Privacy-Preserving Localization using Enclaves

Arslan Khan[†], Joseph I. Choi[*], Dave (Jing) Tian[†], Tyler Ward[*]
Kevin R. B. Butler[*], Patrick Traynor[*], John M. Shea[*], Tan F. Wong[*]
[†]Purdue University, {khan253, daveti}@purdue.edu
[*]University of Florida, {choijoseph007, tsward, butler, traynor, jshea, twong}@ufl.edu

***Abstract*—Localization is one form of cooperative spectrum sensing that lets multiple sensors work together to estimate the location of a target transmitter. However, the requisite exchange of spectrum measurements leads to exposure of the physical location of participating sensors. Furthermore, in some cases, a compromised participant can reveal the sensitive characteristics of all participants. Accordingly, a lack of sufficient guarantees about data handling discourages such devices from working together. In this paper, we provide the missing data protections by processing spectrum measurements within attestable containers or enclaves. Enclaves provide runtime memory integrity and confidentiality using hardware extensions and have been used to secure various applications [1]–[8]. We use these enclave features as building blocks for new privacy-preserving particle filter protocols that minimize disruption of the spectrum sensing ecosystem. We then instantiate this enclave using ARM TrustZone and Intel SGX, and we show that enclave-based particle filter protocols incur minimal overhead (adding 16 milliseconds of processing to the measurement processing function when using SGX versus unprotected computation) and can be deployed on resource-constrained platforms that support TrustZone (incurring only a 1.01x increase in processing time when doubling particle count from 10,000 to 20,000), whereas cryptographically-based approaches suffer from multiple orders of magnitude higher costs. We effectively deploy enclaves in a distributed environment, dramatically improving current data handling techniques. To our best knowledge, this is the first work to demonstrate privacy-preserving localization in a multi-party environment with reasonable overhead.**

I. Introduction

Cooperative spectrum sensing allows for wireless devices to cooperatively measure channel usage across space, frequency, and time. While such monitoring has a wide range of important applications, including finding unused spectrum for opportunistic use and spectrum-aware routing, cooperative localization of RF emitters is among the most important. Sensors deployed in multiple locations perform measurements of a transmitted signal and exchange this information to arrive at a more precise measurement than any single sensor could produce alone. Applications of localization include: military tracking of mobile targets [9], reclaiming unused spectrum for secondary users [10], and monitoring wildlife [11].

One of the challenges with localization is limiting the potential for data compromise. While sensor owners would do best to cooperate in order to produce the best possible estimate of the transmitter's location, the shared spectrum measurements can be intercepted by an adversary to localize the measuring sensors. In many scenarios, sensors are owned by multiple parties. Even in cases where all sensors are under the control of a single party (e.g., military applications), the compromise of any single device may yield the potentially sensitive locations of all other sensors.

In this paper, we address the challenge of minimizing data leakage while maximizing the benefits of measurement sharing in cooperative spectrum sensing. To achieve these ends, we consider a novel application of *enclave-based computing* to preserve the location privacy of sensors. Enclaves are an example of a trusted execution environment (TEE), whereby secure regions of memory are maintained that allow the execution of code unobservable from outside the enclave. Enclaves incur low computational overhead when compared to heavyweight cryptographic techniques. However, this isolation requires that enclaves do not trust any system-level services, such as system calls, IPC services, etc. Hence, we design our work as a self-contained application, with well-defined interfaces for properly vetting untrusted agents in the system. While the low overhead and application partitioning enables us to scale our design for any enclave system, we evaluate our system on Intel Software Guard Extensions (SGX) [12] and ARM TrustZone [13], which constitute the major market share of TEE systems. To the best of our knowledge, this is the first work to show that cooperative spectrum sensing can be done privately with reasonable overhead using trusted execution environments. In summary, we make the following contributions:

- **Apply Enclave-Based Computing to Spectrum Sensing:** We explore the design space of modern secure computation for spectrum sensing and apply a novel enclave-based approach for strong data protection.
- **Design and Implement Hardened Spectrum Sensing:** We design and implement a particle filter-based protocol for localization, protect the execution and privacy of this system using enclaves, and demonstrate its application in both centralized and decentralized architectures.
- **Measure and Evaluate Performance using Enclaves:** We demonstrate that our approach substantially improves security over traditional mechanisms, while adding only minimal overhead. Computing a 20,000 particle measurement function for Intel SGX and TrustZone, in either centralized or distributed configuration, results in an overhead less than 16ms and 267ms overhead respectively.

II. Background

A. Localization by Particle Filter

Localization refers to the determination of some target transmitter's physical location based on the inputs provided by multiple sensors; each sensor contributes information based on its spectrum measurements that can be fused with other sensors' measurements to localize a target. Cooperating sensors are not necessarily owned by a single entity; multiple parties may share a common objective. Participants may not necessarily trust one another and thus have an interest in minimizing exposure of their sensors to any other participants or to outside observers, who may potentially be adversarial. Localization may be a continuous process that occurs at regular intervals, such as when the target is a mobile transmitter.

Localization may be carried out with the help of a central fusion center (FC) or in a distributed manner, discussed further in Section III. One practical method for localization is the particle filter [14]–[16]. Particle filters allow discretization of the posterior belief[1] of a transmitter's location in a way that guarantees the amount of data being transferred. Particle filters are also well-suited for handling noisy measurement data affected by real-world environmental irregularities and an ambiguous channel path-loss exponent (PLE).[2] Particle filters do not require the prior assumption of a Gaussian relationship. Alternative approaches to particle filters use the received signal strength (RSS) and location of each sensor directly [17]–[22] or rely on the power difference of arrival [23].

B. Enclaves

Enclaves are hardware-protected containers for holding and performing operations on sensitive data, providing confidentiality and integrity guarantees. Since enclave protections originate from hardware, solutions built on enclaves run at native execution speed. Another important property provided by enclaves is attestability; that is, enclave users can verify the integrity of the contents and functionality loaded within an enclave before deciding to use it.

The current dominant iteration of the enclave concept is given by Intel Software Guard Extensions (SGX) [12]. SGX sets aside processor-enforced regions of memory, the contents of which are only accessible from within the enclave and are thus protected from unauthorized access by other applications and even privileged code. When connecting to an enclave, users may perform local attestation (for enclaves hosted by the same physical machine) or remote attestation (for enclaves hosted separately) to verify the enclave's integrity.

ARM TrustZone [13] is another widely deployed approach to TEE for embedded systems. TrustZone provides a a *secure world* which acts as a secure enclave. Hardware logic in the bus fabric together with processor core extensions make it possible to isolate secure world assets from rest of the system. System software can access secure world assets with the help of Secure Monitor Calls (SMC).

III. Design Considerations

We begin our discussion of secure localization techniques by presenting our security model. We then explore the design space to evaluate multiple options and finally explain the design of our system.

[1]As a function of the likelihood that the transmitter is located at the particle.
[2]PLE quantifies reduction of a signal or radio wave as it traverses a medium.

