Efficient Oblivious Computation Techniques for Privacy-Preserving Mobile Applications

Henry Carter, Chaitrali Amrutkar, Italo Dacosta, and Patrick Traynor
Converging Infrastructure Security (CISEC) Laboratory
Georgia Tech Information Security Center (GTISC)
Georgia Institute of Technology
{carterh, chaitrali}@gatech.edu, {idacosta, traynor}@cc.gatech.edu

Abstract—**The growth of smartphone capability has led to an explosion of new applications. Many of the most useful apps use context-sensitive data, such as GPS location or social network information. In these cases, users may not be willing to release personal information to untrusted parties. Currently, the solutions to performing computation on encrypted inputs use garbled circuits combined with a variety of optimizations. However, the capability of resource-constrained smartphones for evaluating garbled circuits in any variation is uncertain in practice. In [1], it is shown that certain garbled circuit evaluations can be optimized by using homomorphic encryption. In this paper, we take this concept to its logical extreme with Efficient Mobile Oblivious Computation (EMOC), a technique that completely replaces garbled circuits with homomorphic operations on ciphertexts. We develop applications to securely solve the millionaire's problem, send tweets based on location, and compute common friends in a social network, then prove equivalent privacy guarantees to analogous constructions using garbled circuits. We then demonstrate up to 68% runtime reduction from the most efficient garbled circuit implementation. In so doing, we demonstrate a practical technique for developing privacy-preserving applications on the mobile platform.**

I. Introduction

The confluence of high-speed connectivity and device capability have led to the recent surge in mobile application development. While software common to desktop computing (e.g., word processing, email) exists in this space, the most popular mobile applications often provide services based on a user's current context (e.g., location [2], social interconnections [3], etc.). Such applications allow users to make more informed decisions based on their surroundings. However, these applications also regularly expose sensitive data to potentially untrusted parties.

Cryptographers have long worked to develop mechanisms that allow two parties to compute shared results without exposing either individual's sensitive inputs or requiring assistance from a trusted third-party. Such techniques are referred to as *Secure Function Evaluation* (SFE), and provide a set of powerful primitives for privacy-preserving computation. While theoretical constructions have been known for nearly 30 years [4], efficient realizations of such schemes have only become possible within the last few years [5], [6], [1], [7], [8]. These tools are beginning to produce increasingly efficient privacy-preserving applications for desktop and server-class machines. However, their application to mobile devices, where the use of context sensitive information is the norm and not the exception, has just begun to be assessed [9]. While efforts to improve the performance of SFE have largely focused on optimizing garbled circuit constructions, another option exists. As demonstrated by Henecka, homomorphic encryption can be used to replace selected garbled circuit functions [1]. However, this idea has yet to be taken to its logical extreme – the construction of privacy-preserving applications exclusively constructed with partially homomorphic cryptosystems.

In this paper, we test this hypothesis and develop more efficient alternatives to garbled circuits that are capable of running on mobile platforms. Instead of relying on expensive garbled circuit techniques, our *Efficient Mobile Oblivious Computation* (EMOC) techniques restate such applications as a series of simple arithmetic operations. Using partially homomorphic cryptosystems allows us to perform operations on pairs of ciphertexts that, when decrypted, represent the same operations on their corresponding plaintext values. We design and implement two privacy-preserving examples of popular application classes: location-based Twitter feeds and a social networking tool to identify nearby "friends of friends". In addition to providing equivalent guarantees of data privacy to garbled circuit-based techniques, we demonstrate that our applications can produce the same results at computational and memory costs *reduced by orders of magnitude in some cases.*

In so doing, we make the following contributions:

- **Provide a new approach to efficient privacy-preserving computation:** Rather than attempting to further optimize the garbled circuit construction, we follow the concept of replacing some garbled circuits with homomorphic encryption to its logical extreme, designing the first privacy-preserving mobile applications to employ purely homomorphic encryption.
- **Design privacy-preserving mobile applications replacing garbled-circuit constructions with homomorphic cryptographic primitives:** We design

privacy-preserving versions of programs representative of two of the most popular mobile application classes: *location-based messaging* and *social networking*. For clarity and completeness, we also define a third application based on the canonical example of secure two-party computing - the Millionaire's Problem. We prove that our applications provide equivalent security guarantees to their SFE-based counterparts.
- **Characterize EMOC and traditional SFE mobile performance profiles:** We conduct an extensive performance analysis of both our approach and traditional SFE techniques on the Android mobile platform. We demonstrate that EMOC offers performance improvements upwards of 68% over the most optimized previous techniques. Moreover, we compare two top performing garbled circuit optimizations using Ordered Binary Decision Diagram (OBDD) [6] and pipelined circuits [7] on the mobile platform.

While our technique is not yet generalized for computing arbitrary functions, our performance evaluations demonstrate that the performance gains achievable through partially homomorphic cryptography constructions merit special constructions for certain functions. In the past, optimization of secure computation for specialized applications has led to significant progress within this field of research [10]. Although garbled circuits have the capability of computing arbitrary functions, *the specific functions we examine are especially pertinent to the mobile platform and require maximum efficiency to be of practical value to the user.*

The remainder of this paper is organized as follows: Section II discusses previous research in mobile privacy, two-party computation, and private search; Section III describes basic cryptographic concepts necessary to our protocols as well as the security assumptions used in proving the security of our applications; Section IV describes our protocols for each of the three example applications; Section V lays out the technical proofs of the security of each application; Section VI describes our performance evaluation procedures and discusses the results; Section VII offers concluding remarks.

II. Related Work

The majority of the research on mobile privacy focuses on data queries to a server. The earliest technique developed for this purpose was the concept of k-anonymity [11], which reveals a user's private information to a server only if that information is indistinguishable from $k-1$ other users, preventing the server from identifying a connection between a specific user and their private information. This concept has been used in numerous mobile protocols since its development [12], [13], [14]. Other protocols expand upon k-anonymity by implementing an added parameter, l-diversity. This expansion seeks to make tracking distinct users in a k-anonymous system more difficult by preventing a server from tracing any user through more than l locations [15]. Other systems have attempted to obscure location through the use of dummy information and camouflage [16], [17], [18], [19]. Puttaswamy and Zhao instead proposed that application servers be treated as encrypted repositories, with users downloading their private information to the device before the application performs any computation [20]. Narayanan et al. developed a scheme for proximity testing by examining the characteristics of the surrounding area (e.g., wireless access points, packets crossing wireless networks, active bluetooth devices), and comparing these characteristics with those collected by another user [21]. This technique does not allow for proximity testing beyond the range of a single wireless access point, and also assumes that the two computing parties will have some level of friendship or trust that is previously established. The defining characteristic of all these systems is that they provide privacy for a client accessing an application server. However, some applications require that two users who do not trust one another be able to compute results based on their private information. None of these tools are appropriate for performing such computation between two peer users.

With the development of the Secure Function Evaluation (SFE) protocol, Yao demonstrated the possibility of two peer users computing a function without exposing their private inputs [4]. For years following, implementations of the protocol were too computationally intensive for practical use. In 2004, Malkhi et al. produced the first practical implementation of SFE in the program Fairplay [5]. Fairplay provided a language and compiler for building the "garbled circuits" that are used to compute functions securely (i.e., without revealing either party's inputs to the other). Fairplay offers the same privacy guarantees as the trusted third party model without requiring an actual third party. Building upon the Fairplay compiler, Kruger et al. developed a technique for replacing garbled circuits with ordered boolean diagrams [6] to improve Fairplay's speed for certain functions. Most recently, Huang et al. developed a technique for pipelining circuit construction and evaluation, allowing for circuits to scale to any size without filling up the memory of the constructing machine [7]. These efforts have produced a practical means for performing secure computation in a desktop environment. However, garbled circuit evaluation still requires significant processor and memory overhead when producing and evaluating circuits, and exchanging encrypted inputs. Even with the improved performance of Kruger's OBDD and Huang's pipelining approach, and considering Kerschbaum's assertion that communication overhead is of little importance in secure computation [22], garbled circuits are likely to be too expensive for the hardware constraints of mobile devices. Huang et al. began exploring this question in a work examining the performance of pipelined circuits on mobile phones [9]. We thoroughly evaluate this assertion directly as part of this work.

