You Had to Be There: Private Video Sharing for Mobile Phones using Fully Homomorphic Encryption

Brent Lagesse*, Gabriel Nguyen†, Utsav Goswami‡ and Kevin Wu§
Computing and Software Systems
University of Washington Bothell
Bothell, WA USA
Email: *lagesse@uw.edu, †gaben@uw.edu, ‡utsavg@uw.edu, §kevinw9@uw.edu

***Abstract*—Smart phones and other small, low-cost video cameras have enabled the creation of billions of hours of video content. Many of these videos contain private content that the users want to share with a limited group of people, not all of whom they know. In this paper, we address the problem of sharing videos from different perspectives of the same event without either party revealing any content of the video until it has been established that the videos were of the same event. To address this problem, we designed and implemented a system that can detect similar videos and efficiently be run under a fully homomorphic encryption scheme on mobile devices.**

The resulting system was able to achieve a precision of 0.9932 and an F1 score of 0.9797 for determining the same videos. The algorithm was able to test a video in 0.4 seconds on an Android 9.0 Pixel 2 and run end-to-end in 1.5 seconds. As a result, participants in an event can share videos with each other to gain a full perspective of the event without releasing the videos to people who were not present.

I. Introduction

With billions of smart phones and other devices capable of recording video deployed across the world, there is a great opportunity to archive more of personal and world history than ever before. Sources estimate that YouTube users upload over 500 hours of video per minute [1]. Additionally, people have begun to develop systems that utilize these cameras such as lifelogging systems[2] that record audio, video, images, and other data to assist people in increasing self-awareness and augmenting memories.

These types of systems, along with the general ability to record scenes that a person may want to share with a limited group, but not with the general public have also raised a number of security and privacy concerns[3]. The goal of privacy in this work is to limit video distribution to people who already have video recordings of a given event. Previous work has focused on protecting bystanders in the video and protecting the owner of video from memory manipulation, but no work has addressed the ability to share video privately. There are many situations in which a person may be at an event where they do not know everybody, but the content of the event may not be the type of material for indiscriminant public availability. For example, events involving children such as birthday parties, school plays, and sporting events may be considered sensitive. Similarly, events or meetings that discuss matters sensitive to oppressive governments also fall into this category. **The goal of our work is to enable the user to identify videos from people who were co-present at an event without revealing any of either video to anybody until it has been determined that they were co-present.**

In this paper we present our Proof of Presence system, **PoP-Share**, for sharing videos with previously unknown people who were co-present at an event. We have developed two architectures for the video distribution system. The first architecture, a client-server architecture, enables users to offload computation from the mobile phone to a cloud server. We also developed a P2P architecture that performs computation on the mobile phones, but is secure against a more advanced attacker model. To enable videos to be compared privately, we utilized a fully homomorphic encryption (FHE) scheme which enabled us to perform calculations directly on encrypted data, thus hiding the contents from the party executing the comparison.

Contributions. The major contributions of our work are summarized as follows:

1) Development of a scene similarity detection system using FHE described in section III-E
2) Implementation of a privacy-preserving video sharing application for Android, Raspberry Pi, and PC described in section IV-A
3) Demonstration of the feasibility of FHE video similarity computation on a mobile platform via extensive performance testing described in section IV.

II. Background

A. Similarity of Simultaneous Observations

Wu and Lagesse [4] presented Similarity of Simultaneous Observation (SSO) in which they leverage similarities in video encoding to detect hidden cameras. In this work, they recorded the rate of bytes transmitted over the network by devices using a network interface in monitor mode and compared those data streams with the encodings of a known camera monitoring the same area. This is possible since common interframe compression algorithms used in most modern video codecs

