Oblivious Sensor Fusion via Secure Multi-Party Combinatorial Filter Evaluation

William E. Curran, Cesar A. Rojas, Leonardo Bobadilla, and Dylan A. Shell

***Abstract*— Given sensor units distributed throughout an environment, we consider the problem of consolidating readings into a single coherent view when sensors wish to limit knowledge of their specific readings. Standard fusion methods make no guarantees about what curious participants may learn. For applications where privacy guarantees are required, we introduce a fusion approach that limits what can be inferred. First, it forms an aggregate stream, oblivious to the underlying sensor data, and then evaluates that stream on a combinatorial filter. This is achieved via secure multi-party computation techniques built on cryptographic primitives, which we extend and apply to the problem of fusing discrete sensor signals. We prove that the extensions preserve security under the model of semi-honest adversaries. Also, for a simple target tracking case study, we examine a proof-of-concept implementation: analyzing the (empirical) running times for components in the architecture and suggesting directions for future improvement.**

I. Introduction

Many practical applications (e.g., situational awareness, and activity modeling problems) involve answering queries from a stream of sensor readings. The present paper examines how multiple physically distributed sensors can use computational and communication mechanisms to consolidate their sensed data to form a fused estimate. Unlike prior work, we are interested in circumstances where the elements comprising the system need not fall under the aegis of a single authority. Though participating as honest parties, they may be inquisitive (e.g., providers competing in a market where access to data streams is purchased). Or they may have been compromised so as to stash and/or divulge data (e.g., when adversaries are capable of instrumenting but not modifying devices). The objective, then, is to fuse data from multiple devices, while also limiting what those devices may learn over-and-above that which is known from their individual views.

Our treatment focuses on a class of finite-state transducers called *combinatorial filters* [23] that act as discrete state estimators to convert a stream of inputs into a stream of outputs: the former being sensor readings, the latter estimates of properties of interest encoded abstractly. Figure 1[left] provides a (simple) concrete example: two agents move freely about an environment, each tracing a continuous path and occasionally crossing one of the three-beam sensors (labeled 'a', 'b', and 'c'). As the agents move, they interact with the sensors which generate a sequence of symbols that is processed by tracing over the correspondingly labeled edges. The 4-state filter in Figure 1[right] concisely answers the query *are the two agents together or not?*

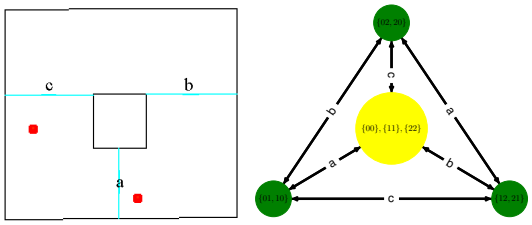

Fig. 1. A simple scenario (from [23]): we track the state of two agents moving in a room that contains a single central obstacle. A small combinatorial filter fuses information obtained from three break-beam sensors ('a', 'b', and 'c') to determine whether the agents are together or not. [left] A screenshot of our python-based simulator showing the environment—the red dots are the agents, the cyan line segments are the beams. [right] A combinatorial filter with 4 states that, from any given initial configuration, track whether the agents are together (yellow) or apart (green).

In an implementation, the beam sensors might be realized as three different devices: each an embedded computer (with limited computational capabilities), connected via a lossless communication network. In addition to the sensors, some separate device (or party) holds the graph structure that describes the filter. Finally, there is the party with a legitimate interest in the current output of the filter, which we dub the *querier*. In the example, this is the color: yellow or green, encoding together or apart, respectively. The method we present ensures that the querier learns only the colors, neither the input symbols nor the filter structure. The device possessing the filter learns neither the input symbols nor the output stream. And, individually, the devices governing the sensors learn nothing other than their own sensed readings.

Using secure Multi-Party Computation (MPC) techniques, we process a stream of events through a staged approach: first, pooling data from the various sensors and then evaluating the given combinatorial filter, incrementally. The security of this scheme is established via proofs in the semi-honest model, a common treatment for cooperative though curious devices, constrained to obey the prescribed protocol. This model allows claims to be made about information that the protocol leaks, as distinct from implicitly disclosed via the input-output relationships [11].

Doing so requires that some technical challenges be addressed. First, while MPC commonly operates on data in a batch fashion, a fundamental part of real-time filtering is the treatment of intrinsically asynchronous exogenous events. But, even if the provenance of particular sensor readings can be obscured, triggering evaluation when an event occurs will leak information (as there is a correlation with activity). Second, existing techniques for oblivious automata evaluation

William E. Curran and Dylan A. Shell are with the Department of Computer Science & Engineering at Texas A&M University, College Station, TX, USA. wec72@tamu.edu, dshell@tamu.edu; Cesar A. Rojas and Leonardo Bobadilla are with the Knight Foundation School of Computing and Information Science at Florida International University, Miami, FL, USA. croja022@fiu.edu, bobadilla@cs.fiu.edu

This work is supported in part by the National Science Foundation awards IIS-2034123, IIS-2024733, IIS-2034097, and by the U.S. Dept. of Homeland Security award 2017-ST-062000002.

cannot be applied directly. Not only is some generalization needed for additional outputs (multiple color outputs must be produced at each point during the evaluation), but these mean that a *mutatis mutandis* modification of the original security proof is inadequate. Third, despite steady progress, current MPC methods remain computationally intensive: an approach should only be deemed practicable if its performance is assessed directly.

The paper's specific contributions can be summarized as: (1) Design of an architecture and multi-stage protocol in which sensor units buffer readings locally, then pool data via peers that also jointly evaluate the filter. The peers only have access to split versions of the aggregated pool and process it at a fixed frequency, so the stream discloses nothing about the amount of activity in the environment. Using this encoding, when events are known a priori to occur infrequently (for instance, when the velocity of the agents in Figure 1[left] is bounded) this can help reduce communication. On the basis of our implementation, we explore treatments for the dense and (known) bounded-sparse regime, showing that they have different computational requirements and behavior.
(2) As part of the approach, we give a method for obvious evaluation of a Moore machine that generalizes techniques within the literature. Unlike automata evaluation that yields a binary output at the end of the evaluation only, the Moore machine produces a stream of (potentially many) colors. This changes the steps taken during evaluation and also alters the proof that the process is secure. More information is disclosed during Moore machine evaluation, so the consumer of this data would appear to have much more information about the input stream — the proof, thus, takes a different form because the straightforward adaptation would no longer give a polynomial-time simulation.
(3) The final section also describes our proof-of-concept implementation and examines its performance, providing an empirical view of the compromise between communication and computation costs. The pooling approach involves a parameter, L, of pool window size, and we conduct a empirical assessment of its effect.

II. Related Work

This paper addresses a discrete estimation problem in an unconventional computational setting, drawing on multiple MPC techniques and related concepts. Technical connections are made in the sections that follow, once the basic definitions and problem formalization have been introduced.