One possible solution to this problem lies in the relatively young area of homomorphic encryption. Several protocols for private information retrieval [23] and private stream search [24], [25] leverage this property of certain encryption schemes to search and compare encrypted data in a manner that prevents the machine performing the search from learning anything about the data. The technique has already been shown to be useful in applications such as voting [26] and distributed location privacy [27], and Gentry's development of the fully homomorphic encryption scheme [28], although still highly impractical, demonstrates the potential of outsourced computation with privacy guarantees. The benefit of the currently available partially homomorphic encryption schemes is that they are efficient, even on mobile devices [29]. More recently, Henecka et al. showed that homomorphic encryption can be used to speed up Fairplay garbled circuits for a specific set of functions [1]. This work in particular begs the question, if using homomorphic cryptography can optimize the performance of garbled circuits, could it be used to completely replace garbled circuits? To demonstrate the feasibility of this notion, and to optimize secure computation to the point where it is usable on mobile devices, our paper presents three sample applications that restate secure computation functions as private search operations over homomorphically encrypted data. This restatement of secure function evaluation as private search removes many of the computational inefficiencies of SFE while providing equivalent privacy guarantees.

III. Cryptographic Assumptions

Before we define and prove the security of our applications, we specify the requirements for the encryption schemes used. We also state basic assumptions that are necessary for the security of our protocols to hold.

A. Homomorphic Cryptography

The main tool our protocols use in guaranteeing the privacy of all inputs is the homomorphic property of certain cryptosystems. For a cryptosystem to be homomorphic, there must be some operation that, when performed on two ciphertexts, causes some predictable change to the underlying plaintexts. Specifically, for a homomorphic encryption scheme $E_k()$, given two plaintext messages $X, Y \in M$; ciphertexts $C, D : E_k(X) = C, E_k(Y) = D$; and operations $\cdot$ and $\diamond$,

$$C \cdot D = E_k(X \diamond Y) \tag{1}$$

The protocols described in this paper capitalize on one particular type of homomorphic property: multiplicative homomorphisms. In a multiplicative homomorphic scheme, the product of two ciphertexts is equal to an encryption of the product of two plaintexts:

$$C \times D = E_k(X \times Y) \tag{2}$$

B. Public Key Encryption

The second requirement for cryptosystems used in our protocols is that they be public key encryption schemes. All of these protocols require that one of the participants in a two-party computation must perform homomorphic operations over encrypted data. This user must be able to encrypt his own inputs, but be *unable* to decrypt the result of the homomorphic operations. This operation is clearly only feasible in a public key cryptosystem, where the user performing homomorphic operations does not possess the decryption key.

C. Encryption Scheme Security

Any encryption scheme used in our applications must be provably indistinguishable under an adaptive chosen-plaintext attack (IND-CPA). That is, any non-uniform, probabilistic polynomial-time adversary A can be given access to an encryption oracle. This encryption oracle, given two message inputs, always encrypts either the left input or the right input, depending on the random bit b chosen at the start of the experiment. The goal of the adversary is to distinguish whether the oracle is encrypting the left input or the right input. Given a security parameter n, the adversary's advantage is [1]:

$$\begin{aligned} Adv(A) = Pr_{pk}[Exp^{Enc_k^1(\cdot,\cdot)}(A) = 1] \\ - Pr_{pk}[Exp^{Enc_k^0(\cdot,\cdot)}(A) = 1] \leq negl(n) \end{aligned} \tag{3}$$

Our protocols require that only a limited number of known values be encrypted and revealed to another party. The cryptosystem used must guarantee that encryptions of these known plaintext values are indistinguishable. In the most extreme example, one protocol generates an array of encrypted values, each being either a '1' or '2'. Each entry in the array can be thought of as a query to the IND-CPA oracle in the experiment described above. For some polynomial number of queries, any adversary has the following advantage:

$$\begin{aligned} Adv(A) = Pr_{pk}[Exp^{Enc_k^1(1,2)}(A) = 1] \\ - Pr_{pk}[Exp^{Enc_k^0(1,2)}(A) = 1] \leq negl(n) \end{aligned} \tag{4}$$

which implies that no adversary can determine the underlying value with probability better than a random guess. Intuitively, the IND-CPA definition of security implies that the cryptosystem must be probabilistic, generating many different ciphertexts that all decrypt to the same plaintext. Thus, *an encrypted array containing only two distinct plaintext values will have a multitude of ciphertext values, effectively obscuring the difference between each entry.*

[1]For a more thorough treatment on IND-CPA security, see [30].

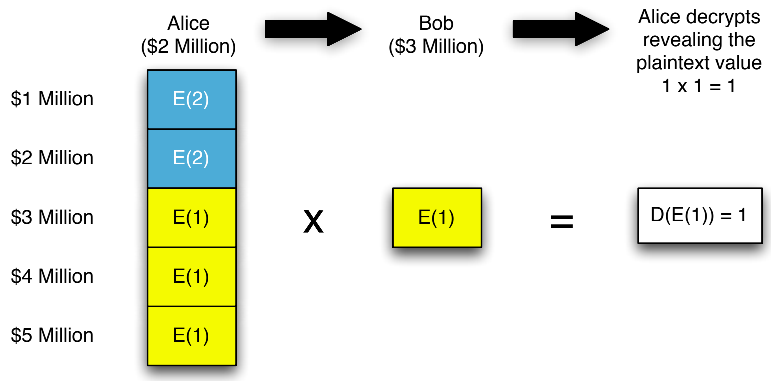

Figure 1. The Millionaire's Problem Protocol: Bob has $3M, so he multiplies E(1) with the third entry.

D. Threat Model

In developing the protocols used in our applications, we make two assumptions to provide results in a fair and secure manner. The first is that finding a trusted third party is difficult or impossible. As an example, while a number of websites currently offer to provide the location of friends within a certain physical radius, they can not be trusted to *not* process, store or sell such data.

The second assumption is that all privacy guarantees in Section V hold against a semi-honest adversary. As defined in Lindell and Pinkas' work [31], this means that an adversary will follow the protocol as written, using valid inputs, but will attempt to learn as much as possible outside the jointly computed results by studying logs of all communications. Since this protocol is meant to guarantee the privacy of inputs, we can do nothing if the user chooses false inputs designed to corrupt the protocol. Secure Function Evaluation also makes this same assumption, *proving security based on semi-honest adversaries* [5], [6], [1], [7], [31]. Not only is our model equivalent to the current security model for two-party computation, we assert that this model is realistic and useful for certain context-sensitive mobile applications in the market. Our protocols developed under this threat model will also provide a foundation for seeking protocols that can guarantee privacy against other adversarial models.

IV. EMOC Application Protocols

In this section, we describe in detail the protocols for our three sample applications. We first present the EMOC version of the millionaire's problem - the canonical two-party secure computation example and a simple initial case. We then present the EMOC version of a Location-Based Twitter application, which allows Alice to subscribe to Bob's tweets without either party revealing their location. Finally, we present the EMOC version of our Social Graph connectivity tool, which allows Alice and Bob to determine where their social networks overlap without exposing the identities of all of their friends - an application with potential use when meeting new (potentially untrusted) people.

A. The Millionaire's Problem

The millionaire's problem represents the canonical example of two-party computation. Although solving this problem has limited practical applications, it can be used to demonstrate general needs of all two-party computation applications. It is for this reason that we developed our applications starting with this problem.

1) Definition of the Millionaire's Problem: Assume two millionaires, Alice and Bob, who wish to compute which millionaire has more wealth without either party revealing the exact size of his or her fortune. Both Alice and Bob input a value from 1-N representing the size of their fortune (in millions of dollars), and receive a Boolean value indicating whether or not Bob is richer than Alice.