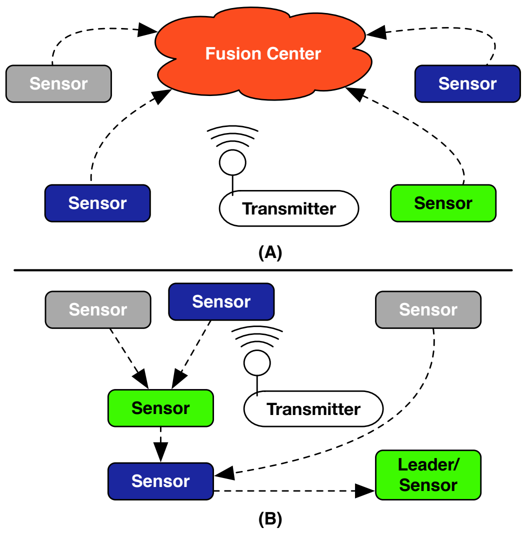

Fig. 1: Sensors (colored according to their various owners) work together to locate a target transmitter. In centralized localization (A), the fusion center (FC) takes measurement inputs from all participants. In decentralized localization (B), a leader elected among the sensors establishes an ordering, and computation is distributed among the sensors, which iteratively combine their measurement inputs and resample particles to arrive at the final estimate.

A. Security Model

We assume the adversary does not already have a global view[3] of all sensors present in the localization area, in which case the location privacy of all sensors is trivially broken. We do not consider colluding adversaries. We exclude side-channel and microarchitectural attacks against enclaves.

1) Goal: Our goal is to preserve *location privacy*: no participant should be able to determine the physical location of any other participant involved in a localization round. Data leakage should be minimized upon compromise (i.e., compromising a single participant should not leak others' locations).

2) Architectures: We consider two different architectures: centralized (*ArchC*) and decentralized (*ArchD*). Figure 1 illustrates the differences between the two architectures. A centralized architecture contains multiple sensor radios and a central fusion center (FC) which processes sensor inputs to estimate the location of a target. A decentralized architecture has no trusted third party such as a permanent FC but instead relies on the radios to perform any requisite computations. For each localization round, a new "leader" is elected from the sensors through a consensus protocol, which we assume cannot be gamed by an adversary to substantially increase its

[3]This might be achieved by physically combing the area to discover sensor radios or acquiring organizational deployment records that contain location information.

likelihood of becoming the leader.[4] The leader is responsible for: (a) coordinating communication between all participant sensors in a round by establishing a hierarchical ordering and (b) outputting the final estimate.

3) Participants: Participants include sensor radios (in both architectures), and an additional fusion center (in *ArchC* only). Participants exist in one of three modes: (a) honest-but-curious, (b) malicious, and (c) compromised. Compromised participants are originally honest-but-curious and later taken over by a malicious adversary, rather than being under a malicious adversary's control from the onset. For the purposes of our analysis, we give the same treatment to malicious and compromised modes.

While we might, in general, assume participants will be honest-but-curious and share the common goal of producing a correct estimated location of the target, we cannot ignore the possibility of malicious adversaries (for example, in a hostile military setting). We describe below what each type of participant is able to do.

a) Honest-but-curious: Honest-but-curious participants faithfully carry out the localization protocol, so as to produce a correct estimate of the target transmitter's location, but may use any available information to attempt to break the location privacy of the other participants. In *ArchC*, the FC has access to the entire set of collected measurements and may use this to learn the physical locations of all contributing sensors in the localization round, as shown in Fig 2. In *ArchD*, individual sensors will attempt to expose the location of those sensors that provide them with input, as shown in Figure 3.

b) Malicious or Compromised: Malicious (or compromised) participants do not care about arriving at a correct estimate of the transmitter's location. Such participants may even actively seek to undermine the localization process. They may inject messages carrying crafted measurements or incorrect intermediate results; they may also drop legitimate messages. Specifically, this means malicious sensors may report fake measurements that sway intermediate and/or final outputs to leak something about other sensors' locations. Malicious sensors may drop legitimate messages and replace these with additional crafted inputs to further misdirect the process; multiple malicious/compromised sensors may collude for more productive misdirection. Alternatively, malicious sensors may engage in this behavior to prevent the target from being localized. A malicious FC may output a wrong estimate for the same purpose. For this reason, hostile military settings specifically require appropriate measures to protect against malicious adversaries.

This architecture also relies on the establishment of symmetric keys between pairs of sensors, generally coordinated by the leader.

[4]Under traditional leader election schemes, such as bully and ring election, an adversary may lie about its identity to win an election. Abraham et al. [24] demonstrate alternative techniques for leader election (informed by game theory) which are able to prescribe an equal probability of being elected to each agent.

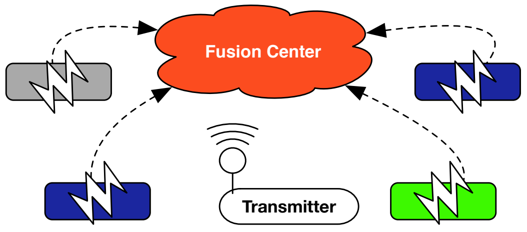

Fig. 2: *Centeralized Architecture:* the FC handles all sensors' inputs, making it a single point-of-faliure and a favorite target for hackers. Hence, malicious software running on an FC can break the location privacy of all sensors.

B. *Design Exploration*

Privacy-preserving localization can be achieved using multiple techniques such as garbled circuits, fully homomorphic encryption (FHE), secure multi-party computation (MPC), and enclave-based execution. The cryptographic techniques such as MPC and FHE provide a good setting for our task. Unfortunately, while such techniques have improved their performance by orders of magnitude in the past decade [25], they remain extremely resource-intensive. For instance, recent work demonstrated that garbled circuit-based techniques required an average of 300x increase in runtime for relatively small circuits [26]. Accordingly, from a performance perspective alone, such techniques remain impractical. However, existing work has shown that cryptography techniques incur a very high overhead, making them infeasible in real systems. Due to these constraints, we revert to enclave-based architectures.

There is a variety of emerging enclave-based architectures and approaches, but the most popular enclave types are Intel's SGX and ARM's TrustZone. Both are of interest for privacy-preserving spectrum sensing based on their properties. We choose SGX as a starting point to take advantage of its remote attestation feature, which we require in our setting. Remote attestation [27], as offered by SGX, makes it possible to verify the integrity of enclave code and execution environment remotely. Jin et al. [28] previously demonstrated how SGX could be used to perform remote attestation of remote terminals and Internet-of-Things (IoT) devices.

While Intel SGX is common in server systems, ARM TrustZone [13] is widely available on Android and embedded devices. A critically important feature missing from TrustZone, which we require in our setting, is remote attestation. To this end, existing work [29], [30] has tried to propose design schemes to bridge this gap. With remote attestation in place, a TrustZone-based solution could be more readily adopted by resource-constrained devices such as sensors, making use of the infrastructure already in place. and can be easily deployed on resource-constrained devices such as sensors. Based on our design exploration, we adopt both Intel SGX and ARM TrustZone to deploy our localization algorithm.

IV. Design

We consider the addition of enclaves to both centralized and decentralized architectures, introduced in Section III. We first present the general requirements and requirements specific to each architecture. Next, we consider the appropriateness

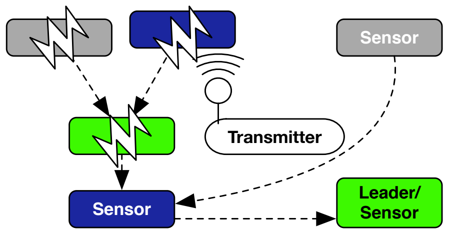

Fig. 3: *Distributed Architecture:* For localization, each round a new leader is elected on run time which acts similarly as a FC. This architecture removes the single point-of-failure, making it more robust compared to *ArchC*. However, a malicious node can still use the spectrum observations to identify thelocation of all downstream nodes.

of enclaves to answer each of these requirements. We then consider the arrangement of enclaves necessary to satisfy each requirement, and analyze the security of each arrangement to make sure *location privacy* is preserved, considering both honest-but-curious and malicious participants.