A. Filters, Estimators, and Trackers

Combinatorial filters, a term introduced in [23] and related to discrete event systems [6], are estimators which process a stream of symbols to produce output encoding structural or stateful properties of interest. These filters are ideal for processing simple sensor inputs [5] and have been the subject of study under the broader banner of minimalism [21]. We know of no work that does multi-sensor/decentralized fusion for such filters, unlike traditional probabilistic filters [7].

For probabilistic filters, [16] consider multiple participants contributing input data. They are concerned with privacy, employing the differential privacy model [8], which limits information about individual contributions. The MPC approach, being built with cryptographic primitives, offers stronger guarantees.

For the discrete setting, several pieces of work consider notions of opacity [14], obfuscation [24], and discreetness [20]. Some work has also examined privacy-preserving tracking problems [26].

B. Secure Multi-Party Computation

Secure MPC deals with situations where multiple parties, each possessing some input data, wish to compute some function of the union of the inputs, but while having their input remain private [11]. Each party only learns the output of the computation, even when inimical participants collude into trying to learn more about the inputs of the others.

An early problem examined by Andrew Yao [25] is the 'Millionaire's problem', where two individuals determine who has greater wealth (evaluating a '$\leq$'), without revealing their own bank balances. Yao's solution involved a process termed 'oblivious transfer' (OT), which involves a sending party transmitting a set of items as a message while remaining unaware of which were received by the receiver (typically only a strict subset). OTs form an important building block for Secure MPC protocols and, in assessing the performance of our implementation, we will report the number involved.

Recently, researchers have begun to apply MPC to problems in robotics, control, and estimation [17], [1], [27].

C. The GMW protocol

In the semi-honest setting, several general protocols to compute deterministic functions exist [11]. We make use of the Goldreich–Micali–Wigderson (GMW) Protocol [12] in which two parties with, respectively, private inputs x and y wish to evaluate a public function $f(x, y)$ via an agreed-upon circuit C [11]. The core idea behind this protocol is that each party supplies additive shares for circuit inputs, and they evaluate C on shared values [11], [15]. In principle, at least for a given length input, the entire Moore Machine Evaluation could be implemented via generic GMW. However, doing so will reduce the compactness of representation we employ and, as mentioned by [10, cited by [19]], the present approach is expected to be significantly more efficient. Also, processing a stream of inputs (of which the final total length may not be known ahead of time) makes it non-trivial to construct an appropriate circuit for the entire computation.

III. Preliminaries

A. Basic Notation

By $x \xleftarrow{R} \{0,1\}^n$, denote the generation of a random binary string of length n using a cryptographically secure random number generator. Let $G : \{0,1\}^n \to \{0,1\}^m$ be a pseudorandom number generator with n-bit seed and m-bit output. And have $u||v$ represent the concatenation of bit strings u and v. We write $u \oplus v$ for the bitwise XOR operation on binary strings u and v. Finally, in cases where we wish to explicitly represent an empty input or empty output, we follow the convention of using the symbol '$\perp$'.

The term 'party' is conventional in the cryptographic literature to describe a computational actor that possesses its own view of the world, information potentially unknown to

others, and possibly its own objectives [11]. Our setting will consider multiple devices, each with access to only partial information, which we treat as private: there are n sensors that fuse their streams and two additional devices which perform subsequent processing, but in a split fashion. Interaction between these devices is primarily pairwise, so that two-party functionalities are our natural focus. Abstractly, a *two-party functionality* is a function $f : \{0,1\}^* \times \{0,1\}^* \to \{0,1\}^* \times \{0,1\}^*$, in which $f = (f_1, f_2)$. The inputs to the functionality f are pairs $x, y \in \{0,1\}^n$ and the output is $f = (f_1(x,y), f_2(x,y))$. The first party, with input x, wants to obtain $f_1(x,y)$ and the second party, with y, desires $f_2(x,y)$. In the case of the functionality just described, thus, x and y respectively represent private inputs. Protocols, denoted π, are used to calculate functionalities.

B. Security Model

Throughout, we restrict our attention to the semi-honest model of adversaries [11]. In such settings, each party is obligated to follow the prescribed protocol but may attempt to learn as much as possible during the computation. To ensure a protocol implements some functionality under the semi-honest model securely, proof via the simulation paradigm is used [18]. Section V-B includes the construction of a simulator to establish the security of a protocol we propose.

C. Combinatorial Filters

Following [21], we consider a structure that processes sequences of events (typically interpreted as observations or actions or both). We are interested in representing sets of such sequences and do this via a graph where each reachable vertex corresponds to an equivalence class of sequences. The vertices are termed *information states* or I-states for short.

Definition 1: An I-state graph $\mathbf{G} = (V, E, l : E \to Y, v_0)$ on event set Y is an edge labeled, directed graph with an initial vertex. Here V is a finite set of vertices (the I-states), E is a set of ordered pairs of vertices (directed edges), the function l labels each edge with an event from Y, and v_0 represents the starting I-state.

Throughout, we consider only deterministic I-state graphs, namely those where no vertices have multiple outgoing edges labeled by the same event. Properties of interest that can be computed on I-states will be encoded as *colors*.

Definition 2: A combinatorial filter $\mathbf{F}$, is a deterministic I-state graph supplemented with an assignment of colors to its vertices, $\mathbf{F} = (\mathbf{G}, c : V \to \mathbb{N})$, where $\mathbf{G}$ is an I-state graph and c assigns a natural number to each vertex.

The output of filter $\mathbf{F} = (\mathbf{G}, c)$ on input $y_1 y_2 \dots y_m$ is the sequence of colors $c(v_0)c(v_1)\dots c(v_m)$, where v_0 is the starting I-state of $\mathbf{G}$, and each $l\big((v_{i-1}, v_i)\big) = y_i$ for $i \in \{1 \dots m\}$. We will also write the filter as if it were a function, $\mathbf{F}[\![y_1 y_2 \dots y_m]\!]$, and the output is a string of colors.

IV. Problem Formulation

Sensors are each connected locally to a processor, forming a single logical unit that we dub an *event detector*. We have n such event detectors, $\mathcal{D}_1, \mathcal{D}_2, \dots, \mathcal{D}_n$, each of which senses or otherwise observes events in the world. After detecting the occurrence of an event, they yield an *event symbol*. Multiple event detectors may produce the same symbol, so we consider $m \leq n$ event symbols that (together with ϵ representing no event being observed) form our *overall event space*, denoted $Y = \{y_1, y_2, y_3, \dots, y_m, \epsilon\}$. For the example in Figure 1, we have 3 event detectors $\mathcal{D}_1$, $\mathcal{D}_2$, and $\mathcal{D}_3$ each associated to a beam sensor. Here the overall event space is $Y = \{a, b, c, \epsilon\}$. Each event detector $\mathcal{D}_j$ will have a *local event space* $Y^j = \{y_i, \epsilon\}$, where $y_i \in Y$ represents an event gathered by $\mathcal{D}_j$. We shall assume each event is covered by at least one event detector, i.e., $Y = \bigcup_{j=1}^{n} Y^j$.