2) Description of the protocol: Before the protocol is initiated, both parties must define the following elements:

- A *multiplicative* homomorphic encryption scheme $E_{pk}(\cdot)$. In all figures, this will be denoted as E$(\cdot)$.
- An integer value N which represents some number of millions that is greater than the wealth of either party.

When Alice wishes to initiate the millionaire's problem protocol, she begins by generating an encrypted query (Figure 1). First, Alice generates a public and private key pairing in the predefined homomorphic encryption scheme. Alice then creates an array of size N and stores a '2' in every entry with an index less than or equal to the number of millions she possesses. For example, if $N = 5$ and Alice has 2 million dollars, then she would store a '2' in the first and second entries. She then fills the rest of the entries with the value '1'. Once this is complete, Alice encrypts each entry in her query using her public key. She then sends her encrypted query array Q_A and her public key pk_A to Bob.

Upon receiving Q_A and pk_A, Bob first selects the entry in Alice's array Q_A that corresponds to the number of millions he possesses. For example, if Alice sends an array $N = 5$ and Bob has 3 million dollars, he selects the third entry (Figure 1).

At this stage, Bob cannot simply select an encrypted value and return it to Alice, as she would be able to correlate

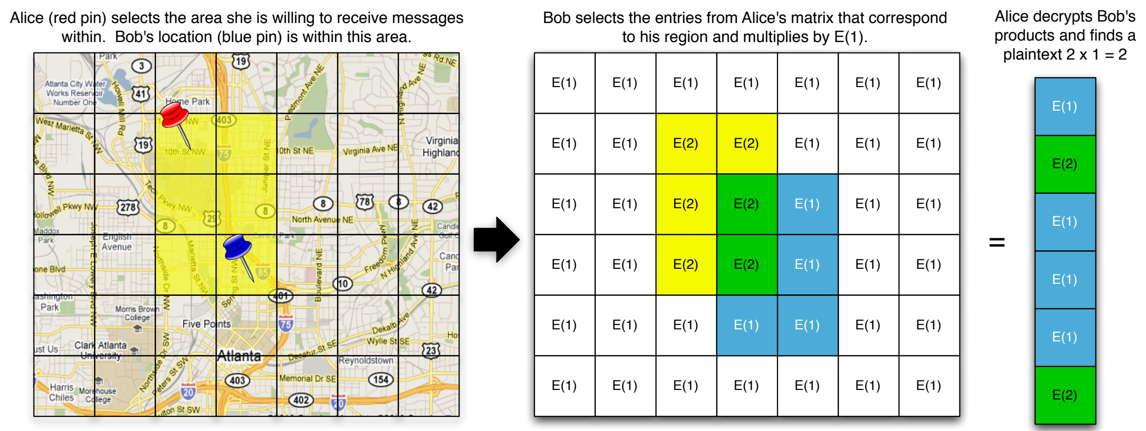
Alice (red pin) selects the area she is willing to receive messages within. Bob's location (blue pin) is within this area.
Bob selects the entries from Alice's matrix that correspond to his region and multiplies by E(1).
Alice decrypts Bob's products and finds a plaintext 2 x 1 = 2
E(1)
E(2)
=
Atlanta
Five Points

Figure 2. The Location-Based Messaging Protocol. Alice builds a location matrix with encryptions of '1' in every entry except those that correspond to the area she is willing to receive tweets within. In her travel area, she enters encryptions of '2'. Bob selects the entries that corresponds to his region, multiplies by an encryption of '1', and returns the product to Alice. When Alice decrypts, she knows that: if any value is a '2', Bob's tweet is relevant to her. If every value is a '1', Bob's message is not relevant to her current location.

the unique ciphertext with the location it came from in her query array. Therefore, Bob encrypts the number '1' under Alice's public key and multiplies the encrypted entry he selected with this encrypted value, re-randomizing the ciphertext and yielding the result R. Bob completes this phase of computation by sending R back to Alice.

Upon receiving the encrypted value from Bob, Alice decrypts R using her private key, which yields the plaintext value '1' or '2', depending on which value Bob selected from Alice's query. To complete the communication, Alice sends this decrypted result to Bob so that he can learn the output of computation. If the final output is a '1', the index he selected in Alice's query is greater than the number of millions Alice possesses. Therefore, Bob's number of millions is higher than Alice's, and he is the wealthier of the two. If the outcome is a '2', Bob's number of millions is less than or equal to Alice's.

B. Location-Based Messaging

Location-based messaging, especially for advertisements, has recently received significant attention. Beyond advertising based on location, it offers the potential for useful applications such as a proximity test to alert two people when they are close enough to arrange a meeting. It could also be combined with applications including Twitter to allow for location-based tweet filtering and following. However, all of these applications must query the physical location of mobile users, which could compromise the user's privacy. To resolve this information leakage, we present a technique that will obscure the user's precise location, only allowing the querying party to know if the user is in a general region.

1) Problem Definition: Assume two Twitter users, a follower Alice and a tweeter Bob. Since Bob generally tweets about events in his general vicinity, Alice wishes to receive tweets from Bob only when she is nearby. Alice selects as her input an area around her current location where, if Bob tweets close to this area, she wants to receive the tweet. Bob inputs an area around his current location where his tweets would be relevant. The goal is to compute whether the area where Alice wishes to receive Bob's tweets intersects with the area where Bob's tweets are relevant.

2) Description of the Protocol: Before the protocol is initiated, both parties must define the following elements:

- A *multiplicative* homomorphic encryption scheme $E_{pk}(\cdot)$. In all figures, this will be denoted as E($\cdot$).
- A matrix of size $M \times N$ where each cell corresponds to a physical region within the city where Alice and Bob are located. Imagine the matrix as a grid laid over a city map. Each cell has a publicly known correlation to the city location beneath it.

This problem expands upon the protocol for solving the millionaires problem. Before receiving any of Bob's location-based tweets, Alice selects an area of her city of any shape or size that defines the area where she wants to receive tweets (Figure 2). She then generates an $M \times N$ location matrix L_A. For each cell, Alice inputs a '2' if that cell corresponds to the area where she wishes to receive tweets, and '1' if it does not. She then encrypts each cell in the matrix with her public key pk_A. When Alice checks Bob's Twitter feed, she initiates the protocol by sending L_A to Bob.

When Bob receives Alice's location matrix L_A, Bob then selects the cells in L_A that correspond to the region where his tweet is relevant, and for each of the n cells, he re-randomizes the entry by multiplying in an encryption of the value '1', as in the Millionaire's problem. Bob then returns an array of the n results R to Alice. Upon receiving R, Alice decrypts the values using her private key to find n values, either '1' or '2'. She then sends this decrypted array of values back to Bob to complete the computation.

If any of the values returned is a '2', the area where Bob's tweet is relevant intersected with the area where Alice wished to receive tweets, and Bob can respond by delivering his latest tweet. If all the values returned are '1', the area where Bob's tweet is relevant does not intersect with the area Alice wishes to receive tweets, and Bob need not respond. In this manner, Alice never gets to see Bob's precise location and vice versa.

It is relevant to note that this protocol, while used for a specific application here, can be expanded into a general proximity test. If Alice and Bob choose their input regions as the area they are willing to search for the other party, they can test to see if the two are within this desired range. The ability to specify an input region of any shape or size allows the proximity test to provide a result at any desired granularity, from the same building to the same city.

C. *Social Graph Privacy*

Social networking applications are a popular channel for communicating with a mobile device. However, they are also a potential channel to leak private information about a mobile user's social life. If two mobile users were to meet at a party or conference, one might only want to allow the other into her social network based on the friends they already have in common. However, there is currently no application which allows this without revealing both users' entire social graphs. This application offers a means for securely revealing only the friends common to both users while maintaining the privacy of the rest of both social graphs. Differing from the previous two problems, which simply find whether or not a set intersection exists, the Social Graph application finds an intersection of two sets and returns all the elements within that intersection. This is useful when a user's privacy must be maintained for information unique to that user. As an example, when determining whether or not to add a friend on a social network, the requester (or recipient) often looks to see what friends he has in common with the other party. Rather than exposing the other party's entire social graph to determine common friends, our application only reveals common connections, keeping the remainder of the other party's social graph private.

