Toward Easier Development of Privacy-Preserving Mobile Crowdsensing Applications

Jeffrey Murray
Computing and Software Systems
University of Washington Bothell
jeffmur@uw.edu

Brent Lagesse
Computing and Software Systems
University of Washington Bothell
lagesse@uw.edu

***Abstract*—Fully Homomorphic Encryption (FHE) schemes allow computations over encrypted data without access to the decryption key. This technique can be a valuable tool for building privacy into crowdsensing systems; however, many existing FHE implementations, such as Microsoft's SEAL, are difficult to implement into mobile applications. This paper presents a natively compiled Dart plugin that abstracts the underlying C/C++ SEAL library. The FHE Library plugin enables developers to access SEAL's full functionality within other Dart plugins and Flutter applications and is extensible to other encryption libraries.**

To evaluate the versatility of the plugin, we develop a Dart plugin to calculate several distance measures between two sets of encrypted inputs and we develop a Flutter application called GhostPeerShare. The Distance Measure plugin implements Kullback-Leibler Divergence, Bhattacharyya Coefficient, and Cramer Distance. GhostPeerShare demonstrates the use of a plugin by re-implementing Proof of Presence Share (PopShare), a mobile application that privately identifies similar videos recorded by users, as a Flutter application. Through these applications, we demonstrate that performance is similar to native applications and that utilizing FHE is more accessible to researchers developing crowdsensing applications.

***Index Terms*—Privacy, Fully Homomorphic Encryption, Mobile Crowdsensing**

I. Introduction

A significant challenge in mobile crowdsensing is the privacy of data collected by the participants. As this data often contains personally identifiable information or is highly correlated to personally identifiable information, it needs to be protected. For example, spatio-temporal data can reveal information about commonly visited locations of a user such as their home, school, or work. Most crowdsensing systems make assumptions about the trustworthiness of centralized servers or other users in the system; however, that may not always be true. To address this, some systems will use homomorphic encryption to perform operations on encrypted data so that its content is not revealed to any third party [1]–[5]. In this work, we are particularly focused on fully homomorphic encryption (FHE) as it gives the developer the most flexibility in their application development, but this work could also be applied to partially homomorphic cryptosystems. Existing FHE libraries are primarily focused on desktop and cloud computing rather than on mobile computing, thus making implementation difficult for many people who would benefit from FHE in their crowdsensing mobile apps. In this work we have developed an abstraction and implementation for FHE to make it more accessible to mobile application developers through Dart and Flutter.

This project contributes a modular Dart plugin[1][2] that abstracts and implements Microsoft Simple Encrypted Arithmetic Library (SEAL) [6]. The abstraction design enables the integration and versioning of additional backend libraries. With the use of CMake configurations, each C library can be synchronized and recompiled as new versions become available, simplifying the update process. Automating these compilations reduces maintenance overhead, ensuring that the libraries remain up-to-date with minimal manual intervention.

To assess the library's performance and accuracy, we re-implement the Proof of Presence Share (PoP-Share) [7] methodology[3][4] which was originally developed as a Java application along with native C++ designed to run on Android. PoP-Share is a mobile crowdsensing application that allows users to record video of events and then aggregate these videos from multiple users at the even in a privacy-preserving manner that only allows people who contribute videos of an event to view the videos of others.

II. Background and Related Work

Microsoft's Simple Encryption Arithmetic Library (SEAL) provides a C++ library for efficient homomorphic encryption, which made it accessible to the open-source community of security researchers and developers. SEAL was among the first of many libraries within the homomorphic encryption community. Other libraries included the Homomorphic Encryption Library (HElib) [8] and Privacy-Preserving Approximate Secure Computation for Encrypted Data (PALISADE) [9]. Collectively, the open-source community aimed to combine their efforts into the Open-Source Fully Homomorphic Encryption Library (OpenFHE). A flexible, community-driven project that was initially released in 2022. Compared to SEAL, OpenFHE offers support for additional encryption schemes for evaluating Boolean circuits and arbitrary functions over larger plaintext spaces using lookup tables. However, OpenFHE is a relatively new framework with a growing community but requires further

[1]https://pub.dev/packages/fhel
[2]https://github.com/jeffmur/fhel
[3]https://github.com/jeffmur/fhe-video-similarity
[4]https://github.com/jeffmur/fhe-similarity-score

adoption in the research community to assess its proficiency. Currently, developers and security researchers are limited in creating FHE applications within the programming language constraints of C or C++, as a result, the mobile crowdsensing community must overcome significant hurdles to access this functionality.