Despite differences between architectures in terms of message flow, the same underlying algorithm is applicable to both. Hence, there are several requirements that are applicable to both architectures. We describe the common needs:

N1: Preventing eavesdropping. Sensor inputs that are transmitted in the clear may be intercepted by an adversary. As a step during remote attestation, a symmetric key is provisioned for all future communication with the target enclave. Hence, the sensor can be confident that its inputs will not be compromised during transit to other enclaves.

N2: Confirming authenticity of the fusion result. A malicious FC (*ArchC*) or leader (*ArchD*) might disregard the true fusion result and disseminate an incorrect result. This could be mitigated by having the fusion code enclave, which is measured by each sensor during attestation, directly return the fusion result to sensors over the established secure channels.

N3: Defeating crafted/dishonest inputs. An enclave may be required of all sensors contributing to the localization when considering the presence of *malicious* sensors. Parts of the spectrum measurement could then be integrated with each sensor's enclave and verified by the enclaves via attestation to ensure only authentic spectrum measurements are contributed to the ongoing localization.

A. Centralized Architecture

The centralized architecture (*ArchC*) contains any number of sensors and a single fusion center (FC). Sensors provide input to the FC but do not interact with the other sensors nor handle their inputs.

The addition of enclaves could help meet multiple needs in this architecture. The needs are illustrated in Figure 4 alongside the enclave arrangement necessary for meeting each need. Besides the general needs, we describe each of *ArchC* specific needs below in more detail:

N1C: Establishing trust in the FC. By having each sensor check the measurement of the destination FC's enclave during remote attestation, the sensor can determine whether its provided input will be processed as expected and without

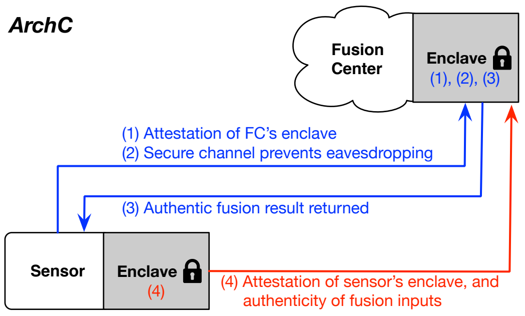
ArchC
Fusion Center
Enclave
(1), (2), (3)
(1) Attestation of FC's enclave
(2) Secure channel prevents eavesdropping
(3) Authentic fusion result returned
Sensor
Enclave
(4)
(4) Attestation of sensor's enclave, and authenticity of fusion inputs

Fig. 4: Various design requirements for *ArchC*. Contributing sensors attests FC's enclave before trusting with sensitive inputs. Inputs and fusion results are transmitted over a secure channel established during attestation. Sensors' enclaves can be attested by the FC to ensure authenticity of the fusion inputs it receives from participating sensors.

being leaked outside the enclave (e.g., to the FC owner).[5] Each sensor may then independently decide whether or not to trust the FC with its localization input during the current localization round.

To meet needs **N1C**, **N1**, and **N2**, it is sufficient to require an enclave to be hosted by the FC. Individual sensor compromise permits an adversary to learn the location of that specific sensor but nothing more. While an enclave is hosted by the FC, even privileged software on the device would be unable to look inside the FC to extract the sensitive inputs. These inputs would only be provided by each sensor to the FC if the attestation result is good.

To meet need **N3**, in which case the FC in turn wants some guarantees about the fusion inputs it is receiving from the sensors, it becomes necessary to have all sensors host their own enclaves. The assurance through attestation that a sensor will faithfully carry out the sense-and-forward procedure eliminates the need for other methods of detecting falsification of fusion inputs [31], [32].

B. Decentralized Architecture

The decentralized architecture (*ArchD*) is entirely dependent on the sensors; there is no separate third party acting as a facilitator. The fusion operation is distributed across the entire set of sensors. One of the sensors is elected as the leader through a consensus protocol. A new leader may be elected for each round if there is concern about the potential abuse of this role. The leader is responsible for facilitating the current round of localization and does not contribute its own input. The leader is restricted from participating in the localization round to prevent affording it any unintended advantage at breaking location privacy of the other sensors. Specifically, the leader will place sensors in a hierarchical ordering according to their Signal-to-Noise Ratio (SNR) values. A higher SNR value represents a signal of better quality that is paired with low levels of unwanted noise or interference. These values are received from all sensors participating in a round. SNR values

[5]No direct exfiltration can be performed. We consider side-channel vulnerabilities and microarchitectural attacks that can lead to covert exfiltration in Appendix B.

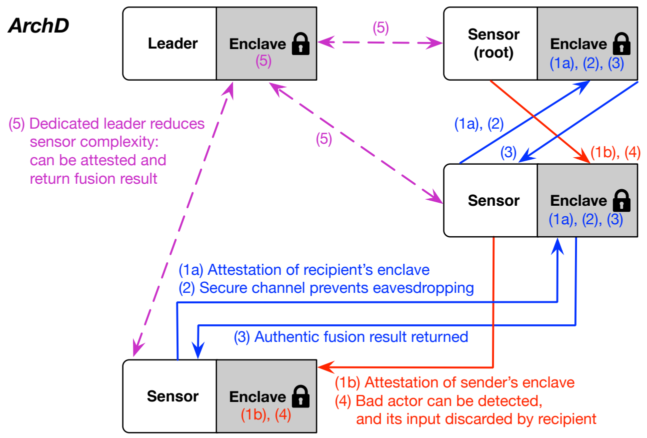
ArchD
Leader
Enclave
(5)
(5)
Sensor (root)
Enclave
(1a), (2), (3)
(5) Dedicated leader reduces sensor complexity: can be attested and return fusion result
(5)
(1a), (2)
(3)
(1b), (4)
Sensor
Enclave
(1a), (2), (3)
(1a) Attestation of recipient's enclave
(2) Secure channel prevents eavesdropping
(3) Authentic fusion result returned
Sensor
Enclave
(1b), (4)
(1b) Attestation of sender's enclave
(4) Bad actor can be detected, and its input discarded by recipient

Fig. 5: Various requirements for *ArchD*. Enclaves can be used to establish bidirectional trust among pairs of sensors that exchange information, prevent eavesdropping on the secure channel established during attestation, and return an authentic fusion result.

provide only limited information about a sensor's relative location to the target; hence, SNR processing may be done outside an enclave, under an honest-but-curious assumption.

The addition of enclaves could help meet a similar set of needs in this architecture as in *ArchC*. The needs are illustrated in Figure 5 alongside the enclave arrangement necessary for meeting each need. We describe *ArchD* specific needs below in more detail:

N1D: Establishing bidirectional trust in one another. By having each sensor check the measurement of the destination sensor's enclave during remote attestation, the sensor can determine whether its provided input will be processed as expected and without being leaked outside the enclave (e.g., to the owner of the enclave).[6] This forward-direction trust is reflected as (1a) in Figure 5. At the same time, if parts of the spectrum measurement could be integrated with each sensor's enclave and verified by the recipient during attestation, it is possible to ensure that crafted/dishonest inputs are not contributed to the ongoing localization round. This reverse-direction trust is reflected as (1b) in Figure 5.

