OBLIVIOUS SENSOR FUSION VIA SECURE MULTI-PARTY COMBINATORIAL FILTER EVALUATION

An Undergraduate Research Scholars Thesis

by

WILLIAM E. CURRAN

Submitted to the LAUNCH: Undergraduate Research office at
Texas A&M University
in partial fulfillment of requirements for the designation as an

UNDERGRADUATE RESEARCH SCHOLAR

Approved by
Faculty Research Advisor: Dr. Dylan Shell

May 2021

Major: Computer Science

RESEARCH COMPLIANCE CERTIFICATION

Research activities involving the use of human subjects, vertebrate animals, and/or biohazards must be reviewed and approved by the appropriate Texas A&M University regulatory research committee (i.e., IRB, IACUC, IBC) before the activity can commence. This requirement applies to activities conducted at Texas A&M and to activities conducted at non-Texas A&M facilities or institutions. In both cases, students are responsible for working with the relevant Texas A&M research compliance program to ensure and document that all Texas A&M compliance obligations are met before the study begins.

I, William Curran, certify that all research compliance requirements related to this Undergraduate Research Scholars thesis have been addressed with my Research Faculty Advisor prior to the collection of any data used in this final thesis submission.

This project did not require approval from the Texas A&M University Research Compliance & Biosafety office.

TABLE OF CONTENTS

ABSTRACT

Oblivious Sensor Fusion via Secure Multi-party Combinatorial Filter Evaluation

William E. Curran
Department of Computer Science
Texas A&M University

Research Faculty Advisor: Dr. Dylan Shell
Department of Computer Science
Texas A&M University

This thesis examines the problem of fusing data from several sensors, potentially distributed throughout an environment, in order to consolidate readings into a single coherent view. We consider the setting when sensor units do not wish others to know their specific sensor streams. Standard methods for handling this fusion make no guarantees about what a curious observer may learn. Motivated by applications where data sources may only choose to participate if given privacy guarantees, we introduce a fusion approach that limits what can be inferred. Our approach is to form an aggregate stream, oblivious to the underlying sensor data, and to evaluate a combinatorial filter on that stream. This is achieved via secure multi-party computational 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 semi-honest adversary model. Though the approach enables several applications of potential interest, we specifically consider a target tracking case study as a running example. Finally, we also report on a basic, proof-of-concept implementation, demonstrating that it can operate in practice;

which we report and analyze the (empirical) running times for components in the architecture, suggesting directions for future improvement.

ACKNOWLEDGEMENTS

Contributors

I would like to thank my faculty advisor, Dr. Shell, and the rest of the team, Dr. Bobadilla and Cesar Rojas for their guidance and support throughout the course of this research.

Thanks also go to my friends and colleagues and the department faculty and staff for making my time at Texas A&M University a great experience.

The analyses depicted in *Oblivious sensor fusion via secure multi-party combinatorial filter evaluation* were conducted in part by Dr. Dylan Shell, Dr. Leonardo Bobadilla, Cesar Rojas and William Curran, and were submitted to the IEEE conference on Decision and Control for review on March 25, 2021.

All other contributions to this work were completed by the student alone.

Funding Sources

Undergraduate research was supported by the computer science department at Texas A&M University.

This work was also made possible in part by the National Science Foundation awards IIS-2034123, IIS-2024733, IIS-2034097, and by the U.S. Department of Homeland Security award 2017-ST-062000002. Its contents are solely the responsibility of the authors and do not necessarily represent the official views of the National Science Foundation, the U.S. Department of Homeland Security or the Texas A&M computer science department.

1. INTRODUCTION

It is natural for humans to be mutually distrusting of each other. It is often a matter of safety for communities to act with an abundance of caution towards others who hold unknown motives. Yet, cooperation is also essential to achieving many of our goals. We humans have a limited number of tools to deal with our frequent mutual distrust of one another, and often distrust manifests in stalemate or conflict, each without satisfactory resolution. As robots become more important to how people interact with the world, human problems become robot problems, and robots also need ways to handle distrust of each other. This thesis documents attempts to ameliorate a fraction of distrust amongst robots, enabling mutually distrusting discourse via cryptographic toolsets founded on mathematical certainty.

The main work of this thesis is concerned with processing streams of input, originally from distributed sources. This kind of aggregation and analysis is done frequently today. One name for distributed networks of devices is the Internet of Things, describing the many internet-connected sensing and acting devices distributed throughout the world. With billions of connected devices thrown into the fray, each belonging to some owner(s) with unknown agendas, a vast landscape of fragmented data is produced. Oftentimes, data must be shared or traded between devices to create a more complete picture (consider implied or explicit contracts between smartphone users and application providers, exchanging data for utility). Other times, centrally owned networks of devices must be made robust against potential attack by assuming that any device might be compromised in some way by an adversary. In either case, individual devices can be considered reluctant to share their private data with another party. (We are actually concerned with the entity ‘in control’ of a particular device, whomever that may be, and each will generally be deemed a *party*). In many cases today, parties either exchange their private data with some regret or choose to not trade their data at all. Our work is focused on enabling meaningful exchange of analyses on data, without disclosing any private data, thus, always allowing a policy of healthy distrust amongst parties.

A relevant real-world issue is how assisted living or elderly care might be aided by technology. Consider the following as a case study: Parker's grandma Susie lives independently in her home but might need additional help from time to time. Parker wishes to ensure Susie's safety and comfort, but Parker cannot always visit in-person. Therefore, they have agreed to allow a series of devices present in Susie's smart home to collect data, and report if she needs immediate care. A possible basic functionality might have sensors track her movement in the home, flagging any abrupt changes, such as, "Susie was standing in the kitchen and is now on the floor." To many, this is likely a shocking proposition. Permitting a sensor array to invade one's most private spaces, possibly broadcasting this data for further analysis, is a troubling solution to the problem. However, the main work present in this thesis presents cryptographic protocols, which prove that privacy of the devices' inputs can be preserved in scenarios like this one, while still revealing the desired outputs.

The following subsections of Section 1 introduce the reader to concepts essential to the main work of this thesis, namely *combinatorial filters* and *secure Multi-Party Computation*.

1.1 Filtering

A filter is a function which consumes a stream of input and produces a corresponding stream of output. A popular category of filters is the probabilistic kind, such as the well-known *Kalman Filter* [2]. The Kalman Filter is used in robotics for purposes such as sensor fusion to produce a state estimation result which is more accurate than individual sensors could produce.

We consider a different class of filters, *combinatorial filters* [10], which have simpler discrete inputs and outputs. A combinatorial filter can be represented by a graph with a finite number of nodes and edges (also called states and transitions, respectively), plus a corresponding “coloring” for the nodes, i.e., the output for a given state.

Today, the Internet of Things has been realized by a multitude of small sensors, which often provide simple data on their own, and fusing them together via a simplistic stateful filter may be the best choice in many scenarios.

Research on combinatorial filters, thus far, has considered a central computational entity which owns the input data sources, but we consider the case where data sources are independent parties, each with possibly ulterior objectives than just reporting to a central entity. We wish to establish security for a protocol which evaluates a combinatorial filter on the aggregate, but without disclosing any information about private events to anyone involved in the protocol. To do this, we make use of special cryptographic protocols.

1.2 Cryptography

Cryptography is generally concerned with making certain functionalities robust to attack by potential adversaries. Cryptographic protocols make use of the assumption that computationally “hard” problems exist. Once a problem is thought to be hard, it can be harnessed to keep important data away from adversaries.

A classical problem in cryptography is keeping data secure over a potentially insecure communication channel. We will call the sender Alice and the receiver Bob, while Charlie wishes to listen in on their conversation and learn any details he can. For Alice and Bob to succeed in having a private discussion, Alice must *encrypt* her message, turning the *plaintext* into a *ciphertext* before transmitting it to Bob. At this stage, the goal is for Bob to be able to *decrypt* the message back into the original plaintext, while leaving Charlie completely in the dark. In other words, we wish the ciphertext to look indistinguishable from any random message in the ciphertext space, so Charlie won't be able to decrypt it with ease. From a computer science perspective, we are satisfied with the notion of *computationally indistinguishability* to protect against this kind of snooping. Additionally, Charlie should not be able to harm the integrity of the data without detection by Bob. In this example, we also see that randomness plays a large role in cryptography. Indeed, secure generators of (practically) random numbers are an important building block of many protocols.

1.2.1 Computational Indistinguishability

Computational Indistinguishability is the most atomic cryptographic concept studied in this thesis, and it is harnessed in proofs of security in Section 2 and Section 3. Computational indistinguishability, defined in [3], is used to compare two *probability ensembles*, which are each sequences of *probability distributions*. We consider probability distributions, which, given a domain of natural numbers, describe the probabilities of sampling each value in this domain. We will consider probability ensembles $X = \{X_n\}_{n \in N}$ and $Y = \{Y_n\}_{n \in N}$. Each X_n and Y_n are distributions with sampling domains $\{1, \dots, n\}$. X and Y are computationally indistinguishable if for every probabilistic polynomial-time algorithm D every polynomial p, and all sufficiently large n,

$$|\Pr[\mathrm{D}(X_n) = 1] - Pr[D(Y_n) = 1| < \frac{1}{p(n)} \tag{1.1}$$

Imagine the bell curve of a normal distribution and the flat line of a uniform distribution. Clearly, there is a trend in the normal distribution that is not present in the uniform distribution. Looking at the sequence of infinite variations on these distributions, there are bound to be many times when the two are distinguishable. In contrast, two computationally indistinguishable probability ensembles would mirror each other to the extent that a computational entity could not tell any difference between the two.

D of *Equation 1.1* can be thought of as a *distinguisher* entity who has realistic computational bounds. Essentially, the definition says that even the best distinguisher that could possibly be designed could not consistently tell two distributions apart over the range of possible n. For a distinguisher's algorithm to output '1' is not necessarily meaningful in itself, but when the distinguisher outputs the same value 1 for two distributions, there is a similarity observed.

A probabilistic polynomial-time algorithm is one which runs in time proportional to a polynomial of n, except for a low probability. This is the upper bound of computational ability we need to consider for any adversary, so long as n is sufficiently large (algorithms which asymptotically dominate polynomial time are considered infeasible for large input if $P \neq NP$).

1.2.2 Pseudorandom Generators

Pseudorandom generators have become essential to cryptography, as they make practically random values accessible to implementations and proofs of cryptographic protocols. This thesis will make frequent use of pseudorandom generators for foundational assumptions of security. A family of pseudorandom functions $F_s\colon \{0,1\}^k \rightarrow \{0,1\}^l$, indexed by a seed $s \in \{0,1\}^n$, is *pseudorandom* if it is efficient to compute a value $F_s(x)$, given x and s, and is computationally indistinguishable from a uniformly random distribution, even if a distinguisher has access to pairs $\big(x_i, F_s(x_i)\big)$ and is able to adaptively choose $x_i's$ [5]. In the remainder of the thesis, a family of pseudorandom functions which also keeps track of its internal state will be called a pseudorandom generator G, which can be indexed by a seed to obtain a generator function $\mathrm{G(s)}\colon \{0,1\}^{\mathrm{k}} \rightarrow \{0,1\}^{\mathrm{l}}$ and can be invoked to obtain a pseudorandom l-bit value $r \leftarrow G(s)$.

