PwoP: Intrusion-Tolerant and Privacy-Preserving Sensor Fusion

Chenglu Jin
University of Connecticut
chenglu.jin@uconn.edu

Marten van Dijk
University of Connecticut
marten.van_dijk@uconn.edu

Michael K. Reiter
UNC Chapel Hill
reiter@cs.unc.edu

Haibin Zhang
UMBC
hbzhang@umbc.edu

***Abstract*—We design and implement, PwoP, an efficient and scalable system for intrusion-tolerant and privacy-preserving multi-sensor fusion. PwoP develops and unifies techniques from dependable distributed systems and modern cryptography, and in contrast to prior works, can 1) provably defend against pollution attacks where some malicious sensors lie about their values to sway the final result, and 2) perform within the computation and bandwidth limitations of cyber-physical systems.**

PwoP is flexible and extensible, covering a variety of application scenarios. We demonstrate the practicality of our system using Raspberry Pi Zero W, and we show that PwoP is efficient in both failure-free and failure scenarios.

I. Introduction

Numerous modern cyber-physical systems (CPS), spanning industry, agriculture, military, and beyond, are increasingly relying on distributed data sources (hereinafter without loss of generality, sensors) to support critical decisions and actions. As depicted in Fig. 1, the integration of these sensor data are most often achieved with the help of a server (proxy, aggregator, or averager), which upon receiving a client request, gets sensor inputs from a set of sensors, integrates the sensor inputs, and returns the result to the client. The client may then inform the actuator what to do. Its application scenarios are almost everywhere, including sensor networks, smart metering, GPS devices and satellites, soldiers in battlefields, smart phones and the cloud, time-keeping mechanisms [55], and so on.

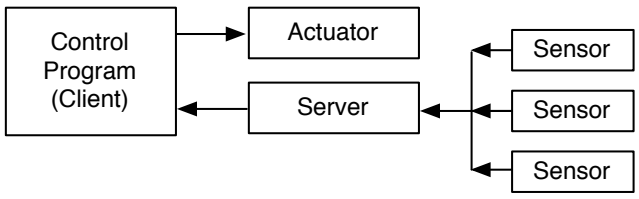

Fig. 1. Distributed sensor fusion architecture.

Privacy and integrity are widely regarded as primary concerns or even hurdles for many of these applications [69]. First, we need to protect the privacy of individual data sources. Ideally, clients should learn only the designated function of the sensor inputs but nothing more, and the server should learn nothing (about sensor inputs or the final result). Second, we require integrity, in the sense that the server should faithfully return to the client the correct, integrated result.

Pollution attacks. An equally important but notoriously difficult goal in multi-sensor fusion is to defend against *pollution attacks*, where some malicious sensors lie about their values to sway the final result. Specifically, motivated attackers can mount this kind of attack by either corrupting sensors and contributing malicious inputs, or maliciously altering environmental variables (say, by manipulating the environmental values for sensors without actually corrupting the sensors themselves).

At first glance, defending against pollution attacks seems to be at odds with attaining privacy. Indeed, to achieve the strongest privacy goal mentioned above, the server is not supposed to learn individual sensor inputs. Thus, in spite of the risk of pollution attacks, the vast majority of existing privacy-preserving systems treat defending against pollution attacks as out-of-scope.

Yet still, there are a handful of prior works attempting to *mitigate* the problem [21, 49]. Their approach is to ask the sensors to provide a cryptographic proof to show their inputs are in a prescribed range (or more generally satisfying some predicate), in the hope that a coalition of malicious sensors would not affect the final result by much. Take the average function for example. Suppose we have ten sensors, each of which can have an input selected from the range [1, 100]. Also assume that the "correct" value is around 20 and correct sensors will output a value around this. If three malicious sensors contribute 100 (which is in the range), they would introduce significant error into the final result. Moreover, for many applications, there are no prescribed limits on sensor inputs. To the best of our knowledge, all existing privacy-preserving aggregation and fusion schemes are vulnerable to pollution attacks to a significant degree.

Working in computation and bandwidth restricted CPS. Despite an impressive amount of work on secure sensor fusion or data aggregation protocols [16, 18, 21, 24, 27, 28, 38, 39, 48, 57, 67], to the best of our knowledge, none of them are implemented in real computation and bandwidth-restricted environments. They either do not provide any implementation, or their systems are run using commodity computers or virtual machines. These systems do not consider computation and bandwidth restricted environments specifically, and a new design with these factors considered is needed. Meanwhile, implementing a real system in these restricted environments involves integration of knowledge and expertise from different areas — hardware, software, and network communications.

Our approach. We design and implement, PwoP ("Privacy w/o Pollutions"), a privacy-preserving and fault-tolerant sensor fusion system that 1) defends against pollution attacks, 2) performs within the computation and bandwidth limitations of cyber-physical systems, 3) covers different application scenarios and different sensor input types, and 4) is efficient and scalable in both failure-free and failure scenarios.

To defend against pollution attacks, instead of relying on validity proofs, our strategy is to "tolerate," so that no matter what inputs malicious sensors provide, the fused value represents the correct physical value with good accuracy, still in a privacy-preserving manner.

Specifically, PwoP combines techniques from distributed systems and multi-party computation. At the core of PwoP are a series of practical sensor fusion algorithms that tolerate malicious inputs themselves [12, 19, 54, 55, 65]. For example, one of Marzullo's algorithms [54] ensures that the range produced by fusing ranges provided from sensors contains the actual value measured by the sensors if at most f out of $2f + 1$ sensors are malicious. We extend the framework of Yao's garbled circuits (GC) for two-party computation to the client-server-sensors setting for these fault-tolerant algorithms, leveraging their built-in fault-tolerance to efficiently achieve system liveness (i.e., guaranteed output delivery).

From the system perspective, we build a new and *general* compiler enabling the multi-sensor setting from TinyGarble [68]. We also make significant tailored optimizations, yielding a system that performs considerably better than direct application of GC compilers produce. Our system also tackles various other security, availability, and reliability concerns.

PwoP covers a variety of sensor input types, including intervals, rectangles, and d-dimensional inputs, and sensors inputs with bounded and unbounded accuracy. Moreover, our system works well in low bandwidth environments, thereby capturing a wide range of real sensor applications.

System assumptions. Throughout the paper, we rely on two assumptions:

- To achieve meaningful robustness against pollution attacks, the number of malicious sensors should be bounded (less than 1/2 or 1/3 of the total number, depending on concrete applications and algorithms).
- While both the server and a fraction of sensors may be malicious, the server should not collude with malicious sensors [40, 42]. (We stress that in our system sensors can collude, and in fact, we consider a strong adversary can coordinate malicious sensors to compromise the system.)

The first assumption is a standard one in distributed systems and multi-party computation systems where sensors, ideally, are independently distributed in different hosts (running diverse software and hardware), and the adversary has only limited capacity to compromise the system overall.

The second assumption is also a common assumption that has been used in a large number of practical multi-party computation systems [14, 15, 21, 24, 40, 42, 58]. Our system therefore is suitable for applications where the server and sensors lack motivation to collude, or the adversary lacks the means to corrupt both the server and some sensors simultaneously. For example, a satellite does not own the GPS devices, and the adversary may lack means to compromise the satellite and some GPS devices together. Another example is that soldiers with sensors may be captured by the enemies who may be unable to compromise the headquarter, which is typically well protected. As another one, the assumption makes sense for submarines and underwater sensors.

Our contributions. We summarize our contribution as follows:

- We motivate, clarify, and formalize the problem of server-aided, privacy-preserving, and intrusion-tolerant multi-sensor fusion.
- We provide an efficient and expressive framework supporting a variety of key fault-tolerant sensor fusion algorithms. The framework opportunistically leverages the bandwidth and computation asymmetry property in the sensor fusion setting and uses a novel combination of techniques from distributed systems (Byzantine fault-tolerant sensor fusion [54]) and multi-party computation (Yao's garbled circuits [73]). In addition, we use new techniques for achieving system liveness (which would otherwise leverage slower solutions using more than one server), make extensive optimizations on the framework, and tackle various other security and reliability issues.
- We make a general compiler specifically for our clients-server-sensors setting by extending and optimizing TinyGarble [68]. TinyGarble is designed for two-party computation (for garbler and evaluator), but our compiler works for settings involving clients (garblers), server (evaluator), and sensors (garbled input providers).
- We build a practical system for server-aided multi-sensor fusion using Raspberry Pi Zero W. Our experimental evaluation demonstrates that the system is highly efficient and scalable for *both* failure-free *and* failure scenarios.

II. Related Work

Comparison with differentially-private systems. Apple [22] and Google [33] use differential privacy [25, 26] mechanisms to compute aggregate statistics. In these systems, each sensor adds random noise and the noisy data will be aggregated to get an estimate of aggregated values. These systems allow us to achieve robustness, but have to trade between accuracy and privacy. Increasing the noise level reduces information leakage for individuals, but also reduces the estimated accuracy.

Essentially, the goals between differentially-private systems and our system are orthogonal. In a differentially-private system, the server will know a noisy version of the value measured by each sensor, and thus learn much more than that in our system. However, the client learns slightly less than our system because of the noise added.

Comparison with privacy-preserving data fusion and aggregation. Most privacy-preserving data aggregation systems only protect individual sensor inputs but fail to handle malicious sensors that attempt to sway the aggregated value [16, 18, 28, 38, 39, 48]. Only a handful provide a partial solution by leveraging a cryptographic proof that sensor input has a specific property (say, is within a predetermined range) [21, 24, 67]. PwoP takes a fundamentally different approach by *tolerating* malicious sensor inputs without asking for input validity proofs. Other works (e.g., [18, 38]) provide privacy-preserving aggregation with tolerance to benign (crash) sensor failures, i.e., where some sensors fail to provide their inputs. Instead, our system deals with Byzantine failures, where corrupted sensors can provide the server arbitrary values.

A number of privacy systems [27, 38, 57, 67] additionally provide differential privacy. These systems still do not formally defend against pollution attacks.

SIA [17] explored a setting where individual sensor inputs do not need to be privacy-protected but the central server needs to *verifiably* provide clients with correct values. This helps

achieve integrity. In contrast, PwoP achieves both privacy and integrity.

Comparison with alternative approaches to intrusion-tolerant and privacy-preserving sensor fusion. In addition to GC-based approach, we also provide alternative solutions with the same goal as PwoP: one using order-preserving encryption, and one using set representation. We describe the two approaches in Appendix IX, and compare them with PwoP in detail. Summarizing, the set-representation-based approach only applies to honest-but-curious sensors and a very limited number of fusion algorithms, and the approach based on order-preserving encryption leaks too much information and cannot easily achieve integrity.

Fault-tolerance and garbled circuits. Nielsen and Orlandi [60] built LEGO for two-party computation in the malicious case. In LEGO, the garbler first sends many gates, and the receiver tests if they are constructed correctly by opening some of them. Then the parties run interactively to solder the gates (as Lego blocks) into a circuit. They use a fault-tolerant circuit to ensure a valid output from a majority of good ones. In contrast, our system exploits the fault tolerant features of the underlying algorithms to achieve a garbled circuit based, privacy-preserving system that can tolerate pollution attacks, returning correct results even in the presence of Byzantine failures and malicious attacks.