The Pyfhel library [10] provides a native Python interface to SEAL cryptosystems, abstracting core SEAL functionalities into a more generic interface. While Pyfhel supports cross-platform compatibility on Windows, Linux, and macOS, it currently lacks support for mobile platforms, e.g. Android and iOS. This project adopts a similar approach, creating a modular interface that supports multiple backend libraries.

Flutter, a state-of-the-art software development kit by Google captured 46% of the cross-platform application market [11]. This framework is built on top of Dart, a programming language recognized for its high performance and robust plugin system. Together, they provide compatibility with all major operating systems, specifically Android, Linux, macOS, iOS, and Windows. As a beneficiary of this work, the growing Flutter community may integrate Fully Homomorphic Encryption (FHE) into existing or new applications. This framework enables security researchers and developers to rapidly develop applications without the required prerequisite knowledge of the underlying C FHE libraries.

III. System Design

A. Fully Homomorphic Encryption Library

This implementation adheres to the core design principle of modularity, remaining agnostic to the underlying C++ library. Instead of directly invoking methods within the plugin, the plugin uses the Bridge and Adapter structural design patterns inspired by the work of Alberto Ibarrondo and Alexander Viand [10]. The Bridge pattern decouples the high-level Dart API from the specific C++ implementation by defining an abstract interface. The Adapter pattern connects the Dart interface to the C++ implementation through a C-based intermediary, translating Dart requests into operations understood by the native library. This approach enables compatibility with multiple Fully Homomorphic Encryption (FHE) backends without modifying the Dart API.

Figure 1 demonstrates a simplified example of how Dart, Afhe_Dart, performs C calls with the Interface, fhe_C, and interacts with C objects, SEAL_C, within the memory stack. The glue between Afhe_Dart and fhe_C uses dart:ffi, a Dart plugin, to facilitate communication between components. Abstract Fully Homomorphic Encryption, Afhe_Dart, library invokes methods in the C interface, fhe_C from Dart. The adapter design shown in Figure 2 demonstrates the Dart abstraction layer invokes the C adapter layer. Using dart:ffi, Dart can execute C functions, reference memory addresses of C objects, and convert primitive data types.

Abstract Fully Homomorphic Encryption defines a set of commands to perform basic functionalities of existing Fully Homomorphic Encryption (FHE) libraries written in C. The bridge design shown in Figure 3 of this library implements an abstraction layer over existing FHE libraries. Through abstraction, we can interface with various backend libraries via the same function calls. The interface layer, fhe_C, exposes Afhe_C concrete classes, ex. SEAL_C, lower level C function. Through the use of pointers, we can create/destroy/reference C objects from Dart.

TABLE I
Class Structure Overview

File	Class	Description
seal.dart	Seal	Entry-point for the end-user API in Dart
afhe.dart	Afhe	Dart adapter connecting to the C interface
fhe.cpp	N/A	C interface bridging Dart and C++
afhe.h	Afhe	Pure abstract class defining the FHE contract
aseal.cpp	Aseal	Concrete implementation of the Afhe abstraction

The Fully Homomorphic Encryption Library provides a modular API that integrates Microsoft SEAL with the Flutter ecosystem. It offers a straightforward interface for developers, abstracting low-level complexities while delivering access to advanced encryption functionalities required for secure applications. The topology of this Dart plugin ensures a clear separation of responsibilities to promote maintainability and extensibility. Table I outlines the distinct roles of each component, including the high-level API, the adapter interface, and the concrete implementation.

The *seal.dart* file provides the primary entry point for the end-user API, allowing developers to interact with the encryption library in Dart. The *afhe.dart* file acts as the adapter, invoking lower-level C functions. The Afhe_Dart class exposes abstracted data types such as Keys, Ciphertext, and Plaintext objects. The *fhe.cpp* file defines the C interface. It exposes the inherited methods from *afhe.h* and references the memory address of the underlying C++ object. Depending on the reference, the corresponding concrete implementation will be invoked. For example, *aseal.cpp* manages Microsoft SEAL objects.

