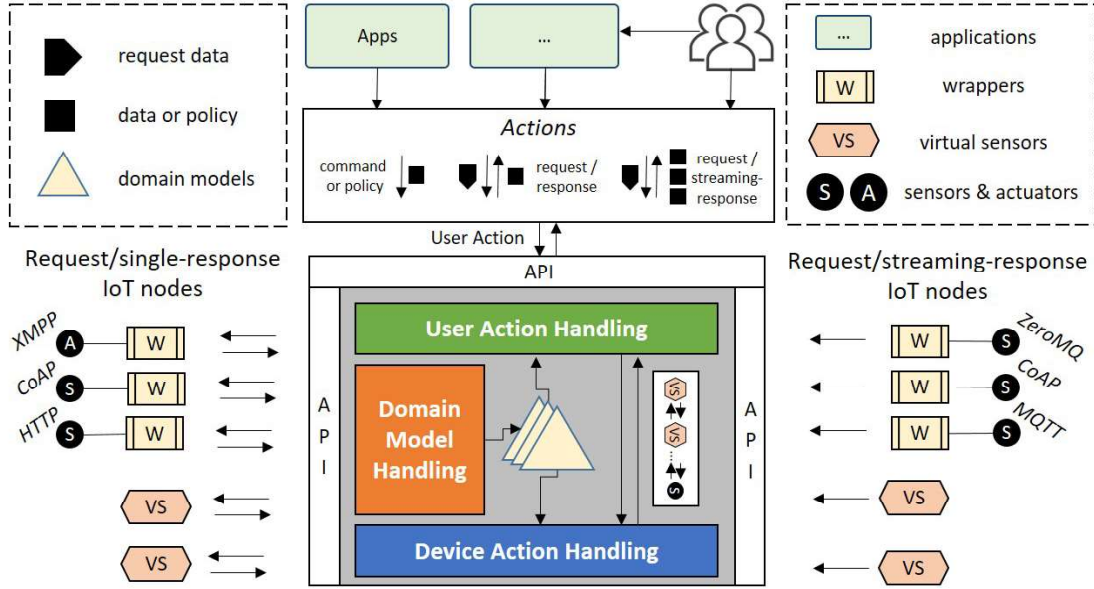


Figure 3: High-level architecture of SemIoTic.

- *Model Handling*, which enables administrators to describe the smartspace in terms of types of *spaces*, *users*, and *devices*, as well as specific instances of those types.
- *User Action Handling*, which takes as input user actions, based on high-level concepts defined in the model, and translates them into an appropriate and feasible plan of actions on the devices deployed in the smartspace.
- *Device Action Handling*, which is responsible to access the devices assigned to execute the plan through *wrappers*, that encapsulate the interaction, and/or *virtual sensors*, that process raw sensor data to produce semantically meaningful information.

Data Model Handling

SemIoTic bases its processing on a model (semic), that describes the smartspace, and a language (SemIoTic Action Language –sal–), to enable users to define complex actions related to the entities in the model. See [1] for the complete specification of *semic* and *sal*. The model, managed by the administrator of the SemIoTic-enabled space, is built on concepts of the *semic* metaontology. *semic* is an extension on the SSN/SOSA ontology [2] to support the automatic translation of user actions defined around higher level concepts into device

actions at a lower level. Fig. 4 shows a snippet of `semic` along with the required elements of SSN/SOSA.

Figure 4: Overview of the semic metaontology to support the description of a smartspace.

IoT Devices, Observations and Actuators. `semic` follows the SSN/SOSA definition of IoT devices extended to introduce two subclasses of the `sosa:Sensor` concept which we use to represent *physical sensors*, that sense the environment, and *virtual sensors*, that are software components that use data from other sensors to generate their observations about higher-level phenomena. Similarly, we specialize the concept of observation to further divide them into *raw observations*, captured by physical sensors, and *semantic observations*, captured by virtual sensors. As before, we expect the administrator to define appropriate subclasses of the sensor or actuator concept and then associate instances to them. For instance, one could define the subclass of sensor “Thermometer” which observes the subclass of raw observation “Temperature”. We use the concept of observation, connected to sensors, along with the aforementioned attribute `semic:observationType`, connected to properties of

entities, to bridge the gap between high-level and low-level concepts (this is the similar for actuators).

semic also introduces two attributes to devices which are used to represent their location, in the same coordinate system used to describe the space as well as the coverage of the device. This way, we can represent, for example, that a thermometer sensor is inside of a room and it covers (i.e., can observe) a radius around its location. Finally, semic supports the definition of *Quality-of-Service* features for IoT devices. We reuse the QoS ontology presented in [3] to represent metrics related to devices such as their response time, latency, error rate, reliability, cost (e.g., dollars per observation).

Action Language. A sal user action (denoted in the rest of the paper as UA) can be either a request for data (UR), a command (UC), or a policy (UP). The general format of such actions includes the following definitions: 1) Entities of interest, E , which is a set of one or more entities $\varepsilon_i \in E$ such that each ε_i is either an entity class (i.e., $\langle \varepsilon_i, \text{rdfs:subClassOf, semic:Entity} \rangle$) or an entity instance (i.e., $\langle \varepsilon_i, \text{rdf:type, semic:Entity} \rangle$). For example, the action can be related to either a general concept such as “Meeting Rooms” or a specific instance such as “Room 111”. 2) Properties of interest, P , which is a set of properties $\rho_i \in P$ for which values have to be obtained or actions have to be performed (i.e., $\langle \rho_i, \text{rdf:type, semic:Property} \rangle$). For example, “occupancy”, “capacity”, and “control temperature” in the case of a room. 3) Conditions, C , which is an expression that has to be satisfied as a condition to perform the actions on the entities. We assume that the condition expression contains one or more properties (e.g., the occupancy of the room has to be greater than the capacity). 4) Parameters, which is a set of parameters (involving both QoS and/or parameters related to the observation/action to obtain/perform). For example, this could be the definition of the measure unit for the temperature values to obtain (e.g., Fahrenheit or Celsius). Additionally, privacy policies contain two more attributes: 5) Interaction to control (i.e., capture, store, share). 6) Specific preferred action (i.e., accept or deny).

Given the above sal definition, we can express actions such as: “retrieve the current location of John and Mary” ($\langle \text{John, Mary, LocationProp, } \emptyset \rangle$), “decrease the temperature of those rooms with occupancy above 50% of their capacity” ($\langle \text{Room, ControlTempProp, OccupancyProp} > 0.5 \times \text{CapacityProp} \rangle$), or “do not capture the location of Mary when she is in a private space” ($\langle \text{Mary, LocationProp, LocationProp=PrivateSpace, capture, deny} \rangle$).

sal supports also the definition of device actions which are used internally by SemIoTic as a result of the translation of user actions. A device action DA , which can be either a sensor request SR or an actuator command AC , contains the following definitions: 1) Device to perform the action, D , (this could be a class of devices or a specific instance); 2) Entity of interest that the device should observe/actuate upon, ε , (this has to be an instance of an entity), and 3) Type of observation/action to request/command the device, a ; 4) Parameters, as in the UA . As an example, we could define a request to capture temperature data from a specific thermometer as $\langle \text{thermometer111, room111, TemperatureObs} \rangle$.

User Action Handling

We explain the steps involved in the handling of high-level user actions illustrated using the BPMN inspired data structure in Fig. 5.

Flattening. Complex user actions, UAs , e.g., containing conditions, require the processing of other internal actions to resolve them. For instance, to handle the action to control the temperature of rooms where the occupancy is above 50% of their capacity, we need to execute user requests to obtain the capacity as well as the occupancy of the different rooms. We refer to this process as *flattening* (borrowing the terminology used in databases to refer to the process to convert a nested query into a non-nested one).

The flattening process takes a user action, $UA = \langle E, P, C \rangle$, and generates a tree structure, \mathcal{T}_{UA} , that contains the high-level plan required to process it in terms of other UAs . Fig. 5(a) shows the resulting data structure after the flattening of a UA . \mathcal{T}_{UA} generated in this step fulfills the following: 1) The first level of the tree flattens UA by extracting the entities of interest from E (e.g., all the instances of the class “Meeting Room” in our running example). Thus, this level contains a set of UAs such that $UA_i = \langle \varepsilon_i, P, C \rangle$ where $\varepsilon_i \in E$ and $\langle \varepsilon_i, \text{rdf:type}, \text{semic:Entity} \rangle$. 2) For each UA_i , the next level flattens the set of internal UAs that need to be processed to compute C (e.g., the requests to get the occupancy and the capacity of each meeting room in our example). This is a set of URs such that $UR_{ij} = \langle \varepsilon_i, \rho_j \rangle$ where ρ_j refers to the j -th property needed to compute a $c \in C$. Notice that we consider that conditions require data obtained through requests and not commands. 3) The last level of the tree flattens the UAs that need to be processed to perform the user action on each property in P (e.g., the actuatable property to control the temperature in our example). Thus, it contains the UAs leaf nodes such that $UA_j = \langle \varepsilon_i, \rho_j \rangle$ where $\rho_j \in P$.

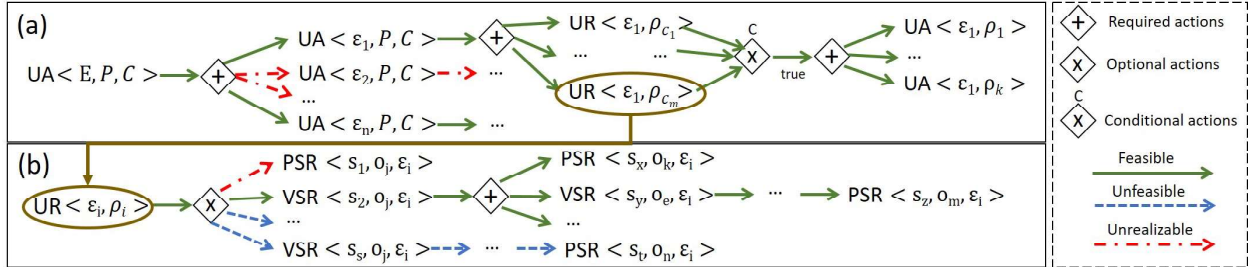


Figure 5: Structures generated to handle a User Action: (a) Flattened tree for UA and (b) Execution plans generated for a UR .

The flattening algorithm takes as input the user action, UA , and the domain model M , and outputs the tree explained above. First, the algorithm extracts the list of entities, properties, and conditions associated to UA . Notice that if E contains a set of entities such that $\langle \varepsilon_i, \text{rdf:type}, \text{semic:Entity} \rangle \forall \varepsilon_i \in E$ then the method $ExtractEnt(UA, M)$ returns that same set. If $\langle \varepsilon_i, \text{rdfs:subClassOf}, \text{semic:Entity} \rangle \forall \varepsilon_i \in E$ then the method uses a *Description Logic reasoner* [4] to obtain any instance $m_j \in M$ such that $\langle m_j, \text{rdf:type}, \varepsilon_i \rangle$. This means that the hierarchical nature of the representation is taken into account and if m_j is an instance of a ε_k such that $\langle \varepsilon_k, \text{rdfs:subClassOf}, \varepsilon_i \rangle$ then m_j will be re-

turned. For instance, if $\varepsilon_i = \langle \text{Room}, \text{rdfs:subClassOf}, \text{semic:Entity} \rangle$, $\varepsilon_k = \langle \text{Meeting Room}, \text{rdfs:subClassOf}, \text{Room} \rangle$, and $m_j = \langle 111, \text{rdf:type}, \text{MeetingRoom} \rangle$, then m_j will be returned by the *ExtractEntities*(UA, M) method for a UA where $\varepsilon_i = \text{"Room"} \in E$.

Execution Plans Generation. After flattening, \mathcal{T}_{UA} contains the set of internal UAs that have to be processed to handle the user action (see for instance, the different UA_i in each level of the flattened tree of Fig. 5(a)). Each $UA_i \in \mathcal{T}_{UA}$ will require a set of device actions, DA , to be executed. Notice that more than one type of device could be able to perform such action so all the possible options have to be included as different plans. For instance, a particular UA_i representing a user request to obtain the occupancy of “Room 111” will result into a DA set to different occupancy counter sensors (either virtual or physical).

The execution plans generation step expands \mathcal{T} by extending each UA_i with a set of execution plans as \mathcal{T}

]. Fig. 5(b) shows the structure of the execution plans for a particular $UR = \langle \varepsilon_i, \rho_j \rangle$ that is used to expand the highlighted node in the Fig. 5(a). The constraints of \mathcal{T}_{UA_i} are as follows: 1) Each level of the tree contains a $SR = \langle s_k, o_j, \varepsilon_i \rangle$ which can be either for a virtual sensor, notated by VSR and such that $\langle s_k, \text{rdfs:subClassOf}, \text{semic:VirtualSensor} \rangle$, or for a physical sensor, PSR and such that $\langle s_k, \text{rdfs:subClassOf}, \text{semic:PhysicalSensor} \rangle$. 2) For each VSR there will be an additional level with requests for those (physical or virtual) sensor requests that need to be executed in order to obtain the input required by s_k . 3) The leaf nodes of the tree contain only PSR .

Thus, this step iterates for each $UA_i = \langle \varepsilon_i, \rho_j \rangle$ node in \mathcal{T}_{UA} and performs an iterative process to extract the different execution plans possible. First, the algorithm determines which device classes can execute the required action (i.e., which sensors can capture observations of the type associated with the property of interest or which actuators can perform actions of the type associated with the property of interest). To this end, it queries M to retrieve any device d such that: $d = s$ if $\langle \rho_j, \text{rdfs:subClassOf}, \text{semic:ObsProperty} \rangle, \langle s, \text{rdfs:subClassOf}, \text{semic:Sensor} \rangle, \langle s, \text{semic:captures}, o_k \rangle, \langle \rho_j, \text{semic:obsType}, o_k \rangle$ and $d = a$ if $\langle \rho_j, \text{rdfs:subClassOf}, \text{semic:ActProperty} \rangle, \langle a, \text{rdfs:subClassOf}, \text{semic:Actuator} \rangle, \langle s, \text{semic:actuates}, o_k \rangle, \langle \rho_j, \text{semic:actionType}, a_k \rangle$.

Notice that more than one device d can be obtained for the same property. For instance, different virtual sensors could retrieve the occupancy values using different inputs (e.g., WiFi connectivity data or video camera feeds). For each d retrieved, the algorithm creates a device action, DA_d where $DA_d = SR_d$ if $d = s$ or $DA_d = AC_d$ if $d = a$, and appends it as a node under the corresponding UA_i node. If $\langle d, \text{rdfs:subClassOf}, \text{semic:VirtualSensor} \rangle$ then such virtual sensor might need additional input sensor data. For each of the input observation types defined for the virtual sensor, the algorithm similarly retrieves devices that can capture such data and appends them to that node. This process is performed iteratively until all the leaf nodes of the tree are PSR or AC .

Plan Realizability and Feasibility Checking. A plan could be *unrealizable* given the deployment of devices in the scenario or the policies defined by users. For example, consider a plan that points out that occupancy of a room can be obtained by a virtual sensor that analyzes images from cameras and by another virtual sensor that uses a movement detector

sensor in the room. Those plans will be unrealizable for a specific room that does not contain any movement detector or that is not in the view frustum of any video camera. Similarly, the plan will be unrealizable if there is policy that dictates that the camera cannot be used at that moment (e.g., because a user is in the space who does not want to be tracked). In addition, some plans can be realizable but *unfeasible*. For instance, for rooms that have both cameras and motion detectors, capturing video data and analyzing it to detect occupancy might be unfeasible depending on the cost constraints of the user.

The plan realizability checking step prunes down branches of the extended \mathcal{T}_{UA} (i.e., \mathcal{T}_{UA} containing all the possible execution plans) that are unrealizable and classify the remaining regarding their feasibility. In the example tree of Fig. 5, we have marked some of the plans according to their realizability and feasibility as an output of this step. Notice that the result could be an empty tree if the whole UA is unrealizable because of a lack of devices that can capture raw observations or perform the required commands. In this step, SemIoTic performs a reverse level order traversal of the tree starting with the leaf nodes, which by definition contain a $DA = \langle D, \varepsilon_i, o_j \rangle$ which is either a *PSR* or a *AC*. Given such node, \mathcal{N}_{DA} , the method `getAptDevices(DA_i, M)` obtains the set of those specific instances of physical sensors/actuators deployed in the space that can perform such action, \mathcal{D} , by using the function `checkCoverage(d, ε_i)` for all d such that $\langle d, \text{semic:captures}, o_j \rangle$ (i.e., d is a device that can capture observations or actuate actions of the type related to the property of interest). The `checkCoverage` function returns true if a specific device d can cover the entity ε_i by using the `semic:Extent` associate with ε_i and the `semic:location` and `semic:coverage` property associated with d . This way, if the observation type to capture is “image” and the entity of interest is “room 111”, for a camera “cam111” that has the room in its view frustum `checkCoverage(“cam111”, “room 111”)` will return true.

For those devices that can cover the space, the `getAptDevices()` method performs an additional call to the `checkAccess(d, \mathcal{P})` method where \mathcal{P} is the set of all *UPs* defined by users of SemIoTic. The goal is to check whether there exists a policy that restricts access to d . Given a user defined policy $UP_n \in \mathcal{P}$ the same process described so far is applied to generate possible execution plans. Thus, SemIoTic generates a \mathcal{T}_{UP_n} that contains all the different devices involved in the processing of the policy. If there exists a node $\mathcal{N}_m \in \mathcal{T}_{UP_n}$ such that $\mathcal{N}_m = DA = \langle d, \varepsilon_j, o_k \rangle$ and the preferred action defined for the UP_n is to deny the access, then `checkAccess()` returns false.

If $\mathcal{D} = \emptyset$ then \mathcal{N}_{DA} is removed from \mathcal{T}_{UA} . In such a case, SemIoTic checks \mathcal{N}_{DA} parent nodes and those are also removed if they require the processing of DA_i (e.g., if the parent is a *VSR* that takes as input the observations of DA_i). This is done recursively and nodes are removed until a parent of an unrealizable node does not require such node (e.g., because it can obtain its input data from another child). If $\mathcal{D} \neq \emptyset$ then the node \mathcal{N}_{DA} that specified an action on a general device type D has to be replaced by specific actions on the devices in \mathcal{D} . For each $d_k \in \mathcal{D}$ SemIoTic creates a $DA_k = \langle d_k, \varepsilon_i, o_j \rangle$ and this action gets added as child to \mathcal{N}_{DA} . At the same time, SemIoTic computes a *feasibility score* for DA_k which can be used to compare whether a plan is more feasible than others. This score gets added to the metadata of \mathcal{N}_{DA_k} .

To compute the feasibility score of a DA_k (represented as a cost $C(DA_k)$), SemIoTic uses the different cost metrics defined in \mathbf{M} for that specific device. These metrics (e.g., processing time, quality of the answer, economical cost, privacy grade, etc.) get aggregated by using a cost model based on weights $C(DA_k) = \sum_{j=1} w_k c_{jk}$ where w_k is the weight assigned to the k -th cost, c_{jk} , associated with the device d_k in DA_k . In cases where the user explicitly requires some preferences regarding the cost or quality of the answer, this information can be leveraged to assign weights to each objective/cost. By default we consider that the value for the weights are assigned uniformly.

Plan Selection and Execution. Given a UA , several execution plans can be feasible and, if the goal is to maximize the chances of carrying out the action (as devices might fail or produce noisy results), SemIoTic can execute them all. However, in general this might result in a waste of resources as we would be duplicating efforts to obtain the same result. Thus, SemIoTic chooses the most feasible plan according to the score computed in the previous step. This way, it computes the feasibility of each plan, by recursively aggregating the cost of its nodes, and removes all the branches of \mathcal{T}_{UA} which do not have minimal cost. Notice that, the flexible and modular design of SemIoTic makes it possible to use other more sophisticated optimization functions to assign costs and select plans.

Once a single plan has been selected, the next step is to execute it. The execution engine executes first each $UA_i \in \mathcal{T}_{UA}$ that is needed to compute the condition component of UA (if any). Then, after checking whether the condition is satisfied, it executes each $UA_j \in \mathcal{T}_{UA}$ related to the properties of interests in UA . Given a $UA_k \in \mathcal{T}_{UA}$ the execution engine performs a reverse level order traversal of the subtree \mathcal{T}_{UA_k} . Each node $N_m \in \mathcal{T}_{UA_k}$ is handled as follows depending on its type. If $m = AC$, the appropriate wrapper is notified for actuating a device. Then, if $m = VSR$, the engine setups the required data inputs based on its children nodes. This is performed by creating consumers that subscribe to receive data from a virtual/physical sensors. Note that if $m = PSR$, the communication with its corresponding wrapper is handled by the VSR sensor parent node. Finally, if m is the root VSR/PSR node, the engine calls the corresponding virtual sensor or wrapper, respectively. In the case of a VSR root that call will trigger the chain of calls to predecessor nodes.

Experimental Evaluation

We implemented a prototype of SemIoTic [5] and a sample application. The prototype, domain models, wrappers, virtual sensors, and application developed for the experiments are available at [1].

Developing an application with SemIoTic. The application developed showcases an exploratory discovery of the space by enabling users to define requests to explore the spaces defined in the model and their properties. Then, the application guides the user (through a GUI) to define the sal parameters of the user action to actuate the property for controlling the temperature of a selected space when its occupancy reaches a percentage of its capacity. It also generates another sal request to retrieve a stream of occupancy and temperature data for the room. After posing the user actions to SemIoTic, the application generates graphs to display the obtained data (see Fig. 7).

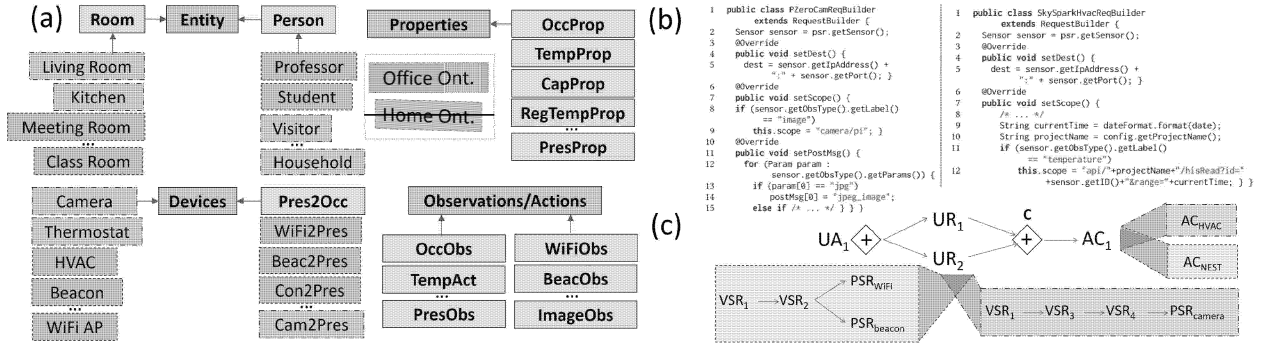


Figure 6: Snippets of (a) Two domain models based on semic, (b) Code of two wrappers, and (c) Tree generated to handle a sample user action.

We developed domain models, wrappers, and virtual sensors for two different scenarios: a smart office building and a smart home. For the domain models, we imported the semic metaontology in the Protégé tool and extended it, with the appropriate classes and instances (e.g., rooms, occupancy, temperature, people, presence, etc.), to describe our smart Donald Bren Hall building at University of California, Irvine and a smart home (see Fig. 6(a)). We developed sensor wrappers to interact with real sensors in both scenarios including, among others, WiFi Access Points, bluetooth beacons, cameras attached to Raspberry Pis, HVAC sensors connected to the SkySpark framework (see Fig. 6(b)). Finally, we developed virtual sensors to generate occupancy, by counting the number of people in a space, and to determine where individuals are, using approaches such as exploiting connectivity data (e.g., from WiFi APs or beacons) or detecting faces on images (using OpenCV).

Handling runtime user actions with SemIoTic. We deployed two instances of SemIoTic (i.e., SemIoTic-Home and SemIoTic-Building) with the previous content. We run the application, which shows a list of spaces extracted from the domain model, and select “Room 111” and “Living Room” in the SemIoTic-Building and SemIoTic-Home instances, respectively. The application generates the following user actions that are posed into SemIoTic-Building to control the temperature of the room and to retrieve a stream of occupancy and temperature readings of the room, respectively (similar requests are generated for SemIoTic-Home):

$\langle \text{Room111}, \text{ControlTemperature}, \text{OccupancyProp} > 0.5 \times \text{CapacityProp} \rangle \quad \langle \text{Room111}, \text{OccupancyProp} / \text{TemperatureProp}, \emptyset \rangle$

Every SemIoTic instance receives and handles the request as explained before. In both scenarios, for the action to control the temperature, SemIoTic detects the need to retrieve the room’s occupancy and capacity properties first and then actuate the property to control the temperature if the condition is met. As both domain models define a virtual sensor that can retrieve occupancy observations (which is the value that the occupancy property requires) this virtual sensor is included in the tree. Next, SemIoTic discovers three virtual sensors in the model (using WiFi observations, bluetooth observations, and images, respectively) that can generate the presence data required as input to the occupancy counter sensor. These are also included in the tree as possible plans. Then, each virtual sensor appends a request

to physical sensors PSR_i for consuming their output data (WiFi APs, bluetooth beacons, and cameras).

When checking realizability and feasibility of the plan, SemIoTic-Building detects that there are no cameras covering room 111 and that the virtual sensor based on beacon observations provides a more accurate answer than the one using WiFi data. Thus, the final plan selected (see Fig. 6(c)) involves a request to the virtual sensor that generates occupancy data from presence data (VSR_1), which is the same in the plan for the home, followed by a request to the virtual sensor that generates presence data using bluetooth beacon observations (VSR_2) followed by requests to the three beacons covering the room (PSR_{1-3}). SemIoTic-Home follows a similar process by detecting a camera with the living room in its view frustum (there are no WiFi APs or beacons deployed). This way, it generates a plan (see Fig. 6(c)) that involves a request to the virtual sensor producing presence based on counting faces (VSR_3), followed by a request to the virtual sensor that extracts faces from images (VSR_4), followed by a request to the camera to capture images. At execution time, each instance calls the appropriate virtual sensors and sensor wrappers according to the selected plan.

The application populates the temperature and occupancy graph (see Fig. 7) by using their definitions in the domain model. Underneath SemIoTic-Building retrieves data from HVAC thermometers and beacons whereas SemIoTic-Home uses a Raspberry Pi with a thermometer and a videocamera. Notice in Fig. 7 how, in the case of the building, the room starts getting full towards 9am (the starting of the meeting) and there is an increase on the temperature. When the occupancy crosses the boundary defined (in this case 75% of the capacity) the parallel user action retrieves these data and turns on the AC. Then, after some delay, the temperature starts lowering down. In the case of the smart home the situation is similar even when the underlying sensors are completely different.

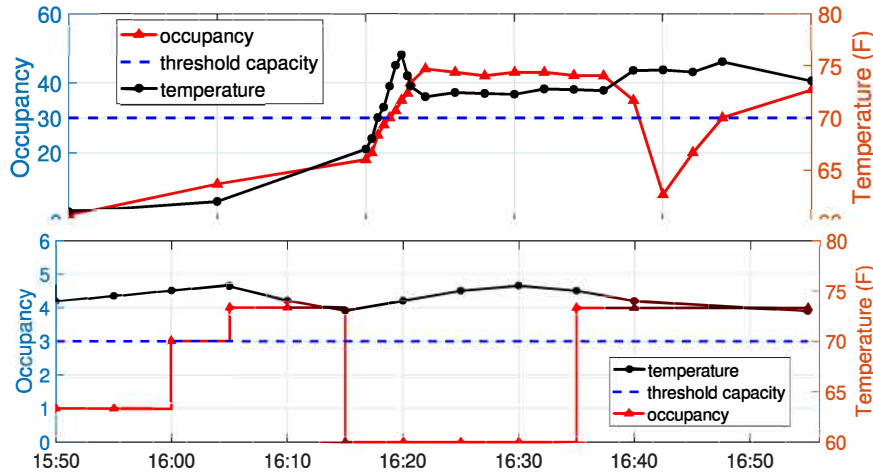


Figure 7: Graphs displayed by the application using SemIoTic.

In the case of the smart home, we decided to include a third user action: a policy that restricts the sharing of data. The policy ($\langle \text{Mary}, \text{LocationProp}, \text{LocationProp}=\text{PrivateSpace},$

`capture, deny`) is processed in parallel with the other two actions and, when translated, prevents access to video camera data as it can be used to derive the location of Mary. Notice that the occupancy curve of the room drops to zero at 16:15. This is the moment when we simulated Mary arriving in the living room. At that moment, video camera data cannot be captured and this prevents the virtual sensor to obtain occupancy as the plan becomes unrealizable. Notice also, that another consequence is that the temperature starts increasing slightly as the action that controls the temperature cannot be processed due to the lack of occupancy data. At 16:35 Mary leaves the leaving room and the processing of both actions gets resumed.

As we have seen, the developer of the application did not have to deal with/understand device related information. Through SemIoTic, the developer relied on the information described in the domain model about entities and their properties and specified sal user actions. Also, the same application run across different spaces without code modifications. This is possible thanks to the space-agnostic and ontology driven architecture of SemIoTic and the definition of both the domain models and wrappers/virtual sensors. Indeed, in this case the development effort of the application developer has been reduced due to the administrator having defined the domain models and due to the wrapper/virtual sensor developer.

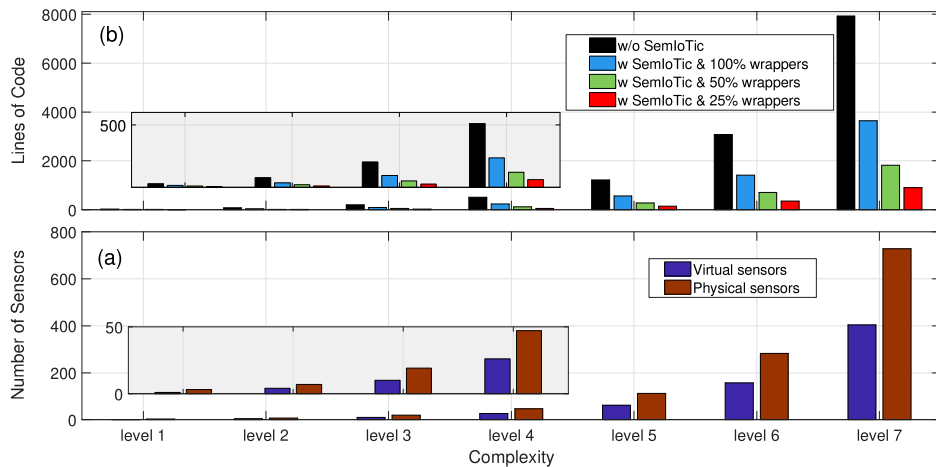


Figure 8: Development effort with (w) and without (w/o) SemIoTic.

Scaling up the use case complexity. For the previous experiment, we required to create 5 wrappers and 4 virtual sensors to facilitate the development of the app, which is significantly less than the effort it would require to create the same application without SemIoTic. In that case, there is a need to implement the logic of the application in order to support complex actions, define the metadata of spaces/devices, implement the logic for the processing of multiple inputs (data sources), implement code for accessing heterogeneous devices, configure destination URIs for handling every device action, etc. In addition, this process must be repeated to deploy the application in different spaces (due to the differences

of the existing devices) and to handle different user actions. Also, this development effort will increase significantly with the increase on the level of the complexity of user actions (e.g., if a complex execution plan is required involving a large chain of virtual sensors). In the following experiment we compare the required effort for developing an application with and without using SemIoTic.

We developed an algorithm that generates scenarios with different levels of complexity. A scenario includes a set of sensors and virtual sensors –with varying number of inputs– which results in multiple execution plans. The algorithm starts by generating a virtual sensor and then a random number of inputs (virtual or physical sensors) by considering a maximum number of inputs $n_{in}(vs_j)$ defined as a parameter. Then, this process is performed iteratively for each new virtual sensor until the execution plan reaches a given level of complexity n_{cl} . The output is an execution plan involving $|PS|$ physical sensors and $|VS|$ virtual sensors. Using this algorithm we generated scenarios with increasing n_{cl} (from 1 to 7) and with $n_{in}(vs_j) = 4$. For each level the algorithm created 500 different scenarios and then computed the average $|PS|$ and $|VS|$.

The algorithm also estimates the development effort to implement these plans. For that, it takes into account the cost in terms of *lines of code* (LoC) to be developed (without considering common tasks that have to be developed with and without SemIoTic such as the definition of the logic/GUI of the app, logic of the virtual sensing task, definition of metadata of the space and devices). Let LoC^{with} be the number of LoC required to develop with SemIoTic as $LoC^{with} = n_{loc}^{wrap} \times |PS|$. Where n_{loc}^{wrap} is the average LoC required to develop the data and scope mapping of a wrapper. The metric does not include virtual sensor development as we provide developers with the appropriate generic artifact so that they just need to implement the logic of the virtual sensing task. We measured the average n_{loc}^{wrap} to be 5 LoC in the simple data type wrappers generated for the previous experiments (e.g., Fig. 6(b)). Then, let $LoC^{w/o}$ be the number of LoC to develop without SemIoTic as $LoC^{w/o} = \sum_{j=1}^{|VS|} n_{loc}^{in} \times n_{in}(vs_j)$. Where n_{loc}^{in} is the average LoC required to setup an input data source (setup a consumer, configure its URI, etc). Defining n_{loc}^{in} is challenging as this may differ depending on the protocol and specific device (e.g., in [6] the authors setup an MQTT subscriber by using 8 LoC without considering the data mapping task). We assume the best case scenario when developing plans without SemIoTic by considering 5 LoC for setting up a consumer and 2 additional LoC to perform data mapping for a simple message type. Thus, we consider $n_{loc}^{in} = 7$.

Fig. 8(a) shows a plot of the average $|PS|$ and $|VS|$ with increasing level of complexity. For instance, a plan with complexity 5 would require interacting with 62 virtual sensors and 112 physical sensors. Fig 8(b) shows the number of LoC required to develop an application, with and without SemIoTic, vs. the complexity of the scenario. In the case of development with SemIoTic we consider situations with different number of wrappers required as a percentage of the number of physical sensors (as some sensors could be of the same type/brand and handled by the same wrapper). For instance, in our previous experiment for the smart building the ratio was less than 1% as we developed 4 wrappers in total for cameras (around 40), HVAC sensors (around 7K), WiFi APs (around 60), and bluetooth

beacons (around 200). Fig 8(b) shows, for instance, that developing the complexity 5 plan requires 1.2K LoC without SemIoTic compared to 500, 300, 100, and 30 LoC using SemIoTic (having to develop wrappers for 100%, 50%, 25%, and 5% of the total physical sensors). Based on the results in Fig. 8(b), developing an application using SemIoTic reduces the effort (in terms of LoC) by 97% to 55%. Notice that this experiment measures development effort just in terms of LoC and thus it does not consider other efforts which SemIoTic alleviates. For instance, the effort required to find/understand/utilize libraries to handle interactions with different protocols, to develop the logic to handle such complex plans, to handle user needs in a more semantically meaningful way, etc.

3.2.1.2 Integrating PETs into a Smart Space Platform

One representative domain of smart space where the IoT is opening the door to potential benefits is that of *smart buildings*. Here traditional HVAC (heating, ventilating, and air conditioning) systems are being enhanced with functionalities that ties to beacons, presence sensors, cameras, and personal devices such as smartphones carried by the building’s inhabitants. The reliance on the collection of data in smart buildings contradicts the expectations of privacy. Especially since in IoT environments, such as smart buildings, users are less likely to be aware of the technologies with which they might be interacting.

We describe a framework for smart spaces which integrates different PETs includes three main components. First, *IoT Resource Registries* (IRRs) which broadcast data collection policies and sharing practices of the IoT technologies with which users interact. Second, *IoT Assistants* which selectively notify users about the policies advertised by IRRs and configure any available privacy settings. Third, *privacy-aware smart buildings*, which publish building policies (e.g., through IRRs), receive the privacy settings of users (e.g., from IoTAs) and enforce them when collecting user data or sharing it with services¹.

To make this possible, it is important to have a language for expressing and communicating the space/building’s policies as well as the privacy preferences of its inhabitants. While the existing privacy policy languages are expressive [7], they do not completely support capturing the policies of the building and preferences of its inhabitants regarding their data. Therefore, this section also presents an overview of a language which can be used for informing users about policies on what data is collected, how it will be safeguarded, what it will be used for, and the choices a user has with respect to these policies.

Privacy Threats in Current Smart Spaces Scenarios

Smart spaces capture a digital representation of a dynamically evolving space at any point in time for purposes such as comfort and security. But this representation might contain distinct patterns which can reveal the absence or presence of people and their activities, potentially resulting in the disclosure of data that people might not feel comfortable disclosing (e.g.,

¹Note that, in order to differentiate the capture policies of the building against user-defined policies, this chapter refers to the former as building policies and to the latter as user preferences. The rest of the thesis focuses on user-defined preferences and hence the term policies will be used to refer to those.

where they go, what they do, when and with whom they spend time, whether they are healthy and more) [8].

For example, when a user connects to a WiFi AP in DBH, this event is logged for security purposes (the information logged includes the MAC address of the device and AP, and a timestamp) as part of the *building policy*. Using background knowledge (e.g., the location of the AP) it is possible to infer the real-time location of a user. Also, using simple heuristics (e.g., non-faculty staff arrive at 7 am and leave before 5 pm, graduate students generally leave the building late, and undergrads spend most of the time in classrooms), it is possible to infer whether a given user is a member of the staff or a student. Furthermore, by integrating this with publicly available information (e.g., schedules of professors and the courses they teach or event calendars), it would be possible to identify individuals. Some people may not object to such data collection, while others might. One challenge associated with privacy is that often not all users feel comfortable about the same data practices. Therefore, it is important to understand *user preferences* and *expectations* with respect to the information collected and used by a system like BMS [9, 10].

Integrating User-Defined Privacy Policies

Figure 9 outlines how a user (who will be referred to as *Mary* from now onward for ease of explanation) interacts with this infrastructure. In particular, consider a scenario where a building, DBH, is managed by a privacy-aware smart building management system (in this case TIPPERS) and users have personal assistants which handle their privacy preferences (in this case IoTA).

First, the building admin of DBH uses the smart building management system (such as TIPPERS) to define policies regarding the collection and management of data within the building (step (1) in Figure 9). Based on these policies, the different sensors in the building are actuated and data from them, some of which might be related to its inhabitants (step (2)), is captured and stored (step (3)). These policies are made publicly available through one or more IoT Resource Registries (step (4)). As Mary walks into the building carrying her smartphone with IoTA installed on it, the IoTA discovers available registries that pertain to resources in her vicinity and obtains machine-readable privacy policies detailing the practices of resources close to her location (step (5)). The IoTA displays summaries of relevant elements of these policies to the user (step (6)) by focusing on the elements of a policy that are important respect to the users privacy preferences. This is done using a model of Mary's privacy preferences learned over time. This might include information about those data collection and use of practices she cares to be informed about (step (7)). If a policy identifies the presence of settings, the IoTA can also use knowledge of Mary's privacy preferences to help configure these settings by communicating with TIPPERS (e.g., submitting requests to change settings) (step (8)). If a service later requests TIPPERS about Mary's location (step (9)), the request will be processed according to the settings communicated by Mary's IoTA to TIPPERS (e.g., the request might be rejected, if Mary's IoTA requested to opt-out of location sharing; step (10)).

To implement this interaction, a machine-readable policy language, as a mechanism to

completely met depending on other policies and user preferences existing in the same space.

Smart buildings such as DBH also provide services, built on top of the collected sensor data, to the inhabitants of the building. Two examples of such services operating at DBH are 1) *Smart Concierge* service, which helps users locate rooms, inhabitants and events in the building, 2) *Smart Meeting* service, which can help organize meetings more efficiently. These services take information from the user captured by the building (e.g., their current location) and return interesting information (e.g., nearest coffee machine). In addition to services provided by the space, there could be other third-party services running on top of the smart space management system. For example, a food delivery company can automatically locate and deliver food to building inhabitants during lunch time.

While using a service inside the space, a user can also specify their policies in the form of permissions allowed for the service. This is similar to how the permissions are managed in mobile apps. This allows a user to directly review what information the service requests and for what purpose.

Communicating Policies and Preferences.

Space policies and user preferences have context specific requirements that need to be captured and communicated in a flexible manner. In this section, we first describe the various elements of our machine-readable policy. Second, we describe a high-level language schema that can be used to capture such policy.

For expressing a space policy, a semi-structured language would be the best fit as the user is cognizant of the IoT space itself. In the case of a user preference, the goal is to reduce privacy fatigue as much as possible and therefore a natural language interface or a privacy assistant like IoTA mentioned earlier would be more suitable.

Space Specific Policy Elements. There are different elements in a space that have to be represented in policies such as space, users, sensors and services. For the elements described below, we use existing ontologies if available.

1) *Spatial Model* includes information about infrastructure, such as buildings, floors, rooms, corridors, and is inherently hierarchical. The spatial model also supports operators such as “contained”, “neighboring”, and “overlap”.

2) *User Profile* models the concept of people in the environment. Profiles can be based on groups (students, faculty, staff etc.) and share common properties (e.g., access permissions). A user can have multiple profiles which includes information such as department, affiliation, and office assignment in our sample scenario.

3) *Sensor* describes the entity which captures information about its environment. Each sensor has a sensor type and can produce a reading based on its type. Sensors of the same type can be organized into sensor subsystems. Examples of such subsystems are camera subsystem, beacon subsystem, and HVAC subsystem (modelled using the haystack² ontology and Semantic Sensor Network ontology [11]).

²<http://project-haystack.org>

4) *Settings of a sensor* is a set of valid parameters associated with the sensor which determines its behavior (e.g., for a camera it could be the capture frequency or the resolution of the image). A sensor is actuated based on the parameters specified in its current settings. A sensor can have multiple settings dictated by its type.

5) *Observation* models the type of data captured by a sensor based on the type and settings associated with it. Each observation has a timestamp and a location (determined based on whether the sensor is mobile or fixed) associated with it.

6) *Service Model* describes the services that run on top of smart building systems and provide interesting information to the users. The service model captures meta-data about the service such as the developer (e.g., building owner or third party), permissions to sensors, and observations. This model also describes details about the service itself such as the information returned or functionality provided.

Privacy Specific Policy Elements. While space and sensor specific models can capture information about different entities, there is a need for a description of the data collection practices in a building from the perspective of a user. Peppet [12] analyzed privacy policies of companies that manufacture IoT devices and concluded that through these policies, users not only want to be informed about what data is collected by which devices and for what purposes, but also about the granularity of data collection (whether or not it is aggregated or anonymized) and with whom the data is shared. Based on this, we introduce the following policy elements to model a user's privacy settings.

1) *Context* describes meta information about the building and the BMS that point users to general information (e.g., who is responsible for data collection in a building, where are sensors located, and whom to contact when it comes to questions regarding the policy). This meta information can also contain a general description of data security and ownership of information which are relevant to the user.

2) *Data collected and inferred.* While the observation model captures information about the data collected, a user might be more interested in knowing what can be inferred from the collected data. Therefore, it is important to specify the abstract information that can be inferred from an observation captured by a sensor. For example, to model the occupancy of a room, it would be better to describe it as "if a room is occupied by anyone" compared to an observation model which might only have information such as "images from camera", "logs from WiFi APs", etc. Data collection description also contains information about the granularity of the data collected as granularity can directly impact the capability of inference.

3) *Purpose* models the requirement of data collection which is closely related to a service that uses this data. In a smart space, some data collections such as temperature monitoring serves a straightforward purpose for setting the thermostat, but for other data collections such as the information of connecting to WiFi APs can be used for different purposes (e.g., for logging as well as to track the location of a particular MAC address). We are currently working on a taxonomy to model purpose which includes information about whether or not the data is shared (e.g., with law enforcement officers for security purposes) and for how long it will be stored (i.e., retention).

Overview of the Language Schema. Based on the aforementioned elements, we are

designing a language schema that is capable of capturing both space policies and user preferences. In the following we give an overview of the language by representing some of the previous examples. We use a JSON-Schema v4³ for the representation. We choose JSON over other formats mainly because of the rapid adoption of JSON-based REST APIs.

Figure 10 shows how *Policy 2* (“Location tracking for emergency response”) can be expressed using the language. The first part of the language expresses the general information about the location and sensor type (in this case location is DBH at UCI with WiFi APs being the sensors) whereas the second part expresses the data collection purpose (emergency response), data type, and retention period of the data itself.

```
{
  "resources": [
    {
      "info": {
        "name": "Location tracking in DBH"
      },
      "context": {
        "location": {
          "spatial": {
            "name": "Donald Bren Hall",
            "type": "Building"
          },
          "location_owner": {
            "name": "UCI",
            "human_description": {
              "more_info": "http://ics.uci.edu"
            }
          }
        },
        "sensor": {
          "type": "WiFi Access Point",
          "description": "Installed inside the building and covers rooms and corridors"
        }
      },
      "purpose": {
        "emergency response": {
          "description": "Location is stored continuously"
        }
      },
      "observations": [
        {
          "name": "MAC address of the device",
          "description": "If your device is connected to a WiFi Access Point in DBH, its MAC address is stored"
        }
      ],
      "retention": {
        "duration": "P6M"
      }
    }
  ]
}
```

Figure 10: Policy related to data collection inside DBH.

In case of the policies related to services such as the *Smart Concierge* ⁴ can be expressed as shown in Figure 11. The first part describes the information required by the service and the second part shows the purpose of collecting this information.

Concerning user’s preference settings, the language can express choices related to policies and services. In the context of *Smart Concierge* service, Figure 12 shows options for the different granularities at which location data can be collected. Thus, if a user is comfortable with sharing fine-grained location data with the Concierge service for directions then our language can capture such *Preference*.

³<http://json-schema.org>

⁴Smart Concierge is a service that acts as a smartspace assistant to a building occupant/user empowering the user with information about the current state as well as past state of the building (e.g., room occupancy, ongoing events), and other occupant (e.g., their location).

```
{
  "observations": [
    {
      "name": "wifi_access_point",
      "description": "Whenever one of your devices connects to the DBH WiFi its MAC address is stored"
    },
    {
      "name": "bluetooth_beacon",
      "description": "When you have Concierge installed and your bluetooth senses a beacon, the room you are in is stored"
    }
  ],
  "purpose": {
    "providing_service": {
      "description": "Your location data is used to give you directions around the Bren Hall."
    },
    "service_id": "Concierge"
  }
}
```

Figure 11: Policy related to a service in the building.

```
{
  "settings": [
    {
      "select": [
        {
          "description": "fine grained location sensing",
          "on": "http://tippers/user/concierge?beacon=opt-in&wifi=opt-in"
        },
        {
          "description": "coarse grained location sensing",
          "on": "http://tippers/user/concierge?beacon=opt-out&wifi=opt-in"
        },
        {
          "description": "No location sensing",
          "on": "http://tippers/user/concierge?beacon=opt-out&wifi=opt-out"
        }
      ]
    }
  ]
}
```

Figure 12: Privacy settings available.

3.2.2 Scaling Encryption and Secret-Sharing Techniques

Many *untrusted cloud computing* platforms (*e.g.*, Amazon AWS, Google App Engine, and Microsoft Azure) offer database-as-a-service using which data owners, instead of purchasing, installing, and running data management systems locally, can outsource their databases and query processing to the cloud. Such cloud-based services available using the pay-as-you-go model offers significant advantages to both small, medium and at times large organizations. The numerous benefits of public clouds impose significant security and privacy concerns related to sensitive data storage (*e.g.*, sensitive client information, credit card, social security numbers, and medical records) or the query execution. The untrusted public cloud may be an *honest-but-curious* (or passive) adversary, which executes an assigned job but tries to find some meaningful information too, or a malicious (or active) adversary, that may tamper the data or query. Such concerns are not a new revelation – indeed, they were identified as a key impediment for organizations adopting the database-as-a-service model in early work on data outsourcing [13, 14]. Since then, security/confidentiality **challenges** have been extensively studied in both the cryptography and database literature, which has resulted in many techniques to achieve *data privacy*, *query privacy*, and *inference prevention*. Existing work can loosely be classified into the following two categories:

1. **Encryption based techniques.** *E.g.*, order-preserving encryption [15], deterministic encryption (Chapter 5 of [16]), homomorphic encryption [17], bucketization [13], searchable encryption [18], private informational retrieval (PIR) [19], practical-PIR (P-PIR) [20], oblivious-RAM (ORAM) [21], oblivious transfers (OT) [22], oblivious polynomial evaluation (OPE) [23], oblivious query processing [24], searchable symmet-

ric encryption [25], and distributed searchable symmetric encryption (DSSE) [26]).

2. **Secret-sharing [27] based techniques.** *E.g.*, distributed point function [28], function secret-sharing [29], functional secret-sharing [30], accumulating-automata [31, 32], and others [33, 34, 35].

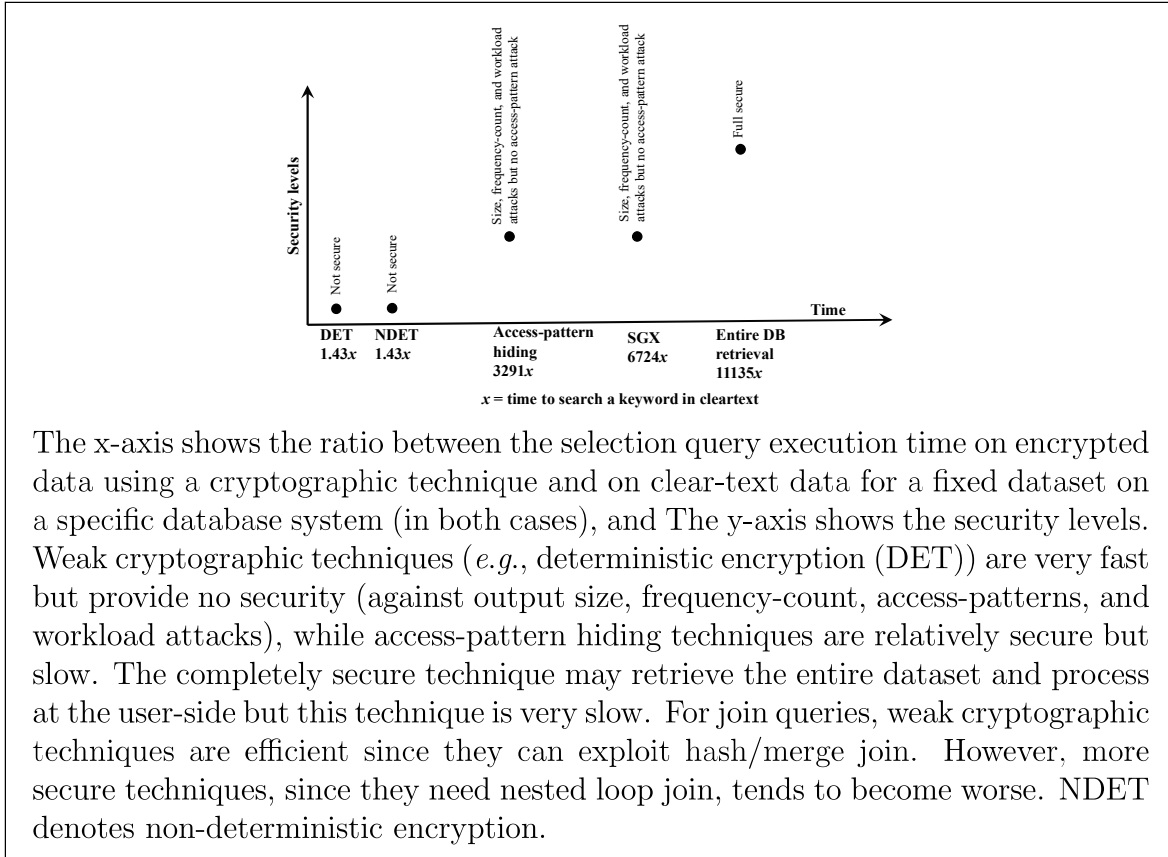


Figure 13: Comparing different cryptographic techniques.

While approaches to compute over encrypted data and systems supporting such techniques are plentiful, secure data outsourcing and query processing remain an open challenge. Existing solutions suffer from several limitations. First, cryptographic approaches that prevent leakage, *e.g.*, fully homomorphic encryption coupled with ORAM, simply do not scale to large data sets and complex queries for them to be of practical value. Most of the above-mentioned techniques are not developed to deal with a large amount of data and the corresponding overheads of such techniques can be very high (see Figure 13 comparing the time taken for TPC-H selection queries under different cryptographic solutions). To date, a scalable non-interactive mechanism for efficient evaluation of join queries based on homomorphic encryption that does not leak information remains an open challenge. Systems such as CryptDB [36] have tried to take a more practical approach by allowing users to explore

the tradeoffs between the system functionality and the security it offers. Unfortunately, precisely characterizing the security offered by such systems given the underlying cryptographic approaches have turned out to be extremely difficult. For instance, [37, 38] show that when order-preserving and deterministic encryption techniques are used together, on a dataset in which the entropy of the values is not high enough, an attacker might be able to construct the entire plain-text by doing a frequency analysis of the encrypted data. While mechanisms based on secret-sharing [27] are potentially more scalable, splitting data amongst multiple non-colluding cloud operators (an assumption that is not valid in a general setting) incurs significant communication overheads and can only support a limited set of selection and aggregation queries efficiently.

While the race to develop cryptographic solutions that (i) are efficient, (ii) support complex SQL queries, (iii) offer provable security from the application’s perspective is ongoing, we explore new paths to scale the cryptographic techniques for large-size databases. In particular, we focus on:

Partitioned computing: is an approach for situations when only part of the data is sensitive, while the remainder (that may consist of the majority) is non-sensitive. In particular, we consider a **partitioned computation model** that exploits such a classification of data into sensitive/non-sensitive subsets to develop efficient data processing solutions with **provable security guarantees**. Partitioned computing potentially provides significant benefits by (i) avoiding (expensive) cryptographic operations on non-sensitive data, and, (ii) allowing query processing on non-sensitive data to exploit indices. We provide partitioned computing for the hybrid cloud (in Section 3.2.2.1) and for the public cloud (in Section 3.2.2.2).

Secret-sharing-based computing: in an approach when the dataset is outsourced in the form of additive or multiplicative shares over multiple non-colluding servers. We design aggregation query execution algorithm for such a setting, while avoid any communication among the server before/during/after a computation, in Section 3.2.2.3. By avoiding communication among servers, we decrease the overall computation time.

3.2.2.1 Partitioned Computations at the Hybrid Cloud

In this section, we provide an approach to execute SQL style queries efficiently in a hybrid cloud while guaranteeing that sensitive data is not leaked to the (untrusted) public machines. At the abstract level, the technique partitions data and computation between the public and private clouds in such a way that the resulting computation (i) minimizes the execution time, and (ii) ensures that there is no information leakage. Information leakage, in general, could occur either directly by exposing sensitive data to the public machines, or indirectly through inferences that can be made based on selective data transferred between public and private machines during the execution.

The problem of securely executing queries in a hybrid cloud naturally leads to two inter-related subproblems:

Data distribution: How is data distributed between private and public clouds? Data distribution depends on factors such as the amount of storage available on private machines, expected query workload, and whether data and query workload is largely static or dynamic.

Query execution: Given a data distribution strategy, how do we execute a query securely and efficiently across the hybrid cloud, while minimizing the execution time and obtaining the correct final outputs?

Since data is stored on public cloud in the clear text, data distribution strategy must guarantee that sensitive data resides only on private machines. Non-sensitive data, on the other hand, could be stored on private machines, public machines, or be replicated on both. Given a data distribution, the query processing strategy will split a computation between public and private machines while simultaneously meeting the goals of good performance and secure execution.

Split Strategy

In order to ensure a secure query execution, we develop a *split strategy* for executing SQL queries in the hybrid cloud setting. In a split strategy, a query Q is partitioned into two sub-queries that can be executed *independently* over the private and the public cloud respectively, and the final results of the query can be computed by appropriately merging the results of the two sub-queries. In particular, a query Q on dataset D is split as follows:

$$Q(D) = Q_{merge}(Q_{priv}(D_{priv}), Q_{pub}(D_{pub})) \quad (1)$$

where Q_{priv} and Q_{pub} are private and public cloud sub-queries respectively. Q_{priv} is executed on the private subset of D (i.e., D_{priv}); whereas Q_{pub} is performed over the public subset of D (i.e., D_{pub}). Q_{merge} is a private cloud merge sub-query that reads the outputs of former two sub-queries as input and creates the outputs equivalent to that of original Q . We call such an execution strategy as *split-strategy*.

Two aspects of *split-strategy* are noteworthy:

1. It offers full security, since the public machines only have access to D_{pub} that do not contain any sensitive data. Moreover, no information is exchanged between private and public clouds during the execution of Q_{pub} , resulting in the execution at the public cloud to be *observationally equivalent* to the situation where D_{priv} could be any random data.
2. Split-strategy gains efficiency by executing Q_{priv} and Q_{pub} in parallel at the private and public cloud respectively, and furthermore, by performing inter-cloud data transfer at most once throughout the query execution. Note that the networks between private and public clouds can be significantly slower compared to the networks used within clouds. Thus, minimizing the amount of data shuffling between the clouds will have a big performance impact.

Split strategy, and its efficiency, depends upon the data distribution strategy used to partition the data between private and public clouds. Besides storing sensitive data, the private cloud must also store part of non-sensitive data (called *pseudo sensitive data*) that may be needed on the private side to support efficient query processing. For instance, a join query may necessitate that non-sensitive data be available at the private node in case sensitive records from one relation may join with non-sensitive records in another. Since in the split-execution strategy, the two subqueries execute independently with no communication, if we do not store non-sensitive data at the private side, we will need to transfer entire relation to the private side for the join to be computed as part of the merge query.

Split-strategy for selection or projection. An efficient *split-strategy* for selection or projection operation is straightforward. In this case, Q_{priv} is equivalent to the original query Q , but is performed only over sensitive records in D_{priv} . Likewise, $Q_{pub} = Q$, but only runs over D_{pub} . Finally, $Q_{merge} = Q_{priv} \cup Q_{pub}$.

Figure 14: Example relations.

Split-strategy for equijoin. An efficient *split-strategy* for performing a join query such as $Q = R \bowtie_C S$ is more complex. To see this, consider the relations R and S as shown above in Figure 14, where sensitive portions of R and S are denoted as R_s and S_s , respectively, and remaining fraction of them are non-sensitive, denoted as R_{ns} and S_{ns} , and the join condition is $C = (R.region = S.Region)$. Let us further assume that R_{ns} and S_{ns} , besides being stored in the public cloud are also replicated on the private cloud.

The *naive split-strategy* for $R \bowtie_C S$ would be:

- $Q_{pub} = R_{ns} \bowtie_C S_{ns}$
- $Q_{priv} = (R_s \bowtie_C S_s) \cup (R_s \bowtie_C S_{ns}) \cup (R_{ns} \bowtie_C S_s)$.

Note that if Q is split as above, Q_{priv} consists of three subqueries which scan 2, 3, and 3 tuples in R and S respectively resulting in 8 tuples to be scanned and joined. In contrast, if we simply executed the query Q on the private side (notice that we can do so, since R and S are fully stored on the private side), it would result in lower cost requiring scan of 6 tuples on the private side. Indeed, the overhead of the above split strategy increases even further if we consider multiway joins (e.g., $R \bowtie_C S \bowtie_{C'} T$) compared to simply implementing the multiway join locally. Thus, if we use split-strategy for computing $R \bowtie_C S \bowtie_{C'} T$, where C' is $S.Region = T.Region$, then the number of tuples that are scanned/joined in the private cloud will be much higher than that of the original query.

A modified approach for equijoin. The cost of executing Q in the private cloud can be significantly reduced by pre-filtering relations R and S based on sensitive records of the other relation. To perform such a pre-filtering operation, the tuples in the relations R_{ns} and S_{ns} have to be co-partitioned based on whether they join with a sensitive tuple from the other table under condition C or not.

Let R_{ns}^S be a set of non-sensitive tuples of R that join with any sensitive tuple in S . In our case, $R_{ns}^S = \langle \text{apple}, 1 \rangle$. Similarly, let S_{ns}^R be non-sensitive tuples of S that join with any record from R_s , *i.e.*, $\langle \text{Chris}, 1 \rangle$. In that case, the new private side computation can be rewritten as:

$$(R_s \cup R_{ns}^S) \bowtie_C (S_s \cup S_{ns}^R). \quad (2)$$

Thus, the scan and join cost of this new plan at the private cloud is 4, which is lower compared to computing the query entirely on the private side that had a cost of 6.

Guarded join. The above mentioned modified strategy, nonetheless, introduces a new challenge. Since $R_{ns}^S \bowtie_C S_{ns}^R$ is both repeated at public and private cloud, the output of $R_{ns}^S \bowtie_C S_{ns}^R$, $\langle \text{apple}, \text{Chris}, 1 \rangle$, is computed on both private and public clouds. To prevent this, we do a guarded join (\bowtie') on the private cloud, which discards the output, if it is generated via joining two non-sensitive tuples. This feature can easily be implemented by adding a column to R and S that marks the sensitivity status of a tuple, whether it is sensitive or non-sensitive, and then by adding an appropriate selection after the join operation. In other words, the complete representation of private side computation for $R \bowtie_C S$ would be

$$\sigma_{R.sens=true \vee S.sens=true}((R_s \cup R_{ns}^S) \bowtie_C (S_s \cup S_{ns}^R)) \quad (3)$$

where *sens* is a boolean column (or partition id) appended to relations R and S on the private cloud. Assume that it is set to true for sensitive records and false for non-sensitive records.

Challenges. There exist multiple challenges in implementing this new approach. First challenge is the cost of creating R_{ns}^S and S_{ns}^R beforehand. Extracting these partitions for a query might take as much time as executing the original query. However, the costs are amortized since these relations are computed once, and used multiple times to improve join performance at the private cloud.