1.3 Security Models

Our goal is to securely compute a filter amongst multiple parties. Before we can create secure protocols to do this, we need to define what ‘secure’ means. There are multiple security models to choose from, and we briefly discuss two of them: *differential privacy* and *secure Multi-Party Computation (MPC)*.

Differential privacy is a security model which attempts to minimize output leakage, i.e., what can be learned about individuals, given the outputs of functionalities and protocols [3]. Differential privacy does not provide concrete claims about what can be learned by a computational entity during the execution of protocols. Therefore, differential privacy is largely an outward-facing security model.

Secure Multi-Party Computation, an inward-facing security model, considers computation amongst several parties who wish to operate on their combined private data without divulging their individual data to any other parties. Under MPC, a protocol is secure if and only if the protocol correctly computes a functionality amongst some parties, and nothing can be learned from the execution of the protocol other than what can be learned by an individual’s inputs and outputs alone. MPC is a strong way to pointedly break down the security of a protocol, leaving the output leakage problem for another time. However, the user of an MPC protocol must understand that MPC does not provide a privacy panacea. The simplest example of a possible MPC protocol which does not preserve the privacy of its participants is one which simply provides all ‘private’ inputs as public output. Clearly, the output leakage question is worth further exploration, as is noted in *Section 3.1.10*, but this thesis will use MPC framework.

MPC provides several tiers of security, which is practical for building protocols brick-by-brick, rather than trying to achieve lockdown security all at once. We can achieve security in the *semi-honest adversary model* or the *malicious adversary model*.

1.4 Secure Multi-Party Computation

The semi-honest model of secure Multi-Party Computation is the first touchstone a protocol tries to reach before achieving higher levels of security. To achieve security against a malicious adversary, one must prove that a more active adversary could not violate the protocol's privacy or correctness properties.

1.4.1 Security by Simulation

Once we understand what an MPC protocol is (i.e., what can be learned is limited to the input and output alone), we need to be able to show that a protocol satisfies this property. To do this, we first must make some assumptions about our adversarial counterparts.

In all 2-party cases, we are up against a single adversary, who has *corrupted* one party. If both parties were simultaneously corrupt, then the protocol cannot be guaranteed to protect us. (Consider that if both parties are corrupt, they could even be under the control of the *same* adversary. In this case, the two parties may be considered under the umbrella of one party in a sandbox world, where "anything goes.")

We give an upper bound to the strength of the adversary by assigning them the labels *semi-honest* or *malicious*. Security against a semi-honest adversary is the weaker claim; it says that the corrupted party is cooperative, but curious, in that it will execute the protocol to specification, but it may try to learn extra information "on the side." Security versus a malicious adversary means that the protocol is robust against even the most aggressive attacks by participants. For instance, a malicious adversary might send malicious data intended to harm the other party, or such an adversary might drop the connection on a whim. Even simply dropping connection can obliterate the security of a protocol which is only secure against semi-honest adversaries. As such, we see that the semi-honest claim can appear quite weak in comparison. However, a protocol secure under the semi-honest model can be transformed into one under the malicious security model via various methods, and it is typically a checkpoint along the way to more robust enhancements. (A protocol secure under the malicious adversary model is also secure against semi-honest adversaries.)

One generalized way that protocols secure versus semi-honest adversaries can be upgraded to secure against malicious adversaries is by using the process called the GMW Compiler. Essentially, this methodology proscribes taking your semi-honest protocol and buffing it up with *zero-knowledge proofs* [4].

In our case, we only concern ourselves with semi-honest adversaries, though the existence of the malicious adversary model is important to understand possible next steps forward in our research.

Once an adversary model is chosen, one needs to prove the MPC property that nothing is learned by involved parties other than their respective private inputs and outputs. This is done via the simulation paradigm [7], which compares the *view* during execution of a potentially corrupted party in the real world with a view generated by an imaginary simulator. A party's view during execution is the collection of its inputs, outputs, and any intermediate messages it may receive from other parties. A proof by simulation for security against a corrupt party is done by allowing the theoretical simulator access to *only* the party's inputs and outputs. If the simulator can generate faked versions of any other intermediate information which would fool a distinguisher (computationally indistinguishable), when only given the input and output, then this proves that the intermediate information obtained does not give away anything more than the input and output does.

1.4.2 Public-key Cryptosystems

Public-key cryptosystems are schemes which describe how to encrypt and decrypt data, such that data in its ciphertext form is computationally indistinguishable from uniformly random. They make use of *trapdoor permutations* [6] to make encryption ($f(x) = y$) efficient to compute and decryption ($f^{-1}(y) = x$) impossible to compute in polynomial time, unless given a secret trapdoor τ. We can use public-key cryptosystems as a building block to construct more complex MPC primitives. In *oblivious transfe*r, for example, classic implementations of the functionality use public-key cryptosystems. While public-key cryptosystems are quite popular, it is often a goal to minimize public-key operations, as they are still computationally expensive.

1.4.3 Oblivious Transfer

Oblivious transfer (OT) was first theorized by Rabin [9], who imagined a random transfer of one of two bits from a sender S to a receiver R, such that neither S nor R know which bit was received. The concept has been expanded upon much since then, including variations with implementations and optimizations [6] [11]. 1-out-of-2 oblivious transfer is one of the most widely used variations on OT. This is where R supplies a *choice bit, b*, to select a message (generalized to any number of bits) out of the two messages m_0, m_1, which S provides. R is guaranteed to get its choice of messages, $\mathrm{m_b}$.

A lesser-known OT variant, which we call upon, is called 1-out-of-2 OT with joint choice (OT_{JC}) [11]. This version dictates that both S and R input choice bits b_s and b_r, respectively, and R receives the message of joint choice. In this case, the idea of "joint" choice means that the choice bits from S and R will be added with modulo 2, such that S receives $m_{b_s \oplus b_r}$. OT_{JC} is especially helpful, we have found, to integrate an MPC protocol that uses an *additive secret-sharing scheme* with another protocol that requires oblivious transfers.

One simple semi-honest protocol for to accomplish 1-out-of-2 OT is this: R generates a keypair (sk, pk) with a public-key cryptosystem and generates a random pk' in the range of the public key generator [4]. If R's choice bit $b = 0$, then send (pk, pk') to S. If $b = 1$, send (pk', pk) instead. S will receive (pk_0, pk_1), encrypt its messages m_0, m_1 with the respective keys, then send the resulting ciphertexts (c_0, c_1) back to R. Finally, R decrypts c_b with pk, obtaining m_b.

Informally, this protocol is secure under the semi-honest model because if the parties behave according to the protocol, R will be able to decrypt only the message of its choice, achieving correctness under the semi-honest model. The protocol also achieves security under the semi-honest model because R does not have a key to decrypt the other message, and S cannot tell pk from pk'. It is trivially insecure in the malicious model, though, as a corrupt R might generate two keypairs (sk, pk), (sk', pk'), which would then allow R to recover both messages.

To do 1-out-of-2 OT with joint choice, S and R may do the same as above, except S will swap its messages if its choice bit is 1 before it encrypts them. R obtains the result as before, but now the message obtained is $m_{b_s \oplus b_r}$ [11].

1.4.4 Garbled Circuit Protocols

Andrew Yao designed the first MPC protocol, which uses a *garbled circuit* to obfuscate intermediate computation from the two parties executing the protocol. Yao's garbled circuit protocol compiles a deterministic functionality into a 'garbled' boolean logic circuit. Two parties participate in this protocol, and it can be extended to multiple parties, but with additional work [4].

GMW Protocol, also using garbled circuits, arose in the wake of Yao's grand entrance. GMW Protocol is used in *Section 2.5.1.* This protocol makes use of *additive secret shares*, which are simple shares r and $x \oplus r$ of a value x. These are made by obtaining a random (or pseudorandom) number and adding the random number with the value to be shared, then taking the modulo 2 of the sum (XOR operation). The result looks just as random as the random number, and now both shares look computationally indistinguishable from uniformly random. One of these shares can be exchanged with another party to share the value, and the scheme is generalizable to many parties. The value x is retrieved by combining shares $x = (x \oplus r) \oplus r$.

1.5 References

[1] D. Beaver, "Correlated Pseudorandomness and the Complexity of Private Computations," in *Proceedings of the Twenty-Eighth Annual ACM Symposium on Theory of Computing*, 1996.

[2] H. Durrant-Whyte and T. C. Henderson, "Multisensor data fusion," 2016, pp. 867-896.

[3] C. Dwork and A. Roth, "The Algorithmic Foundations of Differential Privacy," *Found. Trends Theor. Comput. Sci.,* vol. 9, no. 3–4, pp. 211-407, aug 2014.

[4] D. Evans, V. Kolesnikov and M. Rosulek, "A Pragmatic Introduction to Secure Multi-Party Computation," Now Publishers, 2021.

[5] O. Goldreich, in *Foundations of Cryptography*, Now Publishers, 2005, pp. 24-25.

[6] R. Impagliazzo and S. Rudich, "Limits on the Provable Consequences of One-way Permutations," in *Advances in Cryptology --- CRYPTO' 88*, 1990.

[7] Y. Lindell, "How To Simulate It – A Tutorial on the Simulation Proof Technique," in *Tutorials on the Foundations of Cryptography*, New York City, Springer, Cham, 2017.

[8] M. O. Rabin, "How To Exchange Secrets with Oblivious Transfer," IACR Eprint archive, 1981.

[10] 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),* vol. 10, no. 3, p. 47, 2014.

[11] C. Zhao, S. Zhao, B. Zhang, S. Jing, Z. Chen and M. Zhao, "Oblivious DFA evaluation on joint input and its applications," *Information Sciences,* vol. 528, pp. 168-180, 2020.

2. OBLIVIOUS SENSOR FUSION VIA SECURE MULTI-PARTY COMBINATORIAL FILTER EVALUATION[†]

2.1 Introduction

Many practical applications of either humanitarian or military relevance - including situational awareness, sense-making, and activity modeling problems - involve checking properties, answering queries, and drawing conclusions, from a stream of sensor readings. Motivated by pervasive sensing scenarios, the present paper studies a setting in which multiple sensors, physically distributed across an environment, employ communication and computation form a consolidated estimate by fusing their sensed data. In contrast to prior work on such problems, we are interested in circumstances where the elements comprising the system need not fall under aegis of the same authority. Though participating as honest parties, they may be inquisitive (e.g., where one purchases access to data streams in a market of competing service providers). Or they may have been compromised so as to disclose data, or to stash information they handle to divulge later (e.g., to an adversary who can instrument but not modify devices or communication channels). The objective, then, is to fuse data from multiple devices, while also limiting what additional information may be learned by such devices (here, 'additional' is taken as meaning over-and-above what is known from their individual views).