The Foreign Function Interface (FFI) is this plugin's core dependency; It enables Dart to interact with the pre-compiled C binary. For each target platform, the dynamically linked library must be accessible on the local file system. This plugin exposes methods to cast native C data types into Dart objects and vice versa. Our implementation creates Dart objects that mirror their corresponding underlying C++ class. For example, in Microsoft SEAL, a Ciphertext can be saved or loaded. We expose *save_ciphertext* and *load_ciphertext* in the C interface. In Dart, we create a Ciphertext class that contains both *save* and *load*, referencing the memory address of the underlying abstract Ciphertext object. This implementation pattern is consistent and transparent for security researchers familiar with the underlying C++ API. For new developers, our implementation mirrors the structure of Microsoft SEAL, providing a clear and familiar interface.

The following example demonstrates what the code would look like to implement Cramer Distance using our plugin in Dart. In this example, a mobile crowdsensing application would have received an encrypted list of values that have been

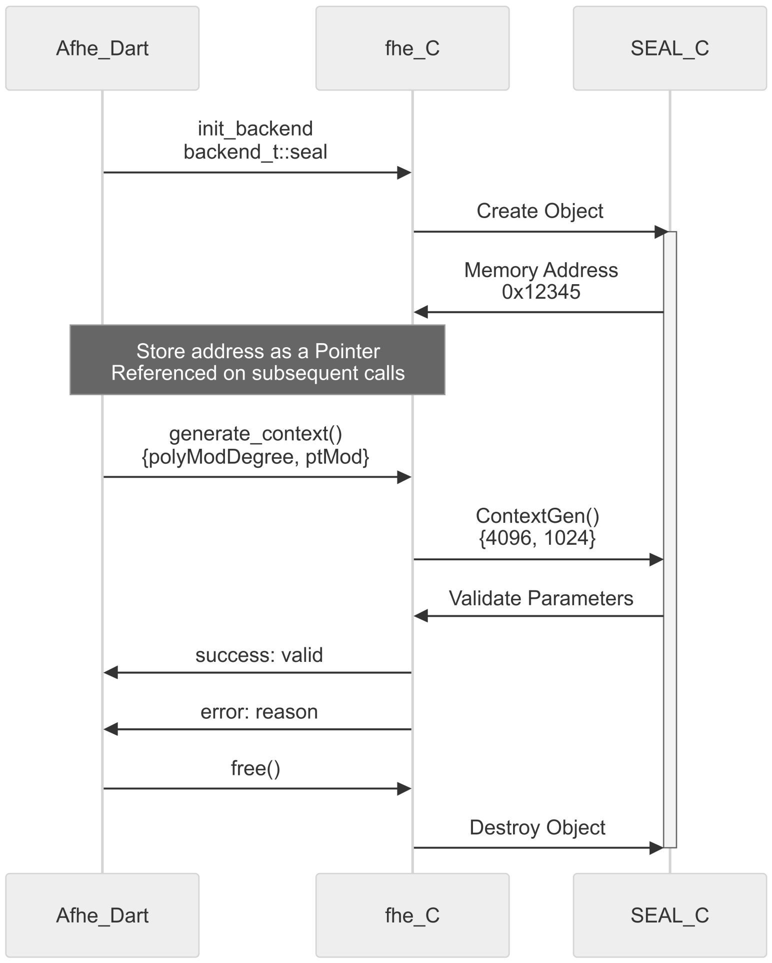

Fig. 1. Foreign Function Interface Between Dart & C/C++ Libraries

sensed by another user, x, and would then compute the Cramer distance between their list of values that they have collected, y, and return the result to user x. In this instance, user y does not need to encrypt their plaintext values since they never leave their device and FHE works much faster when computing on an encrypted value and a plaintext value than it does when computing on two encrypted values.

```
List<Ciphertext> cramer(Afhe fhe,
   List<Ciphertext> x, List<double> y) {
  List<Ciphertext> result = [];
  for (int i = 0; i < x.length; i++) {
    CipherText r =
      fhe.square(fhe.subtractPlain(x[i],
      fhe.encodeDouble(y[i])));
    result.add(r);
  }
  return result;
}
```

IV. Evaluation

A. Computational Performance

In this section, we compare the performance of the Dart and Flutter implementation to the original Java and C++ implementation to demonstrate that there is not a significant degradation of performance by switching to a more usable programming framework.