The second challenge is the creation of co-partitioning tables for complex queries. For instance, in case of a query $R \bowtie_C S \bowtie_{C'} T$, the plan would be to first compute results of $R \bowtie_C S$, and then to join them with T . However, if we do the private side computation of $R \bowtie_C S$, based on Equation 2 (no duplicate filtering) and join the results with T , then we will not be able to obtain the complete set of sensitive $R \bowtie_C S \bowtie_{C'} T$ results.

To see this, consider the sensitive record (in Figure 14) $\langle \text{Japan}, 2 \rangle$ in T that joins with non-sensitive $\langle \text{grape}, 2 \rangle$ tuple in $R - R_{ns}^S$ or joins with non-sensitive $\langle \text{James}, 2 \rangle$ tuple from $S - S_{ns}^R$. Thus, the non-sensitive records of R and S has to be co-partitioned based on the sensitive records of T via their join paths from T . In $R \bowtie_C S \bowtie_{C'} T$, the join path from T to

R is $T \bowtie_{C'} S \bowtie_C R$ and from T to S is $T \bowtie_{C'} S$. Similarly, the non-sensitive T records has to be co-partitioned based on the sensitive R and S records via join paths specified in the query.

Final challenge is in maintaining these co-partitions and feeding the right one when an arbitrary query arrives. Given a workload of queries and multiple possible join paths between any two relations, each relation R in the dataset may need to be co-partitioned multiple times. This implies that any non-sensitive record r of R might appear in more than one co-partition of R . So, maintaining each co-partition separately might be unfeasible in terms of storage. However, the identifiers of each co-partition that record r belongs to can be embedded into r as a new column. We call such a column as the *co-partition* (CPT) column. Note that CPT column is *only defined* on the private cloud data, since revealing it to public cloud would violate our security requirement.

CPT column initially will be set to null for sensitive tuples in the private side, since the co-partitions are only for non-sensitive tuples. Thus, it can further be used to serve another purpose, indicating the sensitivity status of a tuple r by setting it to “sens” only for sensitive tuples.

Join path. To formalize the concept of co-partitioning, we first need to define the notion of join path. Let R_i be a relation in our dataset D , and let Q be a query over the relation R_i . We say a join path exists from a relation R_j to R_i , if either R_i is joined with R_j directly based on a condition C , *i.e.*, $R_j \bowtie_C R_i$, or R_j is joined with R_i *indirectly* using other relations in Q . A join path p can be represented as a sequence of relations and conditions between R_j and R_i relations. Let $PathSet$ be the set of all join paths that are extracted either from the expected workload or a given dataset schema.

$$PathSet_i = \{\forall p \in PathSet : \text{path } p \text{ ends at relation } R_i\}. \quad (4)$$

Let $CP(R_i, p)$ be the set of non-sensitive R_i records that will be joined with at least one sensitive record from any other relation R_j via the join path p . Note that p starts from R_j and ends at R_i that can be used as an id to $CP(R_i, p)$. Any $CP(R_i, p)$ is called as “co-partition” of R_i . Given these definitions, the CPT column of a R_i record, say r , can be defined as:

$$r.CPT = \begin{cases} \text{sens} & \text{if } r \text{ is sens.} \\ \{\forall p \in PathSet_i : r \in CP(R_i, p)\} & \text{otherwise} \end{cases} \quad (5)$$

Figure 15 shows our example R , S and T relations with their CPT column. For instance, the join path $R \bowtie S$ will be appended to the CPT column of all the tuples in S_{ns}^R . Additionally, the CPT column of all tuples in R_s will be set to *sens*.

| R | | | S | | | T | | |
|--------|--------|-------------------------|-------|--------|---------------|---------|--------|---|
| Fruit | Region | CPT | Name | Region | CPT | Country | Region | CPT |
| apple | 1 | $S \bowtie R$ | Matt | 1 | sens | U.S | 1 | $S \bowtie T,$ $R \bowtie S \bowtie T$ |
| grape | 2 | $T \bowtie S \bowtie R$ | James | 2 | $T \bowtie S$ | Japan | 2 | sens |
| orange | 1 | sens | Chris | 1 | $R \bowtie S$ | France | 3 | null |

Figure 15: Example relations with the CPT columns.

Experimental Analysis

To study the impact of table partitioning discussed in the previous section, we differentiate between two realizations of our strategy: in our first technique, entitled (*CPT-C*), every record in a table at the private cloud contains a CPT column and they are physically stored together; whereas in our second approach, entitled *CPT-P*, the tables are partitioned based on their record’s CPT column and each partition is stored separately. Each partition file then appended to the corresponding Hive table as a separate partition, so at querying stage, Hive filters out the unnecessary partitions for that particular query.

Sensitive data ratio. For these experiments, we varied the amount of sensitive records (1, 5, 10, 25, 50%) in *customer* and *supplier* tables. Also, we set the number of public machines to 36. As expected, Figure 16 shows that a larger percentage of sensitive data within the input leads to a longer workload execution time for both, CPT-C and CPT-P in Hadoop and Spark. The reason behind this is that a higher sensitive data ratio results in more computations being performed on the private side and implies a longer query execution time in *split-strategy*. When the sensitivity ratio increases, CPT-P’s scan cost increases dramatically. Since the scan cost of queries is the dominant factor compared to other operators (join, filtering etc.) in Spark, CPT-C provides a very low-performance gain compared to All-Private in Spark. Because the scan cost of these two approaches is same. Overall, when sensitivity ratio is as low as 1%, CPT-P provides $8.7\times$ speed-up in Hadoop and $5\times$ speed-up in Spark compared to All-Private.



Figure 16: Running times for different sensitivity ratios.



Figure 17: The CPT column’s creation for different sensitivity ratios.

Recall that we created the CPT column using a Spark job for CPT-C solution. We then physically partitioned tables for CPT-P solution. Figure 17 shows how much time we spent in preparing private cloud data for both CPT-C and CPT-P. It also indicates the gains of

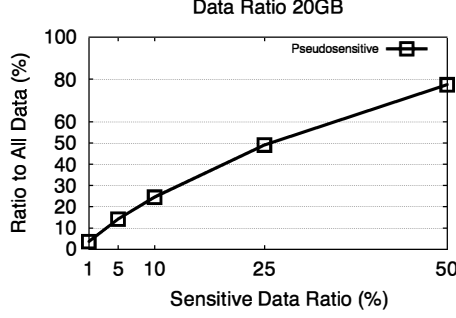


Figure 18: Comparison of pseudo-sensitive data and sensitivity ratio.

these approaches compared to All-Private in terms of the overall workload execution time. As indicated in Figure 17, until 25% sensitivity, CPT-P’s data preparation time is less than that of performance gain in Hadoop; whereas in Spark, data preparation time is always higher than the performance gain for both CPT-P and CPT-C. Note that, we prepare the CPT column only once on a static data for an expected workload that will more likely be executed more than once with different selection and projection conditions. In Spark, if the sensitivity ratio is as high as 10%, executing the workload more than once will be enough for the performance gain of CPT-P solution to be higher than the overhead of data preparation time.

Size of Private Storage. Besides storing sensitive data, in our technique, we also store pseudo-sensitive data on the private cloud. This enables us to execute queries in a partitioned manner while minimizing expensive inter-cloud communication during query execution. In Figure 18, we plot the size of pseudo-sensitive data as a percentage of total database size at different sensitivity levels. We note that even when sensitivity levels are as high as 5-10%, the pseudo-sensitive data remains only a fraction (15-25% of the total data). At smaller sensitivity levels, the ratio is much smaller.

3.2.2.2 Partitioned Computations at the Public Cloud via Panda

In this section, we define the partitioned computation, illustrate how such a computation can leak information due to the joint processing of sensitive and non-sensitive data, discuss the corresponding security definition, and finally discuss system and adversarial models under which we will develop our solutions.

Partitioned Computations

Let R be a relation that is partitioned into two sub-relations, $R_e \supseteq R_s$ and $R_p \subseteq R_{ns}$, such that $R = R_e \cup R_p$. The relation R_e contains all the sensitive tuples (denoted by R_s) of the relation R and will be stored in encrypted form in the cloud. Note that R_e may contain additional (non-sensitive) tuples of R , if that helps with secure data processing). The relation R_p refer to the sub-relation of R that will be stored in plain-text on the cloud.

Naturally, R_p does not contain any sensitive tuples. For the remainder of the report, we will assume that $R_e = R_s$ and $R_p = R_{ns}$, though our approach will be generalized to allow for a potentially replicated representation of non-sensitive data in encrypted form, if it helps to evaluate queries more efficiently. Let us consider a query Q over relation R . A partition computation strategy splits the execution of Q into two independent sub-queries: Q_s : a query to be executed on $E(R_e)$ and Q_{ns} : a query to be executed on R_p . The final results are computed (using a query Q_{merge}) by appropriately merging the results of the two sub-queries at the trusted database (DB) owner side (or in the cloud, if a trusted component, *e.g.*, Intel SGX, is available for such a merge operation). In particular, the query Q on a relation R is partitioned, as follows:

$$Q(R) = Q_{merge}(Q_s(R_e), Q_{ns}(R_p)) \quad (6)$$

Let us illustrate partitioned computations through an example.

| | EId | FirstName | LastName | SSN | Office# | Department |
|-------|------|-----------|----------|-----|---------|------------|
| t_1 | E101 | Adam | Smith | 111 | 1 | Defense |
| t_2 | E259 | John | Williams | 222 | 2 | Design |
| t_3 | E199 | Eve | Smith | 333 | 2 | Design |
| t_4 | E259 | John | Williams | 222 | 6 | Defense |
| t_5 | E152 | Clark | Cook | 444 | 1 | Defense |
| t_6 | E254 | David | Watts | 555 | 4 | Design |
| t_7 | E159 | Lisa | Ross | 666 | 2 | Defense |
| t_8 | E152 | Clark | Cook | 444 | 3 | Design |

Figure 19: A relation: *Employee*.

Example 1: Consider an *Employee* relation, see Figure 19. In this relation, the attribute *SSN* is sensitive, and furthermore, all tuples of employees for the *Department* = “Defense” are sensitive. In such a case, the *Employee* relation may be stored as the following three relations: (i) *Employee1* with attributes *EId* and *SSN* (see Figure 7a); (ii) *Employee2* with attributes *EId*, *FirstName*, *LastName*, *Office#*, and *Department*, where *Department* = “Defense” (see Figure 7b); and (iii) *Employee3* with attributes *EId*, *FirstName*, *LastName*, *Office#*, and *Department*, where *Department* \neq “Defense” (see Figure 22). Since the relations *Employee1* and *Employee2* (Figures 7a and 7b) contain only sensitive data, these two relations are encrypted before outsourcing, while *Employee3* (Figure 22), which contains only non-sensitive data, is outsourced in clear-text. We assume that the sensitive data is strongly encrypted such that the property of *ciphertext indistinguishability* (*i.e.*, an adversary cannot distinguish pairs of ciphertexts) is achieved. Thus, the two occurrences of E152 have two different ciphertexts.

Consider a query Q : `SELECT FirstName, LastName, Office#, Department from Employee where FirstName = 'John'`. In partitioned computation, the query Q is partitioned into two sub-queries: Q_s that executes on *Employee2*, and Q_{ns} that executes on *Employee3*. Q_s will retrieve the tuple t_4 while Q_{ns} will retrieve the tuple t_2 . Q_{merge} in this

| | EId | SSN |
|-------|------|-----|
| t_1 | E101 | 111 |
| t_2 | E259 | 222 |
| t_3 | E199 | 333 |
| t_5 | E152 | 444 |
| t_6 | E254 | 555 |
| t_7 | E159 | 666 |

Figure 20: A sensitive relation: *Employee1*.

| | EId | FirstName | LastName | Office# | Department |
|-------|------|-----------|----------|---------|------------|
| t_1 | E101 | Adam | Smith | 1 | Defense |
| t_4 | E259 | John | Williams | 6 | Defense |
| t_5 | E152 | Clark | Cook | 1 | Defense |
| t_7 | E159 | Lisa | Ross | 2 | Defense |

Figure 21: A sensitive relation: *Employee2*.

| | EId | FirstName | LastName | Office# | Department |
|-------|------|-----------|----------|---------|------------|
| t_2 | E259 | John | Williams | 2 | Design |
| t_3 | E199 | Eve | Smith | 2 | Design |
| t_6 | E254 | David | Watts | 4 | Design |
| t_8 | E152 | Clark | Cook | 3 | Design |

Figure 22: A non-sensitive relation: *Employee3*.

example is simply a union operator. Note that the execution of the query Q will also retrieve the same tuples.

Inference Attack in Partitioned Computations

Partitioned computations, if performed naively, could lead to inferences about sensitive data from non-sensitive data. To see this, consider following three queries on the *Employee2* and *Employee3* relations: (i) retrieve tuples of the employee $Eid = E259$, (ii) retrieve tuples of the employee $Eid = E101$, and (iii) retrieve tuples of the employee $Eid = E199$. We consider an *honest-but-curious* adversarial cloud that returns the correct answers to the queries but wishes to know information about the encrypted sensitive tables, *Employee1* and *Employee2*.

Table 1 shows the adversary’s view based on executing the corresponding Q_s and Q_{ns} components of the above three queries assuming that the tuple retrieving cryptographic approaches are not hiding access-patterns. During the execution, the adversary gains complete knowledge of non-sensitive tuples returned, and furthermore, knowledge about which encrypted tuples are returned as a result of Q_s ($E(t_i)$ in the table refers to the encrypted tuple t_i).

Given the above adversarial view, the adversary learns that employee E259 has tuples in both D_s ($= D_e$) and D_p ($= D_{ns}$). Coupled with the knowledge about data partitioning,

Table 1: Queries and returned tuples/adversarial view.

| Query value | Returned tuples/Adversarial view | |
|-------------|----------------------------------|-----------|
| | Employee2 | Employee3 |
| E259 | $E(t_4)$ | t_2 |
| E101 | $E(t_1)$ | null |
| E199 | null | t_3 |

the adversary can learn that E259 works in both sensitive and non-sensitive departments. Moreover, the adversary learns which sensitive tuple has an *Eid* equals to E259. From the 2nd query, the adversary learns that E101 works only in a sensitive department, (since the query did not return any answer from the Employee3 relation). Likewise, from the 3rd query, the adversary learns that E199 works only in a non-sensitive department.

In order to prevent such an attack, we need a new security definition. Before we discuss the formal definition of partitioned data security, we first provide intuition for the definition. Observe that before retrieving any tuple, under the assumption that no one except the DB owner can decrypt an encrypted sensitive value, say $E(s_i)$, the adversary cannot learn which non-sensitive value is identical to clear-text value of $E(s_i)$; let us denote s_i as clear-text of $E(s_i)$. Thus, the adversary will consider that the value s_i is identical to one of the non-sensitive values. Based on this fact, the adversary can create a complete bipartite graph having $|S|$ nodes on one side and $|NS|$ nodes on the other side, where $|S|$ and $|NS|$ are a number of sensitive and non-sensitive values, respectively. The edges in the graph are called *surviving matches of the values*. For example, before executing any query, the adversary can create a bipartite graph for 4 sensitive and 4 non-sensitive values of EID attribute of Example 1; as shown in Figure 23.

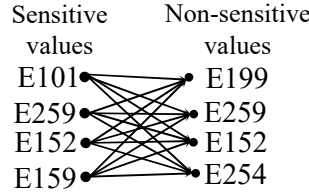


Figure 23: A bipartite graph showing an initial condition sensitive and non-sensitive values before query execution.

The query execution on the datasets creates an adversarial view that guides the adversary to create a (new) bipartite graph of the same number of nodes on both sides. The requirement is to preserve all the edges of the initial bipartite graph in the graph obtained after the query execution, leading to the initial condition that the clear-text of the value $E(s_i)$ is identical to one of the non-sensitive values. Note that if the query execution removes any surviving matches of the values, it will leak that the value s_i is not identical to those non-sensitive values.

We also need to hide occurrences of a sensitive value. Before a query execution, due to ciphertext indistinguishability, all occurrences of a single sensitive value are different, but a

simple search or join query may reveal how many tuples have the same value. Based on the above two requirements, we can define a notion of *partitioned data security*.

Partitioned Data Security at the Public Cloud

Let R be a relation containing sensitive and non-sensitive tuples. Let R_s and R_{ns} be the sensitive and non-sensitive relations, respectively. Let $q(R_s, R_{ns})[A]$ be a query, q , over an attribute A of the R_s and R_{ns} relations. Let X be the auxiliary information about the sensitive data, and Pr_{Adv} be the probability of the adversary knowing any information. A query execution mechanism ensures the partitioned data security if the following two properties hold:

- $Pr_{Adv}[e_i \stackrel{a}{=} ns_j | X] = Pr_{Adv}[e_i \stackrel{a}{=} ns_j | X, q(R_s, R_{ns})[A]]$, where $e_i = E(t_i)[A]$ is the encrypted representation for the attribute value A for any tuple t_i of the relation R_s and ns_j is a value for the attribute A for any tuple of the relation R_{ns} . The notation $\stackrel{a}{=}$ shows a sensitive value is identical to a non-sensitive value. This equation captures the fact that an initial probability of linking a sensitive tuple with a non-sensitive tuple will be identical after executing several queries on the relations.
- $Pr_{Adv}[v_i \stackrel{r}{\sim} v_j | X] = Pr_{Adv}[v_i \stackrel{r}{\sim} v_j | X, q(R_s, R_{ns})[A]]$, for all $v_i, v_j \in Domain(A)$. The notation $\stackrel{r}{\sim}$ shows a relationship between counts of the number of tuples with sensitive values. This equation states that the probability of adversary gaining information about the relative frequency of sensitive values does not increase by the query execution.

The definition above formalizes the security requirement of any partitioned computation approach. Of course, a partitioned approach, besides being secure, must also be correct in that it returns the same answer as that returned by the original query Q if it were to execute without regard to security.

Query Binning: A Technique for Partitioned Computations using a Cryptographic Technique at the Public Cloud

In this section, we will study query binning (QB) as a partitioned computing approach. QB is related to bucketization, which is studied in past [13]. While bucketization was carried over the data in [13], QB performs bucketization on queries. In general, one may ask more queries than original query while adding overhead but it prevents the above-mentioned inference attack. We study QB under some assumption and setting, given below.⁵

Problem Setup. We assume the following two entities in our model: (i) *A database (DB) owner*: who splits each relation R in the database having attributes R_s and R_{ns} containing all sensitive and non-sensitive tuples, respectively. (ii) *A public cloud*: The DB owner outsources the relation R_{ns} to a public cloud. The tuples in R_s are encrypted using any

⁵Some of these assumptions are made primarily for ease of the exposition and will be relaxed in [39].

existing mechanism before outsourcing to the same public cloud. However, in the approach, we use non-deterministic encryption, *i.e.*, the cipher representation of two occurrences of an identical value has different representations.

DB Owner Assumptions. In our setting, the DB owner has to store some (limited) metadata such as searchable values and their frequency counts, which will be used for appropriate query formulation. The DB owner is assumed to have sufficient storage for such metadata, and also computational capabilities to perform encryption and decryption. The size of metadata is exponentially smaller than the size of the original data.

Adversarial Model. The adversary (*i.e.*, the untrusted cloud) is assumed to be honest-but-curious, which is a standard setting for security in the public cloud that is *not trustworthy*. An honest-but-curious adversarial public cloud, thus, stores an outsourced dataset without tampering, correctly computes assigned tasks, and returns answers; however, it may exploit side knowledge (*e.g.*, query execution, background knowledge, and the output size) to gain as much information as possible about the sensitive data. Furthermore, the adversary can eavesdrop on the communication channels between the cloud and the DB owner, and that may help in gaining knowledge about sensitive data, queries, or results. The adversary has full access to the following information: (i) all non-sensitive data outsourced in plain-text, and (ii) some *auxiliary* information of the sensitive data. The auxiliary information may contain the metadata of the relation and the number of tuples in the relation. Furthermore, the adversary can observe frequent query types and frequent query terms on the non-sensitive data in case of selection queries. The honest-but-curious adversary, however, cannot launch any attack against the DB owner.

Assumptions for QB. We develop QB initially under the assumption that queries are only on a single attribute, say A . The QB approach takes as inputs: (i) the set of data values (of the attribute A) that are sensitive along with their counts, and (ii) the set of data values (of the attribute A) that are non-sensitive, along with their counts. The QB returns a partition of attribute values that form the query bins for both the sensitive as well as for the non-sensitive parts of the query.

In this report, we also restrict to a case when a value has at most two tuples, where one of them must be sensitive and the other one must be non-sensitive, but both the tuples cannot be sensitive or non-sensitive. The scenario depicted in Example 1 satisfies this assumption. The *Eld* attribute values corresponding to sensitive tuples include $\langle \text{E101}, \text{E259}, \text{E152}, \text{E159} \rangle$ and from the non-sensitive relation values are $\langle \text{E199}, \text{E259}, \text{E152}, \text{E254} \rangle$. Note that all the values occur only one time in one set.

Full version. In this report, we restrict the algorithm for selection query only on one attribute. The full details of the algorithm, extensions of the algorithm for values having a different number of tuples, conjunctive, range, join, insert queries, and dealing with the workload-skew attack is addressed in [39]. Further, the computing cost analysis and efficiency analysis of QB at different or identical-levels of security against a pure cryptographic technique is given in [39].

The Approach. We develop an efficient approach to execute selection queries securely (preventing the information leakage as shown in Example 1) by appropriately partitioning

the query at a public cloud, where sensitive data is cryptographically secure while non-sensitive data stays in clear-text. For answering a selection query, naturally, we use any existing cryptographic technique on sensitive data and a simple search on the clear-text non-sensitive data. Naturally, we can use a secure hardware, *e.g.*, Intel SGX, for all such operations; however, (as mentioned in Figure 13), SGX-based processing takes a significant amount of time, due to limited space of the enclave.

Informally, QB distributes attribute values in a matrix, where rows are sensitive bins, and columns are non-sensitive bins. For example, suppose there are 16 values, say $0, 1, \dots, 15$, and assume all the values have sensitive and associated non-sensitive tuples. Now, the DB owner arranges 16 values in a 4×4 matrix, as follows:

| | NSB_0 | NSB_1 | NSB_2 | NSB_3 |
|--------|---------|---------|---------|---------|
| SB_0 | 11 | 2 | 5 | 14 |
| SB_1 | 10 | 3 | 8 | 7 |
| SB_2 | 0 | 15 | 6 | 4 |
| SB_3 | 13 | 1 | 12 | 9 |

In this example, we have four sensitive bins: $SB_0 \{11, 2, 5, 14\}$, $SB_1 \{10, 3, 8, 7\}$, $SB_2 \{0, 15, 6, 4\}$, $SB_3 \{13, 1, 12, 9\}$, and four non-sensitive bins: $NSB_0 \{11, 10, 0, 13\}$, $NSB_1 \{2, 3, 15, 1\}$, $NSB_2 \{5, 8, 6, 12\}$, $NSB_3 \{14, 7, 4, 9\}$. When a query arrives for a value, say 1, the DB owner searches for the tuples containing values 2, 3, 15, 1 (*viz.* NSB_1) on the non-sensitive data and values in SB_3 (*viz.*, 13, 1, 12, 9) on the sensitive data using the cryptographic mechanism integrated into QB. While the adversary learns that the query corresponds to one of the four values in NSB_1 , s'nce query values in SB_3 are encrypted, the adversary does not learn any sensitive value or a non-sensitive value that is identical to a clear-text sensitive value.

Formally, QB approach riatel maps a selection query for a keyword w , say $q(w)$, to corresponding queries over the non-sensitive relation, say $q(W_{ns})(R_{ns})$, and encrypted relation, say $q(W_s)(R_s)$. The queries $q(W_{ns})(R_{ns})$ and $q(W_s)(R_s)$, each of which represents a set of query values that are executed over the relation R_{ns} in plain-text and, respectively, over the sensitive relation R_s , using the underlying cryptographic method. The sets W_{ns} from R_{ns} and W_s from R_s are selected such that: (i) $w \in q(W_{ns})(R_{ns}) \cap q(W_s)(R_s)$ to ensure that all the tuples containing w are retrieved, and, (ii) the execution of the queries $q(W_{ns})(R_{ns})$ and $q(W_s)(R_s)$ does not reveal any information (and w) to the adversary. The set of $q(W_{ns})(R_{ns})$ is entitled non-sensitive bins, and the set of $q(W_s)(R_s)$ is entitled sensitive bins. Algorithm 1 provides pseudocode of bin-creation method.⁶ Results from the execution of the queries $q(W_{ns})(R_{ns})$ and $q(W_s)(R_s)$ are decrypted, possibly filtered, and merged to generate the final answer.

Based on QB Algorithm 1, for answering the above-mentioned three queries in Example 1, given in Section 3.2.2.2, Algorithm 1 creates two sets or bins on sensitive parts: sensitive bin

⁶The function *approx_sq_factors* in Algorithm 1 two factors x and y of a number n , such that either they are equal or close to each other so that the difference between x and y is less than the difference between any two factors of n (and $x \times y = n$).

Algorithm 1: Bin-creation algorithm, the base case.

Inputs: $|NS|$: the number of values in the non-sensitive data, $|S|$: the number of values in the sensitive data.

Outputs: SB : sensitive bins; NSB : non-sensitive bins

```

1 Function create_bins( $S, NS$ ) begin
2   Permute all sensitive values
3    $x, y \leftarrow \text{approx\_sq\_factors}(|NS|)$ :  $x \geq y$ 
4    $|NSB| \leftarrow x$ ,  $NSB \leftarrow \lceil |NS|/x \rceil$ ,  $SB \leftarrow x$ ,  $|SB| \leftarrow y$ 
5   for  $i \in (1, |S|)$  do  $SB[i \text{ modulo } x][*] \leftarrow S[i]$ ;
6   for  $(i, j) \in (0, SB - 1), (0, |SB| - 1)$  do  $NSB[j][i] \leftarrow \text{allocateNS}(SB[i][j])$  ;
7   for  $i \in (0, NSB - 1)$  do  $NSB[i][*]$  fill the bin if empty with the size limit to  $x$  ;
8   return  $SB$  and  $NSB$ 
end
9 Function allocateNS( $SB[i][j]$ ) begin
  find a non-sensitive value associated with the  $j^{\text{th}}$  sensitive value of the  $i^{\text{th}}$  sensitive bin
end

```

1, denoted by SB_1 , contains {E101, E259}, sensitive bin 2, denoted by SB_2 , contains {E152, E159}, and two sets/bins on non-sensitive parts: non-sensitive bin 1, denoted by NSB_1 , contains {E259, E254}, non-sensitive bin 2, denoted by NSB_2 , contains {E199, E152}.

Table 2: Queries and returned tuples/adversarial view.

| Query value | Returned tuples/ Adversarial view | |
|-------------|-----------------------------------|------------|
| | Employee1 | Employee2 |
| E259 | $E(t_4), E(t_1)$ | t_2, t_6 |
| E101 | $E(t_4), E(t_1)$ | t_3, t_8 |
| E199 | $E(t_4), E(t_1)$ | t_3, t_8 |

Algorithm 2 provides a way to retrieve the bins. Thus, by following Algorithm 2, Table 2 shows that the adversary cannot know the query value w or find a value that is shared between the two sets, when answering to the above-mentioned three queries. The reason is that the desired query value, w , is encrypted with other encrypted values of the set W_s , and, furthermore, the query value, w , is obscured in many requested non-sensitive values of the set W_{ns} , which are in clear-text. Consequently, the adversary is unable to find an intersection of the two sets, which is the exact value. Thus, while answering a query, the adversary cannot learn which employee works only in defense, design, or in both.

Correctness. The correctness of QB indicates that the approach maintains an initial probability of *associating* a sensitive tuple with a non-sensitive tuple will be identical after executing several queries on the relations.

We can illustrate the correctness of QB with the help of an example. The objective of the adversary is to deduce a clear-text value corresponding to an encrypted value of either

Algorithm 2: Bin-retrieval algorithm.

Inputs: w : the query value.

Outputs: SB_a and NSB_b : one sensitive bin and one non-sensitive bin to be retrieved for answering w .

Variables: $found \leftarrow \text{false}$

```
1 Function retrieve_bins( $q(w)$ ) begin
2   for  $(i, j) \in (0, SB - 1), (0, |SB| - 1)$  do
3     if  $w = SB_i[j]$  then
4       return  $SB_i$  and  $NSB_j$ ;  $found \leftarrow \text{true}$ ; break
5     end
6   end
7   if  $found \neq \text{true}$  then
8     for  $(i, j) \in (0, NSB - 1), (0, |NSB| - 1)$  do
9       if  $w = NSB_i[j]$  then
10        return  $NSB_i$  and  $SB_j$ ; break
11      end
12    end
13  end
14  Retrieve the desired tuples from the cloud by sending encrypted values of the bin
15   $SB_i$  (or  $SB_j$ ) and clear-text values of the bin  $NSB_j$  (or  $NSB_i$ ) to the cloud
16 end
```

$\{E_{101}, E_{259}\}$ or $\{E_{152}, E_{159}\}$, since we retrieve the set of these two values. Note that before executing a query, the probability of an encrypted value, say E_i , (where E_i may be $E_{101}, E_{259}, E_{152}$, or E_{159}) to have the clear-text value is $1/4$, which QB maintains at the end of a query. Assume that E_1 and E_2 are encrypted representations of E_{101} and E_{259} , respectively. Also, assume that v_1, v_2, v_3, v_4 are showing the clear-text value of $E_{259}, E_{254}, E_{199}$, and E_{152} , respectively.

When the query arrives for $\langle E_1, E_2, v_1, v_2 \rangle$, the adversary gets the fact that the clear-text representation of E_1 and E_2 cannot be v_1 and v_2 or v_3 and v_4 . If this will happen, then there is no way to associate each sensitive bin of the new bipartite graph with each non-sensitive bin. Now, if the adversary considers the clear-text representation of E_1 is v_1 , then the adversary have four possible allocations of the values v_1, v_2, v_3, v_4 to E_1, E_2, E_3, E_4 , such as: $\langle v_1, v_2, v_3, v_4 \rangle, \langle v_1, v_2, v_4, v_3 \rangle, \langle v_1, v_3, v_4, v_2 \rangle, \langle v_1, v_4, v_3, v_2 \rangle$.

Since the adversary is not aware of the exact clear-text value of E_1 , the adversary also considers the clear-text representation of E_1 is v_2, v_3 , or v_4 . This results in 12 more possible allocations of the values v_1, v_2, v_3, v_4 to E_1, E_2, E_3, E_4 . Thus, the retrieval of the four tuples containing one of the following: $\langle E_1, E_2, v_1, v_2 \rangle$, results in 16 possible allocations of the values v_1, v_2, v_3 , and v_4 to E_1, E_2, E_3 , and E_4 , of which only four possible allocations have v_1 as the clear-text representation of E_1 . This results in the probability of finding $E_1 = v_1$ is $1/4$.

Note that following this technique, executing queries under for each keyword will not

eliminate any surviving matches of the bipartite graph, and hence, the adversary can find the new bipartite graph identical to a bipartite graph before the query execution. Figure 23 shows an initial bipartite graph before the query execution and Figure 24 shows a bipartite graph after the query execution when creating bins on the values. Note that in Figure 24 each sensitive bin is linked to each non-sensitive bin, that in turns, shows that each sensitive value is linked to each non-sensitive value.

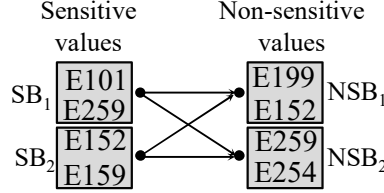


Figure 24: A bipartite graph showing sensitive and non-sensitive bins after query execution, where each sensitive value gets associated with each non-sensitive value.

Effectiveness of QB

From the performance perspective, QB results in saving of encrypted data processing over non-sensitive data – the more the non-sensitive data, the more potential savings. Nonetheless, QB incurs overhead – it converts a single predicate selection query into a set of predicates selection queries over clear-text non-sensitive data, and, a set of encrypted predicates selection queries albeit over a smaller database consisting only of sensitive data. In this section, we compare QB against a pure cryptographic technique and show when using QB is beneficial.

For our model, we will need the following notations: (i) C_{com} : Communication cost of moving one tuple over the network. (ii) C_p (or C_e): Processing cost of a single selection query on plain-text (or encrypted data). In addition, we define three parameters:

α : is the ratio between the sizes of the sensitive data (denoted by S) and the entire dataset (denoted by $S + NS$, where NS is non-sensitive data).

β : is the ratio between the predicate search time on encrypted data using a cryptographic technique and on clear-text data. The parameter β captures the overhead of a cryptographic technique. Note that $\beta = C_e/C_p$.

γ : is the ratio between the processing time of a single selection query on encrypted data and the time to transmit the single tuple over the network from the cloud to the DB owner. Note that $\gamma = C_e/C_{com}$.

Based on the above parameters, we can compute the cost of cryptographic and non-cryptographic selection operations as follows:

$Cost_{plain}(x, D)$ is the sum the processing cost of x selection queries on plain-text data and the communication cost of moving all the tuples having x predicates from the cloud to the DB owner, *i.e.*, $x(\log(D)P_p + \rho DC_{com})$.

$Cost_{crypt}(x, D)$ is the sum the processing cost of x selection queries on encrypted data and the communication cost of moving all the tuples having x predicates from the cloud to the DB owner, i.e., $P_e D + \rho x DC_{com}$, where ρ is the selectivity of the query. Note that cost of evaluating x queries over encrypted data using techniques such as [18, 28, 33], is amortized and can be performed using a single scan of data. Hence, x is not the factor in the cost corresponding to encrypted data processing.

Given the above, we define a parameter η that is the ratio between the computation and communication cost of searching using QB and the computation and communication cost of searching when the entire data (viz. sensitive and non-sensitive data) is fully encrypted using the cryptographic mechanism.

$$\eta = \frac{Cost_{crypt}(|SB|, S)}{Cost_{crypt}(1, D)} + \frac{Cost_{plain}(|NSB|, NS)}{Cost_{crypt}(1, D)} \quad (7)$$

Filling out the values from above, the ratio is:

$$\eta = \frac{C_e S + |SB| \rho DC_{com}}{C_e D + \rho DC_{com}} + \frac{|NSB| \log(D) C_p + |NSB| \rho DC_{com}}{C_e D + \rho DC_{com}} \quad (8)$$

Separating out the communication and processing costs, η becomes:

$$\eta = \frac{S}{D} \frac{C_e}{C_e + \rho C_{com}} + \frac{|NSB| \log(D) C_p}{C_e D + \rho DC_{com}} + \frac{\rho DC_{com} (|NSB| + |SB|)}{C_e D + \rho DC_{com}} \quad (9)$$

Substituting for various terms and cancelling common terms provides:

$$\eta = \alpha \frac{1}{(1 + \frac{\rho}{\gamma})} + \frac{\log(D)}{D} \frac{|NSB|}{\beta (1 + \frac{\rho}{\gamma})} + \frac{\rho}{\gamma} \frac{|NSB| + |SB|}{(1 + \frac{\rho}{\gamma})} \quad (10)$$

Note that ρ/γ is very small, thus the term $(1 + \rho/\gamma)$ can be substituted by 1. Given the above, the equation becomes:

$$\eta = \alpha + \log(D) |NSB| / D \beta + \rho (|NSB| + |SB|) / \gamma \quad (11)$$

Note that the term $\log(D) |NSB| / D \beta$ is very small since $|NSB|$ is the number of distinct values (approx. equal to $\sqrt{|NS|}$) in a non-sensitive bin, while D , which is the size of a database, is a large number, and β value is also very large. Thus, the equation becomes:

$$\eta = \alpha + \rho (|SB| + |NSB|) / \gamma \quad (12)$$

QB is better than a cryptographic approach when $\eta < 1$, i.e., $\alpha + \rho (|SB| + |NSB|) / \gamma < 1$. Thus,

$$\alpha < 1 - \frac{\rho (|SB| + |NSB|)}{\gamma} \quad (13)$$

Note that the values of $|SB|$ and $|NSB|$ are approximately $\sqrt{|NS|}$, we can simplify the above equation to: $\alpha < 1 - 2\rho\sqrt{|NS|}/\gamma$. If we estimate ρ to be roughly $1/|NS|$ (i.e., we assume uniform distribution), the above equation becomes: $\alpha < 1 - 2/\gamma\sqrt{|NS|}$.

The equation above demonstrates that QB trades increased communication costs to reduce the amount of data that needs to be searched in encrypted form. Note that the reduction in encryption cost is proportional to α times the size of the database, while the increase in communication costs is proportional to $\sqrt{|D|}$, where $|D|$ is the number of distinct attribute values. This, coupled with the fact that γ is much higher than 1 for encryption mechanisms that offer strong security guarantees, ensures that QB almost always outperforms the full encryption approaches. For instance, the cryptographic cost for search using secret-sharing is $\approx 10ms$ [33], while the cost of transmitting a single row (≈ 200 bytes for TPC-H Customer table) is $\approx 4 \mu s$ making the value of $\gamma \approx 25000$. Thus, QB, based on the model, should outperform the fully encrypted solution for almost any value of α , under ideal situations where our assumption of uniformity holds. Figure 25 plots a graph of η as a function of γ , for varying sensitivity and $\rho = 10\%$.

To explore the effectiveness of QB under different DB sizes, we tested QB for 3 DB sizes: 150K, 1.5M, and 4.5M tuples. Figure 26 plots η values for the three sizes while varying α . The figure shows that $\eta < 1$, irrespective of the DB sizes, confirming that QB scales to larger DB sizes.

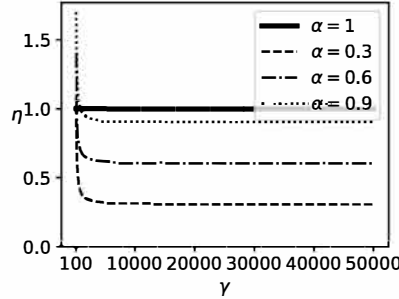


Figure 25: Efficiency graph using equation $\eta = \alpha + \rho(|SB| + |NSB|)/\gamma$.

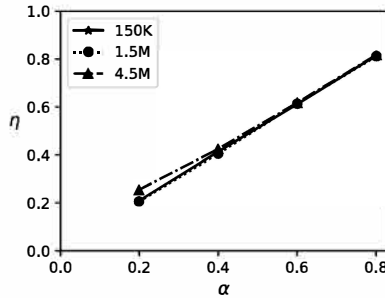


Figure 26: Dataset size.

3.2.2.3 Scaling Secret-Sharing-based Techniques via Obscure

MPC-based databases systems that offer strong security guarantees. Benefits of MPC-based methods in terms of both higher-level security and relatively efficient support for aggregation queries have been extensively discussed in both scientific articles [40, 41, 42, 43] and popular media [44, 45, 46, 47].

Much of the work on MPC-based secure data management requires several servers to collaborate to answer queries. These collaborations require several rounds of communication among non-colluding servers. Instead, we explore secure data management based on SSS that does not require servers to collaborate to generate answers and can, hence, be implemented more efficiently. There is prior work on exploring secret-sharing for SQL processing [48, 33, 49, 50], but the developed techniques suffer from several drawbacks, *e.g.*, weak security guarantees such as leakage of access patterns, significant overhead of maintaining polynomials for generating shares at the database (DB) owner, no support for third-party query execution on the secret-shared outsourced database, etc.

In this section, we provide SSS-based algorithms (entitled OBSCURE) that support a large class of *access-pattern-hiding aggregation queries with selection*. OBSCURE supports count, sum, average, maximum, minimum, top-k, and reverse top-k, queries, without revealing anything about data/query/results to an adversary. OBSCURE also comes with an oblivious result verification algorithm for aggregation queries such that an adversary does not learn anything from the verification. OBSCURE's verification step is not mandatory. A querier may run verification occasionally to confirm the correctness of results. In this report, we will focus only on count and sum queries only.

Building Blocks

OBSCURE is based on SSS, string-matching operations over SSS, and order-preserving secret-sharing (OP-SS). This section provides an overview of these existing techniques.

Shamir's secret-sharing (SSS). In SSS [27], the DB owner divides a secret value, say S , into c different fragments, called *shares*, and sends each share to a set of c non-communicating participants/servers. These servers cannot know the secret S until they collect $c' < c$ shares. In particular, the DB owner randomly selects a polynomial of degree c' with c' random coefficients, *i.e.*, $f(x) = a_0 + a_1x + a_2x^2 + \dots + a_{c'}x^{c'}$, where $f(x) \in \mathbb{F}_p[x]$, p is a prime number, \mathbb{F}_p is a finite field of order p , $a_0 = S$, and $a_i \in \mathbb{N}(1 \leq i \leq c')$. The DB owner distributes the secret S into c shares by placing $x = 1, 2, \dots, c$ into $f(x)$. The secret can be reconstructed based on any $c' + 1$ shares using Lagrange interpolation [51]. Note that $c' \leq c$, where c is often taken to be larger than c' to tolerate malicious adversaries that may modify the value of their shares. For this paper, however, since we are not addressing the availability of data, we will consider c and c' to be identical.

SSS allows an *addition* of shares, *i.e.*, if $s(a)_i$ and $s(b)_i$ are shares of two values a and b , respectively, at the server i , then the server i can compute an addition of a and b itself, *i.e.*, $a + b = s(a) + s(b)$, without knowing real values of a and b .

String-matching operation on secret-shares. Accumulating-Automata (AA) [31] is a new string-matching technique on secret-shares that do not require servers to collaborate to do the operation, unlike MPC-techniques [52, 53, 54, 55, 56, 57]. Here, we explain AA to show how string-matching can be performed on secret-shares.

Let D be the clear-text data. Let $S(D)_i$ ($1 \leq i \leq c$) be the i^{th} secret-share of D stored at the i^{th} server, and c be the number of *non-communicating* servers. AA allows a user to search a pattern, pt , by creating c secret-shares of pt (denoted by $S(pt)_i$, $1 \leq i \leq c$), so that the i^{th} server can search the secret-shared pattern $S(pt)_i$ over $S(D)_i$. The result of the string-matching operation is either 1 of secret-share form, if $S(pt)_i$ matches with a secret-shared string in $S(D)_i$ or 0 of secret-share form; otherwise. Note that when searching a pattern on the servers, AA uses *multiplication* of shares, as well as, the additive property of SSS, which will be clear by the following example. Thus, if the user wants to search a pattern of length l in only *one communication round*, while the DB owner and the user are using a polynomial of degree one, then due to multiplication of shares, the final degree of the polynomial will be $2l$, and solving such a polynomial will require at least $2l + 1$ shares.

Example. Assume that the domain of symbols has only three symbols, namely A, B, and C. Thus, A can be represented as $\langle 1, 0, 0 \rangle$. Similarly, B and C can be represented as $\langle 0, 1, 0 \rangle$ and $\langle 0, 0, 1 \rangle$, respectively.

DB owner side. Suppose that the DB owner wants to outsource B to the (cloud) servers. Hence, the DB owner may represent B as its unary representation: $\langle 0, 1, 0 \rangle$. If the DB owner outsources the vector $\langle 0, 1, 0 \rangle$ to the servers, it will reveal the symbol. Thus, the DB owner uses any three polynomials of an identical degree, as shown in Table 3, to create three shares.

Table 3: Secret-shares of a vector $\langle 0, 1, 0 \rangle$, created by the DB owner.

| Vector values | Polynomials | First shares | Second shares | Third shares |
|---------------|-------------|--------------|---------------|--------------|
| 0 | $0 + 5x$ | 5 | 10 | 15 |
| 1 | $1 + 9x$ | 10 | 19 | 28 |
| 0 | $0 + 2x$ | 2 | 4 | 6 |

User-side. Suppose that the user wants to search for a symbol B. The user will first represent B as a unary vector, $\langle 0, 1, 0 \rangle$, and then, create secret-shares of B, as shown in Table 4. Note that there is no need to ask the DB owner to send any polynomials to create shares or ask the DB owner to execute the search query.

Table 4: Secret-shares of a vector $\langle 0, 1, 0 \rangle$, created by the user/querier.

| Vector values | Polynomials | First shares | Second shares | Third shares |
|---------------|-------------|--------------|---------------|--------------|
| 0 | $0 + x$ | 1 | 2 | 3 |
| 1 | $1 + 2x$ | 3 | 5 | 7 |
| 0 | $0 + 4x$ | 4 | 8 | 12 |

Server-side. Each server performs position-wise multiplication of the vectors that they have, adds all the multiplication resultants, and sends them to the user, as shown in Table 5. An

important point to note here is that the server cannot deduce the keyword, as well as, the data by observing data/query/results.

Table 5: Multiplication of shares and addition of final shares by the servers.

| Computation on | | |
|--------------------|--------------------|---------------------|
| Server 1 | Server 2 | Server 3 |
| $5 \times 1 = 5$ | $10 \times 2 = 20$ | $15 \times 3 = 45$ |
| $10 \times 3 = 30$ | $19 \times 5 = 95$ | $28 \times 7 = 196$ |
| $2 \times 4 = 8$ | $4 \times 8 = 32$ | $6 \times 12 = 72$ |
| 43 | 147 | 313 |

User-side. After receiving the outputs ($\langle y_1 = 43, y_2 = 147, y_3 = 313 \rangle$) from the three servers, the user executes Lagrange interpolation [51] to construct the secret answer, as follows:

$$\frac{(x - x_2)(x - x_3)}{(x_1 - x_2)(x_1 - x_3)} \times y_1 + \frac{(x - x_1)(x - x_3)}{(x_2 - x_1)(x_2 - x_3)} \times y_2 + \frac{(x - x_1)(x - x_2)}{(x_3 - x_1)(x_3 - x_2)} \times y_3 \quad (14)$$

$$= \frac{(x - 2)(x - 3)}{(1 - 2)(1 - 3)} \times 43 + \frac{(x - 1)(x - 3)}{(2 - 1)(2 - 3)} \times 147 + \frac{(x - 1)(x - 2)}{(3 - 1)(3 - 2)} \times 313 = 1 \quad (15)$$

The final answer is 1 that confirms that the secret-shares at the servers have B.

Adversarial Model

We consider two adversarial models, in both of which the cloud servers (storing secret-shares) are not trustworthy. In the *honest but curious* model, the server correctly computes the assigned task without tampering with data or hiding answers. However, the server may exploit side information (*e.g.*, query execution, background knowledge, and output size) to gain as much information as possible about the stored data. Such a model is considered widely in many cryptographic algorithms and is widely used in DaS [58, 13, 59, 60]. We also consider a malicious adversary that could deviate from the algorithm and delete tuples from the relation. Users and database owners, in contrast, are assumed to be not malicious.

Only authenticated users can request query on servers. Further, we follow the restriction of the standard SSS that the adversary cannot collude with all (or possibly the majority of) the servers. Thus, the adversary cannot generate/insert/update shares at the majority of the servers. Also, the adversary cannot eavesdrop on a majority of communication channels between the user and the servers. This can be achieved by either encrypting the traffic between user and servers, or by using anonymous routing [61], in which case the adversary cannot gain knowledge of servers that store the secret-shares. Note that if the adversary could either collude with or successfully eavesdrop on the communication channels between the majority of servers and user, the secret-sharing technique will not apply.⁷ The validity

⁷The DB owner/user can use anonymous routing to send their data to the servers, thereby preventing an adversary from determining which user is connecting to which server. If the adversary knows the majority of the communication channels/servers, then it can construct the secret-shared query, outputs to the query, and the database.

of the assumptions behind secret-sharing has been extensively discussed in prior work [40, 41, 42, 43]. The adversary can be aware of the public information, such as the actual number of tuples and number of attributes in a relation, which will not affect the security of the proposed scheme, though such leakage can be prevented by adding fake tuples and attributes.⁸

Security Properties

In the above-mentioned adversarial model, an adversary wishes to learn the (entire/partial) data and query predicates. Hence, a secure algorithm must prevent an adversary to learn the data (*i*) by just looking the cryptographically-secure data and deduce the frequency of each value (*i.e.*, frequency-count attacks), and (*ii*) when executing a query and deduce which tuples satisfy a query predicate (*i.e.*, access-pattern attacks) and how many tuples satisfy a query predicate (*i.e.*, output-size attacks). Thus, in order to prevent these attacks, our security definitions are identical to the standard security definition as in [62, 63, 64]. An algorithm is *privacy-preserving* if it maintains the privacy of the querier (*i.e.*, query privacy), the privacy of data from the servers, and performs identical operations, regardless of the user query.

Query/Querier’s privacy requires that the user’s query must be hidden from the server, the DB owner, and the communication channel. In addition, the server cannot distinguish between two or more queries of the *same type* based on the output. Queries are of the same type based on their output size. For instance, all count queries are of the same type since they return almost an identical number of bits.

Definition: User’s privacy. *For any probabilistic polynomial time adversarial server having a secret-shared relation $S(R)$ and any two input query predicates, say p_1 and p_2 , the server cannot distinguish p_1 or p_2 based on the executed computations for either p_1 and p_2 .*

Privacy from the server requires that the stored input data, intermediate data during a computation, and output data are not revealed to the server, and the secret value can only be reconstructed by the DB owner or an authorized user. In addition, two or more occurrences of a value in the relation must be different at the server to prevent frequency analysis while data at rest. Recall that due to secret-shared relations (by following the approach given in §3.2.2.3), the server cannot learn the relations and frequency-analysis, and in addition, due to maintaining the query privacy, the server cannot learn the query and the output.

Here, we, also, must ensure that the server’s behavior must be identical for a given query, and the servers provide an identical answer to the same query, regardless of the users (recall that user might be different compared to the data owner in our model). To show that we need to compare the real execution of the algorithm at the servers against the ideal execution of the algorithm at a trusted party having the same data and the same query predicate. An algorithm maintains the data privacy from the server if the real and ideal executions of the algorithm return an identical answer to the user.

⁸The adversary cannot launch any attack against the DB owner. We do not consider cyber-attacks that can exfiltrate data from the DB owner directly, since defending against generic cyber-attacks is outside the scope of this paper.

Definition: Privacy from the server. *For any given secret-shared relation $S(R)$ at a server, any query predicate qp , and any real user, say U , there exists a probabilistic polynomial time (PPT) user U' in the ideal execution, such that the outputs to U and U' for the query predicate qp on the relation $S(R)$ are identical.*

Properties of verification. We provide verification properties against malicious behaviors. A verification method must be oblivious and find any misbehavior of the servers when computing a query. We follow the verification properties from [54], as follows: (i) the verification method cannot be refuted by the majority of the malicious servers, and (ii) the verification method should not leak any additional information.

Obscure Overview

Let us introduce OBSCURE at a high-level. OBSCURE allows single-dimensional and multi-dimensional conjunctive/disjunctive equality queries. Note that the method of OBSCURE for handling these types of queries is different from SQL, since OBSCURE does not support query optimization and indexing⁹ due to secret-shared data. Further, OBSCURE handles range-based queries by converting the range into equality queries. Executing a query on OBSCURE requires four phases, as follows:

PHASE 1: *Data upload by DB owner(s).* The DB owner uploads data to non-communicating servers using a secret-sharing mechanism that allows addition and multiplication (*e.g.*, [31]) at the servers.

PHASE 2: *Query generation by the user.* The user generates a query, creates secret-shares of the query predicate, and sends them to the servers. For generating secret-shares of the query predicate, the user follows the strategies given in §3.2.2.3 (count query), §3.2.2.3 and (sum queries).

PHASE 3: *Query processing by the servers.* The servers process an input query in an oblivious manner such that neither the query nor the results satisfying the query are revealed to the adversary. Finally, the servers transfer their outputs to the user.

PHASE 4: *Result construction by the user.* The user performs Lagrange interpolation on the received results, which provide an answer to the query. The user can also verify these results by following the methods given in §3.2.2.3, §3.2.2.3.

Data Outsourcing

This section provides details on creating and outsourcing a database of secret-shared form. The DB owner wishes to outsource a relation R having attributes A_1, A_2, \dots, A_m and n tuples, and creates the following relations R^1 and R^2 :

- **Relation R^1** that consists of all the attributes A_1, A_2, \dots, A_m along with two additional attributes, namely TID (tuple-id) and Index. The TID attribute will help in finding tuples

⁹For the class of queries considered (viz. aggregation with selection), the main optimization in standard databases is to push selections down and to determine whether an index-scan should be used or not. In secret-sharing, an index scan cannot be used (at least not in any obvious way), since sub-setting the data processed will reveal access-patterns, making the technique less secure. Hence, we avoid using any indexing structure.

having the maximum/minimum/top-k values, and the **Index** attribute will be used to know the tuples satisfying the query predicate. The i^{th} values of the **TID** and **Index** attributes have the *same* and *unique* random number between 1 to n .

- **Relation** R^2 that consists of three attributes **CTID** (clear-text tuple-id), **SSTID** (secret-shared tuple-id), and an attribute, say A_c , on which a comparison operator (minimum, maximum, and top-k) needs to be supported.¹⁰

The i^{th} values of the attributes **CTID** and **SSTID** of the relation R^2 keep the i^{th} value of the **TID** attribute of the relation R^1 . The i^{th} value of the attributes A_c of the relation R^2 keeps the i^{th} value of an attribute of the relation R^1 on which the user wants to execute a comparison operator. Further, the tuples of the relations R^2 are randomly permuted. The reason for doing permutation is that the adversary cannot relate any tuple of both the secret-shared relations, which will be clear soon by the example below.

Note. The relation $S(R^1)$ will be used to answer count and sum queries, while the user uses the two relations $S(R^1)$ and $S(R^2)$ together to fetch a tuple having maximum/minimum/top-k/reverse-top-k value in an attribute.

Table 6: A relation: **Employee**.

| EmpID | Name | Salary | Dept |
|-------|-------|--------|----------|
| E101 | John | 1000 | Testing |
| E101 | John | 100000 | Security |
| E102 | Adam | 5000 | Testing |
| E103 | Eve | 2000 | Design |
| E104 | Alice | 1500 | Design |
| E105 | Mike | 2000 | Design |

Example. Consider the **Employee** relation (see Table 6). The DB owner creates $R^1 = \mathbf{Employee1}$ relation¹¹ (see Table 7a) with **TID** and **Index** attributes. Further, the DB owner creates $R^2 = \mathbf{Employee2}$ relation (see Table 7b) having three attributes **CTID**, **SSTID**, and **Salary**.

Creating secret-shares. Let $A_i[a_j]$ ($1 \leq i \leq m + 1$ and $1 \leq j \leq n$) be the j^{th} value of the attribute A_i . The DB owner creates c secret-shares of each attribute value $A_i[a_j]$ of the relation R^1 using a secret-sharing mechanism that allows string-matching operations at the server. However, c shares of the j^{th} value of the attribute A_{m+2} (*i.e.*, **Index**) are obtained using SSS. This will result in c relations: $S(R^1)_1, S(R^1)_2, \dots, S(R^1)_c$, each having $m + 2$ attributes. The notation $S(R^1)_k$ denotes the k^{th} secret-shared relation of R^1 at the server k . We use the notation $A_i[S(a_j)]_k$ to indicate the j^{th} secret-shared value of the i^{th} attribute of a secret-shared relation at the server k .

¹⁰If there are x attributes on which comparison operators will be executed, then the DB owner will create x relations, each with attributes **CTID**, **SSTID**, and one of the x attributes.

¹¹For verifying results of count and sum queries, we add two more attributes to this relation. However, we do not show here, since verification is not a mandatory step.

Table 7: Two relations obtained from **Employee** relation.

(a) $R^1 = \text{Employee1}$ relation.

| EmpID | Name | Salary | Dept | TID | Index |
|-------|-------|--------|----------|-----|-------|
| E101 | John | 1000 | Testing | 3 | 3 |
| E101 | John | 100000 | Security | 2 | 2 |
| E102 | Adam | 5000 | Testing | 5 | 5 |
| E103 | Eve | 2000 | Design | 4 | 4 |
| E104 | Alice | 1500 | Design | 1 | 1 |
| E105 | Mike | 2000 | Design | 6 | 6 |

(b) $R^2 = \text{Employee2}$ relation.

| CTID | SSTID | Salary |
|------|-------|--------|
| 1 | 1 | 1500 |
| 5 | 5 | 5000 |
| 3 | 3 | 1000 |
| 6 | 6 | 2000 |
| 2 | 2 | 100000 |
| 4 | 4 | 2000 |

Further, on the relation R^2 , the DB owner creates c secret-shares of each value of **SSTID** using a secret-sharing mechanism that allows string-matching operations on the servers and each value of A_c using order-preserving secret-sharing [48, 65, 33]. The secret-shares of the relation R^2 are denoted by $S(R^2)_i$ ($1 \leq i \leq c$). The attribute **CTID** is outsourced in clear-text with the shared relation $S(R^2)_i$. It is important to mention that **CTID** attribute allows fast search due to clear-text representation than **SSTID** attribute, which allows search over shares.

Note that the DB owner's objective is to hide any relationship between the two relations when creating shares of the relations $S(R^1)$ and $S(R^2)$, *i.e.*, the adversary cannot know by just observing any two tuples of the two relations that whether these tuples share a common value in the attribute **TID/SSTID** and A_c or not. Thus, shares of an i^{th} ($1 \leq i \leq n$) value of the attribute **TID** in the relation $S(R^1)_j$ and in the attribute **SSTID** of the relation $S(R^2)_j$ must be different at the j^{th} server. Also, by default, the attribute A_c have different shares in both the relations, due to using different secret-sharing mechanisms for different attributes. The DB owner outsources the relations $S(R^1)_i$ and $S(R^2)_i$ to the i^{th} server.

Count Query and Verification

In this section, we provide techniques to support count queries over secret-shared dataset outsourced by a single or multiple DB owners. The query execution does not involve the DB owner or the querier to answer the query. Further, we develop a method to verify the count query results.

Conjunctive count query. Our conjunctive equality-based count query scans the entire relation only once for checking single/multiple conditions of the query predicate. For example, consider the following conjunctive count query: **select count(*) from R where $A_1 = v_1 \wedge A_2 = v_2 \wedge \dots \wedge A_m = v_m$.**

The user transforms the query predicates to c secret-shares that result in the following query at the j^{th} server: **select count(*) from $S(R^1)_j$ where $A_1 = S(v_1)_j \wedge A_2 = S(v_2)_j \wedge \dots \wedge A_m = S(v_m)_j$.** Note that the single-dimensional query will have only one condition.

Table 8: An execution of the conjunctive count query.

| Name | o_1 | Salary | o_2 | $o_1 \times o_2$ |
|-------|-------|--------|-------|------------------|
| John | 1 | 1000 | 1 | 1 |
| John | 1 | 100000 | 0 | 0 |
| Adam | 0 | 5000 | 0 | 0 |
| Eve | 0 | 2000 | 0 | 0 |
| Alice | 0 | 1500 | 0 | 0 |
| Mike | 0 | 2000 | 0 | 0 |
| | | | | 1 |

Each server j performs the following operations:

$$Output = \sum_{k=1}^{k=n} \prod_{i=1}^{i=m} (A_i[S(a_k)]_j \otimes S(v_i)_j) \quad (16)$$

\otimes shows a string-matching operation that depends on the underlying text representation. For example, if the text is represented as a unary vector, \otimes is a bit-wise multiplication and addition over a vector's elements, whose results will be 0 or 1 of secret-share form. Each server j compares the query predicate value $S(v_i)$ against k^{th} value ($1 \leq k \leq n$) of the attribute A_i , multiplies all the resulting comparison for each of the attributes for the k^{th} tuple. This will result in a single value for the k^{th} tuple, and finally, the server adds all those values. Since secret-sharing *allows the addition of two shares*, the sum of all n resultant shares provides the occurrences of tuples that satisfy the query predicate of secret-share form in the relation $S(R^1)$ at the j^{th} server. On receiving the values from the servers, the user performs Lagrange interpolation [51] to get the final answer in clear-text.

Correctness. The occurrence of k^{th} tuple will only be included when the multiplication of m comparisons results in 1 of secret-share form. Having only a single 0 as a comparison resultant over an attribute of k^{th} tuple produce 0 of secret-share form; thus, the k^{th} tuple will not be included. Thus, the correct occurrences over all tuples are included that satisfy the query's **where** clause.

Example. We explain the above conjunctive count query method using the following query on the **Employee** relation (refer to Table 6): **select count(*) from Employee where Name = 'John' and Salary = '1000'**. Table 8 shows the result of the private string-matching on the attribute **Name**, denoted by o_1 , and on the attribute **Salary**, denoted by o_2 . Finally, the last column shows the result of the query for each row and the final count answer for all the tuples. Note that for the purpose of explanation, we use clear-text values; however, the server will perform all operations over secret-shares. For the first tuple, when the servers check the first value of **Name** attribute against the query predicate **John** and the first value of **Salary** attribute against the query predicate **1000**, the multiplication of both the results of string-matching becomes 1. For the second tuple, when the server checks the second value of **Name** and **Salary** attributes against the query predicate **John** and **1000**, respectively, the multiplication of both the results become 0. All the other tuples are processed in the same way.

Disjunctive count query. Our disjunctive count query also scans the entire relation only once for checking multiple conditions of the query predicate, like the conjunctive count query. Consider, for example, the following disjunctive count query: `select count(*) from R where $A_1 = v_1 \vee A_2 = v_2 \vee \dots \vee A_m = v_m$`

The user transforms the query predicates to c secret-shares that results in the following query at the j^{th} server: `select count(*) from $S(R^1)_j$ where $A_1 = S(v_1)_j \vee \dots \vee A_m = S(v_m)_j$` The server j performs the following operation:

$$Result_i^k = A_i[S(a_k)]_j \otimes S(v_i)_j, 1 \leq i \leq m \quad (17)$$

$$Output = \sum_{k=1}^{k=n} (((Result_1^k \text{ OR } Result_2^k) \text{ OR } Result_3^k) \dots \text{ OR } Result_m^k) \quad (18)$$

To capture the OR operation for each tuple k , the server generates m different results either 0 or 1 of secret-share form, denoted by $Result_i$ ($1 \leq i \leq m$), each of which corresponds to the comparison for one attribute. To compute the final result of the OR operation for each tuple k , one can perform binary-tree style computation. However, for simplicity, we used an iterative OR operation, as follows:

$$temp_1^k = Result_1^k + Result_2^k - Result_1^k \times Result_2^k \quad (19)$$

$$temp_2^k = temp_1^k + Result_3^k - temp_1^k \times Result_3^k \quad (20)$$

\vdots

$$Output^k = temp_{m-1}^k + Result_m^k - temp_{m-1}^k \times Result_m^k \quad (21)$$

After performing the same operation on each tuple, finally, the server adds all the resultant of the OR operation ($\sum_{k=1}^{k=n} Output^k$) and sends to the user. The user performs an interpolation on the received values that is the answer to the disjunctive count query.