1) Definition of the Social Graph Problem: Assume two participants, Alice and Bob, who are both members of a social network. Each participant assigns a subset of the social network members as their friends. Given both Alice and Bob's lists of friends, we wish to compute which members of the social network are friends with both Alice and Bob while keeping the rest of their friend lists private.

2) Description of the protocol: Before the protocol can be initiated, the following elements must be defined:

- A *multiplicative* homomorphic encryption scheme $E_{pk}(\cdot)$ which allows for modular multiplication over some cyclic group G. In all figures, this will be denoted as E($\cdot$).
- A secure keyed hash algorithm.
- A security parameter t.
- A predetermined number of friends N to be compared.

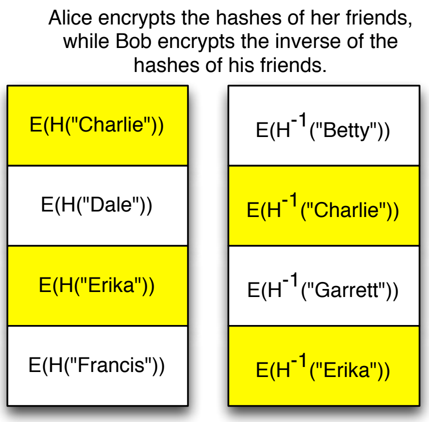

Figure 3. Social Graph query: Alice and Bob build encrypted arrays of their friends by hashing their names using a keyed hash. No ordering need be considered in this setup.

When Alice wishes to initiate the Social Graph protocol, she begins by generating a query array Q_A of size N. She does this by generating a hash key K_H and hashing each of her friends' names. She then stores these hashes in random order in Q_A. Alice then generates a public key pair, encrypts each entry with her public key pk_A and then sends Q_A, K_H, and pk_A to Bob.

Upon receiving Q_A, Bob hashes each member of his friend list using K_H. Then, he finds the multiplicative inverse of each hash within the group G which is the group of elements over which Alice's public key can encrypt. As we will later observe, if one of Bob's hashes matches one of Alice's, the product of her hash and his inverted [2] hash will be '1'. At this point, Bob encrypts each of his inverted hashes with Alice's public key pk_A. He then generates two arrays of length N^2. For the first array, he homomorphically multiplies each of his encrypted and inverted hash values by all of Alice's encrypted hash values, performing N^2 comparisons. In the second list, he generates N^2 random elements of the group G, which he also encrypts with Alice's public key. He then homomorphically multiplies the elements in this array of blinding factors B with the elements in his array of compared values, generating the result of homomorphic computation R (Figure 4). Finally, Bob sends R back to Alice.

When Alice receives R, she decrypts each element using her private key, which yields an array of N^2 random values from the group G due to Bob's blinding factors. She then sends back *only* the t least significant bits of each decrypted value to Bob.

Bob receives the decrypted values from Alice and for each entry in her results, he compares the bits returned with the least significant t bits of the blinding factor he multiplied into that entry. If the results match, it means that entry contained a '1', implying that a friend matched. To calculate

[2]This does not mean Bob inverts the hash to recover the preimage; rather, the mathematical inverse of the hash value mod p.

which friend matched, Bob simply uses integer division on the index, where $i \div N$ is the original index of the matching friend. Thus, Bob can identify which members' of the social network he shares with Alice. Bob then sends the list of names in the intersection back to Alice to complete the computation.

3) Correctness Argument: We quantify here the probability that Bob will have a false positive when matching the last t bits of the blinding factor with Alice's returned result. Since the hash function is assumed to be pseudorandom, multiplying a pseudorandom number x by the multiplicative inverse of a pseudorandom number y^{-1} yields a pseudorandom number z. After this value is multiplied by a random blinding factor b and truncated to the least significant t bits, it is apparent that the probability of the least significant t bits of b matching the least significant t bits of $z*b$, is $\frac{1}{t^2}$, or no better than randomly selecting bits. This value can be increased to yield a higher probability of correctness or decreased to hide more information about the resulting hashes. The amount of information revealed by t bit truncation is defined in section V.

V. Privacy Guarantees

In this section, we define our threat model and prove the privacy guarantees of each protocol from the previous section. For each protocol, we show two properties: the security of the two-party computation and the amount of information revealed by the result of computation.

A. Definitions

In all of our protocols, we assume the standard definition of a semi-honest adversary, described in Lindell and Pinkas' [31]. Essentially, this states that both parties will follow the protocol as written but will attempt to learn information beyond the computed result from transcripts of the interaction. This assumption is also made by related efforts in this space [5], [6], [1], [7], [31]. To prove a protocol secure against semi-honest adversaries, we use the concept of indistinguishability between Alice's view in a real execution and a simulator's generation in an ideal execution. In the ideal world, two participants A, B send their inputs a, b to a trusted third party which performs some computation and returns the result $f(a, b)$. The proof idea is to show that a simulator S in the ideal world can simulate A's view in the real protocol.

Definition 1. *Semi-honest security: For any deterministic functionality $f(x, y)$ and semi-honest parties P_1 and P_2, we say that protocol π securely computes f in the presence of semi-honest adversaries if there exists probabilistic polynomial-time algorithms S_1 and S_2 such that:*

$$S_1(x, f(x,y))_{x,y\in\{0,1\}^*} \overset{c}{\approx} view_1^{\pi}((x,y), output^{\pi}(x,y))_{x,y\in\{0,1\}^*} \quad (5)$$

$$S_2(x, f(x,y))_{x,y\in\{0,1\}^*} \overset{c}{\approx} view_2^{\pi}((x,y), output^{\pi}(x,y))_{x,y\in\{0,1\}^*} \quad (6)$$

B. Millionaire Privacy

1) Protocol Security:

Theorem 1. *Millionaire Privacy: Assuming that the encryption scheme used in the Millionaire's protocol is semantically secure (i.e., indistinguishable under adaptive chosen-plaintext attack), the Millionaire's protocol is secure in the presence of semi-honest adversaries.*

Proof: We prove separately the security of the Millionaire's Protocol when Alice is corrupt and when Bob is corrupt:

Alice is corrupt: Throughout the proof, $f(x, y)$ is defined as the boolean function $x < y$, which is the function being computed by our Millionaire's protocol. We construct the simulator S_A as follows:

S_A is given input $(x, f(x, y))$, where x is the number of millions Alice possesses and $f(x, y)$ is defined as above . Upon receiving the initial message from Alice containing her array of encrypted values A and her public key, S_A generates an encryption of the value $f(x, y)$ using Alice's public key and returns the message $m_1 = E(f(x, y))$, completing Alice's view of the interaction.

We now demonstrate that the message sent by S_A, comprising Alice's view, is computationally indistinguishable from an interaction with Bob. For the first message, two points must be shown. First, we know that:

$$m_1 = E(f(x,y)) \overset{c}{\approx} E(1*1) \overset{c}{\approx} E(1*2) \quad (7)$$

Based on the semantic security of our encryption scheme, S_A's encryption of the result is indistinguishable from either of Bob's result gained by homomorphic multiplication. Second, we know trivially that the decrypted value in both the simulated view and the real execution is $f(x, y)$

Therefore, the Millionaire's Problem protocol is secure when Alice is corrupt.