Experiments used one-minute videos with h.264 encoding. GhostPeerShare was run on a Pixel 3XL that has 8 threads at about 2.5 GHz with 4 GB of memory. From the Pop-Share benchmark, the Pixel 2 has 8 threads at about 2.35 GHz with 4 GB of memory. While not exactly the same hardware, we determined that it is close enough to demonstrate our point that our system does not result in a shift in performance that would render existing systems unusable.

The Fully Homomorphic Encryption computations, shown in Table II, are within ±20% of Pop-Share, validating our abstraction design and interface implementation. This performance increase is likely due to the advancements in Microsoft SEAL. Pop-Share used version v3.3, while our implementation uses v4.1.

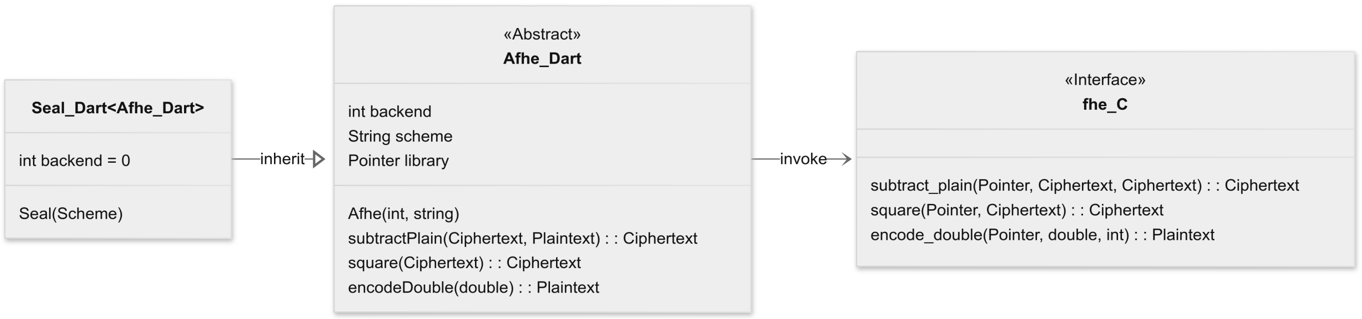
«Abstract»
Afhe_Dart
int backend
String scheme
Pointer library
Afhe(int, string)
subtractPlain(Ciphertext, Plaintext) : : Ciphertext
square(Ciphertext) : : Ciphertext
encodeDouble(double) : : Plaintext
Seal_Dart<Afhe_Dart>
int backend = 0
Seal(Scheme)
inherit
invoke
«Interface»
fhe_C
subtract_plain(Pointer, Ciphertext, Ciphertext) : : Ciphertext
square(Pointer, Ciphertext) : : Ciphertext
encode_double(Pointer, double, int) : : Plaintext