Correctness. The disjunctive counting operation counts only those tuples that satisfy one of the query predicates. Thus, by performing OR operation over string-matching resultants for an i^{th} tuple results in 1 of secret-share form, if the tuple satisfied one of the query predicates. Thus, the sum of the OR operation resultant surely provides an answer to the query.

Information leakage discussion. The user sends query predicates of secret-share form, and the string-matching operation is executed on all the values of the desired attribute. Hence, access-patterns are hidden from the adversary, so that the server cannot distinguish any query predicate in the count queries. The output of any count query is of secret-share form and contains an identical number of bits. Thus, based on the output size, the adversary cannot know the exact count, as well as, differentiate two count queries. However, the adversary can know whether the count query is single-dimensional, conjunctive or disjunctive count query.

Verifying Count Query Results

In this section, we describe how results of count query can be verified. Note that we explain the algorithms only for a single-dimensional query predicate. Conjunctive and disjunctive predicates can be handled in the same way.

Here, our objective is to verify that (i) all tuples of the databases are checked against the count query predicates, and (ii) all answers to the query predicate (0 or 1 of secret-share form) are included in the answer. In order to verify both the conditions, the server performs two functions, f_1 and f_2 , as follows:

$$op_1 = f_1(x) = \sum_{i=1}^{i=n} (S(x_i) \otimes o_i) \quad (22)$$

$$op_2 = op_1 + f_2(y) = op_1 + \sum_{i=1}^{i=n} f_2(S(y_i) \otimes (1 - o_i)) \quad (23)$$

i.e., the server executes the functions f_1 and f_2 on n secret-shared values each (of two newly added attributes A_x and A_y , outsourced by the DB owner, described below). In the above equations o_i is the output of the string-matching operation carried on the i^{th} value of an attribute, say A_j , on which the user wants to execute the count query. The server sends the outputs of the function f_1 , denoted by op_1 , and the sum of the outputs of f_1 and f_2 , denoted by op_2 , to the user. The outputs op_1 and op_2 ensure the count result verification and that the server has checked each tuple, respectively. The verification method for a count query works as follows:

The DB owner. For enabling a count query result verification over any attribute, the DB owner adds two attributes, say A_x and A_y , having initialized with one, to the relation R^1 . The values of the attributes A_x and A_y are also outsourced of SSS form (not unary representations) to the servers.

Server. Each server k executes the count query, as mentioned in §3.2.2.3, *i.e.*, it executes the private string-matching operation on the i^{th} ($1 \leq i \leq n$) value of the attribute A_j against the query predicate and adds all the resultant values. In addition, each server k executes the functions f_1 and f_2 . The function f_1 (and f_2) multiplies the i^{th} value of the A_x (and A_y) attribute by the i^{th} string-matching resultant (and by the complement of the i^{th} string-matching resultant). The server k sends the following three things: (i) the sum of the string-matching operation over the attribute A_j , as a result, say $\langle result \rangle_k$, of the count query, (ii) the outputs of the function f_1 : $\langle op_1 \rangle_k$, and (iii) the sum of outputs of the function f_1 and f_2 : $\langle op_2 \rangle_k$, to the user.

User-side. The user interpolates the received three values from each server, which result in $Iresult$, Iop_1 , and Iop_2 . If the server followed the algorithm, the user will obtain: $Iresult = Iop_1$ and $Iop_2 = n$, where n is the number of tuples in the relation, and it is known to the user.

Example. Here, we explain the above method using the following query on the **Employee** relation (refer to Table 6): `select count(*) from Employee where Name = 'John'`. Table 9 shows the result of the private string-matching, functions f_1 and f_2 at a server. Note that for the purpose of explanation, we use clear-text values; however, the server will perform

Table 9: An execution of the count query verification.

| Name | String-matching results | f_1 | f_2 |
|-------|-------------------------|-------|-------|
| John | 1 | 1 | 0 |
| John | 1 | 1 | 0 |
| Adam | 0 | 0 | 1 |
| Eve | 0 | 0 | 1 |
| Alice | 0 | 0 | 1 |
| Mike | 0 | 0 | 1 |
| | 2 | 2 | 4 |

all operations over secret-shares. For the first tuple, when the servers check the first value of **Name** attribute against the query predicate, the result of string-matching becomes 1 that is multiplied by the first value of the attribute A_x , and results in 1. The complement of the resultant is multiplied by the first value of the attribute A_y , and results in 0. All the other tuples are processed in the same way. Note that for this query, $result = op_1 = 2$ and $op_2 = 6$, if server performs each operation correctly.

Correctness. Consider two cases: (i) all servers discard an entire identical tuple for processing, or (ii) all servers correctly process each value of the attribute A_j , op_1 , and op_2 ; however, they do not add an identical resultant, o_i ($1 \leq i \leq n$), of the string-matching operation. In the first case, the user finds $Iresult = Iop_1$ to be true. However, the second condition ($Iop_2 = n$) will never be true, since discarding one tuple will result in $Iop_2 = n - 1$. In the second case, the servers will send the wrong $result$ by discarding an i^{th} count query resultant, and they will also discard the i^{th} value of the attribute A_x to produce $Iresult = Iop_1$ at the user-side. Here, the user, however, finds the second condition $Iop_2 = n$ to be false.

Thus, the above verification method correctly verifies the count query result, always, under the assumption of SSS that an adversary cannot collude all (or the majority of) the servers, as given in §3.2.2.3.

Sum and Average Queries

The sum and average queries are based on the search operation as mentioned above in the case of conjunctive/disjunctive count queries. In this section, we briefly present sum and average queries on a secret-shared database outsourced by single or multiple DB owners. Then, we develop a result verification approach for sum queries.

Conjunctive sum query. Consider the following query: **select sum(A_ℓ) from R where $A_1 = v_1 \wedge A_2 = v_2 \wedge \dots \wedge A_m = v_m$.**

In the secret-sharing setting, the user transforms the above query into the following query at the j^{th} server: **select sum(A_ℓ) from $S(R^1)_j$ where $A_1 = S(v_1)_j \wedge A_2 = S(v_2)_j \wedge \dots \wedge A_m = S(v_m)_j$.** This query will be executed in a similar manner as conjunctive count query except for the difference that the i^{th} resultant of matching the query predicate is multiplied by the i^{th} values of the attribute A_ℓ . The j^{th} server performs the following operation on each

attribute on which the user wants to compute the sum, i.e., A_ℓ and A_q :

$$\sum_{k=1}^{k=n} A_\ell[S(a_k)]_j \times (\prod_{i=1}^{i=m} (A_i[S(a_k)]_j \otimes S(v_i)_j)) \quad (24)$$

Correctness. The correctness of conjunctive sum queries is similar to the argument for the correctness of conjunctive count queries.

Disjunctive sum query. Consider the following query: **select sum(A_ℓ) from R where $A_1 = v_1 \vee A_2 = v_2 \vee \dots \vee A_m = v_m$.** The user transforms the query predicates to c secret-shares that results in the following query at the j^{th} server: **select sum(A_ℓ) from $S(R^1)_j$ where $A_1 = S(v_1)_j \vee A_2 = S(v_2)_j \vee \dots \vee A_m = S(v_m)_j$**

The server j executes the following computation:

$$Result_i^k = A_i[S(a_k)]_j \otimes S(v_i)_j, 1 \leq i \leq m, 1 \leq k \leq n \quad (25)$$

$$Output = \sum_{k=1}^{k=n} A_\ell[S(a_k)]_j \times (((Result_1^k \text{ OR } Result_2^k) \text{ OR } Result_3^k) \dots \text{ OR } Result_n^k) \quad (26)$$

The server multiplies the k^{th} comparison resultant by the k^{th} value of the attribute, on which the user wants to execute the sum operation (e.g., A_ℓ), and then, adds all values of the attribute A_ℓ .

Correctness. The correctness of a disjunctive sum query is similar to the correctness of a disjunctive count query.

Average queries. In our settings, computing the average query is a combination of the counting and the sum queries. The user requests the server to send the count and the sum of the desired values, and the user computes the average at their end.

Information leakage discussion. Sum queries work identically to count queries. Sum queries, like count queries, hide the facts which tuples are included in the sum operation, and the sum of the values.

Result Verification of Sum Queries

Now, we develop a result verification approach for a single-dimensional sum query. The approach can be extended for conjunctive and disjunctive sum queries. Let A_ℓ be an attribute whose values will be included by the following sum query. **select sum(A_ℓ) from R where $A_q = v$.**

Here, our objective is to verify that (i) all tuples of the databases are checked against the sum query predicates, $A_q = v$, and (ii) only all qualified values of the attribute A_ℓ are included as an answer to the sum query. The verification of a sum query first verifies the occurrences of the tuples that qualify the query predicate, using the mechanism for count query verification (§3.2.2.3). Further, the server computes two functions, f_1 and f_2 , to verify both the conditions of sum-query verification in an oblivious manner, as follows:

$$op_1 = f_1(x) = \sum_{i=1}^{i=n} o_i(x_i + a_i + o_i) \quad (27)$$

$$op_2 = f_1(x) = \sum_{i=1}^{i=n} o_i(y_i + a_i + o_i) \quad (28)$$

i.e., the server executes the functions f_1 and f_2 on n values, described below. In the above equations, o_i is the output of the string-matching operation carried on the i^{th} value of the attribute A_q , and a_i be the i^{th} ($1 \leq i \leq n$) value of the attribute A_ℓ . The server sends the sum of the outputs of the function f_1 , denoted by op_1 , and the outputs of f_2 , denoted by op_2 , to the user. Particularly, the verification method for a sum query works as follows:

The DB owner. Analogous with the count verification method, if the data owner wants to provide verification for sum queries, new attributes should be added. Thus, the DB owner adds two attributes, say A_x and A_y , to the relation R^1 . The i^{th} values of the attributes A_x and A_y are any two random numbers whose difference equals to $-a_i$, where a_i is the i^{th} value of the attribute A_ℓ . The values of the attributes A_x and A_y are also secret-shared using SSS. For example, in Table 10, boldface numbers show these random numbers of the attribute A_x and A_y in clear-text.

Servers. The servers execute the above-mentioned sum query, *i.e.*, each server k executes the private string-matching operation on the i^{th} ($1 \leq i \leq n$) value of the attribute A_q against the query predicate v and multiplies the resultant value by the i^{th} value of the attribute A_ℓ . The server k adds all the resultant values of the attributes A_ℓ .

Verification stage. The server k executes the functions f_1 and f_2 on each value x_i and y_i of the attributes A_x and A_y , by following the above-mentioned equations. Finally, the server k sends the following three things to the user: (i) the sum of the resultant values of the attributes A_ℓ , say $\langle sum_\ell \rangle_k$, (ii) the sum of the output of the string-matching operations carried on the attribute A_q , say $\langle sum_q \rangle_k$,¹² against the query predicate, and (iii) the sum of outputs of the functions f_1 and f_2 , say $\langle sum_{f_1 f_2} \rangle_k$.

User-side. The user interpolates the received three values from each server, which results in $Isum_\ell$, $Isum_q$, and $Isum_{f_1 f_2}$. The user checks the value of $Isum_{f_1 f_2} - 2 \times Isum_q$ and $Isum_\ell$, and if it finds equal, then it implies that the server has correctly executed the sum query.

Example. We explain the above method using the following query on the **Employee** relation (refer to Table 6): `select sum(Salary) from Employee where Dept = 'Testing'`. Table 10 shows the result of the private string-matching (o), the values of the attributes A_x and A_y in boldface, and the execution of the functions f_1 and f_2 at a server. Note that for the purpose of explanation, we show the verification operation in clear-text; however, the server will perform all operations over secret-shares.

For the first tuple, when the server checks the first value of **Dept** attribute against the query predicate, the string-matching resultant, o_1 , becomes 1 that is multiplied by the first value of the attribute **Salary**. Also, the server adds the salary of the first tuple to the first values of the attributes A_x and A_y with o_1 . Then, the server multiplies the summation outputs by o_1 .

For the second tuple, the servers perform the same operations, as did on the first tuple; however, the string-matching resultant o_2 becomes 0, which results in the second values of

¹²If users are interested, they can also verify this result using the method given in §3.2.2.3.

Table 10: An execution of the sum query verification.

| Dept | Salary | o values | A_x and f_1 | A_y and f_2 |
|----------|--------|------------|---------------------------------|----------------------------------|
| Testing | 1000 | 1 | $1(\mathbf{200}+1000+1)=1201$ | $1(-\mathbf{1200}+1000+1)=-199$ |
| Security | 100000 | 0 | $0(\mathbf{1000}+100000+0)=0$ | $0(-\mathbf{101000}+100000+0)=0$ |
| Testing | 5000 | 1 | $1(-\mathbf{5900}+5000+1)=-899$ | $1(\mathbf{900}+5000+1)=5901$ |
| Design | 2000 | 0 | $0(\mathbf{2000}+2000+0)=0$ | $0(-\mathbf{4000}+2000+1)=0$ |
| Design | 1500 | 0 | $0(\mathbf{500}+1500+0)=0$ | $0(-\mathbf{2000}+1500+0)=0$ |
| Design | 2000 | 0 | $0(-\mathbf{2100}+2000+0)=0$ | $0(\mathbf{100}+2000+0)=0$ |
| | | 2 | $\sum f_1 = 302$ | $\sum f_2 = 5702$ |

the attributes A_x and A_y to be 0. The servers perform the same operations on the remaining tuples. Finally, the servers send the summation of o_i (*i.e.*, 2), the sum of the salaries of qualified tuples (*i.e.*, 6000), and the sum of outputs of the functions f_1 and f_2 (*i.e.*, 6004), to the user. Note that for this query, $Isum_{f_1f_2} - 2 \times Isum_q = Isum_\ell$, *i.e.*, $6004 - 2 \times 2 = 6000$. *Correctness.* The occurrences of qualified tuples against a query predicates can be verified using the method given in §3.2.2.3. Consider two cases: (i) all servers discard an entire identical tuple for processing, or (ii) all servers correctly process the query predicate, but they discard the i^{th} values of the attributes A_ℓ , A_x , and A_y .

The first case is easy to deal with, since the count query verification will inform the user that an identical tuple is discarded by the server for any processing. In the second case, the user finds $Isum_{f_1f_2} - 2 \times Isum_q \neq Isum_\ell$, since an adversary cannot provide a wrong value of $Isum_q$, which is detected by count query verification. In order to hold the equation $Isum_{f_1f_2} - 2 \times Isum_q = Isum_\ell$, the adversary needs to generate shares such that $Isum_{f_1f_2} - Isum_\ell = 2 \times Isum_q$, but an adversary cannot generate any share, as per the assumption of SSS that an adversary cannot produce a share, since it requires to collude all (or the majority of) the servers, which is impossible due to the assumption of SSS, as mentioned in §3.2.2.3.

3.2.3 Scaling Privacy Techniques

With strong mathematical guarantees, differential privacy has emerged as a de-facto approach to supporting privacy preserving data sharing when sharing is intended to support aggregate level data. Differential privacy is based on a concept that the answer to the query returned by the database must not reveal whether or not any particular record is in the database used to generate the answer. More specifically,

Differential privacy [66]. A randomized algorithm \mathcal{M} satisfies ϵ -differential privacy if for all D and $D' \in N(D)$ and all $\mathcal{O} \subseteq range(\mathcal{M})$:

$$Pr[\mathcal{M}(D) \in \mathcal{O}] \leq e^\epsilon Pr[\mathcal{M}(D') \in \mathcal{O}]. \quad (29)$$

Where the probabilities are taken over coin tosses of \mathcal{M} .

In DP, ϵ plays the role of a privacy knob – with lower ϵ values corresponding to higher levels of privacy. Since the seminal work in [66] that introduced differential privacy, it has

been applied as an approach to supporting privacy in a variety of settings from database query processing, to machine learning over diverse type of data - stored structural/relational data, streaming data, time-series datasets, multidimensional data, etc.

A key challenge in applying differential privacy is that in a large number of practical settings, for differential privacy guarantee to hold (i.e., small epsilon values), the resulting data is too noisy to be useful for applications. Trading privacy for utility of data has thus emerged as an important challenge in data sharing using differential privacy.

Our work in this context explores two novel dimensions: first, similar to the approach of scaling secure data processing, we extend differential privacy to a setting when only part of data may be sensitive while the other is not sensitive. Such is the nature of data in most practical settings. We exploit such partial sensitivity of the data in a systematic manner to enable data owners to share less noisy data while still supporting strong privacy guarantees on the sensitive data. This led to the concept of one-sided differential privacy discussed in the following subsection.

Another extension of differential privacy is to adapt it to a situation suitable for monitoring. This led us to the concept of minimally invasive monitoring that will be discussed subsequently.

3.2.3.1 Scaling Differential Privacy via One Sided Differential Privacy

Public and private organizations capture and store diverse types of information about individuals that are either obligated to publish or could benefit from sharing with others [67]. A key impediment in sharing is the concern of privacy. For instance, information about online purchases, movie preferences [68], or web searches [69] can lead to inferences about sensitive properties like health, religious affiliation, or sexual orientation.

In recent years, the field of privacy preserving data-sharing has flourished and *differential privacy* [70] (DP) has emerged as the predominant privacy standard. Informally, DP states that for a database that contains a single record per user, the participation of an individual changes the output of an analysis by a bounded factor. Most work in DP (with the exceptions of [71, 72]) has been developed under the assumption that every record in the database is *equally* sensitive.

In this work, we depart from the above well trodden path, as we consider data sharing when only part of the data is sensitive, while the remainder is non-sensitive. In real-world scenarios, the sensitive/non-sensitive dichotomy of data is not always apparent, but in certain cases it naturally occurs as a consequence of legislature, personal preferences, and *privacy policies*. For instance, the General Data Protection Regulation [73, 74] (GDPR) imposes strict rules on how information about EU citizens is handled by companies. Individuals must provide an affirmative consent (opt-in) in order to allow the collection and analysis of their data. Additionally, information about minors under 16 years old can not be stored and processed without parental authorization. The opt-in nature of GDPR implies that a fraction of individuals will declare their data non-private and thus create a dichotomy in the dataset.

For example, consider a smart building management system (SBMS) that can track individuals using indoor localization sensors and devices. The data collected from an SBMS might be restricted due to legislature and privacy policies. Privacy policies might be value-based, e.g., non-residents are by default opted-out from data collection, no data collection in locations like restrooms, etc. Additionally, policies can be value-independent but rather be a result of personal preferences, e.g., “*privacy fundamentalists*” [75] can opt-out from the tracking system, at the expense of enjoying only limited services. Either case results in a dichotomy in the sensitivity status of records in the data. More specifically, non-sensitive records can be either the minority or the majority. For instance, consider a university building and a policy that specifies that residents are non-sensitive, then on most days non-sensitive records will be the majority of the data collected. However, on days where an invited speaker gives a talk the sensitive/non-sensitive ratio will be inversed, as the majority of the data collected will be on visitors of the building whose records are sensitive.

In this work, we used policies as the language for specifying sensitivity. Traditionally, query answering in the presence of policies has been the focus of access control literature [76, 77, 78]. The *Truman* and *non-Truman* models offer two general ways for query answering in access control [79], with the Truman model generalizing the parameterized views framework used in Oracle RDBMs [77]. Based on the Truman model, queries are transparently rewritten to ensure users only see what is contained in the authorized view. Thus, the result is correct only with respect to the restricted view. In non-Truman models queries that can not be answered correctly are rejected. However, both approaches can reveal sensitive record values, as we illustrate in the following example.

Example Consider a smart building with a location-based privacy policy where the smoker’s lounge the only sensitive location. Alice can query whether Bob is in location ℓ , for all locations. If Bob is in the smoker’s lounge, then following the Truman model the answer will be empty, while under the non-Truman model the query is rejected. Under either model Alice can correctly infer that Bob’s true location, thus violating the privacy policy.

This example highlights that simple access control mechanisms fail to protect the privacy of sensitive records. Both approaches do not actually protect the *sensitivity status* of records, but rather transparently reveal whether a record is sensitive or not which in turn can reveal the value of the record. In this work our goal is to hide the sensitivity status of all records in a database.

One solution to that is to directly apply DP; this would hide all properties of a record (including its sensitivity). However, DP algorithms do not leverage the fact that part of the data is non-sensitive to lower the error rates of the query answers.

Our work instead introduces *one-sided differential privacy* (OSDP), a novel privacy definition that provides rigorous privacy guarantees on the sensitive portion of the data while ensuring that the non-sensitive data released does not provide adversary with any knowledge about the sensitive data. To define OSDP, let us first formalize the notion of policies.

[Policy Function] A policy function $P: \mathcal{T} \rightarrow \{0, 1\}$ denotes whether a record $r \in \mathcal{T}$ is

sensitive ($P(r) = 0$) or non-sensitive ($P(r) = 1$).

A *policy function* (or simply *policy*) classifies database records as either sensitive or non-sensitive, examples of policy functions include:

- $\lambda r.if(r.Age \leq 17) : 0; else : 1$ encodes the policy that records corresponding to minors are sensitive.
- $\lambda r.if(r.Race = NativeAmerican \vee r.Optin = False) : 0; else : 1$ implies that all records who have either opted out, or are native Americans are sensitive.

We can now formalize the notion of one-sided differential privacy that provides rigorous privacy guarantees for the sensitive records – i.e., $\{r : P(r) = 0\}_{\forall r \in D}$. To this end, we define one-sided neighboring databases under policy P .

[One-sided P -Neighbors] Let D, D' be two databases and P a policy function. D' is a one-sided P -neighbor of D , i.e., $D' \in N_P(D)$, if and only if: $\exists r \in D$ s.t. $r \in \mathcal{T}_s$, $\exists r' \in D'$ s.t. $r' \neq r$, and $D' = D \setminus \{r\} \cup \{r'\}$.

One-sided neighbors under policy P are created by replacing a single sensitive record with any other possible record. This implies that the N_P relation is asymmetric, i.e., $D' \in N_P(D)$ does not imply $D \in N_P(D')$.

[One-sided Differential Privacy] Let \mathcal{M} be a randomized algorithm. Algorithm \mathcal{M} satisfies (P, ϵ) -one-sided differential privacy, if and only if $\forall \mathcal{O} \subseteq range(\mathcal{M})$, and $\forall D, D' \in N_P(D)$:

$$\Pr[\mathcal{M}(D) \in \mathcal{O}] \leq e^\epsilon \Pr[\mathcal{M}(D') \in \mathcal{O}] \quad (30)$$

Where the probabilities are taken over coin tosses of \mathcal{M} .

Similar to DP, data owners can control the privacy levels of OSDP by appropriately selecting the privacy parameter ϵ and the policy P . OSDP provides an indistinguishability property similar to bounded DP, but only for the sensitive records – i.e., OSDP does not protect whether any record is present in the database or not, but rather protects each record's value and whether it is sensitive or not. Note, that OSDP does not bound the information leakage of non-sensitive records.

We develop a variant of the randomized response algorithm, that can truthfully release a sample of the non-sensitive data, while satisfying OSDP. We use this to release and analyze mobility trajectories of users in the context of smart buildings. DP algorithms are unable to analyze these data with low error. In contrast, data released under OSDP supports these analyses with reasonably high accuracy, for policies with varying sensitive/non-sensitive ratios.

We also proposed a new general recipe for adapting DP algorithms for answering histogram queries to OSDP. Our approach exploits the presence of non-sensitive records and improves state-of-the-art DP algorithms by up to $6\times$ for certain inputs.

3.2.3.2 Minimally Invasive Monitoring

Traditionally, privacy-preserving data sharing has been designed for situations where data is collected and shared in aggregate form, either as a synthetic dataset generated based on the original data, or in the form of query answers. Examples of such situations include privacy-preserving sharing of demographic data (e.g., US Census), medical data (e.g., cancer registries), or click-stream data for vulnerability analysis (e.g., from browsers). While differential privacy (DP) is suited for static sharing situations as above, its usefulness in the context of decision support (DS) applications such as monitoring, event/anomaly detection is limited.

DP-based approaches focus on providing formal privacy guarantees (in the form of a privacy parameter) but do not provide guarantees on the quality of data outputted. DS tasks, on the other hand, require guarantees on the quality of the output, specially, for false negatives, which may prevent, for instance, timely actions/interventions. The DS context poses an interesting dichotomy for privacy technologies: if we release data so as to have strong confidence in our decision, we possibly have to output data quite accurately violating privacy. In contrast, if we output data with strong privacy guarantees, we might not be able to support decision support tasks with much confidence.

Our work explored a radically new concept of minimally invasive monitoring (MIM) that attempts to resolve this paradox. The developed MIM approach, instead of optimizing for utility, while implementing strong privacy guarantees (as is done traditionally), changes the objective to achieve a (probabilistic) bound on utility while optimizing (maximizing) privacy. The utility constraint, itself, is set in a conservative manner such that the decision support task results in a limited level of false negatives.

Decision support tasks such as classification or queries can often be implemented in a manner such that false negatives can be arbitrarily decreased at the cost of increasing false positives. As a simple example, consider a binary classifier over a single variable X corresponding to a threshold query ($X > c$) used to detect anomalously large values of X . One can reduce false negatives by simply reducing the threshold c at the cost of increased false positives. Such a strategy of using generalization/specialization to control the tradeoff between false negatives and positives is broadly applicable in classifiers. The MIM infrastructure exploits such an observation to support guaranteed utility while maximizing privacy. Specifically, in MIM, the classifier conditions are weakened (i.e., generalized) to limit the increase in the false negatives due to noise added to implement differential privacy, such a strategy comes with the increase in false positives, which in turn leads to a more invasive exploration of data to differentiate between true/false positives.

Our goal, in MIM, was to build a new framework along the above direction of minimally invasive exploration that strikes an optimal continuum between the needs of the decision support applications and privacy. A MIM framework can, thus, be viewed as a progressively invasive system that explores data in the context of a monitoring task through a coarse filter with a high level of privacy, but explores the data using a finer filter more invasively only if it passes through the coarse filter. Such a MIM infrastructure is reminiscent to a

degree of the way modern law enforcement has evolved over time wherein one uses a series of (explicit/ implicit) filters to determine whether to explore an incident/suspect progressively more carefully (in MIM context more invasively, i.e., with less privacy) based on evidence collected during the prior steps of the investigation (in MIM context, if object classifies as a positive in the previous tests). We can, thus, view MIM as a software system counterpart of the age-old practice of law enforcement that has evolved over time.

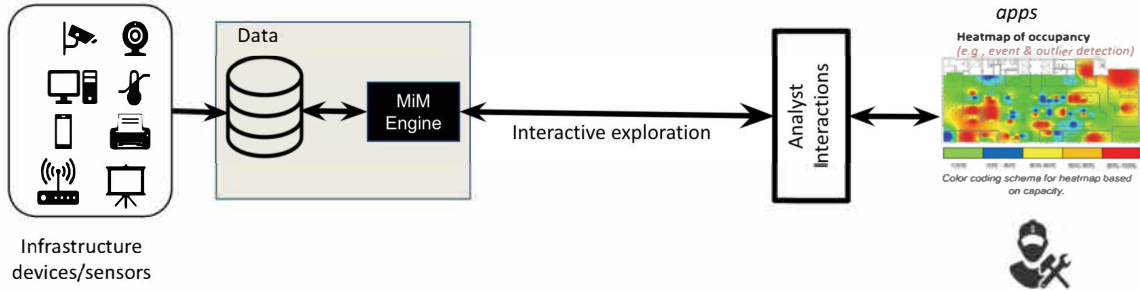


Figure 27: Minimally Invasive Monitoring Architecture

The architecture for minimally invasive monitoring is depicted in figure 27 where analyst interacts with the data through MiM Engine and asks queries over the data set initially with a high level of privacy and interactively decreases the level of privacy based on the application’s accuracy requirements.

The realization of the MIM approach is not straightforward. We need to revisit the very definition of privacy since unlike mechanisms for DP that are characterized by a single privacy parameter, in MIM, different data may have associated different privacy parameters based on the level of privacy we could afford while meeting utility guarantees. For instance, in a progressive MIM implementation, data that was classified as a positive in the previous (weaker) test will be investigated more invasively at lower privacy. We therefore need a generalized measure of privacy that accounts not just for the loss of privacy of an individual (as is often the case when dealing with a differentially private system that output data with privacy guarantees), but that of a group of individuals. We devised a new privacy definition Predicate-wise Differential Privacy where data associated with a particular predicate λ_i can be queried with a different privacy budget epsilon ϵ_i . This concept is inspired by similar ideas as personalized differential privacy where it is possible for individuals to have personalized measure of privacy loss i.e. personalized ϵ . In scenario, where different individuals have different ϵ s, there is no unified privacy loss metric that can be used to compare privacy loss across different scenarios. Consider the following scenarios where different individuals (p_1, p_2, p_3) have different privacy loss $(\epsilon_{p_1}, \epsilon_{p_2}, \epsilon_{p_3})$:

Scenario 1: $\epsilon_{p_1} = 0.1, \epsilon_{p_2} = 0.5, \epsilon_{p_3} = 1$

Scenario 2: $\epsilon_{p_1} = 0.2, \epsilon_{p_2} = 0.4, \epsilon_{p_3} = 1$

In the above scenarios, it is not obvious that which scenario leads to a lower overall privacy loss. Hence, there is a need for unified metric that can capture different levels of privacy loss for different individuals. This metric should capture the privacy loss in terms of adversary’s posterior probability of learning the secret. We devise a new metric to measure the overall

privacy loss for Predicate-wise Differential Privacy. It is the lower bound on entropy (*i.e.*, uncertainty). The intuition is that, higher uncertainty leads to a lower posterior probability in adversary's guess.

Our goal in MiM is to minimize this entropy based privacy loss metric while satisfying accuracy requirement which is specified in terms of bounds on the probability of false negatives for a classification test (β). We first translate the accuracy requirement into privacy metric epsilon (ϵ_{max}) which is required to satisfy the accuracy requirement. We then start with a much smaller epsilon and use a multi iteration approach to utilize epsilon upto the max epsilon, increasing epsilon in the next iteration. In each iteration, since we use a smaller epsilon $\epsilon_i < \epsilon_{max}$ (higher noise), a more generalized test ($X > c - \alpha_i$) for the original test ($X > c$) is used to insure false negative are still bounded at the cost of some false positive. We eliminate all variables X which do not pass the generalized test at a smaller cost (ϵ_i) in each step. This multi step approach results in eliminating variable X that do not satisfy the classification test at a smaller cost. When we classify variables using a weaker test in each iteration ($X > c - \alpha_i$) based noisy values of variable, it results in probabilistic guarantee β on false negative. It additionally provides the same probabilistic guarantee β one false positive when $X < c - 2\alpha_i$. We call $[c - 2\alpha_i, c]$ the region of uncertainty where we do not have a guaranty of false positives. This region keeps getting smaller in subsequent iterations.

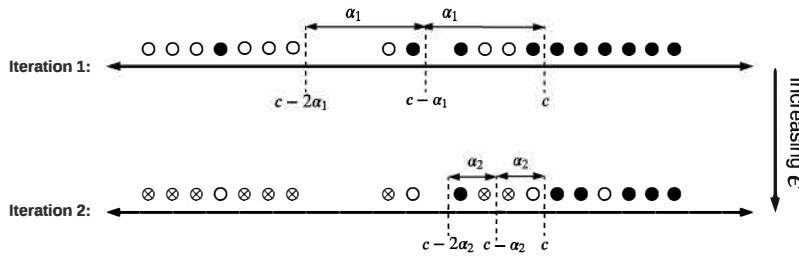


Figure 28: Minimally Invasive Monitoring Architecture

The figure 28 illustrates our multi iteration approach that results in smaller ϵ for variables that get eliminated at an earlier iteration. The figure shows two iterations, where all circles correspond to value of variables that we want to classify for the test ($X > c$). The hollow circles represent variables classified as negatives in an iteration, filled circles represent variables classified as positives and crossed circles represent eliminated variables classified as negatives in previous iterations and do not need to be further explored with a higher ϵ in subsequent iteration.

This multi iteration algorithm results in different privacy loss depending on the number of iteration and ϵ_i chosen for each variable and the data distribution. The privacy loss is quantified using the entropy based aggregate metric. We develop several approaches data dependent/ data independent that choose number of iterations and parameters for each iteration that minimize the privacy loss.

We further need to study the implication of such an approach on fairness since now different subjects may be monitored at different levels of privacy based on the needs of monitoring. The issue of fairness has a counterpart in the law enforcement in the physical world

— police investigating potential crime often develop stereotypes, and preconceived notions leading to unfair and biased treatment of individuals. In the physical world, technological solutions such as body-worn cameras /dashcams attempt to provide a degree of accountability, in MIM we not only need a proper definition of fairness, but also approaches to implement tamperproof evidence demonstrating the need for more invasive exploration to implement accountability and achieve checks/balance in the system Above all we need to create a prototype system and a concrete context to serve as a proof-of-concept that can help explore the idea of MIM and its potential. We will explore MIM in the context of our ongoing work on smart space system entitled TIPPERS that we have built over the past 5 years at UCI funded through the DARPA BRANDEIS program on creating a testbed for the study of privacy technologies in real-world applications. Today, TIPPER is a campus-wide testbed which uses WiFi connectivity data at the campus level to create location-awareness both inside and outside buildings. TIPPERS supports a variety of privacy enhancing technologies – end-to-end policy-based data processing, differential privacy, secure computation over encrypted data, data randomization, etc. TIPPERS has been used to build a variety of campus level apps ranging from locating individual/friends in campus (based on policy), to analytical application for understanding building usage to understanding social networks within workspaces. More recently, it has been used to implement at the campus-level a variety of applications to help organizations (such as UCI) to alleviate the spread of COVID-19 at their premises. In this context, TIPPERS supports real-time monitoring of building occupancy (using Wifi data), and also of potential exposure tracing (over strongly encrypted domain while ensuring full anonymity of individuals) based on co-occupancy of rooms/internal spaces in the buildings. TIPPERS is now actively used as part of the UCI campus reopening in over 20+ buildings and is being transitioned to other campuses including Ball State University and UC, San Diego. It has been transitioned to the Navy, where it was a recipient of the Naval Information Warfare Systems Command (NAVWAR) Innovation Award in 2021. TIPPERS, especially in the context of a campus-level deployment of diverse monitoring application including monitoring occupancy of regions in buildings at UCI while guaranteeing individual’s privacy, provides a unique opportunity for developing the idea of minimally invasive monitoring technology. In this context, we hope to define notions of privacy in the MIM context, develop and implement algorithmic approaches for MIM, formalize the concept of fairness mentioned above, and generalize the concept to a larger context of classification and possibly SQL query processing.

3.2.4 Supporting Fine-grained policies in IoT Systems

Access control policies are a traditional mechanism used in data management to allow users to specify their privacy preferences with respect to usage of their data. In the case of IoT domain, wherein sensors continuously monitor individuals (e.g., continuous physiological monitoring by wearable devices, location monitoring both inside and outside buildings), data management systems need to provide users with mechanisms for finer control over who can access their data and for what purpose. Supporting such fine grained policies in data management systems raises several research challenges and we focus on addressing two of

the important challenges that arise when building such systems for new scenarios such as the IoT.

The first challenge for data management systems is that of efficiently enforcing privacy policies and preferences from different users without loss of utility for the services that exist in the space. Thus the second challenge is that of *scaling enforcement of access control policies* in data management systems. In IoT settings, the set of policies becomes a dominant factor/bottleneck in the query processing due to their large numbers. This has been highlighted as one of the open challenges for Big Data management systems in recent surveys such as [80].

The second challenge deals with *protecting access controlled data from leakages*. In many scenarios including IoT applications, there might be background knowledge available to an adversary who can utilize that as an inference channel to learn more about protected data. One such common form of background knowledge is dependencies that capture the various type of constraints that exist within the data. In the IoT domain, an example of such constraint is through an enrichment function which transforms the raw data collected from sensors and generates the derived data which is shared with application developers and service providers. However, depending upon the properties of this enrichment function it might be possible to learn about raw data from the disclosed derived data. Such unwanted inferences leads to violations of access control policies even when sensitive data is hidden.

To address the first challenge of scalability, we present Sieve, a layered approach of implementing Fine-Grained Access Control in existing DBMSs, that exploits a variety of their features (e.g., UDFs, index usage hints, query explain) to scale to a large number of policies. Given a query, Sieve exploits its context to filter the policies that need to be checked. It also generates *guarded expressions* that save on evaluation cost by grouping policies and exploit database indices to cut on read cost. Our experimental results demonstrate that existing DBMSs can utilize Sieve to significantly reduce query-time policy evaluation cost. Using Sieve DBMSs can support real-time access control in applications such as emerging smart environments.

To address the second challenge of preventing leakages, we study the leakages of access control protected data in data management systems through two important classes of data dependencies: 1) *Denial Constraints* and 2) *Provenance Based Dependencies*. Denial constraints are a general model of integrity constraints and can express commonly used constraints such as functional dependencies, conditional functional dependencies, and key constraints. We introduce *provenance based dependencies* which can capture the relationships between source data and derived data. Considering these dependencies as background knowledge to an adversary, we formally define how the information about a sensitive data could leak through them and methods to compute the leakage. Furthermore, we describe the rules to decide the non-sensitive data that should not be disclosed to prevent leakages of the access control protected data. We describe an algorithm which utilizes these rules and leakage computation to achieve the required deniability guarantees for all sensitive cells.

3.2.4.1 Scaling Policy Enforcement for IoT Data Management

In modern data management systems, data is dynamically captured from sensors and shared with people via queries based on user-specified access control policies. We describe a motivating use case of a smart campus which shows that data involved in processing a simple analytical query might require checking against hundreds to thousands of access control policies. Enforcing that many access control policies in real-time during query execution is well beyond database systems today. While our example and motivation is derived from the smart space and IoT setting, the need for such query processing with a large number of policies also applies to many other domains. This applicability will only increase as emerging legislation such as GDPR empowers users to control their data.

Existing DBMS support Fine-Grained Access Control (FGAC) mechanisms by performing a query rewrite [81]. This is done by appending policies as predicates to the `—WHERE—` clause of the original query. However, they are limited in the complexity of applications they can support due to the increased cost of query execution when the rewriting includes a large number of policies. Thus, scalable access control-driven query execution presents a novel challenge. We evaluated the existing approach of query rewrite on top of a relational DBMS (MySQL) with two different queries from a IoT Benchmark called SmartBench [82]. The results are shown in Figure 29. The first query (on the left) is a real time query from Smart Bench which retrieves the data belonging to an individual user. We perform this experiment for different users and DBMS has to evaluate between 100 to 350 policies depending on the user. In the second query (on the right), it is an analytical query where we progressively decrease the selectivity of the query and thus increasing the number of policies to be evaluated. In both queries, policy evaluation overhead increases linearly with number of policies.

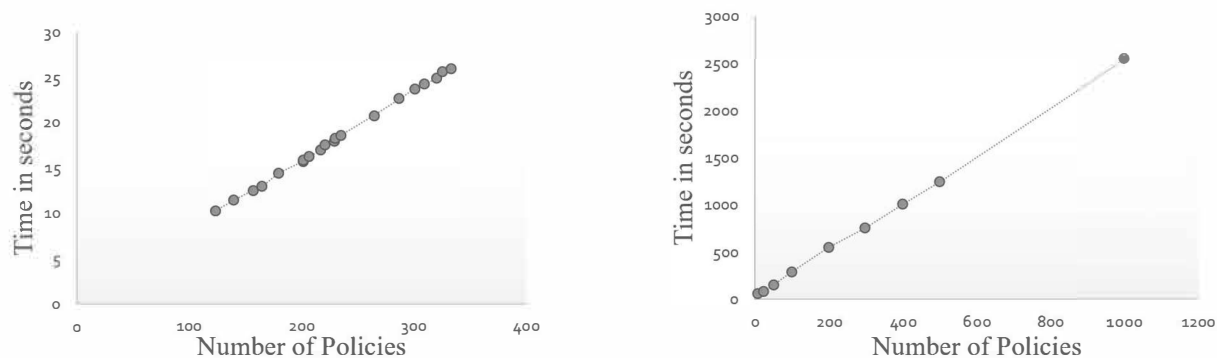


Figure 29: Policy Evaluation overhead vs. Number of Policies.

Sieve incorporates two distinct strategies to reduce overhead: reducing the number of tuples that have to be checked against complex policy expressions and reducing the number of policies that need to be checked against each tuple. First, given a set of policies, it uses them to generate a set of *guarded expressions* that are chosen carefully to exploit the best existing database indexes, thus reducing the number of tuples against which the complete and complex policy expression must be checked. This strategy is inspired by the technique for predicate simplification to exploit indices developed in [83]. Second, Sieve

reduces the overhead of dynamically checking policies during query processing by filtering policies that must be checked for a given tuple by exploiting the context present in the tuple (e.g., user/owner associated with the tuple) and the query metadata (e.g., the person posing the query –i.e., querier– or their purpose). We define a policy evaluation operator Δ for this task and present an implementation as a User Defined Function (UDF).

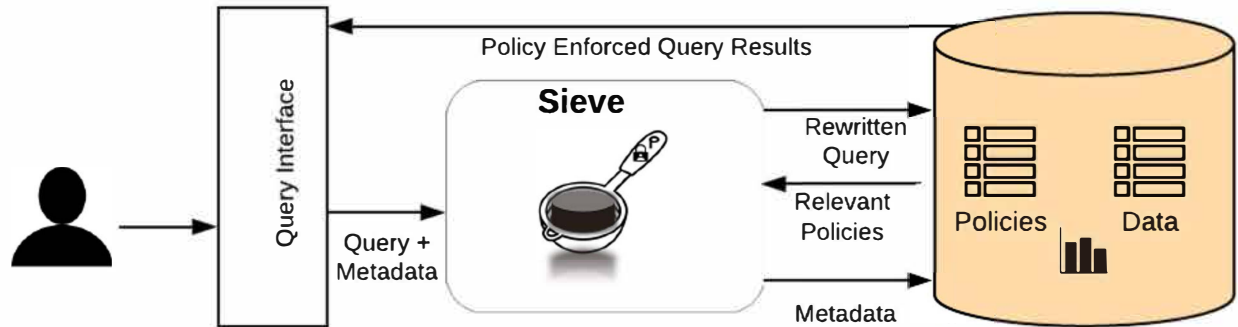


Figure 30: Overview of Sieve.

Sieve combines the above two strategies in a single framework to reduce the overhead of policy checking during query execution. Thus, Sieve adaptively chooses the best strategy possible given the specific query and policies defined for that querier based on a cost model estimation. We evaluate the performance of Sieve using a real WiFi connectivity dataset captured in our building at UC Irvine, including connectivity patterns of over 40K unique devices/individuals. On this real dataset, we generate a synthetic set of policies that such individuals could have defined to control access to their data by others. We also test the performance of our system on a synthetic dataset based on a smart mall where connectivity data of devices are logged inside shops in the mall. Our results highlight the benefit of Sieve-generated query rewrite when compared to the traditional query rewrite approach for access control when processing different queries. Additionally, we perform these experiments on two different DBMSs, MySQL and PostgreSQL, showcasing Sieve’s abilities as a middleware.

Case Study

We consider a motivating application wherein an academic campus supports variety of smart data services such as real-time queue size monitoring in different food courts, occupancy analysis to understand building usage (e.g., room occupancy as a function of time and events, determining how space organization impacts interactions amongst occupants, etc.), or automating class attendance and understanding correlations between attendance and grades [84]. While such solutions present interesting benefits, such as improving student performance [84] and better space utilization, there are privacy challenges [85] in the management of such data. This case study is based on our own experience building a smart campus with variety of applications ranging from real-time services to offline analysis over the past 4 years. The deployed system, entitled TIPPERS [86], is in daily use in several buildings in our UC Irvine campus. TIPPERS at our campus captures connectivity events

(i.e., logs of the connection of devices to WiFi APs) that can be used, among other purposes, to analyze the location of individuals to provide them with services.

We use the UC Irvine campus, with the various entities and relationships as a use case. Consider a professor in the campus posing an analytical query to evaluate the correlation between regular attendance in her class vs. student performance at the end of the semester. Let us assume that within the students in the professor's class, there exist different privacy profiles (as studied in the mobile world by Lin et al. [87]). Adapting the distribution of users by profile to our domain, we assume that 20% of the students might have a common default policy ("unconcerned" group), 18% may want to define their own precise policies ("advance users"), and the rest will depend on the situation (for which we consider, conservatively, 2/3 to be "unconcerned" and 1/3 "advance"). Using this distribution of privacy profiles and applying it to a class of 200 students, we have 120 unconcerned users who will adopt the default policy and 80 advanced users who will define their own set of policies. With the conservative assumption that there are two default policies per default user and at least 4 specific policies per advanced user, we have a total of 560 policies. Typically, advanced users define more policies than this conservative assumption so if we were to add two additional policies per group it will increase the number of policies to 880, or 1.2K (with three additional policies per group).

Given these numbers, together with students taking 1-6 classes per semester, the above query executed over classes a professor taught over the year (faculty teaches 1-4 classes per semester) would involve 3.3K (560 policies/class * 2 classes/quarter * 3 quarters/year) to 7.2K (using 1.2K policies/class estimation) policies. We only focused on a single data type captured in this analysis (i.e., connectivity data) with two conditions per policy (e.g., time and location) and policies defined by a given user at the group-level (and not at the individual-level, which will even further increase the number of policies).

Overview of the Sieve approach

For a given query Q_i , the two main factors that affect the time taken to evaluate the set of policies for the set of tuples \mathcal{T}_{Q_i} required to compute Q_i (i.e., $eval(\mathcal{E}(\mathcal{P}), t_t) \forall t_t \in \mathcal{T}_{Q_i}$) are the large number of complex policies and the number of tuples in \mathcal{T}_{Q_i} . The overhead of policy evaluation can thus be reduced by first eliminating tuples using low cost filters before checking the relevant ones against complex policies and second by minimizing the length of policy expression a tuple t_t needs to be checked against before deciding whether it can be included in the result of Q_i or not. These two fundamental building blocks form the basis for Sieve.

- **Reducing Number of Policies.** Not all policies in \mathcal{P} are relevant to a specific query Q_i . We can first easily filter out those policies that are defined for different queriers/purposes given the query metadata \mathbf{QM}^i . For instance, when Prof. Smith poses a query for grading, only the policies defined for him and the faculty group for grading purpose are relevant out of all policies defined on campus. Thus, given our policy model (that controls access based on querier's identity and purpose), the set of policies relevant to the query can be filtered using \mathbf{QM}^i . We denote the subset of

policies which are relevant given the query metadata QM^i by $\mathcal{P}_{QM^i} \subseteq \mathcal{P}$ where $p_l \in \mathcal{P}_{QM^i}$ iff $QM^i_{purpose} = qc^l_{purpose} \wedge (QM^i_{querier} = qc^l_{querier} \vee qc^l_{querier} \in group(QM^i_{querier}))$. In addition, for a given tuple $t_t \in \mathcal{T}_{Q_i}$ we can further filter policies in \mathcal{P}_{QM^i} that we must check based on the values of attributes in t_t . For instance, the owner of the tuple (i.e., $t_t.owner$) can be used to filter out policies which do not apply to the tuple (i.e., are not part of $\mathcal{P}_{t_t} \subseteq \mathcal{P}_{QM^i}$ where \mathcal{P}_{t_t} is such that $p_l \in \mathcal{P}_{t_t}$ iff $t_t.owner = oc^l_{owner}$ (i.e., the owner of the tuple is the same than the owner/creator of the policy).

- **Reducing Number of Tuples.** Even if the number of policies to check are minimized, the resulting expression $\mathcal{E}(\mathcal{P})$ might still be computationally complex. To speed up processing of $\mathcal{E}(\mathcal{P})$ further, we derive low cost filters (object conditions) from it which can filter out tuples by exploiting existing indexes \mathcal{I} over attributes in the database. We therefore rewrite the policy expression $\mathcal{E}(\mathcal{P}) = OC^1 \vee \dots \vee OC^{|\mathcal{P}|}$ as a *guarded policy expression* $\mathcal{G}(\mathcal{P})$ which is a disjunction of *guarded expressions* $\mathcal{G}(\mathcal{P}) = G_1 \vee \dots \vee G_n$. Each G_i consists of a *guard* oc^i_g and a *policy partition* \mathcal{P}_{G_i} where $\mathcal{P}_{G_i} \subseteq \mathcal{P}$. Note that \mathcal{P}_{G_i} partitions the set of policies, i.e., $\mathcal{P}_{G_i} \cap \mathcal{P}_{G_j} = \emptyset \forall G_i, G_j \in \mathcal{G}(\mathcal{P})$. Also, all policies in \mathcal{P} are covered by one of the guarded expressions, i.e., $\forall p_i \in \mathcal{P} (\exists G_i \in \mathcal{G} \text{ such that } p_i \in \mathcal{P}_{G_i})$. We will represent the guarded expression $G_i = oc^i_g \wedge \mathcal{P}_{G_i}$ where \mathcal{P}_{G_i} is the set of policies but for simplicity of expression we will use it as an expression where there is a disjunction between policies.

The *guard* term oc^i_g is an object condition that can support efficient filtering by exploiting an index. In particular, it satisfies the following properties:

- oc^i_g is a simple predicate over an attribute (e.g., $ts - time > 9am$) and the attribute in oc^i_g has an index defined on it (i.e., $oc^i_g.attr \in \mathcal{I}$).
- The guard oc^i_g is a part of all the policies in the partition and can serve as a filter for them \mathcal{P}_{G_i} (i.e., $\forall p_l \in \mathcal{P}_{G_i} \exists oc^l_j \in OC^l \mid oc^l_j \implies oc^i_g$).

As an example, consider the policy expression of all the policies defined by students to grant the professor access to their data in different situations. Let us consider that many of such policies grant access when the student is connected to the WiFi AP of the classroom. For instance, in addition to John's policy defined before, let us consider that Mary defines the policy — $[W.owner = Mary \wedge W.wifiAP = 1200], [Prof. Smith \wedge Attendance Control], allow$ —. This way, such predicate (i.e., $wifiAP=1200$) could be used as a guard that will group those policies, along with others that share that predicate, to create the following expression: — $wifiAP=1200 \text{ AND } ((owner=John \text{ AND } ts-time \text{ between } 9 \text{ AND } 10am \text{ OR } (owner=Mary) \text{ OR } \dots)$ —.

Sieve adaptively selects a query execution strategy when a query is posed leveraging the above ideas. First, given Q_i , Sieve filters out policies based on QM^i . Then, using the resulting set of policies, it replaces any relation $r_j \in Q_i$ by a projection that satisfies policies in \mathcal{P}_{QM^i} that are defined over r_j . It does so by using the guarded expression $\mathcal{G}(\mathcal{P}_{r_j})$ constructed as a query — $SELECT * FROM r_j \text{ WHERE } \mathcal{G}(\mathcal{P}_{r_j})$ —.

By using $\mathcal{G}(\mathcal{P}_{r_j})$ and its guards oc_g^i , we can efficiently filter out a high number of tuples and only evaluate the relevant tuples against the more complex policy partitions \mathcal{P}_{G_i} . The generation of $\mathcal{G}(\mathcal{P}_{r_j})$ might take place offline if the policy dataset is deemed to undergo small number of changes over time. Otherwise, the generation can be done either when a change is made in the policy table or at query time for more dynamic scenarios (our algorithm is efficient enough for dynamic scenarios).

A tuple that satisfies the guard oc_g^i is then checked against $\mathcal{E}(\mathcal{P}_{G_i}) = \text{OC}^1 \vee \dots \vee \text{OC}^{|\mathcal{P}_{G_i}|}$. This evaluation could be expensive depending upon the number of policies in \mathcal{P}_{G_i} . As it is a DNF expression, in the worst case (a tuple that does not satisfy any policy) will have to be evaluated against each $\text{OC}^j \in \mathcal{P}_{G_i}$. We introduce a policy evaluation operator ($\Delta(G_i, \mathbf{QM}^i, t_t)$) which takes a guarded expression G_i , query metadata \mathbf{QM}^i , and each tuple t_t that satisfied oc_g^i and retrieves a subset of \mathcal{P}_{G_i} (filtered using \mathbf{QM}^i and t_t). Then, policy evaluation on the tuples that satisfy the guard is only performed on this subset of policies instead of \mathcal{P}_{G_i} . Sieve situationally selects based on each $G_i \in G$ whether to use the policy evaluation operator for evaluating \mathcal{P}_{G_i} to minimize the execution cost.

Hence, the main challenges are: 1) Selecting appropriate guards and creating the guarded expression; 2) Dynamically rewriting query by evaluating different strategies and constructing a query that can be executed in an existing DBMS. This generation might take place offline if the policy dataset is deemed to undergo small number of changes over time. Otherwise, the generation can be done either when a change is made in the policy table or at query time for more dynamic scenarios. We later explain how Sieve can be implemented in existing DBMSs and how it selects an appropriate strategy depending on the query and the set of policies that apply to the query.

Experimental evaluation

We used the *TIPPERS* dataset [86] consisting of connectivity logs generated by the 64 WiFi Access Points (APs) at the Computer Science building at UC Irvine for a period of three months. We also used a synthetic dataset containing WiFi connectivity events in a shopping mall for scalability experiments with even larger number of policies. We refer to this dataset as *Mall*. We generated the *Mall* dataset using the IoT data generation tool in [82] to generate synthetic trajectories of people in a space (we used the floorplan of a mall extracted from the Web) and sensor data based on those. The dataset contains 1.7M events from 2,651 different devices representing customers. We used a set of query templates based on the recent IoT SmartBench benchmark [82] which include a mix of analytical and real-time tasks and target queries about (group of) individuals. We also developed a synthetic policy generator which takes into account various real-life privacy requirements to generate policies on the two datasets. We compared Sieve approach against three different baselines. In the first baseline, *Baseline_P*, we append the policies that apply to the querier to the —WHERE— condition of the query. Second, *Baseline_I*, performs an index scan per policy (forced using index usage hints) and combines the results using the —UNION— operator. Third, *Baseline_U* is similar to *Baseline_P* but instead of using the policy expression, it uses a UDF defined on the relation to evaluate the policies.

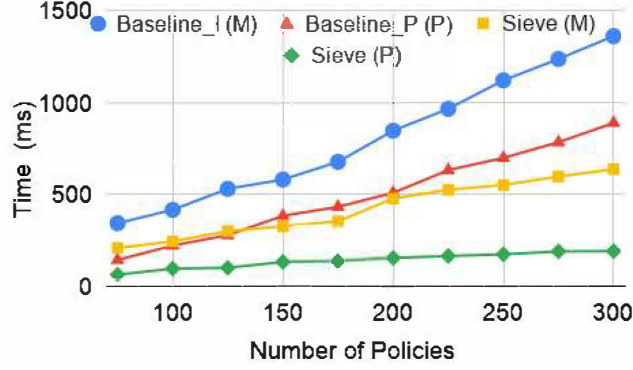


Figure 31: Sieve on MySQL and PostgreSQL.

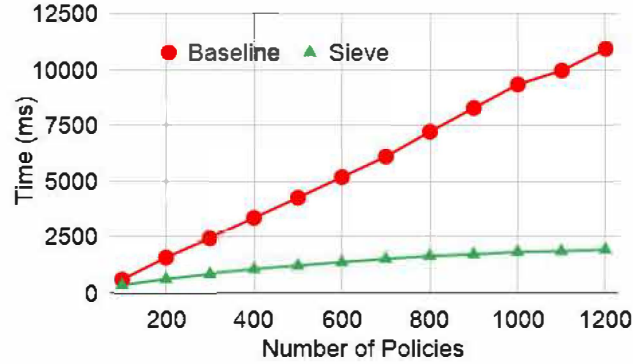


Figure 32: Scalability comparison.

We first study the performance of the guarded expression generation algorithm (Experiment 1). Then, we validate the design choices in Sieve (Experiment 2) and compare the performance of Sieve against the baselines (Experiment 3). The previous experiments are performed on the MySQL system. Next, we study the performance of Sieve on PostgreSQL which, in contrast to MySQL, does not support index usage hints (Experiment 4). In the final experiment, we stress test our approach with a very large number of policies (Experiment 5). The first four experiments use the *TIPPERs* dataset and the final experiment the *Mall* dataset. In this report, we summarize the findings from the last two experiments.

Experiment 1: The four strategies tested in this experiment are: the best performing baseline for MySQL ($Baseline_I(M)$), the baseline in PostgreSQL ($Baseline_P(P)$), and Sieve in both MySQL and PostgreSQL (Sieve (M) and Sieve (P)). The results show that not only Sieve outperforms the baseline in PostgreSQL but also the speedup factor w.r.t. the baseline is even higher than in MySQL. Additionally, the speedup factor in PostgreSQL is the highest at largest number of policies. Based on our analysis of the query plan chosen by PostgreSQL, it correctly chooses the guards for performing index scans (as intended by Sieve) even without the index usage hints. In addition, PostgreSQL supports combining multiple index scans by preparing a bitmap in memory. It used these bitmaps to *OR* the results from the guards whenever it was possible, and the only resultant table rows are visited and obtained from the disk. With a larger number of guards (for larger number of policies), PostgreSQL was

also able to more efficiently filter out tuples compared to using the policies. Thus, Sieve benefits from reduced number of disk reads (due to bitmap) as well as a smaller number of evaluations against the partition of the guarded expression.

Experiment 2: The previous experiment shows that the speedup of Sieve w.r.t. the baselines increases with an increasing number of policies, especially for PostgreSQL. We explore this aspect further on PostgreSQL using the *Mall* dataset where the generation of very large number of policies per querier (in this case the querier is a shop) is more feasible as we can simulate more customers. We used the same process than in Experiment 4 to generate cumulative set of policies by choosing 5 queriers/shops with at least 1,200 policies defined for them. Figure 32 reaffirms how the speedup of Sieve compared against the baseline increases linearly starting from a factor of 1.6 for 100 policies to a factor of 5.6 for 1,200 policies. We analyzed the query plan selected by the optimizer for the Sieve rewritten queries. We observed that with larger number of guards, PostgreSQL is able to utilize the bitmaps in memory to gain additional speedups from guarded expressions (as explained in Experiment 1). Also, this experiment shows that Sieve outperforms the baseline for a different dataset which shows the generality of our approach.

3.2.4.2 Preventing Leakages on Access Controlled Protected Data

Access Control mechanisms enforce policies by either allowing or denying access to a sensitive object. This way, they might not be sufficient for protecting sensitive objects since an adversary with background knowledge might infer information about the sensitive data from non-sensitive data. The problem of learning about sensitive data from non-sensitive data combined with metadata is known as the *inference problem*.

We now study the inference problem in databases with discretionary access control with two classes of data dependencies. The first one consists of commonly used types of data constraints (such as functional dependencies, conditional functional dependencies, etc.) which are expressed in the form of denial constraints [88]. In modern Database Management Systems, raw data is transformed into derived data through various user defined functions [89]. Depending upon the property of the enrichment function, it might be also possible to reconstruct the raw data when only the derived data is shared. The second type of data constraint called *provenance-based dependencies (PBD)* captures these different forms of relationship between raw data and enriched data. These dependencies are publicly known and constitute the inference channels available to the adversary. They can use this along with the disclosed non-sensitive data to limit the set of possible values that the sensitive data can (or cannot) take or in some cases be able to completely reconstruct the sensitive data.

Example of leakage through Dependencies

Let us consider the following example, with a simple conditional functional dependency (CFD), to illustrate the the inference problem.

Consider the Employees table, shown in Table 11 where every tuple from the employee

table specifies an employee in a department with their employee id (*Eid*), employee full name (*EName*), Zip code (*Zip*), state residence (*State*), role in the department (*Role*), number of hours they are allowed to work every week (*WorkHrs*), and the salary they earn per hour (*SalPerHr*). Consider an access control policy specified by a user to hide their *WorkHrs*. If there exists a conditional functional dependency on the Employee table, “[*Role*=‘Staff’] → [*WorkHrs*=‘30’]” the adversary can learn about their weekly work hours by querying roles in the department and checking if it is equal to ‘Staff’.

Table 11: Employee details table.

| Emp | Eid | EName | Zip | State | Role | WorkHrs | SalPerHr |
|------------|------------|--------------|------------|--------------|-------------|----------------|-----------------|
| e_1 | 34 | A. Land | 45678 | AZ | Student | 20 | 40 |
| e_2 | 56 | B. Hill | 54231 | CA | Faculty | 40 | 200 |
| e_3 | 78 | C. Wood | 53567 | CA | Faculty | 40 | 200 |
| e_4 | 12 | D. Boi | 54231 | CA | Staff | 30 | 70 |

It could be trivially observed that the inference attack in the above Example can be defended by simply hiding the corresponding *Role* when *WorkHrs* data is sensitive. In a more realistic setting, there could exist a Functional Dependency such as $SalPerHr \rightarrow Role$ or a more complex Denial Constraint such as $\forall t_i, t_j \in Emp \neg(t_i[State] = t_j[State] \wedge t_i[Role] = t_j[Role] \wedge t_i[SalPerHr] > t_j[SalPerHr])$. Both of these dependencies give more knowledge about the sensitive cell to the adversary. In such situations, identifying and preventing against potential inferences on sensitive data becomes challenging because the leakage can propagate through different dependencies. Furthermore, these dependencies can span over a number of tuples in the database and can include conditions on multiple attributes. This makes it difficult to determine the non-sensitive data that should be hidden to prevent inferences on sensitive data.