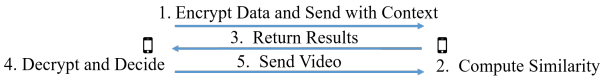

Fig. 1: Peer-to-Peer Protocol

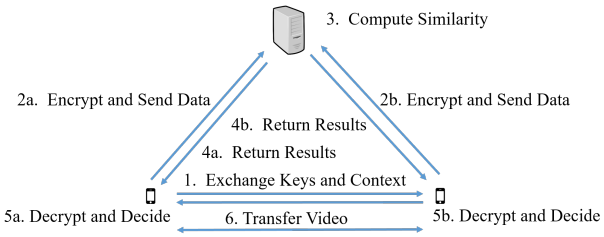

Fig. 2: Client-Server Protocol

use less data to encode frames in a video when there is very little change in the video and use more when there is change occuring within the video scene. They then used several different similarity measurements such as correlation coefficient, Kullback–Leibler divergence (KLD), Jensen-Shannon divergence (JSD), and Dynamic Time Warping (DTW) distance. They then trained classifiers on these measurements and demonstrated F1 scores over 0.95.

B. Homomorphic Encryption

We have chosen to use FHE to perform the calculations over possible partially homomorphic encryption because FHE enables our choice of similartiy measures and provides extensibility and flexibility to improve performance in the future. Homomorphic encryption allows computations on encrypted data that will persist when the data is decrypted. One of our major contributions is designing the system so that it is computationally efficient under fully homomorphic encryption. To do this, we use Microsoft SEAL[5] which implements the BFV cryptosystem and a FractionalEncoder in version 2 and the CKKS cryptosystem in version 3 which is more efficient when computing on real numbers.

III. Design

We address three major challenges that had to be overcome to build this system. The first is the computational complexity of fully homomorphic encryption. The second is the minimalistic FHE libraries that are currently available. The third is the architectural design necessary to ensure both secure and computationally efficient processing. As a result of our design, we were able to drastically reduce the amount of data that needs to be processed and identify algorithms that could process data under a fully homomorphic encryption scheme with minimal errors and sufficiently low computational cost for a usable, secure system.

A. FHE Design Concerns

In order to run FHE efficiently on a mobile device, The appropriate computation and parameters must be selected. We have addressed this through the similarity detection algorithms we have selected to maintain a balance of classification performance, security, and computational performance. We have devised methods for implementing each algorithm in such a way that it avoids costly multiplications, relinearizations, and operations where multiple operands are encrypted.

Polynomial Modulus. The size of the polynomial modulus affects performance. The larger the modulus the longer computations will take; however, a large polynomial modulus means that more operations can be performed before the decoded value becomes invalid. We address this in our similarity measurement choices by identifying algorithms that can run on with a 4096-bit polynomial modulus or less.

Noise Budget. One of the concerns about the CKKS cryptosystem is the noise growth that exists in the system. As operations are performed on a ciphertext, the noise associated with it grows. When that noise grows too much, the decryption will fail to produce the correct result. Relinearization is a process that can reduce a ciphertext back to its original size after multiplication; however, the relinearization process itself is very costly. Our system avoids the need for costly relinearizations as our early experiments showed they were causing our algorithm runtime to increase by a factor of 6.

B. System Model and Assumptions

We define a *scene* to refer to the portion of an event that a video captures. In our system, multiple participants at an event take video of the same scene at the same time. The participants can be positioned at any relative angle to the scene and can have varying levels of overlap of the scene. We do not assume that the users have temporally synchronized recordings, but we do assume that the videos do have some temporal overlap. Our system quickly identifies approximately 95% of true positives with about 15 seconds of video, but we try to identify at least 1 minute of video overlap to reduce false positives to under 3%. The more overlap that is found, the greater the performance.

Many modern mobile phones are equipped with image stabilization, Electronic (EIS) or Optical (OIS). While we assume the user is recording the video in a mostly static position and attempting to hold the phone still, the stabilization features will ensure that there is minimal movement in even very shaky videos[1].

C. Attacker Model and Assumptions

The focus of this work is to prevent an attacker from gaining information that they could not have acquired without being present at the event that was recorded. As a result, we assume that the attacker does not already have a video of the scene that we are trying to share (either through being physically present or otherwise acquiring one). The attacker can serve as either the initiator or the recipient of a video transfer request. The attacker can eavesdrop and modify any communications.