Fig. 2. Dart to C Adapter Class Diagram

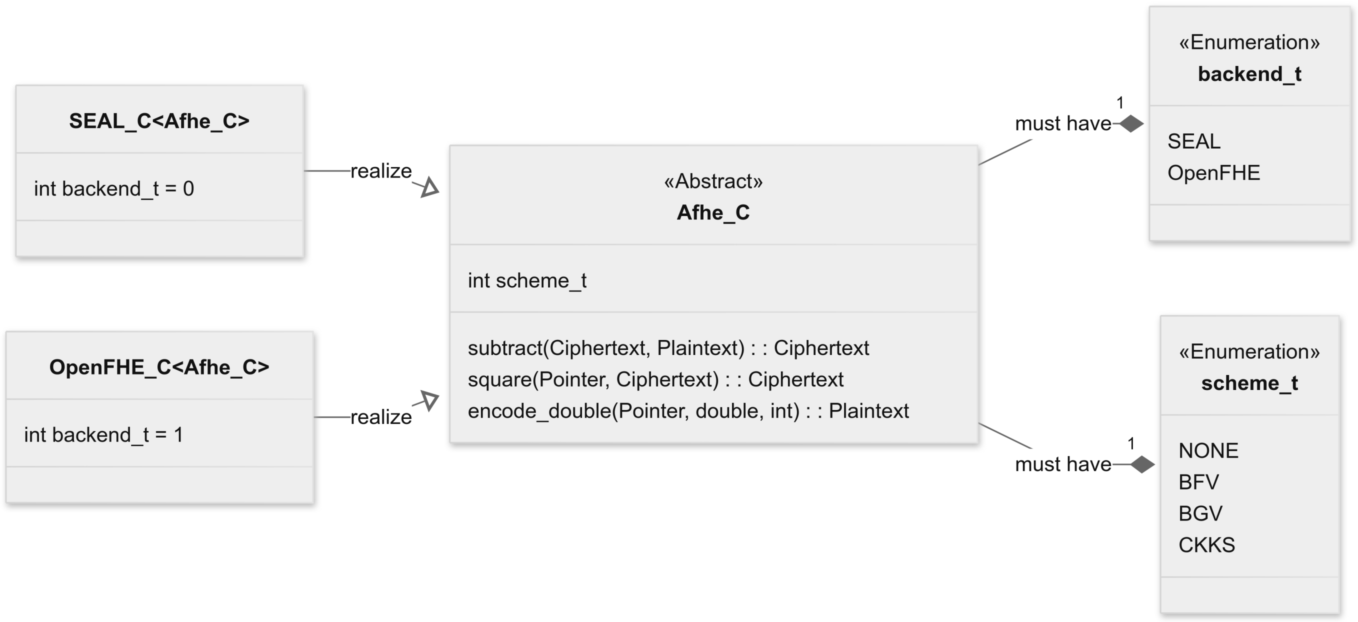

Fig. 3. C++ Bridge Class Diagram

TABLE II
COMPARISON OF FHE OPERATIONS (IN MILLISECONDS) ON ANDROID

Algorithm	Pop-Share (ms)	GhostPeerShare (ms)	% diff
KLD	400	345	-13.5%
BC	386	202	-19.4%
CD	186	309	+9.1%

TABLE III
DIFFERENCE OF MEAN ERROR FOR EACH ALGORITHM

Function	Pop-Share	GhostPeerShare
KLD	4×10^{-9}	1.71×10^{-11}
Cramer	4×10^{-4}	1.61×10^{-9}
BC	8×10^{-4}	3.85×10^{-10}

Overall, this study benchmarked the performance of our implementation on Android to the existing Pop-Share application. The results show that, while not an exact replication of the original system due to the availability of hardware and changes in software libraries, the performance of our system is comparable to the performance of the original Java and C++ based implementation.

B. Noise Accumulation

When performing fully homomorphic operations, a small amount of noise is introduced that affects the accuracy of the decrypted distance measure. If too much noise is accumulated, the plaintext cannot be recovered. Table III represents the baseline Mean Error from Pop-Share [7], compared to our implementation. For Kullback-Leibler Divergence (KLD), Cramer Distance (CD), and Bhattacharyya Coefficient (BC), our implementation introduced half the amount of noise compared to the original. This was computed by subtracting the mean error of GhostPeerShare (GPS) from Pop-Share (PS), such that $PS - GPS$ resulted in approximately the same value as Pop-Share. This difference is likely attributed to the changes in the security parameters and advances in Microsoft SEAL between v3.3 and v4.1, respectively.

TABLE IV
DIRECT COMPARISON OF SSO-BASED SYSTEMS

System	F1 (%)	Precision (%)	Recall (%)	Accuracy (%)	Error (%)
GhostPeerShare	97.09	96.14	98.05	98.05	1.95
Handheld[2]	97.97	99.32	96.67	98.00	2.00
SSO[6]	96.13	92.56	100.00	96.30	3.70
PoP-Share[2]	96.63	97.73	95.56	95.16	4.84

C. Field of View

A significant contribution from Pop-Share [7] was the ability to accurately classify videos from various angles for similarity. Applying the methodology from Similarity of Simultaneous Observation (SSO) [12], [13] to compare videos, Pop-Share re-used the pre-processing procedure to convert videos into a byte-count array. Instead of comparing videos to capture network traffic, the application of SSO was altered to compare two videos to each other for similarity. In addition, Pop-Share expanded upon the distance measure implementation to support Fully Homomorphic Encryption, requiring a new set of algorithms that were simplified into basic arithmetic.