Our treatment focuses on a class of finite-state transducers called *combinatorial filters* [25]. Effectively, these act as discrete state estimators: converting a stream of inputs into a stream of outputs, where the former stand for sensor readings, while the latter form estimates of properties of interest, encoded abstractly. *Figure 2.1 [left]* provides a (simple) concrete example:

[†] © 2021 IEEE. Reprinted, with permission, from [32].

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, thereby generating a sequence symbols. Processing this sequence by tracing over the correspondingly labeled edges, the 4-state filter in *Figure 2.1 [right]* concisely answers the query *are the two agents together or not?*

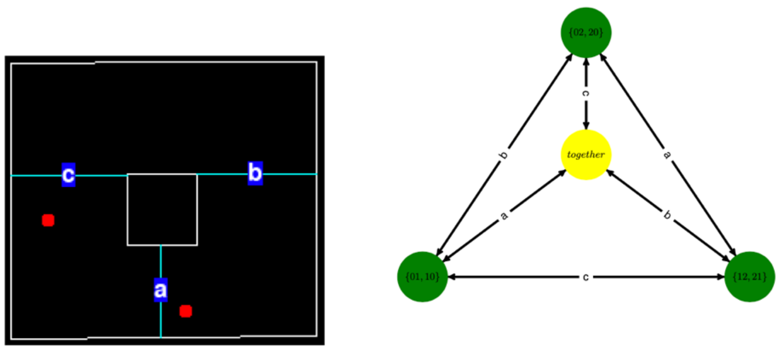


Figure 2.1: A simple scenario: we track the state of two agents moving in a rectangular room that contains a single central obstacle. A small combinatorial filter fuses information obtained from three break-beam sensors (‘a’, ‘b’, and ‘c’) to determine whether the agents are together or not. [left] A 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, tracks whether the agents are together (yellow) or apart (green).

In this basic example, we imagine that in an implementation the beam sensors are realized as three different embedded devices connected via a communication network, and each has basic compute capabilities. In addition to the sensors, some separate device (or party) holds the graph structure that describes the filter. Finally, there is a *querier*, some party who is interested in the current output of the filter (i.e., the color: yellow or green, together or apart). This paper shows how to ensure 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: 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, which is appropriate for devices considered cooperative but curious, devices constrained to obey the prescribed protocol. This model allows claims to be made about information that the protocol leaks (in contradistinction to that which implicitly disclosed via the input-output relationships).

Though it is more common for MPC operations to treat data in a batch fashion, incremental stream-based evaluation is a fundamental part of real-time filtering. The approach we will describe pools, via a mechanism that ensures privacy, data that the sensor monitoring devices aggregate locally. This happens at some fixed rate, for transmission cannot depend on the data received because otherwise the rate and/or quantity of communication would leak information about how much activity is occurring.

Nevertheless, in the encoding we employ, when events are known a priori to occur infrequently (for instance, when the velocity of the agents in *Figure 2.1 [left]* is bounded) this can 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.

The primary contribution of this paper is the application and integration of MPC protocols, including extensions to techniques, to the practical problem of secure filtering for data sensor fusion. Where our protocol extensions warrant it, we present security proofs for the modification. 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.

2.2 Related Work

This paper addresses a discrete estimation problem in an unconventional computational setting, drawing on multiple MPC techniques and related concepts. Primarily to provide context, here we provide a brief and basic overview to help situate the present work within the broader literature. More and deeper technical connections will be made in the sections that follow, once the basic definitions and problem formalization have been introduced.

2.2.1 Filters, Estimators, and Trackers

Combinatorial filters, a term introduced in [25] and related to discrete event systems [7], 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 [4], and have been the subject of study under the broader banner of minimalism [16]. We are not aware of prior work which has considered multi-sensor or decentralized fusion for such filters; this differs markedly from the case of traditional probabilistic filters [6].

A notable piece of work, within the probabilistic filtering setting, is that of Ny and Pappas [17], which relates to our work in that it considers multiple participants contributing input data and is concerned with privacy. In their work, the differentially private model [7] is employed, seeking to limit information about the individual contributions to the output. 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 [26], and discreetness [22]. Some work has also examined privacy-preserving tracking problems [29].

2.2.2 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 [8]. Each party only learns the output of the computation, even when inimical participants collude in to trying learn more about the inputs of the others.

An early problem examined by Andrew Yao [28] is the 'Millionaire's problem', where two individuals determine who has greater wealth (evaluating a '$\leq$'), but without revealing their own bank balances. Yao's solution involved one party generating a truth table with all possible outcomes, encrypting the outputs, and generating a permutation to produce what is known as a 'garbled circuit.' The garbled circuit and keys needed to decrypt the output are then shared. Next, a process termed 'oblivious transfer' (OT) is used to provide the inputs needed to evaluate the garbled circuit.

Oblivious transfer 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). They form an important building block for Secure MPC protocols and, in reporting the performance of our implementation, we will report the quantity of primitive OTs involved.

2.2.3 GMW Protocol

Within the semi-honest setting considered herein, several general protocols to compute deterministic functions exist [12]. We make use of the Goldreich-Micali-Wigderson (GMW) Protocol [11] 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 [8]. The core idea behind this protocol is that each party supplies additive shares for circuit inputs, and they evaluate C on shared values [3][13].

2.2.4 Privacy in Robotics & Control

Recently, researchers have begun to apply MPC to problems in robotics, control, and estimation [18], [1], [30].

2.3 Preliminaries

2.3.1 Basic Notation

Let $x \leftarrow_R \{0,1\}^n$ represent the generation of a random binary string of length n using a cryptographically secure random number generator. Let $G: \{0,1\}^n \rightarrow \{0,1\}^m$ be a pseudorandom number generator with n-bit seed and m-bit output. Let $u||v$ represent the concatenation of binary bit strings u and v. We write $u \oplus v$ for the bitwise XOR operation on binary bit 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 [3].

Our setting will consider $n + 2$ devices, each with access to some partial share of the information, which we treat as private. Interaction between these devices is primarily pairwise, so that two-party functionalities are our natural focus.

Abstractly, a *two-party functionality* is a function $f: \{0,1\}^* \times \{0,1\}^* \rightarrow \{0,1\}^* \times \{0,1\}^*$, in which $f = (f_1, f_2)$. The input to the functionality f are pairs $x, y \in \{0,1\}^n$ and the output is $f = \big(f_1(x,y), f_2(x,y)\big)$. The first party, with input x, wants to obtain $f_1(x,y)$ and the second party, with input y, wants to obtain $f_2(x,y)$. In the case of the functionality just described, thus, x and y respectively represent private inputs. Protocols, denoted π, are used to calculate functionalities.

2.3.2 Security Model

Throughout we restrict our attention to the semi-honest model of adversaries [3]. 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 some protocol implements some functionality under the semi-honest model securely, proof via the simulation paradigm is used [19]. *Section 2.5.2* includes the construction of a simulator in order to establish the security of the protocol that we propose for oblivious Moore machine evaluation.

2.3.3 Combinatorial Filters

Following the formulation in [23], 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 $G = (V, E, l: E \to Y, v_0)$ 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 which can be computed on I-states will be encoded as *colors*.

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

The output of filter $F = (G, c)$ 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 G, and each $l\big((v_{i-1}, v_i)\big) = y_i$ for $i \in \{1, \dots, m\}$. Abusing notation slightly, we will write the filter as if it were a function, so the output is $F[y_1 y_2 \dots y_m]$.

2.4 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, $D_1, D_2, \dots, D_n$, 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 $m \le n$ 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 2.2*, we have 3 event detectors D_1, D_2, D_3, each associated to a beam sensor. Here the overall event space is $Y = \{a, b, c, \epsilon\}$. Each event detector D_j will have a *local event space* $Y^j = \{y_i, \epsilon\}$, where $y_i \in Y$ represents an event gathered by D_j. We impose the constraint that each event is covered by at least one event detector, i.e., $Y = \bigcup_{j=1}^{n} Y^j$.

For event detector D_j, the *local event history*, denoted $\widetilde{h_j} = \big((t_0, x_0), (t_1, x_1), \dots, (t_{L-1}, x_{L-1})\big)$, 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

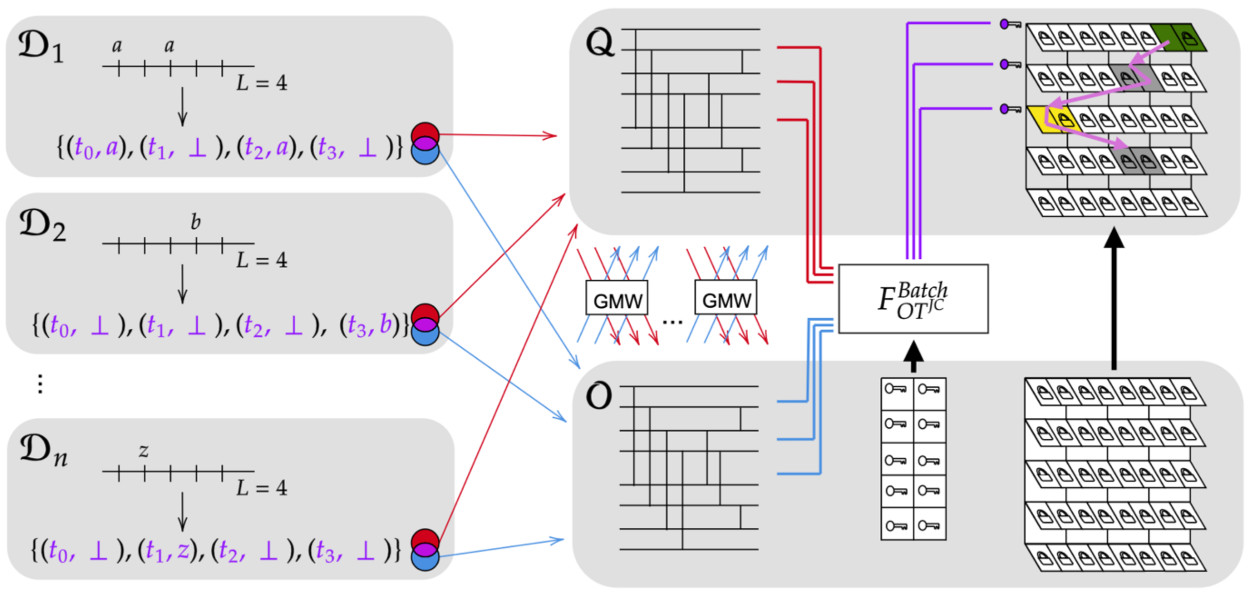

$\mathfrak{D}_1$
a a
$L = 4$
$\{(t_0, a), (t_1, \perp), (t_2, a), (t_3, \perp)\}$
$\mathfrak{D}_2$
b
$L = 4$
$\{(t_0, \perp), (t_1, \perp), (t_2, \perp), (t_3, b)\}$
$\mathfrak{D}_n$
z
$L = 4$
$\{(t_0, \perp), (t_1, z), (t_2, \perp), (t_3, \perp)\}$
Q
O
GMW
GMW
$F^{Batch}_{OT^{JC}}$

Figure 2.2: The overall architecture for MPC-based sensor fusion and combinatorial filter evaluation, shown via a schematic which emphasizes flow and sharing of information. On the far left, event detectors $\boldsymbol{D}_1, \boldsymbol{D}_2, \ldots, \boldsymbol{D}_n$*, queue readings for histories of length L. These are then transmitted as split shares to* $\boldsymbol{Q}$ *and* $\boldsymbol{O}$*. (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* $\boldsymbol{O}$ *knows the filter structure,* $\boldsymbol{Q}$ *(and* $\boldsymbol{Q}$ *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* $F^{Batch}_{OT^{JC}}$ *refers to batch oblivious transfer with joint choice [31].*

used in the algorithms we present. Note, that since $\epsilon \in Y^j$, one may always pad shorter histories to obtain ones of length L. The *global event history*, which we denote $\tilde{y}$, is the union of all events, as a sequence ordered by time.