For event detector $\mathcal{D}_j$, the *local event history*, denoted $\tilde{h}_j = ((t_0, x_0), (t_1, x_1), \dots, (t_{L-1}, x_{L-1}))$, is a sequence representing the information collected at L distinct times, $t_i < t_{i+1}$, and where each $x_i \in Y^j$ for $i \in \{0 \dots L-1\}$. In what follows, we will assume that times are numbers requiring τ bits of storage. Also, length L will be a parameter used in the methods we present. Note, that since $\epsilon \in Y^j$, one may always pad shorter histories to attain length L. The *global event history*, which we denote $\tilde{y}$, is the union of all events, as a sequence ordered by time.

We employ two phases: the first involves pooling data from detectors to construct a subsequence of the global event history; the second that processes the subsequence, evaluating the filter. Both phases must keep data private.

A. Securely Pooling Event Sequences

We consolidate local event histories into a single chronologically-ordered stream by batching subsequences and operating on windows of length L. Evaluation of the combinatorial filter over time proceeds batch-by-batch.

To maintain the privacy of each detector's data, we employ a secret sharing approach: every $\mathcal{D}_j$ splits its event stream into two shares, and sends these to two new parties. These two additional parties, designated $\mathcal{O}$ and $\mathcal{Q}$, bear the burden of subsequent computation and are assumed to be more capable than our embedded event detectors. Parties such as $\mathcal{O}$ and $\mathcal{Q}$ are commonly called *privacy peers* (see, e.g., [15]).

These requirements are formalized as follows:

Functionality 1: *Event Sequence Pooling*

INPUTS:

- Each $\mathcal{D}_j$ inputs $\tilde{h}_j$, its local event history of length L.
- $\mathcal{O}$ inputs $\perp$.
- $\mathcal{Q}$ inputs $\perp$.

OUTPUTS:

- Each $\mathcal{D}_j$ outputs $\perp$.
- $\mathcal{O}$ outputs its share of the global event history $\tilde{y}^{\mathcal{O}}$.
- $\mathcal{Q}$ outputs its share of the global event history $\tilde{y}^{\mathcal{Q}}$.

B. Oblivious Combinatorial Filter Evaluation

Once the event detector's streams have been pooled, the stream of symbols needs to be traced over a filter's labeled edges to generate the associated sequence of colors. Since the symbols are split into pairs of shares, this tracing must be done in some joint fashion. We build on the ideas in [28], [19] to propose our second desired functionality.

In this case, there are two parties: the *querier* who has part of the sequence of symbols and wishes to know the output color stream. The *filter owner*, who knows the filter, and has part of the sequence of symbols. We have one of the privacy

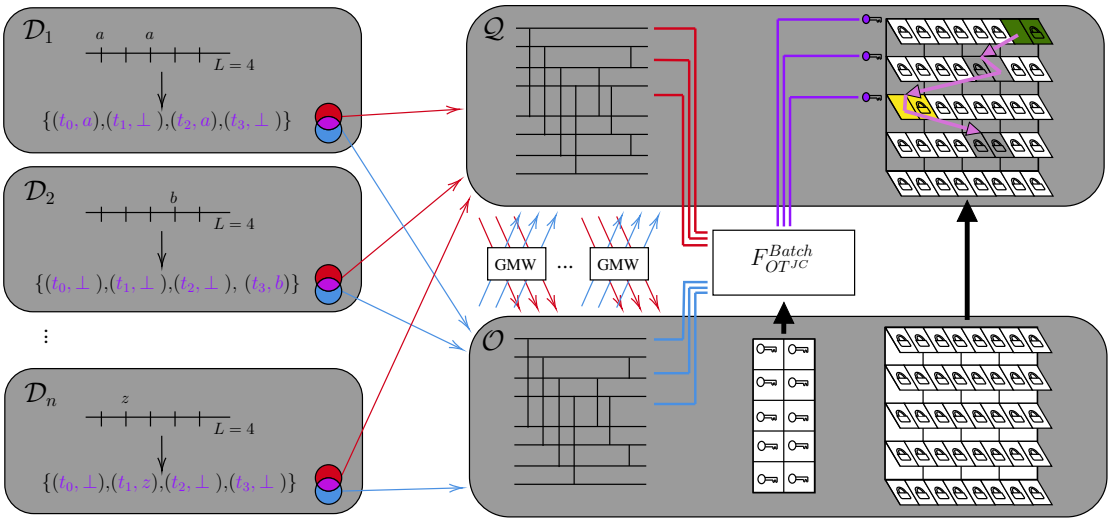

Fig. 2. The overall architecture for MPC-based sensor fusion and combinatorial filter evaluation, shown via a schematic that emphasizes flow and sharing of information. On the far left, event detectors $\mathcal{D}_1, \mathcal{D}_2, \ldots, \mathcal{D}_n$, queue readings for histories of length L. These are then transmitted as split shares to $\mathcal{Q}$ and $\mathcal{O}$. (Visually purple decomposes into red and blue components.) The values are jointly sorted via a sorting network with a comparator circuit using GMW. The resulting time-ordered sequence, still split, is then jointly evaluated as a Moore machine. While $\mathcal{O}$ knows the filter structure, $\mathcal{Q}$ (and $\mathcal{Q}$ alone) learns the sequence of output colors (yellow and green), but nothing more. The grey rectangles represent parties ($n + 2$ in total). Parameter L describes size of local event histories which are pooled for each round. The box $F_{OT^{JC}}^{Batch}$ refers to batch oblivious transfer with joint choice [28].

peers ($\mathcal{Q}$) be the querier, while the other ($\mathcal{O}$) is the filter owner. The functionality is formalized next:

Functionality 2: *Combinatorial Filter Evaluation*

INPUTS:

- $\mathcal{Q}$ inputs $\tilde{y}^{\mathcal{Q}}$, its share of the event sequence.
- $\mathcal{O}$ input $\tilde{y}^{\mathcal{O}}$, its share of the event sequence and combinatorial filter $\mathbf{F}$.

OUTPUTS:

- $\mathcal{Q}$ outputs $\mathbf{F}[\![\tilde{y}]\!]$.
- $\mathcal{O}$ outputs $\perp$.

Next, we examine how to realize these two functionalities.

V. METHODS

Figure 2 provides an encompassing diagram showing the $n + 2$ parties involved: event detectors $\mathcal{D}_1, \mathcal{D}_2, \ldots, \mathcal{D}_n$ and querier $\mathcal{Q}$ and filter owner $\mathcal{O}$. The first functionality is depicted in the portion of the diagram from the left, through splitting of local histories, to the subsequent sorting step (shown via twinned sketches of sorting networks and GMW comparators). The second functionality proceeds to the right: the red and blue lines come together to make a joint choice oblivious transfer, employing cryptographic keys which are used also in tracing through an encoding of the filter as a garbled adjacency matrix. We detail each next.

A. Secure Event Sequence Pooling

To implement Functionality 1, we collect shares of each event history $\tilde{h}_i$ into shares of $\tilde{y}$.