The field-of-view experiment demonstrated that comparing two videos taken at the same time and place of the same subject could be accurately classified using an Artificial Neural Network (ANN). This machine-learning model was trained on the distance measure scores between two videos computing using three probability distribution functions: Kullback-Leibler Divergence (KLD), Bhattacharyya Coefficient (BC), and Cramer Distance (CD). For supervised training, the binary labeling system assigns a one when two scenes are identical, or the first comparison pair is labeled as one to train the model to match similar videos. Otherwise, a zero is assigned for all other comparison pairs, regardless of visual similarity.

SSO and Pop-Share models shown in Table IV were trained for 15 hours using a Nexus 6p and a D-Link Wi-Fi camera (DCS-936L) fixed-motion cameras. The video data from the two cameras include video captures at different resolutions and different relative angles, such as 0, 90, and 180 degrees offset from each other. The videos were also taken at varying distances from each other, ranging from 1 to 25 meters away. In addition, the videos were taken from both an indoor and an outdoor environment with varying levels of motion and lighting conditions.

The Handheld model shown in Table IV was tested using 150 minutes of training data recorded with a Google Pixel 2, a Motorola Moto Z, a Lenovo Phab2 Pro, an LG Nexus 5, and a Huawei Nexus 6p. All phones captured video using h.264 with 3840x2160 resolution at 30 FPS with Optical Image Stabilization (OIS) enabled except the Nexus 5 and Phab2 Pro, which only support 1920x1080 resolution, and the Nexus 6p, which only has Electronic Image Stabilization (EIS).

GhostPeerShare shown in Table IV was trained and tested on a less diverse dataset of 100 minutes of raw videos, which included three twenty-minute videos of low movement featuring an individual sitting at his desk and two twenty-minute videos of high movement capturing an individual vacuuming his living room. The video data was recorded on fixed-motion phones: Pixel 3XL using h.264 with 1920x1080 resolution at 30 FPS and Samsung S9 using h.264 with 1280x720 resolution at 30 FPS. The phones were placed at different relative angles and varying distances. However, all videos were filmed indoors with relatively similar lighting.

For a practical analysis, the ANN was trained on a high-movement scene and tested on a low-movement scene, achieving an Accuracy and Recall score of 98.05%, a Precision Score of 96.14%, and an F1 Score of 97.09%. The lack of diversity within the training and testing dataset is likely a contributing factor to the lower scores for F1 and Precision. Recording more video data in various locations with a wide variety of devices and environments, the F1 and Precision scores may improve. This model demonstrates that GhostPeerShare consistently generates a unique representation of video data and can accurately predict the binary label given the three distance measure scores for a different dataset.

V. FUTURE WORK

In order to address the usability of our Flutter application and Dart plugins, this section aims to propose methods of gathering qualitative metrics of our implementation that benefit the crowdsensing community. For developer surveys, we aim to gather information about each individual's programming knowledge or familiarity. Using a Likert-scale questionnaire, we may rate the clarity of our API documentation on a scale of 1 to 5. Using open-ended questions to capture insightful feedback from users who may be seeking additional functionality or identify gaps in our implementation. This structured survey format enables maintainers to assess the level of difficulty in integrating the plugins relative to each individual's experience with open-source projects. Gathering data on the consumers of our packages will help us understand their use cases and the greater impact of this project within the Flutter community. These surveys may be available on the distribution platforms the packages are hosted on, specifically GitHub and Pub.Dev.

To capture the adoption rate of our packages, we may leverage the platforms on which they are hosted. The source code for this project is available on GitHub. Using the star functionality is a popular method of endorsing a project to receive notifications on development updates. The maintainers of this project may visualize the number of stars over time to measure the project's popularity.

While this work only utilized SEAL as a cryptographic backend, other cryptographic backends could be adopted. In future work, we will provide additional libraries that can be selected by crowdsensing application developers within Dart.

VI. Conclusions

Key contributions demonstrate the efficiency of the Fully Homomorphic Encryption Library (FHEL) and ease of integration into other Dart plugins or Flutter applications. To measure the efficiency of our application, we benchmark GhostPeerShare against Proof of Presence Share (Pop-Share).

When trained on a dataset with continuous movement and tested on a dataset with infrequent movement, the application achieved 0.9614 precision and an F1 score of 0.9709. These results indicate that GhostPeerShare consistently generated a distinct representation of each video in the dataset. When executing FHE operations, GhostPeerShare demonstrated similar performance to Pop-Share.