Bob is corrupt: We prove security by constructing a simulator S_B as follows: the simulator is given input $(y, f(x, y))$, where y is the number of millions Bob possesses, and $f(x, y)$ is defined as above. Upon receiving input, S_B generates an array A of length n where each entry contains the value $f(x, y)$. S_B randomly generates an encryption key pair SK_{S_B} and PK_{S_B}. It then individually encrypts each entry in A using PK_{S_B}, producing $E(A)$. Finally, it sends to Bob $m_1 = E(A)||PK_{S_B}$. Upon receiving Bob's reply containing the result of the homomorphic operation R, S_2 generates $m_2 = D(SK_{S_B}, R)$ (i.e. the decryption of Bob's message) and sends it to Bob.

We now demonstrate that the messages sent by S_B are indistinguishable from Bob's view in a real execution. For

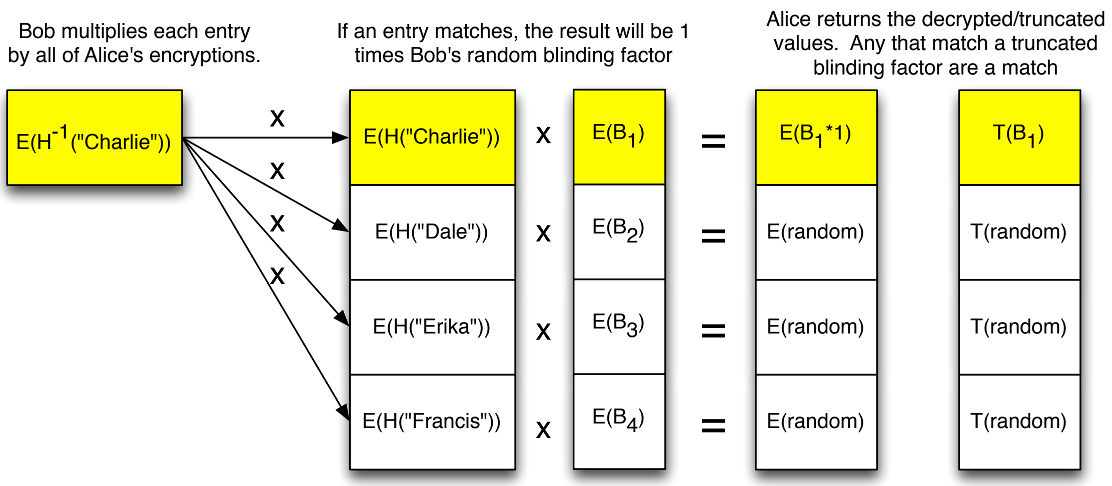
Bob multiplies each entry by all of Alice's encryptions.
If an entry matches, the result will be 1 times Bob's random blinding factor
Alice returns the decrypted/truncated values. Any that match a truncated blinding factor are a match
E(H^-1("Charlie"))
X
E(H("Charlie"))
E(H("Dale"))
E(H("Erika"))
E(H("Francis"))
E(B1)
E(B2)
E(B3)
E(B4)
=
E(B1*1)
E(random)
T(B1)
T(random)

Figure 4. Social Graph protocol: For each entry in Bob's array, he homomorphically multiplies it by every entry in Alice's array. He then multiplies in a unique blinding factor for all of the resulting values. Alice receives these values, decrypts them, and truncates all but the least significant t bits, which she returns. If these t bits match the least significant t bits of the blinding factor multiplied in that entry, Bob knows there is a match.

each entry in the array of m_1, we know that:

$$E(f(x,y)) \overset{c}{\approx} E(1) \overset{c}{\approx} E(2) \tag{8}$$

Based on the semantic security of our encryption scheme, each encryption of the result is indistinguishable from the two possible entries for each cell in a real execution. The second part of m_1, the public key, is trivially indistinguishable, as both the simulator and Alice would randomly generate a key pair.

The second message is again trivially shown to be indistinguishable, as both Alice's message and the simulator's message are $f(x,y)$.

Therefore, the Millionaire's Problem protocol is secure when Bob is corrupt. ■

2) Information Revealed: Having demonstrated that both parties only receive the output of the function $f(x,y)$, we now quantify the probability of either party learning the other party's input from $f(x,y)$. Given an instance of the protocol, with parameters n (the maximum input), x, and y (the inputs of both parties), the resulting boolean value allows each party to guess the other party's input with the following probability: for the wealthier party W, without loss of generality, the probability of guessing the other party's input is $\frac{1}{x}$, where x is W's input. For the less wealthy party L, the probability of guessing the other party's input is $\frac{1}{n-y}$, where y is L's input.

C. Location Privacy

1) Protocol Security:

Theorem 2. *Location Privacy: Assuming the encryption scheme used in the Location-Based Messaging protocol is semantically secure, the Location-Based Messaging protocol is secure in the presence of semi-honest adversaries.*

Proof: Again, we prove the security of the Location-Based Messaging protocol separately for each participant.

Alice is corrupt: For the Location-Based Messaging protocol, we define $f(x,y)$ as follows: given inputs (x,y) that are the grid locations where Alice and Bob wish to send and receive messages, $f(x,y)$ is a shuffled set of size c grid locations which, if an intersection exists, contains the intersecting grid locations. The variable c is defined as the size of the area where Bob will be sending messages. We now prove that a simulator S_A operating in the ideal world can simulate the view of Alice, our semi-honest aborting adversary in the real world. S_A is constructed as follows:

S_A is given input $(x, f(x,y))$, where x is the grid locations where Alice is willing to receive messages, and $f(x,y)$ is as defined above. Upon receiving Alice's initial message containing her encrypted location matrix L_A, S_A encrypts each element in $f(x,y)$ with Alice's public key and returns $m_1 = E(f(x,y))$, completing Alice's view of the interaction.

We now show that S_A's message is indistinguishable from Alice's view of a real protocol execution. First, we know that for any encrypted element in $f(x,y)$ at location i:

$$E(f(x,y)_i) \overset{c}{\approx} E(1*1) \overset{c}{\approx} E(1*2) \tag{9}$$

Based on the semantic security of our encryption scheme, each encryption of an entry in $f(x,y)$ is indistinguishable from an encryption of the result of Bob's homomorphic operations. Second, we know trivially that the decrypted values in m_1 are identical to Bob's message in a real interaction, as both simply contain the values in $f(x,y)$.

Therefore, the Location-Based Messaging protocol is secure when Alice is corrupt.

Bob is corrupt: We construct the simulator S_B as follows: S_B is given $(y, f(x,y))$, where y the grid locations where Bob will be sending messages, and $f(x,y)$ is the result as defined above. S_B prepares a location matrix L_A by storing the values in $f(x,y)$ in the grid locations in y. For any grid location not in y, S_B stores a '1'. S_B then generates an encryption key pair and encrypts each entry of L_A under the public key PK_{S_B}. It then sends the message $m_1 = E(L_A)||PK_{S_B}$. When S_B receives the shuffled array R containing the result of Bob's homomorphic operations, S_B simply decrypts each entry in R using SK_{S_B} and returns $m_2 = D(R)$.

We show that Bob's view in a real execution of the protocol and an interaction with S_B are indistinguishable. For each element of $f(x,y)$ and for each entry in the matrix L_A:

$$E(f(x,y)_i) \overset{c}{\approx} E(1) \overset{c}{\approx} E(2) \tag{10}$$

Based on the semantic security of the encryption scheme, Bob cannot distinguish between encryptions of the resulting values and encryptions of the values Alice would encrypt in a real interaction. The public key PK_{S_B}, being randomly generated, is clearly indistinguishable from Alice's public key.

The second message is clearly indistinguishable, since in both the real view and the simulated view, the message contains a randomly ordered array containing the values of $f(x,y)$.

Therefore, the Location-Based Messaging protocol is secure when Bob is corrupt. ■

2) Information Revealed: We now show the probability of either participant guessing the exact location of the other participant. Given Alice and Bob's areas of willingness to send/receive messages x, y, the number of entries in the location matrix n, and the result of the protocol $f(x,y)$, both parties know the size of the intersection between their send/receive areas as well as the number of cells in Bob's sending area. In the worst case, either all of Bob's sending area is contained within Alice's receiving area or vice versa. Consider if Bob's area is contained within Alice's, without loss of generality. Alice has probability $\frac{1}{|x|}$ of guessing which cell contains Bob's actual location, and Bob has probability $\frac{1}{n}$ of guessing Alice's location since he does not know the size or shape of Alice's receiving area outside of his own sending area. Thus, given $g = max(|x|, |y|)$, the upper bound on the probability of guessing the other party's location is $\frac{1}{g}$.