The protocol operates in rounds run at a frequency appropriate for each event detector to have a local history with L symbols. Every round, each $\mathcal{D}_i$ splits its history $\tilde{h}_i$ into additive secret shares using a secure random number generator (RNG) and the XOR operation. Then $\mathcal{D}_i$ sends these two shares (in binary), denoted $b_{\tilde{h}_i}^{\mathcal{O}}$ and $b_{\tilde{h}_i}^{\mathcal{Q}}$, to the respective privacy peers. In Figure 2, the $\tilde{h}_i$ are illustrated by sensors' individual purple sequences, while the transfer of $b_{\tilde{h}_i}^{\mathcal{O}}$ and $b_{\tilde{h}_i}^{\mathcal{Q}}$ are illustrated by the red and blue arrows towards $\mathcal{O}$ and $\mathcal{Q}$. Keeping these shares separate, $\mathcal{O}$ and $\mathcal{Q}$ jointly sort the aggregate history using sorting networks. Finally, $\mathcal{O}$ and $\mathcal{Q}$ obtain shares of a sorted stream of symbols $\tilde{y}$.

Sorting is implemented using the GMW Protocol, selected primarily because it allows convenient interfacing of the outputs of Protocol 1 with Protocol 2. This is because the protocol operates on additive secret shares, allowing parties to retrieve shares of the output without obtaining the plaintext. The GMW Protocol is efficient [22], and it can also be extended to a multi-party setting. Currently, Functionality 2 expects the global history to be shared between $\mathcal{O}$ and $\mathcal{Q}$, so a multi-party approach would require Functionality 2 to be extended, which is beyond the scope of the present paper.

Protocol 1: *Shared Sequence Assemblage*

INPUTS:

- $\mathcal{D}_i$ inputs secret shares of their histories $b_{\tilde{h}_i}^{\mathcal{Q}}$ and $b_{\tilde{h}_i}^{\mathcal{O}}$.
- $\mathcal{Q}$ and $\mathcal{O}$ previously agree on a sorting network S and a compare-exchange circuit C.

OUTPUTS:

- $\mathcal{D}_i$ output $\perp$.
- $\mathcal{Q}$ and $\mathcal{O}$ share the time-ordered event sequence $\tilde{y}$ element-wise between $b_{\tilde{y}}^{\mathcal{Q}}$ and $b_{\tilde{y}}^{\mathcal{O}}$.

For each component, further elaboration follows.

1) Padding inputs: Padding event detectors' inputs with ϵ symbols, as previously mentioned, ensures $|\tilde{h}_i| = L$, $i \in \{1 \ldots n\}$. This is important so that information —such as the frequency at which $\mathcal{D}_i$ detects events– is not revealed.

2) Creating Secret Shares from Event Detectors: The j^{th} element of an event history $\tilde{h}_i[j]$ is a tuple (t_j, s_j), where t_j is a time and $s_j \in Y$. Any such tuple can be represented as an ℓ-bit binary string, for $\ell = \tau + \lceil \log |Y| \rceil$. Each detector $\mathcal{D}_i$ for $i \in \{1 \ldots n\}$ splits its history $\tilde{h}_i$ into shares by generating random values $r \leftarrow_R \{0,1\}^\ell$ for each element in its history. Then $b_{\tilde{h}_i}^{\mathcal{O}}$ is the resulting sequence of ℓ-bit random values. Let each value $b_{\tilde{h}_i}^{\mathcal{Q}}[j] = b_{\tilde{h}_i}^{\mathcal{O}}[j] \oplus b_{\tilde{h}_i}[j]$, $j \in \{1 \ldots L\}$. Some

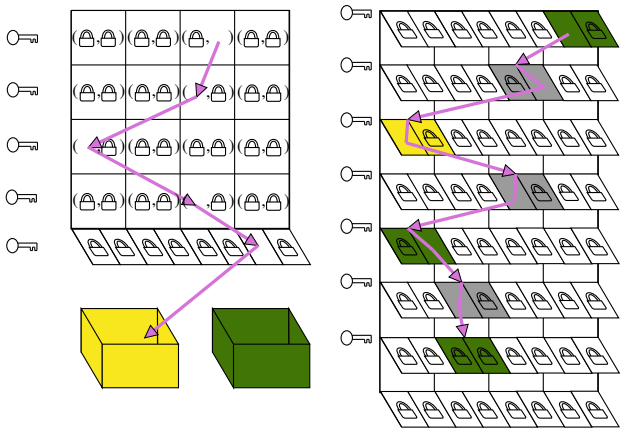

Fig. 3. Oblivious evaluation of DFAs vs. Moore machines. [left] Mohassel's oblivious DFA evaluation. [right] Our Moore machine variant.

(arbitrary) predefined ordering of detectors is known by both $\mathcal{O}$ and $\mathcal{Q}$, which collect the sequences to maintain pre-sorted subsequences, as is formalized next.

3) Sorting: $\mathcal{Q}$ and $\mathcal{O}$ will have concatenated all received histories into a shared unordered global history, and they will proceed to sort the global history in increasing order over time. Jónsson, Kreitz, and Uddin [15] address sorting in the MPC context using sorting networks, and we apply their approach directly with only minor modifications. A sorting network is a structure that specifies an ordering of swaps to sort in place any sequence of pre-defined length. The networks are oblivious to the data of any input sequence. Jónsson et al. [15] use Batcher's odd-even merge sorting network [2] to specify swaps, and perform swaps via an MPC sub-protocol, *compare-exchange*, utilizing MPC primitives. Because sub-sequences of our global history are already sorted, an odd-even merge network suffices. Also, as we are already employing additive secret shares, and they are very effective in practice [22], we use them throughout. Hence, we opt to have $\mathcal{O}$ and $\mathcal{Q}$ evaluate the compare-exchange circuit jointly via GMW. Finally, $\mathcal{O}$ and $\mathcal{Q}$ will strip the global history of its time keys, resulting in $b_{\tilde{y}}$, an ordered *event sequence*, only shared amongst themselves.

It remains to show that this protocol is correct and secure.

Theorem 1: Protocol 1 is correct and secure.

Proof (sketch): Both correctness and security come from Jónsson et al. [15], since the GMW Protocol is an MPC primitive. The other modifications introduced are entirely peripheral to security.

B. Oblivious Combinatorial Filter Evaluation

Next, we build on prior results for Oblivious DFA evaluation [19], [28]. We extend that work to support oblivious Moore machine evaluation under the semi-honest model. First, to make the connection, we must translate an arbitrary combinatorial filter to a binary alphabet Moore machine. By a binary Moore machine we mean an $M = (Q, \{0,1\}, \Delta, q_0, C, c)$ composed of a finite set of states Q, the symbols 1 and 0, a transition function $\Delta : \{0,1\} \times \Sigma \to Q$, an initial state q_0, a finite set of colors C, and a coloring function $c : Q \to C$.