Cell representation

To simplify notational overhead, we use a cell representation instead of relation-tuple-attribute representation. We introduce the notation of cells for reason of flexibility and simplicity in discussing the fine-grained access control policies and the complex compositions among data dependencies. In this representation, a database can be regarded as a set of **cells**, $\mathcal{D} = \{\dots, c_k, \dots\}$, where we use the notation c_k to denote the **cell** with ID k where each $c_k = t_i[A_j]$ from the previous representation. A cell can be **assigned** with a value and we use $c_k.val$ to denote the **cell value** assigned to the cell c_k . The **domain** of c_k is denoted by $Dom(c_k)$ which is the domain of the attribute A_j . The **size of the domain** is correspondingly denoted by $|Dom(c_k)|$, which is equivalent to the size of domain of the attribute that the cell is associated with. The cell notation system is equivalent to or interchangeable with the relation-tuple-attribute notation system, if considering a function (i.e. one-to-one mapping) *flatten* and its inverse function $flatten^{-1}$ that could map $t_i[A_j]$ to a cell in the cell representation and vice versa. Therefore, from now on, we will simply use the notation for a set of cells $\mathbb{C} = \{\dots, c_k, \dots\}$ to represent a row or a set of rows, a column or a set of columns, or a table in the database, if the context is clear.

A given cell c is sensitive to a user u if there exists a deny policy P (policy with *Action* = “*Deny*”) such that subject conditions in P denotes the u and object conditions identify the c . The set of cells sensitive to the user u is denoted by \mathbb{C}_U^S (or simply \mathbb{C}^S when the context is clear). The sensitive set cannot appear in the result of queries by U , which is restricted by the access control policies. Conversely, the set of non-sensitive cells are denoted by \mathbb{C}^{NS} where $\mathbb{C}^{NS} = \mathcal{D} - \mathbb{C}^S$.

Types of Dependencies

Data dependencies restrict the possible set of values for a cell based on another set of values in the database instance. Thus, through existing dependencies, knowledge about sensitive cells restricted by access control policies can leak to queriers. We look at two forms of data dependencies in this work: 1) Denial constraints (DCs) and 2) Provenance-based dependencies (PBDs).

Denial Constraints are traditional types of integrity constraints such as keys, foreign keys, functional dependencies (FDs), conditional functional dependencies (CFDs), and check constraints. We use the general model of denial constraints as our constraint definition language to represent all forms of constraints, including aforementioned integrity constraints. Thus, from now on, we use the term *data constraint* and *denial constraint* interchangeably in this paper. We chose DCs to express the data dependencies as it is capable of modelling common kind of dependencies (such as FDs, CFDs) and also flexible enough to model more complex dependencies among cells. We use the general notation of denial constraints (as in [88]) to represent data dependencies at the schema level. Under this representation, a DC is a first-order formula of the form $\forall t_i, t_j, \dots \in \mathcal{D}, \Delta_k \neg(Pred_1 \wedge Pred_2 \wedge \dots \wedge Pred_N)$ where $Pred_i$ is the i th predicate in the form of $t_x[A_j]\theta t_y[A_k]$ or $t_x[A_j]\theta const$ with $x, y \in \{i, j, \dots\}$, $A_j, A_k \in R$, $const$ is a constant, and $\theta \in \{=, >, <, \neq, \geq, \leq\}$. We skip the universal quantifiers for DC if it is clear from the context. A DC is satisfied if at least one of the predicates evaluates to *False* which results in DC evaluating to *True*.

Provenance Based dependencies: We present a model for dependencies used to capture the relationships between derived data and its inputs. A *Provenance Based Dependency (PBD)* captures this relationship between the derived value and input values based on the function. In general, given a function fn with r_1, r_2, \dots, r_n as the input cell and s_i as the derived or output cell, the PBD is represented by $fn(r_1, r_2, \dots, r_n) = s_i$. If δ is a PBD, then $fn(\tilde{\delta})$ returns the corresponding function associated with it. Function definitions (or schema level PBD) are published as part of the schema (just like FDs, DCs) and is considered as background knowledge.

Security Model

An *assignment to a cell c* is a value, x , assigned to it from its state such that $x \in State(c)$. The assigned world, or simply world, is a set of assigned values, $\{x_1, x_2, \dots, x_n\}$, to cells in an instantiated dependency given by $\tilde{\delta}(c_1 = x_1, c_2 = x_2 \dots, c_n = x_n)$ where each $x_n \in State(c_n)$. The world is denoted by $W(\mathbb{C}) \in State(\mathbb{C})$ where $State(\mathbb{C})$ is the set of all possible worlds based on states of all cells in \mathbb{C} .

In a Denial Constraint, $Preds(\tilde{\delta})$ returns the predicates from an instantiated dependency.

For a given cell $c_i \in \mathbb{C}$ and a dependency $\tilde{\delta}$, $Preds(c_i, \tilde{\delta}(\mathbb{C}))$ returns the predicate(s) $Pred \in Preds(\tilde{\delta})$ such that $Pred = c_i \theta c_j$ or $Pred = c_i \theta const$ where $c_i, c_k \in \mathbb{C}$, and $const$ is a constant. The function $Preds(c_i, \tilde{\delta})$ returns ϕ if $\tilde{\delta}$ doesn't contain such a predicate.

Let \mathbb{C}_{Pred} denote the cells associated with the $Pred$ such that $\forall c_i \in \mathbb{C}_{Pred}, Pred \in Preds(c_i)$. We define the evaluation function for a predicate as: $eval(Pred, W(\mathbb{C}_{Pred})) = True$ if the predicate evaluates to True based on the assignment of values to \mathbb{C}_{Pred} from $W(\mathbb{C}_{Pred})$. Similarly, $eval(Pred, W(\mathbb{C}_{Pred})) = False$ if the predicate evaluates to False based on the assignment of values to \mathbb{C}_{Pred} from $W(\mathbb{C}_{Pred})$. We also define the evaluation function based on $State(\mathbb{C}_{Pred})$ as

$$eval(Pred, State(\mathbb{C}_{Pred})) = \begin{cases} \text{True} & \text{if } \forall W \in State(\mathbb{C}_{Pred}), eval(Pred, W(\mathbb{C}_{Pred})) = True \\ \text{False} & \text{if } \forall W \in State(\mathbb{C}_{Pred}), eval(Pred, W(\mathbb{C}_{Pred})) = False \\ \text{Unknown} & \text{if } \exists W_1, W_2 \in State(\mathbb{C}_{Pred}) \text{ such that} \\ & eval(Pred, W_1(\mathbb{C}_{Pred})) = True \text{ and } eval(Pred, W_2(\mathbb{C}_{Pred})) = False \end{cases}$$

Similarly, the eval function for the instantiated dependency $\tilde{\delta}$ and an assignment $W \in State(\mathbb{C})$ returns true (i.e., $eval(\tilde{\delta}, W) = True$). We call this W a *valid world* for a dependency if it does not lead to dependency violations (i.e., assignments that do not violate the dependency). As our dependencies are expressed in Denial Constraints, W is a valid assignment to the $\tilde{\delta}$ if $\exists Pred \in Preds(\tilde{\delta})$ such that $eval(Pred, W(\mathbb{C}_{Pred})) = False$. An invalid assignment occurs when $W \in State(\mathbb{C})$ and $eval(\tilde{\delta}, W) = False$.

A state of c_i denoted by $State(c)$ is the set of possible values that can be assigned to it from its domain ($Dom(c)$). The state of a cell c changes based on the knowledge of the adversary. When the cell is disclosed, the $State(c) = x$ where $x \in Dom(c)$. On the other hand, when a cell is sensitive and therefore hidden, $State(c) \subseteq Dom(c)$ based on various instantiated dependencies (an instantiated dependency is a schema level dependency instantiated with different cells from the database) the cell is part of and adversary's background knowledge. *State Function (sf)* which computes the $State(c^*)$ based on an instantiated dependency and the state of its cell set ($State(\mathbb{C})$ and $c^* \in \mathbb{C}$) is given by

$$sf(c^* | \tilde{\delta}, State(\mathbb{C})) = \{x \in State(c^*) \mid \exists W \in State(\mathbb{C}), eval(\tilde{\delta}, W) = True\} \quad (31)$$

We first define $State^{max}(\mathbb{C})$ as the set of possible values for all cells in \mathbb{C} when the adversary has no knowledge about any of the cells (other than the previously mentioned background knowledge). We can compute the set of possible values for $c^* \in \mathbb{C}$ based on an instantiated dependency $\tilde{\delta}$ and $State^{max}(\mathbb{C})$ using the previously defined State Function.

$$sf(c^* | \tilde{\delta}, State^{max}(\mathbb{C})) = \{x \in State(c^*) \mid \exists W \in State^{max}(\mathbb{C}) eval(\tilde{\delta}, W) = True\}$$

This returns the set of possible values for a c^* from the valid worlds for $\tilde{\delta}$ and its maximum achievable deniability value. Upon sharing some of the cells in $\mathbb{C} \in \mathbb{C}^{NS}$, partially or completely, the adversary learns more about the state of \mathbb{C} . We denote this updated state

of \mathbb{C} as $State'(\mathbb{C})$. The new set of possible values for c^* based on adversary's knowledge is given by:

$$sf(c^* \mid \tilde{\delta}, State'(\mathbb{C})) = \{x \in State(c^*) \mid \exists W \in State'(\mathbb{C}) \text{ eval}(\tilde{\delta}, W) = True\}$$

We compute the full state of a cell based on $State^{max}(\mathcal{D})$, $State'(\mathcal{D})$, and the set of all dependencies Δ . We now define two different security models for our problem setting: 1) *Full Deniability* and 2) *k-value Deniability*. Full deniability occurs when the adversary cannot distinguish the actual value of the cell from any of the possible values in $sf(c^* \mid \tilde{\delta}, State^{max}(\mathbb{C}))$ (i.e., they learn no new knowledge from the dependencies and disclosure of non-sensitive cells) i.e., For every sensitive cell $c^* \in \mathbb{C}^S$, based on all the dependencies $\tilde{\delta} \in \Delta$, we achieve full deniability when $sf(c^* \mid \Delta, State^{max}(\mathcal{D})) = sf(c^* \mid \Delta, State'(\mathcal{D}))$. *Full deniability* is sometimes hard to achieve and in the worst case might require denying almost all the cells in the database when a relatively large number of cells are marked as *sensitive* by the access control policies. Therefore, inspired by the well-known *k-anonymity* privacy, we relax our security definition to a novel definition called *k-value deniability*. For every sensitive cell $c^* \in \mathbb{C}^S$, based on all the dependencies $\tilde{\delta} \in \Delta$ and \mathcal{D} , we achieve k-value deniability when $sf(c^* \mid \Delta, State^{max}(\mathcal{D})) - sf(c^* \mid \Delta, State'(\mathcal{D})) \leq k \cdot Dom(c^*)$

Leakage Analysis

When c^* is marked as sensitive to u by an access control policy, it is hidden by setting it to *NULL*. This removes the sensitive cell from the query results for that user u . However, if there exist an instantiated dependency $\tilde{\delta}$ which contains the sensitive cell, it is possible for the adversary to learn about the hidden sensitive cell. We explain the conditions under which it is possible for the adversary to learn about the sensitive cell through denial constraints and provenance based dependencies.

Denial Constraints: Suppose $\tilde{\delta}$ is a denial constraint and the predicate corresponding to the sensitive cell is given by $Pred(c^*)$. We first like to note the semantics of a denial constraint that at least one predicate in a DC has to be False in a valid and clean database (only valid worlds from $W \in State(\mathbb{C})$ such that $eval(\tilde{\delta}, W) = True$ are considered). When the denial constraint is not trivial and contains at least two predicates ($|Preds(\tilde{\delta})| \geq 2$), after hiding the sensitive cell c^* , we have $State(c^*) \subseteq Dom(c^*)$ and thus $eval(Pred(c^*), \mathbb{C}) = Unknown$. However, when all the other predicates in the dependency evaluate to True (i.e., $\forall Pred_i \in Preds(\tilde{\delta}) \setminus Pred(c^*) \text{ eval}(Pred_i, State(\mathbb{C})) = True$) the adversary can infer that $eval(Pred(c^*), \mathbb{C}) = False$ even if they do not know the exact value of c^* .

Provenance based Dependencies: Suppose δ is a provenance based dependency given by $fn(r_1, r_2, \dots, r_n) = s_i$. In the instantiation of the PBD denoted by $\tilde{\delta}$, if the sensitive cell c^* is the output of the function then disclosing any of the input values leaks knowledge about c^* . Furthermore, if any of the input values is sensitive (i.e., c^*) disclosing the output value leaks knowledge about it only if the fn is invertible (if a fn is invertible, given output value(s) it is possible to infer about the input). If fn is non-invertible, disclosing output value does not leak any knowledge about c^* .

In both of these situations, we have to hide other cells in the cell set of instantiated dependency (c in \mathbb{C}) so as to prevent the leakage of the sensitive cell through this dependency.

In DCs, our goal is to have one other predicate in the dependency that evaluates to unknown and thus making it impossible for the adversary to infer the truth value of $Pred(c^*)$. We achieve this by hiding a cell $c_j \in \mathbb{C}$ such that $Pred(c_j) \in Preds(\tilde{\delta}) \setminus Pred(c^*)$. This results in $eval(Pred(c_j), State(\mathbb{C})) = Unknown$. As now two predicates in $\tilde{\delta}$: $Pred(c_j)$, $Pred(c^*)$ evaluate to unknown (and either could evaluate to *False* to satisfy the DC semantics), it is impossible for adversary to learn anything about the sensitive cell c^* through this $\tilde{\delta}$.

When the condition for leakage is met, the set of non-sensitive cells, from the cell set of an instantiated dependency $\tilde{\delta}$ that can leak knowledge about the sensitive cell, is called a *cueset*. Conversely, denying at least one of the cells in the cueset will make c^* secure w.r.t that dependency.

Preventing Leakages

Given a database \mathcal{D} which is a collection of cells and a set of data dependencies Δ , and a set of fine-grained access control policies P_u that identify the cells (\mathbb{C}^S) that should be denied while answering the queries by u . Our goal is to generate \mathcal{D}' , on which the queries by u are answered, and which protects the sensitive cells $c_i \in \mathbb{C}^S$ while maximizing utility. Utility is defined as the number of cells that are shared from $\mathcal{D} - \mathbb{C}^S$ while meeting the deniability requirement.

We perform the following steps to prevent data leakages of sensitive cells. All these steps are done via pre-processing at compile time as all the necessary information to do so is available prior to query time.

The first step is sensitivity determination, which takes as input the policies applicable to a user and produces a set of cells which are marked as sensitive. This is done by adding the sensitivity metadata for each tuple and its attribute values similar to the approach by [90]. In the next step, for each of the sensitive cell, we instantiate their relevant dependencies and detect cuesets for them. We take each schema level dependency δ and instantiate it with the sensitive cell c^* and the appropriate selection of the set of other cells $\mathbb{C} \in \mathcal{D}$. The instantiation of the dependency is hinged on the type of dependency. For a unary dependency, instantiation is only based on the tuple the sensitive cell is part of. For a binary dependency (most DCs), instantiation is based on the pairwise comparison of tuple containing the sensitive cell as well as other tuples in the database. Similarly for a N -ary dependency, the tuple consisting of the sensitive cell will be compared against the set of other N tuples. The number of comparisons required for instantiation depends upon the number of predicates in the dependency.

In the next step, we check for each of the instantiated dependency whether we need to generate a cueset based on the condition for leakage presented earlier. We verify the condition by assigning values to each $Pred_j$ (except for $Pred(c^*)$) for its corresponding \mathbb{C}_{Pred_j} and verifying if $eval(Pred_j, State(\mathbb{C}_{Pred_j})) = True$. If one of the predicates evaluates to *False*, we generate an empty cueset and move onto the next instantiation. On the other hand, if the condition is met, we generate the cueset by iterating through all the predicates $Pred_j$ (except for $Pred(c^*)$) and adding the cells to the cueset corresponding to that dependency instantiation. The exception to this rule is when the instantiated dependency contains only a single predicate. We generate a cueset consisting of the non-sensitive cell (c_k) in $Pred(c^*)$.

After iterating through all the dependency instantiations, we return the *cuesets* which is a set of cuesets. Finally, we select cells to hide from the cueset until the specified deniability requirement is met. For a given sensitive cell c^* , the algorithm identifies the cuesets based on the instantiated dependencies by calling cueset detection function. Then it iterates through the set of cuesets and for each cueset, the algorithm selects a cell to hide. Then it identifies the cuesets of the hidden cell and these new cuesets are added to list of cuesets. The previous cueset is removed after this step or if it already contains a hidden cell. We repeat this process until list of cuesets is empty which means every cueset has at least one hidden cell and thus we achieve Full deniability for the sensitive cell.

3.2.5 Supporting Verification of Data Capture Policies via IoT Notary

Smart buildings require end-users to trust data-capturing rules published by the systems. There are several reasons why such a trust is misplaced — IoT systems may violate the rules deliberately or IoT devices may transfer user data to a malicious third-party due to cyberattacks, leading to the loss of individuals' privacy or service integrity. To address such concerns, we propose IoT NOTARY, a framework to ensure trust in IoT systems and applications. IoT NOTARY provides secure log sealing on live sensor data to produce a verifiable 'proof-of-integrity,' based on which a verifier can attest that captured sensor data adheres to the published data-capturing rules. IoT NOTARY is an integral part of TIPPERS. IoT NOTARY imposes nominal overheads. The secure logs only take 21% more storage, while users can verify their one day's data in less than two seconds even using a resource-limited device.

Advantages of IoT Notary

IoT NOTARY has the following distinct advantages:

1. **Privacy-preserving system.** IoT NOTARY is the first privacy-preserving system for the enforcement of IoT sensor data capturing policies and for the data attestation at the user to verify the presence/absence of their data against the data capturing policies. Moreover, during the policies' enforcement and attestation, any entity involved in IoT NOTARY cannot learn anything about other entities.
2. **Minimal overhead in proof generation.** IoT NOTARY does not incur computational and space overhead in secure log generation. In particular, IoT NOTARY takes 310ms for creating the secure log over 37K rows and 21% more space to store the log in a secure manner as compared to store the log in clear-text.
3. **Scalable.** IoT NOTARY is a highly scalable system in terms of data verification. In particular, a resource-constrained user (with a machine of 1-core 1GB RAM) can verify the presence or absence of their data over the data collected in a single day in most 30 seconds.

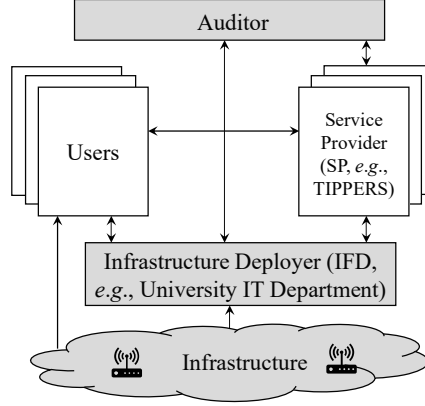


Figure 33: Entities in IoT NOTARY.

4. **A deployed system.** At the University of California, Irvine, IoT NOTARY has been deployed and is in use to capture WiFi connectivity events. The real-world testbed has been established to provide a real environment where IoT NOTARY's usability is tested and verified.

IoT Notary Model

IoT NOTARY model has the following entities, as shown in Figure 33:

Infrastructure Deployer (IFD). IFD (which is the university IT department in our use-case) deploys and owns a network of p sensors devices (denoted by s_1, s_2, \dots, s_p), which capture information related to users in a space. The sensor devices could be: (i) dedicated sensing devices, *e.g.*, energy meters and occupancy detectors, or (ii) facility providing sensing devices, *e.g.*, WiFi access-points and RFID readers. Our focus is on facility-providing sensing devices, especially WiFi access-points that also capture some user-related information in response to services. *E.g.*, WiFi access-points capture the associated user-device-ids (MAC addresses), time of association, some other parameters (such as signal strength, signal-to-noise ratio); denoted by: $\langle d_i, s_j, t_k, param \rangle$, where d_i is the i^{th} user-device-id, s_j is the j^{th} sensor device, t_k is k^{th} time, and $param$ is other parameters (we do not deal with $param$ field and focus on only the first three fields). All sensor data is collected at a controller (server) owned by IFD. The controller may keep sensor data in clear-text or in encrypted form; however, it only sends encrypted sensor data to the service provider.

Service Providers (SP). SP (which is TIPPERS in our use-case) utilizes the sensor data of a given space to provide different *services*, *e.g.*, monitoring a location and tracking a person. SP receives encrypted sensor data from the controller.

Data-capture rules. SP establishes data-capture rules (denoted by a list DC having different rules dc_1, dc_2, \dots, dc_q). Data-capture rules are conditions on device-ids, time, and space. Each data-capture rule has an associated *validity* that indicates the time during which a rule is valid. Data-capture rules could be to capture user data by default (unless the user has explicitly opted out). Alternatively, default rules may be to opt-out, unless, users opt-in explicitly. Consider a default rule that individuals on the 6th floor of the building will be

monitored from 9pm to 9am. Such a rule has an associated condition on the time and the id of the sensor used to generate the data. Now, consider a rule corresponding to a user with a device d_i opting-out of data capture based on the previously mentioned rule. Such an opt-out rule would have conditions on the user-id, as well as, on time and the sensor-id. For sensor data for which a default data-capture rule is opt-in, the captured data is forwarded to SP, if there do not exist any associated opt-out rules, whose associated conditions are satisfied by the sensor data. Likewise, for sensor data where the default is opt-out, the data is forwarded to SP only, if there exists an explicit opt-in condition. We refer to the sensor data to have a *sensor state* ($s_i.state$ denotes the state of the sensor s_i) of 1 (or active), if the data can be forwarded to SP; otherwise, 0 (or passive). In the remaining paper, unless explicitly noted, opt-out is considered as the default rule, for simplicity of discussion.

Whenever SP creates a new data-capture rule, SP must send a *notice message* to user devices about the current usage of sensor data (this phase is entitled *notification phase*). SP uses Intel Software Guard eXtension (SGX) [91], which works as a trusted agent of IFD, for securely storing sensor data corresponding to data-capture rules. SGX keeps all valid data-capture rules in the secure memory and only allows to keep such data that qualifies pre-notified valid data-capture rules; otherwise, it discards other sensor data. Further, SGX creates immutable and verifiable logs of the sensor data (this phase is entitled *log-sealing phase*). The assumption of secure hardware at a machine is rational with the emerging system architectures, *e.g.*, Intel machines are equipped with SGX [92]. However, existing SGX architectures suffer from side-channel attacks, *e.g.*, cache-line, branch shadow, page-fault attacks [93], which are outside the scope of this paper.

Users. Let d_1, d_2, \dots, d_m be m (user) devices carried by $u_1, u_2, \dots, u_{m'}$ users, where $m' \leq m$. Using these devices, users enjoy services provided by SP. We define a term, entitled *user-associated data*. Let $\langle d_i, s_j, t_k \rangle$ be a sensor reading. Let d_i be the i^{th} device-id owned by a user u_i . We refer to $\langle d_i, s_j, t_k \rangle$ as user-associated data with the user u_i . Users worry about their privacy, since SP may capture user data without informing them, or in violation of their preference (*e.g.*, when the opt-out was a default rule or when a user opted-out from an opt-in default). Users may also require SP to prove service integrity by storing all sensor data associated with the user (when users have opted-in into services), while minimally being involved in the attestation process and storing records at their sides (this phase is entitled *attestation phase*).

Auditor. An auditor is a *non-mandatory* trusted third-party that can (periodically) verify entire sensor data against data-capture rules. Note that a user can only verify his/her data, not the entire sensor data or sensor data related to other users, since it may reveal the privacy of other users.

Security Properties of IoT Notary

An adversary wishes to learn the (entire/partial) data about the user, without notifying or by mis-notifying about data-capture rules, such that the user/auditor cannot detect any inconsistency between data-capture rules and stored sensor data at SP. Hence, a secure

attestation algorithm must make it detectable, if the adversary stores sensor data in violation of the data-capture rules notified to the user. To achieve a secure attestation algorithm, we satisfy the following properties:

Authentication. Authentication is required: (i) between SP and users, during notification phase; thus, the user can detect a rogue SP, as well as, SP can detect rogue users, and (ii) between SP and the verifier (auditor/user), before sending sensor data to the verifier to prevent any rogue verifier to obtain sensor data. Thus, authentication prevents threats such as impersonation and repudiation. Further, a periodic mutual authentication is required between IFD and SP, thereby discarding rogue sensor data by SP, as well as, preventing any rogue SP to obtain real sensor data.

Immutability and non-identical outputs. We need to maintain the immutability of notice messages, sensor data, and the sealing function. Note that if the adversary can alter notice messages after transmission, it can do anything with the sensor data, in which case, sensor data may be completely stored or deleted without respecting notice messages. Further, if the adversary can alter the sealing function, the adversary can generate a proof-of-integrity, as desired, which makes flawless attestation impossible. The output of the sealing function should not be identical for each sensor reading to prevent an adversary to forge the sealing function (and to prevent the execution of a frequency-count attack by the user). Thus, immutability and non-identical outputs properties prevent threats, *e.g.*, inserting, deleting, modifying, and truncating the sensor data, as well as, simulating the sealing function.

Minimality, non-refutability and privacy-preserving verification. The verification method must find any misbehavior of SP, during storing sensor data inconsistent with pre-notified data-capture rules. However, if the verifiers wish to verify a subset of the sensor data, then they should not verify the entire sensor data. Thus, SP should send a minimal amount of sensor data to the verifier, enabling them to attest to what they wish to attest. Further, the verification method: (i) cannot be refuted by SP, and (ii) should not reveal any additional information to the user about all the other users during the verification process. These properties prevent SP to store only sensor data that is consistent with the data-capture rules notified to the user. Further, these properties preserve the privacy of other users during attestation and impose minimal work on the verifier.

IoT Notary: An Overview

Now, we present an overview of the three phases and dataflow among different entities and devices in IOT NOTARY, see Figure 34.

Notification phase: SP to Users messages. This is the first phase that notifies users about data-capture rules for the IoT space using notice messages (in a verifiable manner for later stages). Such messages can be of two types: (i) notice messages, and (ii) notice-and-acknowledgment messages. SP establishes (the default) data-capture rules and informs trusted hardware (❶). Trusted hardware securely stores data-capture rules (❷, ❺) and informs the *trusted notifier* (❸) that transmits the message to all users (❹). Only notice messages need a trusted notifier to transmit the message.

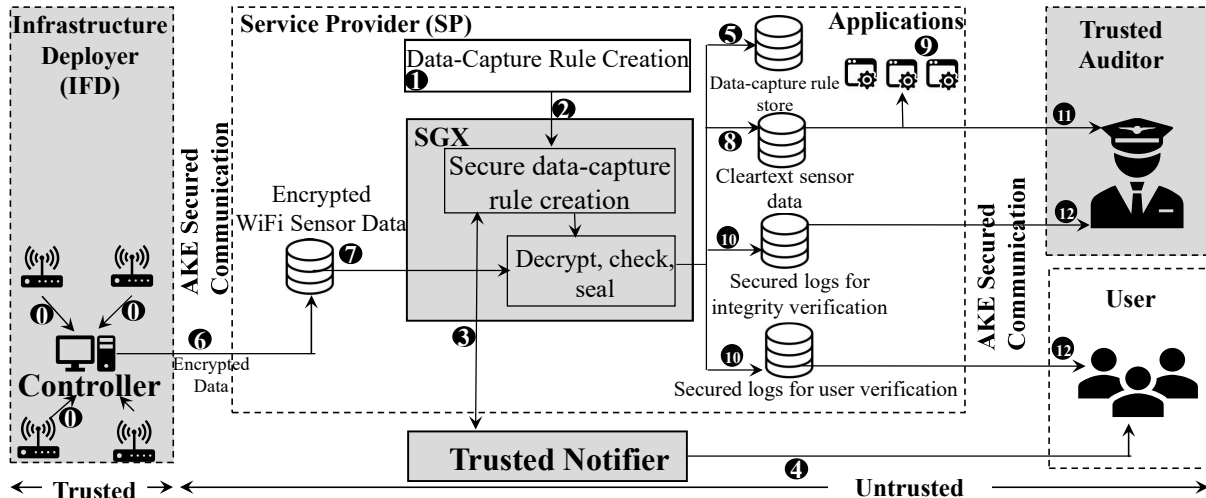


Figure 34: Dataflow and computation in the protocol. Trusted parts are shown in shaded box.

Log-sealing phase: Sensor devices to SP messages. Each sensor sends data to the controller (0). The controller receives the correct data, generated by the actual sensor, as per our assumptions (and settings of the university IT department). The controller sends encrypted data to SP (6) that authenticates the controller using any existing authentication protocol, before accepting data. Trusted hardware (Intel SGX) at SP reads the encrypted data in the enclave (7).

Working of the enclave. The enclave decrypts the data and checks against the pre-notified data-capture rules. Recall that the decrypted data is of the format: $\langle d_i, s_j, t_k \rangle$, where d_i is i^{th} user-device-id, s_j is the j^{th} sensor device, and t_k is k^{th} time. After checking each sensor reading, the enclave adds a new field, entitled *sensor (device) states*. The sensor state of a sensor s_j is denoted by $s_j.state$, which can be **active** or **passive**, based on capturing user data. For example, $s_j.state = \text{active}$ or (1), if data captured by the sensor s_j satisfies the data-capture rules; otherwise, $s_j.state = \text{passive}$ or (0). For all the sensors whose $state = 0$, the enclave deletes the data. Then, the enclave cryptographically seals sensor data, regardless of the sensor state, and provides clear-text sensor data of the format: $\langle d_i, s_j, s_j.state = 1, t_k \rangle$ to SP (8) that provides services using this data (9). Note that the cryptographically sealed logs and clear-text sensor data are kept at untrusted storage of SP (8, 10).

Verification phase: SP to verifier messages. In our model, an auditor and a user can verify the sensor data. The auditor can verify the entire/partial sensor data against data-capture rules by asking SP to provide clear-text sensor data and cryptographically sealed logs (8, 10). The users can also verify their own data against pre-notified messages or can verify the results of the services provided by SP using only cryptographically sealed logs (12). Note that using an underlying authentication technique (as per our assumptions), auditor/users and SP authenticate each other before transmitting data from SP to auditor/users.

3.3 System Testbed

In the following, we explain the results of the project in terms of prototypes and systems developed. We divide the section into the following elements:

- Sensing infrastructure, which explains the deployment of sensors and software developed to manage those.
- System, which details the software components developed to realize the TIPPERS system.
- Applications and other software developed to showcase the system and highlight privacy challenges.

3.3.1 Sensing Infrastructure

We instrumented the DBH building with different sensors including bluetooth beacons, video cameras, and microphones in addition to the sensors already present in the building. For instance, we set up the necessary firewalls in our server through which we were able to pull the HVAC dataset using SkySpark REST APIs. There are around 200 VAV boxes in this building and each box contains 30 different sensors. We have registered these 6000 sensors in our TIPPERS system. The frequency of each of these data points is usually once in every 15 minutes but for some sensors it is very frequent (once every minute). The frequency of the update depends on the amount of change in their values and the duration from the last update time. At any given instant there can be around 6000 sensor readings from these sensors based on their frequency of updates.

To communicate with those sensors, we developed “wrappers”, that is, code to provide sensors with an interface to TIPPERS. This code can be deployed in a server and provides TIPPERS with a set of APIs to start/stop the collection of data from the sensors associated with it. For instance, in the case of the HVAC sensors above, we developed a wrapper that encapsulates the communication with the SkySpark server. We designed such a wrapper in order to support the access to specific sensors and sensor types and their observations taken during a period of time. We have developed other wrappers to, for example, communicate with the microphone in our smart meeting room, and another one to record video from the video cameras (previously we were just capturing images from the cameras).

The most important sensor used in our deployments is the WiFi Access Point (AP). The observations that such a sensor captures (i.e., the MAC address of the device connected to it at a certain time instant) can be leveraged to produce localization of individuals and occupancy of spaces. We also have created a service, WiFi2Location which converts WiFi AP observations (along with metadata such as the coverage of each WiFi AP and the owner of each specific device connected to the AP) into location of users within the space. We defined another service, WiFi2Occupancy, which converts WiFi AP observations and metadata into occupancy of the spaces covered by such WiFi AP. Along with these two services, we had defined Video2Occupancy service, which given an image observed by a video camera uses a

deep neural network to count the number of people present in it. Then, it assigns such count to the space(s) covered by the video camera.

Localization Virtual Sensor

The goal of this virtual sensor is to predict the location of a person given WiFi connectivity data. The focus was on increasing the accuracy of the prediction. This results in better performance of the applications and allows us to understand how different information captured in the building can be used to obtain precise location of people and study mechanisms to protect individuals' from this. Specifically, we designed a mechanism to make predictions for people's locations on a given timestamp only based on WiFi in the following three levels: building, region, and room. Building-level localization predicts if a user is in the building at the given time; region-level solution predicts the WiFi AP to which a user's device would be connected if their device is not currently connected to the network; and room-level decides the room where the user is in.

The goal of building and region level localization is to fill in the gaps between user's WiFi data observations. When the user is in the building, his/her device sends signals to nearby access points periodically. Each row in the WiFi observation table in TIPPERS represents a connection signal received. The building and region level localization try to predict whether a user is inside the building or not and the possible WiFi access point the user's device should connect to if the user is inside. The intervals between these signals vary a lot due to different device types and different user's habits using devices, which makes the prediction difficult.

Our prediction method uses an unsupervised learning model mainly based on the user's historical connection entries. For inside and outside classification, the model analyzes each minute in a day and calculates the probability that the user is inside the building for each minute. The inside and outside decision for each time interval is based on the inside probability of each minute. For access point prediction, the model calculates the probabilities that the user's device can connect to each access point at each minute and predict the access point based on these probabilities

Room level localization seeks to find the accurate room user is in among several candidate rooms only based on WiFi data. The core idea is to draw the affinity between person and space and the affinity among people to make predictions based on historical connectivity data. For instance, people that are connected at the same time to the same access point for different periods of time are considered to have a higher affinity to each other. Based on that, our solution includes three parts. First of all, we propose a graph-based algorithm to model the observations of users as well as the affinity information and make predictions automatically. Second, we present techniques for learning affinity from historical WiFi data, and further an incremental offline learning to speed up the overall algorithm greatly. Third, considering scalability, we develop an online fashion by maintaining a social graph to select the strongest affinity group with queried users to minimize I/O time cost.

3.3.2 System Infrastructure

The main components of the system are:

- PE-IoT
- TIPPERS System
 - Front-end (TPortal)
 - Backend (APIs, translation, wrappers, virtual sensors)
 - TIPPERS Smartphone Client

PE-IoT

PE-IoT is our framework that helps IoT data providers to achieve privacy compliance when they want to share their data with other service providers. PE-IoT takes as input the raw sensor data collected from sensors or existing infrastructures, checks the organizational and individual data policies, and applies privacy-enhancing technologies to generate privacy-preserving data. As shown in Figure 35, An in-production instance of PE-IoT is developed by the TIPPERS team, and it is owned and hosted by OIT as a long-term service deployed at the UCI campus. It de-identifies the WiFi connectivity data coming from WiFi AP, removes the link between it and the users, and sends de-identified data to TIPPERS.

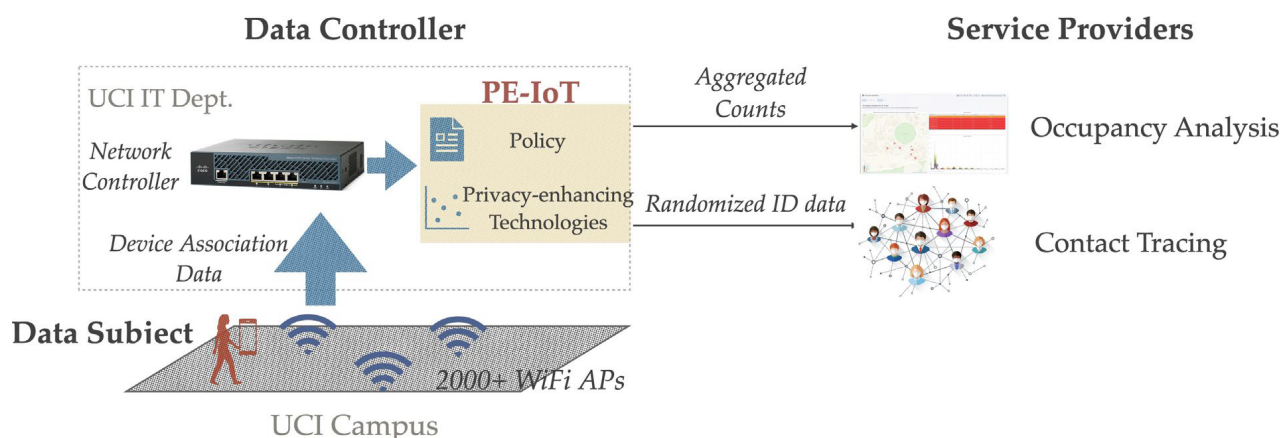


Figure 35: The PE-IoT system at UCI OIT. PE-IoT adds privacy compliance to OIT and generates privacy-preserving WiFi connectivity data service providers

The development of the PE-IoT system is motivated by the privacy-compliance needs of our data provider, UCI OIT. Since the WiFi connectivity data from OIT includes personally identifiable information, OIT needs to satisfy numerous privacy requirements to share the data. For example, individual consents and privacy policies need to be collected and honored, and OIT must apply proper privacy-enhancing technologies to ensure the data is privacy-preserving. To fulfill the requirements, We developed PE-IoT as a middleware that intercepts WiFi connectivity data sharing flow in OIT to add privacy compliance.

There are still some challenges in designing the system. OIT may need to share the data with multiple authorized service providers for heterogeneous services provisioning as a data provider for WiFi connectivity data. However, different service providers have various requirements for the data, e.g., different data subsets and PETs. The lack of a unified data model significantly increases OIT's overhead since OIT will need to instrument independent and separate workflow for each service provider.

To address the challenges, we first design and use a data model in PE-IoT for privacy-preserving data sharing, which is a trade-off between fine-grained control and the overhead of privacy interventions. The key concept in the model is Data Product, an abstraction for privacy-enhanced data that data controllers can share with service providers. Precisely, three components form a data product: (1) *Filter* is a set of selection conditions on sensor data that defines a subset of the sensor data that constitutes a data product. For example, a possible filter might select sensor data from the sensors deployed in a specific building or room. Policy in a data product consists of two components: a data controller's sharing policies that specify conditions under which a service provider can access the data product, and the data subject's choices that determine the inclusion of a data subject's data in a data product. PET defines the information about the Privacy Enhancing Technology that a data controller uses when sharing the data product with service providers.

To realize the data-sharing model, PE-IoT incorporates a modularized design. This system consists of a *Data Product manager* which conducts the flow of intercepted sensor stream through PE-IoT and coordinates with different resource managers to produce the units of privacy-preserving data stream defined by Data Products. The various aspects of the PE-IoT data model are implemented as independent resource managers(as shown in Figure 36), which can be executed separately. The rationale for such a system design is two-fold: 1) decoupling functions allows each resource manager to perform its tasks independently, 2) some of the resources (e.g., data controller policy, PET) might be stored remotely, and independent resource managers allow us to move them closer to the resource. Each resource manager also includes interfaces to interact with the Data Product Manager and other resource managers.

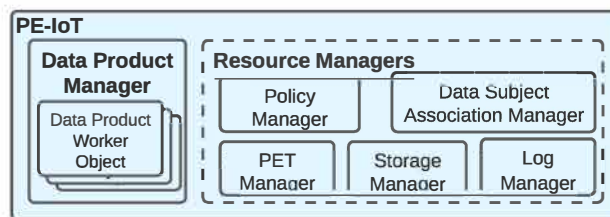


Figure 36: Components of PE-IoT System

Besides the modules mentioned above, we also developed a web interface for users to manage their WiFi connectivity data sharing policy. The interface is publicly accessible for the UCI community, and it integrates an existing campus authentication service for authentication. After logging in, this web interface utilizes the campus identity management service to get the list of users' registered devices in the UCI campus wireless network, as shown in Figure 37. The authorized users can then select their devices which they would

like to give/revoke consent of sharing WiFi connectivity data with TIPPERS.

UCI

University of
California, Irvine

Hello [redacted].

Logout

Using the UCI campus WiFi network generates logs about your device connecting to the WiFi infrastructure. OIT uses such data for operational and security purposes. OIT may further anonymize and share such data with research teams building campus level smart services. You may use this application to opt-out all or some of your devices from sharing the anonymized Wi-Fi connection information. Please visit the [OIT WiFi Security Page](#) for more information.

Registered Devices for UCInet Mobile Access:

In order for you to get Internet access from UCInet Mobile Access, you will need to register the hardware (MAC) addresses of you mobile devices. Please visit the [UCInet Mobile Access - Manual Registration page](#) to manage your registered devices

Opt-out Devices:

No WiFi connection information generated by the devices in opt-out devices list will be shared. You can add/remove the devices to/from opt-out list using this application

| Your Registered Devices for UCInet Mobile Access | |
|--|---------------------|
| MAC Address: [redacted] | Opted Out |
| MAC Address: [redacted] | Add |

Figure 37: The web interface for users to specify their policies (opt-out)

The PE-IoT system also includes a requested functionality by UCI to log certain information to support attestation by the university of the proper enforcement of policies. We have implemented functionality that logs into a database all the opt-out/in changes. In this way, if a person requests to be opted-out from data capture, or opted-in for sharing of data with a specific TIPPERS instance, this event is logged along with the timestamp. The logging mechanism incorporated into the prototype also logs the events in which connectivity data belonging to an opted-out user reached the PE-IoT system and was not shared.

TIPPERS System

Figure 38 shows different deployments of the TIPPERS system (and its components) in our server along with the technologies used to develop each of the components. In this setup, there are three instantiations of the system (on the right of the figure): Dev, Home, UCI. TIPPERS-Dev is used for development purposes and to test new functionality. TIPPERS-Home is a mocking setup to simulate a smart home. TIPPERS-UCI is the deployment that manages data from the UC Irvine campus. In each of the deployments there is a backend component developed using Java and a frontend (Tportal) which we will explain in the next sections. Also, in each instantiation there is one sample application deployed (Occupancy App) which we will also explain in next sections. Finally, in addition to the TIPPERS deployments there are three additional components (on the left of the figure): Auth, Marketplace, and Hub. Auth is the server that authenticates users and manages user information across deployments. Marketplace is the prototype of a website where others can download the system and content (e.g., applications) for it. Finally, the Hub is a repository of all the TIPPERS deployments and enables people to access each of them.

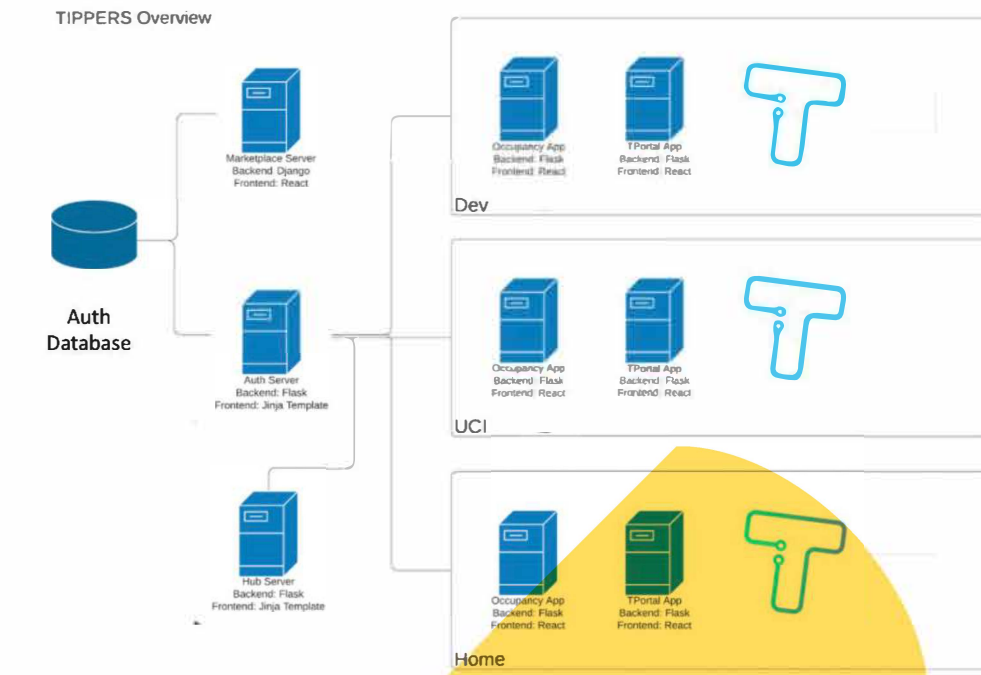


Figure 38: Different deployments of the TIPPERS System.

Figure 39 shows the deployment of the previous systems and components in our server along with the configuration of the virtual machines in which each component is installed.

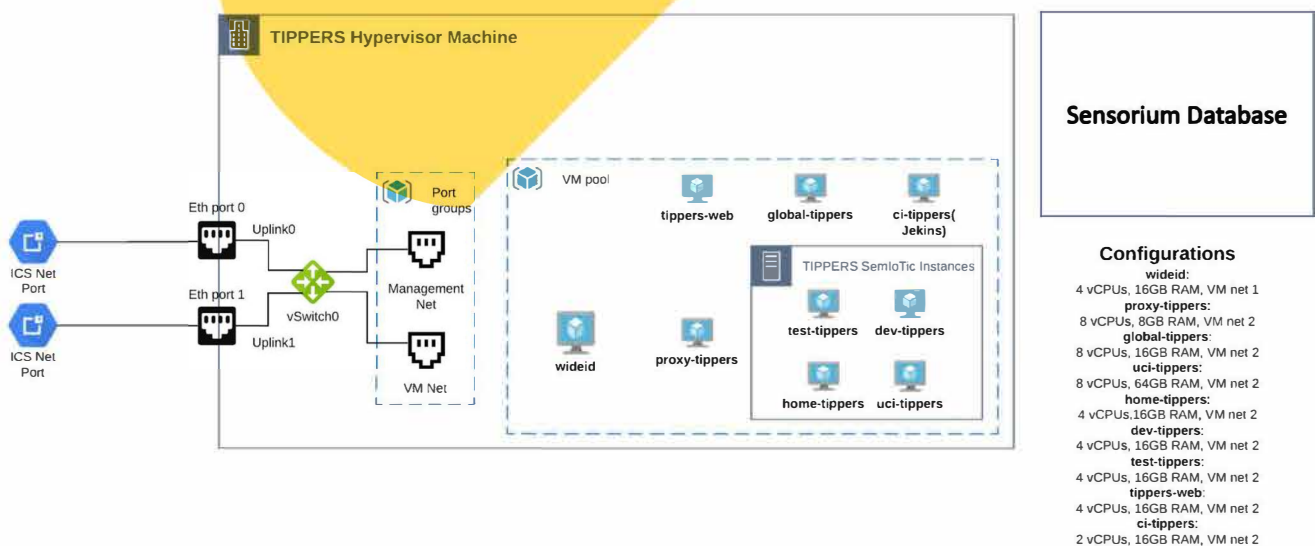


Figure 39: Server infrastructure.

In order to transfer the TIPPERS system, as well as the privacy technologies developed by our CRT partners, to others (e.g., the US Navy), TIPPERS includes a mechanism to make installation and setup of the system easier. For that, we created a docker image of TIPPERS. The docker image can be loaded at any platform (MACOS, Linux, or Windows) running docker. The loaded docker image then can be used to create and run appropriate containers while setting up the database, configuring and starting up the TIPPERS server.

In the following sections we will focus on the following components: 1) Supporting infrastructure, which includes the Auth, Hub, and T Portal; 2) TIPPERS backend.

Supporting Infrastructure

TIPPERS AUTH. Authentication of users is especially important since the access to information collected by TIPPERS is done through APIs. Such APIs use as a parameter the id of the person who is requesting data through an application and the goal of the Auth server is to authenticate such information to secure the TIPPERS APIs. The Auth server is developed using OAuth2.0 through the Python Authlib library at <https://authlib.org>. It supports Google Sign in as well as TIPPERS sign in (see Figure 40).

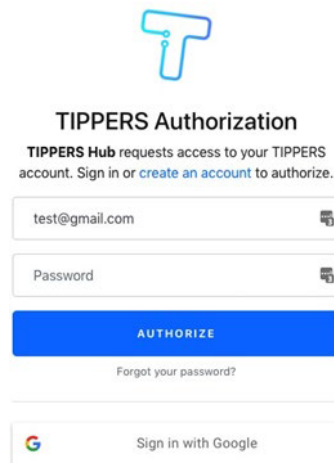


Figure 40: TIPPERS authentication GUI.

The TIPPERS Auth server includes three different authorization flows available to TIPPERS developers which want to retrieve data from TIPPERS using its APIs. Table 12 summarizes the flows and their features which can be utilized depending on the application. Templates have been developed for each flow and are available for developers.

TIPPERS HUB. The TIPPERS Hub (see Figure 41) provides users with a centralized repository of different TIPPERS deployments. It enabled users to discover TIPPERS deployments around them (see Figure 42) and to manage information about their TIPPERS accounts (which can be used to access TIPPERS deployments and applications).

TIPPERS TPORTAL. TPortal provides users of a smart space with access to TIPPERS.

Table 12: Different authorization flows.

| | | |
|--------------------|---|---|
| Authorization Code | Gives persistent access to a TIPPER user's account. (Only one login is necessary). Provides a <i>refresh_token</i> which can be used to retrieve additional <i>access_token</i> without prompting user. Most secure flow. | Requires backend to securely store <i>client_secret</i> . Most complicated flow. |
| Implicit Grant | Does not require a backend. Directly returns <i>access_token</i> . Very simple to implement. | Gives only temporary access to a TIPPER user's account. (Must login every time). Does not provide a <i>refresh_token</i> . Least secure flow. |
| Client Credentials | User interaction is not required. Used for simple backend to TIPPERS API access. | Does not provide any user information. Requires backend to securely store <i>client_secret</i> . |

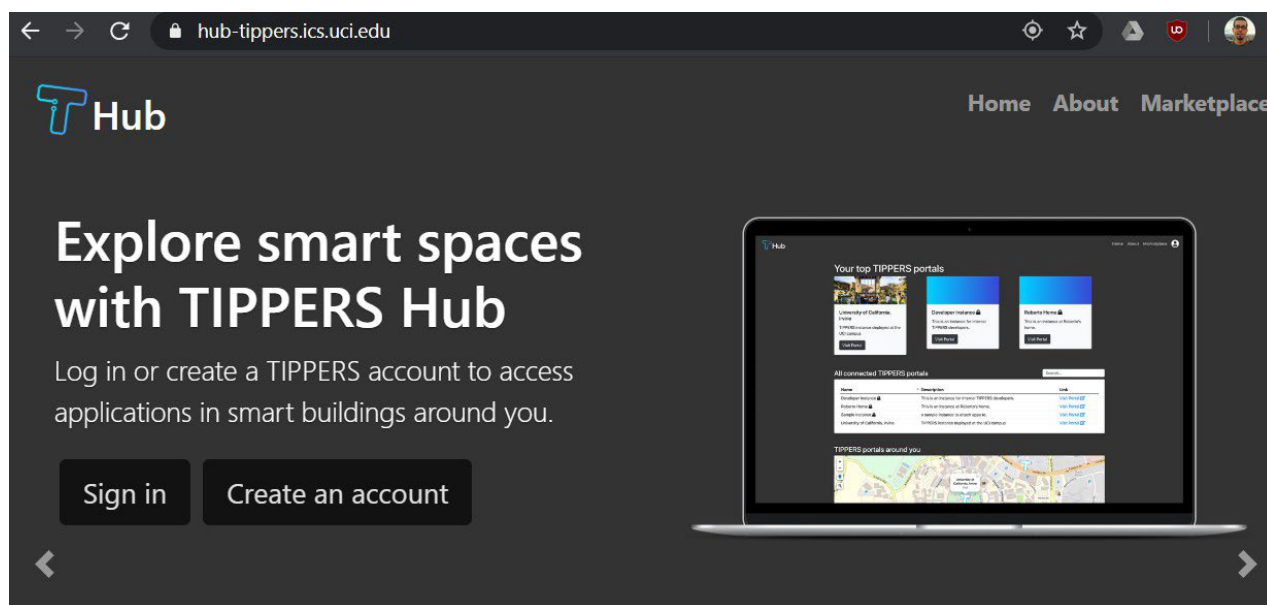


Figure 41: Main GUI of the Hub.

TPortal is a web application which provides an interface to the TIPPERS backend (see Figure 44). It allows users, among others, to: Discover and access the applications available at the specific TIPPERS-enabled smart space (see Figure 45a). Define metadata about the space such as the sensors deployed or information about the devices they own (see Figure 45b). Define privacy policies to control data sharing (e.g., to allow TIPPERS to share the location of the individual with their friends using the Concierge application) (see Figure 45c).

A special component of TPortal is the tool to facilitate the definition of the geospatial element of the smart space (see Figure 46). The TIPPERS model of smart spaces enables

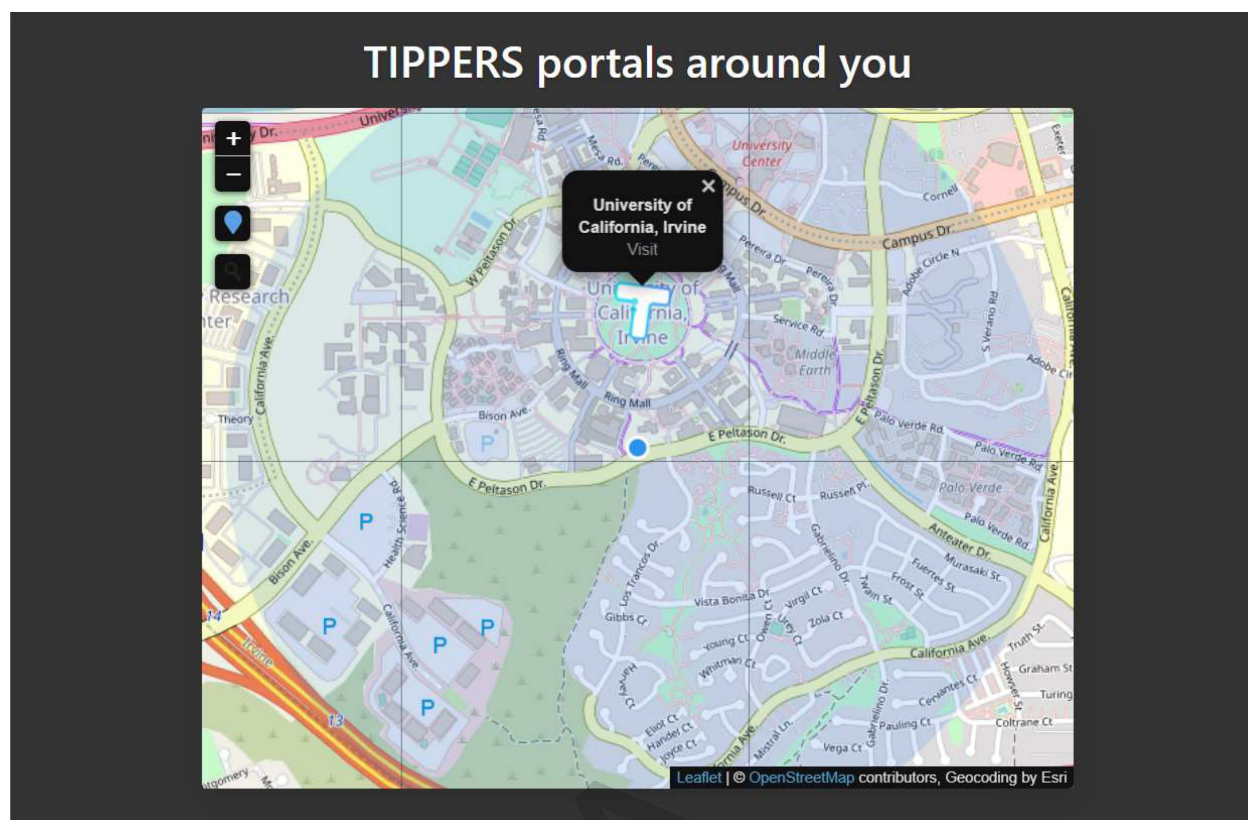


Figure 42: Finding TIPPERS portals in the Hub.

the definition of complex and fine-granularity space/building models. The more detailed the description of the building is, the more the potential of the schema that can be exploited (e.g., to enable TIPPERS to automatically derive the sensors that can be used to obtain high-level information given their placement and coverage). However, manual fine-grained annotation can be challenging and prone to errors, especially of locations as there are multiple elements that have to be defined correctly (e.g., coordinate systems, extents, hierarchical aspect, relationship between elements, etc.). We developed an annotation support tool to facilitate fine-grained annotation/definition of the spatial/-geographic aspect of the smart space. The tool is a web application with a Graphical User Interface (GUI) that enables users to draw locations and fill in a form to specify metadata (see the figure below).

The tool has three definition components: 1) Metadata (e.g., to annotate the type, name, capacity, coordinate system, boundaries of a location); 2) Extent (to draw geometries graphically); 3) Hierarchical representation (which displays the hierarchy defined so far and enables people to define new elements within it). The form in the metadata panel obtains its options from the properties defined in our schema and the possible ranges defined in the knowledge base. For instance, if the specific installation of TIPPERS (e.g., TIPPERS running at UCI) contains four subclasses of location (e.g., campus, building, floor, room), then the form that asks for the type of location being defined will show those. The defini-

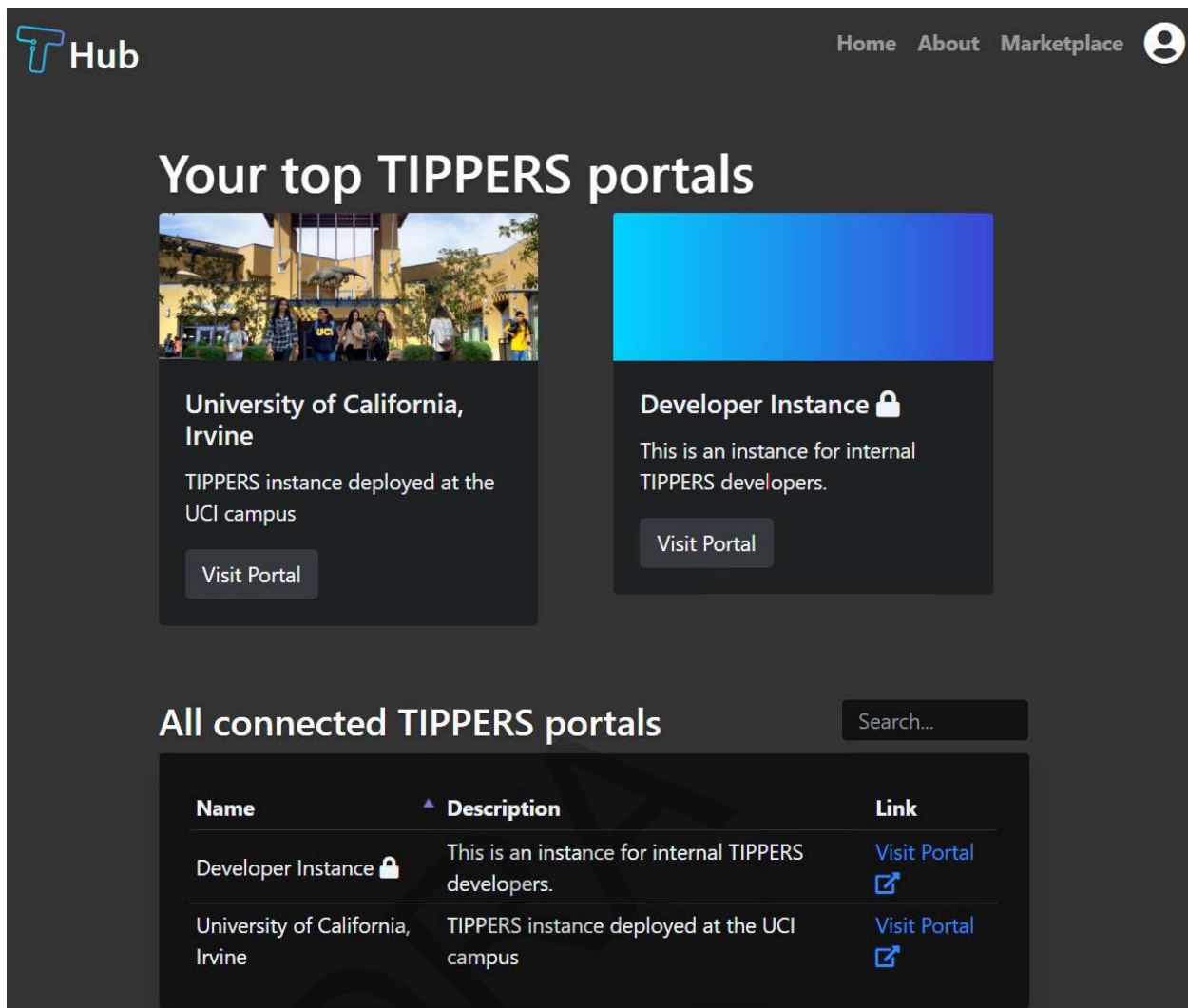


Figure 43: Most visited portals.

tion panel enables the graphical definition of geometries such as circle, rectangle, polygons, etc. Additionally, it can show in the background either a map (in this case OpenStreetMap – <https://www.openstreetmap.org>), so that the geometry defined is automatically associated to its GPS coordinates, or a Cartesian system in which the user can upload their background image (e.g., the floor plan of a floor of a building).

TIPPERS Backend

TIPPERS APIs. TIPPERS includes a set of endpoints and specifications to facilitate the interaction with the system and the development of services for it. Through the endpoints, application developers can obtain data from TIPPERS that the system collected from sensors and abstracted after applying virtual sensors (including those based on usage of PETs).