Non-colluding multi-party computation. The assumption that a number of parties do not collude is not only used to build theoretical multi-party computation [6, 9, 32, 40], but used in practical multi-party computation systems [14, 15, 21, 24, 42, 58]. Among these systems, many use garbled circuits as a building block (e.g., [14, 15, 40, 42, 58]). Our notion of non-collusion follows this line of research, but has an architecture that is different from all these existing systems. In our setting, we assume that clients only have means to contact the server, and we assume the server and sensors do not collude. Note that non-collusion of parties does not imply that the parties are trusted. Rather, it simply means these parties do not work together (i.e., share internal states).

A few systems [14, 15, 58] relying on the non-colluding assumption work in the three-party computation only, using modern mobile devices. They attempt to resolve different problems from ours.

Efficient GC implementations. Starting from Fairplay [53], a large number of GC tools (compilers or implementations) have been proposed [8, 36, 47, 52, 68, 74]. Our system is based on (but makes significant modifications to) TinyGarble [68], an approach that in addition to using state-of-the-art optimizations such as free-XOR [46], row reduction [59], fixed-key AES garbling [8], and half gates [75], leverages logic synthesis to reduce the size of circuits.

Related attacks. Pollution attacks have also been studied in other areas such as network coding [1] and personalized services [72]. We work with a fundamentally different setting, focusing on data and sensor pollution attacks.

Our pollution attack scenarios are also different from those of Sybil attacks [23] where an adversary may forge multiple or even an unlimited number of identities to damage distributed systems. While Sybil attacks and pollution attacks may share somewhat the same goal, Sybil defenses [4] offer no help for defending against data pollution.

III. System and Threat Model

The setting. As depicted in Fig. 1, our system, PwoP, consists of a number of clients (control programs), a single server (proxy, averager, aggregator), and a set of sensors. Client and server are denoted c and S respectively. We denote the number of sensors by n, and a bound on the number of faulty sensors by g. The set of sensors is denoted as $\Pi = (s_1, \cdots, s_n)$. Let l be the length of the sensor input. Let k be the security parameter of cryptographic primitives.

In PwoP, a client sends a request to the single server, and the server collects readings from some or all of the sensors. Then the server runs a sensor fusion algorithm and sends the aggregated result to the client. Clients and sensors only communicate with the single server. In particular, sensors neither need to know each other, nor send one another any information.

Throughout the paper, we assume authenticated and private channels.

Threat model. A participant (client, the server, or sensor) that faithfully executes our protocol to completion is said to be *correct*; otherwise it is *Byzantine* or *malicious*. The behaviors of malicious participants are limited only by the cryptographic assumptions that we employ. A Byzantine participant that conforms to the protocol until some point at which it simply stops executing (permanently) is said to *crash*. A correct participant can nevertheless be *semi-honest*, namely it conforms to the protocol but may additionally preserve the transcript of everything it observes, in an effort to glean information to which it is not entitled. Participants (Byzantine or semi-honest) may also be required to be *non-colluding* (e.g., [40, 42]), which informally means that they do not share information except as explicitly prescribed by the protocol. The non-colluding assumption has been used in many practical multi-party computation systems [14, 15, 42, 58].

In PwoP, clients are semi-honest, while the server and some sensors can be malicious. We have designed but have not implemented an enhanced system that can defend against malicious clients; see Sec. IX.

Goals. PwoP aims to achieve privacy, correctness, and liveness.

- **Privacy**: If a semi-honest client does not collude with the server and does not collude with sensors, then it learns only the aggregate result returned from the server but nothing more. If the server does not collude with the client and does not collude with sensors, then it learns nothing about sensor inputs or the final result.
- **Correctness**: If the server is correct and no more than g sensors are Byzantine, then the only response a correct client will accept is the correctly aggregated result, in the sense discussed in the next section.
- **Liveness**: If the server is correct, if no more than g sensors are Byzantine, and if all Byzantine sensors crash, then each correct client receives a reply to its request.

IV. Fault-Tolerant Sensor Averaging Algorithms

We briefly survey the key fault-tolerant sensor fusion/averaging algorithms [19, 54–56, 65].

Marzullo's algorithms [54, 55]: M-g-U, M-g, M-g-m, and M-op. The study of fault-tolerant sensor averaging algorithms

dates to two seminal works by Marzullo [54, 55]. We collectively call them Marzullo's algorithms.

In the presence of n sensor values from n replicated sensors, a *fault-tolerant sensor averaging algorithm* [54] is used to compute a correct aggregated value even if some of the individual sensors are incorrect (in which case the sensor is said to be malicious, Byzantine, or simply faulty). Marzullo [54] considered the case where each individual sensor value can be represented by an interval $I = [u, v]$ over the reals. Let $(u - v)$, the width of the interval, denote the *accuracy* (or *inaccuracy*) of a sensor. Let $(u + v)/2$ be the *midpoint* or *center* of the interval. Then, a sensor value is correct if the interval I it returns contains the actual value of the measured feature, and the sensor is faulty otherwise. The goal of Marzullo's algorithm is to find the minimum (and correct) interval given n different intervals $I = \{I_1, \cdots, I_n\}$, with at most $g < n$ of those being faulty. The fused interval is at least as accurate as the range of the *least* accurate individual non-faulty sensors.

We start by introducing algorithms where the number of failed sensors g is known. The underlying idea is follows: Since g or less sensors are incorrect, any $(n - g)$ mutually intersecting sensors (i.e., *clique*) may contain the correct value. The algorithm computes the "cover" (not the "union") of *all* $(n - g)$-cliques.[1] Let lo be the smallest value contained in at least $n-g$ of the intervals and hi be the largest value contained in at least $n - g$ of the intervals. Then, the correct aggregated result is the interval $[\mathsf{lo}, \mathsf{hi}]$.

Marzullo [54] describes a general algorithm with $O(n \log n)$ complexity to compute this result. It uses a *sweeping* idea: First, sort all the endpoints of all the intervals. Second, moving from the lowest value to the highest value, keep track of the number of intervals containing each value. The final result can then be determined from these counts. The algorithm cost is dominated by the sorting procedure. Additional care is needed when the rightmost endpoint value of one interval coincides with the leftmost one of another interval, indicating that one interval ends exactly as another begins. Whether such an occurrence is deemed as a valid duration may depend on applications, but this should be agreed upon beforehand. In this paper we follow Marzullo's convention [54], which considered the occurrence as being valid.

The above algorithm can apply to the case of arbitrary failures with unbounded inaccuracy, and to the case of arbitrary failures with bounded inaccuracy, where the maximum length of the interval is known and values that are too inaccurate can be detected. Marzullo's algorithm needs $3g + 1$ and $2g + 1$ sensors to tolerate g arbitrary failures with unbounded inaccuracies and bounded inaccuracies, respectively. We used M-g-U and M-g to denote the above two cases.

It is not uncommon to require the averaging algorithm to only return the midpoint of the interval. This may even be more desirable in a privacy-preserving setting, as providing the lo and hi values might reveal too much unnecessary information. We write M-g-m to denote this variant.

Marzullo [55] also gave a solution to the case where the system parameter g is unknown or unspecified. The goal is to find the cover for the *maximum* intersection groups for all the intervals. Thus, the algorithm is "optimistic" (instead of "optimal"), and we write M-op to denote this one.

[1]Picking the cover instead of the union can help preserve the shape of the sensor value.

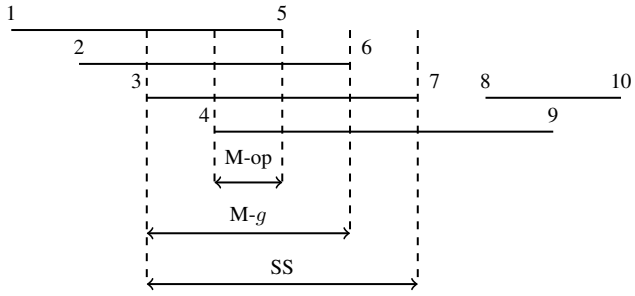

Fig. 2. A five-sensor system with M-g, M-op and SS. The sensor input intervals are [1, 5], [2, 6], [3, 7], [4, 9], [8, 10]. The resulting intervals are [3, 6], [4, 5] and [3, 7] for M-g, M-op and SS respectively.

Schmid and Schossmaier [65]: SS. Marzullo's algorithms may exhibit a somewhat irregular behavior: it is possible that when Marzullo's algorithms are applied to two slightly different input sets, the output may be quite different. This is formalized as violation of the *Lipschitz condition* regarding a certain metric [50].

Schmid and Schossmaier [65] offered a solution, SS, which can satisfy the Lipschitz condition. The algorithm is simple: Given n intervals $I_i = [u_i, v_i]$ $(1 \leq i \leq n)$, (at most) g of which may be faulty, SS simply outputs $[\max^{g+1}\{u_1, \cdots, u_n\}, \min^{g+1}\{v_1, \cdots, v_n\}]$, where $\max^{g+1}\{u_1, \cdots, u_n\}$ denotes the element $u_{j_{g+1}}$ in the ordering $u_{j_1}, \cdots, u_{j_n}$ of $\{u_1, \cdots, u_n\}$ from largest to smallest, and $\min^{g+1}\{v_1, \cdots, v_n\}$ denotes the element $v_{j_{g+1}}$ in the ordering $v_{j_1}, \cdots, v_{j_n}$ of $\{v_1, \cdots, v_n\}$ from smallest to largest. While SS shares the same worst case performance as Marzullo's, SS may generate a larger output interval.

Chew and Marzullo [19]: ChM. Chew and Marzullo generalized the approach to handle the case of *multidimensional* values. We will cover all the most efficient algorithms described in their paper, including an algorithm for general d-dimensional rectangles (ChM-dD) and an algorithm for general d-dimensional rectangles that have the same size and orientation (ChM-dD-sso). To tolerate g failed sensors, these algorithms use $dg + 1$ and $2g + 1$ sensors respectively. Note that for the case of ChM-dD-sso, the total number of sensors n does not depend on the dimension d. We defer the concrete description of these algorithms to where we need them.

Example. To help understand the algorithms described, we describe an example in Fig. 2 which shows how three one-dimensional algorithms (M-g, M-op, and SS) work. All the three algorithms deal with bounded accuracy and use $2g + 1$ sensors to tolerate g failures.

As in Fig. 2, the input intervals for the sensors are [1, 5], [2, 6], [3, 7], [4, 9] and [8, 10]. For all the algorithms, all input endpoints need to be sorted.

To find the left endpoint of the resulting interval for M-g, we can imagine that there is a vertical line sweeping from left to right. The vertical line can stop at the leftmost point that intersects $n - g = 3$ intervals. In the example, this point is 3. Similarly, to find the right endpoint, a vertical line can sweeping from right to left, and find the right end of the

resulting interval (6). Thus, the resulting interval is [3, 6].

Instead of outputting an interval, M-g-m will output the midpoint of the resulting interval generated by M-g.

In contrast to M-g, M-op algorithm does not need to know the g value a-priori. A vertical line will sweep over all the endpoints and finds the leftmost and rightmost points that intersect with the maximum input intervals. In the example, point 4 and 5 are covered by four input intervals, while the rest endpoints are covered by at most three input intervals. Thus, M-op will output [4, 5] as the result.

For SS, one need to find the $(n-g)$-th smallest left endpoint and the $(n-g)$-th largest right endpoint. In the example, point 4 and point 7 are picked as they are the third smallest left end and the third largest right end, respectively.

The example would be easily extended to explain M-g-U with unbounded accuracy. However, it requires at least $3g+1$ sensors to tolerate g failures.