As we create a binary Moore machine M from a filter $\mathbf{F}$, states and corresponding colorings for M are given directly from the filter. For each state $q_i \in Q \cap V$, add states and transitions to create a complete binary tree of depth $\lceil \log |Y| \rceil$, with leaves transitioning to another original state, specified by the edges E. Create a new color *grue*, and assert that new states $q_j \in Q \setminus V$ map to this color $c : q_j \to$ *grue*. Note that in our construction, ϵ will result in a self-loop. The Moore machine M constructed in this way from a filter $\mathbf{F}$ has the property that for all $\mathbf{F}[\![\tilde{y}]\!] = \zeta_0\zeta_1 \ldots \zeta_{|\tilde{y}|}$ and $M[\![b_{\tilde{y}}]\!] = \eta_0\eta_1 \ldots \eta_{|b_{\tilde{y}}|}$, where $b_{\tilde{y}}$ is the binary representation of $\tilde{y}$ with $\lceil \log |Y| \rceil$-bits per symbol, then $\zeta_0 = \eta_0$, $\zeta_1 = \eta_{\lceil \log |Y| \rceil}$, $\ldots$, $\zeta_k = \eta_{k \cdot \lceil \log |Y| \rceil} \ldots$, $\zeta_{|\tilde{y}|} = \eta_{|\tilde{y}| \cdot \lceil \log |Y| \rceil} = \eta_{|b_{\tilde{y}}|}$.

Mohassel et al. [19] introduce the *DFA Matrix* (see Figure 3 [left]) as a building block to oblivious DFA evaluation. A binary DFA Matrix M represents states $s_i = (j,k)$ as pairs of transitions $s_i \to_0 s_j$ and $s_i \to_1 s_k$, where $s_i \in Q$. Evaluating a binary DFA Matrix on a bit sequence b involves choosing the $b[i]^{\text{th}}$ element of the current state pair at each row, then advancing to the next row. In the final row of the matrix, a string is either 'accepted' or 'rejected' by the DFA matrix. A DFA Matrix of size $n \times |Q|$ is needed to process sequences of length n on a DFA with states Q. In our case, we wish to have Q obtain the colors along the evaluation path in an incremental fashion. We can encode a binary Moore machine matrix in the same way as a DFA Matrix by simply adding a color label to each state pair to form a triple, $s_i = (j,k,c(i))$, and by outputting colors at each state rather than producing only an 'accept'/'reject' output in the last row (see Figure 3 [right]). In this case, we require the Moore machine matrix to be of the size $(|b_{\tilde{y}}| + 1) \times |Q|$ because the machine outputs one color at each state, unlike the DFA Matrix formulation, which may output one of two different results at the special final case. Following the definition of a Moore machine, there is a corresponding output of size $|b_{\tilde{y}}| + 1$, since a color is output before processing any input. Our formulation does not include a special output row because the matrix is designed to extend in response to new inputs.

Mohassel et al. go on to permute a DFA Matrix into a *permuted matrix* PM by shuffling the states randomly but keeping the correctness of the transitions. Thereafter, a *garbled matrix* is produced so that, for any particular input string, only an authorized evaluator can reveal the states along the evaluation path, while the rest of the matrix is computationally indistinguishable from random. The method for constructing a garbled binary Moore machine matrix that can be evaluated on the joint input of $\mathcal{O}$ and $\mathcal{Q}$ is largely based upon [28], and has the following steps:

First, start with binary Moore machine matrix M of triples, and create a permuted matrix PM. Second, create a matrix RM of size $(|b_{\tilde{y}}| + 2) \times |Q|$ of random values $RM[i,j] \xleftarrow{R} \{0,1\}^{\kappa}$, where κ is the security parameter. Then, for each row i, create a pair of *garbled keys* $k_0^i, k_1^i \xleftarrow{R} \{0,1\}^{\kappa + \log |Q| - 1}$. Then, manipulate the last bit of the garbled keys with the *point-and-permute* [4] bit $\sigma_i \xleftarrow{R} \{0,1\}$, which will assist evaluator in choosing the right tuple element to decrypt, $k_0^i \leftarrow (k_0^i || \sigma_i)$, $k_1^i \leftarrow (k_1^i || (1 - \sigma_i))$.

We will use RM to obscure all matrix elements from each other, and employ garbled keys to obscure the two transitions

in each triple from one another. The garbled matrix entries will be updated as follows:

$$GM[i,j]_0 \leftarrow (PM[i,j]_0||RM[i+1, PM[i,j]_0]_0) \oplus k_0^i$$
$$GM[i,j]_1 \leftarrow (PM[i,j]_1||RM[i+1, PM[i,j]_1]_1) \oplus k_1^i$$
$$GM[i,j]_2 \leftarrow c(PM[i,j]_2)$$

When examining state j in row i, $PM[i,j]$ gives the transitions away from the state, and the color of this state. Matrix RM will be used to decrypt the elements that $PM[i,j]$ points to. Garbled keys are used to mask the values of this pair of items.

For each triple in each row, swap the first two items if $\sigma_i = 1$. Now generate ciphers $p_0^{i,j}, p_1^{i,j} \leftarrow G_1(RM[i,j])$ to mask transitions and $p_2^{i,j} \leftarrow G_2(RM[i,j])$ to mask colors, then mask the elements of each triple: $GM[i,j]_0 \leftarrow GM[i,j]_0 \oplus p_0^{i,j}$, $GM[i,j]_1 \leftarrow GM[i,j]_1 \oplus p_1^{i,j}$, $GM[i,j]_2 \leftarrow GM[i,j]_2 \oplus p_2^{i,j}$.

Once the construction of the garbled matrix is complete, $\mathcal{O}$ sends $\mathcal{Q}$ the initial state index q_0, the pseudorandom generators G_1 and G_2 and the initial random value $r = RM[1, q_0]$. $\mathcal{Q}$ proceeds by retrieving stream ciphers $p_0^{1,q_0}, p_1^{1,q_0} \leftarrow G_1(r)$, $p_2^{1,q_0} \leftarrow G_2(r)$. $\mathcal{Q}$ also needs garbled keys $k_{\tilde{y}}^i$ for each bit in the shared input string $\tilde{y}$, accomplished by *batch oblivious transfer with joint choice* ($F_{OT^{JC}}^{Batch}$). This MPC protocol, secure under the semi-honest model [28], entails $\mathcal{O}$ transferring its input string of joint choice $k_{\tilde{y}}$ to $\mathcal{Q}$ via both parties' shares of the choice, $b_{\tilde{y}}^{\mathcal{O}}$ and $b_{\tilde{y}}^{\mathcal{Q}}$.

Given garbled keys, $\mathcal{Q}$ can evaluate the matrix row-by-row. At each row i, decrypt $output\_color \leftarrow GM[i,q_i]_2 \oplus p_2^{i,q_i}$ to obtain the initial color. Also, for each row i and state j, $\mathcal{Q}$ decrypts $GM[i,j]_c \leftarrow p_c^{i,j} \oplus k_i \oplus GM[i,j]_c$, where c is the last (point) bit of k_i. $\mathcal{Q}$ can repeat this procedure, finding the next stream ciphers $p_0^{i,j}, p_1^{i,j}, p_2^{i,j}$ with every new state visited, and outputting the color $GM[i,j]_2 \oplus p_2^{i,j}$. This is formalized via Protocol 2.