In the case of the client-server architecture, we assume that the attacker does not control both the server and one of the participating clients. The security of that scheme relies on the

[1]https://www.youtube.com/watch?v=x5rHog6RnNQ

server not having a copy of the private key used for decryption of the results.

D. Algorithm Selection

The algorithm used for SSO in [4] included 4 distance measurements, KLD, JSD, Correlation Coefficient, and DTW. We analyzed the construction of these algorithms under FHE and determined that the existing work was not feasible due to computational cost and errors from approximation. We have modified the distance measurements used for our system due to the fact that the functions supported by SEAL require very costly and sometimes inaccurate approximations for some of the distance measurements. Instead, we identified KLD, Bhattacharyya distance, and Cramer distance as measurements that are easy to implement in FHE and provided improved results over the previous work. Table I shows the mean error achieved on our approximations required by the FHE library.

TABLE I: Error Caused by FHE Approximations

Func	Mean Error
KLD	4×10^{-9}
Cramer	8×10^{-4}
BC	1.3×10^{-1}

E. System Architecture

Our system provides support for both a Client-Server architecture and a P2P architecture. The client-server architecture in figure 2 enables mobile devices to offload computation to a (presumably more powerful) third party. Due to space constraints, we describe only the P2P version of the architecture in detail. A nonce is included to prevent replay attacks. The keypair used is unique for each transaction.

1) Peer-to-Peer: The P2P architecture in figure 1 enables mobile devices to avoid the possibility of an attacker collaborating with the third party server.

(a) Encrypt Data and Send with Context. The user that is making the decision whether or not to share the video preprocesses the video and encrypts it. The encrypted data and the context is sent over a TLS connection.

(b) Compute Similarity. The other user computes all of the requisite similarity scores. The other user cannot see what effect their inputs have other than if they get the file or not.

(c) Return Results. The encrypted results are returned to the decision-making user.

(d) Decrypt and Decide. The decision-making user decrypts the results and makes a decision.

(e) Send Video. If the decision-making user is satisfied, they send the video to the other participant.

2) Data Preprocessing and Precomputation: To efficiently use FHE, we needed to identify operations that could be performed on the unencrypted data so that we minimize the amount of work that had to be performed on encrypted data. In particular, multiplication causes the most rapid degradation of noise budget that can lead to the inability to decrypt the final result. Likewise, we identified operations that could be performed on a single ciphertext and an encoded plaintext (which is much less expensive than an operation performed on two ciphertexts).

Figure 3 describes the pre-processing portion of the system. The first step that we take is to identify all the videos that will be compared. In the case of the initiating party, this is all of the videos the user selects related to the event that the want to acquire more videos from. In the case of the sharing party, we do this by using timing metadata from the videos themselves to identify videos that could have feasibly taken place at the same time. It also allows us to align the video arrays appropriately. This step is not absolutely required, but it drastically cuts down on the number of times the FHE computations need to be run.

The next step we take is to convert videos into a bytecount array and then normalize it so that it sums to 1. These arrays capture the number of bytes required by the intra-frame compression algorithm to encode a single second of video. The more movement in a second of video, the more bytes are used to encode it. This process has two main advantages, one for privacy and one for performance. Since we are only performing operations on the number of bytes encoded in a particular timestep, if an attacker later broke the encryption on this data, they would not learn the content of the video; however, we must use FHE for the similarity check because if the attacker knew the byte count values, they could craft a fake bytecount array that would result in the an incorrect classification that the videos were the same. Our algorithm is now performing operations on a much smaller amount of data. For examples, *many of our experiment videos are reduced from 50-100 MB down to 2560 bytes of data, thus making it feasible to perform fully homomorphic encryption operations in a reasonable amount of time.*

We chose to precompute the logarithm of all the values we needed on the client side in plaintext, then encrypt those values for computation later. Additionally, direct division is not possible in the SEAL library; however, it is possible to compute the inverse of a value and then multiply by the encrypted inverse, so by avoiding division, we save on the number of encrypted values we have to send. This design decision resulted in a small increase in preparation time and data sent, but saved us significant amounts of time and error in the final computation.