In Appendix XIII, we show an example on how d-dimensional algorithms (ChM) work.

V. PwoP

This section presents PwoP. We first describe the key building block of PwoP—garbling schemes [7]—and then the design of PwoP.

A. Garbling Schemes

Bellare, Hoang, and Rogaway (BHR) [7] introduced the notion of a *garbling scheme* as a first-class cryptographic primitive. Here we mainly adopt this abstraction but tailor it for our purposes; specifically, we require that *all* the garbling scheme algorithms be *dominated* by random coins. The change is only notational.[2]

A garbling scheme is a tuple of algorithms $\mathcal{G} = (\mathsf{Gb}, \mathsf{En}, \mathsf{Ev}, \mathsf{De})$. Gb takes as input 1^k, a random coin r and a Boolean circuit f, and outputs a garbled circuit F. En takes an input x and a random coin r and outputs a garbled input X. Ev takes a garbled circuit F and garbled input X and outputs a garbled output Y. De takes a garbled output Y and a coin r and outputs a plain-circuit output y (or $\perp$).

We require a correctness condition on garbling schemes: if $\mathsf{F} \leftarrow \mathsf{Gb}(1^k, r, f)$, then $\mathsf{De}(r, \mathsf{Ev}(\mathsf{F}, \mathsf{En}(r, x))) = f(x)$.

In our work, we require the prv.sim (privacy), obv.sim (obliviousness), and aut (authenticity) security definitions in BHR, which we briefly describe here:

- prv.sim (privacy): There is a simulator $\mathcal{S}$ that takes as input $(1^k, f, f(x))$ and produces output which is indistinguishable from (F, X, r) generated normally.
- obv.sim (obliviousness): There is a simulator $\mathcal{S}$ that takes as input $(1^k, f)$ and produces output which is indistinguishable from (F, X) generated normally.
- aut (authenticity): Any adversary should not be able to generate a $Y' \neq \mathsf{Ev}(\mathsf{F}, X)$ such that $\mathsf{De}(r, Y') \neq \perp$.

B. PwoP Design

The system presented in this section deals with the scenario where clients are semi-honest, the server is malicious, a fraction of sensors are malicious, and the server should be non-colluding with any other participants. We deal with the scenario with malicious clients in Sec. IX.

PwoP with no liveness. The general idea behind PwoP is as follows: The client is responsible for generating a garbled circuit; then sensors contribute garbled inputs; and finally the server evaluates the function using the garbled inputs, and sends the client the garbled output.

Each time the client wants to obtain a fused result of sensors inputs, the client and the sensors need to agree on a *fresh*, random coin r that is used to garble the circuit and garble the inputs respectively, and they should prevent the value r from being known by the server. In the semi-honest model, we can easily achieve this by allowing the client to dictate the coin.[3] In PwoP, we assume that a client shares a symmetric, pairwise key with each sensor. A client chooses a random coin and wraps the coin using an authenticated encryption with the pairwise keys shared. The ciphertexts will be sent to the server who will distribute them to respective sensors. Alternatively, we can assume public key infrastructure and our system can be easily adapted.

Also, the client can send both the wrapped coins and the garbled circuit in the same round, saving one communication round.

The above approach opportunistically leverages the bandwidth and computation asymmetry property in the sensor fusion setting, where the bandwidth between clients and the server is ample, but the bandwidth between sensors and the server is limited. In fact, it is very common in modern systems to shift part of work to clients to improve the service throughput and reduce the latency. As we will show, the overhead incurred by the circuit generation and the circuit transmission in PwoP, for practical parameters, is negligible. Moreover, in our approach, the circuit size (related to the accuracy of the returned results to clients) can be flexibly decided by clients. In addition, letting the client code allows the circuit to be precomputed off-line.

We describe PwoP with no liveness (i.e., with no guaranteed output delivery) in Fig. 3, using a language of garbling schemes that is slightly modified from BHR. We make black-box use of a general sensor averaging function fta, and we defer the circuit design, optimization, and justification to the next section.

Setup and inputs: Let fta be any sensor averaging function in Sec. IV. Let $\mathcal{G} = (\mathsf{Gb}, \mathsf{En}, \mathsf{Ev}, \mathsf{De})$ be a garbling scheme. Let $\langle \text{REQ} \rangle$ be a client request that contains the function description. Let $x_\Pi = \{x_1, \cdots, x_n\}$ and $X_\Pi = \{X_1, \cdots, X_n\}$ be sensors' inputs and garbled inputs respectively.

00 Client c selects a random coin r, runs Gb (using r) to garble a circuit F for fta, and sends F, $\langle \text{REQ} \rangle$, encrypted coins to server S.
01 S forwards $\langle \text{REQ} \rangle$ and the corresponding encrypted coins to sensors.
02 Sensors Π run En (using r and x_Π) and send S garbled inputs X_Π.
03 Server S runs Ev on X_Π and sends the garbled output Y to c.
04 Client c runs De (using r and Y) to get $\mathsf{fta}(x_\Pi)$.

Fig. 3. PwoP with no liveness. When the server receives a garbled circuit, the server collects data from sensors and returns the reply to the client.

[2] In BHR's original definition, only Gb is probabilistic, while the rest are deterministic. In their syntax, there are two more notations e (encoding information) and d (decoding information). For all the efficient garbling schemes known, both e and d can be generated by a single random coin together with some associate data.

[3] In the malicious model, agreeing on a common coin can be achieved by modifying a threshold coin-flipping protocol [13] to the server-aided setting; each message in this coin-setup protocol is transmitted between the client and corresponding sensor using end-to-end encryption, with the server simply passing these encrypted messages.

We have the following theorem establishing the security of the above scheme.

Theorem 1: The PwoP protocol achieves Privacy and Correctness for sensor fusion function fta.

Privacy follows from both privacy and obliviousness of the garbling scheme. Specifically, the first part of Privacy (i.e., the client only learns the aggregated value) can be derived from the privacy of the garbling scheme. This is because in PwoP the client only obtains (F, Y, r), while adversary in the privacy game of the garbling scheme obtains (F, X, r) and Y is fully determined by X and F. The second part of Privacy can be trivially obtained from the privacy of the garbling scheme. Correctness follows from the authenticity of the garbling scheme and the correctness of fta. Note that correctness holds even against a malicious server as long as the server does not collude with sensors.

However, the scheme in Fig. 3 does not achieve liveness: if some of these sensors fail to provide their garbled inputs, the server cannot evaluate the circuit.

Supporting general feedback function. In PwoP, after the server evaluates the garbled circuit, it can also send the garbled outputs to sensors which may run De to get the fused value. In a control program, the data sent to sensors which may be co-located with actuators are *feedback data*, according to which actuators can perform some prescribed operations.

Moreover, PwoP can be easily extended to the case where S provides sensors with output from an *arbitrary* feedback function (not necessarily the same function which the client asks to compute). To achieve this, clients not only garble the function that they need but also the feedback function for sensors and actuators.

Discussion. This basic scheme shares some similarities with both Feige, Kilian, and Naor (FKN) [30] and Kamara, Mohassel, and Raykova (KMR) [40]. The difference is that FKN and KMR only involve a server and parties (in our case, sensors) and the server needs to send back the garbled output to the parties, while in our model, the server needs to return the garbled output to the client and *optionally* to the sensors. In FKN and KMR, the server and one party do heavy work that is linear in the size of the circuit, while in our case, each sensor's role is symmetric and each sensor only does work that is linear in the size of its input. The security of KMR requires only obliviousness and authenticity of the garbling scheme, while PwoP additionally requires privacy of the garbling scheme.

Our scheme is also similar to Naor, Pinkas, and Sumner (NPS) [59], one designed specifically for auctions. In NPS, there is an auction issuer who generates the circuit, a number of bidders who send their garbled values, and an auctioneer who computes the garbled values and returns the final result to all bidders. Instead of relying on an external, trusted circuit issuer, our circuit generator is just one participating client (who would also expect a reply from the server). Moreover, NPS uses proxy oblivious transfer to provide the parties with the garbled input, but we choose to use an agreed common coin, just as FKN and KMR, for the purpose of efficiency and scalability.

The servers in both FKN and NPS can learn the output, while KMR and ours do not.

C. Achieving Liveness

We now describe PwoP with liveness. In our approach, the absence of a reply from a sensor will be treated as an input of $[-\infty, +\infty]$ (or the prescribed upper and lower bounds), which means this reply will not be counted. The reason why we can do this is that our fault-tolerant algorithms can natively tolerate empty (meaningless) inputs as long as the number of these inputs (and together with malicious inputs) are g-bounded. More specifically, if the server does not receive the garbled input from some sensors in time, it will ask the client to send corresponding garbled inputs for the missing sensors for values $[-\infty, +\infty]$. When using algorithms with bounded accuracy, the client will generate a random interval with maximum accuracy. PwoP with liveness is described in Fig. 4.

Setup and inputs: Let fta be any sensor averaging function in Sec. IV. Let $\mathcal{G} = (\mathsf{Gb}, \mathsf{En}, \mathsf{Ev}, \mathsf{De})$ be a garbling scheme. Let $\langle \text{REQ} \rangle$ be a client request that contains the function description. Let $x_\Pi = \{x_1, \cdots, x_n\}$ and $X_\Pi = \{X_1, \cdots, X_n\}$ be sensors' inputs and garbled inputs respectively.

00 Client c selects a random coin r, runs Gb (using r) to garble a circuit F for fta, and sends F, $\langle \text{REQ} \rangle$, encrypted coins to server S.
01 S forwards $\langle \text{REQ} \rangle$ and the corresponding encrypted coins to sensors.
02 Sensors Π run En (using r and x_Π) and send S garbled inputs X_Π.
If S does not receive all the garbled inputs before the times expires, it requests from the client missing garbled inputs that encode $[-\infty, +\infty]$.
04 Server S runs Ev on X_Π and sends the garbled output Y to c.
05 Client c runs De (using r and Y) to get $\mathsf{fta}(x_\Pi)$.

Fig. 4. PwoP with liveness. The protocol completes with 1 round in the failure-free scenario, and with 2 rounds if some garbled inputs are missing.

We thus have the following theorem.

Theorem 2: PwoP described in Fig. 4 implements Liveness in synchronous environments.

In synchronous environments, if the server is correct and if all Byzantine sensors crash, the server will request garbled inputs from the correct client after the timer expires. As the client is correct, the server will receive these "dummy" garbled inputs and the server can evaluate the garbled circuit and send a reply to the client.

Discussion. Achieving liveness *efficiently* has been a difficult problem for garbled circuit based multi-party computation. Most of prior works on GC (as surveyed in Sec. II) achieve liveness by introducing multiple servers and use secret-sharing based techniques to help liveness. This is not only less efficient in general, but also requires significant communication and interaction, which makes it ill-suited in bandwidth and energy restricted environments.

Schemes in [40, 42] do not achieve liveness. NPS [59] considered a "denial of service attack by bidders," which is essentially a form of liveness. Similar to our case, NPS needs to prevent a corrupt bidder from sending incorrect values or simply not sending values to the server. Their approach is to prove to the auction issuer that the bad event occurs, and then a dummy zero value will be provided. However, in our case we aimed at minimizing the interaction with the client (circuit generator). In addition, our scheme achieves liveness for random coin based approach, while NPS uses proxy oblivious transfer.

A similar problem was studied by Feigenbaum, Pinkas, Ryger, and Saint-Jean [31]. They simply provided a solution that requires all the parties to pre-commit their values to two "non-colluding" servers *before* a circuit is garbled. Their application scenarios are very different from ours.