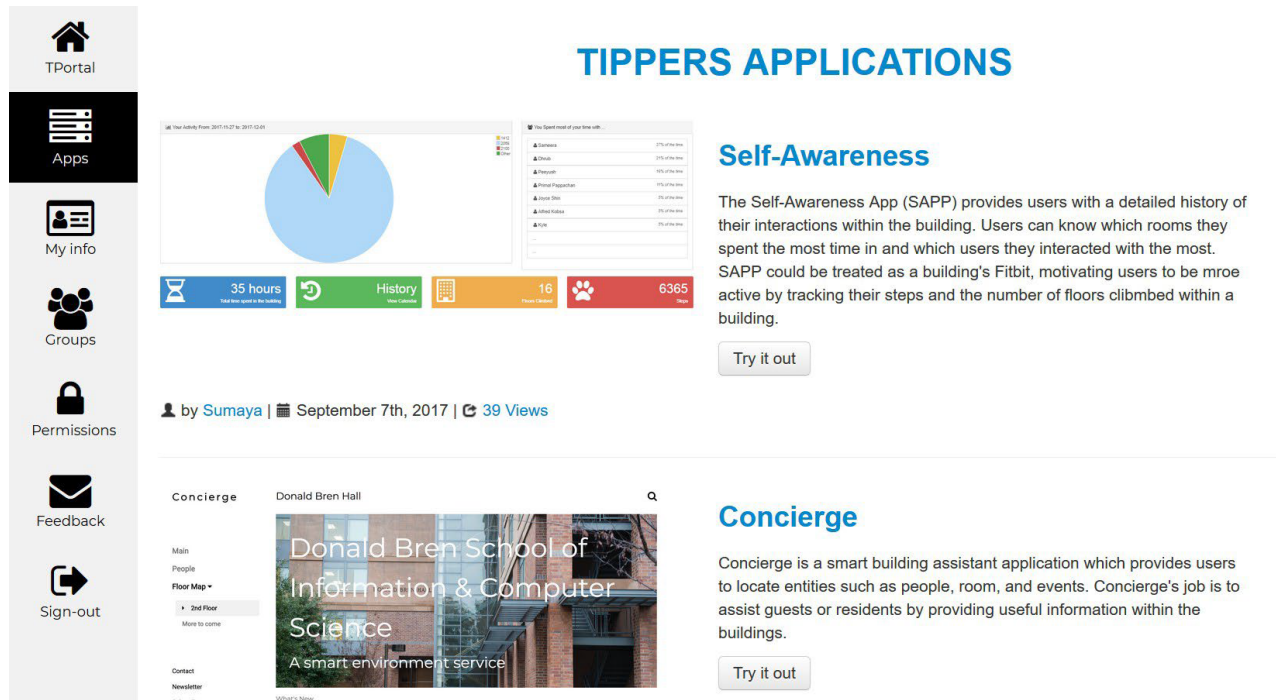


Figure 44: Screenshot of TPortal showing the apps available for users.

Figure 47 shows an example of the Swagger¹³ specification of the TIPPERS APIs.

As an example of the different APIs available, the policy definition API enables users (directly or through the CMU's Privacy Assistant) define their privacy policies. This includes information about what information the user wants to share (e.g., location), with whom (i.e., TIPPERS users and groups), through which service (e.g., Concierge), and when (i.e., time period).

Internally, the TIPPERS APIs translate the requests into SQL queries that are posed against the underlying DBMS. Additionally, one of the APIs enables developers to directly pose SQL queries to the system which is required to develop more complex app functionalities.

As mentioned above, the TIPPERS APIs include an authentication mechanism so that the system has a way to authenticate requests/policies posed by applications on behalf of users. This is an important aspect for real deployments of the system which can include applications developed by third parties and a large user base. To this end, the TIPPERS API code includes an authentication layer based on the OAuth 2.0 delegated authorization framework for REST/ APIs. This is an approach widely used in the industry that assigns tokens (with different validity criteria) to both application developers and users of the system. Those tokens are appended to API calls posed to TIPPERS and used to authenticate the requestor app/user.

Other TIPPERS components. The main purpose of the TIPPERS backend is to enable

¹³<https://swagger.io/>

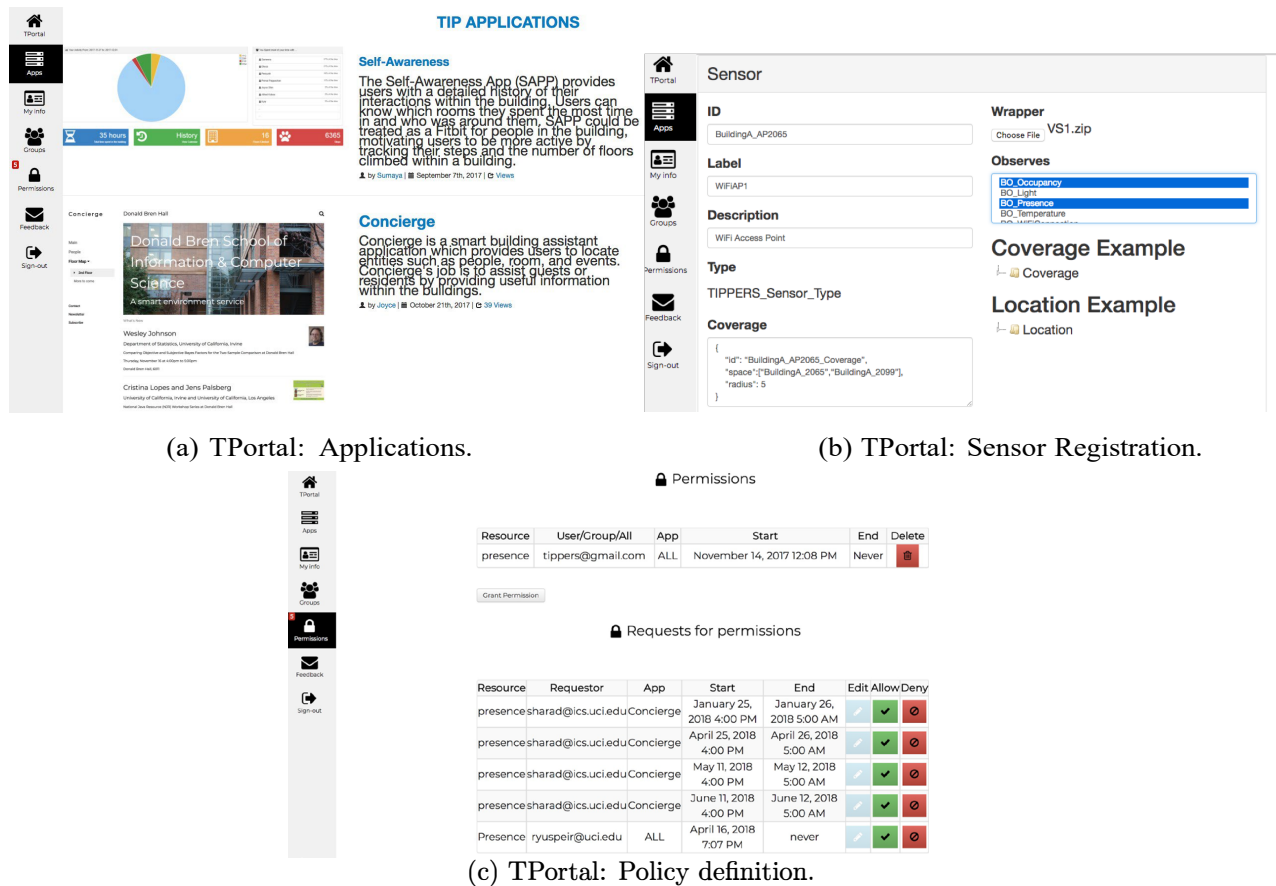
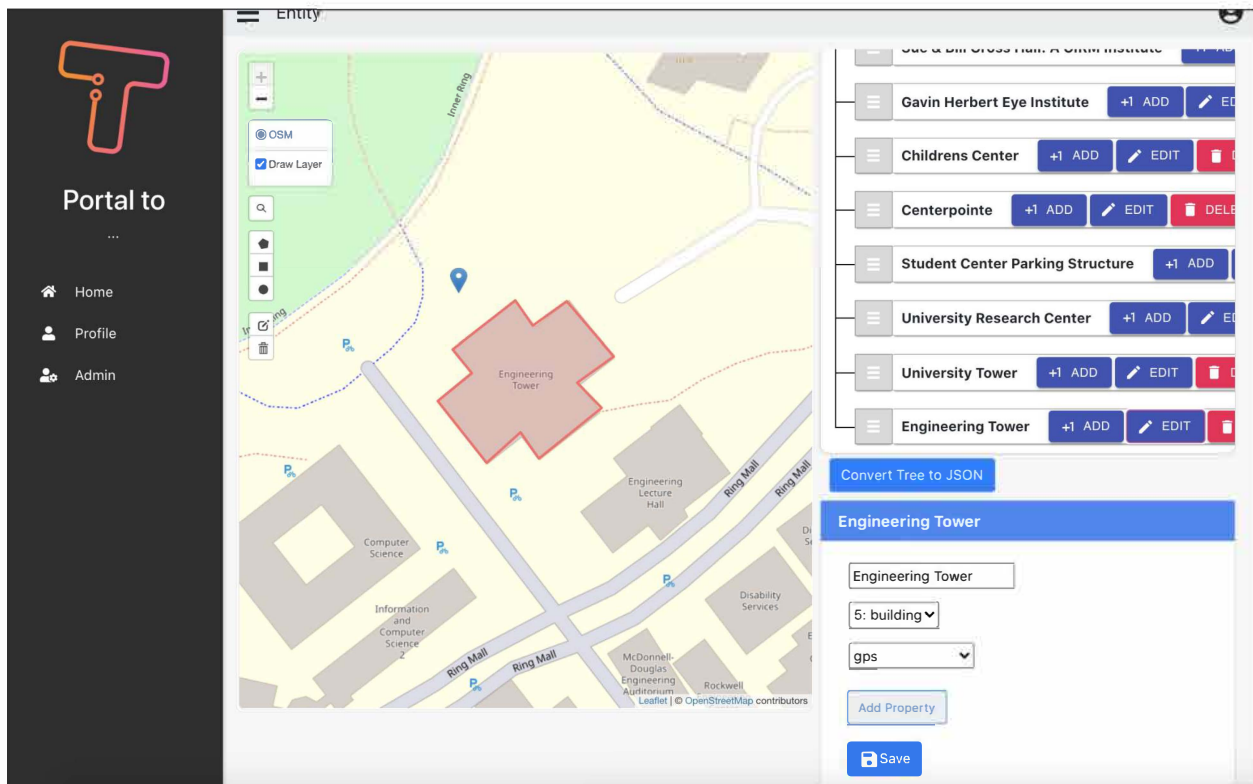


Figure 45: Screenshots of Tportal.

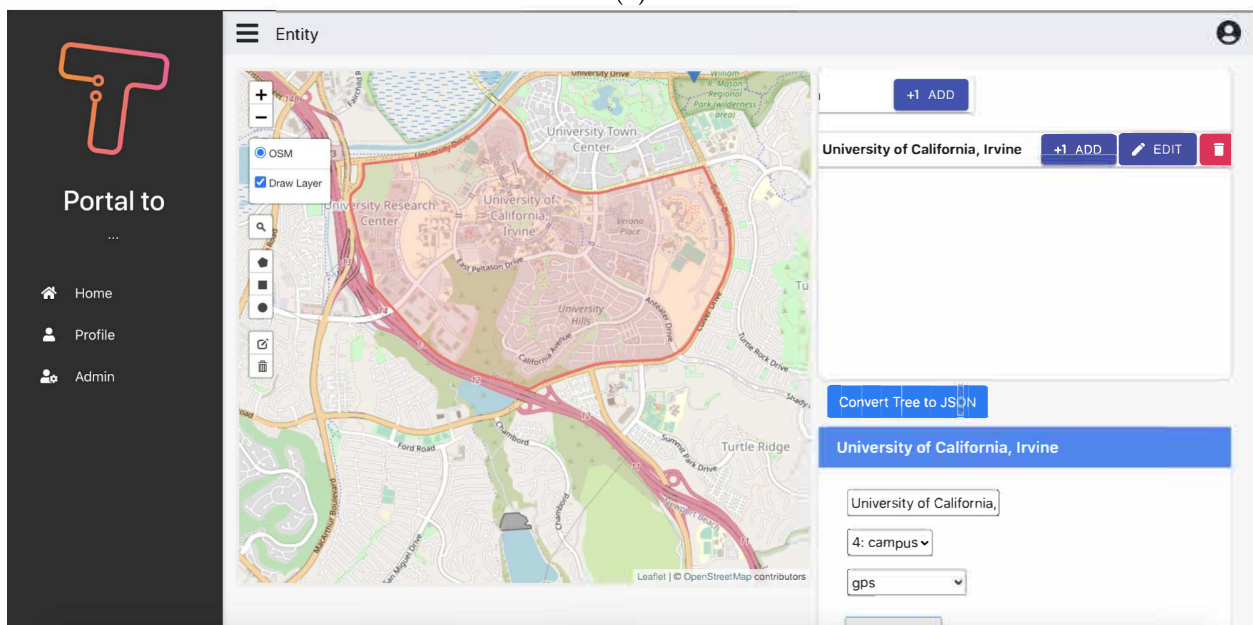
developers and end-users to use a semantic domain-relevant view of smart spaces for building smart applications. This hides the complexity of having to deal with/understand lower-level information generated by specific sensors and actuators. To achieve this, we have designed and implemented several components that manage the storage of data, translation of requests from the high-level to the sensor level, and policy enforcement, among others.

In particular the translation component uses a metamodel that is designed to represent relationships between the low-level IoT devices' world (i.e., devices, observations) and semantic concepts (i.e., users and spaces and their observable attributes). This model is an extension of the popular SOSA/SSN ontology. The request translation mechanism takes a high-level user request or policy (e.g., "what is the occupancy of room 2065?", "do not share my location when I'm in my office") and translates it into the set of devices related with it given the underlying sensor/actuator infrastructure. By relying on the above metamodel, we have introduced a language that enables users to express their action requirements (i.e., requests for sensor data, commands for actuators, and privacy preferences) in terms of user-friendly high-level concepts. To translate user-defined actions into sensor/actuators commands, we rely on an ontology-based algorithmic approach.

The TIPPERS policy engine includes a mechanism that can rewrite queries posed to the system (through the TIPPERS APIs) by taking into account the user preferences. The



(a)



(b)

Figure 46: Screenshots of TMapper.

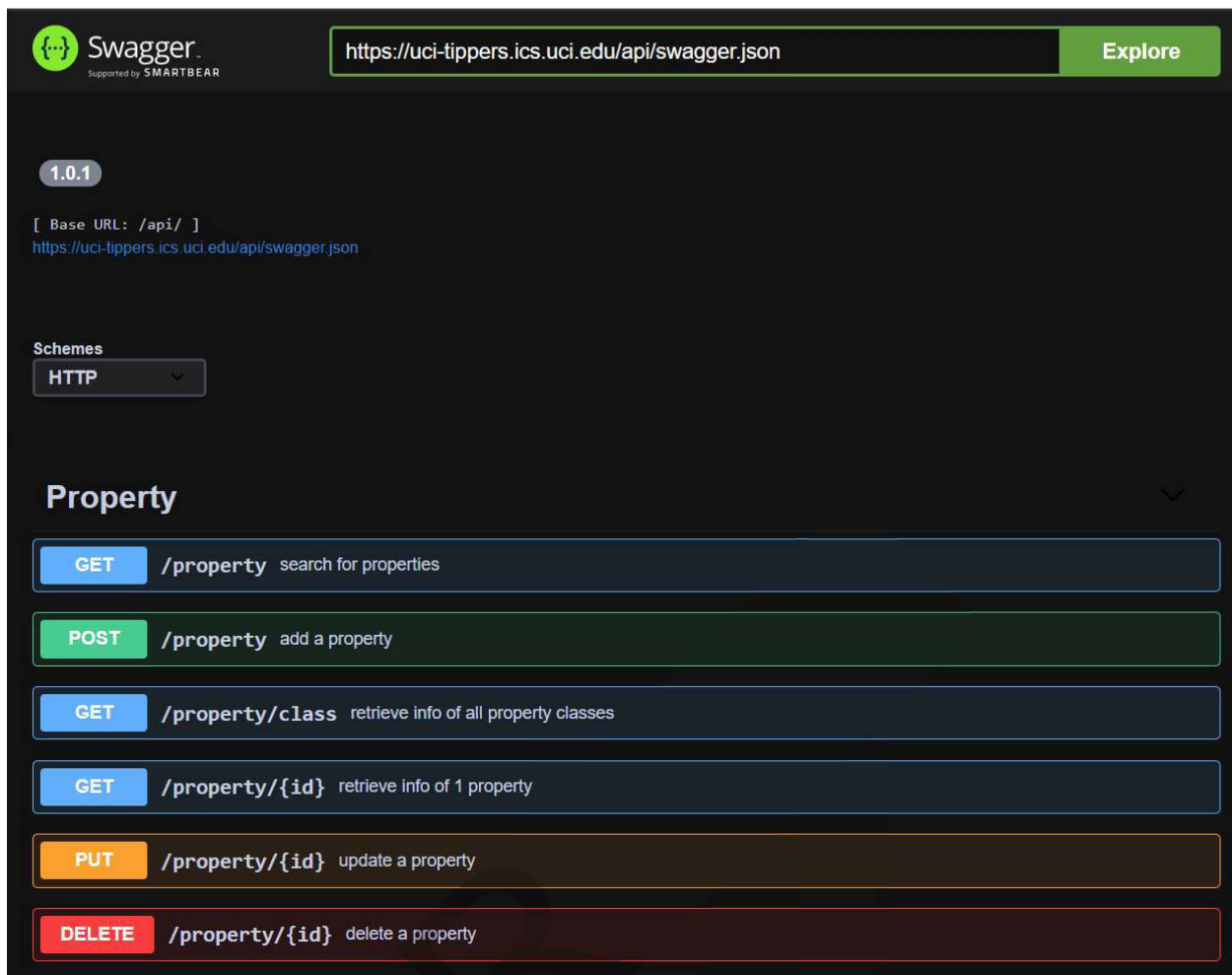


Figure 47: Sample APIs.

policy engine supports the specification of different kinds of policies required by the users of the different apps. These policies are either defined through the TIPPERS front-end or captured through the CMU policy assistant. The engine supports user and application-level policies. We identified 'Groups' (set of users) as an important requirement based on the user demand and added support for specification and enforcement of policies based on user groups.

TIPPERS Smartphone Client. The TIPPERS smartphone client is a software component for Android and iOS devices that connects them to a TIPPERS space. The TIPPERS smartphone client collects data including data produced by the embedded sensors on the device (e.g., accelerometers, gyroscopes) and about devices discovered using bluetooth. Additionally, it provides access to a user interface to configure the discovery or data collection frequency. After devices/sensor data are discovered/collected our technology handles their transmission to the TIPPERS core system in the proper TIPPERS-like data format. The smartphone client also performs some virtual sensing on the device. In particular, it is able

to detect falls given the accelerometer and gyroscope data captured.

Finally, the client transmits data from a smartphone to the appropriate TIPPERS space taking into account periods with lack of connectivity. We refer to this technology as Data Mule. Connectivity between a TIPPERS client and a TIPPERS space is not always guaranteed. Smart spaces are, in general, equipped with network access points (e.g., WiFi APs) to enable the interaction between users and devices. However, there might be parts of the space where connectivity is not available and devices/users cannot interact with each other or collect data from sensors found around this space. This is particularly relevant in the context of the deployment of TIPPERS in US Navy vessels where large areas of the ship did not have connectivity.

In particular, Data Mule maintains a local database on the device in order to store the observations labeled with their status – i.e., pending or sent to TIPPERS. A background service checks periodically for WiFi connectivity in order to push pending data to TIPPERS.

3.3.3 Applications & Other Software

We developed several smart space applications that cover different types of IoT applications from real-time to analytics and from user-centric to group-centric systems.

Concierge

Concierge is a smart-environment service to help visitors and/or building residents to locate entities (rooms, people, events) in the building (see Figure 48). It helps visitors and residents in answering questions like "Where is professor Mehrotra?", "Where are the restrooms", "Where is the CS seminar taking place?", etc. In addition to these functionalities, Concierge also lets users take control over which information would be visible to others.

Concierge provides users with useful information in the building (see figure below):

- Real time location of group/users.
- Current events.
- Available meeting rooms/lounge space.
- Office.
- Context aware messages. With Concierge, a user can create a notification when:
 - A group of people are present in the building.
 - Meeting room becomes available.
 - Lounge space becomes available.

By incorporating policy APIs in TIPPERS and IoTA technology from CMU, users are given access to groups/people only if they are given permission.

Concierge helps identifying privacy challenges and some of them are:

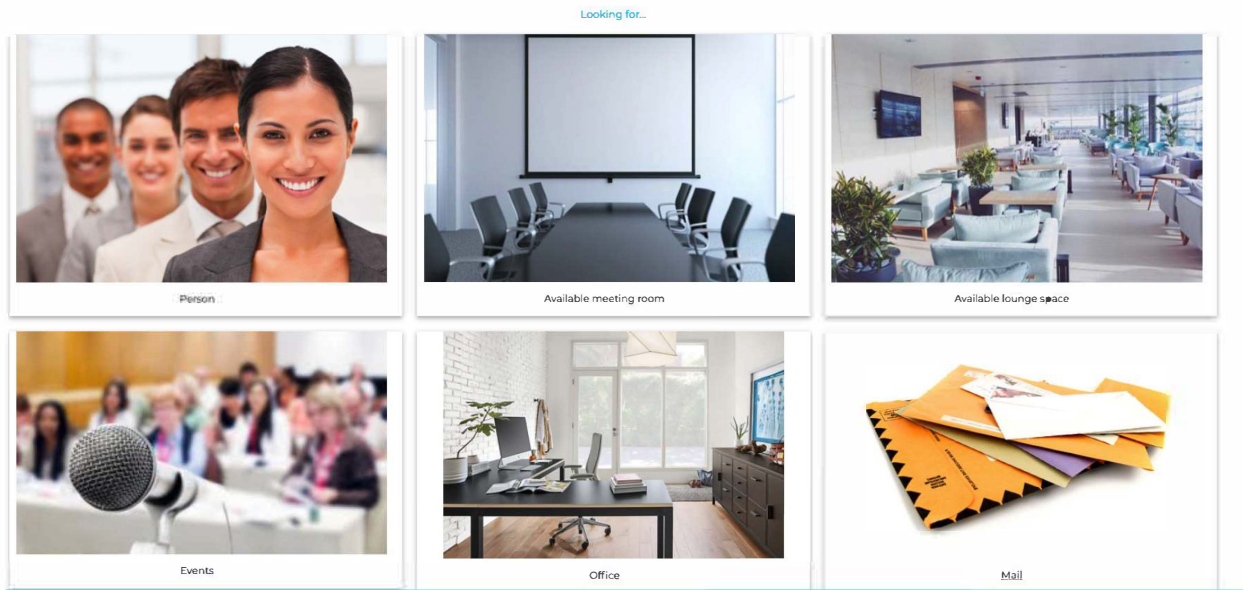


Figure 48: Screenshot of the Concierge app showing the different services offered.

- Who should I share with my location in real time? Can I limit sharing location with a certain group of people, not all the users?
- While showing the occupancy of meeting rooms, I don't want to reveal my presence. Will my occupancy be captured by Concierge?
- Sending context aware messages by tracking my colleague is okay, but the user does not want the server to know that his location is tracked by whom. How can I prevent it?

The context-aware messaging feature of Concierge incorporates complex conditions (see Figure 49). A user can specify under which conditions a message should be delivered given a list of recipients, subject, delivery start and end time, and message body. Along with these conditions, a user can specify availability of rooms with desired capacity or multiple users with location (building, floor, or region) condition. For example, a user wants to be notified when the meeting room 2065 becomes available with minimum capacity 10. Or, she might want to be notified whether my friend Alice is in Donald Bren Hall.

This functionality opens up a set of interesting privacy issues. For example, imagine that a user John does not want to share his location with another user David but shares the information with a third user Mary. If Mary sets up a context-aware message for David in which the condition is that the message should be delivered once John is in a specific room 2065 (e.g., "we can all meet now in room 2065"), this could potentially leak John's location to David. By receiving such a context-aware message and by knowing the message conditions, David could know that John is in room 2065.

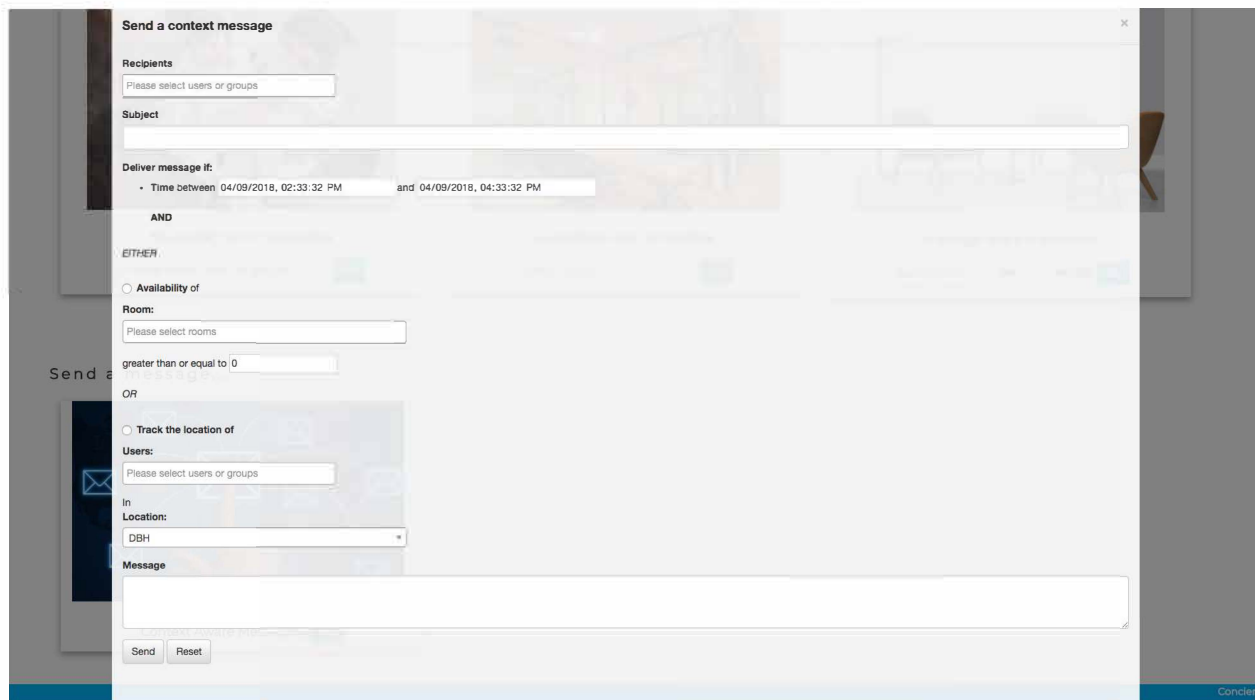


Figure 49: Screenshot of the Concierge app showing sending a context aware message feature.

Self-Awareness app

The Self-Awareness app (SAPP) provides users with a detailed history of their interaction within a building (see Figure 50):

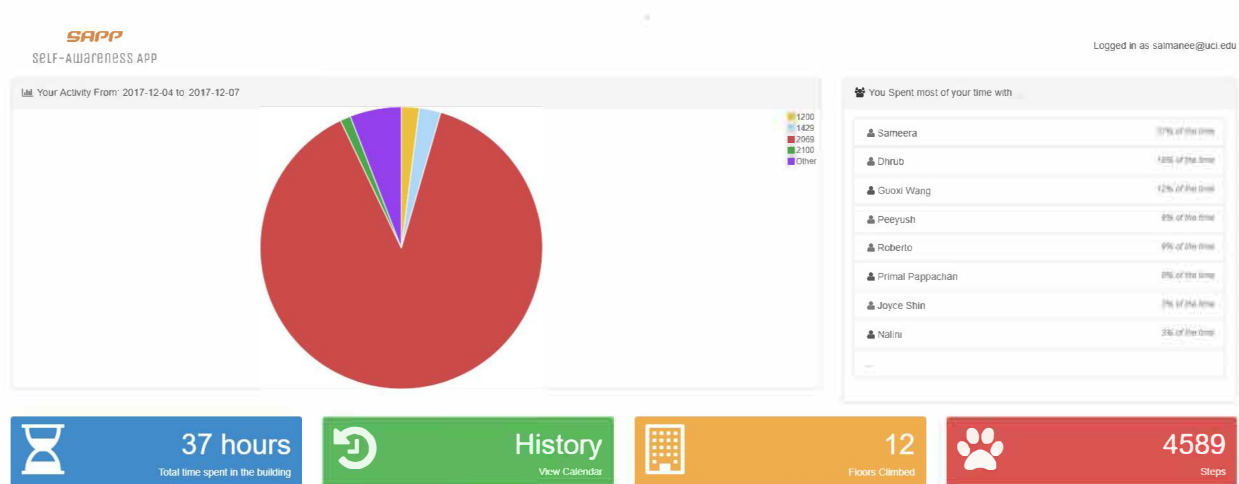


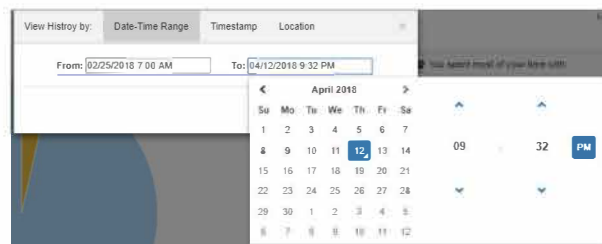
Figure 50: Screenshot of the Self-Awareness App.

- The graph shows the % of time a user spent in a specific room.
- The menu on the right shows a list of people that a user interacted with the most.

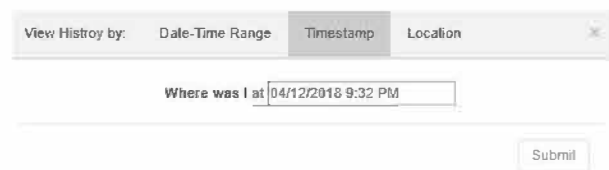
- SAPP has a physical activity tracker within the building which keeps count of the steps walked, floors climbed, and the total time spent within a building
- Users can also view their interaction history by clicking on the History tab and selecting a certain date range. Based on the selected date, the graph in addition to the other tabs will be updated accordingly.

The historical component of SAPP enables users to interact with data by either:

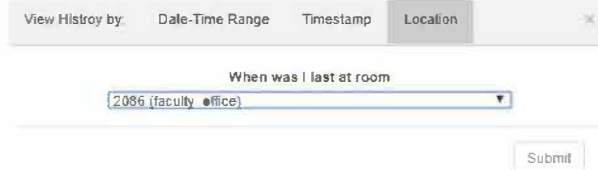
- Selecting a certain date/time range (see Figure 51a). Based on the selected date, the graph in addition to the other tabs will be updated accordingly.
- Selecting a specific timestamp (see Figure 51b). This option is more fine-grained as compared to the previous feature.
- Selecting a specific location (see Figure 51c). A user can view when she was last in a certain room/region.



(a)



(b)



(c)

Figure 51: Screenshots of Self-Awareness app.

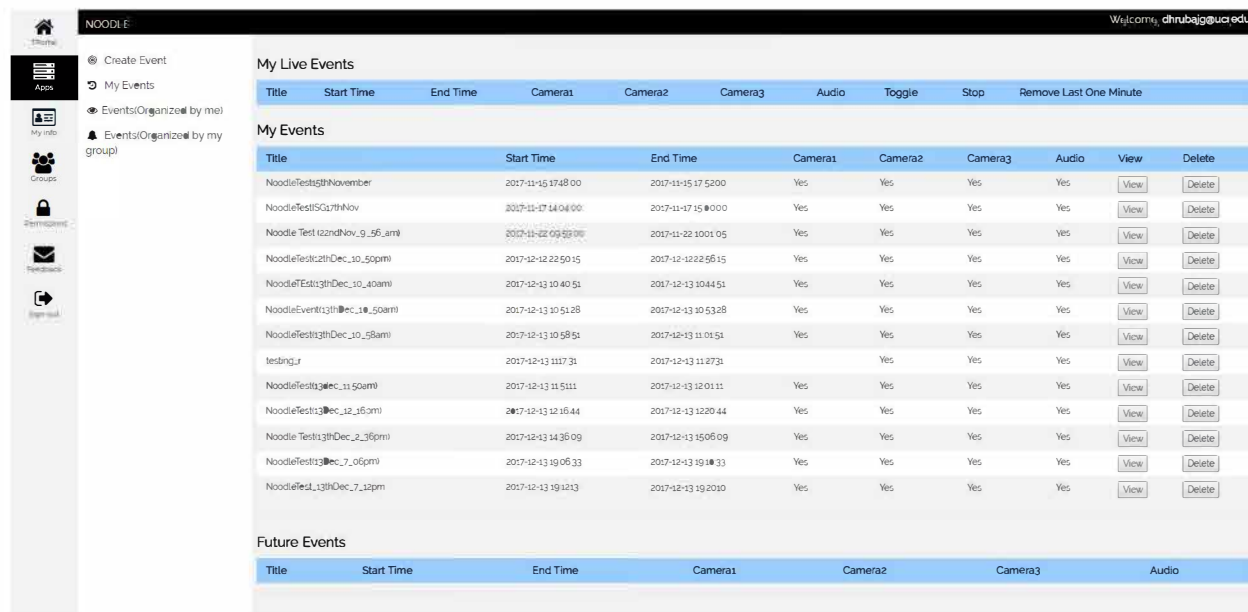
SAPP helps identify real privacy challenges and hence introduces some countermeasures to circumvent them. Some of these privacy concerns are:

- Privacy inferences.
- Users might be concerned about the building capturing fine grained information about their location (how much time they spent in the bathroom, how much time they spent in a friend's office etc.) and who they spent most of their time with, how active they are (based on the number of steps walked, floors climbed etc.).

We plan to extend SAPP by adding more features such as incorporating energy consumption data and adding a social aspect to it. The advantage of adding a social component to SAPP is the fact that it introduces competition among users and makes it easier and more fun to stay active (imagine having reminders to move, or messages stating that you walked or stayed in the building more than x% of the people etc.). On the other hand, the social networking aspect will also introduce more privacy challenges. For instance, if Dhrub grants Roberto the permission to access his location data, who's the owner of the data in this case? Can Roberto share the names of the people that he spent most of his time with in his social network?

Noodle App

Noodle is a smart meeting room application which is capable of policy based data capturing from sensors (e.g., video cameras, microphones) present inside the smart space. This application can be very useful for capturing data about important meetings, events, seminars, conferences inside a building. It can be very beneficial to all types of users inside a building, ranging from residents of the building, visitors, to building administrators, security personnel, etc. In our prototype, and through TIPPERS, Noodle connects to all video sensors at Donald Bren Hall. Users can create an event (see Figure 52 and Figure 53) and specify policies using any cameras installed in the building and the system can capture data from these sensors.



The screenshot shows the Noodle web application interface. On the left is a sidebar with navigation icons for Home, Apps, My info, Groups, and Sign out. The main content area is titled 'NOODLE' and shows a user profile 'WuLcom, dhrubaj@uci.edu'. The 'My Live Events' section has a table with columns: Title, Start Time, End Time, Camera1, Camera2, Camera3, Audio, Toggie, Stop, and Remove Last One Minute. Below this is the 'My Events' section with a table listing various events with columns: Title, Start Time, End Time, Camera1, Camera2, Camera3, Audio, View, and Delete. The 'Future Events' section is partially visible at the bottom.

| Title | Start Time | End Time | Camera1 | Camera2 | Camera3 | Audio | View | Delete |
|---------------------------|---------------------|---------------------|---------|---------|---------|-------|------|--------|
| NoodleTest5@November | 2017-11-15 17:48:00 | 2017-11-15 17:52:00 | Yes | Yes | Yes | Yes | View | Delete |
| NoodleTest5@7thNov | 2017-11-17 14:04:00 | 2017-11-17 15:00:00 | Yes | Yes | Yes | Yes | View | Delete |
| NoodleTest12@ndNov_9_56am | 2017-11-22 09:56:00 | 2017-11-22 10:01:05 | Yes | Yes | Yes | Yes | View | Delete |
| NoodleTest12@Dec_10_50pm | 2017-12-12 22:50:15 | 2017-12-12 22:58:15 | Yes | Yes | Yes | Yes | View | Delete |
| NoodleTest13@Dec_10_40am | 2017-12-13 10:40:51 | 2017-12-13 10:44:51 | Yes | Yes | Yes | Yes | View | Delete |
| NoodleEvent13@Dec_10_50am | 2017-12-13 10:51:28 | 2017-12-13 10:53:28 | Yes | Yes | Yes | Yes | View | Delete |
| NoodleTest13@Dec_10_58am | 2017-12-13 10:58:51 | 2017-12-13 11:03:51 | Yes | Yes | Yes | Yes | View | Delete |
| testing_1 | 2017-12-13 11:17:31 | 2017-12-13 11:27:31 | | Yes | Yes | Yes | View | Delete |
| NoodleTest13@Dec_11_50am | 2017-12-13 11:51:11 | 2017-12-13 12:01:11 | Yes | Yes | Yes | Yes | View | Delete |
| NoodleTest13@Dec_12_16pm | 2017-12-13 12:16:44 | 2017-12-13 12:20:44 | Yes | Yes | Yes | Yes | View | Delete |
| NoodleTest13@Dec_2_16pm | 2017-12-13 14:16:09 | 2017-12-13 15:06:09 | Yes | Yes | Yes | Yes | View | Delete |
| NoodleTest13@Dec_7_06pm | 2017-12-13 19:06:33 | 2017-12-13 19:18:33 | Yes | Yes | Yes | Yes | View | Delete |
| NoodleTest13@Dec_7_12pm | 2017-12-13 19:12:13 | 2017-12-13 19:20:10 | Yes | Yes | Yes | Yes | View | Delete |

Figure 52: Noodle: View Event Interface.

The features of the Noodle Application are as follows:

- A meeting organizer can set up a meeting using the web application and specify the policies of recording during the meeting. The organizer can choose any or all the

Create New Event

Event Name

Start Time: 12/14/2017 10:38:37 AM

End Time: 12/14/2017 11:08:37 AM

Audio Option

☐ Enable Microphone Recording

Video Option

☐ Audience Camera (Left Corner) ☐ Audience Camera (Right Corner) ☐ Presentation Camera

Create Event

Figure 53: Noodle: Create Event Interface.

sensors present in the room for the purpose of data capturing.

- During a meeting, if any participant wants to pause the data capture for some time, then it can be done through the web interface. This functionality can be very useful if any part of the meeting is strictly confidential.
- We have added a functionality in which a participant will be able to delete the last n-minute of recording. The system will be deleting the appropriate segments of the recorded data and store it along with the meeting information. During a meeting, a participant might want to erase the data captured during the last n-minutes due to several reasons. The reason for deleting data of last n-minutes can be the occurrence of some confidential discussion during that time or there might be some breaks during that time and the organizer forgot to pause the recording etc. For this scenario, After the meeting is over, all the participants will be able to view the recorded data through the same web interface.

The Noodle app provides users with a GUI to book our smart meeting room and set up information about the meeting (such as participants, topic, duration) and about what information should be recorded (which cameras should be on and if audio should be recorded). Then, the app generates requests for data capture and sends them to TIPPERS through the Open API. TIPPERS schedules the requests and carry them out when the time comes. Then, the Noodle app queries TIPPERS (again through the Open API) to obtain the recordings

of the different meetings which are finally available to the creator of the event and the participants.

Building Analytics

The Building Analytics App (see Figure 54) provides analytics about data gathered from multiple sensors in the building (e.g. occupancy, temperature, energy and sensor health, etc.).

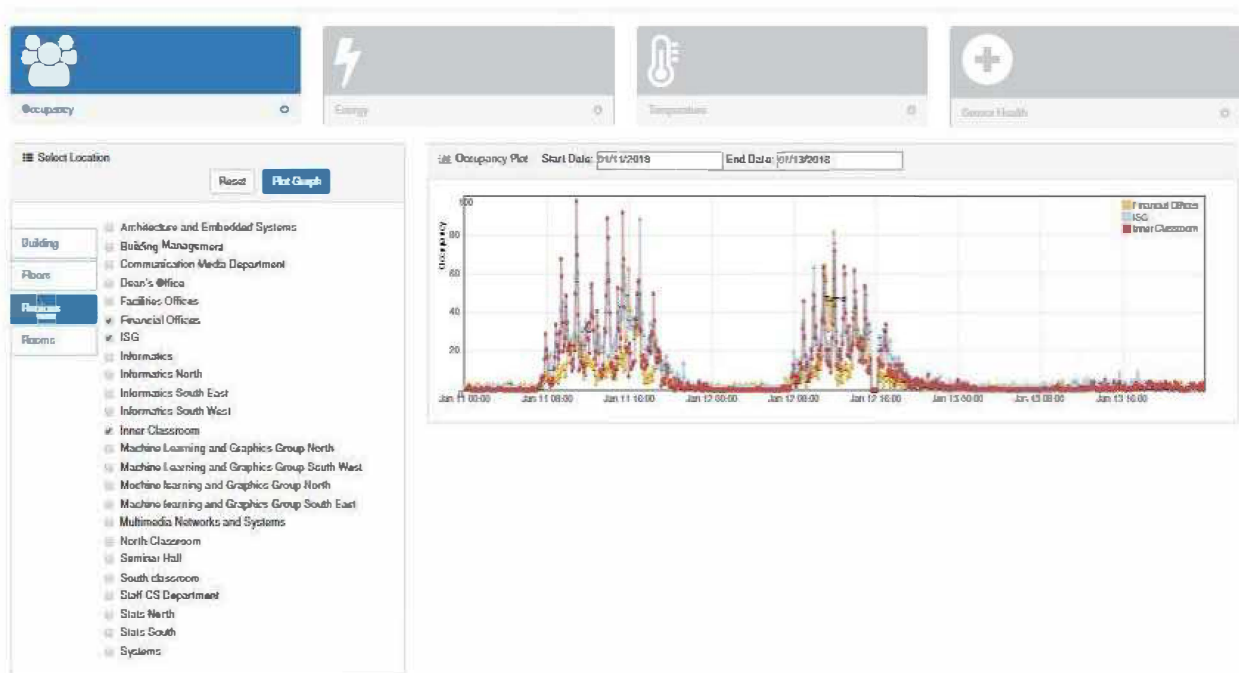


Figure 54: Screenshot of the Building Analytics app.

This application can be used for managing efficient space utilization, devising emergency exit plans, monitoring energy usage, discovering defective sensors etc by the building administrators. At Donald Bren Hall, and through TIPPERs, Building Analytics provides analytics about occupancy data that is gathered from Wifi Access Points and HVAC data. The user can analyze the occupancy data over any specified time and space granularity. The potential users for this Building Analytics App are building managers. The occupancy data can help building managers to do the following:

- Patterns of building usage by occupants for different regions of the building could lead to customized HVAC settings that save energy without inconveniencing occupants.
- Occupancy data can also be used to determine if there are regions in the building that are under/over utilized and such information can lead to plans for better space management (e.g., understanding class rooms that are overflowing or underflowing or determining which lounge spaces are popular).

- Occupancy information in the building can also help devise better emergency exit plans. It can also help notify the building manager about fire code violations i.e. room occupancy is more than the room's capacity.

Although building analytics provide analytics over aggregate data, it can still leak sensitive information about individuals e.g. time spent in the building by individuals, energy consumption by an occupant of an office etc. Building Analytics also retrieves private occupancy data from TIPPERS through the PeGaSus System developed by the DP team.

COVID-19 Monitoring Applications

We also developed several applications focused on helping the community to monitor adherence to COVID-19 regulations. In particular, we developed and deployed at UCI the following applications:

- *Occupancy adherence*: which assesses if the community is indeed implementing occupancy and social distancing advisories suggested by the campus administration. As shown in the figure below (Figure 55 shows the spaces that do not adhere to the social distance criteria in red), it has the ability to monitor when regions such as classrooms, meeting spaces, and shared offices exceed desired occupancy levels, identify locations within the organization where there has been a high rate of people passing by, and hotspots where groups of individuals tend to congregate. Such monitoring allows organizations to conduct an in-depth analysis to reveal potential exposure hotspots, inject interventions (dispatching supervisors who can nudge individuals to relocate) or make remedial operational changes such as scheduling more-frequent sanitization where needed. TIPPERS includes data cleaning algorithms to account for multiple devices carried by individuals (that would result in over-counting), as well as, sporadic and noisy nature of the WiFi connectivity data in computing occupancy.
- *Exposure Tracing*: which enables individuals to share information (without revealing their identity) to empower TIPPERS to determine location (and time) when those individuals were in the region. Exposure tracing helps identify regions at the organization's premises where exposure by others to those infected could have occurred. It also includes a mechanism for individuals to check (again, without revealing identity) if they were overlapped with an infected individual in a region. Exposure tracing is extremely useful to help focus and target contact tracing efforts that are often time-consuming, by rapidly and accurately determining those who potentially may have come in contact with an infected individual.

US Navy Applications

Through our collaboration with SPAWAR, we developed several applications to demonstrate the TIPPERS technology in the context of the US Navy.

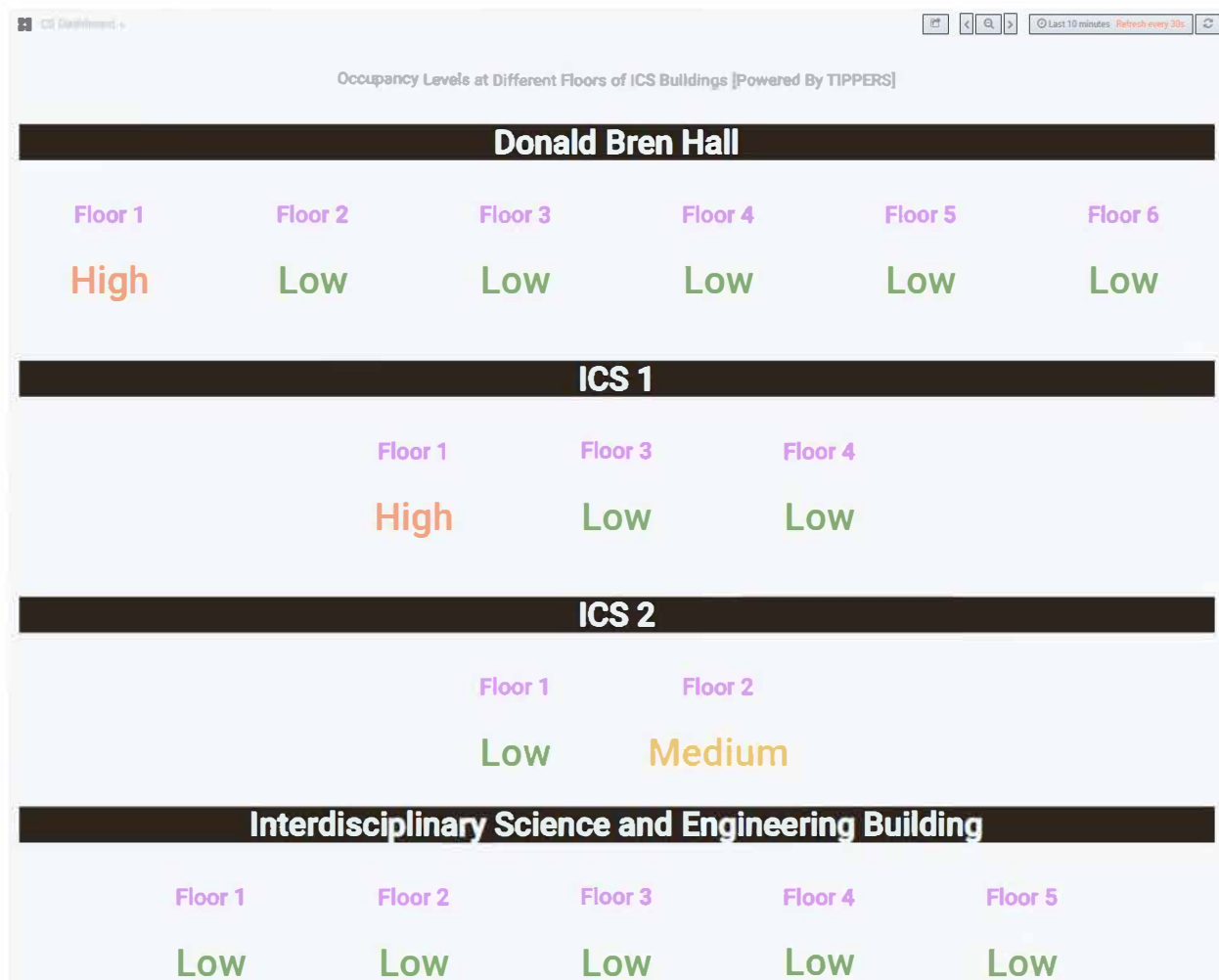


Figure 55: Occupancy adherence app.

First, we developed in collaboration with SPAWAR an application to showcase the technologies in the context of privacy-aware surveillance. The application includes the following main functionalities:

- Ability to show occupancy of different spaces (e.g., buildings) as a heatmap. This data is extracted from TIPPERS using different APIs (depending on the privacy level required – from differential private data with epsilon 0.1 to real count from the Pulsar engine).
- Ability to check whether visitors are inside of any classified room and if so, obtain information about their hosts and their location. This functionality uses the Jana and Pulsar technologies from Galois and Stealth.
- Ability to display information about the privacy vs. utility trade off of the noisy data. The app shows a set of graphs displaying the privacy metric (number of people for

which their location can be guessed at different granularities) and the utility metric (number of color changes in the heatmap given the real occupancy data and the noisy one).

Additionally, we extended our COVID-19 monitoring applications to take into account specific requirements from the US Navy. In particular, the applications developed to demonstrate the potential of the technology to the US Navy are the following:

- *Social Distancing*: The social distancing application (see Figure 56) deployed at UCI was modified to show how much social distancing the sailors are observing on a navy pier.
- *Contact Awareness*: A new application was developed to show all the contacts of a particular person (see Figure 57). The contacts can be filtered based on the duration and distance of the contact. The application also shows the locations the person has visited at different times of the day.
- *Exposure Alert*: This particular application (see Figure 58) is an extension of the contact awareness application and shows the locations visited (along with the time of visit) by infected persons. The visits can be filtered based on the duration of stay.
- *Infection Propagation*: In order to show how the infection might have spread we developed an application that shows multiple levels of contacts of an infected person (see Figure 59).
- *Alerts Dashboard*: This application (see Figure 60) was developed to support a scenario where some parts of the pier are designated as quarantined locations and remaining as safe locations. Infected persons are restricted to move from quarantined locations to safe locations and non-infected persons are restricted to move from safe locations to quarantined locations. Alerts Dashboard displays all such restricted movements.

Simulator of people trajectories

Given the limitation in the number of sensors that we can deploy in the ships used in the context of the Trident Warrior exercise, as well as the number of spaces available, we developed a simulator to generate synthetic trajectories of people within the ship. This simulator tool is also essential to evaluate the TIPPERS system along with the different technologies included (such as secure storage or differential privacy) prior to the deployment of TIPPERS in the ship. The simulator tool, given a description of a scenario, generates as output the trajectories of people as well as the connectivity events that WiFi APs would capture in the scenario. The description of a scenario that the simulator takes as input consists of the following:

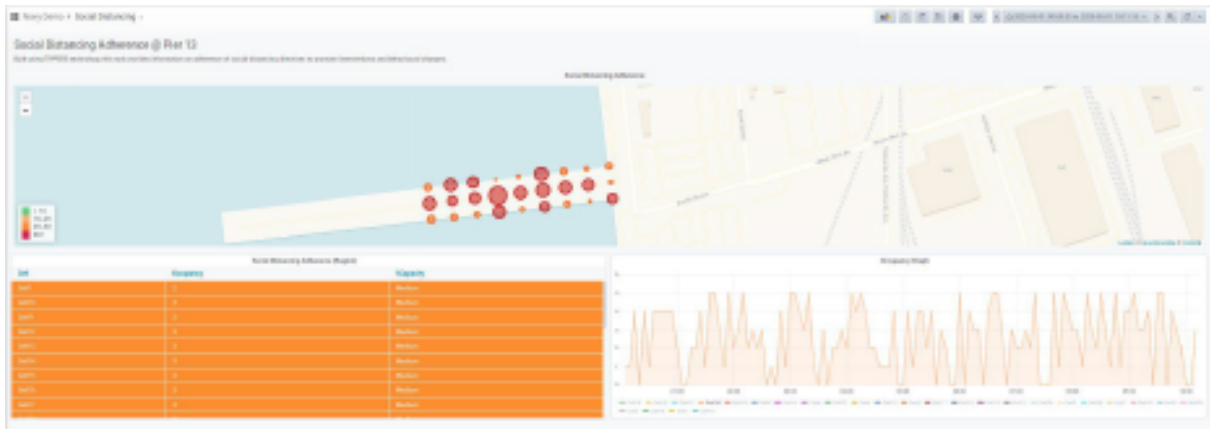


Figure 56: Social distancing app.



Figure 57: Contact awareness app.

- Space definition: which includes the names and types of the different rooms in the space (e.g., a ship) along with a directed graph where the vertices of the graph represent spaces in a scenario, and the edges represent adjacency between spaces.
- Event definition: which describes the events (e.g., cooking service hours) that occur in a space and that are used as drivers of the simulation as people move in the space with the goal of attending events. Notice that some events are labeled periodic, meaning that they will occur on a periodic basis (e.g., every Monday).
- People definition: which includes the template of the different person profiles (e.g.,



Figure 58: Exposure alert app.

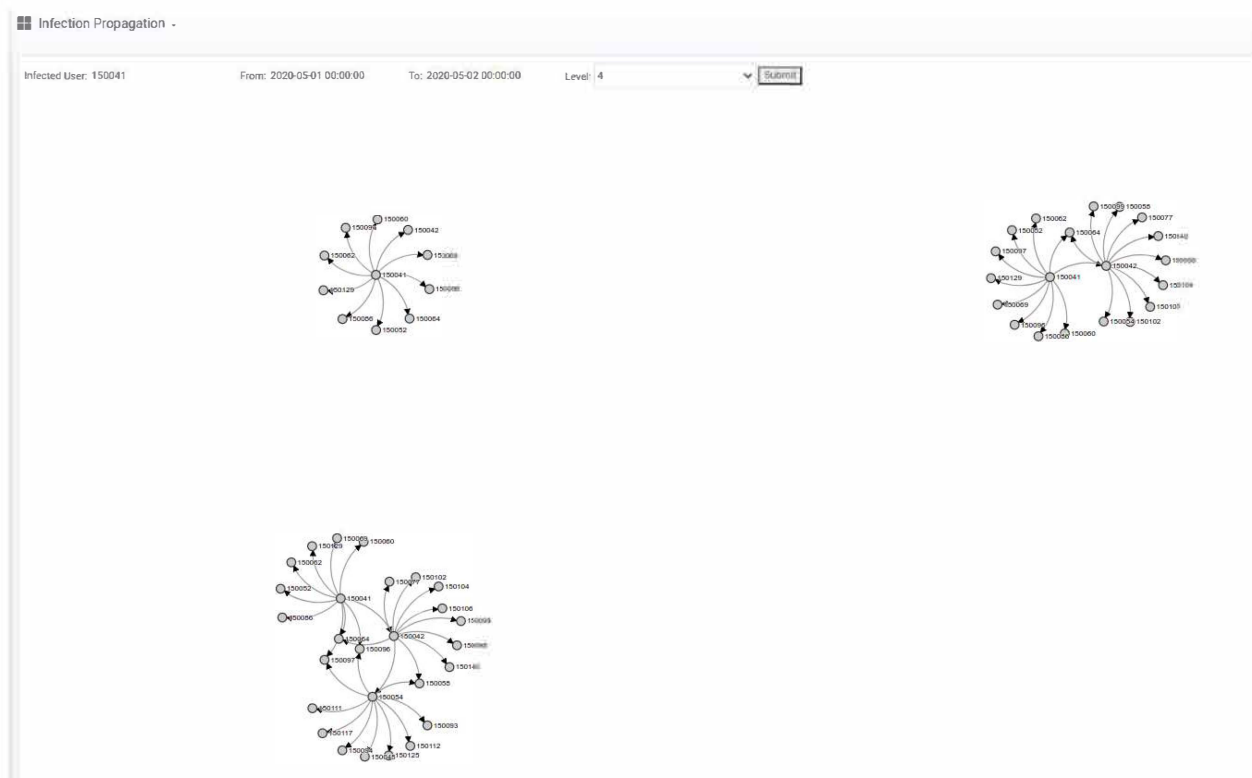


Figure 59: Infection propagation app.

sailors) in the scenario including the events that each profile is likely to attend.

To generate the trajectories of these people, the simulator tool uses the profiles input to randomly generate n people of varying profiles. Then, for each person, it assigns events that they can attend with some probability p . For the events labeled periodic, we have the person attending an event when appropriate; this periodicity allows the simulator to project the patterns that arise in the person's day. To add additional noise into the simulation, the simulator allows people to take leisure breaks when they are not assigned to attend some

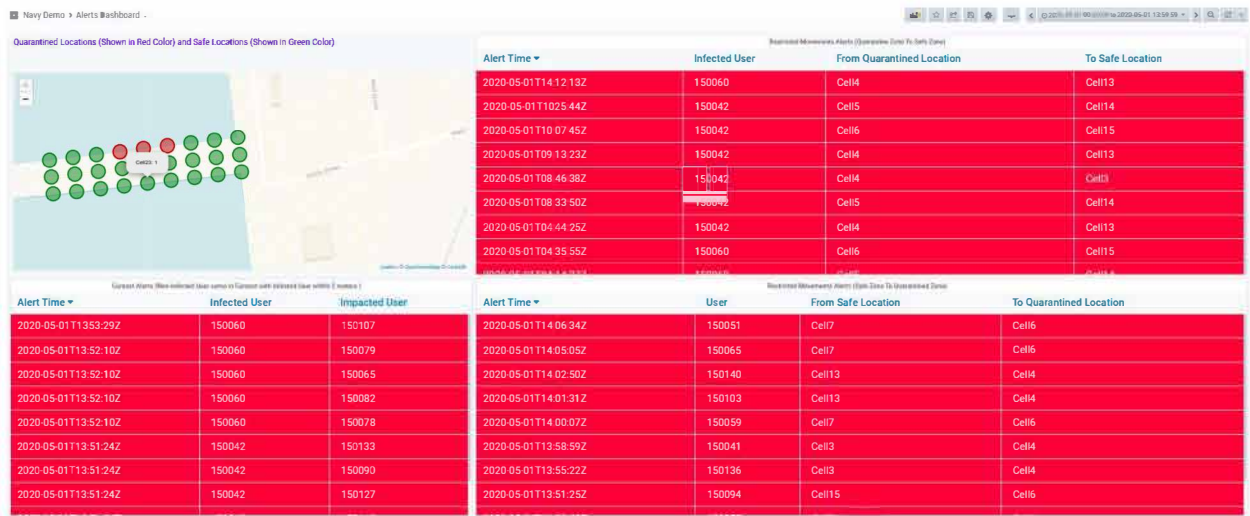


Figure 60: Alerts dashboard app.

event, and take restroom breaks throughout the day.

3.4 Evaluation and Analysis of PETs

In this section, we describe the evaluation of the TIPPERS system and the integrated privacy technologies. We showcase the evaluation performed in three different settings. First, we explain the evaluation of differential privacy techniques in the context of the UC Irvine deployment. Then, we show an analysis of the privacy challenges that arise in workplaces performed in the context of a deployment at Honeywell. Finally, we explain the evaluation of the system and its PETs in the context of the deployment done in a US Navy vessel during the Trident Warrior 2019 exercise.

3.4.1 Evaluation at UC Irvine

In this section, we first describe the development of a tool to analyze the trade-off privacy-utility of the DP technology developed by the UMass/Colgate/Duke team in the context of real analytical IoT tasks. This tool, that we collaboratively developed with the DP team, resulted into a demo paper publication in ACM SIGMOD 18. The goal of the demonstration is to assess the usefulness of private streaming data in a real-world IoT application setting and to explore privacy-utility trade offs of differentially privacy algorithm called PeGaSus which is specifically designed for streaming data to be released under the formal guarantee of differential privacy. Second, we present a study which evaluates the practical implications to individual's privacy, defined as the certainty at which the location of an individual's location at a given point, of publishing a continuous occupancy map generated from streaming sensor data. The study considers an experimental evaluation of diverse privacy enhancing technologies (PETs) to publish occupancy data derived from continuous sensor streams in emerging smart buildings. In particular, we study the performance of techniques that are derivative of differential privacy vs. the de-linking technique used when sharing connectivity data in our deployment.

3.4.1.1 Testing Differential Privacy in Window Analytical Tasks

Emerging Internet of Things (IoT) technologies [94, 95] promise to revolutionize domains like health, transportation, smart buildings, smart infrastructure, and emergency response. IoT has the potential to connect a large number of commodity devices (e.g., sensors, actuators, controllers) into an integrated network that can empower systems with new capabilities and bring transformational improvements to existing systems [95, 96]. In IoT systems, sensors are used for fine-grained monitoring of the evolving state of the infrastructure and the environment. Our interest is in user-centric IoT spaces (as per the IEEE P2413 standard [97]) wherein sensors of diverse types (e.g., cameras, cell phones, WiFi access points (APs), beacons) are used to create awareness about subjects/end-users, their interactions with one another, and with the space.

While fine-grained continuous monitoring offers numerous benefits, it raises several privacy concerns [98, 99, 100]. To appreciate such concerns, consider smart buildings, such as smart office spaces and/or smart retail spaces, that track individuals' location and activity

to provide customized experience based on user's context. Such services could include customized HVAC control based on user's preference, help locating nearby resources, and/or customized coupons/incentives in a retail setting. Fine-grained monitoring, besides enabling customized services, also raises significant concerns about the data collector being able to use the data captured to infer properties such as religious beliefs, gender, personal habits of individuals (e.g., smoker/non-smoker), among others, which individuals may not be comfortable sharing without explicit consent. Our own experience in developing TIPPERS shows that low-level sensor data captured by WiFi APs, motion/light sensors can allow for inferences about individuals, their locations, and their work habits.