Due to the choices we have made in the similarity measures and how we calculate each of those, we can achieve usable computation times by only sending the following encrypted data (each array is 640 bytes):

- Normalized byte count array in Probability Distribution Form (Used in KLD)
- Natural Log of normalized byte count array in Probability Distribution Form (Used in KLD)
- Normalized byte count array in Cumulative Distribution Form (Used in Cramer)
- Square root of normalized byte count array in Probability Distribution Form (Used in Bhattacharyya)

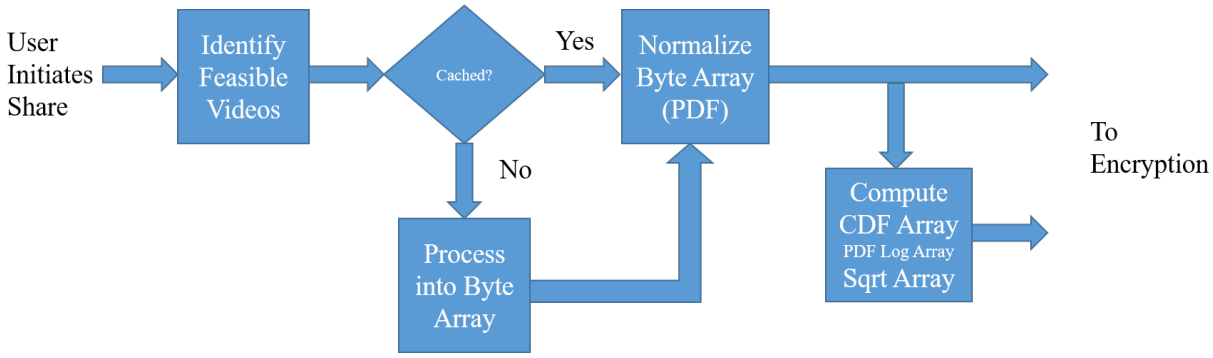

Fig. 3: Preprocessing and Precomputation Process

IV. Results

In this section we describe our implementation of PoP-Share and the computational performance results on a variety of devices. We demonstrate that PoP-Share can effectively run on mobile devices and include results from several other types of devices for comparison.

A. Implementation

We have two implementations of PoP-Share. The first implementation, designed to run on most platforms, is written in Python 3.6 and uses PySeal[6] as a wrapper for the SEAL library. The current version of PySeal supports SEAL 2.3, so this implementation uses the BFV implementation of fully homomorphic encryption. Our PC tests run the 64-bit Linux version on Ubuntu 18.04.2. Our Raspberry Pi tests run the AArch64 build on Ubuntu MATE 18.04.2 for Raspberry Pi ARMv8 64-bit system.

Our second implementation is designed to run as an Android App. The app is built for Android 9.0 and runs on 64 bit CPUs. The app uses SEAL 3.3[5] built using the Android NDK, so this implementation uses the CKKS implementation of fully homomorphic encryption. We have implemented all of the FHE functionality in native C++ with a JNI wrapper to be accessed through the Android app.

B. Experimental Setup

Unless otherwise noted, experiments were run on the following devices. FHE experiments on Android were run on a Google Pixel 2 with 4 GB of RAM. Experiments on the PC were run on a 4^{th} generation Lenovo Thinkpad Carbon X1 laptop with an Intel i7-6600U CPU and 16 GB of RAM. Experiments on the Raspberry Pi were run on a Raspberry Pi 3b with a 1 GB of RAM.

TABLE II: Table of FHE Parameters

Parameter	Value
Polynomial Modulus	4096
Coefficient Modulus	111
Plaintext Modulus	512
Decomposition Bit Count	60
FractionalEncoder Integer Part	32
FractionalEncoder Fractional Part	16

Table II describes the parameters that were used for our experiments unless otherwise noted. Polynomial Modulus defines the degree of the ring on which we are performing operations. The larger the modulus, the greater the security and our ability to perform more operations; however, larger ring sizes will slow computation. We are able to perform all of our computations with a polynomial modulus of 4096 with 128 bits of security.