N2D: Providing additional guarantees about the leader. Although we wrote in Section III-A that we assume the existence of a consensus protocol for the election of a leader that cannot be gamed by an adversary, such leader election schemes require rounds of communication that could be sidestepped with the introduction of a trusted enclave. A dedicated, central node hosting an enclave could rely on a hardware random number generator to appoint a leader. Alternatively, this dedicated enclave could itself be established as the dedicated leader for ordering sensors, thus reducing the complexity that each individual sensor must be provisioned with when they must support the potential to be elected as leader.

[6]No direct exfiltration can be performed. We consider side-channel vulnerabilities and microarchitectural attacks that can lead to covert exfiltration in Appendix B.

To fully meet need **N1D**, all sensors must host their own enclaves. This enables bidirectional attestation between each pair of parent and child sensors to ensure authenticity of the child's input and proper configuration of the parent's enclave for ingesting and processing inputs while preserving location privacy.

In the presence of only honest-but-curious sensors, it may be enough to partially meet need **N1D**, establishing a unidirectional trust by each source of its respective destination sensor one level up on the hierarchy. This also fulfills needs **N1** and **N3**. Non-leaf sensors will receive and process inputs from other sensors. To preserve the location privacy of their children, non-leaf sensors should only handle these inputs within an enclave. Leaf sensors do not require enclaves of their own, because they do not receive input (in that particular localization round), and they are expected to supply the correct particles to their parent sensors. For future rounds:

- If we fix certain sensors to always be leaves when chosen for a localization round, enclaves are not required of them. This may be potentially unfair to the designated leaves and open them up to association attacks[7] over multiple rounds.
- If we allow prior leaves to be placed higher in the hierarchy in a future ordering, they will process other sensors' inputs (a role that can only be fulfilled by hosting an enclave). To allow such role agility, all sensors must have enclave support, though their enclaves may remain inactive in certain rounds.

To meet need **N2**, all sensors are again required to host their own enclaves so as to validate the propagation of the fusion result back down the ordering hierarchy. The alternative is to either have each sensor establish a direct channel with the root, or to pass control to the leader to distribute the fusion result; either option would require additional attestations to establish the necessary trust relationships.

To meet need **N2D**, it is unavoidable to introduce a new node that will have a dedicated role, be it that of leader-appointer or leader, but this node will take no part in the localization workflow. This new node will host an enclave that can be measured by each active sensor via remote attestation. Although additional attestations become necessary as a result, the cost could potentially be amortized over many localization rounds as long as there is a way to verify the persistence of that enclave. At the same time, the communication cost associated with leader election schemes can be avoided.

Cost of Attestation To mitigate the cost of attestation in *ArchC*, participants might allow attestation to carry-over across rounds, as long as the enclave of the target has not changed. In *ArchD*, allowing attestation carry-over across rounds is not as effective. Unlike *ArchC*, where participants' roles are unchanging, sensors in *ArchD* may take different roles (leader, non-leaf, or leaf) in different rounds, assuming

[7]If a sensor always contributes an initial set of particles, a pattern may appear when analyzing the entire set of localization results over an extended period of time. This is of greater concern when the total number of participants is small.

Algorithm 1: Particle Filter-based Localization: Main function

```
Function main(treeHeight, servFraction, sensors, latPrecis, longPrecis, leafParticleCount, pleRange):
    for $treeLevel$ = $treeHeight$; $treeLevel$ ¿ 0; $treeLevel$– do
        for $sensor \in sensors[treeLevel]$ do
            if $treeLevel$ == $treeHeight$ then
                $sensor.particles$ = leafInit($leafParticleCount$, $sensor.position$, $pleRange$)
            end
            else
                $sensor.particles$ = receiveParticles($sensor.child1$, $sensor.child2$)
            end
            $sensor.particles$ = updateParticles($sensor.particles$, $servFraction[treeLevel]$, $sensor.RSS$)
        end
    end
    $grid$ = $\langle maxLat, minLat, maxLong, minLong\rangle$ from $root.particles$
    while ($maxLat - minLat$ ¿ $latPrecis$) and ($maxLong - minLong$ ¿ $longPrecis$) do
        $rootParticles$ = partitionParticles($rootParticles$, $grid$)
        recalculate $grid$ from $root.particles$
    end
    $estimate$ = ⟨latitude, longitude⟩ at center of $rootParticles$
    return $estimate$
```

Algorithm 2: Particle Filter-based Localization: Sub-functions

```
Function generateParticle(pos, pleRange):
    init $particle$
    select $particle.lat$ and $particle.lon$ uniformly at random from within circle centered at $pos$
    select uniformly at random from $pleRange$ to set $particle.ple$
    $particle.weight$ = 1
    return $particle$
Function leafInit(count, pos, pleRange):
    init $initParticles$
    for $index$ = 0; $index$ ¡ $count$; $index$++ do
        $initParticles$.append(generateParticle($pos$, $pleRange$))
    end
    return $initParticles$
Function receiveParticles(child1, child2):
    return $child1.particles \cup child2.particles$
Function updateParticles(inputParticles, frac, RSS):
    init $outputParticles$
    for $particle \in inputParticles$ do
        update $particle.weight$ using $RSS$
    end
    $servingParticles$ = ($inputParticles.count * frac$) particles with greatest $particle.weight$
    for $particle \in inputParticles$ do
        $outputParticles$.append($particle$)
        if $particle \in servingParticles$ then
            $newParticle$ = generateParticle($particle$)
            $outputParticles$.append($newParticle$)
        end
    end
    return $outputParticles$
Function partitionParticles(rootParticles, grid):
    split $grid$ into $regions$ = $\{gNW, gNE, gSW, gSE\}$
    init count = {0, 0, 0, 0}
    for $particle \in rootParticles$ do
        for $region \in regions$ do
            if $particle.lat$ and $particle.lon$ within $region$ then
                increment $count[region.index]$
                $regions[region.index]$.append($particle$)
            end
        end
    end
    return $regions$[index of max element in $count$]
```

they are selected at all. Even if enclave operation could be configured to match the sensor's role in each round, the ordering of sensors may change dramatically between rounds. Since trust is neither symmetric nor transitive, any new parent/child relation demands new attestation. One way to work around this might be to designate, at random, some subset of the sensors to attest the enclaves of all other participants in each round. since both sensors and FC have a defined, unchanging role in every round of localization. New attestations would be required only by/of sensors newly selected to participate in the next round. which may be all of the sensors in the worst case (this sensor-turnover ratio could be bounded by a parameter).

V. Full Localization Algorithm

Localization begins at the bottom level of the tree, at the leaves. and proceeds upward toward the root. Each leaf sensor initializes a predetermined number of particles (Alg. 2, lines 1–12). Particles are quartets of form $\langle latitude, longitude, PLE, weight\rangle$. Each particle has its weight updated according to the leaf's RSS (Alg. 2, lines 15–28). As part of the update process, some portion of the particles are designated as serving particles (the size of a sensor's serving particle set may be dependent on its position in the tree), in which case they will be used to generate additional new particles.