In PwoP, malicious sensors may contribute ill-formed garbled inputs so that the server ends up with outputting $\perp$. It

is vital that the server can "quickly" tell if a garbled input is correct. We discuss how PwoP can achieve fast detection for ill-formed inputs in Sec. IX. This way, PwoP can be extended to achieve liveness for g-bounded Byzantine sensors.

VI. Circuit Design and Optimizations

This section describes how to design *efficient* circuits for the fault-tolerant algorithms in Sec. IV. There are three good reasons why we need the effort.

First, while multiple generic GC compilers or tools that can translate a program to a circuit exist [8, 36, 47, 52, 68, 74], there is significant room for improvement for some specific programs. Our optimization requires non-trivial efforts and analysis for the correctness of the design.

Second, among all the GC compilers, TinyGarble [68] is generally deemed to be (one of) the most efficient one, especially for large programs, as it incorporates state-of-the-art optimizations such as free-XOR [46], row reduction [59], fixed-key AES garbling [8], and half gates [75], and more importantly, uses *logic synthesis* to reduce the size of circuits. However, TinyGarble only supports a limited number of components that we need. We therefore aim to build modular components that can be readily used for our fault-tolerant algorithms.

Third, different from the conventional circuit design, the sensors in our setting can be malicious and may contribute malicious garbled inputs. Garbled circuits, or in general, multiple-party computation, offer no protection on malicious inputs. For instance, we cannot rely on sensors to contribute well-formed input intervals (see below).

Convention. Before the protocol starts, all participants should agree on a representation for the intervals and d-dimensional values. For intervals, each possible value is given an integer label with l bits, and we assume the lower and upper bounds are 0 and σ respectively. Clearly, $l = \log \sigma$. Likewise, d-dimensional values are given a d-dimensional vector with each component being an interval which can be represented by a fixed number of bits.

A. Circuit design for M-g-U

Overview. Our complete circuit design for M-g-U is depicted in Fig. 5. To implement Marzullo's algorithm, we first need to sort all the input endpoints of sensor inputs, resulting in a sorted array of $2n$ values (considering each sensor is providing one interval in the form of two endpoints). This is achieved using *modified sorting networks*. For each point in this sorted array, we need to count how many input intervals can cover this point. This is handled by adding 1 to an intersecting interval counter if the point is a left end of an input interval, and subtracting 1 if it is a right end. After that, we compare the intersecting interval counters for each point with $n-g$, in order to find the points that are covered by *exact* $n-g$ intervals. We do this using *index select*. As in Fig. 5, the left end of the resulting interval is the output of the *max value min index* module.

Likewise, the circuit for computing the right end of the resulting interval can be implemented in a symmetric way by again running the modified sorting networks and index select. However, we will show that we can *reuse* the modified sorting networks and index select module for computing the right interval, as shown in Fig. 5.

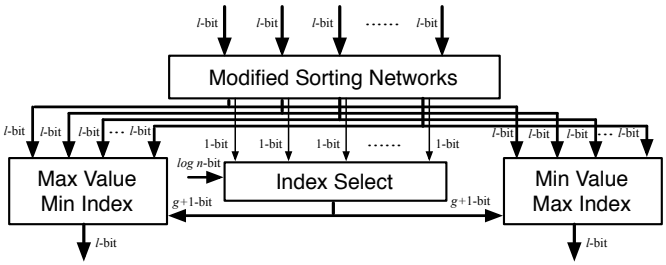

Fig. 5. The complete circuit design for M-g-U. n is the number of sensors, l is the length of the endpoint of the input interval, and g is the number of faulty sensors.

Our circuit design in detail. Instead of using $(+1, u)$ and $(-1, v)$ to represent an interval $[u, v]$ (as in the code version of Marzullo's algorithm), in our circuit design, each sensor provides an interval in the form of two values u, v to the server. This is because sensors may be malicious and it will result in wrong result if the left end provided by some malicious sensor is actually larger than the right end. Therefore, at the first level of our modified sorting network, we need to add an array of compare-and-swap modules to sort the two values from the same sensor. Note that the above problem is not what the conventional garbled circuit design would care about.

Before we proceed, let's recall sorting networks [5, 20, 44], which are circuits that sort a sequence into a monotonically increasing sequence. The core building block of a sorting network is a compare-exchange circuit, which takes as input a pair of values (x, y) and returns a sorted pair (x', y') so that $x' = \mathit{min}(x, y)$ and $y' = \mathit{max}(x, y)$. To realize this building block, prior constructions [35, 45, 46] used the idea of *compare then conditional-swap:* the circuit keeps two inputs unchanged if and only if the comparator returns 1, i.e., x is less than y. For our design, the first layer of our modified sorting network guarantees that the two values from the same sensor will form an interval with its right end always greater or equal to its left end. Then we taint these two values with $+1$ and -1 respectively to indicate the order of two endpoints. For our implementation, we use one bit to represent for ± 1, i.e., we use "1" and "0" to represent $+1$ and -1 respectively.

Starting from the second layer of our modified sorting networks, it is essentially sorting networks built from compare-exchange components that have a modified comparator, as illustrated in Fig. 6. Instead of using the *less-than* comparator, we follow Marzullo's algorithm [54] to realize a comparator, $<_m$, as defined below: Given two inputs x and y, each of which is of the form $s||u$ where $s \in \{1, 0\}$ and $|u| = l$, define $x <_m y = (\mathit{lsb}_l(x) < \mathit{lsb}_l(y)) \vee (\mathit{lsb}_l(x) = \mathit{lsb}_l(y) \wedge \mathit{msb}_1(x) > \mathit{msb}_1(y))$, where lsb_l and msb_l represent the least significant l bits and the most significant l bits respectively. In words, $x <_m y$ if and only if the value part of x is less than that of y, or the value parts happens to be equal and the sign part of x is greater than that of y. Note that the sign part of a left end is encoded by 1.

We follow [45] to realize the conventional less-than circuit and equal-to circuit (which takes advantage of the free-xor technique). However, observing that when $\mathit{lsb}_l(x) = \mathit{lsb}_l(y)$ and $\mathit{msb}_1(x) = 1$ it does not matter we swap or not the two inputs, we can further simplify the circuit, leading to a circuit

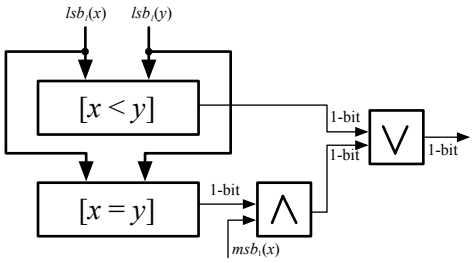

Fig. 6. The comparator component.

exactly as in Fig. 6.

For our implementation, while asymptotically optimal sorting network exists [2], PwoP uses Batcher sorting network [5, 44] which has much better performance for practical parameters, as studied in [52].

Now we describe how to find the position of the minimum value of the resulting interval by our index select module composed by a prefix sum circuit and an array of equality checkers, as shown in Fig. 7. All the sorted one-bit inputs, representing $+1$ or -1, first go through a prefix sum circuit to compute their prefix sums. Prefix sum circuit allows one to compute on input $(z_1, z_2, \cdots, z_n)$ and produce as output $(m_1, m_2, \cdots, m_n)$, where $m_j = z_1 + z_2 + \cdots + z_j$ for $1 \leq j \leq n$. Indeed, this circuit fits perfectly for our purpose as the intersecting interval counter in M-g-U. A straightforward instantiation of n-prefix sum circuit requires n additions.[4] Note that for each addition when performing prefix sums, we can exploit the free-xor circuit from [45].

The next layer is to convert every prefix sum which is equal to the value $n-g$ to 1 and convert the rest to 0 otherwise. This can be trivially instantiated via simple equal-to circuit [45]. Observing that not every position in the array of prefix sum can possibly equal $n-g$, we can apply another optimization that only implements comparators for the positions where $n-g$ can be the possible output. To be precise, for an array with $2n$ values provided by n sensors, only $g+1$ positions can possibly have a prefix sum equal $n-g$. We prove the correctness of this optimization in Theorem 3 in Appendix XII. This observation reduces the number of comparators and the width of max value min index by roughly 83.3% of that with a straightforward implementation, because it only compares at $g+1$ positions, instead of $2n = 6g+2$ positions in a straightforward implementation. However, the size reduction for the other algorithms is slightly smaller, with a 75% reduction, since $n = 2g+1$ for the other algorithms.

Then these $g+1$ values go through a max value min index circuit which computes the value with the minimum index and the maximum value (which is 1). Effectively, it outputs the leftmost point which covers $n-g$ sensor input intervals. We can easily modify the circuit from [45] to obtain this circuit.

To compute the maximum value, we find that one can reuse the result of modified sorting networks and index select modules. Specifically, we just need to left shift one position of the sorted input value array (shifting is free in circuits), and apply it to a max value max index circuit. Then we will generate the right end of the result interval. The proof of correctness of this optimization is in Theorem 4 in Appendix XII. Since the whole circuit complexity is dominated by the sorting network, this optimization avoids another sorting networks and index select circuit for computing the right end of the resulting interval, thereby halving the computation overhead.

[4]In a system that can evaluate garbled circuits in parallel, we recommend to implement a parallel prefix sum circuit as mentioned in (cf. [64, Chapter 2.6]), which has a depth of $O(\log n)$ and $O(n)$ additions.

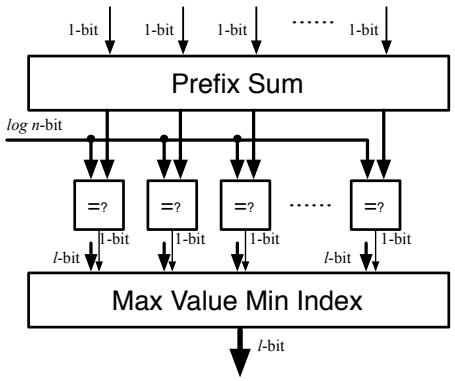

Fig. 7. The index select module and the max value min index module. The index select module is composed by a prefix sum circuit and an array of equality checkers.

Caveat. A tempting way of designing the circuit for M-g-U might be to first use the sorting network and then "release" all the flag information, which the server could use to easily determine the indexes and the final result. One might think that this flag information would not reveal much information, or one might be willing to trade this information for efficiency and the compactness of the circuit. However, we are not sympathetic to this viewpoint, stressing that the flag information leaks too much to the server. Indeed, the server can tell from the flag information the topology of all the intervals, and even tell the number of failed sensors. Also note that only sorting the entries (without giving the final interval) would not lead to a garbling scheme that achieves authenticity when the server is malicious.

B. Circuit Design for Other Algorithms

Circuit design for M-g. M-g is a sensor averaging algorithm for arbitrary failures with *bounded* accuracy. Recall that our convention is that each interval is of the range $[0, \sigma]$, where σ is some pre-determined, maximum value. Here we need to levy an additional requirement on the input interval $[u, v]$ (where $0 \leq u \leq v \leq \sigma$): the difference between u and v must be less than or equal to some threshold t, i.e., $v - u \leq t$; otherwise the interval is deemed as being "invalid."

Circuit design for M-g-m. Our circuit for M-g-m builds on that of M-g. We calculate the midpoint of the resulting interval by adding the two end points together, without dividing it by 2. Since it is a division by 2 (a public constant), users can divide it by themselves without using garbled circuits.