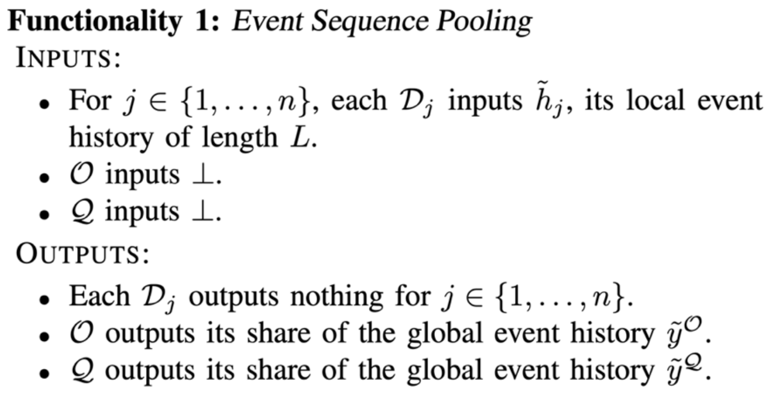
We employ two phases: the first involves pooling data from detectors to construct a subsequence of the global event history; the second processes that subsequence of events, evaluating the filter. Both phases must keep the data private.

2.4.1 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 then proceeds batch-by-batch.

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

These requirements are formalized as follows in *Figure 2.3*:



Functionality 1: *Event Sequence Pooling*

INPUTS:

- For $j \in \{1, \ldots, n\}$, 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 nothing for $j \in \{1, \ldots, n\}$.
- $\mathcal{O}$ outputs its share of the global event history $\tilde{y}^{\mathcal{O}}$.
- $\mathcal{Q}$ outputs its share of the global event history $\tilde{y}^{\mathcal{Q}}$.

Figure 2.3: Specification for Functionality 1: Event Sequence Pooling.

2.4.2 Oblivious Combinatorial Filter Evaluation

Once the event detector's streams have been pooled, the stream symbols need to be traced over a filter's labeled edges to generate the associated sequence of colors, i.e., to produce the filter's output. Since the symbols are split into pairs of shares, this tracing must be done in some joint fashion. We build on the ideas in [31] [20] to proposed 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 peers ($\boldsymbol{Q}$) be the querier, while the other ($\boldsymbol{O}$) is the filter owner. The functionality is formalized next in *Figure 2.4*:

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 $\mathbf{F}[(\tilde{y}^{\mathcal{Q}} \oplus \tilde{y}^{\mathcal{O}})]$.
- $\mathcal{O}$ outputs $\perp$.

Figure 2.4: Specification for Functionality 2: Combinatorial Filter Evaluation.

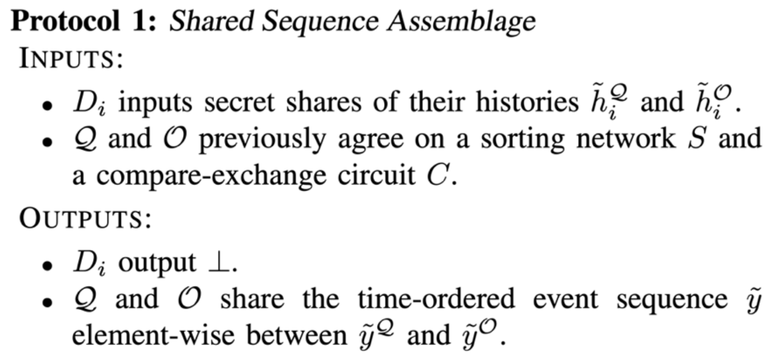
Next, we examine how to realize these two functionalities.

2.5 Methods

Figure 2.2 provides an encompassing diagram showing the $n + 2$ parties involved: event detectors $D_1, D_2, \dots, D_n$ and querier $\boldsymbol{Q}$ and filter owner $\boldsymbol{O}$. 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 further to the right: the red and blue lines are 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 turn to each of these in detail next.

2.5.1 Secure Event Sequence Pooling

To implement Functionality 1 (*Figure 2.3)*, we collect shares of each event history $\widetilde{h_\iota}$ into shares of an *agglomerated history* $\tilde{y}$. 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 D_j splits its history $\widetilde{h_j}$ into additive secret shares using a secure random number generator (RNG) and the XOR operation. Then D_j sends these two shares, denoted $\widetilde{h_j^O}$ and $\widetilde{h_j^Q}$, to the respective privacy peers. In *Figure 2.2*, the $\widetilde{h_j}$ are illustrated by sensors' individual purple sequences, while the transfer of $\widetilde{h_j^{\boldsymbol{O}}}$ and $\widetilde{h_j^{\boldsymbol{Q}}}$ are illustrated by the red and blue arrows towards O and Q. Keeping these shares separate, $\boldsymbol{O}$ and $\boldsymbol{Q}$ jointly sort the aggregate history using sorting networks and GMW Protocol. Finally, $\boldsymbol{O}$ and $\boldsymbol{Q}$ obtain shares of a sorted stream of symbols $\widetilde{y'}$.



Protocol 1: *Shared Sequence Assemblage*

INPUTS:

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

OUTPUTS:

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

Figure 2.5: Specification for Protocol 1: Shared Sequence Assemblage.

The following subsections provide further explanation for the components of the method outlined in *Figure 2.5*.

2.5.1.1 Padding Inputs

Padding event detectors' inputs with ϵ symbols, as previously mentioned, ensures $|\widetilde{h_\iota}| = L, \forall i \in \{1, \dots, n\}$. This is important so that information - such as the frequency at which D_i detects events - is not revealed.

2.5.1.2 Creating Secret Shares from Event Detectors

An element of an event history $\widetilde{h_\iota}[k]$ 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 $\mathrm{l} = \tau + \lceil \log |Y| \rceil$. Each detector D_i for $i \in \{1, \dots, n\}$ splits its history $\widetilde{h_\iota}$ into shares by generating random values $r \leftarrow_R \{0,1\}^{\mathrm{l}}$ for each element in its history. Then $\widetilde{h_j^{\boldsymbol{O}}}$ is the resulting sequence of random values. Let each element $\widetilde{h_j^{\boldsymbol{Q}}}[k] = \widetilde{h_\iota^{O}}[k] \oplus \widetilde{h_\iota}[k]$. Some (arbitrary) predefined ordering of detectors is known by both $\boldsymbol{O}$ and $\boldsymbol{Q}$, which collect the sequences to maintain pre-sorted subsequences, as is formalized next.

2.5.1.3 Sorting

$\boldsymbol{Q}$ and $\boldsymbol{O}$ will have concatenated all received histories into a shared agglomerated history, and they will proceed to sort the agglomerated history in increasing order by time. Jónsson, Kreitz, and Uddin [15], address sorting in the MPC context using sorting networks, and we use their work directly with only minor modifications. A sorting network is a structure which 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 agglomerated 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 [24], we use them throughout. Hence, we opt to have $\boldsymbol{O}$ and $\boldsymbol{Q}$ evaluate the compare-exchange circuit jointly via GMW. Finally, $\boldsymbol{O}$ and $\boldsymbol{Q}$ will strip the agglomerated history of its time keys, resulting in $\tilde{y}$, 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. Both correctness and security flow from Jónsson et al. [15] and the fact that GMW Protocol is an MPC primitive. The other modifications introduced are entirely peripheral to security.

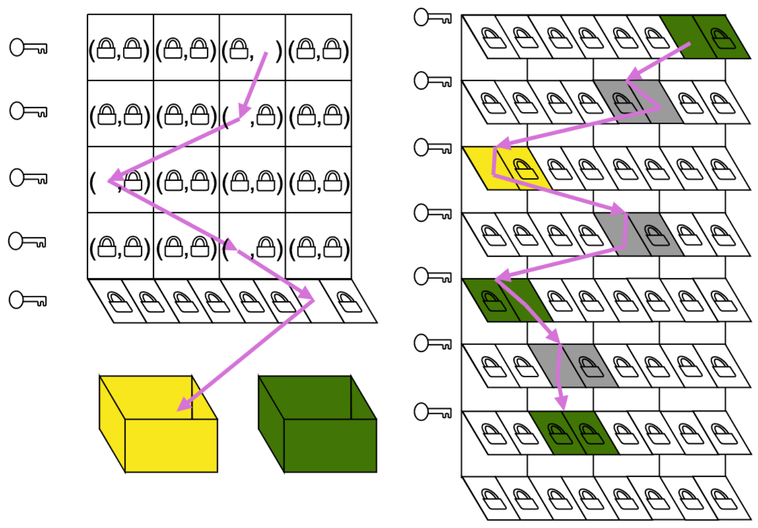


Figure 2.7: Oblivious evaluation of DFAs vs. Moore machines.

2.5.2 Oblivious Combinatorial Filter Evaluation

In this section, we build on prior results for Oblivious DFA evaluation [20] [31]. 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.

We define a binary Moore machine as $M = (Q, \{0,1\}, \Delta, q_0, C, c)$ composed of a finite set of states Q, the symbols 1 and 0, a transition function $\Delta: Q \times \Sigma \to Q$, 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 F, states and corresponding colorings for M are given directly from the filter. Next, 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 $q_j \in Q/V$ map to this color $c: q\_j \to grue\}$. Note that in our construction, ϵ will result in a self-loop. An example of a translated filter is shown in *Figure 2.7*. The Moore machine M

constructed in this way from a filter F has the property that for all $F[\tilde{y}] = c_0 c_1 \ldots c_m$ and $M[\widetilde{b_y}] = C_0 C_1 \ldots C_M$, where $\widetilde{b_y}$ is the binary representation of $\tilde{y}$ with $\lceil \log |Y| \rceil$-bits per symbol, then $c_0 = C_0$, $c_1 = C_{\lceil \log |Y| \rceil}$, …, $c_k = C_{k \cdot \lceil \log |Y| \rceil}$, …, $c_m = C_{m \cdot \lceil \log |Y| \rceil} = C_M$.