Protocol 2: *Obliv. Moore Machine Evaluation w/ Joint Input*

COMMON INPUTS:

- Security parameter κ, length of input $|\tilde{y}|$, the size of the event space $|Y|$, the number of states $|Q|$, the number of colors $|C|$, the pseudorandom number generators $G_1 : \{0,1\}^\kappa \rightarrow \{0,1\}^{\kappa+\lceil \log |Q| \rceil}$, where $G_2 : \{0,1\}^\kappa \rightarrow \{0,1\}^{\lceil \log |C| \rceil}$.

PRIVATE INPUTS:

- $\mathcal{O}$ inputs its share of the event sequence, $b_{\tilde{y}}^{\mathcal{O}}$, and a Moore machine matrix M.
- $\mathcal{Q}$ inputs its share of the event sequence, $b_{\tilde{y}}^{\mathcal{Q}}$.

OUTPUTS:

- $\mathcal{O}$ outputs $\perp$.
- $\mathcal{Q}$ outputs the color stream $M[\![\tilde{y}]\!]$.

Now, turn to the correctness and security of the protocol.

For a protocol to follow the semi-honest model, it must produce the correct output, and guarantee that nothing can be learned during execution other than what can be obtained from inputs and outputs. The simulation proof technique [18] prescribes an analysis of the inputs and outputs of a party by constructing a polynomial-time simulator of its real view, and asserting that all intermediate data obtained can be faked by the simulator, such that no polynomial-time distinguisher could tell the difference between simulated and real data. For more complete details on the role of the simulation paradigm in *Theorem 2*, please refer to simulator construction for semi-honest adversaries [18].

For Garbled Moore machine evaluation on joint input to be correct, the output of the ungarbled and garbled Moore Machines must match, given the same input: $M[\![y]\!] = GM[\![y]\!]$.

Lemma 1: Protocol 2 evaluates the Moore Machine correctly: given input string obliviously shared as $b_{\tilde{y}}^{\mathcal{Q}}$ and $b_{\tilde{y}}^{\mathcal{O}}$, the protocol outputs $M[\![b_{\tilde{y}}^{\mathcal{Q}} \oplus b_{\tilde{y}}^{\mathcal{O}}]\!]$.

Proof: To evaluate the garbled matrix, $\mathcal{Q}$ must keep track of several values: the *current row*, the *state index*, and the *random seed*, represented by i, q_i and r_i, respectively. Initially, $\mathcal{Q}$ gets q_0 and r_0 from $\mathcal{O}$, then sets $r_1 \leftarrow r_0$, $q_1 \leftarrow q_0$ and $i \leftarrow 1$. Then, for each row $i \in \{1 \ldots |\tilde{y}|\}$, $\mathcal{Q}$ performs the following procedure. First, use G_1 and G_2 to generate three stream ciphers: $p_0^{i,q_i} \leftarrow G_1(r_i)$; $p_1^{i,q_i} \leftarrow G_1(r_i)$; $p_2^{i,q_i} \leftarrow G_2(r_i)$.

The color output at the current state is obtained:

$$output\_color \leftarrow GM[i,q_i]_2 \oplus p_2^{i,q_i}. \quad (1)$$

Next, $\mathcal{Q}$ obtains the garbled key k_c^i, corresponding to the shared input $\tilde{y}_i$. The *point-and-permute* approach was taken to construct each garbled key, such that the last bit c of k_c^i is the index of the *correct* state transition. Below, α is just $\tilde{y}_i$, the i^{th} bit of the input being processed.

$$GM[i,q_i]_c \leftarrow (PM[i,q_i]_\alpha||RM[i+1, PM[i,q_i]_\alpha]) \oplus p_c^{i,q_i} \oplus k_c^i.$$

During construction of a matrix item in column j, $GM[i,j]_0$ and $GM[i,j]_1$ are not swapped if $\sigma_i = 0$. Consider that $\mathcal{Q}$ comes prepared to $GM[i,j]$ with a garbled key $k_0^i = (\{0,1\}^{\kappa+\lceil \log |Q| \rceil - 1}||0)$. The elements were not swapped, so $GM[i,j]_0$ is indeed encrypted with k_0^i. Likewise, a $k_1^i = (\{0,1\}^{\kappa+\lceil \log |Q| \rceil - 1}||1)$ would match with $GM[i,j]_1$. If $\sigma_i = 1$ during construction of a $GM[i,j]$, the elements of $GM[i,j]$ are swapped. Observe that $GM[i,j]_0$ is *still* the item which is encrypted with k_0^i, and $GM[i,j]_1$ is encrypted with k_1^i.

Given the fact that c points $\mathcal{Q}$ to the correct transition item, $\mathcal{Q}$ obtains the next index, $q_{i+1} \leftarrow PM[i,q_i]_c$, and the seed, $r_{i+1} \leftarrow RM[i+1, PM[i,q_i]_\alpha]$, via the XOR operation:

$$GM[i,q_i]_c \oplus p_c^{i,q_i} \oplus k_c^i = (PM[i,q_i]_c \| RM[i+1, PM[i,q_i]_\alpha])$$

Finally, for row $|\tilde{y}|+1$, party $\mathcal{O}$ obtains the color (1) and has no further input to process. ■

We emphasize an important difference between our proof of security and the proof by Zhao et al. [28] for oblivious DFA evaluation. They take a random approach: the simulator generates a random garbled matrix and evaluates it. If the output is the same as the output in the real execution, then the simulator can provide this garbled matrix, which is indistinguishable from that of real execution. Otherwise, try again with another garbled matrix, until one is found with the correct output. As a DFA matrix only outputs a single terminal bit, the above algorithm is probabilistic polynomial-time. In our case, since we need a $|C|$-ary output at each row, their technique would become probabilistically $O(C^n)$. Accordingly, we use a more direct approach to simulate in polynomial-time, an approach akin to that of [19].

Lemma 2: Protocol 2 is private against semi-honest adversaries.

Proof: Under the semi-honest model of secure two-party computation, only one party may be corrupted by an adversary. We consider each case, one at a time:

Corrupted $\mathcal{O}$: We must construct the party's real view via a simulator that is granted $\mathcal{O}$'s input and output, but

no information from other parties. The only work that this simulator must do is to fabricate messages that cannot be distinguished from those $\mathcal{O}$ receives. All messages received during Protocol 2 are only those needed to evaluate $F_{OT^{JC}}^{Batch}$. Therefore, we rely on the privacy of $F_{OT^{JC}}^{Batch}$ to achieve privacy in the case of Corrupted $\mathcal{O}$. The specification and security proof of $F_{OT^{JC}}^{Batch}$ can be found in [28].

Corrupted $\mathcal{Q}$: The simulator $S_{\mathcal{Q}}$ is given the inputs common to both parties, plus $\mathcal{Q}$'s inputs ($\tilde{y}^{\mathcal{Q}}$) and outputs ($M[\![\tilde{y}]\!]$). The simulator will provide a view indistinguishable from $\mathcal{Q}$'s real view using only the inputs and outputs available to it:

$$\{S_{\mathcal{Q}}(\kappa, |Y|, |Q|, G_1, G_2, |\tilde{y}|, \tilde{y}^{\mathcal{Q}}, M[\![\tilde{y}]\!])\} \stackrel{c}{\equiv}$$
$$\{\text{view}_{\mathcal{Q}}^{\pi}(\kappa, |Y|, |Q|, G_1, G_2, |\tilde{y}|, \tilde{y}^{\mathcal{Q}}, M[\![\tilde{y}]\!])\}.$$

Braces are used to show that the views are thought of as distributions over all possible executions of the protocol $\pi =$ Protocol 2.