Circuit design for M-op. Recall that M-op wishes to find the cover for the maximum intersection groups for all the intervals. Our circuit for M-op is similar to that of M-g-U. The difference is that no equality checkers are needed in index select, since g is unknown. Also, the comparators in max value min (max) index will need to be integer comparators, instead of binary comparators, because we are looking for the points that have the highest prefix sum, which implies that these points intersect

TABLE I. PWOP CHARACTERISTICS. The column labeled "type" specifies if the algorithm handles intervals (I) or d-dimensional values (dD). n is the total number of sensors, g is the upper bound on the number of malicious sensors, k is the security parameter (in this paper, 128 bit), and l is the length of sensor input. The column labeled "complexity" specifies the time complexity of the algorithm in the plain setting. The columns labeled "server circuit," "sensor time," "sensor communication," and "#rounds" specify for server circuit complexity, sensor time complexity (measured using the number of pseudorandom function calls), sensor communication complexity, and the round complexity for PwoP, respectively.

algorithms	type	#sensors	description	complexity	server circuit	sensor time	sensor communication	#rounds
M-g-U	I	$3g+1$	unbounded accuracy	$O(n\log(n))$	$O(ln\log^2(n))$	$O(l)$	$O(lk)$	1 or 2
M-g	I	$2g+1$	bounded accuracy	$O(n\log(n))$	$O(ln\log^2(n))$	$O(l)$	$O(lk)$	1 or 2
M-g-m	I	$2g+1$	only reveal midpoint	$O(n\log(n))$	$O(ln\log^2(n))$	$O(l)$	$O(lk)$	1 or 2
M-op	I	$2g+1$	"optimistic"	$O(n\log(n))$	$O(ln\log^2(n))$	$O(l)$	$O(lk)$	1 or 2
SS	I	$2g+1$	Lipschitz condition	$O(n\log(n))$	$O(ln\log^2(n))$	$O(l)$	$O(lk)$	1 or 2
ChM-dD	dD	$dg+1$	d-dimension	$O(dn\log(n))$	$O(ldn\log^2(n))$	$O(dl)$	$O(dlk)$	1 or 2
ChM-dD-sso	dD	$2g+1$	same size & orientation	$O(dn\log(n))$	$O(ldn\log^2(n))$	$O(dl)$	$O(dlk)$	1 or 2

with the maximal number of input intervals.

However, it is worth noting that some of the optimizations for M-g-U circuit cannot be applied here, because g is unknown to M-op algorithm. Specifically, we are unable to select only $g+1$ values for comparisons and selections, we have to feed all the values into the max value min (max) index modules.

Circuit design for SS. The core functionality of SS is selection, with which one can easily derive the final interval. To realize the selection circuit, one solution is to directly apply a conventional sorting circuit on the input values, and we can easily obtain a circuit with $O(ln\log^2 n)$ complexity. Another possibility is to use the probabilistic method from Wang et al. [70]. This gives a non-zero error probability, but leads to a circuit with $O(ln\log k)$ complexity. In PwoP, we implement only the method with zero error probability.

Circuit design for ChM-dD. One can project the region for sensors onto each of the d-dimensions, and thus obtain d independent 1-dimensional problems, which can be resolved separately. The combined rectangles are called *projection rectangles*, and this approach is called the *projection approach*. It is easy to see that since the projection algorithm naturally gives rise to a d-rectangle, no further adjustments on the shape of the rectangles are needed.

We can generalize our M-g-U circuit to handle the case of d-dimensional projection rectangles in a straightforward manner: d independent circuits will be generated and the size for ChM-dD is just d times as large as that of M-g-U.

According to Chew and Marzullo [19], the projection approach for ChM-dD is "the method-of-choice for some situations," but there are two disadvantages to the approach [19, Section 3.1]: first, some information may be lost; second, the size of the average result may be larger than necessary.

Circuit design for ChM-dD-sso. ChM-dD-sso is referred to as the algorithm finding the fault-tolerant rectangle from ones with the same size and orientation. In this case, one may simply compute the projection d-rectangles in $O(dn\log n)$ time tolerating $g \leq n/2$ failures. Note that the maximum tolerable failures g is independent of the number of dimensions d. We note that the combined rectangle may be smaller than the original size. For our design, we do not attempt to construct a rectangle with exactly the same size as hinted by [19], and instead we simply provide the user with the minimum one and the user can easily construct a rectangle from the result.

Circuit design for d-rectangles with bounded accuracy. For ChM-dD and ChM-dD-sso, there are variants for arbitrary failures with bounded accuracies. Their circuits can be constructed following an analogous line as what we did for the case of M-g-U and M-g.

VII. PWOP CHARACTERISTICS

Table I shows the characteristics of PwoP. PwoP covers seven Byzantine fault-tolerant sensor fusion algorithms. PwoP is designed specifically for cyber-physical systems that work in computation and bandwidth restricted environments.

In PwoP, the communication round is optimal: one round for failure-free scenarios, and two rounds for failure scenarios. Meanwhile, sensors performs "minimal" computation, depending on l (and d) only. The total bits that each sensor sends to the server only depend on l and the security parameter k (and d). These metrics in PwoP outperform the ones in prior privacy-preserving sensor aggregation systems with pollution attacks mitigated such as [21, 49] and an alternative approach using set representation described in Appendix XI.

VIII. IMPLEMENTATION AND EVALUATION

A. Implementation

We make a general garbled circuit compiler specifically for our clients-server-sensors setting. We achieve this by modifying and extending TinyGarble [68], the state-of-the-art garbled circuit compiler for secure two-party computation, which incorporates state-of-the-art garbled circuit optimizations [8, 46, 59, 75] and leverages logic synthesis to optimize and compress circuits. Specifically, we first decompose TinyGarble into three parties: clients (garbled circuit generators), the server (garbled circuit evaluator), and sensors (garbled input providers). We then modify the system to be coin-based: clients and sensors now take as input shared random coins. Last, we modify the garbled circuit evaluation function so that garbled output is hidden from the server (which is essential to achieving Privacy).

As TinyGarble does not support efficient sorting networks or other primitives we need, we directly build optimized modules needed and then build the circuits described in Sec. IV. We manually write the hardware circuits in Verilog, as we find doing so also provides circuit reduction compared to the ones using high-level programming languages and high level synthesis. The resulting circuits go through another logic synthesis process by Synopsys Design Compiler to obtain the netlists of the implemented algorithms. Lastly, we applied the V2SCD tool in TinyGarble to convert netlists into simple circuit description files, which can be taken as the inputs of TinyGarble framework.

To show the practicality of our system, we build a multi-sensor fusion system using Raspberry Pi Zero W (1GHz,

single core CPU and 512MB RAM). In our system, each Raspberry Pi functions as a sensor, providing the server with garbled inputs for its sensor inputs. Raspberry Pi Zero W is the cheapest Raspberry Pi device supporting WiFi connection, and is smaller than a credit card. Notice that, we selected Raspberry Pi Zero W just because of its built-in WiFi module, not its computational power. As a fact, the computation on each sensor only contains a few pseudorandom function evaluations and XOR operations, so it is very efficient for any sensor with minimum computational power to compute. The server runs on an Intel Core i7-4790 processor, and the client runs on another computer with an Intel Core i7-4702HQ processor. Sensors and the server are connected using one wireless router. To support concurrent transmissions of sensor inputs, we implement multi-threading for data collection at the server side. The client and the server are physically connected via an Ethernet cable.

The length of all sensor values (i.e., $l = \log(\sigma)$) is set to eight bits. For an interval, each sensor needs to garble a 16-bit input, resulting in a 256-byte garbled input. Similarly, garble inputs for a rectangle in our setting takes 512 bytes.

Our system achieves liveness for sensors that are subject to crash failures. As described in Sec. V, our system works in synchronous environments and needs a two-round communication when there are sensor failures.

Initially, PwoP uses heartbeat messages to detect failed sensors before the sensor fusion protocol starts, and sets a timer for each request to detect new crash faulty sensors. Specifically, the server sets a timer each time a request is sent to sensors. If it does not receive some garbled inputs in time, it marks the corresponding sensors as faulty. It then asks the client to provide garbled inputs for $[-\infty, +\infty]$ (or random values in the case of algorithms with bounded accuracy) for these sensors, evaluate the circuit, and send the garbled output to the client.

B. Evaluation

Overview. We have evaluated PwoP using up to 19 sensors, with seven different algorithms: M-g, M-g-m, M-g-U, M-op, SS, ChM-dD, and ChM-dD-sso. We evaluate PwoP for these algorithms under different network sizes (the number of sensors). We use g to represent the network size for these algorithms, and total number of sensors can be found in Table I. We measure the latency, throughput, and scalability in both failure and failure-free scenarios. For failure-free scenarios, each sensor will provide a well-formed garbled input, even if the underlying value is faulty. In contrast, our failure scenario captures crash failures where some sensors do not provide the server with garbled inputs in time.

Overall, PwoP has low latency and high throughput, and can be deployed in many real-time applications, e.g., monitoring pressure and water leaks in a water distribution system [3].

Latency. The latency evaluation for the seven algorithms are depicted in Fig. 8 for $g = 1$ and 2. The latency of the whole process takes only 12 to 32 milliseconds.

To understand the performance bottleneck of the system, we look into the time consumed by each operation for the whole process. We find that the communication between the server and sensors is two orders of magnitude larger than the cryptographic operations and the rest of the communication. Circuit generation (at the client side), circuit transfer (between the client and the server), garbled input generation (at the sensor side), and circuit evaluation (at the server side), collectively, take only several milliseconds, even when $g = 9$ and the total number of sensors is 19. This implies that if we can reduce the communication latency between the server and sensors, we can easily boost the performance of PwoP.

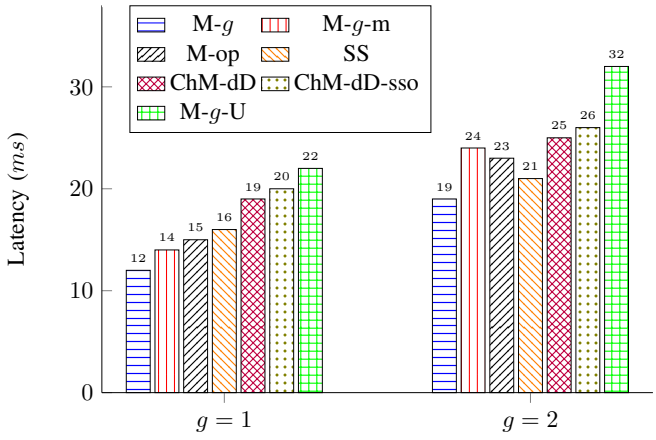

Fig. 8. Latency (in ms) of PwoP in failure-free scenarios for $g = 1$ and 2. This and subsequent figures are best viewed in color.

To compare the performance of different algorithms, we notice that for a given g, M-g, M-g-m, M-op, and SS (the four one-dimensional algorithms) have roughly the same latency, because the latency is dominated by the communication time between the server and sensors, and the total size of garbled inputs transmitted for these algorithms is exactly the same. This observation is less visible small g's (see Fig. 8), but becomes apparent for larger g's (see Fig. 10).

For the same g, the latency of M-g-U is larger than that of the other one-dimensional algorithms. Indeed, to tolerate g sensor failures, M-g-U requires more sensors ($3g+1$) than the other algorithms ($2g+1$). As a consequence, it requires more data to be transmitted, and takes longer time to process all the data.