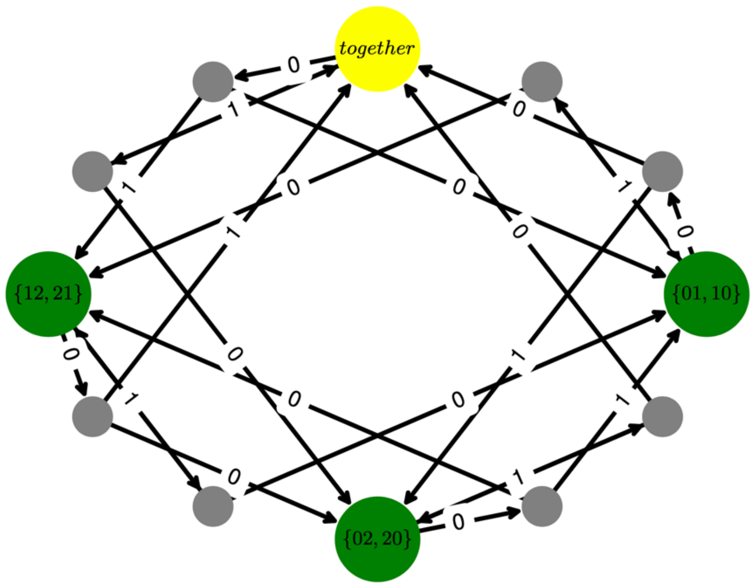


Figure 2.7: The combinatorial filter shown in Figure 1.1 [right], but converted to use a binary alphabet. A new output (shown in grue) is introduced to indicate that the Moore machine is processing a syncopated step.

Mohassel et al. [20] introduce the *DFA Matrix* (see *Figure 2.6 [left]*) as a building block to oblivious DFA evaluation. A binary DFA Matrix M represents states $s_i = (j, k)$ as pairs of transitions $s_i \rightarrow_0 s_j$ and $s_i \rightarrow_1 s_k$, where $s_i \in Q$. Evaluating a binary DFA Matrix on a bit sequence b involves choosing the $b[i]$-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 × |Q| is needed to process sequences of length n on a DFA with states Q. In our case, we wish to have $\boldsymbol{Q}$ 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 $M(b)$ at each state rather than an producing only an ‘accept’/‘reject’ output in the last row (see *Figure 2.6 [right]*).

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 $\boldsymbol{O}$ and $\boldsymbol{Q}$ is largely based upon [31], 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 the same size, $n \times |Q|$, of random values $RM[i,j] \leftarrow_R \{0,1\}^{\kappa}$, where κ is the security parameter. Then, for each row i, create a pair of *garbled keys* $k_0^i, k_1^i \leftarrow_R \{0,1\}^{k+s}$, where $k = \kappa + \log|Q|$ and s is the statistical security parameter.

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:

$$\begin{gathered} GM[i,j]_0 = \left(PM[i,j]_0 \| RM[i+1, PM[i,j]_0]_0 \| 0^s\right) \oplus k_0^i \\ GM[i,j]_1 = \left(PM[i,j]_1 \| RM[i+1, PM[i,j]_1]_1 \| 0^s\right) \oplus k_1^i \\ \mathrm{GM[i,j]}_2 = PM[i,j]_2 \end{gathered} \tag{2.1}$$

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 $PM[i,j]$ points to. Garbled keys are used to mask the values of these pair items, and s zeroes are concatenated together at the end of the bit string, which the evaluator can use to determine whether or not they are decrypting the correct pair of items.

For each triple in each row, flip a coin and swap the first two items if the coin is heads. 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:

$$\begin{aligned} \mathrm{GM[i,j]_0} &= \mathrm{GM[i,j]_0} \oplus \mathrm{p_0^{i,j}} \\ \mathrm{GM[i,j]_1} &= \mathrm{GM[i,j]_1} \oplus \mathrm{p_1^{i,j}} \\ \mathrm{GM[i,j]_2} &= \mathrm{GM[i,j]_2} \oplus \mathrm{p_2^{i,j}} \end{aligned} \tag{2.2}$$

Once the construction of the garbled matrix is complete, $\boldsymbol{O}$ sends $\boldsymbol{Q}$ the initial state index q_0, the pseudorandom generators G_1 and G_2 and the initial random value $r = RM[1, q_0]$. $\boldsymbol{Q}$ proceeds by retrieving stream ciphers $p_0^{1,q_0}, p_1^{1,q_0} \leftarrow G_1(r), p_2^{1,q_0} \leftarrow G_2(r)$. $\boldsymbol{Q}$ also needs garbled keys $k_{\tilde{y}}^i$ for each bit in the shared input string $\tilde{y}$. This is accomplished by *batch oblivious transfer with joint choice* (F_{OTJC}^{Batch}). This MPC protocol, secure under the semi-honest model [31], entails $\boldsymbol{O}$ transferring its input string of joint choice $k_{\tilde{y}}$ to $\boldsymbol{Q}$ via both parties' shares of the choice $\widetilde{y^O}$ and $\widetilde{y^Q}$.

Given garbled keys, $\boldsymbol{Q}$ can evaluate the matrix row-by-row. At the first row, decrypt $GM[1, q_0]_2$ to obtain the initial color. Next, for each subsequent row i and state j, $\boldsymbol{Q}$ decrypts $GM[i,j]_0 = p_0^{i,j} \oplus k_i \oplus GM[i,j]_0$ and $\mathrm{GM[i,j]_1} = p_1^{i,j} \oplus k_i \oplus GM[i,j]_1$. Then, $\boldsymbol{Q}$ can observe which element of the pair provides a valid decryption. $\boldsymbol{Q}$ can repeat this procedure, finding the next stream ciphers p_0, p_1, p_2 with every new state visited, and outputting the color $GM[i,j]_2 \oplus p_2$. This is formalized via Protocol 2, given by *Figure 2.8*.

Protocol 2: *Oblivious Moore Machine Evaluation with Joint Input*

COMMON INPUTS:

- Security parameter κ, statistical security parameter s, the number of states $|Q|$, the number of colors $|C|$, the pseudorandom number generators $G_1 : \{0,1\}^\kappa \to \{0,1\}^{k+s}$, where $k = \kappa + \log|Q|$ and $G_2 : \{0,1\}^\kappa \to \{0,1\}^{\lceil \log|C| \rceil}$.

PRIVATE INPUTS:

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

OUTPUTS:

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

Figure 2.8: Moore Machine Evaluation with Joint Input.

We will next prove the correctness and security of the proposed protocol. For a protocol to follow the semi-honest model, it must produce the correct output, and guarantee nothing can be learned during execution other than what can be obtained from inputs and outputs. The simulation proof technique [19] prescribes an analysis of the inputs and outputs of a party by constructing a polynomial-time simulator of its real view, then 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.

Theorem 2: Protocol 2 (*Figure 2.8*) is correct and secure in the semi-honest model.

Correctness: Garbled Moore machine evaluation on joint input traces the same path through states as a corresponding garbled DFA evaluation would, since they use the same mechanisms for encoding and traversing a matrix. We have shown how our color output extension reports the correct color at a given state; thus, correctness follows from Oblivious DFA Evaluation on Joint Input.

Security: $\boldsymbol{O}$ only interacts with $\boldsymbol{Q}$ during batch OT garbled key exchange, which Zhao et al., [31] have shown to be an MPC protocol. Therefore, we will examine $\boldsymbol{Q}$'s view of execution by constructing simulator $S_{\boldsymbol{Q}}$: The simulator is given all inputs and outputs of $\boldsymbol{Q}$, the same inputs and outputs of oblivious DFA evaluation, with the exception of the output color stream $M(\tilde{y})$, replacing a solitary 'accept'/'reject' output with a stream of colors. $S_{\boldsymbol{Q}}$ must fabricate messages received in real execution, namely the garbled keys K and garbled matrix GM_M received from $\boldsymbol{O}$. $S_{\boldsymbol{Q}}$ will create a binary Moore machine matrix M' of the same size (the size can be computed from the common input) as the real matrix during execution, but with random states and transitions. Each element in row i of GM'_M will be colored with $M(\tilde{y})[i]$. $S_{\boldsymbol{Q}}$ will also obtain a corresponding set of garbled keys, K', which is indistinguishable from K (trivially, as both are sets of random numbers). We have shown that a garbled matrix is computationally indistinguishable from randomness everywhere except along the evaluation path. The evaluation of GM'_M given garbled keys K' results in $M[\tilde{y}]$, just as GM_M does with keys K, so GM'_M and GM_M are indistinguishable.

All messages during execution can be simulated by S_Q, so the protocol is secure.

We emphasize an important difference between our sketch and the proof by Zhao et al. [31] for oblivious DFA evaluation. They take a random approach: the simulator generates a random garbled matrix; evaluates it; if the output is the same as the output in 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.

2.6 Experimental Results

We have implemented a proof-of-concept version of the protocols just described in Python, in a simple multiprocessing environment, with $\boldsymbol{Q}$ and $\boldsymbol{O}$ running as two processes on the same machine.

2.6.1 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 $\boldsymbol{Q}$ and $\boldsymbol{O}$. The primary work is a third multi-process program where $\boldsymbol{Q}$ and $\boldsymbol{O}$ 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 2.9*.

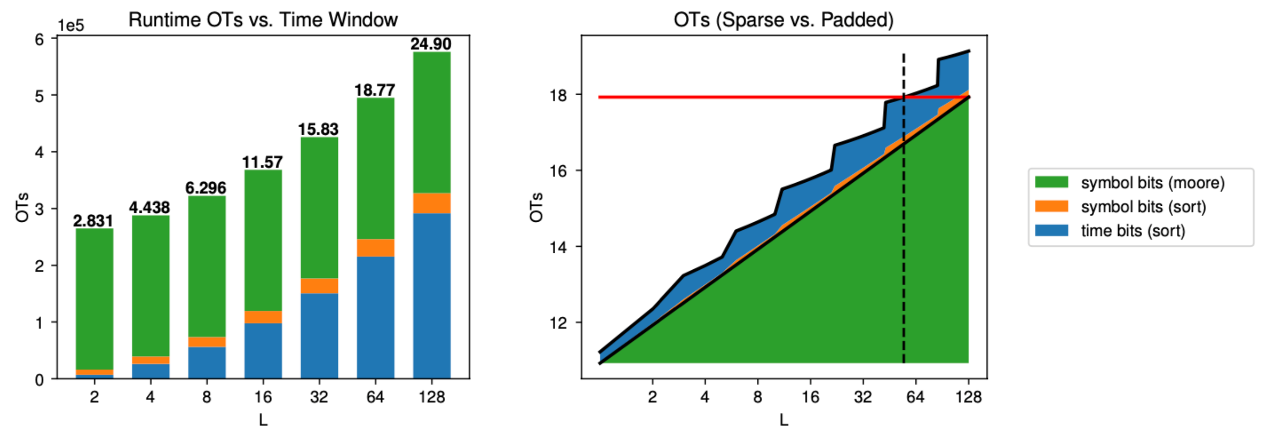


Figure 2.9: [left] A breakdown of pre-computed 1-bit random OTs consumed during run time, as the local history length L is increased, and $2^\tau = L$. [right] The total number of OTs when sorting local histories where we vary the sparsity of events within the L-sized window. The red line allows for comparison to when we take single element windows and forgo sorting altogether.