At all levels of the tree above the leaves (including at the root), sensors receive particles from their two children (Alg. 2, lines 13–14), which they combine. Each parent sensor updates the weight of each particle in its combined particle set according to its own RSS and performs resampling with any designated serving particles.

Once the initial loop completes, it is the root's responsibility to use the final particle set to perform the final estimation of the target transmitter's position. The root does this by first determining the minimum and maximum latitude and longitude values among all particles, using this to establish a two-dimensional grid (Alg. 1, line 13). While the difference between min and max latitude and between min and max longitude are greater than a predetermined goal precision, the root partitions the set of particles into quadrants: NW, NE, SW, and SE (Alg. 2, line 30). The root places each of its particles into the respective quadrant. Once all particles are placed, the root selects the quadrant with the highest particle density to be the refined particle set (Alg. 2, lines 29–40). This process is recursively repeated until the goal precision is reached, at which time the root estimates the transmitter to be located at the center of the grid containing the remaining particles (Alg. 1, line 18).

VI. Implementation

We implement the particle filter-based localization algorithm presented in Section A and use it to reason about both the centralized and decentralized architectures. We first describe the partitioning of the simulator code done to con-

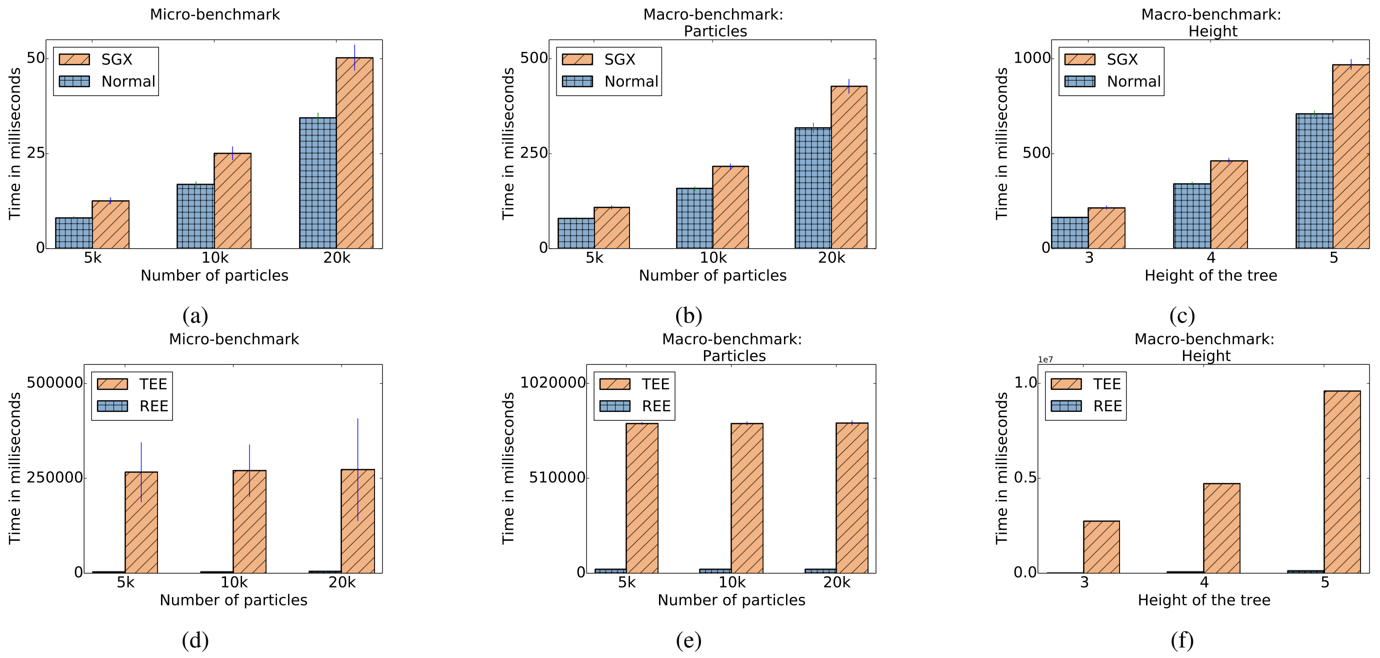

Fig. 6: Results are shown for performance testing: (a) and (d) show the micro-benchmark on (`process_measurement`) function with varying particle counts, averaged over 100 runs for SGX and TrustZone respectively. (b) and (e) show the macro-benchmark with different particle counts for SGX and TrustZone respectively. (c) and (f) show the macro-benchmark with different heights for SGX and TrustZone respectively. All macro-benchmarks are averaged over 20 runs.

struct an enclave component, followed by the implementation challenges specific to each choice of TEE.

A. Enclave

For a cross-platform application, an obvious choice could be using a hardware-agnostic SDK, such as OpenEnclave [33]. However, we found the project is not mature and seems to lack support for various platforms. Instead, we used the Intel SGX SDK for Intel and OP-TEE[8] for ARM platforms. As both of these projects provide a call interface to invoke the enclave, we choose the `process_measurement` function as the main focus of enclave porting, as it provides the simulation logic for the processing of input particles by each sensor. We also placed several helper functions, such as normal distribution sampler, latitude/longitude and bearing/distance coordinates converter, etc., in the enclave to aid the measurements process. The particles supplied by the currently simulated sensor's children are passed into the function as a parameter to simulate the actual sending of child inputs, though pre-combined, across the enclave boundary into the parent's enclave.

SGX-specific implementation details Intel SGX SDK lacks support for C++'s *random*, used to sample from a normal distribution. To work around this, we implement a custom function for sampling from a normal distribution using *sgx_read_rand*. Intel SGX SDK does not support C++ vectors across the enclave boundary. To this end, we flatten vectors into arrays while passing data to enclaves and vice versa.

TrustZone-specific implementation details: Similar to the SGX SDK, the programming environment in the TA is also restricted. We implement our own function for sampling from a normal distribution using `TEE_GenerateRandom`. As OP-TEE does not provide a math library, we statically link the `fdlibm`[9] library against our application. Due to limited secure memory, OP-TEE restricts the TA memory size. However, for our application we modified the OP-TEE core to increase the maximum available memory for our TA.

VII. Evaluation

Platform We conduct the evaluation of our SGX implementation on an HP machine with an Intel Skylake quad-core CPU supporting SGXv1 and 8 GB of memory. The machine is equipped with Ubuntu v18.04 and Intel SGX SDK v2.4. For TrustZone, we used the Hisilicon HiKey 620 with an ARM Cortex A53 octa-core CPU, 2GB memory, and Hisilicon TrustedCore as the TEE. Our TA is built for OP-TEE v3.13. In our experiments, we feed real spectral measurements to our simulation. We take as many RSS measurements (with the same fixed transmitter as the target) as there are sensors; each sensor in our simulation is assigned one of these measurements. When taking these measurements, we also record the GPS coordinates of the collection point; each sensor assumes as its own the location coordinates corresponding to its prescribed RSS measurement. The evaluation results are available in Figure 6. We discuss the results in the following subsections.

[8]`https://www.op-tee.org`

[9]Freely Distributable LIBM, available at `https://www.netlib.org/fdlibm/`.