Besides these one-dimensional algorithms, we implement and evaluate d-dimensional algorithms. In particular, we set $d = 2$ for our evaluation, as 2-dimensional sensor fusion is useful for many applications (e.g., measuring the location of a physical object). The size of each sensor input is set to 8 bits, and a two-dimensional rectangle requires 32 bits to represent. This doubles the size of the garbled inputs, comparing to the case for one dimensional algorithms. We find that the increased data size is reflected in latency: the latency of ChM-dD and ChM-dD-sso is larger than that of M-g, M-g-m, M-op, and SS for the same number of sensors.

Throughput. The throughput of PwoP for all seven algorithms for $g = 1$ and 2 is shown in Fig. 9. Similar to the latency analysis, the throughput of M-g, M-g-m, M-op, and SS is almost the same for the same g values. The throughput of M-g-U is consistently lower than the other one-dimensional algorithms for the same g, due to a larger number of connected sensors. In addition, the throughput of ChM-dD and ChM-dD-sso is lower than the other algorithms, because of more transmitted input data. The throughput bottleneck is again due to the communication between the server and sensors.

In PwoP, we did not implement batching of sensor inputs for the communication between the server and sensors. We conjecture that the use of batching would provide higher

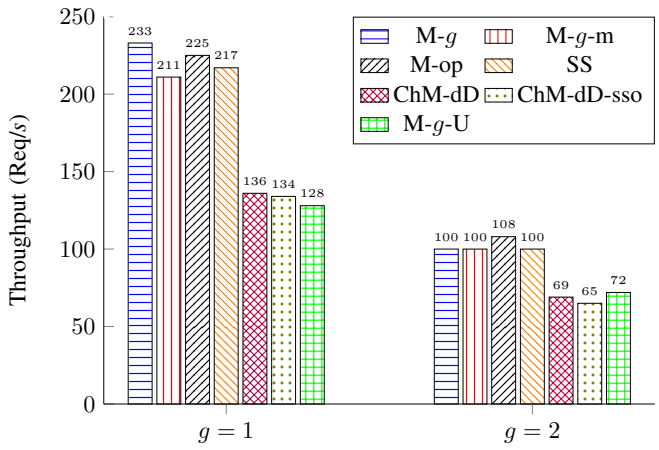

Fig. 9. Throughput of PwoP in failure-free scenarios for $g = 1$ and 2.

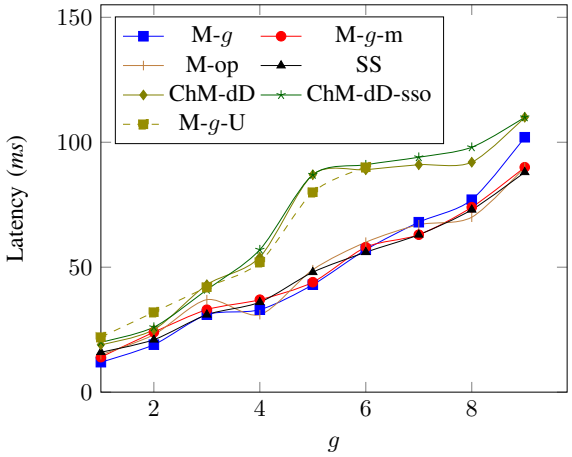

Fig. 10. Scalability of the latency of PwoP in failure-free scenarios for $g = 1$ to 9. (except M-g-U, for which we measured for $g = 1$ to 6)

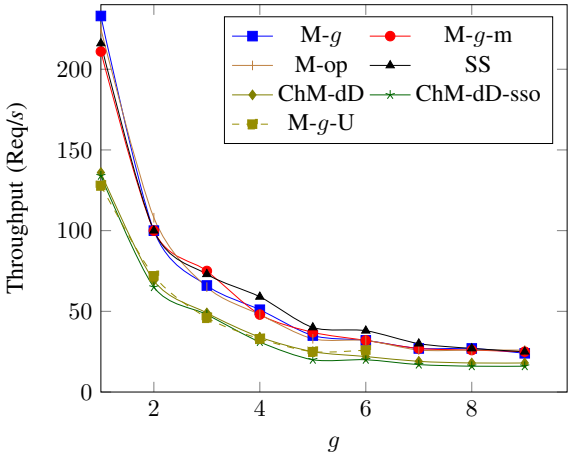

Fig. 11. Scalability of the throughput of PwoP in failure-free scenarios for $g = 1$ to 9. (except M-g-U, for which we measured for $g = 1$ to 6)

throughput and a meaningful trade-off between latency and throughput. We did not implement parallel evaluation of garbled circuits, and this is not needed in our current implementation for which the communication is the bottleneck.

Scalability. We evaluate the scalability of PwoP using up to 19 sensors. The largest latency we obtained in our evaluation is

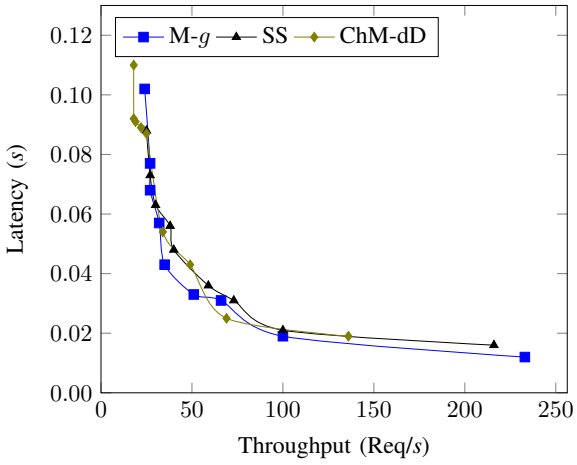

Fig. 12. Latency vs throughput of PwoP in failure-free scenarios.

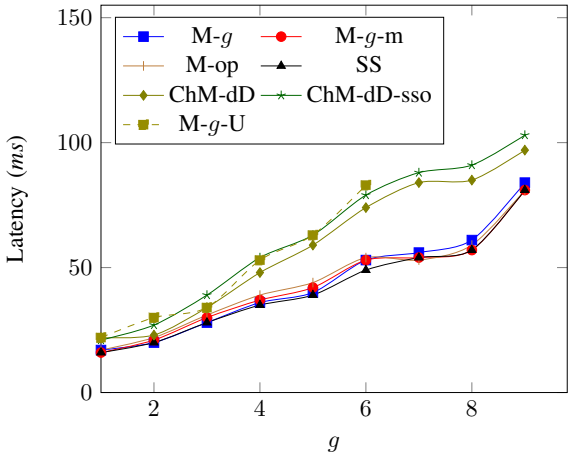

Fig. 13. Scalability of the latency of PwoP in failure scenarios for $g = 1$ to 9 (except M-g-U, for which we measured for $g = 1$ to 6)

the latency of ChM-dD for $g = 9$, which is only 0.11 seconds, as shown in Fig. 10. This demonstrates that PwoP, for all algorithms and all network sizes that we tested, is efficient. In Fig. 10, as the number of sensors increases from 3 to 19, the latency grows slowly and linearly from 0.012 to 0.11 seconds.

If we compare the latency between M-g-U and the other algorithms for the case where the total number of sensors are equal, e.g., $g = 6$ for M-g-U and $g = 9$ for M-g-m, the latency difference is almost not noticeable. This, from a different angle, confirms that the latency is dominated by the communication, not the cryptographic operations.

In addition, the scalability of throughput of PwoP is shown in Fig. 11. As the number of sensors increases, the throughput reduces from 233 Req/s to 24 Req/s for M-g, and similarly for others. This is due to the network congestion incurred by the increase of the number of sensors.

We also report latency vs. throughput of three algorithms M-g, SS and ChM-dD in Fig. 12. For clarity, we omit the rest as the curves of the other algorithms are very similar to the three.

Performance in failure scenarios. In failure scenarios, we test the case where g out of n sensors fail at the same time. The latency and the throughput for PwoP for $g = 1$ to 9 in failure scenarios are shown in Fig. 13 and Fig. 14, respectively.

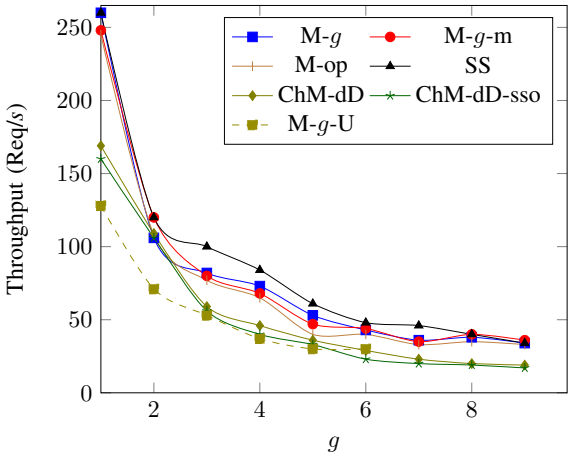

Fig. 14. Scalability of the throughput of PwoP in failure scenarios for $g = 1$ to 9. (except M-g-U, for which we measured for $g = 1$ to 6)

The latency part does not count the time-out value set by the server. It should be set to different values for different applications. We find that there is no observable difference between the failure and failure-free scenarios for latency if not counting the time-out value. Indeed, the extra communication does not add visible overhead.

We also find that the throughput in failure scenarios are slightly better than the that in failure-free cases. The reason is that comparing with getting garbled inputs from sensors via WiFi, it is faster to get garbled inputs from the client, which is connected with the server using an Ethernet cable physically. Moreover, as now there are less sensors competing for the bandwidth between sensors and the server, their communication is faster.

IX. Discussion

PwoP with malicious clients. We have studied how to design schemes with semi-honest clients. Indeed, clients have incentives to learn correct results from the server. However, malicious clients can learn information which should be kept private. In particular, malicious clients might code a circuit that computes functions that are different from the one as claimed by clients. For instance, clients may code circuits on computing how many and which sensors have failed, or a circuit that reveals (some) sensors' individual inputs.

It is easy to secure our system against malicious clients. This is (easily) achieved by augmenting and modifying the Salus protocol due to Mamara, Mohassel, and Riva [42]. As PwoP, we require the clients to garble the circuit (instead of one of the parties), and we let the client and the server run a cut-and-choose protocol. To reduce the communication overhead, we may use the optimal strategy of Shelat and Shen [66] that challenges 3/5 of the circuits, and may use the random seed checking technique due to Goyal, Mohassel, and Smith [34].

False garbled inputs detection. We describe how the GC evaluator (the server, in our case) can efficiently detect if garbled inputs are valid.

Recall that malicious sensors may provide the evaluator with invalid garbled inputs. Depending on which garbling schemes are used, an invalid garbled input may cause the evaluator to return $\perp$ (before all the gates are evaluated), or return an invalid garbled output that appears correct to the evaluator but will not be accepted by the client.

Our goal is to detect false garbled inputs in an as efficient way as possible. The technique can be used for PwoP to achieve Liveness even against fully Byzantine sensors. To this end, we first need a garbling scheme with the underlying encryption being able to detect false garbled inputs. Ideally, we require the evaluator to perform a minimum number of circuit gate evaluation, preferably, only on circuit input gates.

Our notion of security for the underlying encryption scheme is similar to that of Lindell and Pinkas (LP) [51], where they used a symmetric encryption scheme with an *elusive* and *efficiently verifiable* range. It allows the evaluator to tell which gate entry in a four-row table is correct. If one is unlucky, it would try all of the four entries. Most recent garbled circuit implementations chose to use the "point-and-permute" technique [62], which leads to garbling schemes that allow to decrypt exactly one entry per gate.