2.6.2 Results

We set security parameter $k = 256$ and statistical security parameter $s = 64$. 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. 128GB memory. *Figure 2.9 [left]* gives a break-down 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 the 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 low, a unary construction of Functionality 1 may be warranted. A unary variant would have a time window $L = 1$, and the sorting aspect of Functionality 1 becomes degenerate.

But the story is not so clean cut. 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 2.9 [right]* shows that before a certain point ($L \approx 54$), when the input symbols are sparse enough, it is useful to represent these and sort them and only evaluate the fraction that are known to be non-ϵ values (e.g., owing to some a priori model of sparsity). Once L exceeds 54, 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 2.9 [right]* is the same treatment as *Figure 2.9 [left]*.)

2.7 Conclusion

This paper has presented 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 $\boldsymbol{Q}$ and $\boldsymbol{O}$, but we do not take advantage of many of the optimizations to GMW Protocol that are available, including extending oblivious transfers [13], parallelizing oblivious transfers in batch, using multiplication triples [3] and performing load balancing [24]. Moore machine evaluation can be further improved by implementing suggestions from [31]. Our implementation, though rudimentary, lays the foundation for future work in practical applications of oblivious sensor fusion.

2.8 References

[1] A. Alexandru and G. Pappas, in *Privacy in Dynamical Systems*, Springer, 2020, pp. 179-207.

[2] K. Batcher, "Sorting Networks and their applications," in *Proceed. AFIPS Spring Joint Comput. Conf.*, 1968.

[3] D. Beaver, "Efficient Multiparty Protocols Using Circuit Randomization," 1991.

[4] L. Bobadilla, O. Sanchez, J. Czarnowski and S. M. LaValle, "Minimalist Multiple Target Tracking Using Directional Sensor Beams," in *ConferenceProceedings IEEE International Conference on Intelligent Robots and Systems*, 2011.

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

[6] H. Durrant-Whyte and T. C. Henderson, "Multisensor data fusion," 2016, pp. 867-896.

[7] C. Dwork and A. Roth, "The Algorithmic Foundations of Differential Privacy," *Found. Trends Theor. Comput. Sci.,* vol. 9, no. 3–4, pp. 211-407, aug 2014.

[8] D. Evans, V. Kolesnikov and M. Rosulek, "A Pragmatic Introduction to Secure Multi-Party Computation," Now Publishers, 2021.

[9] J. Garay, B. Schoenmakers and J. Villegas, "Practical and Secure Solutions for Integer Comparison," vol. 4450, Springer, 2007, pp. 330-342.

[10] O. Goldreich, "Basic applications," in *Foundations of cryptography. II*, 2004.

[11] O. Goldreich, S. Micali and A. Wigderson, "How to play any mental game," in *ConferenceProceedings of the Nineteenth Annual ACM Symposium on Theory of*

Computing, 1987.

[12] C. Hazay and Y. Lindell, Efficient Secure Two-Party Protocols: Techniques and Constructions, Springer, 2010.

[13] Y. Ishai, J. Kilian, K. Nissim and E. Petrank, "Extending Oblivious Transfers Efficiently," in *Advances in Cryptology - CRYPTO 2003*, 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,* vol. 41, pp. 135-146, 2016.

[15] K. Jónsson, G. Kreitz and M. Uddin, "Secure Multi-Party Sorting and Applications.," *IACR Cryptology ePrint Archive,* vol. 2011, p. 122, 01 2011.

[16] S. M. LaValle, "Sensing and filtering: A fresh perspective based on preimages and information spaces," *Foundations and Trends in Robotics,* vol. 1, no. 4, pp. 253-372, 2012.

[17] J. Le Ny and G. J. Pappas, "Differentially Private Filtering," *IEEE Transactions on Automatic Control,* vol. 59, no. 2, pp. 341-354, 2014.

[18] L. Li, A. Bayuelo, L. Bobadilla, T. Alam and D. A. Shell, "Coordinated multi-robot planning while preserving individual privacy," in *2019 International Conference on Robotics and Automation (ICRA)*, 2019.

[19] Y. Lindell, "How To Simulate It – A Tutorial on the Simulation Proof Technique," in *Tutorials on the Foundations of Cryptography*, New York City, Springer, Cham, 2017.

[20] 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*, 2012.

[21] J. M. O'Kane, "On the value of ignorance: Balancing tracking and privacy using a two-bit

sensor," in *WAFR*, Berlin, Heidelberg, 2008.

[22] J. M. O'Kane and D. A. Shell, "Automatic design of discreet discrete filters," in *ConferenceProceedings of the IEEE International Conference on Robotics and Automation*, 2015.

[23] J. M. O'Kane and D. A. Shell, "Concise planning and filtering: hardness and algorithms," *IEEE Transactions on Automation Science and Engineering,* vol. 14, no. 4, pp. 1666-1681, 2017.

[24] T. Schneider and M. Zohner, "GMW vs. Yao? Efficient Secure Two-Party Computation with Low Depth Circuits," in *Financial Cryptography and Data Security*, Berlin, Heidelberg, 2013.

[25] 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),* vol. 10, no. 3, p. 47, 2014.

[26] Y.-C. Wu, V. Raman, B. C. Rawlings, S. Lafortune and S. A. Seshia, "Synthesis of obfuscation policies to ensure privacy and utility," *Journal of Automated Reasoning,* vol. 60, no. 1, pp. 107-131, 2018.

[27] A. C. Yao. "Protocols for secure computations." in *ConferenceProceedings of the23rd Annual Symposium on Foundations of Computer Science*, pp. 160–164, 1982.

[28] A. C.-C. Yao, "How to generate and exchange secrets," in *ConferenceProceedings of the 27th Annual Symposium on Foundations of Computer Science*, 1986.

[29] Y. Zhang and D. A. Shell, "Complete characterization of a class of privacy-preserving tracking problems," *The International Journal of Robotics Research,* 2018.

[30] Z. Zhang, J. Wu, D. Yau, P. Cheng and J. Chen, "Secure kalman filter state estimation by partially homomorphic encryption," in *ConferenceProceedings of the 9th ACM/IEEE International Conference on Cyber-Physical Systems*, 2018.

[31] C. Zhao, S. Zhao, B. Zhang, S. Jing, Z. Chen and M. Zhao, "Oblivious DFA evaluation on joint input and its applications," *Information Sciences,* vol. 528, pp. 168-180, 2020.

[32] D. A. Shell, W. E. Curran, L. Bobadilla and C. Rojas, "Oblivious sensor fusion via secure multi-party combinatorial filter evaluation," in *IEEE Conference on Decision and Control*, Austin, 2021.

3. CONCLUSION

The research team's primary work established a novel protocol for securely evaluating a combinatorial filter amongst many parties and implemented a proof-of-concept program to demonstrate its feasibility. Some important detail is left out of the above paper to maintain brevity and to not repeat what other researchers have already concluded. In the area of cryptography, however, correctness is vital, so more complete detail will be provided below to help establish the basis for claims of validity. First, it is worth exploring why we chose to use Mohassel et al.'s and Zhao et al.'s work. Second, a more concrete discussion of Moore machine evaluation is warranted. Next, we will provide more detail on the implementation. Finally, we discuss output leakage in more detail.

3.1 Footnotes on the Paper

3.1.1 Why Oblivious DFA evaluation with joint choice?

The notion of joint choice is quite convenient to merge additive secret shares of a choice without actually exchanging the plaintext. Zhao et al.'s most interesting contribution to DFA evaluation, to us, is this idea oblivious transfer with joint choice. They create a new OT protocol with joint choice, which uses a partially-homomorphic encryption scheme to mix the choices.

One thing we could do is use DFA evaluation (*Figure 2.6 [left])* to get a colored output after processing all inputs. However, the stream of information which one obtains from a filter is desirable in many cases.

3.1.2 Alternate solutions to secure combinatorial filter evaluation

At the start of this document, one of the first things mentioned is Yao's Garbled Circuit protocol. This is an extensively studied protocol, and many optimizations make it one of the most feasible 2-party MPC general-purpose frameworks. One could also accomplish Functionality 2 (*Figure 2.4*) with a garbled circuit protocol. Mohassel et al. mention this possibility in their formulation for oblivious DFA evaluation, and in fact they can rephrase their protocol as a tweak on Yao's garbled circuits. They use a custom DFA protocol for a few reasons. First, one would need a special circuit compiler to transform a DFA to a garbled circuit. Second, there are some structural aspects of the DFA which can be taken advantage of with a specialized protocol, which a generalized approach may be able to capture, but with some difficulty. In other words, since their construction of oblivious DFA evaluation is simple and features excellent performance, it is better to use this instead of a more complex formulation of a general garbled circuit. Third, Mohassel et al.'s construction creates a client-server relationship between the evaluator and DFA owner.

The first two reasons still apply to Moore machine evaluation. In fact, the Moore machine does not suit itself to a garbled circuit as well as a DFA might. A DFA might be used to perform pattern-matching on an input string, and check if it belongs to a certain class of patterns. In a Moore machine, the output at each round of evaluation must be reported, which would require a more substantial modification to a garbled circuit method.

As for the third reason, Zhao et al.'s implementation takes us further away from the notion of a client-server relationship, and we take it even further away, as we prefer to think of the privacy peers as equal participants, while the event detectors are more considered to be "clients."

A homomorphic encryption approach may also be considered, although these schemes generally involve more asymmetric operations than the best garbled circuit implementations.

3.1.3 Garbled Matrix Construction and Proofs

In Section 2, we explain the construction of a garbled matrix to evaluate a Moore machine, rooted in Mohassel et al.'s construction for oblivious DFA evaluation, and Zhao et al.'s extension for DFA evaluation on joint input.