We performed tests using the dataset collected in [4]. This dataset includes 15 hours of video captured from a Nexus 6p and a D-Link Wi-Fi camera (DCS-936L) with H.264 encoding along with 4.33 hours of other videos from the Internet. These videos include a wide variety of movement, angles, and activities from indoor and outdoor settings.

We have also augmented this dataset with 150 minutes of additional video that we have collected by a handheld mobile phone to ensure a realistic user model since the videos collected by [4] were by mounted devices, so they lacked the natural shakiness that would be present when a person was recording an event. This is shown in table V as "Handheld". These videos were 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 OIS enabled except the Nexus 5 and Phab2 Pro which only support 1920x1080 resolution and the Nexus 6p which only has EIS. The recordings varied both distance and angles.

C. FHE Timing

The purpose of these results is to provide a performance baseline and demonstrate the conditions under which it is feasible to use PoP-Share. Preprocessing on a Pixel 2 mobile phone took, on average, 17 seconds for a 1 minute video, but it can be done offline between the recording and sharing of videos. The TLS handshake and data transfer takes 0.25 seconds. Similarity computation takes 1 second. Result transfer takes less than 1 ms. Decryption takes 0.25 seconds. The overall online latency for checking a single video is approximately 1.5 seconds for a 1 minute video. Table IV demonstrates the performance of the algorithms on different system types with 128-bit security. Table III shows the cost

increase by using FHE over plaintext operations on a mobile phone.

TABLE III: Comparison of FHE Operations vs Plaintext Operations in Milliseconds

Measure	FHE	PT	Percent
KLD	400	3.8	10526%
Cramer	386	5.2	7423%
Bhattacharyya	186	3.2	5812%

Pre-Processing Time. Relative to calculating similarity measures, pre-processing, including encoding and encryption, can be expensive. In SEAL 3.3 on a PC, preprocessing for Bhattacharyya and Cramer each take about 110 ms and in SEAL 2.3 they take about 372 milliseconds on average. KLD takes about 220 milliseconds and 763 milliseconds, respectively since twice as much data is required for KLD. For the same reason, on the Android phone, Bhattacharyya and Cramer each take about 4.2 seconds and KLD takes 8.5 seconds. While this is a large amount of time for pre-processing on the mobile phone, this is a process does not have to be run every time. As noted in figure 3, we cache the results after a video has been processed, so this is only a one time cost that can be incurred during the time between recording a video and sharing the video, so it is unlikely to be noticed by the user.

TABLE IV: Comparison of Computational Time in ms for Different Systems under CKKS

System	Bhatt	Cramer	KLD
Android	186	386	400
PC	18.2	19.1	19.3
Server	20	21	22

Mobile Performance. The performance on the Android device was similar to or better than that of the SEAL 2.3 PC implementation that uses FractionalEncoder rather than CKKS. On average, KLD ran in 400 ms, Cramer in 386 ms, and Bhattacharyya in 186 ms.

PC Performance. On the PC in P2P mode, KLD ran in 19.3 ms, Cramer ran in 19.1 ms, and Bhattacharyya ran in 18.2 ms on the CKKS cryptosystem. Bhattacharyya produces so little noise that it can run on a much lower dimension ring than the other algorithms. At 2048 bits it runs in 21 ms and at 1024 bits it runs in 10 ms for SEAL 2.3 and 5.8 ms and 3.2 ms, respectively, for SEAL 3.3.

Server Performance. In the case of the client-server architecture, a 3^{rd} party computes the scores for the two parties interested in sharing videos. As a result, computations are performed on two cryptotexts rather than on a cryptotext and an encoded plaintext. This results in a slight performance decrease, so KLD ran in 22 ms, Cramer ran in 21 ms, and Bhattacharyya ran in 20 ms on the CKKS cryptosystem.