D. Social Graph Privacy

1) Protocol Security:

Theorem 3. *Social Graph Privacy: Assuming the encryption scheme used in the Social Graph Protocol is semantically secure and that the secure hash function used is pseudorandom and one-way, the Social Graph protocol is secure in the presence of semi-honest adversaries.*

Proof: Again, we prove separately the security of our protocol when Alice and Bob are corrupt.

Alice is corrupt: For the Social Graph protocol, we define $f(x,y)$ as follows: given inputs (x,y), which are the list of friends for Alice and Bob respectively, $f(x,y) = x \cap y$. We now construct a simulator S_A in the ideal world that, given x and $f(x,y)$, is capable of simulating Alice's view of a real interaction:

S_A is given $(x, f(x,y))$ as defined above. Upon receiving Alice's initial message containing an array of encrypted friend hashes, S_A generates an array of n^2 random blinding factors $B_1, ..., B_{n^2} \in G$, encrypts them with Alice's public key, and returns message $m_1 = E(B)$. Upon receiving Alice's second message, the decryption and truncation of m_1, S_A responds with message $m_2 = f(x,y)$.

We now show that both of S_A's messages are indistinguishable from their counterparts in Alice's view of a real execution. For the first message, we again make two points. First, we know that, for $i = 1...n^2$ and friend names x, y:

$$E(B_i) \overset{c}{\approx} E(H(x) * H^{-1}(y) * B_i) \tag{11}$$

Based on the semantic security of the encryption system, S_A's encryption of the blinding factor is indistinguishable from Bob's result gained by homomorphic multiplication. Second, we know for any two names x, y:

$$B_i \overset{s}{\approx} H(x) * H^{-1}(y) * B_i \tag{12}$$

Based on the definition of statistical indistinguishability, the result Alice sees in both a simulated and a real interaction appears random.

Therefore, the Social Graph protocol is secure when Alice is corrupt.

Bob is corrupt: We construct simulator S_B as follows: S_B is given $(y, f(x,y))$. S_B begins by generating a random key K_h for the agreed symmetric hash, a random public key encryption pair PK_{S_B}, SK_{S_B}, and an array F of the pre-defined length n. S_B then enters the hashes of the names in $f(x,y)$ in uniformly random indices of F. For the remaining unfilled entries, S_B generates random numbers from the range of the hash. Bob then encrypts each entry with PK_{S_B} and sends $m_1 = E(F)||PK_{S_B}||K_h$. Upon receiving Bob's response R, S_B decrypts the response with SK_{S_B} and returns m_2, which is the last t bits of each entry in $D(R)$, where t is the predetermined security parameter for the length of the resulting values.

We show that Bob's view in a real execution of the protocol and an interaction with S_B are indistinguishable. Considering m_1, for each entry in F and for each hashed name x, given a uniformly random group element r:

$$E(r) \overset{c}{\approx} E(x) \tag{13}$$

Based on the semantic security of our encryption scheme, Bob cannot distinguish between an encryption of a hashed name and an encryption of a random number. Again, since S_B and Alice both generate their hash keys and public keys randomly, they are trivially indistinguishable.

Considering m_2, we see that, given a random key, for any two names x, y:

$$H(x) \overset{c}{\approx} \mathcal{U} \overset{c}{\approx} H(y) \tag{14}$$

Based on the pseudorandomness of the hash function, any hashes produced will be indistinguishable from randomness. Because of this, these samples will remain indistinguishable after any polynomial time operations are performed over them. Therefore, for any non-matching entry in m_2 and any names x, y, z, where x is in Bob's friend list, y is in Alice's, and z is randomly chosen by S_B:

$$T(H(x) * H^{-1}(y)) \overset{c}{\approx} T(H(x) * H^{-1}(z)) \tag{15}$$

Where $T()$ is the truncation function.

Therefore, the Social Graph protocol is secure when Bob is corrupt. ■

2) Information Revealed: We now show the probability of either party guessing the input of the other party. For an instance of the protocol we are given inputs x, y that are the inputs of both parties (without loss of generality), some number t which is the number of bits returned by Alice in the third message, some number n which is the security parameter of the hash function being used, and $f(x, y)$ which is the intersection of these two lists. Alice receives only the list of names in $f(x, y)$, so the probability of her guessing any friend in Bob's friend list outside of this intersection is random selection over all possible friends. Bob receives the last t bits of Alice's hash values multiplied by the inverse of his own hash values. To reverse this value back into the name of one of Alice's friends, Bob must examine all possible values for the $n - t$ truncated bits, and must then reverse the hash of Alice's friend's name. Therefore, Bob's ability to learn any of Alice's friends not in the intersection is no better than random over the possible truncated values and the possible hash pre images, $\frac{1}{(n-t)^2 * n^2}$.

We do note a slight difference in the security guarantees provided by our protocol and the guarantees of garbled circuits. While garbled circuit constructions keep all data cryptographically secured by the garbling function, our scheme reveals to Bob t bits of each of Alice's hashed inputs multiplied by the multiplicative inverse of Bob's hashed inputs. However, based on the pseudorandomness and one-wayness of a secure keyed hash, we maintain that in practice it is still computationally infeasible to recover the remaining bits of the hashed value *and* reverse the hashed value to the exact preimage that generated it.

VI. Performance Analysis

We have developed new protocols for solving specific two-party computation problems and proven the privacy guarantees of each protocol. However, the primary benefit of these protocols is the efficiency they demonstrate over the current model for two-party computation, garbled circuits. To rigorously demonstrate this performance improvement, we implemented our example applications on three platforms: using the EMOC technique in C, the Secure Function Definition Language (SFDL) used by Fairplay, and the pipelined circuit evaluation technique developed by Huang et al. [7]. We chose these two garbled circuit optimizations because they represent two of the most efficient optimizations currently available. In addition, the performance of OBDD compiled circuits relative to pipelined circuits has never been evaluated. Because the optimizations used in Henecka's work are largely done by hand on a per-application basis, we chose not to evaluate the TASTY framework [1]. Our goal in these evaluations is to provide an *objective comparison* against general-purpose garbled circuit compilation techniques.

In our implementations, we used ElGamal [32] as a multiplicative homomorphic, public-key encryption scheme and HMAC-md5 as a secure keyed hash function. In ElGamal, we use a 1024-bit modulus and exponent keys of 160 bits, which is common secure practice with this scheme [33]. For our C implementations, we cross-compiled each program using the Android Toolchain [34], allowing us to execute our tests on the ARM processor in our mobile device. To compile and test the Fairplay versions of our applications, we first ported the Fairplay runtime environment, written in Java, into an Android application. For our first experiment, we compiled the garbled circuits used for evaluation using the standard Fairplay compiler as well as an SFDL compiler using Ordered Binary Decision Diagrams (OBDD) [6]. The OBDD compiler creates more efficient circuits than the standard compiler for a large number of functions, and still executes in the Fairplay runtime environment. To test the pipelined circuit technique, we used the platform code built by Huang in Java for the Android OS.

Our performance evaluations were run on an HTC Nexus S containing a 1 GHz processor and 512 MB of RAM. The operating system used was Android 2.3. All execution times given are averaged over at least ten executions and shown with a confidence interval of 95%.