However, our motivation is different from LP. Here we simply require that the evaluator would return "no" if the garbled input for a single bit is not one of the required two encodings. Not surprisingly, LP does meet the simple notion above. Other (and more efficient) constructions are possible, such as using (deterministic) authenticated encryption and PRP with redundancy.

For efficiency, we may use a hybrid construction as follows: At entry level of garbled circuit, we use a garbled scheme which can efficiently detect invalid garbled inputs. When evaluating the remaining garbled circuits, we can still use the most efficient GC implementation so far. We comment that for the entry level garbling scheme, we can still use point-and-permute technique, which requires the evaluator to decrypt only one table entry — we only give malicious sensors one chance.

Additional care is needed: we need to design a circuit that has a small number of entry-level gates and encompasses all the inputs. Ideally, the verification should be run in parallel. Also, we need to consider free-XOR gates, as evaluating them does not take any encryption/decryption and they do not allow efficient detection.

Take the circuit for M-g-U as an example. We may choose to use Batcher sorting network [5], not only because it is the most efficient one, but also because it allows efficient and parallel false garbled inputs detection. Before going to the subsequent circuit, Batcher sorting network first runs $2n/2 = n$ compare-exchange operations in parallel. The entry-level circuit (with n compare-exchange operations) is rather small compared to whole sorting network circuit (with $O(n \log^2 n)$ compare-exchange operations).

X. Conclusion

We describe the design and implementation of PwoP, an efficient and scalable system for intrusion-tolerant and privacy-preserving multi-sensor fusion. PwoP has two distinguishing features: 1) PwoP can provably defend against pollution attacks without compromising privacy, and 2) PwoP is designed specifically to perform in computation and bandwidth restricted cyber-physical systems. To show the practicality of our approach, we build a secure multi-sensor fusion system, covering a variety of practical application scenarios.

ACKNOWLEDGMENT

This work was partially supported by NSF grant CNS-1413996 for MACS: A Modular Approach to Cloud Security.

REFERENCES

[1] S. Agrawal, D. Boneh, X. Boyen, and D. M. Freeman. Preventing pollution attacks in multi-source network coding. *Public Key Cryptography 2010*, pp. 161–176.

[2] M. Ajtai, J. Komlós, and E. Szemerédi. An $O(n \log n)$ sorting network. In *STOC*, 1983.

[3] M. Allen, and A. Prels, and M. Lqbal, and S. Srirangarajan, and H.B. Llm, and L. Glrod, and A. J. Whittle, Real-time in-network distribution system monitoring to improve operational efficiency. *Journal-American Water Works Association 2011.*

[4] L. Alvisi, A. Clement, A. Epasto, S. Lattanzi, and A. Panconesi. SoK: The evolution of Sybil defense via social networks. *2013 IEEE Symposium on Security and Privacy*.

[5] K. E. Batcher. Sorting networks and their applications. In *Proc. AFIPS Spring Joint Computer Conference*, 1968.

[6] D. Beaver, S. Micali, and P. Rogaway. The round complexity of secure protocols (extended abstract). In *Proc. of 22nd STOC*, pages 503–513, 1990.

[7] M. Bellare, V. Hoang, and P. Rogaway. Foundations of garbled circuits. In *ACM CCS 12*, pages 784–796. ACM Press, October 2012.

[8] M. Bellare, V. Hoang, S. Keelveedhi, and P. Rogaway. Efficient garbling from a fixed-key blockcipher. In *IEEE Symposium on Security and Privacy 2013*, pp. 478–492, IEEE Computer Society Press, May 2013.

[9] M. Ben-Or, S. Goldwasser, and A. Wigderson. Completeness theorems for non-cryptographic fault-tolerant distributed computation. *STOC 1988.*

[10] A. Boldyreva, N. Chenette, Y. Lee, and A. O'Neill. Order-preserving symmetric encryption. In *EUROCRYPT*, 2009.

[11] A. Boldyreva, N. Chenette, and A. O'Neill. Order-preserving encryption revisited: improved security analysis and alternative solutions. In *CRYPTO*, 2011.

[12] R. R. Brooks and S. S. Iyengar. Robust distributed computing and sensing algorithm. *Computer 29*, 6 (1996), 53–60, 1996.

[13] C. Cachin, K. Kursawe, and V. Shoup. Random oracles in Constantinople: Practical asynchronous Byzantine agreement using cryptography. *Journal of Cryptology* 18(3), 219–246.

[14] H. Carter, C. Lever, and P. Traynor. Whitewash: outsourcing garbled circuit generation for mobile devices. In *ACSAC 2014*, pages 266–275, 2014.

[15] H. Carter, B. Mood, P. Traynor, and K. Butler. Secure outsourced garbled circuit evaluation for mobile devices. In *Proceedings of the USENIX Security Symposium*, 2013.

[16] C. Castelluccia, A. C.-F. Chan, E. Mykletun, and G. Tsudik. Efficient and provably secure aggregation of encrypted data in wireless sensor networks. *TOSN* 5(3): 20:1-20:36 (2009).

[17] H. Chan, A. Perrig, B. Przydatek, and D. Song. SIA: Secure information aggregation in sensor networks. In *Journal of Computer Security – Special Issue on Security of Ad-hoc and Sensor Networks*, Volume 15 Issue 1, Pages 69-102, January 2007. Early version in *SenSys '03.*

[18] T.H. Chan, E. Shi, and D. Song. Privacy-preserving stream aggregation with fault tolerance. *FC 2012.*

[19] P. Chew and K. Marzullo. Masking failures of multidimensional sensors. *SRDS 1991*, 32–41, 1991.

[20] T. H. Cormen, C. E. Leiserson, R. L. Rivest, and C. Stein. *Introduction to Algorithms.* MIT Press, 2nd edition, 2001.

[21] H. Corrigan-Gibbs and D. Boneh. Prio: Private, robust, and scalable computation of aggregate statistics. *NSDI 2017*, pages 259–282, 2017.

[22] Differential privacy overview — Apple. https://images.apple.com/privacy/docs/Differential_Privacy_Overview.pdf

[23] J. R. Douceur. The Sybil Attack. *International Workshop on Peer-to-Peer Systems (2002)*, pp. 251–260.

[24] Y. Duan, J. Canny, and J. Zhan. P4P: practical large-scale privacy-preserving distributed computation robust against malicious users. *USENIX Security 10*, 2010.

[25] C. Dwork. Differential privacy. *ICALP 2006*, pp. 1–12.

[26] C. Dwork, K. Kenthapadi, F. McSherry, I. Mironov, and M. Naor. Our data, ourselves: Privacy via distributed noise generation. *EUROCRYPT 2006*, pp. 486–503.

[27] T. Elahi, G. Danezis, and I. Goldberg. PrivEx: Private collection of traffic statistics for anonymous communication networks. *CCS (2014)*, ACM, pp. 1068–1079.

[28] Z. Erkin and G. Tsudik. Private computation of spatial and temporal power consumption with smart meters. *ACNS 2012*, pp 561–577.

[29] Z. Erkin, A. Piva, S. Katzenbeisser, R. L. Lagendijk, J. Shokrollahi, G. Neven, and M. Barni. Protection and retrieval of encrypted multimedia content: When cryptography meets signal processing. In *EURASIP Journal on Information Security*, 2007, Article ID 78943, 2007.

[30] U. Feige, J. Kilian, and M. Naor. A minimal model for secure computation (extended abstract). In *26th ACM STOC*, pages 554–563, ACM Press, May 1994.

[31] J. Feigenbaum, B. Pinkas, R. Ryger, and F. Saint-Jean. Secure computation of surveys. In *EU Workshop on Secure Multiparty Protocols (SMP)*, ECRYPT, 2004.

[32] O. Goldreich, S. Micali, and A. Wigderson. How to play any mental game. *STOC (987.*

[33] U. Erlingsson, V. Pihur, and A. Korolova. RAPPOR: Randomized aggregatable privacy-preserving ordinal response. *CCS (2014)*, ACM, pp. 1054–1067.

[34] V. Goyal, P. Mohassel, and A. Smith. Efficient two party and multi party computation against covert adversaries. In *EUROCRYPT '08*, Springer, pages. 289–306, 2008.

[35] Y. Huang, D. Evans, and J. Katz. Private set intersection: Are garbled circuits better than custom protocols? *NDSS 2012*, 2012.

[36] Y. Huang, D. Evans, J. Katz, and L. Malka. Faster secure two-party computation using garbled circuits. In *USENIX Security Symposium*, 2011.

[37] Y. Ishai and E. Kushilevitz. Private simultaneous messages protocols with applications. *Proceedings of the Fifth Israeli Symposium on Theory of Computing and Systems*, 1997.

[38] M. Jawurek and F. Kerschbaum. Fault-tolerant privacy-preserving statistics. *PETS 2012.*

[39] M. Joye, and B. Libert. A scalable scheme for privacy-preserving aggregation of time-series data. *FC 2013*, 2013.

[40] S. Kamara, P. Mohassel, and M. Raykova. Outsourcing multi-party computation. Cryptology ePrint Archive, report 2011/272, October 25 2011.

[41] S. Kamara, P. Mohassel, M. Raykova, and S. Sadeghian. Scaling private set intersection to billion-element sets. *FC 2014*, LNCS 8437, pages 195–215, 2014.

[42] S. Kamara, P. Mohassel, and B. Riva. Salus: A system for server-aided secure function evaluation. In *Proceedings of the ACM conference on Computer and communications security (CCS)*, 2012.

[43] F. Kerschbaum and Axel Schröpfer. Optimal average-complexity ideal-security order-preserving encryption.

[44] D. E. Knuth. *The Art Of Computer Programming — Volume 3 / Sorting and Searching.* Addison-Wesley, 2nd edition, 1998.

[45] V. Kolesnikov, A.-R. Sadeghi, and T. Schneider. Improved garbled circuit building blocks and applications to auctions and computing minima. In *International Conference on Cryptology and Network Security (CANS)*, 2009.

[46] V. Kolesnikov and T. Schneider. Improved garbled circuit: Free XOR gates and applications. In *ICALP 2008*, volume 5126 of LNCS, pages 486–498, Springer, July 2008.

[47] B. Kreuter, a. shelat, B. Mood, and K. Butler. PCF: A portable circuit format for scalable two-party secure computation. In *Proceedings of the USENIX Security Symposium*, 2013.

[48] K. Kursawe, G. Danezis, and M. Kohlweiss. Privacy-friendly aggregation for the smart-grid. *PETS 2011*, pp. 175–191.

[49] L. Lamport. Using time instead of timeout for fault-tolerant distributed systems. In *ACM Transactions on Programming Languages and Systems* 6 (2): 254–280, 1984.

[50] L. Lamport. Synchronizing time servers. Technical Report 18, Digital System Research Center, 1987.

[51] Y. Lindell and B. Pinkas. A proof of security of Yao's protocol for two-party computation. In *Journal of Cryptology, Volume 22, Issue 2*, pages 161–188, April 2009.

[52] C. Liu, X. Wang, K. Nayak, Y. Huang, and E. Shi. ObliVM: A programming framework for secure computation. In *IEEE Symposium on Security and Privacy (S&P)*, 2015.