We assume that the IoT infrastructure is trusted but that privacy violations for monitored individuals may result from the release of collected data through the many applications envisioned for IoT. The privacy literature has shown that serious disclosures can result even when data is anonymized or the released data consists of aggregate statistics about groups of individuals. We use differential privacy [101], with appropriate privacy parameters, to offer protection to individuals whenever data is released beyond the trust boundary of the IoT system.

Our goal in developing this tool is to explore privacy-utility trade offs offered by methods such as PeGaSus [102] (which provides differential privacy guarantees over streaming data) in supporting real-world applications for everyday use in a real IoT testbed. TIPPERS uses diverse sensor data to generate a dynamic state of the building and its occupants—in particular, sensors such as WiFi APs, video cameras, and bluetooth beacons are used to determine the location of individuals in the 6-story Computer Science building as a function of time. Such location data is used, in turn, to create a variety of applications (described briefly later) used by building occupants and visitors PeGaSus, and other privacy mechanisms which introduce noise into released aggregates, are currently being integrated into TIPPERS to offer rigorous privacy protections.

We focus on the *Building Analytics* application built into the TIPPERS testbed. The application offers end-users an ability to monitor occupancy levels at various granularities (e.g., room, floor, region) and types (e.g., faculty offices, student spaces, conference rooms, meeting rooms, lounge spaces). Historical data can be analyzed at various temporal granularities (minutes, hours, days). Motivated by the tasks for which the *Building Analytics* application is typically used, we have created a game, IOT-DETECTIVE, in which a player is asked to perform one (or more) interactive analytics tasks using a visual analytics tool based on private streams. These include identifying high-occupancy regions, finding unresponsive sensors, or counting the number of times occupancy exceeded a threshold. As part of the game, users are offered differential private views of the data and are rewarded for both their accuracy and timeliness in finishing the task.

The IOT-DETECTIVE Game

The game involves a player who is challenged to identify a real world event or pattern using tools provided by TIPPERS on the differentially private data, much like a building manager might in a real-world deployment. The purpose of the game is two-fold: 1) To illustrate

the privacy-utility tradeoffs in the differentially private data generated by PeGaSus in a way that engages SIGMOD attendees; 2) User-test this tool for a future study of whether users can use differentially private data for IoT analytics.

To play the game, the demo participant interacts with the IoT-DETECTIVE game (see Figure 61), which is very similar to the Building Analytics app, but has some additional game-specific features, such as a timer, leader board, etc. The game is played in rounds and a player can play as many rounds as possible in the allotted time. In each round, the player is given a specific task which requires answering a factual question about types of events during certain time periods (e.g., to identify the most likely time a weekly meeting occurs). The player can then use the app to navigate through the data to identify the relevant (differentially private) data streams and temporal windows and derive an estimate for the answer. The accuracy of the answer is measured in terms of the difference between the player's estimate and the correct answer on the true (non-differentially private) data. Players will be rewarded with points after accurately accomplishing each task. The amount of points will depend on a combination of the accuracy of their estimate, the time taken to complete the task, and the number of tasks they have completed (to incentivize participants to play more than one round). The demo will track player points and maintain a leader board to encourage friendly competition.

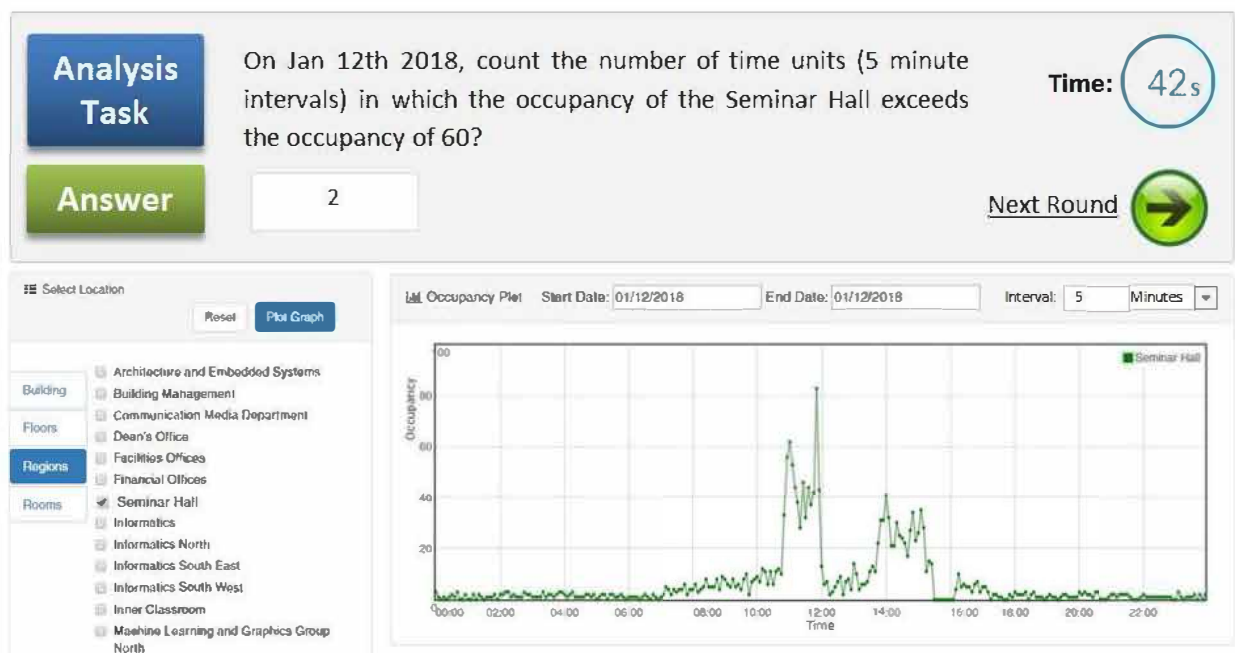


Figure 61: Screenshot of the IoT-DETECTIVE game interface.

The accuracy of a player's answer depends on two primary factors. First, it depends on the player's ability to successfully navigate the user interface—thus, the demo is serving as a valuable user test to see if the tool is intuitive and effective for these analytics tasks. Second, it depends on the amount of noise injected into the data stream by PeGaSus. By varying the privacy parameters across users and rounds, we can gather some preliminary data on how

much noise is tolerable for varying tasks—thus exploring the practical viability of differential privacy in streaming data settings.

Example Tasks

The Building Analytics Game app will be initialized with a differentially private dataset that reports occupancy information at 5 minute intervals for each room in the building.

An example of a task might be: “On [specific date], count the number of time units (5 minute intervals) in which the occupancy of the [main conference room] exceeds [60].” The parts in brackets can be varied to generate different versions of this task. The motivation for this task is that building managers may wish to detect when a room exceeds its maximum permitted occupancy under fire code regulations, or identify rooms/times in which space is heavily-utilized. Players will be asked to perform a variety of tasks. The following are additional illustrative examples:

- *High occupancy regions.* The rooms can be naturally organized into a fixed set of regions e.g., Facilities Offices, Department of Informatics, etc. This task is to identify which region is the most occupied at night (6pm to 6am) on [a particular day]. Most occupied could mean average number of people are highest during night time. The motivation for this task is better HVAC control at late hours when there are fewer occupants in the building. The accuracy measure can be the difference in rank between the user’s choice and the true answer.
- *“Broken” sensors.* We presume here that when a sensor breaks, it no longer senses its environment and continuously reports a constant value, such as zero. Thus, we formulate the task as follows: identify the earliest point in time in which [a particular sensor] starts continuously reporting zero. This is motivated by the practical challenges that building managers face with equipment maintenance. The accuracy measure is distance to the actual time the sensor breaks (we will artificially modify the dataset to make a sensor appear broken).
- *Occupancy at routine events.* The task is to identify the start time of a regularly occurring event in a particular room e.g. start time of a lecture in a classroom. The motivation is to facilitate better scheduling or detecting events that deviate from a schedule. The accuracy measure is the distance between the player’s estimate and the actual start time of the event.

Post-game empirical evaluation

IOT-DETECTIVE records traces of the games of all participants. In addition to providing immediate feedback to users on their success, we intend to analyze the complete trace to better understand the impact of the privacy mechanism on the usefulness of visually displayed stream data. The trace of game play will allow us to answer questions such as: *For what setting of the privacy loss parameter (ϵ) is task success negligibly impacted? For what settings*

of ϵ does task success break down? To what extent does task success vary across the user population (e.g. due to differences in skill or attention)? How do the above vary across different tasks? (e.g. are some tasks more tolerant of distortion in the data or of poorly skilled players?) Is there systematic bias in task answers that results from the perturbed data? Each one of these factors is crucial to a successful integration of PeGaSuS in a real IoT system, and will provide insight into feasible privacy settings and improvements to privacy mechanisms.

3.4.1.2 Privacy vs Utility Analysis of Connectivity Data

In this study, we are targeting systems where location of individuals is collected, but only occupancy data is available through the queries. For this class of systems, we want to evaluate the privacy implications of a given technique following the methodology summarized in Figure 62. Notice that the methodology is general and can be used in other contexts too. Therefore, along with the explanation of each component we explain how they have been instantiated in our scenario.

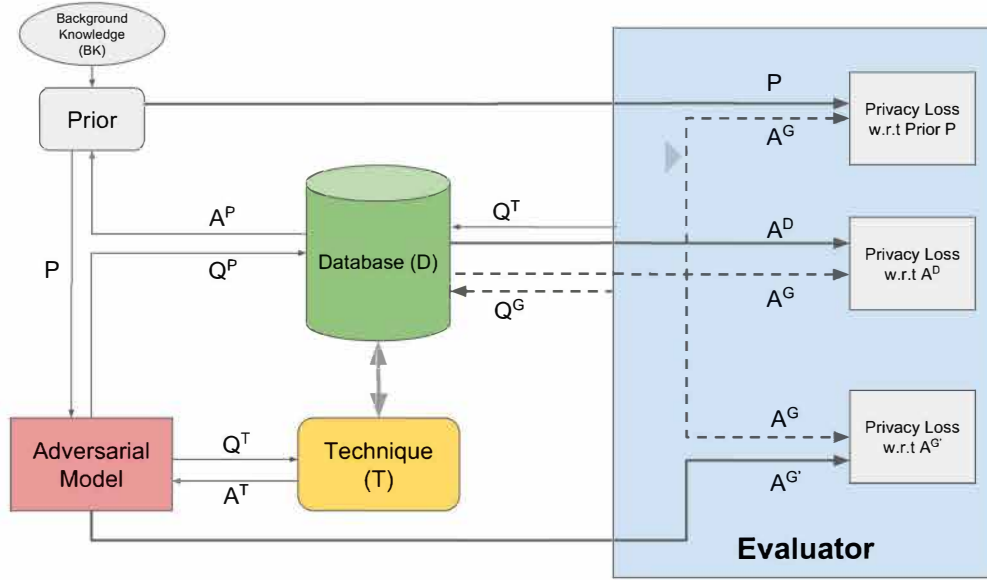


Figure 62: Methodology defined.

We define a database D that contains all the data captured by the smart building. The data, as well as its schema, is fixed. We define \mathcal{T} to be the set of Privacy Enhancing Techniques utilized for answering the adversary's allowed queries. Thus, we consider that a technique $T \in \mathcal{T}$ answers the query Q^T as A^T in a privacy preserving manner after consulting D .

Adversarial Model. As an adversary, we consider a tuple of algorithms whose goal is to retrieve/guess information about individuals in the dataset. We consider that such information cannot be directly accessed by the adversary by posing queries on the dataset. Also,

we consider that the adversary can pose some queries on the dataset. More formally, we define such tuple of algorithms for: 1) Creating the prior P by asking questions Q^P , that the adversary can pose on D , in addition to his/her background knowledge, for prior creation. 2) Computing the guess $A^{G'}$ to the question he/she wants to retrieve/guess from D by posing questions Q^T to the technique and using answer to Q^T (i.e., A^T) and the prior P . The goal A^G that the adversary wants to retrieve/guess from D that can be retrieved by posing questions Q^G may be specified as a set of SQL queries against the database D . Notice that Q^G cannot be posed on D by adversary.

In the context of our study, we consider that the adversarial prior P is created by observing past occupancy as well as presence data Q^P . Also, the adversary aims at guessing (A^G) the location of individuals at a time t . The adversary may have access to some background knowledge about some users location at time t . The adversary is allowed to pose queries (Q^T) to obtain occupancy counts (A^T) through technique T at the given time t .

Privacy Metric. As the goal is to determine the practical privacy implications of technique T , we define a measure based on privacy loss. This is defined as the number of people for which the adversary can guess the correct location at time t with a specific confidence class (in our experiments we defined six levels of confidence classes). First, we compute the adversary's guess based on prior P . This measure indicates how good is the adversary's guess of A^G based on just the prior. Then, we computed the adversary's guess A^D based on answers to Q^T without using any technique. This measure indicates how good is the adversary's guess of A^G if the adversary was allowed to run query Q^T over database D without any privacy preserving mechanism T . Finally, we compute the adversary's guess $A^{G'}$ based on prior P and answers to Q^T using technique T . This measure indicates how good is the adversary's guess of A^G based on the prior (P) and answers to queries Q^T using privacy preserving mechanism T .

Next, we present our method to compute posterior for an adversary with and without privacy mechanisms.

Posterior without Privacy Techniques

Let \mathcal{U} be the set of users, \mathcal{L} the set of locations, and \mathcal{T} the set of observed timepoints. For all $u \in \mathcal{U}$, $l \in \mathcal{L}$, $t \in \mathcal{T}$, let p_{ul}^t denote the prior probability of the user u being in the location l at the timepoint t . Now fix a particular timepoint t . Assume that, for each location l , the attacker has obtained the total count c_l^t of users that are in location l at time t . We want to see how the probability p_{ul}^t changes after the attacker observes the counts $(c_\ell^t)_{\ell \in \mathcal{L}}$. Let C_ℓ^t be the random variable corresponding to occupancy of room ℓ at time t . Define an event $C := \bigwedge_{\ell \in \mathcal{L}} (C_\ell^t = c_\ell^t)$. Let the notation " $u \in l@t$ " denote the event "the user u was in location l at time t ". Using Bayesian inference, we get

$$\Pr[u \in l@t \mid C] = \frac{1}{1 + \frac{\sum_{l' \neq l \in \mathcal{L}} \Pr[u \in l'@t \wedge C]}{\Pr[u \in l@t \wedge C]}} \quad (32)$$

To compute (32), we need to estimate $\Pr[u \in l@t \wedge C]$ for all $l \in \mathcal{L}$. This can be done as follows. The user u is assigned the location l . The remaining $|\mathcal{U}| - 1$ users need to be

distributed to locations in such a way that the resulting occupancies would satisfy C . That is, the users are partitioned into $|\mathcal{L}|$ disjoint sets S_ℓ ($\ell \in \mathcal{L}$), where $|S_l| = c_l^t - 1$, and $|S_\ell| = c_\ell^t$ for $\ell \neq l$. The quantity $\Pr[u \in l@t \wedge C]$ can be computed by summing up the probabilities of all possible partitionings of users to such sets, which gives us

$$\Pr[u \in l@t \wedge C] = p_{ul}^t \cdot \sum_{\substack{(|S_\ell|=c_\ell^t)_{\ell \neq l}, |S_l|=c_l^t-1 \\ \forall k, \ell: S_k \cap S_\ell = \emptyset, \forall \ell: u \notin S_\ell}} \prod_{\ell \in \mathcal{L}} \prod_{v \in S_\ell} p_{v\ell}^t .$$

We could directly compute the sum over all possible partitionings, but it would be computationally too expensive. We need to make some additional assumptions to simplify the computation.

Non-individualized Distributions

Let us assume that we have $p_{ul}^t = p_{vl}^t$ for all users $u, v \in \mathcal{U}$, i.e., the distribution of locations does not depend on the individuality of a particular user. This is reasonable in the case where the potential attacker only knows how an average user behaves in general, but does not distinguish between them, e.g., the attacker learned this by a prior observation of occupancy counts for a certain period of time. Let n_u^t be the total number of users recorded at time t . The rooms are occupied according to multinomial distribution: if we order the users and assume that the first c_1^t go to S_1 , the next c_2^t got to S_2 , etc, there are $(n_u^t - 1)!$ possibilities to rearrange the users, and since the ordering inside S_j does not matter, we get $\frac{(n_u^t - 1)!}{c_1^t! \dots (c_l^t - 1)! \dots c_{n_l}^t!}$ possible partitionings for $n_l = |\mathcal{L}|$ rooms. We get

$$\Pr[u \in l@t \wedge C] = c_l^t \cdot \frac{(n_u^t - 1)!}{c_1^t! \dots c_{n_l}^t!} \prod_{\ell \in \mathcal{L}} (p_\ell^t)^{c_\ell^t} , \quad (33)$$

which gives us the posterior probability

$$\Pr[u \in l@t \mid C] = \frac{1}{1 + \frac{\sum_{l \neq \ell \in \mathcal{L}} c_\ell^t}{c_l^t}} = \frac{c_l^t}{n_u^t} . \quad (34)$$

This is quite an intuitive result, since if all users are treated equally, then any user will most likely be located in the most popular place, even if the prior probability of being there is very small. In fact, the posterior probability does not depend on the prior probability at all, but only on the counts, as the prior probability only defines the distribution of room occupancy, which is overridden by actual counts.

Increasing Attacker's Knowledge

If the attacker only knows the counts, his/her probability of guessing will be quite low even if no privacy mechanism is used. In reality, it is unlikely that the attacker has no other information at all. It is quite possible that he/she already knows the location of some people

at the time of the attack. For example, some locations may be observed by the attacker directly (e.g., if the attacker is physically located in the space, he/she could observe who is around him/her) or through security cameras (e.g., for an attacker with access to the security camera system). We will refer to those areas for which the attacker has information about who is located inside of it at the time of the attack as *open regions*. Considering such open regions reduces the total number of locations where the victim could potentially be, so the posterior probability increases. The prior probability will also change, and will be scaled according to the number of open regions.

Posterior with a Privacy Technique

Let \mathbf{X} be the random variable representing attacker's opinion about the input x , and \mathbf{C} about the true output c (without noise). Let \mathbf{Y} be the random variable representing the noisy output y , and $f_Y(\cdot)$ its probability density function. We let x denote the part of the input, guessing which is the attacker's goal, i.e., the location of the victim. Let X be the total space of possible values of x , i.e., all possible locations. The particular timepoint that we consider is implicit, and we do not use it in the notation.

In Sec. 3.4.1.2, we have shown how to compute $\Pr[u \in l \mid \mathbf{C} = c]$ for a count histogram $c = (c_\ell^t)_{\ell \in \mathcal{L}}$. In this case, $\mathbf{X} = x$ iff $u \in l$, so we can use the results of Sec. 3.4.1.2 to compute $\Pr[\mathbf{X} = x \mid \mathbf{C} = c]$. We can use these results also to compute $\Pr[\mathbf{X} = x \mid \mathbf{C} = c, \mathbf{A}]$ where \mathbf{A} is the additional knowledge that comes from opening some regions to the attacker. We want to estimate $\Pr[\mathbf{X} = x \mid \mathbf{Y} = y, \mathbf{A}]$.

A Worst Case Bound for Laplace Mechanism

First of all, let us discuss a known upper bound on posterior probability for Laplace mechanism, taken from [103]. Assume that the attacker already knows the location of all other users except the victim. Let X be the set of possible choices for the attacker, i.e., locations of the victim. Laplace mechanism parametrized by ϵ gives us an upper bound $f_Y(y \mid \mathbf{X} = x) \leq e^{\epsilon|x-x'|} f_Y(y \mid \mathbf{X} = x')$ for all $x, x' \in X$. Using Bayesian inference, for all $y \in Y$, we can write

$$\Pr[x \mid \mathbf{Y} = y] = \frac{f_Y(y \mid \mathbf{X} = x) \Pr[x]}{f_Y(y)} \quad (35)$$

$$= \frac{1}{1 + \frac{\sum_{x' \in X \setminus \{x\}} f_Y(y \mid \mathbf{X} = x') \Pr[x']}{f_Y(y \mid \mathbf{X} = x) \Pr[x]}} \quad (36)$$

$$\leq \frac{1}{1 + e^{-\epsilon} \frac{\sum_{x' \in X \setminus \{x\}} \Pr[x']}{\Pr[x]}} \quad (37)$$

so, in our case study an upper bound on posterior guessing probability is $1/(1 + e^{-\epsilon} \cdot (1 - p_{ul}^t)/p_{ul}^t)$, where p_{ul}^t is the prior probability of user u being in location l at time t .

One disadvantage of the obtained upper bound is that we assumed a very strong attacker who already knows locations of all other users. We could try to increase the number of unknown users, which changes the definition of X . This would decrease the prior, but at the same time we would get $e^{-\epsilon k}$ instead of $e^{-\epsilon}$, where k is the number of unknown users. The exponent grows too fast with k . Hence, we cannot experiment with adversarial knowledge parameter A to get smaller levels of posterior probability.

Another problem is that the obtained upper bound is very generic and does not depend on the true query output $q(x)$, so it approaches 1 as noise approaches 0, and we cannot use it to evaluate a *particular query output*. Intuitively, if the privacy mechanism releases $q(x) + \eta$ for some randomly sampled η , then the attacker cannot get more advantage in guessing x than from observing $q(x)$. Hence, we are looking for other approaches that would give us an upper bound $\Pr[x \mid q(x)]$ on posterior guessing advantage.

Posterior for a Particular Noisy Output Instance

Let us try to evaluate the quantity $\Pr[x|y, A] = \Pr[x|Y = y, A]$ directly. Let C be the set of all possible true outputs (e.g., count histograms). Using chain rule, we can write it out as

$$\Pr[x|y, A] = \sum_{z \in C} \Pr[x|y, C = z, A] \cdot \Pr[z|y, A] . \quad (38)$$

This equality can be viewed as an attacker making a guess z about the real output $q(x)$ and checking how likely this z could be obtained from the noisy output y . After the attacker has selected z according to y , it makes it guess purely from z and the additional knowledge A , so $\Pr[x|y, C = z, A] = \Pr[x|C = z, A]$. To estimate $\Pr[z|y, A]$, the attacker takes into account the likelihood of the noise that would turn z into y , as well as the probability of z itself. It can be done using Bayesian inference $\Pr[z|y, A] = \frac{f_Y(y|C=z, A) \cdot \Pr[z \mid A]}{f_Y(y \mid A)}$. Since A only contains knowledge about the data, and not the distribution, we have $f_Y(y|C = z, A) = f_Y(y|C = z)$, which can be computed from the noise distribution. The quantity $\Pr[z \mid A]$ can be computed from prior probabilities, taking into account the additional knowledge. From these two quantities, we can in turn compute $f_Y(y \mid A) = \sum_{z \in C} f_Y(y|C = z) \Pr[z \mid A]$. We get

$$\Pr[x|y, A] = \frac{\sum_{z \in C} \Pr[x|C = z, A] \cdot f_Y(y|z) \Pr[z \mid A]}{\sum_{z \in C} f_Y(y|z) \Pr[z \mid A]} . \quad (39)$$

Intuitively, we want that our estimated posterior probability would stay between the prior $\Pr[x \mid A]$ and the probability $\Pr[x \mid c, A]$ of guessing from the true count c .

Posterior for a Particular True Output Instance

Fixing a particular $y \in Y$ can make the attacker seem too successful or too unlucky, depending on the $y \in Y$ that we have got. Knowing a particular distribution on Y , we may estimate how much the attacker may guess in average for a particular output $c := q(x) \in C$. First of

all, we could directly compute the average posterior probability for all possible outcomes y as

$$\Pr[x|A] = \int_Y \Pr[x|y, A] \cdot f_Y(y|c) dy . \quad (40)$$

This approach is good if the resulting integral has closed form, or at least can be approximated efficiently. However, in practice it may be computationally hard to compute the integral precisely. Alternatively, we can *empirically* compute the posterior probability on many instances of randomly generated noise.

Experiments

The experiments are performed on presence and occupancy records for 3 months (February, March, April, 2018), which comprises $N = 89$ days. We use the first $N - 1$ days of presence data for constructing prior probabilities. We then use the last N -th day of the occupancy table to compute posterior probabilities, showing how attacker's guesses improve compared to guessing from prior. Our analysis consists of the following steps.

1. We split a day into 10-min spans. This gives us $T = 144$ time units per day.
2. For each 10-min span t of a day, for each location l and each user u , we compute prior probabilities $P_{prior(u,t,l)}$ from the first $N - 1$ days.
3. Based on the prior probabilities $P_{prior(u,t,l)}$ and the noisy occupancy counts generated by a particular privacy mechanism \mathcal{M} on the N -th day, we compute the posterior probabilities $P_{noisyOcc(u,t,l)}^{\mathcal{M}}$. Among other mechanisms, we estimate $\mathcal{M}(x) = x$ (guessing from true occupancy counts) and $\mathcal{M}(x) = \perp$ (guessing just from prior).
4. Let $P_{true(u,t,l)} \in \{0, 1\}$ be the actual user locations, i.e., $P_{true(u,t,l)} = 1$ iff u was in location l at time t . Compute the following for each user u and time t :

$$P_{guess(u,t)}^{\mathcal{M}} = \sum_{l \in \mathcal{L}} P_{noisyOcc(u,t,l)}^{\mathcal{M}} \cdot P_{true(u,t,l)} \quad (41)$$

5. Plot aggregate privacy metric: how many people have been localized correctly from $P_{guess(u,t)}^{\mathcal{M}}$ with a probability within certain range, excluding those who have been localized with similar confidence purely from prior.

Prior Distribution

For a fixed timepoint t , the attacker receives a prior distribution of location of an “average user”, expressed as $p_l^t \in [0..1]$ for all $l \in \mathcal{L}$, where $\sum_{l \in \mathcal{L}} p_l^t = 1$. The values p_l^t are computed from the training period using counting. That is, for each time of day t , we count the total number of users m_l^t recorded in region j at time t , and define $p_l^t = m_l^t / \sum_{l \in \mathcal{L}} m_l^t$. Hence, the prior defines an expected distribution over region counts for different times of the day. Here we use the meta-knowledge that similar pattern repeats periodically. The priors would be

more precise if we generated, for instance, a separate prior for each weekday, or found some more interesting meta-data like exceptional holidays which should be discarded as outliers. The problem is that, the more we partition the prior, the less data we have to estimate it. Also, too strong prior may nullify guessing advantage, as the attacker would learn too much already from the prior.

Posterior Distribution

The attacker receives noisy occupancy counts $(y_l^t)_{l \in \mathcal{L}}$ of all regions at timepoint t . Depending on the attacker type, certain regions in the buildings are opened. If the opening does not reveal the location of u immediately, it modifies the priors as $p_{ul}^t = \frac{p_{ul}^t}{\sum_{\ell \in \mathcal{L}_{closed}} p_{u\ell}^t}$ for $l \in \mathcal{L}_{closed}$, where \mathcal{L}_{closed} is the set of regions that remained closed.

For true outputs, the victim's location depends solely on the counts of the current timepoint. This is however different for privacy enhancing mechanisms, where sequential timepoints may leak information about each other's randomness. For example, if the attacker knows that the counts most likely do not change during a 1.5 hour span (e.g., if the region is a classroom and it is a lecture time), then the noise will be essentially applied to exactly the same counts, and multiple outputs help in undoing it. Hence, let us only estimate how much the attacker learns from the output of single timepoint. That is, while we still report results for all timepoints t , we assume a separate attacker for each reported timepoint.

Let m be the total number of objects, and $p := p_l^t$ be the probability of each object being in region l . The question is now how to efficiently compute Eq. 39. Since z is not a single count, but a vector of region occupancies, summing up all combinations z_1, \dots, z_n is infeasible. Hence, let us estimate how much the attacker learns from observing the occupancy of one region without taking into account the others. For guessing from true counts, we only need to know z_l and m , so similarly to the issue with different timepoints, taking into account more z_j -s can only help in undoing the noise, e.g., if the attacker knows that some regions are occupied simultaneously.

While these two constraints are fine for guessing for true counts, they give us only a lower bound for general privacy mechanisms, showing how much an attacker can learn at least. Similarly to composition theorems of differential privacy, we can extend our results to several outputs as described in App. ??, but it may give us too rough upper bounds. As a result, we are comparing different privacy mechanisms based on observing a single output. It may be that some mechanism scales better with the number of observed outputs than another, which remains out of scope of this paper.

We are now ready to estimate the posterior probability. Using Eq. 39, we get

$$\Pr[x|\mathbf{Y} = y] = \frac{\sum_{z=1}^m \Pr[x|\mathbf{C} = z] \cdot f_Y(y|z) \Pr[z]}{\sum_{z=1}^m f_Y(y|z) \Pr[z]}, \quad (42)$$

where y is the occupancy of *one particular region*, and the probabilities are instantiated as follows.

- $\Pr[x|\mathbf{C} = z] = \frac{z}{m}$.

- $\Pr[z] = \binom{m}{z} p^z (1-p)^{m-z}$.
- $f_Y(y|z)$ depends on the particular analyzed privacy mechanism.

The quantity $f_Y(y|z)$ is either given in advance (for a known noise distribution like Laplace), or approximated from (y, z) points of training data using *kernel density estimation* (KDE). This is a quite standard density approximation technique, which can be viewed as assigning to each sample point a bell-shaped curve centered at that point, and then summing all curves up, scaling the result to get a probability distribution. We use Gaussian kernel of Python `scipy` library. By default, the Gaussian KDE bandwidth parameter in `scipy` library is $n^{-1/(d+4)}$, where n the number of data points and d the number of dimensions. In our case, n comes from the training data (88 days), and since we are computing a separate kernel for each true count z (the approximated probability density is conditional), we have $d = 1$. We note that attacker's success may depend on the bandwidth parameter, and choosing one that approximates the noise distribution most precisely is out of scope of this work.

In general, we do not know which parameters we should take into account in training. E.g., even if we include all occupancies of all regions to predict occupancy of a single region, some meta-knowledge like day of the week or time of the day can actually affect the noise distribution. For DP, we actually know that the noisy output depends only on the randomness and the true output, but in general we do not know all dependencies in advance. To simplify the training process, in our experiments we compute the noise distribution for a single location at a single timepoint, as if the randomness was sampled independently.

The posterior confidence of the attacker may be erroneous due to improperly computed noise distribution, or improperly computed priors. The latter may happen even if we use a well-defined DP mechanism. Hence, we add an important condition to our privacy metric. We model a particular attacker who actually *makes a particular guess* about victim's location. We then check whether that guess has been correct or not, and nullify the estimated advantage if the guess was incorrect.

Results

The results of our experiments are presented as plots (a sample plot is given in Figure 63). For each of the T timepoints, we count the total number of people whom the attacker managed to localize correctly with certain *confidence*, defined as the posterior probability of being in the room where the user has actually been according to the presence table. On each plot, the x -axis denotes the timepoint, and the y -axis is the number of localized people. The colors, ranging from light blue to dark red, correspond to localization confidence p , where light blue is the lowest confidence class ($0.0 < p \leq 0.1$), and dark red is the largest confidence class ($0.9 < p \leq 1$). Notice that the plot shows for each time point the total amount of people localized in the building broken into different confidence classes. This means that, for instance, out of the 123 people located in the building at 11:40am of the particular day in Figure 63, 22 are localized with the lowest confidence $0.0 < p \leq 0.1$, 55 with confidence $0.1 < p \leq 0.25$, 26 with confidence $0.25 < p \leq 0.5$, and 20 with confidence $0.5 < p \leq 0.75$. There are no red and dark red areas for 11:40am, so there have been no people localized

with confidence $p > 0.75$. The plot format will be the same for all plots in this paper, so we will avoid repeating labels and legends on further plots to conserve space.

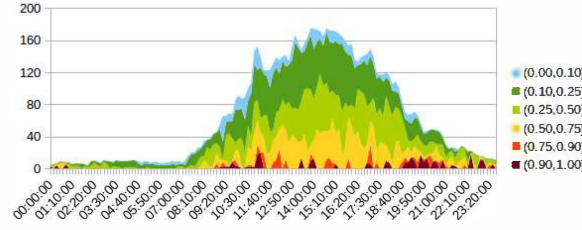


Figure 63: A sample plot: x -axis is the timepoint, y -axis the total number of people localized within each confidence class.

Exact vs Predicted Noise Distribution

First of all, we need to estimate how well predicting $f_Y(y|z)$ using kernel density estimation from training data works compared to true noise distribution. We do it on the example of Laplace noise, for which we already know the true distribution of noise $f_Y(y|z) = \frac{\epsilon}{2} \cdot e^{\epsilon \cdot |y-z|}$.

The plots of prior probabilities are given in Figure 64. The three columns correspond to the initial knowledge of the attacker, where he knows $\delta \in \{0\%, 50\%, 90\%\}$ of people locations. In this experiment, the regions have been opened randomly. We compute the posterior probability for each user on the condition that their location has not been revealed to the attacker directly, so formally for each potential victim we consider a separate attacker who knows δ of the *other* users. This is why we do not observe that δ of the graph is dark red.

For posteriors we will only show those probabilities that have been *improved compared to the prior*, thus demonstrating the *advantage*. The results are given in Figure 65. The rows of the plot matrices correspond to $\epsilon \in \{0.1, 1.0, 5.0, \infty\}$, where ∞ is guessing from true outputs. We can see how confidence increases with ϵ . We see that $\epsilon \geq 5.0$ already gives us a plot very similar to guessing from true outputs, so it does not make sense to consider larger epsilons. For smaller epsilons, we indeed get smaller confidence, which converges to 0 as $\epsilon \rightarrow 0$. While the posterior *probability* always increases with δ , we see that the *advantage* may sometimes be larger for smaller δ , which means that the attacker guesses so poorly from prior that even a very noisy answer gives some benefits.

We compare obtained results with the worst-case upper bound estimate considered in Sec. 3.4.1.2, which does not depend neither on δ nor the particular counts, and holds for any ϵ -DP mechanism. The results are given in Figure 66 for $\epsilon \in \{0.1, 1.0, 5.0\}$. We see that, for larger ϵ , the upper bound gets larger than the probabilities of guessing from true counts, so the upper bound is too rough for our type of attacker. These bounds nevertheless seem to be good for small ϵ .

Table 13 shows the times of computing the posterior probabilities. The first row corresponds to computation of the worst case bound of Sec. 3.4.1.2. The second row corresponds to precise computation of posterior (Eq. 39). In general, the precise computation of posterior

is $O(n)$ times slower for n rooms since we are evaluating a sum over n terms. In our example, it is ca 9 times slower for $n = 64$ rooms. The preprocessing time is spent on the bookkeeping related to loading data from the database. We do not count the time spent on generating the occupancy tables.

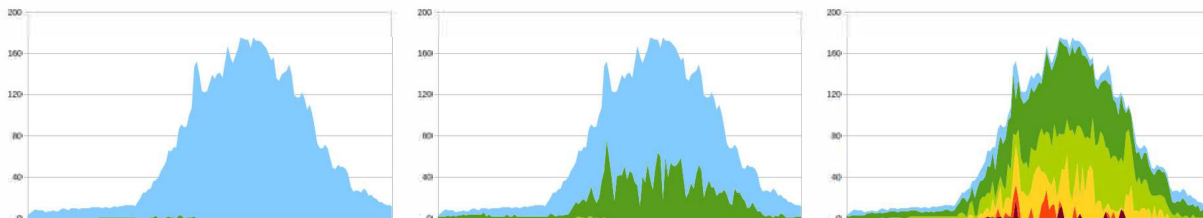


Figure 64: Prior guessing probabilities for $\delta = 0\%$ (left), $\delta = 50\%$ (middle), $\delta = 90\%$ (right)

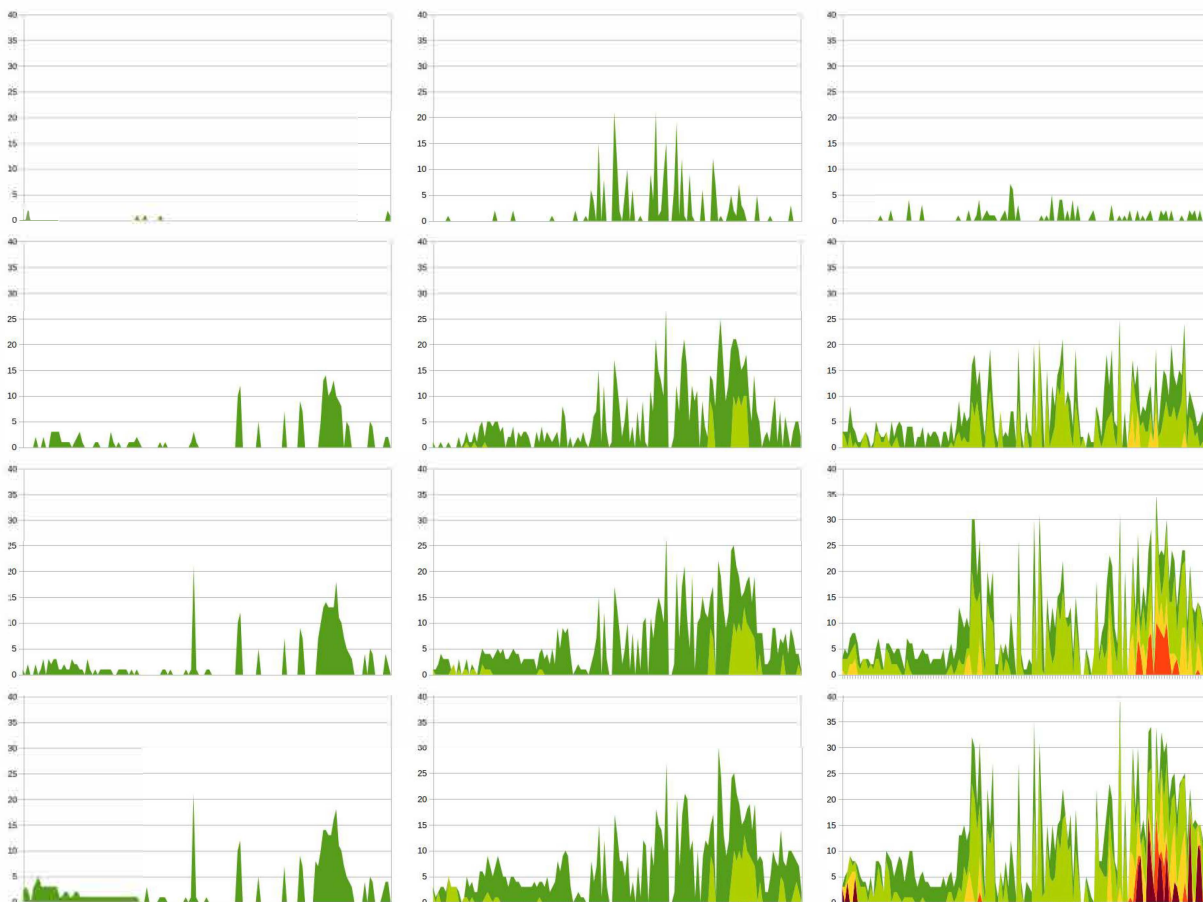


Figure 65: Posterior guesses from noisy Laplace counts for $\delta = 0\%$ (left), $\delta = 50\%$ (middle), $\delta = 90\%$ (right), $\epsilon \in \{0.1, 1.0, 5.0, \infty\}$ (top to bottom)

Since the trained noise distribution does not depend on δ anyway, let us only consider $\delta = 0.9$.

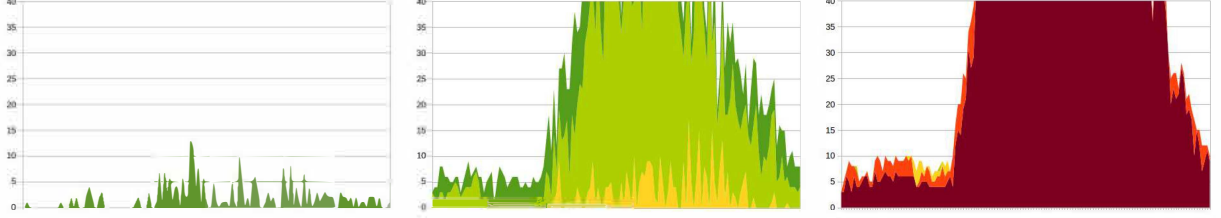


Figure 66: Posterior guesses for worst-case DP, $\delta = 90\%$, $\epsilon \in \{0.1, 1.0, 5.0\}$ (left to right)

First of all, we repeat the experiment with known DP noise distributions, filtering out the guesses that have actually been correct (top row of Figure 67). We then repeat a similar experiment with noise distribution learned from prior data, using Gaussian kernel with default parameters (bottom row of Figure 67). The plots are shown for $\epsilon \in \{0.1, 1.0, 5.0\}$ (left to right). We see that the guesses based on trained distribution perform similarly to the true distribution of Laplace noise.

Table 13: Running times (in seconds) of computing posterior probabilities for Laplace noise.

| | Preprocess | Training | Posterior evaluation |
|-----------------------------------|------------|----------|----------------------|
| worst-case DP bound | 4 | 0 | 0.6 |
| $\Pr[x y]$ for known $f_Y(y z)$ | 4 | 0 | 5.4 |
| $\Pr[x y]$ for unknown $f_Y(y z)$ | 4 | 6.9 | 7.3 |

The third row of Table 13 shows the times for the trained distribution experiment. We see that the testing time is a bit higher, which is due to computing $f_Y(y|z)$ from kernel. In addition, there is now also some time spent on one training to compute the kernel itself. We note that for DP experiments, we need a separate training for each ϵ .

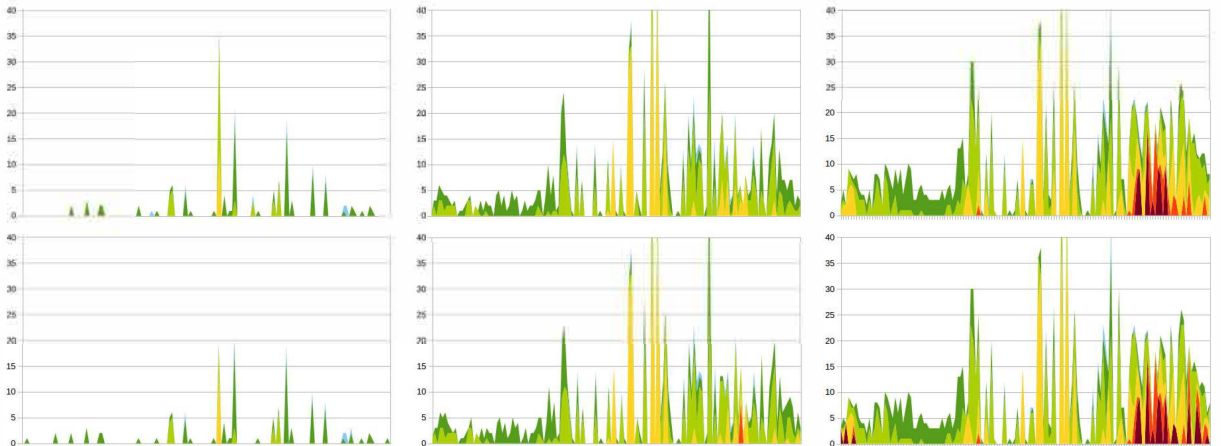


Figure 67: Successful guesses from noisy Laplace counts with $\delta = 90\%$ using true distribution (top) and scaled kernel density estimation (bottom), $\epsilon \in \{0.1, 1.0, 5.0\}$ (left to right)

Comparing Different Privacy Techniques

We now compare the techniques w.r.t. the attacker's success in breaking user privacy for the same level of utility. The noise density functions of all methods is approximated from training data using Gaussian kernel with scaling 0.1. The parameter 0.1 has been chosen empirically as the one for which the attacker was more successful. We note that a different scaling parameter can be preferred for different mechanisms, and that the goodness scaling in turn depends on the size of training data sample: the more datapoints we have, the more we want to narrow the kernels to get a more precise estimate.

TTL changes the unique identifiers of the objects every k seconds for a fixed parameter k . However, counting does not depend on the identifiers anyway. The only way in which TTL can affect released counts is that the same user may be recorded multiple times if his identity has been updated within the 10-minute span for which the count is computed. If the attacker knows the value k , it can just divide all counts by the expected number of repetitions to get the true counts back. We see that there are no provable security guarantees at all. However, in practice the number of repetitions turns out to be less predictable, as the identities are not being updated "for all users at once", but depend on the time when the user has connected to the system. While there is no randomness, the non-determinism of user movement can be viewed as a random variable whose distribution is difficult to estimate theoretically.

To fairly compare privacy loss across diverse PETs, their utility should be similar as there is an inherent privacy vs. utility tradeoff. We have developed a tool which, given a specific task, obtains the configuration parameters per PET (i.e., epsilon and TTL) that will satisfy a given utility requirement. The tool takes as input the WiFi connectivity data, the PET to apply, a function that computes utility, and a requirement for the utility. Then, it tries different values for the privacy parameter until it finds one that satisfies the utility requirement. In our set up, we define the task to be that of generating a heatmap given the occupancy data (real data or data generated through a PET). Our utility metric is computed as a percentage difference between the heatmap color assigned to the occupancy value generated by the PET and the heatmap color assigned to the real occupancy value (100% utility means real data and data generated by the PET are assigned same color). Then, we average this utility across time and space to get the utility value for a given dataset.

We computed parameters for PeGaSus, Laplace, and TTL that give us the utility 75% and 90%. These parameters are summarized in Table 14. We compute these parameters over 5 runs for non-deterministic techniques (i.e., Laplace, PeGaSus). The utility in each run lies within $75 \pm 0.2\%$ and $90 \pm 0.2\%$ for the given parameters. Since the privacy parameters for the same utility are very different for day and night time, we also extract privacy parameters for day and night times and perform different experiments. Since TTL tends to map empty rooms to empty in most cases, for TTL we get high utility at night when the most true counts are 0, which allows us to introduce more noise, i.e., refresh the user identities more frequently. To get 90% utility for PeGaSus for day time, ϵ turns out to be very high i.e. 15. For high values of utility, we see a very small increase (only 2%) in utility from $\epsilon = 2$ to $\epsilon = 15$ which is possible due to error introduced by grouping/smoothing of contiguous similar

occupancy counts. Since the utility is averaged over time and space, we also show average variance in utility over time and space σ_u (which turns out to be similar across techniques).

Table 14: Parameters achieving the same utility for different privacy mechanisms.

| | utility | Laplace | | PeGaSus | | TTL | |
|-------|---------|------------|------------|------------|------------|------------|------|
| | | σ_u | ϵ | σ_u | ϵ | σ_u | T |
| day | 75% | 23.7 | 0.1 | 24.1 | 0.04 | 20.4 | 1sec |
| | 90% | 14.5 | 0.66 | 16.9 | 15 | 14.9 | 2min |
| night | 75% | 26.6 | 0.18 | 26.6 | 0.00001 | 19.5 | 1sec |
| | 90% | 19.3 | 1.675 | 19.5 | 4 | 19.5 | 1sec |

Instead of opening regions to the attacker randomly as in Sec. 3.4.1.2, we consider in the following certain types of realistic attackers in the context of the dataset. There are 64 regions in the building, each having typically granting access to different profiles of people (e.g., students, professors, staff). We will consider two types of attackers based on such information.

1. An external attacker who is not present in the building and thus, does not have access to any region (i.e., 0 open regions).
2. A building administrator who has access to the security camera system and thus has access to all the regions covered by cameras (i.e., 39 open regions). This set of spaces includes public areas as well as corridors near offices.

Figures 68-69 compare different privacy techniques in different settings for the previous attackers. The columns correspond to the four types of experiments (day/night, 75%/90%-utility), and the rows to different privacy mechanisms, including guessing from true counts (the last row). In each graph, the X-axis is time (7am-7pm for day and 7pm-7am for night in intervals of 10 minutes) and the Y-axis is the number of people correctly localized at each confidence level¹⁴. As the noise added by DP techniques will be different in different executions, each experiment has been repeated $n=30$ times, and for each posterior probability class we took the average number of people that has been guessed with that probability. Finally, we consider that in the case of these realistic adversaries an open region implies that the adversary knows exactly who is inside of it. Thus, in the following we consider such information to be prior and in the plots we focus on how the different PETS affect the guessability of those individuals in closed rooms. Let us now discuss the results for different attackers.

¹⁴Note that the scaling of Y-axis for day and night time are different, as the total number of people in the building is very different for them.

External attacker. In Figure 68, we see the results for an external attacker who has a weaker prior as he/she does not have access to real location information for any user at the time of the attack. Given the true counts, the external attacker can localize some users from, albeit with confidence $p \leq 50\%$. In particular, during the day time and early morning, when the occupancy of the building reaches its peak, the external attacker can perform the most successful attack by correctly guessing the localization of 50 people with a confidence of $0.25 \leq p \leq 50\%$. Notice that during the night, the number of people in the building is small and the most successful attack occurs in the early morning with 10 individuals localized with low confidence. Using privacy mechanisms eliminates most of these localizations. Indeed, with parameters satisfying 75% utility, which increase the privacy protection, the localization possible when publishing real counts is completely eliminated. Notice that when a higher utility is required (90%), the mechanisms might publish data closer to the real occupancy and therefore there is some leakage. Notice that the results are averages from different runs which means that mechanisms based on differential privacy, which are not deterministic, might perform better or worse in specific situations. We explore this aspect further in App. ???. From the plots, we cannot determine whether one mechanism is better than the other one, as the number of localizations is small for all of them.

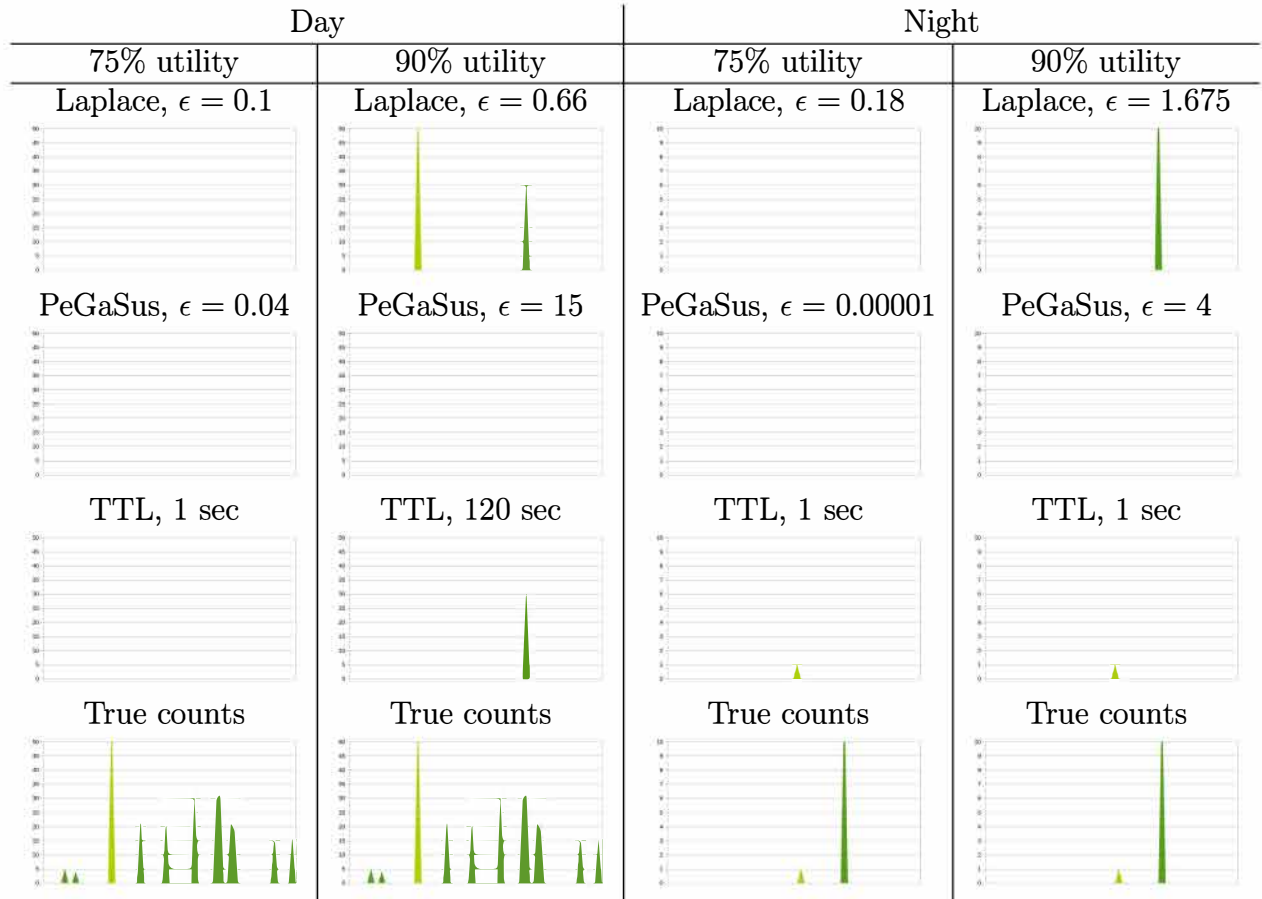


Figure 68: Comparison of different mechanisms for the external visitor attacker.

Administrator attacker. An administrator is given access to 39 of the 64 regions. This situation is similar to the previous but the set of open regions is different including those that contain offices. The results are also similar to those of the student attacker, although in general there is more privacy loss across techniques and for the true counts. This occurs because this adversary has access to real location of more people as the open regions cover a higher amount of the building's population. As in the case of student attacker, both the techniques based on differential privacy perform slightly better than the TTL technique. Similarly, the Laplace technique performs slightly better than PeGaSus. When comparing TTL and PeGaSus for the daytime and 90% utility we notice how both perform very similarly in the afternoon when the building is less occupied. Also, when focusing on the nighttime, at 90% utility all the techniques perform very similarly.

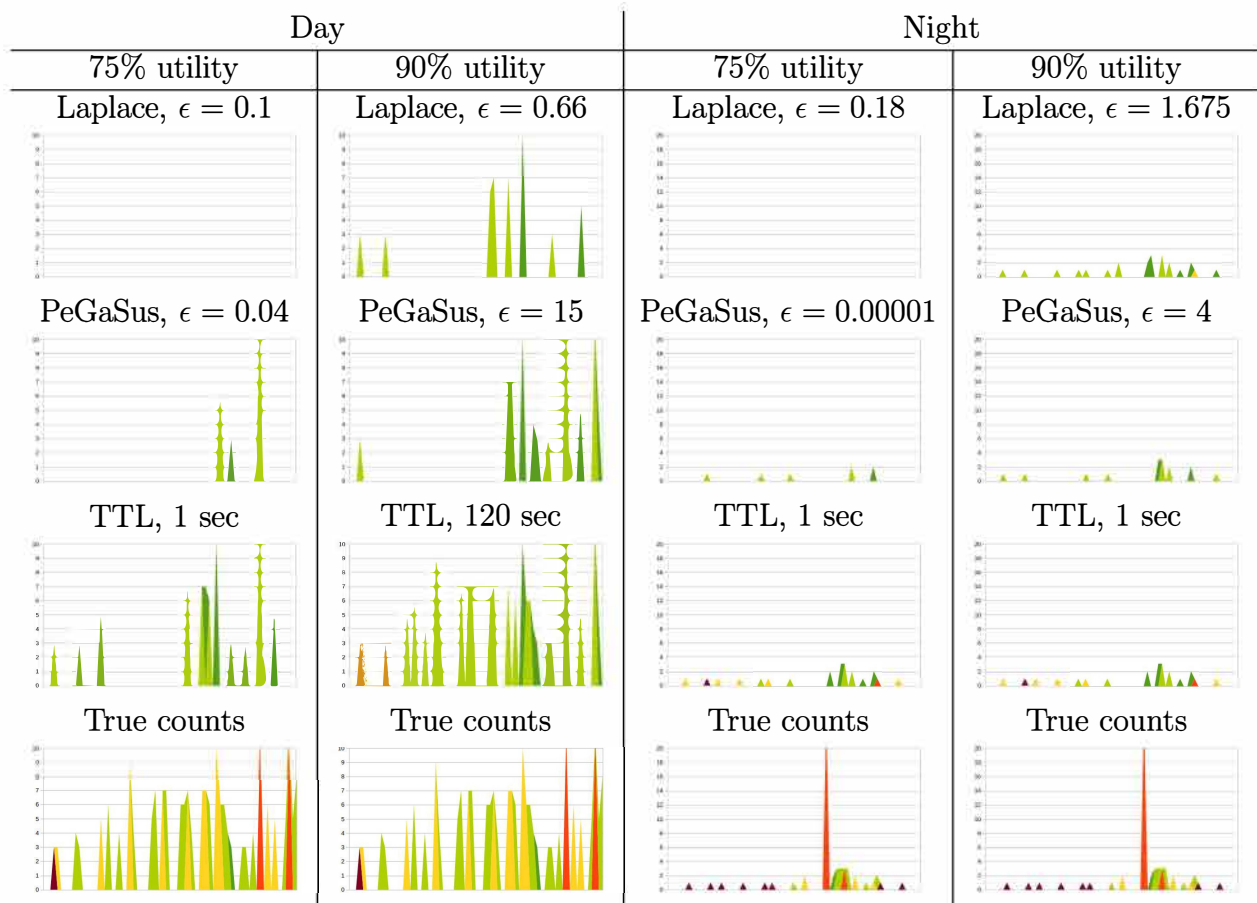


Figure 69: Comparison of different mechanisms for the administrator attacker.

Summary

Differential privacy in general consider very strong attackers that have access to almost unlimited information. In our set up with more realistic adversaries, we have seen that when the adversary is weaker (e.g., our external attacker). The practical privacy offered by Laplace

and PeGaSus is almost the same than the one offered by TTL. Even when the practical privacy for Laplace and PeGaSus is similar, for higher utility values the formal privacy guarantee for PeGaSus is less than Laplace during the daytime and nighttime, whereas for lower utility values it is the opposite. When the adversaries become stronger (e.g., the student or administrator attacker), Laplace and PeGaSus offer more practical privacy than TTL, as it is expected. However, in some specific situations (e.g., in the afternoon when the building is less occupied) all the techniques behave similarly. Additionally, with stronger attackers the privacy loss due to the prior and adversarial knowledge at the time of the attack is high already. This means that effectively, in such situations the privacy of most of the individuals would be already compromised. Therefore, the difference between the differential privacy based techniques and TTL in terms of number of people being localized is small.

We would like to highlight that even when in terms of practical privacy the techniques behave similarly, TTL lacks of formal privacy guarantees which means that stronger attackers using more sophisticated attack methods could potentially result in higher privacy loss. Additionally, the results for Laplace and PeGaSus, which are the average over 30 counts, could potentially be worse depending on the noisy count generated in a single run at publishing time. However, comparing the *distributions* of attacker's success for different privacy mechanisms remains out of scope of this work.

3.4.2 Evaluation at Honeywell

Smart buildings are becoming increasingly prevalent as an integral part of society. Equipped with various sensors and actuators, smart buildings enable automated building controls including light, Heating, Ventilation, and Air Conditioning (HVAC), and space optimization [104]. This results in personalized comfort controls for the building occupants as well as economic savings for the owners of buildings. However, as more fine-grained data at individual levels are collected for building automation, dangers of potential privacy breach also increase [105]. In collaboration with Honeywell, we studied privacy implications of occupancy sensors in smart buildings. We were interested in a setting where each individual is given a primary location where the person is expected to be throughout the day, such as cubicles or offices.

Occupancy sensors detect presence of an individual which enables dynamic control of light and HVAC systems based on the detected occupancy information. Use of occupancy sensors for various building applications has been a subject of active research for both academia and industry [106, 86, 107, 108, 85]. Occupancy-driven building applications include both energy [106] as well as space optimization [108]. As occupancy data is being widely implemented, identifying privacy leakage and developing potential mitigation strategies against the potential privacy breaches have been gaining increasing attention [107, 109]. In particular, the collected occupancy data could potentially be used for inference of individual's occupancy patterns and tracking of individuals [109].

In many cases, the automation control is indifferent to the identities of building occupants. For example, detection of a presence is sufficient to trigger the light to be turned on at the sensed location. The automated building control does not require the knowledge of *who*

is triggering the occupancy sensor, but only requires the information that a presence is detected at a particular location. For this reason, the database containing user-identifying information is typically kept separately from the database storing occupancy information, and there is no communication between those databases. Because of this separation, it is easy to believe that unless the adversary compromises both databases, it is difficult to infer the mapping between the set of sensors and the identities of individuals. The findings of this paper demonstrate that maintaining separate databases is insufficient for mitigating privacy leakage from occupancy sensors.

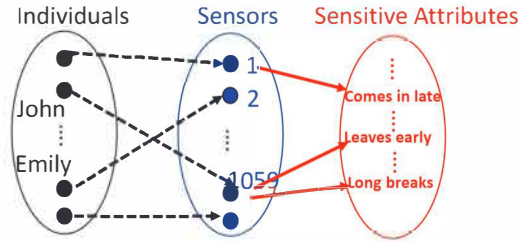


Figure 70: Figure illustrating the privacy implications of the presence data.

3.4.2.1 Model and Preliminaries

In this section, we present the adversary and building models.

Building Model. We consider a building that is divided into multiple locations where each location is equipped with a presence sensor. We assume that a presence sensor is capable of inferring whether or not the location is currently occupied, but is not capable of inferring which specific individual is triggering the occupancy. An example of such a sensor would be a motion detector which uses ultrasonic or microwave technologies. Each individual is given a primary location where the individual will be spending most of the time when the individual is in the building. For example, a primary location might be a designated office, a particular room, or a particular cubicle.

The set of presence sensors are denoted as \mathcal{S} . Each sensor periodically detects presence in its neighborhood. The occupancy pattern of sensor $s_j \in \mathcal{S}$ at time t is denoted as o_j^t and is defined as

$$o_j^t = \begin{cases} 1, & \text{if sensor } s_j \text{ detects presence at time } t \\ 0, & \text{else.} \end{cases}$$

Similarly, we define $o_j^{[t_0, t_f]} = 1$ if occupancy is continuously detected during the time interval $[t_0, t_f]$ and $o_j^{[t_0, t_f]} = 0$ otherwise.