The view of the real execution can be written

$$\text{view}_{\mathcal{Q}}^{\pi}(\kappa, |Y|, |Q|, G_1, G_2, |\tilde{y}|, \tilde{y}^{\mathcal{Q}}, M[\![\tilde{y}]\!]) \quad = \quad (GM, K, q_0, r_0),$$

where, as before, $r_0 \leftarrow RM[1, q_0]$.

So, $S_{\mathcal{Q}}$ must construct some plausible GM', K', q_0' and r_0'. The simulator uses a modification of the procedure described in [19, sect. 4.3]. We omit details owing to limited space but, in essence, it places uniformly random values in GM', everywhere except along the evaluation path of a single consistent execution.

Like Mohassel et al. [19], we show the indistinguishability of the real and simulated views with an inductive argument. Suppose $D_0 = \{(GM, K, q_0, r_0)\}$ is the distribution of the real view of $\mathcal{Q}$ during execution. To craft subsequent D_{i+1}, $i \in \{1 \ldots |\tilde{y}| - 1\}$, start with D_i and modify the garbled matrix row $GM[i+1]$ by replacing each element, except that which lies on the transit path, with a uniformly random value. In the end, $D_n = \{(GM', K', q_0', r_0')\}$ will be the view that $S_{\mathcal{Q}}$ produces using the algorithm above. The goal is to show that the two distributions are computationally indistinguishable, i.e., $D_0 \stackrel{c}{\equiv} D_{|\tilde{y}|}$.

To show that $D_i \stackrel{c}{\equiv} D_{i+1}$, first observe that the output at row $i+1$, specifically the output color from the Moore machine after processing $\tilde{y}[1...i]$, is unchanged, and transitions pointing from $GM[i]$ to $GM[i+1]$ along the transition path are identical across D_i and D_{i+1}. It only remains to show that the elements replaced in D_{i+1} are indistinguishable from the real garbled matrix entries in D_i.

In D_i, elements of $GM[i+1]$ are encrypted with stream ciphers, which are generated with pseudorandom number generators. Since the seeds we provide to the pseudorandom generators are assumed to be uniformly random in the protocol specification, the stream ciphers are indistinguishable from uniformly random. When the stream ciphers are indistinguishable from uniformly random, the XOR operation makes elements of $GM[i+1]$ indistinguishable from uniformly random.

In D_{i+1}, the elements of $GM[i+1]$ are chosen at random, uniformly. Any two values which are individually computationally indistinguishable from uniformly random will also be computationally indistinguishable from each other, so the row $GM[i+1]$ is indistinguishable across D_i and D_{i+1}.

Transitivity gives $D_0 \stackrel{c}{\equiv} D_{|\tilde{y}|}$, thus $\{S_{\mathcal{Q}}\} \stackrel{c}{\equiv} \{\text{view}_{\mathcal{Q}}^{\pi}\}$. ■

Note that in our implementation, we evaluate a Moore Machine in an iterative fashion. This proof of security can be extended to the entire execution by imagining concatenating all iterative rounds into one round. Messages inbound to $\mathcal{O}$ are simply the communication involved in $F_{OT^{JC}}^{Batch}$. Messages inbound to $\mathcal{Q}$ are the concatenated garbled keys, the concatenated garbled matrix, and the initial state and pad. Finally, the inputs and outputs are the same, so using the above proof in subsequent iterations is admissible, as it is just like increasing the length of input for one execution of the protocol.

Theorem 2: Under the MPC simulation proof paradigm, Lemma 1 and Lemma 2 yield that Protocol 2 is correct and secure in the semi-honest model.

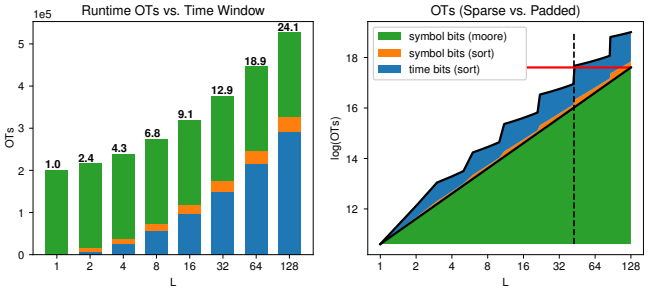

Fig. 4. [left] A breakdown of pre-computed 1-bit random OTs consumed during run time, as the local history length L is increased, and $2^{\tau} = L$. [right] The total number of OTs where we constrain the sparcity of events to at most L per $2^{\tau} = 128$ time units. The red line allows for comparison to when we take single element windows and forgo sorting.

VI. Experimental Results

We now present the proof of concept for the proposed protocols as a simple multi-process python program where $\mathcal{Q}$ and $\mathcal{O}$ run as two processes on the same machine.

A. Implementation Details

We begin with a simulator shown in Figure 1[left] that produces the crossing logs for individual event detectors in the environment. The logs are processed and transformed into shares by an individual process acting as an event detector and transmitted to $\mathcal{Q}$ and $\mathcal{O}$. The primary work is a third multi-process program where $\mathcal{Q}$ and $\mathcal{O}$ sort shares of the event history as per Protocol 1, then evaluate a combinatorial filter on the resulting event sequence, via Protocol 2. We use an additional Python script to pre-compute oblivious transfers as an optimization technique. The code is instrumented to record running times and the number of OTs under different scenarios; a summary appears in Figure 4.

B. Results

We set security parameter $k = 2^8$. Run times were obtained by running our program on the following hardware: Macbook Pro 13-inch, 2017. 2.3GHz Dual-Core Intel Core i5. 8GB 2133 MHz LPDDR3 RAM. Figure 4[left] gives a breakdown of the consumption of OTs for different aspects of the protocol. Sorting requires OTs for the time (for τ width times) and for the symbols (2 bits for all examples). The parameter L and τ increase to right, with $2^{\tau} = L$. Each bar

involves multiple rounds in order to consume the same total number of symbols. Notice, consequently, that the OTs for the Moore machine evaluation are constant, as this is a factor of the total sequence length. As L increases, the cost to sort the inputs increases super-linearly. The conclusion from this appears to be that sorting is ineffective. Indeed, in scenarios where communication between detectors and privacy peers is an abundant resource, or the frequency of detection is high, a simpler construction of Functionality 1 may be warranted. A simpler variant would have a time window $L = 1$, and the sorting aspect of Functionality 1 becomes degenerate.