Micro-benchmark For micro-benchmarking, we vary the number of particles initially generated by each leaf sensor (5,000, 10,000, and 20,000) and measure the average execution time of `process_measurement` function over 100 executions. In all cases, we observe that the enclave version of the simulation is slower than the normal C++ simulation. We observe that doubling particle count from 10,000 to 20,000: the average execution time increase from 8.27 ms to 15.9 ms (1.92x increase) for SGX (6a). Similarly for TrustZone (6d), the average execution time increase from 174.45 ms to 272.8 ms (1.56x increase). For TrustZone, the majority of the high overhead comes from the session setup with the trusted world. *Macro-benchmark* We perform macro-benchmarking to measure the performance overhead across the entire simulation(from initial leaf particle generation to final transmitter location estimation). Our simulation does not capture actual networking overhead incurred from communication between enclaves, but we make a best effort to simulate the complexity of passing inputs into and outputs out of the enclave by each participating sensor. In this set of experiments, we observe the effects of varying the height of the tree (which determines the number of sensors that participate in a given localization round) and the initial leaf-generated particle count. Results are averaged over 20 complete simulation runs.

Similar to microbenchmarks, we vary the particle count from 5000 to 20,000. When varying the number of particles, for SGX (6b), we observe that simulation time roughly doubles as the number of particles increases; this behavior is the same for both normal and SGX variants. Average overhead is increased from 29.4 ms to 109.6 ms, when varying particles count from 10,000 to 20,000 particles. In contrast, for the same configuration, in TrustZone (6e) overhead increases from 804.2 ms to 807.6 ms (1.01x increase). The low increase in the overhead is because the overhead of the call from the non-secure world to the secure world (also known as Secuure Monitor Call (SMC)) overshadows the overhead due to particle count increase.

When varying the tree height (Figure 6(c)), we observe that simulation time roughly doubles as the height of the tree increases by 1, for both SGX and TrustZone. This is expected since increasing tree height effectively doubles the number of participating sensors. For SGX (6c), the average overhead is increased from 53.2 ms for tree height of 3; to 259.1 ms for tree height of 5 (7.9x increase). For TrustZone (6f) the simulation time is increased from 2.7 seconds on average with tree height of 3 to 9.5 seconds on average with a tree height of 5 (3.51x increase). Here we see that the percentage overhead increases alongside the increase in tree height, but only by a small amount, while the raw performance hit remains within expectations.

In all cases, we observe a higher execution time compared to native execution. However, the overall overhead is considerably lower compared to cryptographic techniques, making it feasible for real systems. With TrustZone the higher overhead can be mitigated using various techniques such as pseudo TAs, persistent TA sessions, and Scalable Vector Extension v2 (SVE2). However, our evaluation suggests that the particle filters with TrustZone can benefit from using an SGX node in the cluster via the cloud.

VIII. Conclusion

The naive sharing of information in cooperative spectrum sensing allows a compromised participant to potentially uncover the sensitive locations of all sensors performing such monitoring. In this paper, we design and implement a secure particle filter method for localization based on enclaves. In effect, we illustrate how we can dramatically improve the protections for participants in collaborative wireless applications. To our best knowledge, this is the first work demonstrating privacy-preserving attestable localization in a multi-party environment with reasonable overhead.

IX. Acknowledgments

We thank the anonymous reviewers for their valuable comments. This work was supported in part by NSF under Grants NSF CNS-1642973 and NSF CNS-1815883. This material is also based on research sponsored by AFRL under contract number AFRL FA9550-19-1-0169. Any opinions, findings, and conclusions, or recommendations expressed in this material are those of the authors and do not necessarily reflect the views of these sponsors.

References

[1] K. Kim, C. H. Kim, J. J. Rhee, X. Yu, H. Chen, D. J. Tian, and B. Lee, "Vessels: Efficient and scalable deep learning prediction on trusted processors," in *Proceedings of the 11th ACM Symposium on Cloud Computing*, ser. SoCC '20. New York, NY, USA: Association for Computing Machinery, 2020.

[2] F. Schuster, M. Costa, C. Fournet, C. Gkantsidis, M. Peinado, G. Mainar-Ruiz, and M. Russinovich, "Vc3: Trustworthy data analytics in the cloud using sgx," in *2015 IEEE Symposium on Security and Privacy*, 2015, pp. 38–54.

[3] F. Tramer and D. Boneh, "Slalom: Fast, verifiable and private execution of neural networks in trusted hardware," in *International Conference on Learning Representations*, 2019.

[4] C. Priebe, K. Vaswani, and M. Costa, "Enclavedb: A secure database using sgx," in *2018 IEEE Symposium on Security and Privacy (SP)*, 2018, pp. 264–278.

[5] P. Mishra, R. Poddar, J. Chen, A. Chiesa, and R. A. Popa, "Oblix: An efficient oblivious search index," in *2018 IEEE Symposium on Security and Privacy (SP)*, 2018, pp. 279–296.

[6] R. Krahn, B. Trach, A. Vahldiek-Oberwagner, T. Knauth, P. Bhatotia, and C. Fetzer, "esos: Policy enhanced secure object store," in *Proceedings of the Thirteenth EuroSys Conference, EuroSys 2018, Porto, Portugal, April 23-26, 2018*, ser. EuroSys '18. Association for Computing Machinery, 2018.

[7] T. Kim, J. Park, J. Woo, S. Jeon, and J. Huh, "Shieldstore: Shielded in-memory key-value storage with sgx," in *Proceedings of the Fourteenth EuroSys Conference 2019*, ser. EuroSys '19. New York, NY, USA: Association for Computing Machinery, 2019.

[8] S. Eskandarian and M. Zaharia, "Oblidb: Oblivious query processing for secure databases," *Proc. VLDB Endow.*, vol. 13, no. 2, p. 169–183, Oct. 2019.

[9] T. Alhmiedat, A. A. Taleb, and M. Bsoul, "A Study on Threats Detection and Tracking Systems for Military Applications using WSNs," *International Journal of Computer Applications*, vol. 40, no. 15, pp. 12–18, Feb. 2012.

[10] J. Ma, G. Y. Li, and B. H. Juang, "Signal Processing in Cognitive Radio," *Proceedings of the IEEE*, vol. 97, no. 5, pp. 805–823, May 2009.

[11] R. N. Handcock, D. L. Swain, G. J. Bishop-Hurley, K. P. Patison, T. Wark, P. Valencia, P. Corke, and C. J. O'Neill, "Monitoring Animal Behaviour and Environmental Interactions Using Wireless Sensor Networks, GPS Collars and Satellite Remote Sensing," *Sensors*, vol. 9, no. 5, pp. 3586–3603, May 2009.

[12] Intel Corporation, “Intel Software Guard Extensions for Linux OS,” https://01.org/intel-softwareguard-eXtensions, 2018.

[13] ARM, Ltd., “Building a Secure System using TrustZone Technology,” ARM, Ltd., Tech. Rep., 2009.

[14] T. Ward, J. I. Choi, K. Butler, J. M. Shea, P. Traynor, and T. Wong, “Privacy Preserving Localization Using a Distributed Particle Filtering Protocol,” in *IEEE MILCOM*, 2017.

[15] S. S. Dias and M. G. S. Bruno, “Cooperative Target Tracking Using Decentralized Particle Filtering and RSS Sensors,” *IEEE Transactions on Signal Processing*, vol. 61, no. 14, Jul. 2013.