First, let's visit the intuition on the Oblivious Moore machine evaluation protocol *(Figure 2.6 [right])*. The stream ciphers $p_0^{i,j}$, $p_1^{i,j}$ and $p_2^{i,j}$ provide the top layer of encryption for the garbled matrix. Without the random seed obtained from the previous state visited, one cannot decrypt any state except those on the evaluation path (pointed at from the previous state). The two garbled keys for each row k_0^i and k_1^i form the next layer of encryption. There are two garbled keys per row, one used to encrypt each '0' transition, and one to encrypt each '1' transition. Since garbled keys are random numbers, it is hard to guess one, so, the evaluator $\boldsymbol{Q}$ needs to obtain the proper one in order to proceed. Therefore, the garbled keys protect the transitions away from each state. The garbled keys of joint input $k_{\tilde{y}}^i$ are obtained via F_{OTJC}^{Batch}. The next level of obfuscation is accomplished by permuting the transitions and permuting the rows. A random permutation of the transitions protects the joint choice bit from being learned by $\boldsymbol{Q}$ during evaluation. A random permutation of the rows of states protects the knowledge during evaluation of if a change in state occurred across a transition, and what the current state is.

Both stream ciphers and garbled keys are related to security parameters κ and s. The security parameter κ is the length of the seed used in the pseudorandom generator to produce stream ciphers, which relates to the security of the protocol, as it must be sufficiently large, in practice, to protect against brute force attacks. (Computational indistinguishability provides protection in an asymptotic sense, but the value κ is public information amongst participants, and if κ is a small enough value, a corrupted party may have success in trying all possible seed values.) The statistical security parameter s is how many zeros are appended to the end of each transition to prove that it was the right transition. This is proportional to the correctness of the protocol. If given too small of an s, the protocol could often be terminated with failure. Note that the statistical parameter can be avoided altogether by using a more efficient 'point-and-permute' construction, given by Zhao et al., but we will not go into detail on this.

INPUTS:
Moore machine $M = (Q, \{0,1\}, \Delta, q_0, C, c)$, input length n, pseudorandom generators $G_1 : \{0,1\}^\kappa \to \{0,1\}^{k+s}$ and $G_2 : \{0,1\}^\kappa \to \{0,1\}^{\lceil log|C| \rceil}$, where $k = \kappa + \lceil log|C| \rceil$.

GARBLED MATRIX GENERATION:

1. Create a Moore machine matrix M of size $n \times |Q|$, where each state is defined as a tuple of transitions and a color:
   $M[i,j] = (\Delta(i,0), \Delta(i,1), c(i))$
2. Create a permutation matrix PER of size $(n+1) \times |Q|$, where each row is a scrambled list of indices $\{1, \ldots |Q|\}$, permuted uniformly at random.
3. Create a permuted Moore machine matrix PM of size $n \times |Q|$.
   $PM[i, PER[i,j]]_0 = PER[i+1, M[i,j]_0]$
   $PM[i, PER[i,j]]_1 = PER[i+1, M[i,j]_1]$
   $PM[i, PER[i,j]]_2 = M[i,j]_2$
4. Create a random matrix RM of size $n \times |Q|$, with each element chosen uniformly at random (some private source of randomness should be used, not a publicly shared pseudorandom generator).
   $RM[i,j] \leftarrow_R \{0,1\}^\kappa$
5. Create a garbled matrix GM of size $n \times |Q|$.
   $GM[i,j]_0 = (PM[i,j]_0 || RM[i+1, PM[i,j]_0]_0 || 0^s) k_0^i$
   $GM[i,j]_1 = (PM[i,j]_1 || RM[i+1, PM[i,j]_1]_1 || 0^s) k_1^i$
   $GM[i,j]_2 = PM[i,j]_2$
   $b \leftarrow_R \{0,1\}$. *If* $b = 1$, *then swap* $GM[i,j]_0$ *and* $GM[i,j]_1$.
   $p_0^{i,j} \leftarrow G_1(RM[i,j])$
   $p_1^{i,j} \leftarrow G_1(RM[i,j])$
   $p_2^{i,j} \leftarrow G_2(RM[i,j])$
   $GM[i,j]_0 = GM[i,j]_0 \oplus p_0^{i,j}$
   $GM[i,j]_1 = GM[i,j]_1 \oplus p_1^{i,j}$
   $GM[i,j]_2 = GM[i,j]_2 \oplus p_2^{i,j}$

Figure 3.1: Garbled binary Moore machine matrix construction summary

We will go in depth now in a full proof under the semi-honest model for the security of oblivious Moore machine evaluation, given a refresher on construction of a garbled binary Moore machine matrix (*Figure 3.1*).

3.1.4 Detailed Security Proof for Protocol 2 (Figure 2.8)

3.1.4.1 Correctness

First, $\boldsymbol{Q}$ uses G_1 and G_2 to generate three stream ciphers using the initial seed.

$$\begin{aligned} p_0^{1,q_0} &\leftarrow G_1(RM[1,q]) \\ p_1^{1,q_0} &\leftarrow G_1(RM[1,q]) \\ p_2^{1,q_0} &\leftarrow G_2(RM[1,q]) \end{aligned} \tag{3.1}$$

Next, $\boldsymbol{Q}$ decrypts the first items.

$$\begin{aligned} x_0 &= \mathrm{GM}[1,q_0]_0 \oplus p_0^{1,q_0} \oplus k^1 \\ x_1 &= \mathrm{GM}[1,q_0]_1 \oplus p_1^{1,q_0} \oplus k^1 \\ \mathrm{c} &= \mathrm{GM}[1,q_0]_2 \oplus p_2^{1,q_0} \end{aligned} \tag{3.2}$$

$\boldsymbol{Q}$ then checks x_0 and x_1 for s trailing zeros. The probability that the *correct* element x_c has s trailing zeros is 1, as $x_c = (PM[1,q]_c||RM[2,PM[1,q]_c]_c||0^s) \oplus k_c^1 \oplus k^1$, where $k^1 = k_c^1$. The element $x_{1-c} = (PM[1,q]_{1-c}||RM[2,PM[1,q]_{1-c}]_{1-c}||0^s) \oplus k_{1-c}^1 \oplus k^1$ is computationally indistinguishable from random because $k_{1-c}^1 \oplus k^1$ is simply the XOR of two unrelated random numbers.

Then, the probability that x_{1-i} has s trailing zeros is the same as the probability of choosing s bits uniformly at random. Therefore, the probability that any transition selection is correct is $1 - 2^{-s}, s > 0$. Certainly, the correctness of the protocol is contingent upon all transitions being correct, even if a particular Moore machine produces the correct output along incorrect transitions. So, the probability of correctness of the protocol is $1 - |\tilde{y}|2^{-s}, s > 0$. We rely on the assumption that $|\tilde{y}| \ll 2^s$.

3.1.4.2 Corrupted $\boldsymbol{O}$

Messages received during Protocol 2 are only those needed to evaluate $F_{OT^{JC}}^{Batch}$. $\boldsymbol{O}$ receives no output from the protocol, and a garbled matrix can be constructed honestly to complete the view of $\boldsymbol{O}$ during execution. Therefore, we rely on the correctness and security of $F_{OT^{JC}}^{Batch}$. The specification and security proof of $F_{OT^{JC}}^{Batch}$ is in the following subsection.

3.1.4.3 Corrupted $\boldsymbol{Q}$

We construct a simulator $S_{\boldsymbol{Q}}$ given common inputs, $\boldsymbol{Q}$'s inputs: $\widetilde{y^Q}$ and outputs: $M(\tilde{y})$. The simulator will provide a view indistinguishable from $\boldsymbol{Q}$'s real view using only the inputs and outputs available to it. In other words, we will show:

$$\{S_{\mathcal{Q}}(\kappa, s, |Q|, G_1, G_2, |\tilde{y}|, \tilde{y}^{\mathcal{Q}}, M(\tilde{y}))\} \overset{c}{\equiv} \{view_{\mathcal{Q}}^{\pi}(\kappa, s, |Q|, G_1, G_2, |\tilde{y}|, \tilde{y}^{\mathcal{Q}}, M(\tilde{y}))\} \quad (3.3)$$

The view of real execution can be written:

$$\{view_{\mathcal{Q}}^{\pi}(\kappa, s, |Q|, G_1, G_2, |\tilde{y}|, \tilde{y}^{\mathcal{Q}}, M(\tilde{y}))\} = (GM, K, q, RM[1, q]) \quad (3.4)$$

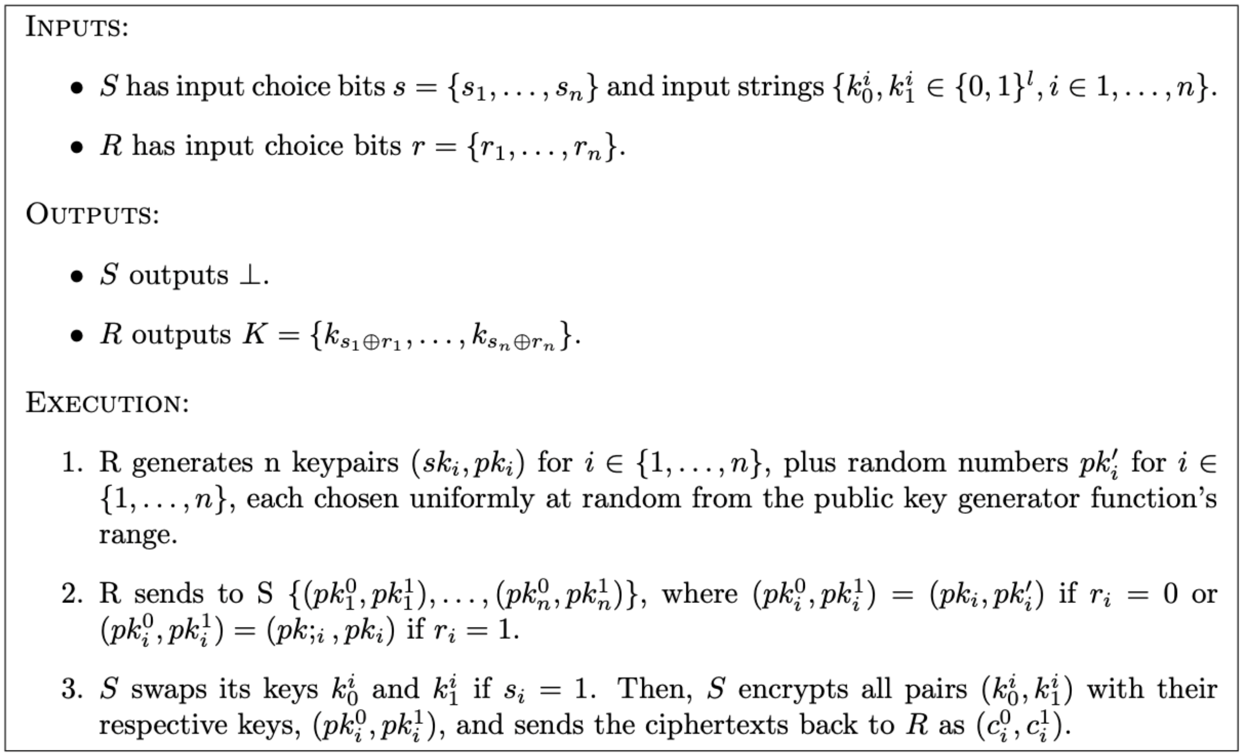
$S_{\boldsymbol{Q}}$ constructs GM', K', q', $RM[1,q]'$ as follows: Create a random binary Moore machine matrix M', where $M'[i,j] = (x, y, M(\tilde{y})[i])$, $x \leftarrow_R \{1, \dots, |Q|\}$, $y \leftarrow_R \{1, \dots, |Q|\}$, $q' \leftarrow_R \{1, \dots, |Q|\}$. GM' is constructed honestly as $\boldsymbol{O}$ would construct it, if given M'. Consequently, garbled keys K' and $RM[1,q]'$ are obtained.