Raspberry Pi Performance. Raspberry Pi performance, as expected, was the worst overall. The Raspberry Pi only has results for SEAL 2.3.

D. Classification Performance

We performed a grid search and evaluated a neural network with 10-fold cross validation. The resulting model used a L-BFGS solver with 4 hidden layers of 15 neurons each and a logistic activation function. We optimized the model for precision rather than F1 score. Optimizing for precision was our main goal as the cost of a false positive is that a video that should not be shared will be shared, thus violating privacy, whereas the cost for a false negative is only that no sharing will take place (which we considered much less costly). As shown in table V, our model produces a result with a slightly better F1 score than the previous work on the same dataset, but with significantly higher precision. Note that [4] separates their videos into indoor and outdoor, so the values presented here are the weighted average of indoor (68%) and outdoor (32%) videos that were in the dataset. After manually comparing the videos that produced false positives, we identified that 75% of these videos were skype videos of the same location as the comparison video where there was no movement in the scene in either, but at a different time. We reran the test without these videos as it is unlikely that a user would want to exchange a video of nothing happening and this improved our F1 score to 97.73. This produced a very similar result to our experiments that we ran using comparisons of video only when a user was holding a mobile phone by hand which produced and F1 score of 97.97.

TABLE V: Direct Comparison of SSO-based Systems

System	F1	Prec	Recall	Acc	Error
PoP-Share	96.63	97.73	95.56	95.16	4.84
SSO[4]	96.13	92.56	100.00	96.30	3.70
Handheld	97.97	99.32	96.67	98.00	2.00

V. Discussion

A. Performance

The results in section IV are run on a single core each, so the time for the system to run is bound by the slowest algorithm. All of the distance measures that we use can be run simultaneously and each distance measure is also computed with no dependency between the individual data computations. As a result, if performance needs mandated that the system run faster, we could utilize multiple cores to provide near-linear performance improvements as a function of the number of cores available (assuming usual caveats regarding other activity on the system).

B. Security

The security equivalency of homomorphic encryption is a function of the dimension of the ring and the size of the ciphertext modulus. The larger the dimension of the ring, the larger the ciphertext modulus must be to provide equivalent levels of security and as a result, performance suffers. For reference, see table VI for the balance between ring dimension (n), bits of security, and ciphertext modulus.

TABLE VI: Security Parameter Table

n	Bits of Security	CT Modulus Bits
4096	128	111
	256	60
8192	128	220
	256	120

Replay Attacks: Since the system generates a new keypair for each transactions, replaying an encrypted byte array would result in the decision-making client decrypting junk values. The client can check the nonce to see if it is correct and if it has been altered due to incorrect decryption, it can ignore transaction.

C. Limitations and Future Work

PoP-Share has not been fully analyzed for security. It may be possible for an attacker to strategically craft a video that causes false positives. As future work, we will perform a security analysis to determine if an attacker can do better than a brute force attack. We are also implementing a context-aware system that will leverage additional context information if it is available based on our preliminary experiments on Context-Aware Proof of Presence (CAPP)[7].

VI. Related Work

Fan[8] describes a privacy preserving system specifically for video sharing. They focus on content privacy at the video clip level and prevent statistical inferences from video collections. Their work uses computer vision techniques to identify objects and actions of interest in the video and can replace them with virtual objects that protect the identity of privacy-leaking content while still capturing the intended content of the video. Upmanyu [9] enables privacy-preserving video surveillance. They obscure video content in surveillance systems that is irrelevant to the person viewing. Thuraisingham [10] focuses on access control in video databases. They provide privacy preserving video techniques deriving from the work of [8] and limit access based on a credential hierarchy. Yang [11] focuses on access control in a cloud-based video distribution system. The goal of their work compared to previous work is to be able to share videos with a group of people during a particular time period. Their system is built on top of the CP-ABE scheme[12] and utilize it to ensure that users cannot decrypt video outside of a particular time.