[16] C. Morelli, M. Nicoli, V. Rampa, U. Spagnolini, and C. Alippi, “Particle Filters for RSS-Based Localization in Wireless Sensor Networks: An Experimental Study,” in *IEEE ICASSP*, 2006.

[17] T. Stoyanova, F. Kerasiotis, C. Antonopoulos, and G. Papadopoulos, “Rss-based localization for wireless sensor networks in practice,” in *Comm. Sys., Networks, and Digital Sign (CSNDSP)*, 2014.

[18] S. Tomic, M. Beko, R. Dinis, and J. P. Gomez, “Target Tracking with Sensor Navigation Using Coupled RSS and AoA Measurements,” *Sensors (Basel)*, vol. 17, no. 11, p. 2690, 2017.

[19] Z. Yang, Z. Zhou, and Y. Liu, “From RSSI to CSI: Indoor Localization via Channel Response,” *ACM CSUR*, vol. 46, no. 2, Nov. 2013.

[20] G. Wang and K. Yang, “A New Approach to Sensor Node Localization Using RSS Measurements in Wireless Sensor Networks,” *IEEE Transactions on Wireless Communications*, vol. 10, no. 5, May 2011.

[21] M. B. Jamâa, A. Koubâa, and Y. Kayani, “EasyLoc: RSS-based Localization Made Easy,” in *RoboSense*, 2012.

[22] J. Shirahama and T. Ohtsuki, “RSS-Based Localization in Environments with Different Path Loss Exponent for Each Link,” in *IEEE VTC*, 2008.

[23] A. Robertson, S. Kompella, J. Molnar, F. Fu, M. Dillon, and D. Perkins, “Distributed Transmitter Localization by Power Difference of Arrival (PDOA) on a Network of GNU Radio Sensors,” Naval Research Laboratory, Tech. Rep., Mar. 2015.

[24] I. Abraham, D. Dolev, and J. Y. Halpern, “Distributed Protocols for Leader Election: A Game-Theoretic Perspective,” in *DISC*, 2013.

[25] B. Mood, D. Gupta, H. Carter, K. Butler, and P. Traynor, “Frigate: A Validated, Extensible, and Efficient Compiler and Interpreter for Secure Computation,” in *IEEE Euro S&P*, 2016.

[26] R. Bahmani, M. Barbosa, F. Brasser, B. Portela, A. Reza-Sadeghi, G. Scerri, and B. Warinschi, “Secure Multiparty Computation from SGX,” in *FC*, 2017.

[27] I. Anati, S. Gueron, S. Johnson, and V. Scarlata, “Innovative Technology for CPU Based Attestation and Sealing,” in *HASP*, 2013.

[28] J. Wang, Y. Zhang, and Y. Jin, “Enabling Security-Enhanced Attestation With Intel SGX for Remote Terminal and IoT,” *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems (TCAD)*, vol. 37, no. 1, 2018.

[29] Z. Wang, Y. Zhuang, and Z. Yan, “Tz-mras: A remote attestation scheme for the mobile terminal based on arm trustzone,” *Security and Communication Networks*, vol. 2020, 2020.

[30] Z. Ling, H. Yan, X. Shao, J. Luo, Y. Xu, B. Pearson, and X. Fu, “Secure boot, trusted boot and remote attestation for arm trustzone-based iot nodes,” *Journal of Systems Architecture*, vol. 119, p. 102240, 2021.

[31] X. Luo, “Secure Cooperative Spectrum Sensing Strategy Based on Reputation Mechanism for Cognitive Wireless Sensor Networks,” *IEEE Access*, vol. 8, pp. 131 361–131 369, 2020.

[32] Z. Luo, S. Zhao, Z. Lu, J. Xu, and Y. E. Sagduyu, “When Attackers Meet AI: Learning-empowered Attacks in Cooperative Spectrum Sensing,” *arXiv preprint arXiv:1905.01430*, 2020.

[33] “openenclave/openenclave: Sdk for developing enclaves,” https://github.com/openenclave/openenclave, (Accessed on 06/10/2021).

[34] S. Johnson, https://software.intel.com/en-us/articles/intel-sgx-and-side-channels, Feb. 2018.

[35] Y. Zhang, M. Zhao, T. Li, and H. Han, “Survey of Attacks and Defenses against SGX,” in *Proceedings of the IEEE 5th Information Technology and Mechatronics Engineering Conference (ITOEC)*, 2020.

[36] A. Nilsson, P. N. Bideh, and J. Brorsson, “A Survey of Published Attacks on Intel SGX,” *arXiv preprint arXiv:2006.13598*, 2020.

[37] Y. Xu, W. Cui, and M. Peinado, “Controlled-Channel Attacks: Deterministic Side Channels for Untrusted Operating Systems,” in *Proceedings of the IEEE Symposium on Security and Privacy (IEEE S&P)*, 2015.

[38] M. Schwarz, S. Weiser, D. Gruss, C. Maurice, and S. Mangard, “Malware Guard Extension: Using SGX to Conceal Cache Attacks,” in *DIMVA*, 2017.

[39] F. Brasser, U. Müller, A. Dmit rienko, K. Kostiainen, S. Capkun, and A.-R. Sadeghi, “Software Grand Exposure: SGX Cache Attacks Are Practical,” in *WOOT*, 2017.

[40] W. Wang, G. Chen, X. Pan, Y. Zhang, X. Wang, V. Bindschaedler, H. Tang, and C. A. Gunter, “Leaky Cauldron on the Dark Land: Understanding Memory Side-Channel Hazards in SGX,” in *ACM CCS*, 2017.

[41] J. Wichelmann, A. Moghimi, T. Eisenbarth, and B. Sunar, “MicroWalk: A Framework for Finding Side Channels in Binaries,” in *ACSAC*, 2018.

[42] M. Lipp, M. Schwarz, D. Gruss, T. Prescher, W. Haas, A. Fogh, J. Horn, S. Mangard, P. Kocher, D. Genkin, Y. Yarom, and M. Hamburg, “Meltdown: Reading Kernel Memory from User Space,” in *Proceedings of the 27th USENIX Security Symposium*, 2018.

[43] P. Kocher, J. Horn, A. Fogh, D. Genkin, D. Gruss, W. Haas, M. Hamburg, M. Lipp, S. Mangard, T. Prescher, M. Schwarz, and Y. Yarom, “Spectre: Exploiting Speculative Execution,” in *Proceedings of the 40th IEEE Symposium on Security and Privacy (IEEE S&P)*, 2019.

[44] J. Van Bulck, D. Moghimi, M. Schwarz, M. Lipp, M. Minkin, D. Genkin, Y. Yuval, B. Sunar, D. Gruss, and F. Piessens, “LVI: Hijacking Transient Execution through Microarchitectural Load Value Injection,” To appear at the IEEE Symposium on Security and Privacy (IEEE S&P), 2020.

[45] J. Van Bulck, M. Minkin, O. Weisse, D. Genkin, B. Kasikci, F. Piessens, M. Silberstein, T. F. Wenisch, Y. Yarom, and R. Strackx, “Foreshadow: Extracting the Keys to the Intel SGX Kingdom with Transient Out-of-Order Execution,” in *Proceedings of the 27th USENIX Security Symposium*, 2018.