But the story is a bit more involved: using the preceding data, we are also able to calculate the cost for use of the protocol when the relative density of symbols in the history varies. By pooling data from relatively large time windows but with small L (i.e., $2^\tau \gg L$) we model filtering of sparse inputs. Figure 4[right] shows that before a certain point ($L \approx 42$), when the input symbols are sparse enough, it is useful to sort them and only evaluate the fraction that is known to be non-ϵ (e.g., owing to some a priori model of sparsity). Once L exceeds 42, it is better to spend fewer resources on pooling, and simply evaluate the Moore machine on the inputs on $L = 1$ slices. (For comparison, the red line in Figure 4[right] is the same treatment as Figure 4[left].)

VII. Conclusion

We present an approach to aggregate and filter data from several sensors with privacy guarantees. We introduced and implemented protocol constructions based on the integration of known primitives (like OT and the GMW protocol) and techniques (like sorting, privacy peers, and split secret shares) in Secure Multi-Party Computation. We extend prior work on joint DFA evaluation to the case of Moore machines, proving that security is preserved under the semi-honest adversary model. We implemented our MPC-based sensor fusion and evaluated it in a simple study case. There are several exciting avenues for future work.

The implementation of Protocol 1 makes use of a low-depth comparison circuit [9] to minimize rounds of oblivious transfers between $\mathcal{Q}$ and $\mathcal{O}$, but we do not take advantage of many of the optimizations to the GMW Protocol that are available, including extending oblivious transfers [13], parallelizing oblivious transfers in batch, using multiplication triples [3] and performing load balancing [22]. Moore machine evaluation can be further improved by implementing suggestions found in [28]. This paper has described a first treatment of the combinatorial filtering problem in a private setting that we are aware of; no claims have been made about the absolute efficiency of the approach, but it has been shown by our rudimentary implementation to be practically feasible for small-scale instances. Future work should look at further instances of oblivious sensor fusion.

References

[1] A. B. Alexandru and G. J. Pappas. Secure multi-party computation for cloud-based control. In *Privacy in Dynamical Systems*, pages 179–207. Springer, 2020.

[2] K. E. Batcher. Sorting networks and their applications. In *Proceedings of the April 30–May 2, 1968, Spring Joint Computer Conference*, AFIPS '68 (Spring), page 307–314, New York, NY, USA, 1968.

[3] D. Beaver. Efficient multiparty protocols using circuit randomization. In J. Feigenbaum, editor, *Advances in Cryptology — CRYPTO '91*, pages 420–432, Berlin, Heidelberg, 1992. Springer Berlin Heidelberg.

[4] D. Beaver, S. Micali, and P. Rogaway. The round complexity of secure protocols. In *Proceedings of the Twenty-Second Annual ACM Symposium on Theory of Computing*, STOC '90, page 503–513, New York, NY, USA, 1990. Association for Computing Machinery.

[5] L. Bobadilla, O. Sanchez, J. Czarnowski, and S. M. LaValle. Minimalist multiple target tracking using directional sensor beams. In *Proc. IEEE Int. Conf. on Intell. Robots and Systems (IROS)*, 2011.

[6] C. G. Cassandras and S. Lafortune. *Introduction to discrete event systems*. Springer Science & Business Media, 2nd edition, 2009.

[7] H. Durrant-Whyte and T. C. Henderson. Multisensor data fusion. In B. Siciliano and O. Khatib, editors, *Springer Handbook of Robotics*, pages 867–896. Springer, 2016.

[8] C. Dwork and A. Roth. The algorithmic foundations of differential privacy. *Found. Trends Theor. Comput. Sci.*, 9(3–4):211–407, 2014.

[9] J. Garay, B. Schoenmakers, and J. Villegas. Practical and secure solutions for integer comparison. In *Public Key Cryptography (PKC)*, pages 330–342, 2007.

[10] R. Gennaro, C. Hazay, and J. Sorensen. Text search protocols with simulation based security. In *Public Key Cryptography (PKC)*, pages 332—350, 2010.

[11] O. Goldreich. *Foundations of Cryptography: Volume 2, Basic Applications*. Cambridge University Press, USA, 2004.

[12] O. Goldreich, S. Micali, and A. Wigderson. How to play ANY mental game. In *Proc. of the Symposium on Theory of Computing (STOC)*, pages 218–229, 1987.

[13] Y. Ishai, J. Kilian, K. Nissim, and E. Petrank. Extending oblivious transfers efficiently. In D. Boneh, editor, *Advances in Cryptology - CRYPTO 2003*, pages 145–161. Springer, 2003.

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

[15] K. Jónsson, G. Kreitz, and M. Uddin. Secure multi-party sorting and applications. *IACR Cryptology ePrint Archive*, 2011:122, 01 2011.

[16] J. Le Ny and G. J. Pappas. Differentially private filtering. *IEEE Transactions on Automatic Control*, 59(2):341–354, 2014.

[17] L. Li, A. Bayuelo, L. Bobadilla, T. Alam, and D. A. Shell. Coordinated multi-robot planning while preserving individual privacy. In *Proc. IEEE Int. Conf. on Robotics & Automation (ICRA)*, pages 2188–2194, 2019.

[18] Y. Lindell. How to Simulate It — A Tutorial on the Simulation Proof Technique. In Y. Lindell, editor, *Tutorials on the Foundations of Cryptography*, pages 277–346. Springer, Apr. 2017.

[19] P. Mohassel, S. Niksefat, S. Sadeghian, and B. Sadeghiyan. An efficient protocol for oblivious dfa evaluation and applications. In *Cryptographers' Track at the RSA Conference*, pages 398–415. Springer, 2012.

[20] J. M. O'Kane and D. A. Shell. Automatic design of discreet discrete filters. In *Proceedings of the IEEE International Conference on Robotics and Automation*, pages 353–360, 2015.

[21] J. M. O'Kane and D. A. Shell. Concise planning and filtering: hardness and algorithms. *IEEE Transactions on Automation Science and Engineering*, 14(4):1666–1681, 2017.

[22] T. Schneider and M. Zohner. GMW vs. Yao? efficient secure two-party computation with low depth circuits. In A.-R. Sadeghi, editor, *Financial Cryptography and Data Security*, pages 275–292, Berlin, Heidelberg, 2013. Springer Berlin Heidelberg.

[23] B. Tovar, F. Cohen, L. Bobadilla, J. Czarnowski, and S. M. Lavalle. Combinatorial filters: Sensor beams, obstacles, and possible paths. *ACM Transactions on Sensor Networks (TOSN)*, 10(3):47, 2014.

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

[25] A. C.-C. Yao. How to generate and exchange secrets. In *Proceedings of the 27th Annual Symposium on Foundations of Computer Science*, pages 162–167, 1986.

[26] Y. Zhang and D. A. Shell. Complete characterization of a class of privacy-preserving tracking problems. *The International Journal of Robotics Research*, 38(2-3):299–315, 2019.

[27] Z. Zhang, J. Wu, D. Yau, P. Cheng, and J. Chen. Secure kalman filter state estimation by partially homomorphic encryption. In *Proceedings of the 9th ACM/IEEE International Conference on Cyber-Physical Systems*, pages 345–346. IEEE Press, 2018.

[28] C. Zhao, S. Zhao, B. Zhang, S. Jing, Z. Chen, and M. Zhao. Oblivious DFA evaluation on joint input and its applications. *Information Sciences*, 528:168–180, 2020.