[53] D. Malkhi, N. Nisan, B. Pinkas, and Y. Sella. Fairplay — a secure two-party computation system. In *Proceedings of the 13th conference on USENIX Security Symposium*, Volume 13, pages 287–302, USENIX Association, 2004.

[54] K. Marzullo. Tolerating failures of continuous-valued sensors. *ACM Trans. Comput. Syst.* 8(4): 284–304, 1990.

[55] K. Marzullo. Maintaining the time in a distributed system: an example of a loosely-coupled distributed service (synchronization, fault-tolerance, debugging). Ph.D. dissertation, Stanford University, Department of Electrical Engineering, February 1984.

[56] K. Marzullo and S. Owicki. Maintaining the time in a distributed system. *PODC 1983*, pages 295–305, 1983.

[57] L. Melis, G. Danezis, and E. De Cristofaro. Efficient private statistics with succinct sketches. *NDSS 2016*, Internet Society.

[58] B. Mood, D. Gupta, K. Butler, and J. Feigenbaum. Reuse it or lose it: More efficient secure computation through reuse of encrypted values. In *ACM CCS 2014*, pages 582–596, 2014.

[59] M. Naor, B. Pinkas, and R. Sumner. Privacy preserving auctions and mechanism design. In *Proceedings of the 1st ACM conference on Electronic commerce*, pages 129–139. ACM, 1999.

[60] J. B. Nielsen and C. Orlandi. LEGO for two-party secure computation. In *6th TCC*, Springer Volume LNCS 5444, pages 368–386, 2009.

[61] R. A. Popa, F. H. Li, and N. Zeldovich. An ideal-security protocol for order-preserving encoding. In *34th IEEE Symposium on Security and Privacy*, 2013.

[62] P. Rogaway. The round complexity of secure protocols. MIT Ph.D. Thesis, 1991.

[63] P. Rogaway. Authenticated-encryption with associated-data. *ACM CCS'02*, ACM Press, pp. 98–107, 2002.

[64] J. E. Savage. *Models of Computation — Exploring the Power of Computing*, Addison-Wesley, 1997.

[65] U. Schmid and K. Schossmaier. How to reconcile fault-tolerant interval intersection with the Lipschitz condition. *Distributed Computing* 14(2): 101–111, 2001.

[66] a. shelat and C.-H. Shen. Two-output secure computation with malicious adversaries. In *Proceedings of the Annual international conference on Theory and applications of cryptographic techniques*, 2011.

[67] E. Shi, T. H. Chan, E. Rieffel, R. Chow, and D. Song. Privacy-preserving aggregation of time-series data. In *NDSS 2011*, pp. 1–17, 2011.

[68] E. M. Songhori, S. U. Hussain, A.-R. Sadeghi, T. Schneider, and F. Koushanfar. TinyGarble: Highly compressed and scalable sequential garbled circuits. *SP '15.*

[69] Jeffrey M. Voas. Networks of 'Things.' NIST Special Publication (NIST SP) - 800-183.

[70] G. Wang, T. Luo, M. Goodrich, W. Du, and Z. Zhu. Bureaucratic protocols for secure two-party sorting, selection, and permuting. In *ASIACCS 2010*, pages 226–237, 2010.

[71] D. Westhoff, J. Girao, and M. Acharya. Concealed data aggregation for reverse multicast traffic in sensor networks: Encryption, key distribution, and routing adaptation. In *IEEE Transactions on Mobile Computing, 5(10)*: 1417–1431, 2006.

[72] X. Xing, W. Meng, D. Doozan, A. Snoeren, N. Feamster, and W. Lee. Take this personally: Pollution attacks on personalized services. *USENIX Security 13.*

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

[74] S. Zahur and D. Evans. Obliv-C: A Language for extensible data-oblivious computation. IACR Cryptology ePrint Archive 2015: 1153 (2015).

[75] S. Zahur, M. Rosulek, and D. Evans. Two halves make a whole: Reducing data transfer in garbled circuits using half gates. *EuroCrypt 2015.*

XI. Alternative Approaches to Intrusion-Tolerant and Privacy-Preserving Sensor Fusion

We describe two alternative approaches to intrusion-tolerant and privacy-preserving sensor fusion, and compare them with PwoP.

A. Order-Preserving Encryption Based Approach

Order-preserving encryption (OPE) [10, 11, 43, 61] is an encryption scheme where the order of ciphertexts matches that of the corresponding plaintexts. OPE is a powerful primitive most known to enable performing a large class of queries on encrypted databases. However, OPE is also suggested for use in encrypted data aggregation in sensor networks [71] and multimedia content protection [29].

We observe that non-interactive, deterministic OPE [10, 11] may be used to achieve the goal of fault-tolerant and privacy-preserving sensor fusion, yet for a limited class of problems, and with a much weak security guarantee. For instance, we consider how to do this for M-g-U: Initially, assume all sensors and the client share a group OPE key. Using this group key, each user encrypts only two values, i.e., the leftmost and rightmost endpoint of its input interval. Since all the encrypted values reveal their order information, the server is able to run the fault-tolerant sensor averaging algorithm in "plaintexts," and returns the leftmost and rightmost endpoint of the resulting interval to the client. Then the client can use the group key to decrypt the two ciphertexts.

The above construction is simple, but suffers from several problems. First, the construction leaks all the order information. For many applications, the order information is exactly what one strives to protect. Even worse, all existing non-interactive OPEs leak more than just the order of the values [61]. Correspondingly, even with the non-collusion assumption between the server and sensors, and allowing leaking all the order information, it is difficult, if not impossible, to prove the construction secure against even a semi-honest server under (any) appropriate simulation-based definition of security. Further, it is also unclear how to achieve integrity (i.e., ensure the server to faithfully return the client correct OPE values, rather than arbitrary values). Last, the construction can only apply to the sensor fusion algorithms with unbounded accuracy, but we do not know how to deal with the ones with bounded accuracy.

Interactive, ideal-secure OPEs that reveal no additional information on the plaintexts besides their order do exist [43, 61]. The interactive nature of these schemes, however, makes them ill-suited for our setting where some sensors may want to learn more information about other sensors.

B. Set Representation Based Approach

We extend the idea of multi-party set representation (SR) [40] to provide solutions for *some* (but not all) of the fault-tolerant fusion algorithms.

The idea for the set representation method is as follows: To compute the resulting interval, one may simply approximate

the *real* interval with as many discrete elements as possible. Let σ denote the number of elements in the entire universe. The idea can naturally lead to an algorithm with $O(n\sigma)$ time.

Before proceeding to our findings, let's briefly describe the server-aided private set intersection protocol by Kamara, Mohassel, Raykova, and Sadeghian (KMRS) [41]. Let S denote the set of party p_i. All the parties who have private inputs should first jointly generate a secret key for a pseudorandom permutation (PRP) E. Then each party randomly permutes the set $\mathsf{E}_k(S)$, and sends the permuted set to the server, which simply computes and returns the intersection of all the encrypted sets. The protocol above is secure with a semi-honest server or any collusion of malicious parties. Further techniques were developed by KMRS to ensure the protocol to remain secure against a malicious server.

We may base the idea to build a privacy-preserving and fault-tolerant scheme. Compared to the GC based approach, set representation based one has much lower client to server communication complexity, but much larger communication complexity and time complexity between sensors and the server. The property makes SR based approach not suitable for applications where sensors has limited bandwidth and computational power. Meanwhile, SR based approach only works for semi-honest sensors. (The reason is that a malicious sensor might not provide consecutive encrypted data.) This assumption is hard to justify, as the number of the sensors may be large and the sensors are distributed in different locations. Last, SR based approach only works for a rather limited set of fault-tolerant algorithms.

XII. Correctness Proof

Theorem 3: In the circuit designs of M-g-U, M-g, M-g-m, ChM-dD and ChM-dD-sso, only $g+1$ positions out of $2n$ can possibly have a prefix sum equal $n-g$.

Proof: Let $z_1, z_2, z_3, ...z_{2n}$ be an array of sign values (in the form of ± 1) of a sorted array in an ascending order generated by our modified sorting network described in Sec. VI. We denote the prefix sum by $A_j = \sum_{i=1}^{j} z_i$.

To prove the theorem, we just need to show the following two claims are correct:

1) If $\exists j$ such that $A_j = n-g$, then only A_{j+2k} can possibly equal $n-g$, where k is an integer and $1 \leq j+2k \leq 2n$.
2) If $A_j = n-g$, then $n-g \leq j \leq n+g$.

Indeed, given the two claims, we can find that there are in total $g+1$ positions that can possibly have a prefix sum equal $n-g$. They are $n-g+2h$, where h is an integer and $0 \leq h \leq g$. The theorem will then follow.

Proof of claim 1: Suppose $A_j = n-g$. Let us consider the set of z_i for $1 \leq i \leq j$. Since z_i can only be ± 1, we can only replace $+1$ with -1 or replace -1 with $+1$ to change A_j. Therefore A_j can only be $n-g+2k$, where k is an integer and $0 \leq n-g+2k \leq n$. This also implies that it is impossible for A_{j-1} and A_{j+1} to equal $n-g$; they can only possibly be $n-g+1+2k$.

Proof of claim 2: Since $A_j = \sum_{i=1}^{j} z_i$, and z_i can only be ± 1, the smallest index j, such that $A_j = n-g$, is $n-g$. Similarly, the largest index j, such that $A_j = n-g$, is $2n-(n-g) = n+g$.

Theorem 4: In the circuit designs of M-g-U, M-g, M-g-m, M-op, ChM-dD and ChM-dD-sso, modified sorting networks and index select are only needed once.

Proof: We use the same notation as above. To compute the minimum value (left end) of the resulting interval, we need to compute the prefix sum $A_j = \sum_{i=1}^{j} z_i$, and find the leftmost position j such that $A_j = n-g$. Similarly, to compute the maximum value of the resulting interval, we need to compute the postfix sum $B_j = \sum_{i=j}^{2n} z_i$, and find the rightmost position j such that $B_j = -(n-g)$. Notice that all z_i's come in pairs of $+1$ and -1, so the sum of all z_i's must be 0. This implies that for any j, $A_j + B_{j+1} = \sum_{i=1}^{2n} z_i = 0$. Thus we can directly obtain the array of B_j from A_j without performing the addition operations again. This saves an additional copy of modified sorting network and index select, including prefix or postfix sum and equality checkers (no equality checkers in the circuit of M-op, since g is unknown), for computing the right end of the resulting interval. Since $B_j = -A_{j-1}$, when we apply the sorted array of sensor inputs to the max value max index module, we need to left shift the array of sensor inputs for one position. Note that bit shifting is completely free in circuits.

XIII. Example for Multidimensional Algorithms

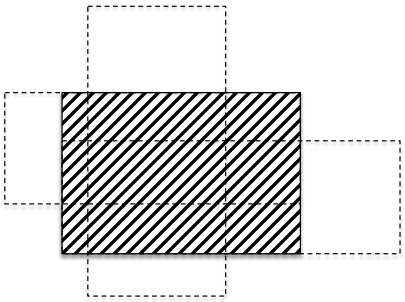

Fig. 15. Example of ChM-dD with $d = 2$. Three rectangles with dashed lines are the input rectangles, and the shaded area is the aggregated rectangle.

Fig. 15 shows an example of ChM-2D algorithm for a three-sensor system. The three rectangles with dashed lines are the input rectangles from three sensors. We run a M-g algorithm for both dimensions, leading to two resulting intervals. These two intervals construct the final resulting rectangle (shaded area in Fig. 15).