First, K', $RM[1,q]'$ and q' are computationally indistinguishable from uniformly random even in a real execution, so a distinguisher will not find that these simulated versions are different from their real counterparts. GM' and GM are also filled with numbers which can be considered random, thanks to the pseudorandom generator assumption that the stream ciphers are computationally indistinguishable from random, in addition to the additive secret sharing scheme enabling us to make elements of GM' and GM also indistinguishable from random; therefore, elements of GM' and GM are indistinguishable from each other, with one exception. The exception, of course, is that elements of GM' and GM are not indistinguishable along their respective evaluation paths, where stream ciphers and garbled keys are known, so plaintext information may be obtained. Note that the actual states traversed are randomly permuted, as are transitions within states, so these are indistinguishable. It only remains to show that the outputs along the evaluation paths of GM' and GM are identical. This is trivially the case, as all states in GM' at row i are colored $M(\tilde{y})[i]$.

Note that in our implementation, we use garbled matrix evaluation to evaluate a Moore machine in 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 $\boldsymbol{O}$ are simply the communication involved in batch OT_{JC}^{Batch} (just a larger batch, which makes no difference to the proof). Messages inbound to $\boldsymbol{Q}$ 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. Therefore, *Equation 3.3* holds, and Protocol 2 is secure under the semi-honest model.

3.1.5 $F_{OT^{JC}}^{Batch}$ Protocol and Security

The following is a possible version of the semi-honest batch OT protocol described by Zhao. This proof is directly inspired by the basic OT proof in Chapter 3 of *A Pragmatic Introduction to Secure Multi-Party Computation*, cited above. The benefit of batching OTs is that fewer communication rounds are needed, and one can make use of *OT reduction*, which is a way to reduce the actual number of OTs needed, while still maintaining correctness and security. We will not describe this extension technique in detail; instead, we provide the general protocol specification (*Figure 3.2*) for batch OT with joint choice as if all OTs were done in practice.



INPUTS:

- S has input choice bits $s = \{s_1, \ldots, s_n\}$ and input strings $\{k_0^i, k_1^i \in \{0,1\}^l, i \in 1, \ldots, n\}$.
- R has input choice bits $r = \{r_1, \ldots, r_n\}$.

OUTPUTS:

- S outputs $\perp$.
- R outputs $K = \{k_{s_1 \oplus r_1}, \ldots, k_{s_n \oplus r_n}\}$.

EXECUTION:

1. R generates n keypairs (sk_i, pk_i) for $i \in \{1, \ldots, n\}$, plus random numbers pk_i' for $i \in \{1, \ldots, n\}$, each chosen uniformly at random from the public key generator function's range.
2. R sends to S $\{(pk_1^0, pk_1^1), \ldots, (pk_n^0, pk_n^1)\}$, where $(pk_i^0, pk_i^1) = (pk_i, pk_i')$ if $r_i = 0$ or $(pk_i^0, pk_i^1) = (pk;_i, pk_i)$ if $r_i = 1$.
3. S swaps its keys k_0^i and k_1^i if $s_i = 1$. Then, S encrypts all pairs (k_0^i, k_1^i) with their respective keys, (pk_i^0, pk_i^1), and sends the ciphertexts back to R as (c_i^0, c_i^1).

Figure 3.2: F_{OTJC}^{Batch} Protocol

3.1.5.1 Correctness

Correctness is trivially obtained by exhausting 4 possibilities for the values of s_i and r_i for any pair i.

3.1.5.2 Corrupted S

Construct a simulator to simulate the view of S during execution. S_S will sample two random numbers $(pk_i^{0}{}', pk_i^{1}{}')$ for $i \in \{1, \ldots, n\}$ from the range of the public key generator of the agreed-upon public-key cryptosystem, known *a priori*. The simulator gives these to S just as they were received from R. To a distinguisher, there is no difference between these randomly generated keys and those given by R during real execution, since it is assumed that the numbers are sampled uniformly at random from the public key space. Therefore, the view of S during execution is indistinguishable from the view which S_S produces.

3.1.5.3 Corrupted R

Construct a simulator S_R given the input and output of R. The simulator will “run” R as a subroutine, operating as usual, except that when it is to communicate with S, it instead will interact with its container S_R, which fabricates each $(c_i^0, c_i^1) = \left(Enc(0, pk_i'), Enc\left(k^i_{s_i \oplus r_i}, pk_i\right)\right)$ if $r_i = 0$ or $(c_i^0, c_i^1) = \left(Enc(0, pk_i'), Enc\left(k^i_{s_i \oplus r_i}, pk_i\right)\right)$ if $r_i = 1$.

3.1.6 MPC under the malicious model

Mohassel et al. achieve security against malicious clients, and privacy against malicious servers. There is no security against malicious servers because the server may not be using the appropriate DFA, or even a real DFA at all. This could be “patched up” by using the GMW Compiler, for example. This essentially means add a zero-knowledge proof on top of the protocol to assert that the protocol which was agreed upon is indeed the one being executed. This requires additional overhead, and Mohassel et al. are concerned with achieving excellent performance, so they do not implement this.

In order to make Moore machine evaluation secure against malicious adversaries, garbled key oblivious transfer would need to be secure under the malicious model. The reason why Zhao does not achieve security against malicious adversaries is that their protocols for oblivious transfer with joint choice are inherently semi-honest. Even if given a partially-homomorphic encryption scheme secure against malicious adversaries, the protocol for F_{OTJC}^{Batch} relies on both parties following the protocol in semi-honest fashion. Future research on Moore machine (or DFA) evaluation on joint input may entail constructing an OT with joint choice protocol which is secure under the malicious model.

3.1.7 Output Leakage

The work of this thesis operates mostly on the set of MPC assumptions, which care only for the security of a protocol, in that nothing is learned by a party other than its inputs and outputs. In designing protocols with end-to-end security, however, it is important to identify any weak links. Even when a protocol is fundamentally secure in the eyes of the MPC framework, we must look elsewhere for weaknesses. In particular, what might be learned from the output of a functionality, and on a related note, what might be learned from the output, *especially when given the input*?

There are multiple ways in which this train of thought impacts this research. In fact, the team has given the idea of output leakage some thought in designing the functionalities present in Section 2. For one, consider why the event detectors are specified as only providing inputs and obtaining no outputs. As a consequence of output opacity towards event detectors, there is room for potential discord, which is not present in many typical MPC protocols, which usually provide rewarding outputs to all participants. Indeed, it may not be realistic to expect detectors to be willing to participate, but we consider them to be compensated in some other way. If all event detectors also obtained the output stream of colors which $\boldsymbol{Q}$ obtains, what else might they be able to learn? It turns out that this is a hard question to answer with completeness.

Imagine there are two event detectors a and b, whose beam sensors segment a rectangular room into 3 rooms, the red room, the yellow room and the green room (*Figure 3.3*). In this scenario, protocols 1 and 2 are used as normal, not sharing outputs with a and b. $\boldsymbol{Q}$ receives a stream of outputs $\{\dots, Yellow, Red\}$. $\boldsymbol{Q}$ might wish to know about the aggregate history of inputs of a and b.

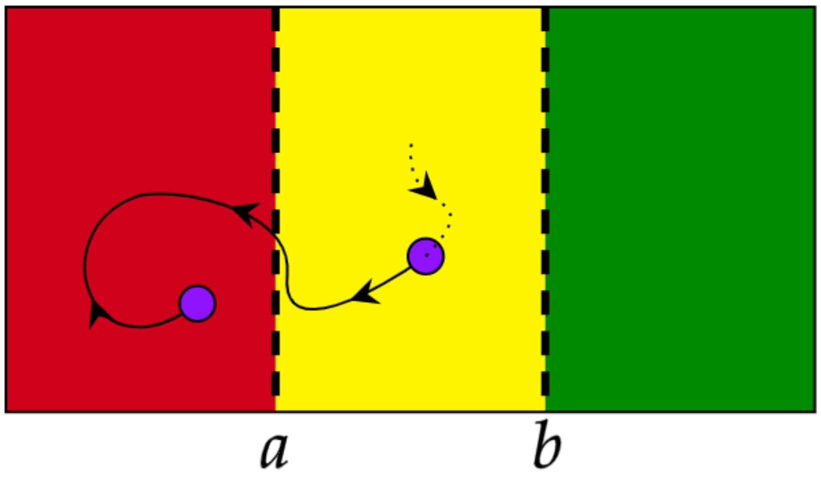


Figure 3.3: Dotted vertical lines represent event detectors, realized as beam sensors. Purple circles represent the position of the single mobile agent at two points in time.

If $\boldsymbol{Q}$ has no knowledge of the physical layout of event detectors in the environment, then perhaps this reveals nothing extra to $\boldsymbol{Q}$. However, if $\boldsymbol{Q}$ knows details about the physical layout, and which detectors are which, there actually is a breach to privacy, even if our secure protocols were used to evaluate the filter. $\boldsymbol{Q}$ sees that $Yellow$ and Red were observed one after the other, so $\boldsymbol{Q}$ knows that a was triggered most recently.

This example shows that one still must be careful to know when an MPC protocol alone is a full solution, and when it is not. In this case, the triviality of the question gives away information which individual detectors would have preferred to be private.

A direction of further research in regard to end-to-end security for combinatorial filter evaluation might be making some predetermination on what is allowed to be learned from the filter and what is not. An ambitious goal might be designing an algorithm which, given a combinatorial filter, restructures it to minimize possible leakage of information from its output, while retaining most of the original correctness.

3.2 Closing Statements

In conclusion, the research team assembled MPC protocols to securely evaluate combinatorial filters, then implemented these protocols from scratch. In the process, we learned about some of the nuance present in MPC, and we find that there may be future work ahead of us on processing streams of data using MPC protocols. People and their devices demand increasing amounts of privacy as the amount of data collected on the world, and consequently on people's personal lives, continues to increase. This provides a wide landing zone for MPC protocols to deliver utility to people who need it. Works like this thesis are advancing MPC closer to wider adoption, fulfilling this need.