Adversary Model. Any application that consumes the occupancy data could potentially use the data other than its intended purposes. The goal of the adversary is to infer the sensor ID that is installed at the primary location of the targeted individual i denoted as s_i . It is assumed that the adversary can query any sensor ID to observe the presence information at any given time t .

The adversary also has a set of auxiliary information of the individual i , which is denoted as \mathcal{A}_i . Auxiliary information is defined as occupancy-pattern information that the adversary has about the individual. An example of a piece of auxiliary information is a travel schedule of the individual from which the adversary can infer that the individual was not in the building for a given time interval. Such information is often public information that can easily be obtained from social network sites including LinkedIn¹⁵, where postings may include an individual receiving an award or giving a presentation at a conference. Such information would indicate that the individual was not present in the building on the days when this was posted.

For each auxiliary information $a_i^{[t_0, t_f]} \in \mathcal{A}_i$, we denote $a_i^{[t_0, t_f]} = 1$ if the individual i was known to be in the building during the time interval $[t_0, t_f]$ and $a_i^{[t_0, t_f]} = 0$ if the individual i was known to be absent from the building during the time interval $[t_0, t_f]$.

3.4.2.2 Proposed Attack

In this section, we propose an attack in which the adversary infers the sensor ID of the targeted individual i with high probability by combining the auxiliary information with the queried occupancy patterns. The main idea of the attack is to exploit the fact that the sensor s_i has to satisfy all the conditions presented by the auxiliary information \mathcal{A}_i . An illustration of the proposed attack is shown in Figure 71.

Without loss of generality, we index the set \mathcal{A}_i as

$$\mathcal{A}_i = \{a_i^{[t_0, t_1]}, \dots, a_i^{[t_{n-1}, t_n]}\} \quad (43)$$

\mathcal{A}_i represents the set of auxiliary information obtained by the adversary for individual i . For each item of auxiliary information $a_i^{[t_{m-1}, t_m]} \in \mathcal{A}_i$, define the set $\mathcal{S}_m^i \subset \mathcal{S}$ as

$$\mathcal{S}_m^i = \{s_j : o_j^{[t_{m-1}, t_m]} = a_i^{[t_{m-1}, t_m]}\} \quad (44)$$

In other words, the set \mathcal{S}_m^i is the set of sensors such that the corresponding occupancy patterns are consistent with the auxiliary information $a_i^{[t_{m-1}, t_m]}$. Given n auxiliary information, the adversary can construct n such sets, and the sensor s_i has to be in all of the sets $\mathcal{S}_1^i, \dots, \mathcal{S}_n^i$. Assuming the adversary does not have any a priori information regarding s_i , the probability of correctly guessing s_i given \mathcal{A}_i is written as

$$\mathbf{P}(\hat{s}_i = s_i | \mathcal{A}_i) = \frac{1}{|\bigcap_{m=1}^n \mathcal{S}_m^i|} \quad (45)$$

where \hat{s}_i is the adversary's estimate of s_i .

¹⁵<https://www.linkedin.com/>

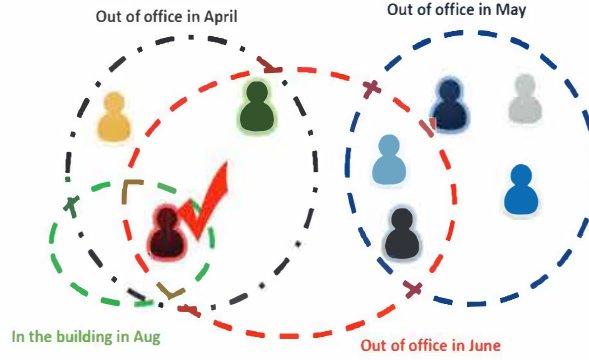


Figure 71: Figure illustrating the proposed attack.

3.4.2.3 Experimental Study

We conducted an experimental study using real-world occupancy sensor data from an office building. We selected one zone of the office building for the experimental study which contained 1663 motion detectors. Each motion detector records a binary value of 1 if it senses presence via motion and 0 otherwise. The presence value is recorded every 5 seconds and transmitted to a centralized database. The auxiliary information we considered was daily travel schedules of individuals. The travel schedules used for this study were obtained either with user consent or through public data from LinkedIn. The set of auxiliary information was gathered within the period of 5 months.

For each day, we assumed that an individual was present in the building if person stayed in his or her primary location for more than an hour, which is equivalent to having more than 720 number of 1's in the presence data for one day. The assumption was made to account for triggering of the motion detector by bypassing people or patrolling security guards at night time. If the presence data for a sensor contained less than 720 number of 1's for one day, then we assume that the area covered by the sensor was unoccupied for that day. We conducted our experiment on four employees. For the first case, we used three pieces of auxiliary information.

1. The person traveled for two days during weekdays and therefore was not present in the building during the travel.
2. The person has not worked during the weekends for a particular month.
3. The person returned to work after two days of travel and was present in the building on the third day.

Without any additional information, the probability of correctly guessing the sensor ID that is located at the person's primary location is $\frac{1}{1663}$. After identifying the set \mathcal{S}_1 that satisfied the first auxiliary information 1), the cardinality of the set was reduced to 14 from 1663. In addition, after obtaining the intersection $\mathcal{S}_1 \cap \mathcal{S}_2 \cap \mathcal{S}_3$, where \mathcal{S}_2 and \mathcal{S}_3 are the sets that satisfied the auxiliary information 2) and 3) respectively, the cardinality of the new set was

reduced to 9 from 1663. We have verified that the person’s primary location (an office space) was indeed one of the nine candidate locations that we identified.

Figure 72 shows how probabilities of correctly identifying primary locations (Equation (45)) change with auxiliary information for all four cases. The horizontal axis is the relative time of the year when the auxiliary information was gathered within the period of five months. For cases 2,3, and 4, the *off* label shown in Figure 72 indicates that the person was not present in the building during weekdays and the *on* label indicates that the person was present in the building. In all four cases, we observed that with two to three pieces of auxiliary information, the cardinalities of candidate sets will reduce to approximately 10, resulting in significant increase of probabilities of correctly identifying the primary locations for all four individuals to approximately 0.1 from $\frac{1}{1663}$.

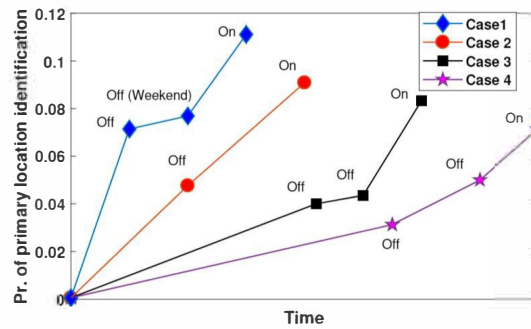


Figure 72: Figure illustrating the increase of probability of correctly identifying the primary locations given auxiliary information.

In all four cases, it was observed that a person’s absence in the building during weekdays significantly reduces the size of the candidate set initially. This is due to two reasons. First, identifying the candidate set of locations that are occupied during weekdays will eliminate almost all sensors that are located in hallways, which are usually occupied during weekdays. Second, since the building chosen for this study does not allow working from home, being absent during weekdays is a rare event.

3.4.2.4 Mitigation Strategy

In this section, we propose a set of potential mitigation strategies against the proposed attack.

Principle of Least Privilege for Privacy. Principle of least privilege has been widely accepted as one of the governing principles in designing mitigation strategies in security. The principle of least privilege states that the minimum amount of information and resources should be given to a subject in order for the subject to perform required duties. The same principle can be used for privacy enhancement as stated in [110].

For building applications, we need to ensure that only the minimum, necessary level of occupancy information is disclosed. For example, for occupancy-driven HVAC control, the controller only requires occupancy information at the zone level, but does not require the information at the individual sensor level [111].

The adversary could still potentially perform the attack using the zone level occupancy information to at least identify which zone includes the targeted individual's primary location. On the days when the individual is not present in the building, the adversary would expect to observe a decrease in the number of occupants in the zone. However, unlike the attack performed on the individual sensor level, the behaviors of other occupants in the zone would affect the accuracy of the identified sets in (45). It is possible that the decrease in the number of occupants could be due to other individuals not present in the building when the targeted individual is present. Similarly, even when the targeted individual is not present in the building, other individuals that were not previously in the zone could be present, resulting in no change or even an increase in the number of occupants in the zone. Moreover, even if the adversary were able to correctly identify the zone where the targeted individual resides in, k -anonymity [112] will be provided where k is the number of sensors in the zone.

Differential Privacy. When the occupancy information at the zone level is released, it is possible to implement additional privacy enhancing mechanisms that will provide differential privacy. In this setting, probabilistic noise is added to the released occupancy information to ensure that answers to queries regarding the occupancy level in a zone would be statistically similar even when a single individual is present or not present in the zone [113].

We have implemented **PeGaSus** [114] on the collected occupancy data to examine how the utility of the occupancy data would change as we vary the parameter ϵ . However, initial results suggest that even for a relatively large value of ϵ , the utility of the gathered occupancy data decreases significantly. Finding the optimal trade-off between the utility and privacy for the differential privacy based mitigation would be part of future work.

Deletion of Past Information. The proposed attack does not rely on the particular times when the presence data are collected. Whether the presence data was collected in the past or present, any data that matches the auxiliary information would be equally valuable in narrowing down the set of candidate sensor IDs of the targeted individual.

While the past data could be valuable in many applications that rely on predicting future occupancy levels as well as energy consumption through machine learning techniques, keeping a large amount of past data will provide additional opportunities for the adversary. Setting an expiration time on the gathered occupancy data and either deleting or encrypting it automatically at the time of expiration will provide additional privacy against the proposed attack by eliminating the occupancy data that correspond to auxiliary information in the past.

3.4.3 Evaluation at US Navy

3.4.3.1 Trident Warrior 2019 TIPPERS Technology Evaluation

An essential part of this project was technology evaluation efforts. In this section, details on the **PET** evaluation efforts performed by the researchers and the USNavy will be discussed. The Trident Warrior exercises was a collaborative effort between privacy and security researchers at nine different institutions along with researchers at the Naval Information Warfare Center to deploy, test, and demonstrate privacy-preserving technologies in creating

sensor-based awareness using the Internet of Things (IoT) aboard naval vessels in the context of the US Navy's Trident Warrior 2019 exercise. Privacy Enhancing Technologies (PETs), including differential privacy, computation on encrypted data, and fine-grained policies were evaluated and analyzed in the context of TIPPERS and its use in creating a smart ship that offers IoT-enabled services such as occupancy analysis, fall detection, detection of unauthorized access to spaces, and other situational awareness scenarios. The analysis describes the privacy implications of creating IoT spaces that collect data that might include individuals' data (e.g., location) and analyze the tradeoff between privacy and utility of the supported PETs in this context.

Over the course of this project, this research team has worked toward transforming TIPPERS to be deployed in tactical naval settings. In particular, TIPPERS was deployed on a Navy ship as part of the annual Trident Warrior exercise in 2019. Trident Warrior is an annual large-scale, at-sea field experiment where the Navy selects potential initiatives that address capability gaps and provide inventive solutions in an operational environment. Below we describe several PETs that were part of the TIPPERS system deployed in the US Navy ship.

PULSAR

PULSAR is a novel secure data management system based on function secret sharing (FSS) and MPC to support real-time privacy preserving data aggregation and retrieval that has been applied to sensors and mobile devices in TIPPERS. A standard secret-sharing scheme allows a dealer to randomly split a secret into two or more shares, such that certain subsets of the shares can be used to reconstruct the secret and others reveal nothing about it. Secret sharing is additively homomorphic, that is, if many secrets are shared, the two parties can individually compute shares of the sum of the secrets by locally adding their shares, without any communication.

The notion of FSS can be viewed as a natural generalization of additive secret sharing to functions. A special case of interest is the class of point functions f , which have a nonzero output on at most one input. A FSS for this class is called a distributed point function (DPF). One exemplary application for such a special case is secure distributed histograms, or "distograms," that allow for the ability to privately aggregate information into histogram buckets. Stealth (the makers of PULSAR) incorporates these distograms into the PULSAR solution as a means of real-time privacy-preserving data aggregation and retrieval.

Jana

Jana technology implements the paradigm of Private Data as a Service (PDaaS). Using a combination of advanced cryptographic techniques and a commercially reliable database, such technology provides a full-featured, robust, relational database management system (RDBMS). The RDBMS cryptographically secures data from before it leaves the platform of a data contributor, until after it reaches the platform of an analyst authorized to see the query results. Data does not need to be decrypted during query processing in Jana. Results

of queries are additionally protected using DP mechanisms (where appropriate) to prevent rediscovery of sensitive data from those results.

Jana is also intended as an operational environment to study the trade-space between security and performance scalability for real-world data and queries. In contrast, typical cryptographic research platforms fix a level of security, argue on standard assumptions, and offer no trade space in which to conduct such research. In addition, Jana allows for the study of implications on performance of full, end-to-end security, while typical cryptography research fails to address the security of all steps in the information flow from data provider to query result. The data within Jana remains encrypted at all times, unless explicitly chosen by the database administrator (DBA) (and agreed to by data contributors) for storage in plaintext form. Ephemeral public key encryption protects data in transit from contributors' platforms into each Jana instance, as well as results in transit from the Jana instance to analyst platforms. Depending on choices made by the DBA, public key encryptions or order revealing encryptions are used to protect data at rest in Jana's relational data store (deterministic encryption is also supported as a research capability, although it is not recommended for the obvious security concerns).

PeGaSus

PeGaSus (PGS) is a specific algorithm for analyzing streaming data under DP.¹⁴ The technology of DP is appropriate in contexts where data about individuals has been curated for the purpose of analysis. The goal is to release the outcome of the analysis while disclosing as little information as possible about individual records in the data. The conventional technologies for this problem fall under the broad category of disclosure limitation and include approaches such as data de-identification and suppression. Unfortunately, these techniques are known to be brittle. For instance, so-called "anonymous" records can often be reidentified.

In contrast, DP offers a rigorous mathematical guarantee: to an individual whose data has been collected, whatever can be learned about the individual when her data is included is essentially no greater than what can be learned where her data is omitted from the collection. DP is distinct from adjacent technologies, such as secure multi-party computation (SMC). With SMC, the goal is to compute the exact answer to a function where each party contributes one private input, with the goal that during the execution of the computation, no party is able to extract information about the inputs supplied by other parties. DP, on the other hand, aims to prevent the output of the function from leaking information. For instance, if the function computed a vote tally and it was unanimous, then the output of SMC would reveal individual votes; in contrast, with DP, one would learn only that the vote was approximately unanimous.

DP is a mathematical definition that can be applied to a variety of data types and for a variety of analyses. In considering its use in a particular application, it is essential to consider what is sensitive and private information, as well as what kinds of analyses should be supported. PGS is applicable in contexts where individuals are continually observed by a collection of sensors. Each sensor produces a stream of data, and the goal is to analyze

these streams in real time without disclosing sensitive personal information about specific individuals. With PGS, the privacy guarantee is on individual events, such as a person being observed by a particular sensor, but extends naturally to small windows of events (observations of a particular individual within the last hour).

3.4.3.2 Integration of PETs in TIPPERS

TIPPERS supports two ways of integrating PETs into sensor data management, as mentioned before. Firstly, virtual sensor technology supported by TIPPERS can be leveraged to modify/perturb the sensor data stream that is shared with applications. Such modifications could include scrubbing of sensitive data (e.g., removing faces from images or identifiers from sensor data), adding noise, de-linking data, etc. A good example of the use of virtual sensors is the integration of PGS to make occupancy data streams differentially private. Secondly, TIPPERS supports mediation between the system and the underlying storage mechanism. For instance, TIPPERS mediates with FSS and MPC technologies supported by PULSAR, as well as deterministic encryption, order-preserving encryption, and secret-sharing based technologies supported by Jana. This way, TIPPERS can be configured to store low-level sensor data (such as WiFi connectivity) in PULSAR such that insertions can be fast and aggregations (e.g., occupancy levels) can be determined quickly. This is crucial for real-time policy enforcement when policies depend on who is in a particular space or how many people are inside it. In contrast, data that may require more complex operations (such as, for instance, Structured Query Language (SQL) joins operations to combine sensor data with metadata) can be stored in Jana, which supports different techniques for encryption and supports more complete complex SQL operations.

3.4.3.3 Evaluation of TIPPERS in Preparation of Trident Warrior

As part of the preparation for TW 2019, the TIPPERS system was deployed first at the Naval Information Warfare Center (NIWC) Pacific. The objective was to enable a demonstration of the capability of TIPPERS, including its integrated PETs, to operational Navy personnel to explore opportunities wherein systems such as TIPPERS can be used within the Navy. TIPPERS was deployed in the NIWC Mobility Center of Excellence (MCoE), which is a lab dedicated to developing, testing, and evaluating mobile technologies. The deployment required developing an application specific to NIWC, referred to as the Security Surveillance application.

The Security Surveillance application provides a bird's eye view of the evolving state of the NIWC facility based on the sensor data that is captured and translated into occupancy levels of each space. The sensor data is securely encrypted in the underlying secure data storage TIPPERS technology. To enforce the need to know concept, the application provides two views of the data. The first shows only differentially private occupancy counts (obtained using the PGS virtual sensor) with no identifying information to preserve the privacy of individuals involved in the data. From this view, a user of the application should not be able

to make any inferences about individuals, their locations, or their habits. The second view shows the actual occupancy counts after decryption (as this data is stored in the PULSAR secure database). The latter can be used in situations where there is a requirement to access more granular information (e.g., an emergency situation). The access to data regardless of the level of granularity is internally logged by TIPPERS so that attestation can be performed at any point.

In order to simulate a building within the NIWC Pacific campus with multiple rooms, the MCoE was divided into five areas: meeting space, visitor area, offices, machines, and kitchen. Each of these areas was represented as a zone. This zoning enables the definition of granular policies for different spaces (e.g., notify the administrator when a visitor moves to an area that is not a visitor or meeting space). The graphical user interface (GUI) of the Security Surveillance application includes a heatmap showing the occupancy data. The current implementation includes the bird's eye view of NIWC Pacific topside, with simulated data to show the estimated occupancy of each building.

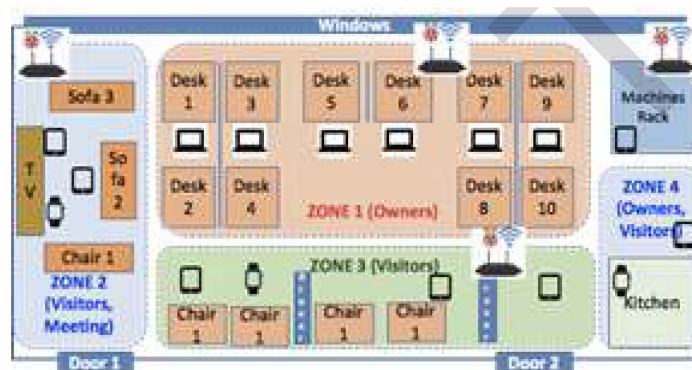


Figure 73: Mobility Center of Excellence room design.

3.4.3.4 PET Evaluation Results from Trident Warrior

In Trident Warrior, we evaluated potential benefits of the IoT technologies for naval use cases. There were a number of concerns related to privacy that emerged, from the enlisted sailors as well as the entire chain of command. Concerns over usage of the system (e.g., monitoring a sailor's whereabouts in their off-duty time) as well as data retention emerged.

Our results indicate that war fighters that are outfitted with these often-invasive technologies tend to experience a lack of privacy. Such infringement can result in adverse effects on morale, quality of life, and personnel retention. However, these same technologies bring efficiencies to command and control efforts, enhance situational awareness, and make warfighters more safe and secure.

3.4.3.5 Trident Warrior 2019 Privacy Study

An important part of the data collection during the exercise was the collection of data to infer the location of individuals. This data can be used to further compute various aggregated statistics (e.g., occupancy of spaces along time, average time spent by an individual in a particular location, average number of people an individual interacted with, etc.). Since the underlying data is private (i.e., location of individuals along time), the idea of an aggregated statistics leak of information about individual records is a precautionary concern. Well-known mechanisms can be applied like Differential Privacy (DP) to protect individual records, but knowledge on how to choose the appropriate privacy parameters is required. In this section, the different values of ϵ for a DP mechanism are estimated.

Assume a data table is displayed with user-location-time-table consisting of categories such as: user, day, location, daytime, and time spent. The categories are used to describe the amount of time the user has spent in each area per day. Let us consider, for instance, a commanding officer that observes aggregated averages on the time spent for each recorded location+daytime combination, stated as the following query:

```
SELECT day, location, daytime, AVG(timespent) FROM user-location-time-table
GROUP BY day, location, daytime;
```

The goal is to estimate how much a commander (treated here as an adversary), can learn about the particular time spent by a specific user. A quite strong attacker that may already have knowledge regarding the exact amount of time spent by other people who have been together with the victim at the same time in the same location can be assumed. This idea is motivated by the definition of differential privacy, which is aimed to protect against such attackers.

In our experiment, the attacker first fixes a single victim out of n users. He/She computes the prior assumption of the victim's data based on the data of the other $n-1$ users (or only a certain fraction of these users). He/She tries to guess the victim's spent times based on the prior he/she has already learned, and on the aggregated statistics that depend on the victim's data. The researchers assume that the attacker wins even if he/she does not guess the spent time precisely, but with some precision. For example, if the attacker says that a user has been in a room for 17 minutes, but it actually was 17.5 minutes, the guess is still considered sufficiently correct.

There are n users that participate in the experiment. For each location+daytime+day combination, each user u_i has spent times distributed according to normal distribution $N(\mu_i, \sigma_i)$. The attacker predicts μ_i and σ_i based on the data of the other $n-1$ users.

Fixing some posterior probability t , (e.g., $t = 0.9$), the researchers want to compute the precision r within which the attacker's guess stays with probability t . For example, if the actual time is x_0 , then with probability t the attacker's guess will be $x \in [x_0 - r, x_0 + r]$.

It can be assumed that an ϵ -differential privacy mechanism is applied to the released average. In particular, the sensitivity of AVG query w.r.t. attribute timespent is $1/n$, so e.g., Laplace mechanism $\text{Lap}(\lambda)$ where $\lambda = 1/(n\epsilon)$ can be used.

Using existing results on relating differential privacy to guessing advantage (e.g. [8,9]), if p_i is the prior guessing probability, then the posterior p'_i is bounded by:

$$p'_i(1 + eR\epsilon(1 - p_i)/p_i) - 1$$

where $R = \max_{x, x' \in X} d(x, x')$ is in this case the largest possible spent time. While normal distribution is unbounded, it shows as $\Pr[x - \mu \leq a] = \text{erf}(a / (\sqrt{2} \cdot \lambda))$, where erf is the error function, so e.g., for $a = 3\sqrt{2} \cdot \lambda$ shown as $\Pr[x - \mu \leq a] = \text{erf}(3) \approx 0.9998$, which essentially covers the set of possible inputs. A smaller value of a can be taken to reduce the size of the exponent, but it also reduces the attacker's search space, so this parameter can be optimized to improve the upper bound on guessing probability.

The team computes:

Then p'_i is computed from p_i and ϵ as described above. The experiments are performed for $r \in \{0.5, 1.0, 2.0, 3.0, 4.0, 5.0\}$ and $\epsilon \in \{0.1, 0.5, 0.75, 1.0, 2.0, 3.0, 4.0\}$, computing the posterior p'_i . For each ϵ , it is found that the smallest r for which $p'_i \geq t$.

Since the actual data of the users' movement on the ship collected during the exercise cannot be shared even for the privacy study, the researchers simulated the behavior of 150 users, such that the replicated data has the same statistical moments (means and standard deviations) as the actual data. This imitated data serves as the "real" data for the privacy study. Therefore, ϵ -differential privacy is applied to this dataset and the adversary's success is computed.

The posterior guessing probabilities for 5 data samples had been estimated. The results are depicted in Figure 74. The number of spent times guessed is plotted with probability $\geq 90\%$ for different precisions, where the precisions are represented with different colors. The dark green color represents the roughest guess (± 5 minutes), and the dark red color the most precise guess (± 0.5 minutes). For $\epsilon = 0.1$, only few people are depicted in the bar, and this means that for the others the guessing precision was more than ± 5 minutes.

While the datasets are different, similar trends in these five plots are noticed. If $\epsilon \geq 3$ is taken, then very little privacy guarantees are obtained, and each user's spent time may be guessed within one minute of precision. On the other hand, for $\epsilon \leq 0.5$, the guessing precision ranges between 4 and 5 minutes, which is much better, considering that the actual spent times are on average 8-9 minutes in the given datasets. There are always several people for whom the guessing probability is large even for small ϵ , as their behavior is more predictable, but there are not too many such people. Since smaller ϵ means more noise in aggregated statistics, data utility also needs to be taken into account, which would be a separate study and depends on how the statistics are actually going to be applied. Alternatively, weaker attackers could be taken, who do not know "everyone except the victim", but only some of the other users. In that case, it could be possible to get better privacy for larger values of ϵ . Modeling a particular attacker would require knowledge about the context, who the attacker is and what he already knows. This remains out of scope of this privacy study.

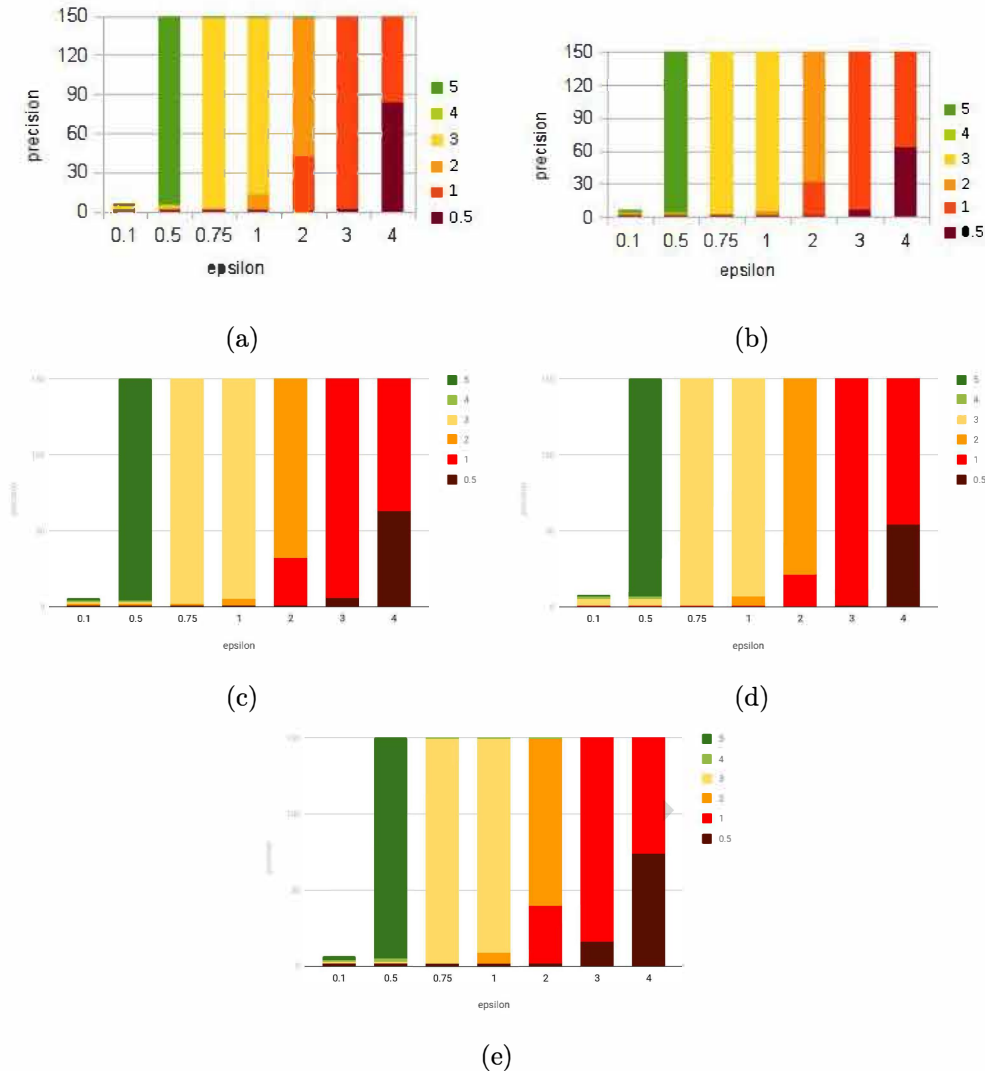


Figure 74: Results privacy study on Trident Warrior data.

3.5 Transition Efforts

In this section the outreach, technology transfer and technology transition efforts are addressed. There are two points of focus for these efforts: US Navy and COVID-19 mitigation efforts. For the US Navy, the TIPPERS system was deployed in two US Navy Trident Warrior Exercises: Trident Warrior 2019 and Trident Warrior 2020. Also, the system was tested by the Navy in their efforts to combat COVID-19.

The TIPPERS research team worked quite closely with the Naval Information Warfare Center Pacific (NIWC Pacific-formerly SPAWAR). This partnership spanned the years of the project and is ongoing today. The combined team performed a large number of testing and development projects and developed an excellent academic/military partnership. This partnership was instrumental in the efforts to transition the TIPPERS technologies to the US Navy.

In other COVID-19 mitigation efforts, the TIPPERS system was deployed at several

sites. At UC, Irvine, the TIPPERS system was deployed for the Henry Samueli School of Engineering, the Donald Bren School of Information and Computer Sciences, as well as the California Institute for Telecommunications and Information Technology. Additionally, the TIPPERS system is being deployed at Ball State University to assist with their COVID-19 efforts.

While the TIPPERS system is a large-scale information technology and information system, it is robust and deployable. The researchers on this project were able to deploy the TIPPERS system to ships in Trident Warrior 2019 as well as in Trident Warrior 2020. In both of those cases, the system was up , running and collecting data in one day and was stable for the entire exercise. Furthermore, the system is robust in that it is providing COVID-19 mitigation assistance to a number of schools, institutes, and other universities.

3.5.1 US Navy's Trident Warrior Exercises

3.5.1.1 Trident Warrior 2019

In the first week (08/12/19-08/16/19), the instrumentation of the four spaces assigned to the TIPPERS team was completed. The four spaces were: The ship's library, the weight room, the mess deck, and the ship's store. The TIPPERS system was deployed and running in one day with the remaining days of week one spent testing and verifying the system. As part of the deployment of TIPPERS in the assigned ship, the first step was to instrument the space with different IoT sensors. This required both the physical installation of the sensors, and the deployment of a network infrastructure. With respect to sensors, the following list itemizes all of the equipment deployed on the Navy ship during TW 19:

- Wi-Fi access points (2)
- Bluetooth Beacons (32)
- Power outlet meters (6)
- Raspberry Pi (4)
- Smart Card Reader (1)
- Smartphones (30)

Each sailor participating in the testing of TIPPERS during Trident Warrior 2019 was issued a smartphone. WiFi APs, Bluetooth beacons, and smart card readers were used to passively locate people in the ship (through their assigned smartphones). Additionally, smartphones are used to capture information about their integrated sensors (e.g., accelerometers and gyroscopes). Power outlet meters are used to capture information about energy utilization in the ship.

In the second week (08/19/19-08/23/19), the team executed ten different experiments that were planned. Each experiment was performed multiple times. The TIPPERS system



Figure 75: TIPPERS team with sailors.

performed flawlessly in the ten experiments that were designed by the Navy and NIWC to test the system. The experiments were:

Experiment 1: Privacy preserving activity monitoring. This scenario was used to understand sailor activity and movement within the ship and specifically focused on the mess deck. Reference points for time spent in line, time spent eating and time moving in space were collected. Data for space utilization by the individual was collected as well. Additionally, data on movement through the ship (e.g., ship's store, mess deck, gym, and library) was collected for simulation purposes.

Experiment 2: Command support. This scenario involved tasking sailors to carry out orders (e.g., movement from location to location) and provide command level feedback on progress. This scenario also provided hands-on exposure to the commanders and sailors of the TIPPERS technologies.

Experiment 3: Fall detection. This scenario was used to test the fall detection capabilities of the TIPPERS mobile client application (Figure 77) as well as the fall verification and alerting capabilities of the larger TIPPERS system. While this scenario started as a Man Overboard scenario, it soon evolved into a more generalized Fall Detection use case based on feedback (e.g., stairwell dangers, shaft alley) from Navy personnel.

Experiment 4: Physical security. In this scenario, the TIPPERS team tests physical security applications and see whether the CAC reader triggers the expected alerts. This scenario was tested in the ship's library.

Experiment 5: Space aggregation. Tests the ability of TIPPERS to provide aggregation and counts of sailors on the mess decks. The results show how many people use the mess

deck, when they use it, where they sit, and their trajectories during usage.

Experiment 6: Device management. Tests the ability of TIPPERS to capture specific characteristics of the supplied phones, both registered and non-registered. All phones used in this scenario are TIPPERS-supplied smartphones. Selected phones are unregistered and categorized as “rogue” cell phones.

Experiment 7: Energy management. Monitors energy consumption of TIPPERS equipment via the power outlet meter devices. Such devices, deployed in the ship’s library, feed the TIPPERS system with data regarding energy consumption on individual electrical circuits.

Experiment 8: Messaging and meeting scheduling. This scenario tests the TIPPERS secure messaging capability to the ship’s company. The point to point (sailor to sailor) and the point to multipoint (intercom) features were tested (see Figure 10).

Experiment 9: Activity self-awareness. This scenario has sailors carrying the TIPPERS mobile client while onboard. Then, sailors can use the CB application to monitor how many times they visited a specific area and/or how much time they spent there (e.g., how many hours did they spend in the gym in the last month).

Experiment 10: SysAdmin and privacy. In this scenario, the TIPPERS team walks through privacy leakage evaluation with Radio/IT personnel. The team solicits feedback on privacy as it relates to systems administration of TIPPERS.

These detailed experiments were designed and implemented to cater to shipboard activities, test new technologies in a military environment and facilitate a privacy study in tactical settings. There are a number of challenges in working with ship networks to execute IoT-based applications in a secure and privacy-aware manner. PETs technologies studied included policy aware data release, differential privacy and encrypted query processing. A privacy analysis on data collected during the exercise shows that differential privacy technologies applied with the appropriate privacy parameter ϵ (i.e., ϵ -differential privacy mechanisms) can be used to hide the precise time a sailor stays in a particular space while still offering some value for the analyst.

Lessons learned as a result of the technical deployment of the TIPPERS system in the TW2019 setting highlighted the important role of reliable real-time communications, privacy technologies and tools for project management. Experiences gained from Trident Warrior 2019 were used to design an enhanced deployment of TIPPERS that was deployed in Trident Warrior 2020.

3.5.1.2 Trident Warrior 2020

In the first week (10/13/20—10/16/20), the instrumentation of the four spaces assigned to the TIPPERS team was completed. The four spaces on the ship were: The ship’s library, the rec room, the mess deck, and the training office. The TIPPERS system was deployed and running in one day with the remaining days of week one spent testing and verifying the system. In the second week (10/19/20—10/23/20), the team executed seven different demonstration/tests as part of four scenarios that were planned. Each scenario was per-

formed multiple times. The TIPPERS system performed flawlessly in the four scenarios that were designed by the Navy and NIWC to test the system. The scenarios were:

1. Emergency Response: Fall Detection (Man Overboard), Fire/Flood/Damage Control (F2DC).
2. Mustering: General Quarters (GQ), Onboarding (OB) for Visitor Control
3. Situational Awareness: Watchstanding (WS)
4. Sailor Messaging and Management (SMM)



Figure 76: Sailors using TIPPERS mobile devices.

Demonstration Script 1 Scenario 1: Fall Detection Description: This scenario demonstrates the detection of sailors' fall within the ship and alerting supervisors and sailors nearby without violating their privacy.

Users: 1 Operator, 2 Dashboard Observers, 2 Experiment Observers, 4 Participants, 2 Designated Response Personnel, 4 Nearby Personnel, 2 Damage Control Central, 2 Bridge

Locations: TIPPERS Observation — Library: Zone L1 Experiment — Crew Rec Room: Zones C1, C2, C3, C4 Designated Response Personnel — Library: Zone L4 Nearby Personnel

— Crew Rec Room: Zone C5, C6, Training Room Zone: Zone T1, T2 Damage Control Central — Mess Decks: Zone 1 Bridge: — Mess Decks: Zone 10

Log: Emergency Response Log

Demonstration Script:

Experiment

- At time t1, all participants and observers go to their assigned zones.
- At time t2, the 2 Experiment Observers record the locations of the 4 Participants in the Log.
- At time t3, the 4 participants start the experiment. They let their devices fall down on the cover as directed.
- The 4 participants pick up their devices and show the 2 Experiment Observers.
- The 2 Experiment Observers record the following into the Log: [Participant ID, Time, Location, Fall Detected or Fall Not Detected].
- After the 2 Designated Response Personnel arrive at the location of the fall, the 2 Experiment Observers record the following from the 2 devices: [Designated Response Personnel, Time, Location].

TIPPERS Dashboard

- At time t1, the 2 Dashboard Observers at the TIPPERS Dashboard start their observations.
- At time t2, the 2 Dashboard Observers record the locations of all participants as seeing on the Map in the Dashboard.
- Whenever an alert comes in, they record the following data points: Participant ID, Time, Location, Fall Detected or Fall Not Detected.
- They also check if the alert was forwarded to all parties: Alert Forwarded: YES or NO, Time, Recipients].

Alerting Others

- At time t1, the following participants go to their designated locations: 2 Designated Response Personnel, 4 Nearby Personnel, 2 Damage Control Central, 2 Bridge.
- At time t2, they start their observations.
- Whenever an alert comes in, they record the following data points: Alert Type: Fall Alert, Participant ID, Time, Location].

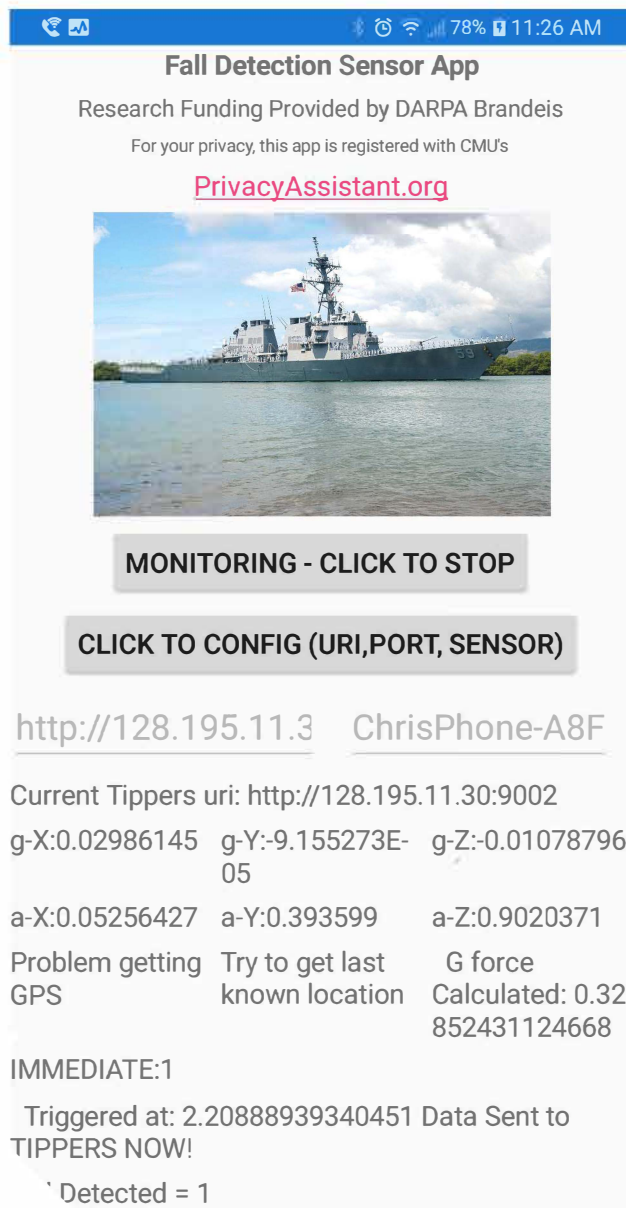


Figure 77: TIPPERS Mobile Fall Detection application deployed on ship

- The two Designated Response Personnel move to the location of the fall and the following alert: Designated Response Personnel, Time, Location].

Demonstration Script 2 Scenario 2: Fire/Flood/Damage Control (F2DC) Description: This scenario demonstrates the mechanisms for tracking sailors when they are in a fire, flood, or damage control emergency situation while preserving their privacy.

Users: 1 Operator, 2 Dashboard Observers, 4 Experiment Observers, 2 Participants, 2 Designated Response Personnel, 10 Nearby Personnel, 5 Damage Control Central, 1 Bridge

Locations: TIPPERS Observation Library: Zone L1 Experiment Crew Rec Room: Zones

C1 Designated Response Personnel Library: Zone L4 Nearby Personnel Crew Rec Room: Zone C5, C6, Training Room Zone: Zone T1, T2 Damage Control Central Mess Decks: Zone 1 Bridge: Mess Decks: Zone 8

Log: Emergency Response Log

Demonstration Script: Experiment

- At time t1, all participants and observers go to their assigned zones.
- At time t2, the 2 Experiment Observers record the participants locations in the Log.
- At time t3, the 2 Participants send Fire Alert to the TIPPERS system.
- The 2 Experiment Observers record the following from the Participants devices: Alert Type: Fire Alert, Time, Location.

TIPPERS Dashboard

- At time t1, the 2 Dashboard Observers start their observations.
- At time t2, the 2 Dashboard Observers record the locations of all participants as seeing on the Map in the Dashboard.
- Whenever an alert comes in, the 2 Dashboard Observers record the following data points: [Participant ID, Fire Alert, Time, Location].
- They also record from the Dashboard if the alert was forwarded to all parties including the Nearby Personnel: [Alert Forwarded: YES or NO, Time, Recipients].

Damage Control Central

- At time t4, they start their observations.
- If they receive a Fire Alert, they record the following data points: [Participant ID, Fire Alert, Time, Location].
- At time t5, the 2 Designated Response Personnel send the following message to TIPPERS: [Message Type: Designated Response Personnel responding, Time, Location]
- Then, they record the following from their devices: [Message Type: Designated Response Personnel responding, Time, Location]
- At time t6, they leave.

Bridge

- At time t2, the Experiment Observer at the Bridge starts the observation.
- Whenever an alert comes in, the Experiment Observer at the Bridge records the following data points: [Participant ID, Fire Alert, Time, Location].

Nearby Personnel

- Whenever the Nearby Personnel receive a fire alert, they send the following message to TIPPERS: [Participant ID, Fire Alert Received, Time, Location].
- The 2 Experiment Observers record the following: [Participant ID, Fire Alert Received, Time, Location].

Experiment: Containing the Fire

- After arriving at the fire location, the 2 Designated Response Personnel send the following message to TIPPERS: [Message Type: Designated Response Personnel arrived, Time, Location].
- After arriving at the fire location, the 2 Designated Response Personnel send the following message to TIPPERS: [Fire Contained, Time, Location].

Demonstration Script 3 Scenario 2: General Quarter (GQ)

Description: This scenario demonstrates the automation of general quarters monitoring while preserving the privacy of the sailors.

Users: 1 Operator, 2 Dashboard Observers, 3 Experiment Observers, 15 Participants, 1 GQ Station Team Leader, 1 Damage Control Central, 1 Bridge

Locations: TIPPERS Observation — Library: Zone L1 Experiment — 4 Compartments GQ Station Team Leader — Library: Zone L4 Damage Control Central — Crew Rec Room: Zone 1 Bridge — Training Room: Zone 1

Assignments Group A (5 participants): Start Station: Mess Decks, Start Time: xx:xx — End Station: Library, End Time: xx:xx Group B (5 participants): Start Station: Training Room, Start Time: xx:xx — End Station: Library, End Time: xx:xx Group C (5 participants): Start Station: Crew Rec Room, Start Time: xx:xx — End Station: Library, End Time: xx:xx

Demonstration Script: Experiment

- At time t1, all participants and observers go to their assigned zones.
- In each starting location, the Experiment Observer records the following: [Participant ID, Location, Time]
- At time t3, after GQ Station Team Leader announces GQ, all participants start moving.
- The Experiment Observer recording the following: [Start Time]
- When participants arrive at destination, send the following message to TIPPERS. Message: [Participant ID, Reporting GQ, Time]

TIPPERS Dashboard

- At time t1, the 2 Dashboard Observers start their observations.

- At time t2, the 2 Dashboard Observers record the locations of all participants as seeing on the Map in the Dashboard.

GQ Station

- At time t2, GQ Station Team Leader announces GQ by broadcasting a message to everyone: [GQ start at time: xx:xx]
- The Experiment Observer records the data point from the GQ Station Team Leader device: [Alert: GQ, Time: xx:xx, Location]
- When participants arrive at their destinations and send the Reporting GQ message, the Experiments Observer records the confirmation message: [GQ Notification Received, Time: xx:xx, Location]

Demonstration Script 4 Scenario 2: Onboarding (OB) for Visitor Control

Description: This scenario demonstrates the automation of onboarding and offboarding of visitors on the ship.

Users: 1 Operator, 2 Dashboard Observers, 4 Experiment Observers, 15 Participants (Visitors), 1 GQ Station Team Leader, 1 Damage Control Central, 1 Bridge

Visitor Info: Visitor Name, Organization, Security Clearance Level, Authorized Ship-board Destination(s), Expected Completion Time Violators: 11, Visitor Violators: 4

Locations: TIPPERS Observation — Library: Zone L1 Experiment — 4 Compartments Restricted Areas to Visitors: Training Room Zone 6, Crew Rec Room Zone 2

GQ Station Team Leader — Library: Zone L4 Damage Control Central — Crew Rec Room: Zone 1 Bridge: — Training Room: Zone 1

Demonstration Script: Experiment

- At time t1, participants (Visitors) start checking into the simulated entrance of the ship (Library). Each Visitor will have a phone or watch.
- The Experiment Observer at the Library records the following data points: [Visitor ID, Time Check-in, Location, Time to Check-out].
- After checking-in, the Visitors go to the other locations as directed.
- The Violators go to the restricted areas.
- TIPPERS system detects the violations and sends the following alert to each violator: [Visitor ID, Time, Location, Accessing Restricted Area]
- At time t2 (end visit time), visitors start checking out.
- Whenever a Visitor checks-out, the Experiment Observer records the following data points from the message displayed on the Dashboard: [Visitor ID, Time Check-out, Location, Time Check-out].

TIPPERS Dashboard

- At time t1, the 2 Dashboard Observers start their observations.
- Whenever a Visitor checks-in, they record the following data points from the message displayed on the Dashboard: [Visitor ID, Time Check-in, Location, Time Check-out].
- Whenever a Visitor Violation Alert is received, the 2 Dashboard Observers record the following: [Alert Type: Visitor Violation Alert, Location, Time]
- At time t2 (end visit time), visitors start checking out.
- Whenever a Visitor checks-out, the 2 Dashboard Observers record the following data points from the message displayed on the Dashboard: [Visitor ID, Time Check-out, Location, Time Check-out].

Demonstration Script 5 Scenario 4: Inventory Management (IM)

Description: This scenario demonstrates the management of tools and parts used for the maintenance of the ship.

Users: 1 Operator, 2 Dashboard Observers, 4 Experiment Observers, 15 Participants (Visitors), 1 Inventory Manager

Tools Users: Name, ID, Workcenter Tools: Inventory ID, Type, Number in Inventory, Number Available

Locations: TIPPERS Observation — Library: Zone L1 Experiment — Library, Crew Rec Room Workcenter — Crew Rec Room Zones Tool User 1 — R1, Tool User 2 — R2, Tool User 3 — R3, Tool User 4 — R4, Tool User 5 — R5, Tool User 6 — R6

Demonstration Script: Experiment

- At time t1, participants (Tool Users) go to their assigned workcenters.
- The Experiment Observers enter the following data into the Log: [Participant ID, Location, Time]
- At time t2, each Tool User goes to the Storage to check out the approved tools.
- At time t2, all Tool Users return their tools on time except the 2 Tool Violators.
- After the time expired, TIPPERS sends an Tool Missing Alert to the Inventory Manager.
- The Inventory Manager records the following: [Last User ID, Tool ID, Last Time, Last Location]

TIPPERS Dashboard

- At time t1, the 2 Dashboard Observers start their observations.

- The 2 Dashboard Observers conform the locations of all Tool Users and Tools from the map.
- After TIPPERS sends an Tool Missing Alert to the Inventory Manager, the 2 Dashboard Observers record the following from Dashboard: [Last User ID, Tool ID, Last Time, Last Location]

Demonstration Script 6 Scenario 3: Watchstanding (WS)

Description: This scenario demonstrates how the watchstanding activities can be monitored automatically.

Users: 1 Operator, 2 Dashboard Observers, 4 Experiment Observers, 2 Participants, 1 Watch Supervisor

Locations: TIPPERS Observation Library: Zone L1 Experiment 4 Compartments

Watchstanding Route: Library Zone L2 — Mesh Desks Zone M2, M4, M8 — Training room Zone T1, T3, T6 — Crew Rec Room Zones R1, R3, R6 — Library Zone L4

Log: Situational Awareness Log

Demonstration Script: Experiment

- At time t1, Watchstanding sends the first check point message to TIPPERS and records in the Log, the notification message displayed: [Watchstanding ID, Check Point Location, Time]
- At time t2, Watchstanding start moving.
- Whenever the Watchstanding reaches a check point, she or he sends a check point message to TIPPERS and records in the Log the notification message displayed: [Watchstanding ID, Check Point Location, Time]

TIPPERS Dashboard

- At time t1, the Operator shows the Watchstanding Route on the map.
- The 2 Dashboard Observers record the following: [Route, List Check Points, List Times]
- Whenever the Watchstanding reaches a Check Point, through the route, the 2 Dashboard Observers record the followings: [Check Point, Time]
- The 2 Dashboard Observers records how often the Dashboard updates the watchstander track updated (sec). [Track Update Frequency]
- The 2 Dashboard Observers check and record if TIPPERS is registering the Watchstanding at each checkpoint. [Watchstanding registration at check points]
- If the Watchstanding is too late or too early to a checkpoint, TIPPERS sends a Watchstanding Violation Alert to Watch Supervisor. [Watchstanding ID, Check Point Location, Time Difference]

- The 2 Dashboard Observers record in Log: [Watchstanding ID, Check Point Location, Time Difference]

Watch Supervisor

- Whenever the Watch Supervisor receives Watchstanding Alert, she or he records the following: [Watchstanding ID, Watchstanding Alert, Time]

Demonstration Script 7 Scenario 4: Sailor Messaging and Management (SMM)

Description: This scenario demonstrates how the Messaging App is used by the sailors to communicate as well as the management on the ship.

Users: 1 Operator, 2 Dashboard Observers, 4 Experiment Observers, 10 Participants, 2 Division Supervisors

Locations: TIPPERS Observation — Library: Zone L1 Experiment — 4 Compartments

Division 1: 5 Participants, 1 Experiment Observer — Training Room Division 2: 5 Participants, 1 Experiment Observer — Crew Rec Room 2 Division Supervisors, 1 Experiment Observer — Library

Demonstration Script: Supervisor Communication Experiment

- The participants send back receipt acknowledgements.
- The Experiment Observers record the following: [Division 1, Time, Location]
- After completing the tasks, the participants send a message to Division 1 Supervisor.
: [Participant ID, Time, Message]
- Repeat these steps for Division 2.

TIPPERS Dashboard

- At time t1, the 2 Dashboard Observers start their observations.
- Division 1 Supervisor sends a message to his or her 5 sailors to go to M8 to perform a task.
- The Experiment Observer records the following: [Division 1 Supervisor, Time, Location, Message]
- Division 1 Supervisor observes the participants in the location where they are performing the tasks.
- Repeat these steps for Division 2.

Sailor-to-Sailor Communication

- Participant 1 in Division sends a text message to Participant 6 in Division 2.

- The Experiment Observer in Division 1 records the following from Participant 1 device: [sender, receiver, location sender, time]
- If message received, the Experiment Observer in Division 2 record the following from Participant 2 device: [sender, receiver, location receiver, time]

The ship's leadership was supportive of the demonstrations. The TIPPERS team was assigned 10 to 20 sailors daily to help execute the scenarios. While the team was not able to have a VIP demonstration due to COVID-19, a number of ship leaders participated or observed the demonstrations and filled out the survey. The TW20 management was also very supportive of the TIPPERS system and the demonstrations.

3.5.2 COVID-19 Mitigation

US Navy Deployment (COVID-19). As a direct result of the success of TIPPERS in the Trident Warrior Exercise, US Navy leadership selected the system to be tested for their COVID-19 mitigation strategies. The testing and demonstration of the TIPPERS system was conducted in two parts: June 27, 2020 and July 6, 2020. The testing and demonstrations were performed at Pier in Naval Base, San-Diego. The following scenarios were tested:

1. Contact Tracing
2. Hotspot Exploration
3. Social Distancing
4. Quarantine and Restricted Access to Infected Areas App

The following is the testing scenarios designed by the Navy. All web apps will be developed for desktops, laptops, or large tablets. Users

- Data Generators – Sailors (iphone apps or watches)
- Data Consumers – Medical Personnel (laptop or tablet)

1- Contact Tracing User: Medical Personnel Scenario 1: Tracing an infected person

- Select the Name of the Infected person from a drop-down.
- Select the date of the test.
- Select the infection date window.
- Submit the request.
- A list of contacted people with their info within the time window.



Figure 78: Sailors testing Covid-19 mitigation with TIPPERS.

- From the list, click on the name of the person to get more information.
- Details Window displays
 - A list (Location, Time, Duration)
 - A map visualizing this info.

Scenario 2: Display Trajectory of a Infected Person

- Enter the name of infected person to select the name from the drop-down.
- Select the date of the test.
- Select the infection window.
- Submit.
- Map appears with the trajectory of the infected person throughout the spaces within the time window.

2- Hotspot Exploration Scenario 1: Display a heatmap of all contaminated areas within a specified time window.

- Open tab/app.
- Show map of space with heatmap overlaid.
- User zooms into a specific space to see the heatmap for that space.
- Mouse hover shows popup menu with info.

3- Social Distancing Same scenario and UI as Contact Tracing application, but includes a dropdown with degrees of separation of contact from person who tested positive.

4- Quarantine and Restricted Access to Infected Areas App Live notification to sailor and admin when the sailor tries to enter a restricted space.

On June 27, 2020, the TIPPERS team was able to successfully complete 3 out of 4 demonstration scripts (10 scenarios) in the plan (Bubble Transfer, Access Restriction to Quarantine/Contaminated Areas, and Contaminated Areas Tracing). There were some difficulties performing the last demonstration script (COVID-19 Contact Tracing - less than 6 feet for 2 minutes). To complete the demonstration, the team agreed with the Trident Warrior team to perform this remaining demonstration script on July, 6 2020. At that time, TIPPERS performed with 100 percent success.

UCI Computer Science Deployment (COVID-19). The Bren School of ICS is the first non-military deployment of TIPPERS for COVID-19 mitigation. A dashboard was created that shows the occupancy counts of the buildings within the school. Additionally, a user can drill down into the floor or any area of interest. In addition to occupancy, social distancing adherence and crowd flow can be displayed. All is done in a privacy preserving, non-invasive fashion.

UCI Engineering Deployment (COVID-19). The Samueli school was the next adopter. Similar functionality was provided for this school.

UCI CalIT2 Deployment (COVID-19). The CalIT2 institute also adopted TIPPERS technology for COVID-19 mitigation. Similar functionality was provided for the institute.

BSU Deployment (COVID-19). As of the time of writing this report, TIPPERS is approximately 75 percent implemented at Ball State University. The TIPPERS system is receiving data from BSU's library with their Rec Center on the list as well. These are two important areas of interest for the University that receive high traffic and occupancy. Similar functionality (dashboard, etc.) will be provided to them.

4 Results and Discussion

The key results of the TIPPERS project are summarized below.

TIPPERS introduced a new layered approach to represent information in sensor-driven smart spaces that allows data to be modeled at the raw sensor level, as well as, at the

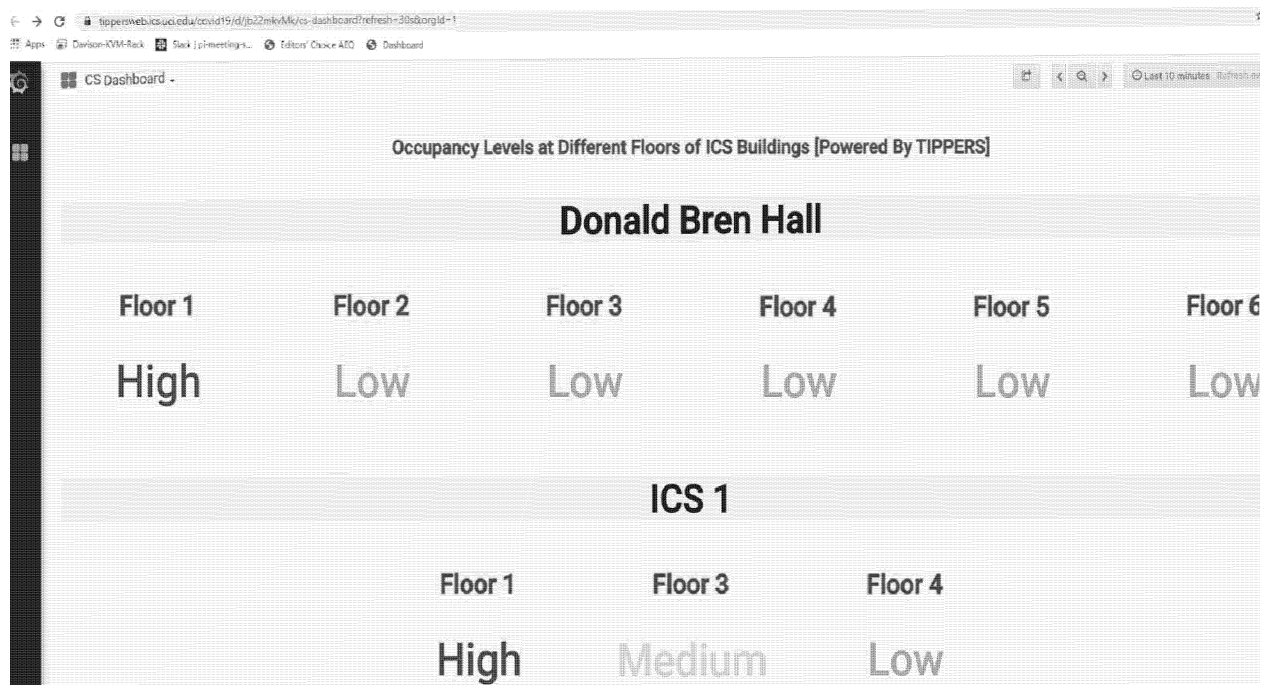


Figure 79: UCI Computer Science COVID-19 Dashboard.