Ultimately, this project lays the foundation for future applications of FHE in mobile environments, especially in fields like healthcare and secure data transactions, where privacy and accuracy are paramount. By making SEAL accessible within a mobile-compatible, modular framework, we are opening pathways for broader adoption of FHE and setting the stage for innovation in mobile security.

A. Acknowledgments

This work was partially supported by the National Science Foundation Award OISE-1853953.

References

[1] X. Zheng, Q. Yuan, B. Wang, and L. Zhang, "A Homomorphic Encryption Based Location Privacy Preservation Scheme for Crowdsensing Tasks Allocation," *Wireless Personal Communications*, vol. 126, no. 1, pp. 719–740, Sep. 2022. [Online]. Available: https://doi.org/10.1007/s11277-022-09767-y

[2] R. Ganjavi and A. R. Sharafat, "Edge-Assisted Public Key Homomorphic Encryption for Preserving Privacy in Mobile Crowdsensing," *IEEE Transactions on Services Computing*, vol. 16, no. 2, pp. 1107–1117, Mar. 2023, conference Name: IEEE Transactions on Services Computing. [Online]. Available: https://ieeexplore.ieee.org/abstract/document/9767554

[3] Y. Zheng, H. Duan, and C. Wang, "Learning the Truth Privately and Confidently: Encrypted Confidence-Aware Truth Discovery in Mobile Crowdsensing," *IEEE Transactions on Information Forensics and Security*, vol. 13, no. 10, pp. 2475–2489, Oct. 2018, conference Name: IEEE Transactions on Information Forensics and Security. [Online]. Available: https://ieeexplore.ieee.org/abstract/document/8323405

[4] X. Ding, R. Lv, X. Pang, J. Hu, Z. Wang, X. Yang, and X. Li, "Privacy-preserving task allocation for edge computing-based mobile crowdsensing," *Computers & Electrical Engineering*, vol. 97, p. 107528, Jan. 2022. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0045790621004730

[5] L. Kong, L. He, X.-Y. Liu, Y. Gu, M.-Y. Wu, and X. Liu, "Privacy-Preserving Compressive Sensing for Crowdsensing Based Trajectory Recovery," in *2015 IEEE 35th International Conference on Distributed Computing Systems*, Jun. 2015, pp. 31–40, iSSN: 1063-6927. [Online]. Available: https://ieeexplore.ieee.org/document/7164890

[6] "Microsoft SEAL (release 4.1)," https://github.com/Microsoft/SEAL, Jan. 2023, microsoft Research, Redmond, WA.

[7] B. Lagesse, G. Nguyen, U. Goswami, and K. Wu, "You had to be there: Private video sharing for mobile phones using fully homomorphic encryption," in *2021 IEEE International Conference on Pervasive Computing and Communications Workshops and other Affiliated Events (PerCom Workshops)*. IEEE, Mar. 2021.

[8] S. Halevi and V. Shoup, "Helib," C++ library, Oct. 2020. [Online]. Available: https://github.com/homenc/HElib

[9] D. Cousins, K. Rohloff, and Y. Polyakov, "Palisade: A library for privacy-preserving cryptography," C++ library, Dec. 2022. [Online]. Available: https://palisade-crypto.org/

[10] A. Ibarrondo and A. Viand, "Pyfhel," in *Proceedings of the 9th on Workshop on Encrypted Computing & Applied Homomorphic Cryptography*. New York, NY, USA: ACM, Nov. 2021.

[11] L. S. Vailshery, "Cross-platform mobile frameworks used by global developers 2023," Jun. 2024. [Online]. Available: https://www.statista.com/statistics/869224/worldwide-software-developer-working-hours/

[12] K. Wu and B. Lagesse, "Detecting hidden webcams with delay-tolerant similarity of simultaneous observation," in *Pervasive and Mobile Computing*, 2020.

[13] ——, "Do you see what i see? detecting hidden streaming cameras through similarity of simultaneous observation," in *2019 IEEE International Conference on Pervasive Computing and Communications (PerCom)*. IEEE, Mar. 2019. [Online]. Available: https://doi.org/10.1109/PERCOM.2019.8767411