A. Experiment Setup

To provide the most accurate comparison to other two-party computation techniques, we broke our experiments into two groups. In the first group of experiments, we compared our technique to Fairplay, using circuits compiled with the

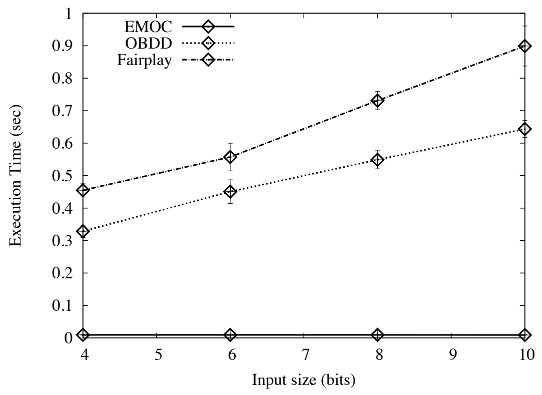

Figure 5. Millionaire's Problem online execution times versus Fairplay. Note that for all input sizes, our application runs in a fixed amount of time while both circuits compiled for Fairplay show increasing execution times with increasing input size.

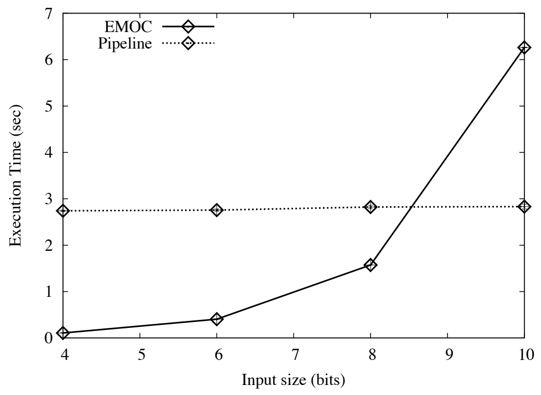

Figure 6. Millionaire's Problem total execution times versus pipelined circuits. Note that the pipelined circuits outperformed EMOC for one input size due to the simplicity of the function being evaluated. The majority of the EMOC execution time can be amortized through preprocessing.

Fairplay compiler and the OBDD compiler. In both our applications and Fairplay, we divided the execution into a preprocessing phase (for circuit generation in Fairplay and encrypted query generation in EMOC), and online execution. Since both the Fairplay compiler or the OBDD compiler required too much memory to be executed on the mobile device, we compiled circuits on a desktop machine and compared only the online execution times of both techniques on the mobile device. The preprocessing required by EMOC, generating encrypted query values, can be amortized over spare clock cycles prior to the initiation of computation. To minimize the use of battery power, these operations could be set to only execute while the phone is charging. The encrypted values generated during preprocessing, which are approximately 3 KB each, could easily be stored until needed. Thus, the majority of our preprocessing time could be reduced to the time required to populate an array with stored values.

In our second group of experiments, we compared our technique to the pipelined circuit technique developed by Huang et al. This technique splits circuit generation into layers, generating and evaluating one layer of circuits at a time during the execution of the protocol. As such, there is no logical separation of preprocessing and online computation. Therefore, when comparing EMOC to the pipelined circuits, we compared the entire execution time. It is relevant to note that, although these comparisons do not reflect it, EMOC is still capable of amortizing preprocessing time, allowing users to execute *only the online computation in practice.*

B. The Millionaire's Problem

To accurately compare all techniques, we built each application to execute the millionaire's problem given inputs within the range $1...N$. In both Fairplay and pipelined circuits, this is represented by an input of $log_2(N)$ bits. In EMOC, this is represented by query table of length N. In our first comparison, between Fairplay and EMOC, the execution time of our protocol outperformed circuits compiled both with the Fairplay compiler and the OBDD compiler, as seen in Figure 5. Even for the smallest tested input size, $N = 16$, our protocol ran in 0.009 seconds($\pm$0.0001 seconds), while the OBDD-compiled circuits ran in 0.33 seconds ($\pm$0.0191 seconds), a 97.27% reduction, and the Fairplay-compiled circuits ran in 0.45 seconds ($\pm$0.0202 seconds), a 98.00% improvement in execution time. As input sizes increased, the time to evaluate both Fairplay and OBDD circuits increased as well, while the single multiplication required in EMOC allowed for a constant evaluation time across all input sizes.

In our second comparison with pipelined circuits, EMOC outperformed the pipelined circuits for all test input sizes except for the largest, $N = 1024$. For $N = 16$, EMOC demonstrated a total execution of only 0.11 seconds ($\pm$0.0006 seconds), a 95.93% reduction from pipelined circuits, which took 2.7 seconds ($\pm$0.0169 seconds). However, for the largest input, $N = 1024$, EMOC ran in 6.26 seconds ($\pm$0.0068 seconds), while pipelined circuits ran in 2.8 seconds ($\pm$0.0514 seconds). The reason for this discrepancy is in the representation of the inputs. The input for pipelined circuits is simply a 1024-bit integer, while EMOC requires 1024 encrypted integers. It is also important to recall that the majority of the EMOC execution time for this application can potentially be amortized through preprocessing, while the pipelined circuit requires that the circuits be generated at execution time. We demonstrate with the next two applications that even a small increase in the complexity of the function results in a *significant increase* in the execution time, while the execution times of EMOC remain within a feasible range for practical use.

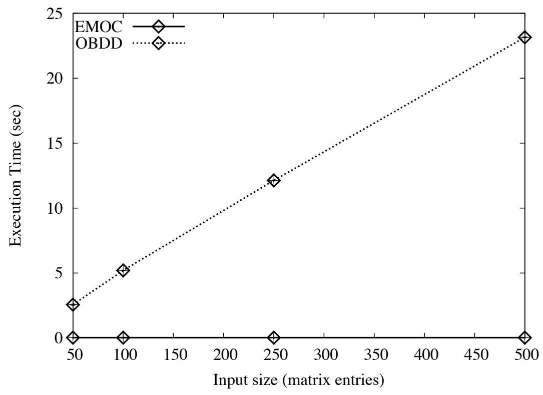

Figure 7. Location-Based Messaging online execution times versus Fairplay. Note that, as with the Millionaire's Problem, for all input sizes our application runs in a fixed amount of time while the OBDD circuits show increasing execution times with increasing input size.

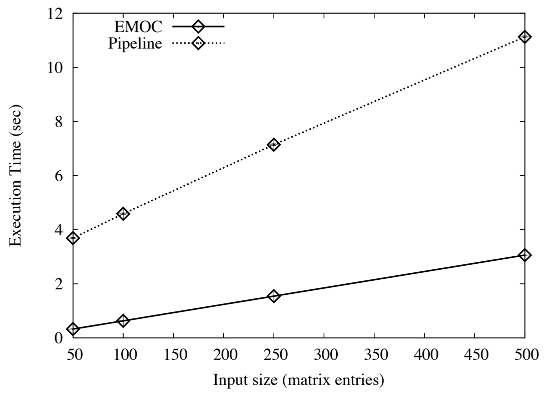

Figure 8. Location-Based Messaging total execution times versus pipelined circuits. Note that for a more complex function, the pipelined circuits are less efficient than EMOC. Even here, the majority of the EMOC execution time can be amortized through preprocessing.

C. Location-Based Messaging

For the Location-Based messaging program, we examined performance over two different input size variables. The first variable is performance as the size of the grid increases, where inputs are represented as an $M \times N$ matrix. The second variable is performance as the size of Alice and Bob's query areas increase, where the overall size of the location matrix is a fixed-sized $M \times N$ matrix and the query areas are represented as Q grid locations within the location matrix. In our comparison with Fairplay, the improvement of EMOC was apparent even before running our experiments. When compiling circuits using the Fairplay compiler, we found that even on a desktop machine, the standard Fairplay compiler was unable to compile this application for inputs larger than approximately $M \times N = 20$. Thus, our Fairplay performance evaluation only used circuits compiled with OBDD. For an $M \times N = 500$ grid, EMOC executed the online phase in 0.03 seconds ($\pm$0.0002 seconds), while the OBDD-compiled circuit took 23.15 seconds ($\pm$0.0351 seconds), a 99.87% improvement on our part. Again, as in the Millionaire's Problem, our execution times remain constant when the query areas of Alice and Bob remain constant. When we examined increasing the query area size, EMOC remained the most efficient choice. Even for query areas as large as 100 entries, EMOC executed in 0.17 seconds ($\pm$0.0061 seconds), while the OBDD-compiled circuit took a constant 12.13 seconds ($\pm$0.0424 seconds) for all query area sizes.