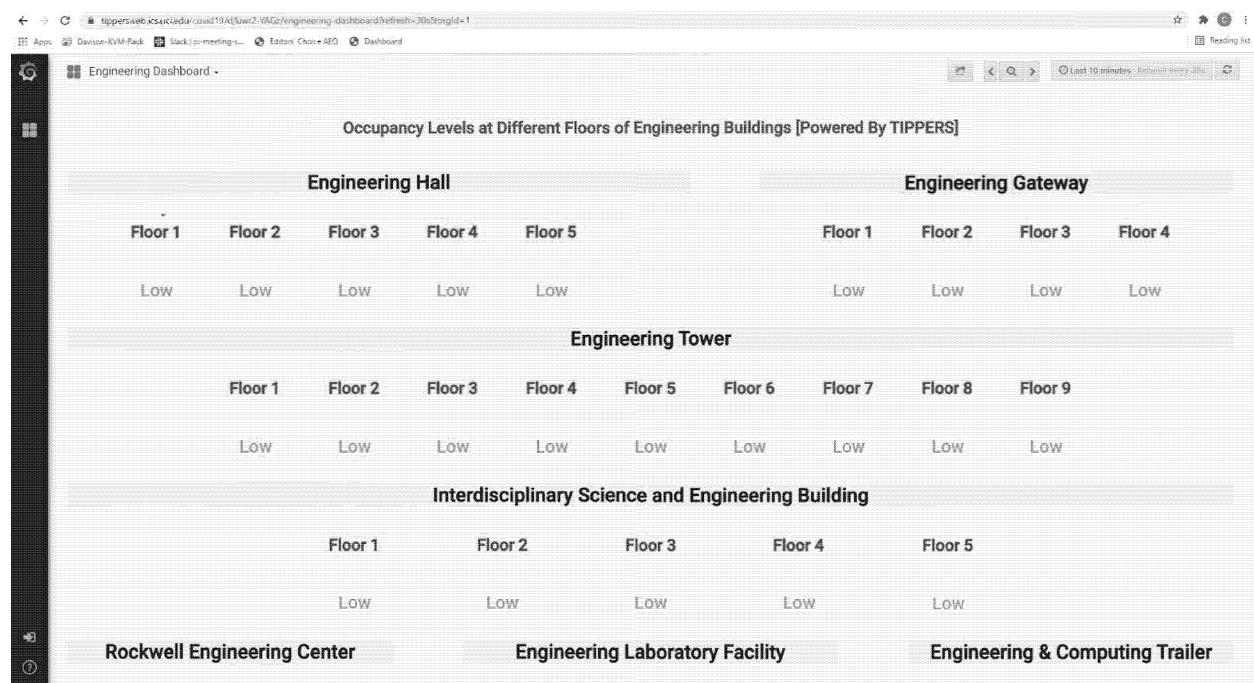


Figure 80: UCI Engineering COVID-19 Dashboard.

semantically enriched level by dynamically interpreting sensor data as it arrives. Semantic representation makes both the task of specification and reasoning with privacy policies, as well as, application development easier. The semantic model led to the creation of a

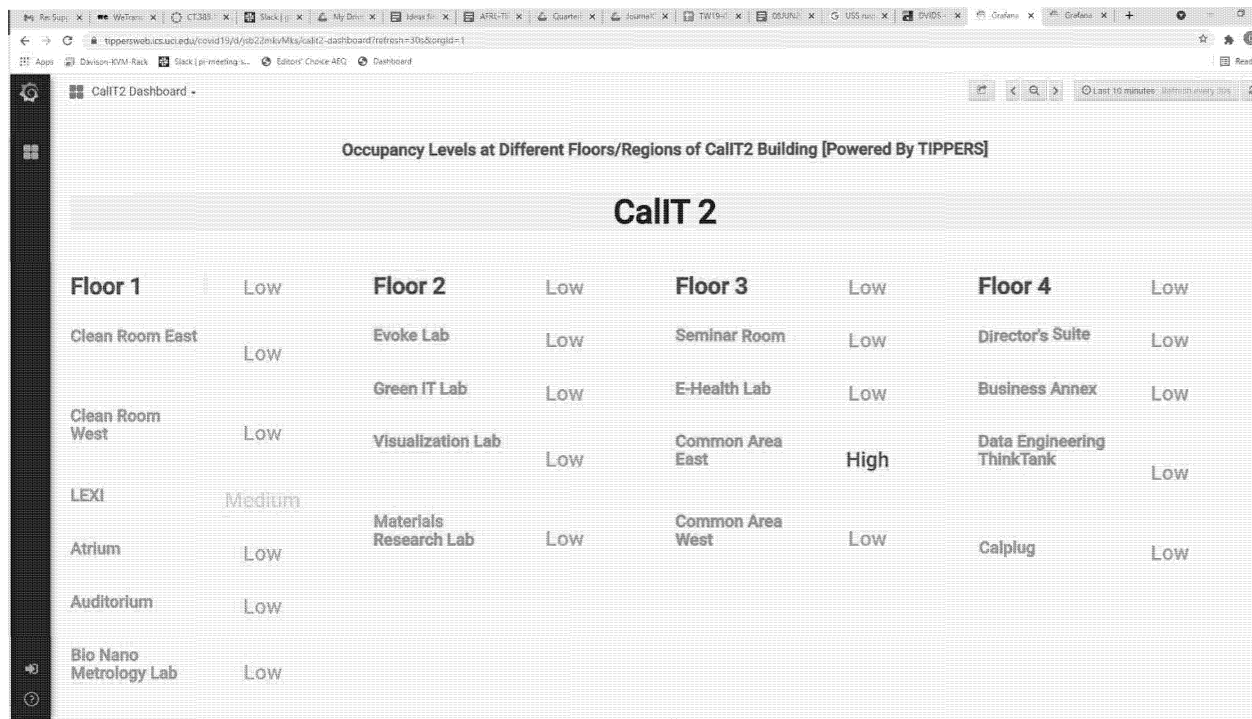


Figure 81: CalIT2 COVID-19 Dashboard.

novel data management technology that embodies “privacy-by-design” principle by creating a policy-based approach to data management wherein all aspects of data flow – from creation, ingestion, processing, sharing, storage, and retention are controlled by potentially fine-grained policies. The database system implemented supports plug-n-play mechanisms to integrate a variety of privacy technologies including differential privacy and secure storage. Furthermore it supported a policy based PE-IoT framework that allows data streams to be intercepted and for diverse privacy technologies (differential privacy, encryption, anonymization, randomization, etc.) to be applied to data flows in the context of sharing.

This work on data management initiated in TIPPERS has led to several new projects funded by National Science Foundation (NSF) including a project entitled EnrichDB where the project’s goal is to develop a new type of data management technology that allows data enrichment to be seamlessly integrated into query processing and data analysis. It also helped create additional new projects entitled SWADE and CAREDEX funded by NSF and SparX by University of California that uses the underlying sensor data management technology of EnrichDB to build systems to allow real-time analysis of water systems, senior living facilities, and monitoring prescribed fires respectively.

Another key result of TIPPERS was the development of an end-to-end testbed at UCI that instrumented part of the campus - over 30 buildings - with sensors and sensor processing mechanisms to create a digital representation of activities in the buildings. Such activities include people in the building, their location, ongoing events, attendance by people in the events, etc. In addition, the testbed included a diverse set of end-user application such

as finding colleagues, analysing building usage, analyzing individual's interaction with the building and/or other users, support for contextualised messages etc.

Through the plug-n-play interface, diverse privacy technologies were integrated into the testbed including Pegasus differential privacy technology to hide information about individual's short term and longer term behavior, Stealth's Pulsar System and Galois's Jana system for secure data storage and processing, and the privacy assistant and privacy registry technologies developed by CMU for policy notification and preference selection. Integration of these technologies allowed for a detailed privacy analysis of different privacy technologies in supporting location privacy using the UCI testbed.

In addition to serving as a testbed for studying efficacy of privacy technologies, one of the key outcomes of TIPPERS was the identification of limitations of existing privacy/secure computing technologies when it comes to at-scale deployments. It also allowed the team to explore novel directions to overcome such limitations. In particular, we developed SEMIOTIC framework to support semantic abstraction of data in databases in order to specify privacy policies for sensor data that can be automatically translated and enforced by the system. Formalization of a challenge of preventing data leakage through inferences when policies are specified at the semantic level of abstraction and techniques to prevent such leakage were developed.

Several new techniques to scale performance of secure data management were developed including development of PANDA framework to support partitioned computation over sensitive and non-sensitive data in order to scale secure data management solutions to very large data sets, development of OBSCURE and PRISM frameworks to support efficient evaluation of verifiable aggregation queries and private set operations (intersection and union) using secret sharing, development of CONCEALER and CRYSTAL frameworks to outsource dynamic sensor data and compute both aggregation and SQL queries, and development of SIEVE infrastructure to scale database policy enforcement to large number (millions) of fine-grained policies prevalent in large scale smart space applications.

New techniques to scale differential privacy to be applied to smartspace settings were also developed. These included a new model for one-sided differential privacy (OSDP) that exploits partial sensitivity of data to support increased utility (by allowing more data to be shared) while ensuring strict privacy properties on sensitive data, and extension of differential privacy framework to support minimally invasive monitoring that allows analyst to explore data at different levels of invasiveness based on the needs.

New compliance technologies to build trust amongst subjects in the sensor-based smart spaces were developed. In particular, IOT NOTARY framework was designed and implemented to verify compliance to organizational data capture policies, IOT EXPUNGE protocol was developed to verify the data deletion against the user's data retention policies, and CANOPY algorithm was developed for preventing user privacy in a smart home by exploiting round-robin style dataflow algorithms.

A key outcome of TIPPERS was transition and deployment of the technologies to the US Navy and to other organizations (where it was a recipient of the NAVWAR 2021 Innovation Award)) and to other schools such as BSU and to Honeywell Labs.

5 Conclusion

The goal of the TIPPERS project was to create an IoT smart space testbed with a plug-n-play architecture to infuse privacy enhancing technologies in order to explore their effectiveness in protecting users' privacy while still supporting a diverse set of smart space applications. Privacy technologies included specification and enforcement of privacy policies, use of differential privacy in both collecting and sharing data, and secure computation using cryptographic approaches including multi-party computation and secret sharing.

As part of the project, we created a novel data management technology that supported integration of privacy technologies, and used the system to create a testbed and a set of applications of everyday use that allowed us to explore use of smart space technologies at UCL. The modular and portable design of the system enabled us to deploy the system at other organizations including transition it to the US Navy and to participate working alongside researchers at NIWC in several naval exercises.

Our research, development, and deployment efforts led to several observations about the existing state of the privacy technologies, identification of gaps in existing technologies and new directions of exploration of technologies (some of which are discussed in the following section as part of suggested future research directions). In deploying TIPPERS system at scale, among the first observations we made was that continuous monitoring of human in-interactions with each other and with the environment in smart spaces can lead to significant leakage of data that might be deemed as sensitive, even when the sensors used to monitor activities do not directly monitor personally identifying information about individuals. Such leakage includes detailed fine-grained location of people which can lead to inferences about their habits, health, as well as work ethics and performance. While continuous monitoring can lead to novel applications of significant benefit to individuals and organizations, the aforementioned privacy risks pose a significant challenge to technology adoption.

Our second important observation was that deploying privacy enhancing technologies to real-life testbed opens several important challenges related to deployment and scale. As an example, while significant progress has been made over the past two decades of design of homomorphic techniques that allow computation to be supported on the encrypted domain, both encryption and secret sharing techniques (that offer information theoretic security) do not scale to large data sets, support limited operations, and making such systems to work at scale require a delicate balance between security and performance. The same is also true for privacy technologies such as differential privacy. Such techniques that offer provable privacy guarantees do so by limiting the utility of the data. Making such methods useful to real world applications requires systems to support mechanisms to explore the tradeoffs between privacy and utility. In the area of IoT and smart spaces, where monitoring and situational awareness are key components of system design, such a challenge becomes even more complex since such systems depend upon automated detection of events using sensor data and data misclassification as a result of noise added to ensure privacy can prevent the system from meeting its goals. Adopting differential privacy in such setting, thus, requires rethinking how it should be integrated into the system. The approach to minimally invasive monitoring we developed as part of the TIPPERS project is an attempt in such a direction.

Another key outcome of the TIPPERS project was identification of challenges that arise due to the heterogeneity of sensor technologies deployed in building smart spaces. The diversity and multiplicity of sensor devices that are deployed and implicit inferences and correlations amongst the data collected from these devices make reasoning about privacy complex. TIPPERS pioneered ways to express privacy policies and requirements in smart spaces at a semantic level which can then be translated to device level policies. Such an approach enables privacy reasoning to be decoupled from the specific analysis of device data. As the implicit inferences and correlations hidden in sensor data is unravelled, the system can automatically adjust to new (possibly more conservative) policies and techniques to ensure privacy.

The key achievement of TIPPERS was its successful transition to US Navy through joint participation in naval exercises and also in the demonstration of a TIPPERS based system for COVID monitoring. This achievement was recognized through the NAVWAR Innovation award in 2021. TIPPERS has also been ported and made available to campuses other than University of California, Irvine. Discussions are ongoing to deploy TIPPERS based smart space applications in several other universities and organizations in the near future.

6 Suggested Further Research

Our work on TIPPERS opened several new directions of research in both system design and privacy and security technologies which we believe should be fertile grounds of further exploration. We list several of them below.

6.1 Policy Compliance in Big Data Ecosystems

TIPPERS project has clearly established that smart space and IoT applications applications collect users' access and mobility information, the processing and sharing of which may violate the users' rights for privacy and consumers' protections. Such data is subject to policies such as General Data Protection Regulation (GDPR) in the European Union and similar regulations and laws in other parts of the world, such as the California Consumer Privacy Act (CCPA). Such regulations include various aspects that influence how data is collected, processed, stored, analyzed and shared. For example, one requirement is to support "the right to be forgotten", which enables users to delete all data that corresponds to them. This requires building functionality that would allow access to all data that corresponds to a user either directly or indirectly which entails additional meta-data and indexing structures. Building systems that are compliant with policies such as GDPR from scratch is not practical for many existing systems. An alternate is to extend existing data management ecosystems to make them GDPR-compliant. The PE-IoT architecture (that intercepts flow of data from sensors to enforce organizational and subject policies) coupled with the secure logging service (that we built as part of the IoT notary mechanism) which can serve as a building block for policy compliance provide an interesting opportunity to build systems in which compliance to regulations such as GDPR can be retrofitted.

6.2 Understanding the Implications of Data Leakage

In building TIPPERS using state-of-the-art secure data management solutions, it was clear that given orders of magnitude overhead of cryptographic techniques that provide guaranteed security, such systems choose to support several cryptographic primitives that offer varied levels of security thereby offering users a choice between performance and security. In particular, several systems have explored weaker encryption techniques that may leak certain information such as output size, properties such as ordering in data, access patterns, etc. While individually researchers have explored what different techniques leak, the implication of choosing cross cryptographic approaches to represent data and to use such techniques in computing results to the query have not been systematically understood. While there exists a series of papers that highlight dangers of mixing cryptographic solutions, techniques to safely compose cross cryptographic solutions leading to a theoretical framework to reason about their security, and mechanisms to systematically explore security guarantees and performance in a system that exploits multiple cryptographic primitives has not been explored. The work on secure normal form in TIPPERS provides a starting point for such an exploration but advancing the existing research to the level that it can become the basis of system design with provable security guarantees requires significant further research.

6.3 Scaling Cryptographic & Secret Sharing based Approaches

One of the key insights that allowed TIPPERS to scale secure data processing was the observation that in a large number of application domains, such as smart spaces, not all data collected is sensitive. In fact, often sensitive data comprises of only a fraction of data that may be collected. TIPPERS explored novel ideas of partitioned computing wherein data and computation is split into sensitive (that exploits cryptographic techniques) and non-sensitive (that stores and executes in plaintext). A notion of partitioned security that prevents such a partitioned computation to leak data was designed and our experimental results show orders of magnitude improvements as a result of exploiting such an approach. Nonetheless, the approach, to realize its full potential has several unanswered questions – e.g., automated ways to identify sensitivity of the data, expanding the approach to several levels of sensitivity, exploiting secure hardware, etc. Most importantly, it opens an opportunity to design and develop a practical new data management technology based on the theoretical underpinnings of exploiting partial sensitivity of data developed in the context of TIPPERS. Likewise, TIPPERS explored novel ways to exploiting secret sharing based ways to support certain database operations that provide an opportunity to explore a cloud-based data management service based on secret sharing.

6.4 Embedding Data Enrichment in Databases

A key factor in the success of TIPPERS project was the early realization by the team that a successful creation of a system and a testbed that serves as an integrator of privacy technologies in the domain of smart spaces required a new approach to data management

that abstracts sensor data into semantically meaningful observations. Such an approach could lead to understanding and reasoning about privacy policies and setting up privacy requirements from individual's perspective which can then be translated to lower level sensor policies. While our objective behind such an abstraction was initially dictated by our desire to reason about privacy, it became self evident that such an abstraction leads to a much more semantically enriched representation of data making the task of application programming significantly simpler. In TIPPERS, a developer can program new functionalities and applications purely dealing with semantic concepts without having to explicitly program and reason with sensors (which is hidden by the underlying system). Such high level programs are then automatically translated into sensor level code to realize the functionality. While TIPPERS has clearly established the power of such an abstraction both from the perspective of software development and reasoning about privacy, it has opened a need and opportunity for a new type of data management system that seamlessly integrates such data abstraction achieved through interpreting/enriching lower level sensor data to create higher level observations. This work is the genesis of EnrichDB now funded in part by NSF and exploration of the data management technology in domains ranging from smart spaces to IoT and cyber physical systems for smart water management, prescribed fire, and for smart elderly shared living facilities.

6.5 Minimally Invasive Monitoring

Our work on applying differential privacy in decision support applications led us to a radically new concept of minimally invasive monitoring (MIM). As discussed earlier in this report, traditionally, privacy-preserving data sharing has been designed for situations where data is collected and shared in aggregate form, either as a synthetic dataset generated based on the original data, or in the form of query answers. Differential privacy based approaches focus on providing formal privacy guarantees (in the form of a privacy parameter) but do not provide guarantees on the quality of data outputted. Such approaches are not suitable for decision support (DS) tasks that require guarantees on the quality of the output, specially, for false negatives, which may prevent, for instance, timely actions/interventions. The MIM approach builds a new framework that strikes an optimal continuum between the needs of the decision support applications and privacy. A MIM framework can, thus, be viewed as a progressively invasive system that explores data in the context of a monitoring task through a coarse filter with a high level of privacy, but explores the data using a finer filter more invasively only if it passes through the coarse filter. Such a MIM infrastructure is reminiscent to a degree of the way modern law enforcement has evolved over time wherein one uses a series of (explicit/ implicit) filters to determine whether to explore an incident/suspect progressively more carefully (in MIM context more invasively, i.e., with less privacy) based on evidence collected during the prior steps of the investigation (in MIM context, if object classifies as a positive in the previous tests). We can, thus, view MIM as a software system counterpart of the age-old practice of law enforcement that has evolved over time. Our work on MIM in the context of TIPPERS has established the viability of the idea which provides enough evidence for a full scale exploration of the idea in building decision support systems that respect the privacy requirements of individuals.

7.0 Human Research Protection Official (HRPO) Review

In accordance with AF Human Research Protection Official (HRPO) review requirements per DoDI3216.02_AFI40-402 "Protection of Human Subjects and Adherence to Ethical Standards in Air Force supported Research", the Institutional Review Board's (IRB) and HRPO has reviewed and concured with a Category 2 Exempt Determination for the Protocol Titled, Privacy Cognizant IoT Environment with Protocol Number FWR20150161X for work performed under this program.

References

- [1] SemIoTic, 2019.
- [2] A. Haller et al. The modular ssn ontology: A joint w3c and ogc standard specifying the semantics of sensors, observations, sampling, and actuation. *Semantic Web*, 10:9–32, 2018.
- [3] I. V. Papaioannou et al. A QoS ontology language for Web-services. In *AINA*, 2006.
- [4] E. Sirin et al. Pellet: A practical owl-dl reasoner. *Journal of Web Semantics*, 5:51–53, 2007.
- [5] S. Almanee et al. Semiotic: Bridging the semantic gap in iot spaces. In *BuildSys*, 2019.
- [6] G. Bouloukakis et al. Automated synthesis of mediators for middleware-layer protocol interoperability in the iot. *FGCS*, 2019.
- [7] Yang Wang and Alfred Kobsa. Privacy-enhancing technologies. *Social and Organizational Liabilities in Information Security*, 2008.
- [8] Michael Benisch, Patrick Gage Kelley, Norman Sadeh, and Lorrie Faith Cranor. Capturing location-privacy preferences: quantifying accuracy and user-burden tradeoffs. *Personal and Ubiquitous Computing*, 2011.
- [9] Bin Liu, Mads Schaarup Andersen, Florian Schaub, Hazim Almuhammedi, Shikun (Aerin) Zhang, Norman Sadeh, Yuvraj Agarwal, and Alessandro Acquisti. Follow My Recommendations: A Personalized Privacy Assistant for Mobile App Permissions. In *12th Symposium on Usable Privacy and Security (SOUPS)*, 2016.
- [10] Ashwini Rao, Florian Schaub, Norman Sadeh, Alessandro Acquisti, and Ruogu Kang. Expecting the Unexpected: Understanding Mismatched Privacy Expectations Online. In *12th Symposium on Usable Privacy and Security (SOUPS)*, 2016.
- [11] Michael Compton, Payam Barnaghi, Luis Bermudez, Raúl GarcíA-Castro, Oscar Corcho, Simon Cox, John Graybeal, Manfred Hauswirth, Cory Henson, Arthur Herzog, et al. The SSN ontology of the W3C semantic sensor network incubator group. *Web semantics: science, services and agents on the World Wide Web*, 17, 2012.
- [12] Scott R. Peppet. Regulating the Internet of Things: First Steps toward Managing Discrimination, Privacy, Security and Consent. *Texas Law Review*, 93, 2014.
- [13] Hakan Hacigümüs, Balakrishna R. Iyer, Chen Li, and Sharad Mehrotra. Executing SQL over encrypted data in the database-service-provider model. In *Proceedings of the 2002 ACM SIGMOD International Conference on Management of Data, Madison, Wisconsin, USA, June 3-6, 2002*, pages 216–227, 2002.

- [14] Hakan Hacigümüs, Sharad Mehrotra, and Balakrishna R. Iyer. Providing database as a service. In *ICDE*, pages 29–38. IEEE Computer Society, 2002.
- [15] Rakesh Agrawal, Jerry Kiernan, Ramakrishnan Srikant, and Yirong Xu. Order-preserving encryption for numeric data. In *Proceedings of the ACM SIGMOD International Conference on Management of Data, Paris, France, June 13-18, 2004*, pages 563–574. ACM, 2004.
- [16] Oded Goldreich. *The Foundations of Cryptography - Volume 2, Basic Applications*. Cambridge University Press, 2004.
- [17] Craig Gentry. *A fully homomorphic encryption scheme*. PhD thesis, Stanford University, 2009.
- [18] Dawn Xiaodong Song, David A. Wagner, and Adrian Perrig. Practical techniques for searches on encrypted data. In *IEEE Symposium on Security and Privacy*, pages 44–55. IEEE Computer Society, 2000.
- [19] Benny Chor, Eyal Kushilevitz, Oded Goldreich, and Madhu Sudan. Private information retrieval. *J. ACM*, 45(6):965–981, 1998.
- [20] Shuhong Wang, Xuhua Ding, Robert H. Deng, and Feng Bao. Private information retrieval using trusted hardware. *IACR Cryptology ePrint Archive*, 2006:208, 2006.
- [21] Oded Goldreich. Towards a theory of software protection and simulation by oblivious rams. In *Proceedings of the 19th Annual ACM Symposium on Theory of Computing, 1987, New York, New York, USA*, pages 182–194, 1987.
- [22] Michael O. Rabin. How to exchange secrets with oblivious transfer. *IACR Cryptology ePrint Archive*, 2005:187, 2005.
- [23] Moni Naor and Benny Pinkas. Oblivious polynomial evaluation. *SIAM J. Comput.*, 35(5):1254–1281, 2006.
- [24] Arvind Arasu and Raghav Kaushik. Oblivious query processing. In *ICDT*, pages 26–37. OpenProceedings.org, 2014.
- [25] Reza Curtmola, Juan A. Garay, Seny Kamara, and Rafail Ostrovsky. Searchable symmetric encryption: Improved definitions and efficient constructions. *Journal of Computer Security*, 19(5):895–934, 2011.
- [26] Yuval Ishai, Eyal Kushilevitz, Steve Lu, and Rafail Ostrovsky. Private large-scale databases with distributed searchable symmetric encryption. In *Topics in Cryptology - CT-RSA 2016 - The Cryptographers’ Track at the RSA Conference 2016, San Francisco, CA, USA, February 29 - March 4, 2016, Proceedings*, pages 90–107, 2016.
- [27] Adi Shamir. How to share a secret. *Commun. ACM*, 22(11):612–613, 1979.

- [28] Niv Gilboa and Yuval Ishai. Distributed point functions and their applications. In *EUROCRYPT*, volume 8441 of *Lecture Notes in Computer Science*, pages 640–658. Springer, 2014.
- [29] Elette Boyle, Niv Gilboa, and Yuval Ishai. Function secret sharing. In *Advances in Cryptology - EUROCRYPT 2015 - 34th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Sofia, Bulgaria, April 26-30, 2015, Proceedings, Part II*, pages 337–367, 2015.
- [30] Ilan Komargodski and Mark Zhandry. Cutting-edge cryptography through the lens of secret sharing. In *TCC*, pages 449–479, 2016.
- [31] Shlomi Dolev, Niv Gilboa, and Ximing Li. Accumulating automata and cascaded equations automata for communicationless information theoretically secure multi-party computation: Extended abstract. In *Proceedings of the 3rd International Workshop on Security in Cloud Computing, SCC@ASIACCS '15, Singapore, Republic of Singapore, April 14, 2015*, pages 21–29, 2015.
- [32] Shlomi Dolev, Yin Li, and Shantanu Sharma. Private and secure secret shared MapReduce - (extended abstract). In *DBSec*, pages 151–160, 2016.
- [33] Fatih Emekçi, Ahmed Metwally, Divyakant Agrawal, and Amr El Abbadi. Dividing secrets to secure data outsourcing. *Inf. Sci.*, 263:198–210, 2014.
- [34] Wouter Lueks and Ian Goldberg. Sublinear scaling for multi-client private information retrieval. In Rainer Böhme and Tatsuaki Okamoto, editors, *Financial Cryptography and Data Security - 19th International Conference, FC 2015, San Juan, Puerto Rico, January 26-30, 2015, Revised Selected Papers*, volume 8975 of *Lecture Notes in Computer Science*, pages 168–186. Springer, 2015.
- [35] Lichun Li, Michael Miltzer, and Anwitaman Datta. rPIR: Ramp secret sharing based communication efficient private information retrieval. *IACR Cryptology ePrint Archive*, 2014:44, 2014.
- [36] Raluca A. Popa, Catherine M. S. Redfield, Nickolai Zeldovich, and Hari Balakrishnan. Cryptdb: protecting confidentiality with encrypted query processing. In *SOSP*, pages 85–100. ACM, 2011.
- [37] Muhammad Naveed, Seny Kamara, and Charles V. Wright. Inference attacks on property-preserving encrypted databases. In *Proceedings of the 22nd ACM SIGSAC Conference on Computer and Communications Security, Denver, CO, USA, October 12-6, 2015*, pages 644–655, 2015.
- [38] Georgios Kellaris, George Kollios, Kobbi Nissim, and Adam O’Neill. Generic attacks on secure outsourced databases. In *Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security, Vienna, Austria, October 24-28, 2016*, pages 1329–1340, 2016.

- [39] Sharad Mehrotra, Shantanu Sharma, Jeffrey D. Ullman, and Anurag Mishra. Partitioned data security on outsourced sensitive and non-sensitive data. Technical report, Department of Computer Science, University of California, Irvine, 2018. <http://isg.ics.uci.edu/publications.html>.
- [40] Anjana Rajan, Lucy Qin, David W. Archer, Dan Boneh, Tancrede Lepoint, and Mayank Varia. Callisto: A cryptographic approach to detecting serial perpetrators of sexual misconduct. In *Proceedings of the 1st ACM SIGCAS Conference on Computing and Sustainable Societies, COMPASS 2018, Menlo Park and San Jose, CA, USA, June 20-22, 2018*, pages 49:1–49:4, 2018.
- [41] Jonathan Frankle, Sunoo Park, Daniel Shaar, Shafi Goldwasser, and Daniel J. Weitzner. Practical accountability of secret processes. In *27th USENIX Security Symposium, USENIX Security 2018, Baltimore, MD, USA, August 15-17, 2018.*, pages 657–674, 2018.
- [42] Ronald Cramer, Ivan Damgård, and Jesper Buus Nielsen. *Secure Multiparty Computation and Secret Sharing*. Cambridge University Press, 2015.
- [43] Claudio Orlandi. Is multiparty computation any good in practice? In *Proceedings of the IEEE International Conference on Acoustics, Speech, and Signal Processing, ICASSP 2011, May 22-27, 2011, Prague Congress Center, Prague, Czech Republic*, pages 5848–5851, 2011.
- [44] <https://www.csoononline.com/article/3237685/identity-management/biometrics-and-blockchains-the-horcrux-protocol-part-3.html>.
- [45] <https://bitcoinexchangeguide.com/binance-pays-6-cent-fee-for-moving-204-million-worth-of-ethereum-eth/>.
- [46] <https://cryptoslate.com/thailands-democrat-party-holds-first-ever-election-vote-with-blockchain-technology/>.
- [47] <https://blockonomi.com/coinbase-moves-5-billion-crypto/>.
- [48] Fatih Emekçi, Divyakant Agrawal, Amr El Abbadi, and Aziz Gulbeden. Privacy preserving query processing using third parties. In *Proceedings of the 22nd International Conference on Data Engineering, ICDE 2006, 3-8 April 2006, Atlanta, GA, USA*, page 27, 2006.
- [49] Tao Xiang, Xiaoguo Li, Fei Chen, Shangwei Guo, and Yuanyuan Yang. Processing secure, verifiable and efficient SQL over outsourced database. *Inf. Sci.*, 348:163–178, 2016.
- [50] Shlomi Dolev, Yin Li, and Shantanu Sharma. Private and secure secret shared MapReduce (extended abstract). In *Data and Applications Security and Privacy XXX - 30th*

- Annual IFIP WG 11.3 Conference, DBSec 2016, Trento, Italy, July 18-20, 2016. Proceedings*, pages 151–160, 2016.
- [51] Robert M Corless and Nicolas Fillion. A graduate introduction to numerical methods. *AMC*, 10:12, 2013.
 - [52] Martin Burkhart, Mario Strasser, Dilip Many, and Xenofontas A. Dimitropoulos. SEPIA: privacy-preserving aggregation of multi-domain network events and statistics. In *19th USENIX Security Symposium, Washington, DC, USA, August 11-13, 2010, Proceedings*, pages 223–240. USENIX Association, 2010.
 - [53] Ivan Damgård, Matthias Fitzi, Eike Kiltz, Jesper Buus Nielsen, and Tomas Toft. Unconditionally secure constant-rounds multi-party computation for equality, comparison, bits and exponentiation. In *Theory of Cryptography, Third Theory of Cryptography Conference, TCC 2006, New York, NY, USA, March 4-7, 2006, Proceedings*, pages 285–304, 2006.
 - [54] Wei Jiang, Chris Clifton, and Murat Kantarcioglu. Transforming semi-honest protocols to ensure accountability. *Data Knowl. Eng.*, 65(1):57–74, 2008.
 - [55] Keith Bonawitz, Vladimir Ivanov, Ben Kreuter, Antonio Marcedone, H. Brendan McMahan, Sarvar Patel, Daniel Ramage, Aaron Segal, and Karn Seth. Practical secure aggregation for privacy-preserving machine learning. In *Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security, CCS 2017, Dallas, TX, USA, October 30 - November 03, 2017*, pages 1175–1191. ACM, 2017.
 - [56] Dan Bogdanov, Sven Laur, and Jan Willemson. Sharemind: A framework for fast privacy-preserving computations. In Sushil Jajodia and Javier López, editors, *Computer Security - ESORICS 2008, 13th European Symposium on Research in Computer Security, Málaga, Spain, October 6-8, 2008. Proceedings*, volume 5283 of *Lecture Notes in Computer Science*, pages 192–206. Springer, 2008.
 - [57] David W. Archer, Dan Bogdanov, Y. Lindell, Liina Kamm, Kurt Nielsen, Jakob Illeborg Pagter, Nigel P. Smart, and Rebecca N. Wright. From keys to databases - real-world applications of secure multi-party computation. *IACR Cryptology ePrint Archive*, 2018:450, 2018.
 - [58] Ran Canetti, Uriel Feige, Oded Goldreich, and Moni Naor. Adaptively secure multi-party computation. In *Proceedings of the Twenty-Eighth Annual ACM Symposium on the Theory of Computing, Philadelphia, Pennsylvania, USA, May 22-24, 1996*, pages 639–648, 1996.
 - [59] Cong Wang, Ning Cao, Jin Li, Kui Ren, and Wenjing Lou. Secure ranked keyword search over encrypted cloud data. In *2010 International Conference on Distributed Computing Systems, ICDCS 2010, Genova, Italy, June 21-25, 2010*, pages 253–262, 2010.

- [60] Shucheng Yu, Cong Wang, Kui Ren, and Wenjing Lou. Attribute based data sharing with attribute revocation. In Dengguo Feng, David A. Basin, and Peng Liu, editors, *Proceedings of the 5th ACM Symposium on Information, Computer and Communications Security, ASIACCS 2010, Beijing, China, April 13-16, 2010*, pages 261–270. ACM, 2010.
- [61] David M. Goldschlag, Michael G. Reed, and Paul F. Syverson. Onion routing. *Commun. ACM*, 42(2):39–41, 1999.
- [62] Ran Canetti. Security and composition of multiparty cryptographic protocols. *J. Cryptology*, 13(1):143–202, 2000.
- [63] Michael J. Freedman, Yuval Ishai, Benny Pinkas, and Omer Reingold. Keyword search and oblivious pseudorandom functions. In *Theory of Cryptography, Second Theory of Cryptography Conference, TCC 2005, Cambridge, MA, USA, February 10-12, 2005, Proceedings*, pages 303–324, 2005.
- [64] Cheng-Kang Chu and Wen-Guey Tzeng. Efficient k -out-of- n oblivious transfer schemes with adaptive and non-adaptive queries. In *Public Key Cryptography - PKC 2005, 8th International Workshop on Theory and Practice in Public Key Cryptography, Les Diablerets, Switzerland, January 23-26, 2005, Proceedings*, pages 172–183, 2005.
- [65] Mohammad Ali Hadavi, Ernesto Damiani, Rasool Jalili, Stelvio Cimato, and Zeinab Ganjei. AS5: A secure searchable secret sharing scheme for privacy preserving database outsourcing. In *Data Privacy Management and Autonomous Spontaneous Security, 7th International Workshop, DPM 2012, and 5th International Workshop, SETOP 2012, Pisa, Italy, September 13-14, 2012. Revised Selected Papers*, pages 201–216, 2012.
- [66] Cynthia Dwork, Frank McSherry, Kobbi Nissim, and Adam Smith. Calibrating noise to sensitivity in private data analysis. In *TCC*, pages 265–284, 2006.
- [67] Bauchner Howard, M. Golub Robert, and B. Fontanarosa Phil. Data sharing, an ethical and scientific imperative, March 2016.
- [68] Ryan Singel. Netflix cancels recommendation contest after privacy lawsuit. <https://www.wired.com/2010/03/netflix-cancels-contest/>, 2010.
- [69] Michael Barbaro and Tom Zeller Jr. A face is exposed for aol searcher no. 4417749. <http://www.nytimes.com/2006/08/09/technology/09aol.html>, 2006.
- [70] Cynthia Dwork and Aaron Roth. The algorithmic foundations of differential privacy. *Found. Trends Theor. Comput. Sci.*, 9:211–407, August 2014.
- [71] Mohammad Alaggan, Sébastien Gambs, , and Anne-Marie (Kermarrec. Heterogeneous differential privacy. *Journal of Privacy and Confidentiality*, 7(6), 2017.

- [72] Zach Jorgensen, Ting Yu, and Graham Cormode. Conservative or liberal? personalized differential privacy. In *ICDE 2015*, pages 1023–1034, 2015.
- [73] General data protection regulation. *Official Journal of the European Union*, L119:1–88, May 2016.
- [74] Gabe Maldoff. Top 10 operational impacts of the gdpr: Part3 - consent. <https://iapp.org/news/a/top-10-operational-impacts-of-the-gdpr-part-3-consent/>, 2017.
- [75] Oliver Berendt et al. Privacy in e-commerce: Stated preferences vs. actual behavior. *Commun. ACM*, 48(4):101–106, April 2005.
- [76] Kristy Browder and M Davidson. The virtual private database in oracle9ir2. *Oracle Technical White Paper, Oracle Corporation*, 500, 2002.
- [77] Shariq Rizvi et al. Extending query rewriting techniques for fine-grained access control. *SIGMOD '04*, pages 551–562. ACM, 2004.
- [78] Ravi S Sandhu, Edward J Coyne, Hal L Feinstein, and Charles E Youman. Role-based access control models. *Computer*, 29(2):38–47, 1996.
- [79] Marco Guarnieri and David Basin. Optimal security-aware query processing. *Proc. VLDB Endow.*, 7(12):1307–1318, August 2014.
- [80] Pietro Colombo and Elena Ferrari. Access control technologies for big data management systems: literature review and future trends. *Cybersecurity*, 2(1), 2019.
- [81] Michael Stonebraker and Eugene Wong. Access control in a relational data base management system by query modification. In *1974 Annual Conf.*, pages 180–186, 1974.
- [82] Peeyush Gupta, Michael J. Carey, Sharad Mehrotra, and Roberto Yus. Smartbench: A benchmark for data management in smart spaces. In *PVLDB*, volume 13, 2020.
- [83] Surajit Chaudhuri, Prasanna Ganesan, and Sunita Sarawagi. Factorizing complex predicates in queries to exploit indexes. In *ACM SIGMOD Int. Conf. on Management of data*, pages 361–372, 2003.
- [84] Joon Heo, Hyounghoon Lim, Sung Bum Yun, Sungha Ju, Sangyoon Park, and Rebekah Lee. Descriptive and predictive modeling of student achievement, satisfaction, and mental health for data-driven smart connected campus life service. In *9th Int. Conf. on Learning Analytics & Knowledge*, 2019.
- [85] Primal Pappachan and et al. Towards privacy-aware smart buildings: Capturing, communicating, and enforcing privacy policies and preferences. In *Distributed Computing Systems Workshops (ICDCSW), 2017 IEEE 37th International Conference on*, pages 193–198. IEEE, 2017.

- [86] Sharad Mehrotra and et al. Tippers: A privacy cognizant iot environment. In *IEEE International Conference on Pervasive Computing and Communication Workshops (Per-Com Workshops)*, pages 1–6. IEEE, 2016.
- [87] Jialiu Lin, Bin Liu, Norman Sadeh, and Jason I. Hong. Modeling users’ mobile app privacy preferences: Restoring usability in a sea of permission settings. In *10th USENIX Conf. on Usable Privacy and Security*, pages 199–212, 2014.
- [88] Xu Chu, Ihab F. Ilyas, and Paolo Papotti. Discovering denial constraints. *Proc. VLDB Endow.*, 6(13), 2013.
- [89] Pierre Senellart, Louis Jachiet, Silviu Maniu, and Yann Ramusat. Provsq: Provenance and probability management in postgresql. *Proceedings of the VLDB Endowment*, 11(12), 2018.
- [90] Ji-Won Byun and Ninghui Li. Purpose based access control for privacy protection in relational database systems. *The VLDB Journal*, 17(4), 2008.
- [91] Victor Costan and Srinivas Devadas. Intel SGX explained. *IACR Cryptology ePrint Archive*, 2016:86, 2016.
- [92] <https://newsroom.intel.com/newsroom/wp-content/uploads/sites/11/2017/09/8th-gen-intel-core-product-brief.pdf>.
- [93] Wenhao Wang, Guoxing Chen, Xiaorui Pan, Yinqian Zhang, XiaoFeng Wang, Vincent Bindschaedler, Haixu Tang, and Carl A. Gunter. Leaky cauldron on the dark land: Understanding memory side-channel hazards in SGX. In *Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security, CCS 2017, Dallas, TX, USA, October 30 - November 03, 2017*, pages 2421–2434, 2017.
- [94] Somayya Madakam and Hema Date. Security mechanisms for connectivity of smart devices in the internet of things. In *Connectivity Frameworks for Smart Devices: The Internet of Things from a Distributed Computing Perspective*, 2016.
- [95] Jayavardhana Gubbi, Rajkumar Buyya, Slaven Marusic, and Marimuthu Palaniswami. Internet of things (iot): A vision, architectural elements, and future directions. *Future Generation of Computer Systems*, (7):1645–1660, 2013.
- [96] Sye Loong Keoh, Sandeep S. Kumar, and Hannes Tschofenig. Securing the internet of things: A standardization perspective. *IEEE Internet of Things Journal*, 1(3):265–275, 2014.
- [97] IEEE p2413 standard for an architectural framework for the internet of things (iot). 2016. Also available as <http://grouper.ieee.org/groups/2413/Intro-to-IEEE-P2413.pdf>.

- [98] Stephen Kwamena Aikins. Connectivity of smart devices: Addressing the security challenges of the internet of things. In *Connectivity Frameworks for Smart Devices: The Internet of Things from a Distributed Computing Perspective*, 2016.
- [99] Kai Zhao and Lina Ge. A survey on the internet of things security. In *Computational Intelligence and Security (CIS), 2013 9th International Conference on*, pages 663–667. IEEE, 2013.
- [100] Muhammad Umar Farooq, Muhammad Waseem, Anjum Khairi, and Sadia Mazhar. A critical analysis on the security concerns of internet of things (iot). *International Journal of Computer Applications*, 111(7), 2015.
- [101] Cynthia Dwork, Frank McSherry, Kobbi Nissim, and Adam Smith. Calibrating noise to sensitivity in private data analysis. In *3rd Theory of Cryptography Conference*, pages 265–284, 2006.
- [102] Yan Chen, Ashwin Machanavajjhala, Michael Hay, and Gerome Miklau. PeGaSus: Data-adaptive differentially private stream processing. In *ACM Conf. on Computer and Communications Security (CCS)*, pages 1375–1388, 2017.
- [103] Jaewoo Lee and Chris Clifton. How much is enough? choosing ϵ for differential privacy. In *Information Security, 14th International Conference, ISC 2011, Xi'an, China, October 26-29, 2011. Proceedings*, pages 325–340, 2011.
- [104] Deborah Snoonian. Smart buildings. *IEEE spectrum*, 40(8):18–23, 2003.
- [105] Antoni Martínez-Ballesté and et al. The pursuit of citizens’ privacy: a privacy-aware smart city is possible. *IEEE Communications Magazine*, 51(6):136–141, 2013.
- [106] Yuvraj Agarwal and et al. Occupancy-driven energy management for smart building automation. In *Proceedings of the 2nd ACM workshop on embedded sensing systems for energy-efficiency in building*, pages 1–6. ACM, 2010.
- [107] Ruoxi Jia and et al. Privacy-enhanced architecture for occupancy-based HVAC control. In *ACM/IEEE 8th International Conference on Cyber-Physical Systems (ICCPS)*, pages 177–186. IEEE, 2017.
- [108] Himanshu Khurana and Daniel Kirschner. Space utilization and building management system analysis, September 27 2018. US Patent App. 15/467,824.
- [109] Xiao Wang and Patrick Tague. Non-invasive user tracking via passive sensing: Privacy risks of time-series occupancy measurement. In *Proceedings of the Workshop on Artificial Intelligent and Security Workshop*, pages 113–124. ACM, 2014.
- [110] Lin Liu and et al. Security and privacy requirements analysis within a social setting. In *Proceedings. 11th IEEE International Requirements Engineering Conference, 2003.*, pages 151–161. IEEE, 2003.

- [111] Varick L Erickson and et al. Observe: Occupancy-based system for efficient reduction of hvac energy. In *Proceedings of the 10th ACM/IEEE International Conference on Information Processing in Sensor Networks*, pages 258–269. IEEE, 2011.
- [112] Latanya Sweeney. k-anonymity: A model for protecting privacy. *International Journal of Uncertainty, Fuzziness and Knowledge-Based Systems*, 10(05):557–570, 2002.
- [113] Cynthia Dwork. Differential privacy. *Encyclopedia of Cryptography and Security*, pages 338–340, 2011.
- [114] Yan Chen and et al. PeGaSus: Data-adaptive differentially private stream processing. In *Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security*, pages 1375–1388. ACM, 2017.

Appendix A - Published Papers

2021

- Yiming Lin, Daokun Jiang, Roberto Yus, Georgios Bouloukakis, Andrew Chio, Sharad Mehrotra, and Nalini Venkatasubramanian. LOCATER: Cleaning WiFi Connectivity Datasets for Semantic Localization. Proceedings of the VLDB Endowment, 2021.
- Abdulrahman Alsaoudi, Yasser Altowim, Sharad Mehrotra, Yaming Yu. TQEL: Framework for Query-Driven Linking of Top-K Entities in Social Media Blogs. Accepted in Proceedings of the VLDB Endowment, 2021
- Shlomi Dolev, Peeyush Gupta, Yin Li, Sharad Mehrotra, and Shantanu Sharma. Privacy-Preserving Secret Shared Computations using MapReduce. IEEE Trans. Dependable Secur. Comput. 18(4): 1645-1666, 2021.
- Nisha Panwar, Shantanu Sharma, Guoxi Wang, Sharad Mehrotra, Nalini Venkatasubramanian, Mamadou H. Diallo, and Ardan Amiri Sani. IoT Notary: Attestable Sensor Data Capture in IoT Environments. ACM Transactions on Internet of Things, 2021.
- Yin Li, Dhrubajyoti Ghosh, Peeyush Gupta, Sharad Mehrotra, Nisha Panwar, and Shantanu Sharma. Prism: Private Verifiable Set Computation over Multi-Owner Outsourced Databases. Proceedings of the International Conference on Management of Data (ACM SIGMOD).
- Peeyush Gupta, Sharad Mehrotra, Shantanu Sharma, Nalini Venkatasubramanian, and Guoxi Wang. Concealer: SGX-based Secure, Volume Hiding, and Verifiable Processing of Spatial Time-Series Datasets. Proceedings of the International Conference on Extending Database Technology (EDBT).
- Yiming Lin, Pramod Khargonekar, Sharad Mehrotra, and Nalini Venkatasubramanian. T-Cove: An exposure tracing System based on Cleaning Wi-Fi Events on Organizational Premises. Proceedings of the VLDB Endowment, 2021.
- Christopher Davison, Edward Lazaros, Jensen Zhao, Allen Truell, Brianna Bowles. Data privacy in the age of big data analytics. Proceedings of the 61st International Association for Computer Information Systems Annual Conference, 2021. Best Pedagogy Paper Award.
- Hakan Hacigi  m   , Bijit Hore, Sharad Mehrotra, and Shantanu Sharma. [Book Chapter] Privacy of Outsourced Data. Encyclopedia of Cryptography, Security and Privacy.
- Sharad Mehrotra and Shantanu Sharma. [Book Chapter] Search over Ciphertext Datasets using Partition Computation. Encyclopedia of Cryptography, Security and Privacy.

- Ali Bagherzandi, Bijit Hore, Sharad Mehrotra, Shantanu Sharma. [Book Chapter] Search over Encrypted Data. Encyclopedia of Cryptography, Security and Privacy.
- Sharad Mehrotra, and Shantanu Sharma. [Book Chapter] Search over Secret Shared Datasets. Encyclopedia of Cryptography, Security and Privacy.

2020

- Joshua Cao, Jesse Chong, Marissa Lafreniere, Owen Yang, Primal Pappachan, Sharad Mehrotra, and Nalini Venkatasubramanian. The ZotBins solution to Waste Management using Internet of Things. 18th ACM Conference on Embedded Networked Sensor Systems (SenSys '20).
- Michael August, Christopher Davison, Mamadou Diallo, Dhrubajyoti Ghosh, Peeyush Gupta, Christopher Graves, Shanshan Han, Michael Holstrom, Pramod Khargonekar, Megan Kline, Sharad Mehrotra, Shantanu Sharma, Nalini Venkatasubramanian and Guoxi Wang, and Roberto Yus A Privacy-Enabled Platform for COVID-19 Applications. 18th ACM Conference on Embedded Networked Sensor Systems (SenSys '20)
- Sameera Ghayyur, Primal Pappachan, Guoxi Wang, Sharad Mehrotra, and Nalini Venkatasubramanian. Designing Privacy Preserving Data Sharing Middleware for Internet of Things. The 3rd International SenSys+BuildSys Workshop on Data: Acquisition to Analysis (DATA '20).
- Nisha Panwar, Shantanu Sharma, Peeyush Gupta, Dhrubajyoti Ghosh, Sharad Mehrotra, and Nalini Venkatasubramanian. IoT Expunge: Implementing Verifiable Retention of IoT Data. CODASPY'20: Tenth ACM Conference on Data and Application Security and Privacy, New Orleans, LA, USA, March 16-18, 2020.
- Peeyush Gupta, Michael Carey, Sharad Mehrotra, and Roberto Yus. SmartBench: A Benchmark For Data Management In Smart Spaces. Proceedings of the VLDB Endowment, 2020
- Primal Pappachan, Roberto Yus, Sharad Mehrotra, and Johann-Christoph Freytag. SIEVE: A Middleware Approach to Scalable Access Control for Database Management Systems. Proceedings of the VLDB Endowment, 2020.
- Sharad Mehrotra, Shantanu Sharma, Jeffrey D. Ullman, Dhrubajyoti Ghosh, and Peeyush Gupta. PANDA: Partitioned Data Security on Outsourced Sensitive and Non-sensitive Data. ACM Transactions on Management Information Systems (ACM TMIS), Volume 11, Issue 4, 2020.
- Peeyush Gupta, Yin Li, Sharad Mehrotra, Nisha Panwar, and Shantanu Sharma. OBSCURE: Information-Theoretically Secure, Oblivious, and Verifiable Aggregation Queries. CODASPY'20: Tenth ACM Conference on Data and Application Security and Privacy, New Orleans, LA, USA, March 16-18, 2020.

- Peeyush Gupta, Yin Li, Sharad Mehrotra, Nisha Panwar, Shantanu Sharma, and Sumaya Almanee. OBSCURE: Information-Theoretically Secure, Oblivious, and Verifiable Aggregation Queries on Secret-Shared Outsourced Data. *IEEE Transactions on Knowledge and Data Engineering (IEEE TKDE)*, 2020.
- Nisha Panwar, Shantanu Sharma, Guoxi Wang, Sharad Mehrotra, and Nalini Venkatasubramanian. CANOPY: A Verifiable Privacy-Preserving Token Ring based Communication Protocol for Smart Homes. *ACM Transactions on Cyber-Physical Systems (ACM TCPS)*, 2020.
- Shantanu Sharma, Anton Burtsev, and Sharad Mehrotra. Advances in Cryptography and Secure Hardware for Data Outsourcing. 36th IEEE International Conference on Data Engineering (IEEE ICDE), 2020.
- Sharad Mehrotra, and Shantanu Sharma. Recent Advances in Information-Theoretically Secure Data Outsourcing. 6th ACM International Workshop on Security and Privacy Analytics (IWSPA) associated with 10th ACM Conference on Data and Application Security and Privacy (CODASPY), 2020.
- Ios Kotsogiannis, Stelios Doudalis, Samuel Haney, Ashwin Machanavajjhala, and Sharad Mehrotra. One-sided Differential Privacy. 36th IEEE International Conference on Data Engineering (IEEE ICDE), 2020.
- Dave Archer, Michael A August, Georgios Bouloukakis, Christopher Davison, Mamadou H Diallo, Dhrubajyoti Ghosh, Christopher T Graves, Michael Hay, Xi He, Peeter Laud, Steve Lu, Ashwin Machanavajjhala, Sharad Mehrotra, Jerome Miklau, Alisa Pankova, Shantanu Sharma, Nalini Venkatasubramanian, Guoxi Wang, and Roberto Yus. Transitioning from testbeds to ships: an experience study in deploying the TIPPERS Internet of Things platform to the US Navy. *The Journal of Defense Modeling and Simulation: Applications, Methodology, Technology*.

2019

- Peeyush Gupta, Yin Li, Sharad Mehrotra, Nisha Panwar, Shantanu Sharma, and Sumaya Almanee. OBSCURE: Information-Theoretic Oblivious and Verifiable Aggregation Queries. *Proceedings of the VLDB Endowment*, 2019.
- Sharad Mehrotra, Shantanu Sharma, Jeffrey D. Ullman, and Anurag Mishra. Partitioned Data Security on Outsourced Sensitive and Non-Sensitive Data. 35th IEEE International Conference on Data Engineering (ICDE 2019).
- Nisha Panwar, Shantanu Sharma, Guoxi Wang, Sharad Mehrotra, Nalini Venkatasubramanian, Mamadou H. Diallo, and Ardalan Amiri Sani. IoT Notary: Sensor Data attestation in Smart Environment. *International Symposium on Network Computing and Applications (IEEE NCA)*, 2019, Best Paper Award.

- Sameera Ghayyur, Dhrubajyoti Ghosh, Xi He, and Sharad Mehrotra. Towards Accuracy Aware Minimally Invasive Monitoring, Theory, and practice of Differential Privacy. ACM Cloud Computing Security Workshop.
- Nisha Panwar, Shantanu Sharma, Guoxi Wang, Sharad Mehrotra, and Nalini Venkatasubramanian. Verifiable Round-Robin Scheme for Smart Homes. Proceedings of the Ninth ACM Conference on Data and Application Security and Privacy, CODASPY 2019, Richardson, TX, USA, March 25-27, 2019.
- Sharad Mehrotra, Shantanu Sharma, and Jeffrey D. Ullman Scaling Cryptographic Techniques by Exploiting Data Sensitivity at a Public Cloud. Proceedings of the Ninth ACM Conference on Data and Application Security and Privacy, CODASPY 2019, Richardson, TX, USA, March 25-27, 2019. Distinguished Poster Award.
- Eun-Jeong Shin, Dhrubajyoti Ghosh, Sharad Mehrotra, and Nalini Venkatasubramanian. SCARF: a scalable data management framework for context-aware applications in smart environments. MobiQuitous 2019, Proceedings of the 16th EAI International Conference on Mobile and Ubiquitous Systems: Computing, Networking and Services, Houston, Texas, USA, November 12-14, 2019.
- Sameera Ghayyur, Dhrubajyoti Ghosh, Xi He, and Sharad Mehrotra. Towards Accuracy Aware Minimally Invasive Monitoring (MiM). ACM CCS (TPDP workshop).
- Roberto Yus, Georgios Bouloukakis, Sharad Mehrotra, and Nalini Venkatasubramanian. Abstracting Interactions with IoT Devices Towards a Semantic Vision of Smart Spaces. 6th ACM International Conference on Systems for Energy-Efficient Buildings, Cities, and Transportation, BuildSys 2019, New York, NY, USA, November 13-14, 2019.
- Sumaya Almanee, Georgios Bouloukakis, Daokun Jiang, Sameera Ghayyur, Dhrubajyoti Ghosh, Peeyush Gupta, Yiming Lin, Sharad Mehrotra, Primal Pappachan, Eun-Jeong Shin, Nalini Venkatasubramanian, Guoxi Wang, and Roberto Yus. SemIoTic: Bridging the Semantic Gap in IoT Spaces. 6th ACM International Conference on Systems for Energy-Efficient Buildings, Cities, and Transportation, BuildSys 2019, New York, NY, USA, November 13-14, 2019.
- Anton Burtsev, Sharad Mehrotra, and Shantanu Sharma. Secure and Privacy-Preserving Big-Data Processing. IEEE Big-Data, Los Angeles, CA, USA, Dec 09-12, 2019.

2018

- Mamadou H. Diallo, Nisha Panwar, Sharad Mehrotra, and Ardalan Amiri Sani. Trustworthy Sensing in an Untrusted IoT Environment. 2018 IEEE International Conference on Pervasive Computing and Communications Workshops, PerCom Workshops 2018, Athens, Greece, March 19-23, 2018.

- Mamadou H. Diallo, Nisha Panwar, Roberto Yus, and Sharad Mehrotra. Trustworthy Privacy Policy Translation in Untrusted IoT Environments. 3rd International Conference on Internet of Things, Big Data and Security, IoTBDS 2018, Funchal, Madeira, Portugal, March 19-21, 2018.
- Sameera Ghayyur, Yan Chen, Roberto Yus, Ashwin Machanavajjhala, Michael Hay, Gerome Miklau, and Sharad Mehrotra. IoT-Detective: Analyzing IoT Data Under Differential Privacy. 2018 International Conference on Management of Data, SIGMOD Conference 2018, Houston, TX, USA, June 10-15, 2018.
- Sharad Mehrotra, Kerim Yasin Oktay, and Shantanu Sharma. [Book Chapter] Exploiting Data Sensitivity on Partitioned Data. From Database to Cyber Security - Essays Dedicated to Sushil Jajodia on the Occasion of His 70th Birthday, 2018.

2017

- Kerim Yasin Oktay, Murat Kantarcioglu, and Sharad Mehrotra. Secure and Efficient Query Processing over Hybrid Clouds. 33th IEEE International Conference on Data Engineering (IEEE ICDE), 2017.
- Abdulrahman Alsaudi, Mehdi Sadri, Yasser Altowim, Sharad Mehrotra. Adaptive Topic Follow-up on Twitter. 33rd IEEE International Conference on Data Engineering (IEEE ICDE), 2017
- Primal Pappachan, Martin Degeling, Roberto Yus, Anupam Das, Sruti Bhagavatula, William Melicher, Pardis Emami Naeini, Shikun Zhang, Lujo Bauer, Alfred Kobsa, Sharad Mehrotra, Norman M. Sadeh, and Nalini Venkatasubramanian. Towards Privacy-Aware Smart Buildings: Capturing, Communicating, and Enforcing Privacy Policies and Preferences. 37th IEEE International Conference on Distributed Computing Systems Workshops, ICDCS Workshops 2017, Atlanta, GA, USA, June 5-8, 2017.
- Saeed Mirzamohammadi, Justin A. Chen, Ardalan Amiri Sani, Sharad Mehrotra, and Gene Tsudik. Ditio: Trustworthy Auditing of Sensor Activities in Mobile & IoT Devices. Proceedings of the 15th ACM Conference on Embedded Network Sensor Systems, SenSys 2017, Delft, Netherlands, November 06-08, 2017.
- Eun-Jeong Shin, Roberto Yus, Sharad Mehrotra, and Nalini Venkatasubramanian. Exploring fairness in participatory thermal comfort control in smart buildings. 4th ACM International Conference on Systems for Energy-Efficient Built Environments, BuildSys 2017, Delft, The Netherlands, November 08-09, 2017.

2016

- Sharad Mehrotra, Alfred Kobsa, Nalini Venkatasubramanian, and Siva Raj Rajagopalan. TIPPERS: A privacy cognizant IoT environment. 2016 IEEE International Conference on Pervasive Computing and Communication Workshops, PerCom Workshops 2016, Sydney, Australia, March 14-18, 2016.

2015

- Primal Pappachan, Roberto Yus, Prajit Kumar Das, Sharad Mehrotra, Tim Finin, and Anupam Joshi. Building a Mobile Applications Knowledge Base for the Linked Data Cloud. Proceedings of the 1st International Workshop on Mobile Deployment of Semantic Technologies (MoDeST 2015) co-located with 14th International Semantic Web Conference (ISWC 2015), Bethlehem, Pennsylvania, USA, October 11th, 2015.
- Primal Pappachan, Roberto Yus, Prajit Kumar Das, Sharad Mehrotra, Tim Finin, and Anupam Joshi. Mobipedia: Mobile Applications Linked Data. Proceedings of the ISWC 2015 Posters & Demonstrations Track co-located with the 14th International Semantic Web Conference (ISWC-2015), Bethlehem, PA, USA, October 11, 2015.
- Kerim Yasin Oktay, Sharad Mehrotra, Vaibhav Khadilkar, and Murat Kantarcioglu. SEMROD: Secure and Efficient MapReduce Over Hybrid Clouds. SIGMOD Conference 2015.

Appendix B - Unpublished Papers

- Roberto Yus, Georgios Bouloukakis, Sharad Mehrotra, and Nalini Venkatasubramanian. The SemIoTic Ecosystem: A Semantic Bridge between IoT Devices and Smart Spaces.
- Roberto Yus, Nada Lahjouji, Georgios Bouloukakis, Daokun Jiang, and Sharad Mehrotra. Modeling Inhabited Smart Spaces to Support Interoperable Sensor-Based Applications.
- Andrew Chio, Peeyush Gupta, Daokun Jiang, Roberto Yus, Georgios Bouloukakis, Nalini Venkatasubramanian, and Sharad Mehrotra. SmartSPEC: Customizable Smart Space Datasets via Event-driven Simulations.
- Sameera Ghayyur, Peeter Laud, Alisa Pankova, Sharad Mehrotra, and Roberto Yus. Empirical Evaluation of Diverse PETs to Publish Smart Space Occupancy Data.
- Peeyush Gupta, Shanshan Han, Pramod Khargonekar, Sharad Mehrotra, Nisha Panwar, Nalini Venkatasubramanian, and Guoxi Wang. SODA: Secure and Privacy-Preserving Outsourcing of Organization Sensor Data.
- Dhrubajyoti Ghosh, Peeyush Gupta, Sharad Mehrotra, and Shantanu Sharma. A Case for Enrichment in Data Management Systems.
- Dhrubajyoti Ghosh, Peeyush Gupta, Sharad Mehrotra, and Shantanu Sharma. Implementing Scalable Query Time Data Enrichment.

Appendix C - In-progress Papers

- Primal Pappachan, Shufan Zhang, Xi He, and Sharad Mehrotra. Don't be a tattletale: Preventing leakages through data dependencies on access control protected data.
- Dhrubajyoti Ghosh, Roberto Yus, Yasser Altowim, and Sharad Mehrotra. Optimizing Enrichment with Progressive Query Processing.
- Sameera Ghayyur, Dhrubajyoti Ghosh, Xi He, Sharad Mehrotra. Accuracy Aware Minimally Invasive Event Detection Using Differential Privacy

List of Symbols, Abbreviations, and Acronyms

- ACM → Association for Computing Machinery
- API → Application Program Interface
- AWS → Amazon Web Services
- BMS → Building Management System
- BSU → Ball State University
- CMU → Carnegie Mellon University
- CoAP → Constrained Application Protocol
- DARPA → Defense Advanced Research Projects Agency
- DB → Database
- DBH → Donald Bren Hall building at UCI
- DBMS → Database Management System
- DP → Differential Privacy
- DS → Decision support
- DSSE → Distributed Searchable Symmetric Encryption
- EU → European Union
- RFID → Radio Frequency Identification
- FGAC → Fine-Grained Access Control
- GCC → Group Comfort Control
- GDPR → General Data Protection Regulation
- GPS → Global Positioning System
- HTTP → Hypertext Transfer Protocol
- HVAC → Heating, ventilation, and air conditioning
- IEEE → Institute of Electrical & Electronics Engineers
- IFD → Infrastructure Deployer
- IoT → Internet of Things
- IoTA → Internet of Things Assistant
- IRB → Institutional Review Board

- IRR → IoT Resource Registry
- IT → Information Technology
- JSON → JavaScript Object Notation
- LoC → Lines of Code
- MAC → Media Access Control
- MiM → Minimally Invasive Monitoring
- MQTT → MQ Telemetry Transport
- NIWC → Naval Information Warfare Center
- OIT → Office of Information Technology
- ORAM → Oblivious Random Access Memory
- OSDP → One-Sided Differential Privacy
- OT → Oblivious Transfers
- PC → Personal Computer
- PE-IoT → Privacy Enhancing Internet of Things
- PET → Privacy Enhancing Technologies
- PIR → Private Information Retrieval
- PPT → Probabilistic Polynomial Time
- QB → Query Binning
- QoS → Quality of Service
- RAM → Random Access Memory
- RDBMS → Relational Database Manage System
- RESTful → Representational state transfer
- SGX → Secure Guard eXtension
- SOSA → Sensor, Observation, Sample, and Actuator Ontology
- SP → Service Providers
- SPAWAR → Space and Naval Warfare Systems Command
- SSN → Semantic Sensor Networks
- TIPPERS → Testbed for IoT-based Privacy preserving PERvasive Spaces
- TQL → TIPPERS Query Language

- TTL → Time to Live
- TW → Trident Warrior
- UCI → University of California, Irvine
- UDF → User Defined Function
- URI → Uniform Resource Identifiers
- US → United States
- VIP → Very Important Person
- VM → Virtual Machine
- WiFi AP → WiFi Access Point
- XMPP → Extensible Messaging and Presence Protocol