[46] M. Lipp, A. Kogler, D. Oswald, M. Schwarz, C. Easdon, C. Canella, and D. Gruss, “PLATYPUS: Software-based Power Side-Channel Attacks on x86,” in *IEEE Symposium on Security and Privacy (SP)*, 2020.

[47] K. Murdock, D. Oswald, F. D. Garcia, J. Van Bulck, D. Gruss, and F. Piessens, “Plundervolt: Software-based Fault Injection Attacks against Intel SGX,” in *Proceedings of the 41st IEEE Symposium on Security and Privacy (IEEE S&P)*, 2020.

[48] J. Beekman, “On the recent side-channel attacks on Intel SGX,” https://jbeekman.nl/blog/2017/03/sgx-side-channel-attacks/ Accessed, May 2017.

[49] M.-W. Shih, S. Lee, T. Kim, and M. Peinado, “T-SGX: Eradicating Controlled-Channel Attacks Against Enclave Programs,” in *NDSS*, 2017.

[50] J. Seo, B. Lee, S. Kim, M.-W. Shih, I. Shin, D. Han, and T. Kim, “SGX-Shield: Enabling Address Space Layout Randomization for SGX Programs,” in *NDSS*, 2017.

[51] S. Matetic, M. Ahmed, K. Kostiainen, A. Dhar, D. Sommer, A. Gervais, A. Juels, and S. Capkun, “ROTE: Rollback Protection for Trusted Execution,” in *Proceedings of the 26th USENIX Security Symposium*, 2017.

[52] V. Costan, I. Lebedev, and S. Devadas, “Sanctum: Minimal Hardware Extensions for Strong Software Isolation,” in *USENIX Security Symposium*, 2016.

[53] Keystone Enclave, “Keystone: An Open Framework for Architecting Trusted Execution Environments,” https://keystone-enclave.org, 2018.

Appendix A
Simulator Implementation and Tunable Parameters

a) Tunable parameters: The simulator accepts the following parameters, inheriting the default values selected by Ward et al. [14].

- *Tree height* determines the number of sensors that are accepted for a certain round. With height h, a binary tree contains $2^h - 1$ nodes. For example, a binary tree height of 4 permits 15 sensors to participate. Out of a larger set of available sensors, a subset of sensors is chosen to populate the tree according to the sensors' Signal-to-Noise Ratio (SNR).
- *Particle count* sets the number of particles generated uniformly at random by leaf sensors in the tree hierarchy.
- *Seed particle fraction* is the fraction of particles to keep when resampling (0.1 by default).

- *Resampling fraction* is the ratio of new samples that will be drawn to the number of input samples (1 by default).
- *Transmitter location* is the transmitter's latitude, longitude pair, against which the computed estimate can be compared.
- *Max particle distance* is the maximum distance away from a leaf that a particle it generates may be (75 km by default).
- *Path Loss Exponent (PLE)* is uniformly selected between 3 and 4.5 for particles generated by leaf sensors. Actual PLE varies depending on the localization target; we calculate the actual PLE for our simulation to be 3.2.

b) Simulation steps: The simulator begins by taking in a vector of PLEs (chosen uniformly from the PLE selection range to approximate the actual PLE) and a vector of signal power measurements (indexed by receiver and frequency band). A tree hierarchy is established, with sensors sorted by their signal power measurements (sensors with higher SNR occupy higher levels of the tree). The following steps correspond to those first presented in Section A.

1) *Particle generation at leaf sensors.* We use a loop to generate particles uniformly at random; we loop as many times as there are simulated leaves. A particle is implemented as a vector of four floats, and a sensor's particle set is implemented as a vector of particles. Each leaf's vector of particles is itself stored in a vector indexed by leaf ID.
2) *Particle processing.* We set aside a dedicated function for resampling particles (`process_measurement`); the function gets called once for each sensor that populates the tree. On each invocation, the function has access to the location and RSS measurement of one particular sensor. The function *(i)* calculates expected signal value when not considering noise; *(ii)* calculates the probability density function (PDF) using the expected signal value and the sensor's RSS measurement; and *(iii)* runs the particle filter, updating the weights of each particle while using any "serving" particles to generate new ones. After completing these three tasks, the function returns the set of newly resampled particles to the main simulation function, which stores the returned set of particles in a vector.
3) *Merging particle sets.* Upon returning, and unless the current sensor is the root, pairs of particle sets are combined in preparation for the simulation of computations at the next level up in the tree (in a binary tree arrangement, a parent sensor receives particles from its two children as input). The loop will repeat and again call `process_measurement`.
4) *Recursive particle grid partitioning.* Output is prepared by first setting up 2-D grid boundaries using the *max particle distance* and the locations of the sensors furthest out. Particles from the final particle set produced by the root are overlaid onto this grid, and the region with the most particles is selected. Grid partitioning is recursive and continues in a loop until the desired precision is reached, at which point, the center of the final region is ascertained as the transmitter's location.

APPENDIX B
ATTACKS AGAINST ENCLAVES

SGX has proven to be a controversial mechanism for enforcing protection, particularly since Intel specifically made the decision that side channels were outside of the threat model and that developers were responsible for preventing side channel attacks [34]. As a result of this decision, a number of attacks have been demonstrated against SGX. We provide a sampling of the major attacks against SGX below. For a more complete characterization of all the attacks demonstrated against SGX and the corresponding mitigations, refer to work by Zhang et al. [35] and Nilsson et al. [36].

Controlled-channel attacks [37] use memory access patterns to exfiltrate sensitive information from secure enclaves. Cache-based side-channel attacks [38], [39] have also been effectively deployed against SGX. Meanwhile, memory side channel hazards were discovered by Wang et al. [40] that affect system elements ranging from TLBs to DRAM modules. Other side-channel vulnerabilities [41] are also found within the Integrated Performance Primitives (IPP) cryptographic library used by Intel SGX SDK. More recently, microarchitectural attacks have been demonstrated to work on SGX enclaves, notably the high-profile Meltdown [42], Spectre [43], and LVI [44] attacks, while Foreshadow [45] attacks extract the attestation key from enclaves thus breaking SGX remote attestation. The novel PLATYPUS [46] attack demonstrated the ability of an attacker to remotely collect power consumption information via Intel's Running Average Power Limit (RAPL) interface, where previous power analysis attacks required physical access. Alternatively, active power-based attacks such as Plundervolt [47] can be remotely mounted against SGX enclaves to inject faults and reconstruct cryptographic keys. The net result of this critical examination of SGX is that side channels appear to represent a substantial attack surface.

However, defenses against side channel attacks have been developed by the academic community and by Intel, whose recent version of the SGX SDK introduces a number of countermeasures [48]. Intel continues to work closely with security researchers to release microcode patches and updates to the Intel SGX SDK as new vulnerabilities surface. Within the academic literature, proposals have included T-SGX [49], which leverages Intel TSX instructions to hide the enclave page fault from the untrusted operating system, defending against controlled-channel attacks. SGX-Shield [50] enables ASLR for enclave memory space, defending the enclave code against exploitations, while ROTE [51] leverages multiple SGX machines in a distributed environment to provide a monotonic counter and prevent rollback attacks against enclaves. Existing work has proposed architecture-level mitigations for cache-based attacks. Sanctum [52] proposes a RISC V-based ISA that mitigates side channels while enforcing enclave guarantees; a RISC V-based, open-source alternative to Sanctum is provided by Keystone [53].