To the best of our knowledge, no existing work meets the requirements of our video sharing system. The focus of some previous work has been on obscuring the video itself whereas we want all videos that are shared to be in original condition. Also, these systems are designed to release information to the user, albeit in an obfuscated form, even if they were not present at the event. Other systems are designed to limit the sharing of video to known groups or users whereas we need videos to be distributed without having to have a pre-existing relationship with the other users. In many cases, a cloud server performs the processing on plaintext data and breaks our requirement that an honest, but curious, server should also not learn the content of the video.

VII. Conclusions

We have presented our system, PoP-Share, that enables privacy-enhancing video sharing for people who were co-present at an event. We have built an efficient video scene comparison system on the CKKS fully homomorphic cryptosystem to ensure that the videos can be processed for similarity without revealing information about the video itself. We have demonstrated the feasibility of using FHE on a mobile phone by implementing and testing this system both entirely on a mobile phone via a P2P architecture and leveraging cloud resources to offload computation via a client-server architecture. The decision can be made in as little as 400 ms on a mobile device and 20 ms on a cloud server. We have demonstrated the classification performance of our modified SSO algorithm by achieving and F1 score of 0.966 and a precision of 0.977, both of which exceed previously published results. Additionally, we achieved and F1 score of 0.9797 and a precision of 0.9932 on a newly recorded dataset. As a result, we conclude that not only is privacy-enhanced video sharing within arbitrary groups of users that were co-present at an event now possible, it is possible to perform directly on modern mobile phones without requiring excessive wait times.

References

[1] James Loke Hale, "More Than 500 Hours Of Content Are Now Being Uploaded To YouTube Every Minute," May 2019.

[2] C. Dobbins and S. Fairclough, "A mobile lifelogging platform to measure anxiety and anger during real-life driving," in *EmotionAware)*, Mar. 2017, pp. 327–332.

[3] P. Aditya, R. Sen, P. Druschel, S. Joon Oh, R. Benenson, M. Fritz, B. Schiele, B. Bhattacharjee, and T. T. Wu, "I-Pic: A Platform for Privacy-Compliant Image Capture," ser. MobiSys '16, 2016, pp. 235–248.

[4] K. Wu and B. Lagesse, "Do You See What I See? Detecting Hidden Streaming Cameras Through Similarity of Simultaneous Observation," *IEEE Pervasive Computing and Communications*, p. 10, 2019.

[5] *Microsoft SEAL (release 3.3)*, Jun. 2019. [Online]. Available: https://github.com/Microsoft/SEAL

[6] "Simple encrypted arithmetic library," Jul. 2019, original-date: 2017-10-19T18:53:48Z. [Online]. Available: https://github.com/Lab41/PySEAL

[7] N. Handaja and B. Lagesse, "CAPP: A Context-Aware Proof of Presence for Crowdsensing Incentives," *orkshop on Context and Activity Modeling and Recognition*, p. 6, 2020.

[8] J. Fan, H. Luo, M.-S. Hacid, and E. Bertino, "A Novel Approach for Privacy-preserving Video Sharing," ser. CIKM '05. New York, NY, USA: ACM, 2005.

[9] M. Upmanyu, A. M. Namboodiri, K. Srinathan, and C. V. Jawahar, "Efficient privacy preserving video surveillance," in *2009 IEEE 12th International Conference on Computer Vision*, Sep. 2009.

[10] B. Thuraisingham, G. Lavee, E. Bertino, J. Fan, and L. Khan, "Access control, confidentiality and privacy for video surveillance databases," 2006.

[11] K. Yang, Z. Liu, X. Jia, and X. S. Shen, "Time-Domain Attribute-Based Access Control for Cloud-Based Video Content Sharing: A Cryptographic Approach," *IEEE Transactions on Multimedia*, vol. 18, no. 5, pp. 940–950, May 2016.

[12] A. Lewko and B. Waters, "Decentralizing Attribute-Based Encryption," in *Advances in Cryptology – EUROCRYPT 2011*, 2011.