In our second comparison, the performance advantage of EMOC became apparent. For the largest examined input of size $M \times N = 500$, EMOC performed the total setup and execution in 3.06 seconds ($\pm$0.0071 seconds), while pipelined circuits ran in 11.13 seconds ($\pm$0.0332 seconds), a 72.51% reduction. As above, when we compare the execution for variable query sizes, EMOC again outperformed pipelined circuits by 76.33%, running in 1.69 seconds ($\pm$0.0061 seconds) compared to the pipelined circuits time of 7.14 seconds ($\pm$0.0330 seconds) From these results, it is clear that for even slightly more complex functions, EMOC provides the more efficient solution.

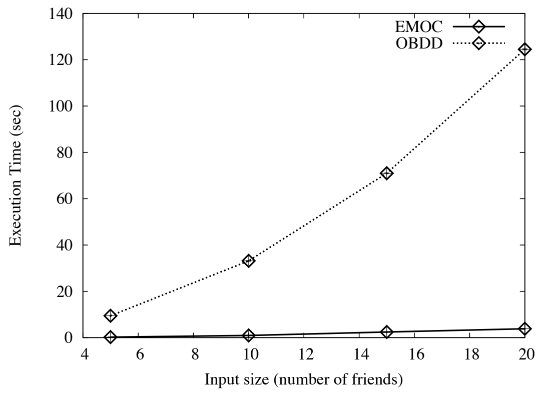

Figure 9. Social Graph Privacy online execution times versus Fairplay. We were only able to run experiments up to inputs of size 20 due to the large memory requirements of Fairplay.

D. Social Graph Privacy

With the Social Graph Privacy application, we consider two parameters when comparing applications. The first parameter, which we vary within our experiments, is the size of the friend lists being compared, represented in every application as the size of the input array. The second parameter, which we fix for each application, is the number of unique members that can be represented within the social network. EMOC uses a secure 128-bit hash taken from the string representation of an individual's name, allowing 2^{128} different unique identifiers. For OBDD-compiled circuits,

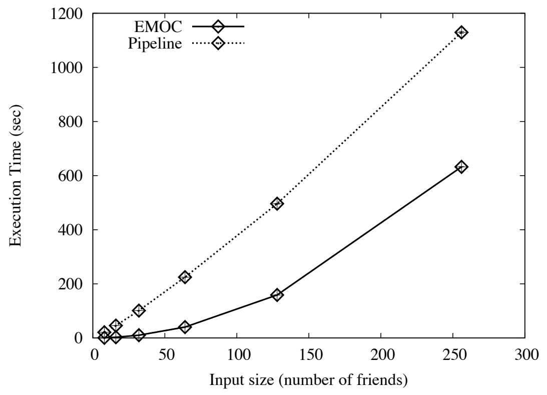

Figure 10. Social Graph Privacy total execution times versus pipelined circuits. Note that input sizes are a more practical number than with Fairplay.

we use an 8-bit identifier for each member of the social network because, as we demonstrate later, circuits evaluating over larger identifiers could not be compiled. For pipelined circuits, we used a 30-bit hash to represent each member of the social network. Because the security of pipelined circuits does not depend on the security of this hash, we felt it was unnecessary to use a full 128-bit hash when performing these evaluations. However, taking Facebook as an example social network, we wanted to be able to uniquely represent at least this 700 million member population with unique hash values. Accordingly, our performance values differ from those generated by Huang et al. because we support larger inputs.

In our first comparison, we were again unable to compile circuits for any size input using the Fairplay compiler, so our experiments only examined OBDD-compiled circuits. Additionally, we were only able to execute circuits comparing friend lists of size $N = 20$ before the phone ran out of memory and crashed with each attempted execution. Even at this unrealistically small input size, the difference in execution times was significant, with EMOC executing in 3.84 seconds ($\pm$0.0034 seconds) and the OBDD-compiled circuit executing in 124.49 seconds ($\pm$0.2809 seconds), a 96.92% improvement.

To demonstrate the effectiveness of our application for practical input sizes, we compared EMOC to the pipelined circuits technique for input sizes up to $N = 256$. As expected, our application again demonstrated significantly improved performance over garbled circuits. At the largest input size, EMOC evaluated in 632.43 seconds ($\pm$0.3728 seconds), while pipelined circuits evaluated in 1129.58 seconds ($\pm$3.2322 seconds). If we drop the input size by half, EMOC executes in as little as 158.84 seconds ($\pm$0.0291 seconds), while pipelined circuits required 496.11 seconds ($\pm$1.2560 seconds), an improvement of 67.98%. These results clearly show that, for specific two-party computation problems, partially homomorphic encryption offers a practical technique for secure oblivious computation even on the restricted resources of the mobile platform.

VII. Conclusion

As mobile phones become more popular for computing tasks, new techniques will be needed to protect the private information used in many of their applications. Garbled circuit constructions effectively solve this problem in the desktop space, but are too processor and memory intensive to be practical on the mobile platform. By replacing garbled circuits with homomorphic encryption operations, our EMOC protocols demonstrate that certain privacy-preserving functions can be evaluated with great efficiency on the mobile platform. Through our performance evaluation, we demonstrate improvements greater than 68% over the most efficient garbled circuit constructions. Based on these results, we present our protocols as an efficient method for implementing provable privacy into some location-based and social networking applications.

Acknowledgments

This material is based on research sponsored by DARPA under agreement number FA8750-11-2-0211. The U.S. Government is authorized to reproduce and distribute reprints for Governmental purposes notwithstanding any copyright notation thereon. The views and conclusions contained herein are those of the authors and should not be interpreted as necessarily representing the official policies or endorsements, either expressed or implied, of DARPA or the U.S. Government.

References

[1] W. Henecka, S. Kögl, A.-r. Sadeghi, T. Schneider, and I. Wehrenberg, “TASTY : Tool for Automating Secure Two-partY computations,” in *Proceedings of the ACM conference on Computer and Communications Security (CCS)*, 2010.

[2] I. Constandache, X. Bao, M. Azizyan, and R. Choudhury, “Did you see Bob?: human localization using mobile phones,” in *Proceedings of the ACM International Conference on Mobile Computing and Networking (Mobicom)*, 2010.

[3] N. Banerjee, S. Agarwal, P. Bahl, and R. Chandra, “Virtual compass: relative positioning to sense mobile social interactions,” Microsoft Research, Tech. Rep., 2010. [Online]. Available: http://www.springerlink.com/index/K81H08U2767N2117.pdf

[4] A. C. Yao, “How to generate and exchange secrets,” in *Proceedings of the IEEE Symposium on Foundations of Computer Science (FOCS)*, 1986.

[5] D. Malkhi, N. Nisan, B. Pinkas, and Y. Sella, “Fairplay – A Secure Two-Party Computation System,” in *Proceedings of the USENIX Security Symposium (SECURITY)*, 2004.

[6] L. Kruger, S. Jha, E.-J. Goh, and D. Boneh, “Secure Function Evaluation with Ordered Binary Decision Diagrams,” in *Proceedings of the ACM conference on Computer and communications security (CCS)*, 2006.

[7] Y. Huang, D. Evans, J. Katz, and L. Malka, “Faster Secure Two-Party Computation Using Garbled Circuits,” in *Proceedings of the USENIX Security Symposium*, 2011.

[8] B. Mood, L. Letaw, and K. Butler, “Memory-Efficient Garbled Circuit Generation for Mobile Devices,” in *Proceedings of the 16th IFCA International Conference on Financial Cryptography and Data Security (FC 2012)*, Bonaire, Feb. 2012.

[9] Y. Huang, P. Chapman, and D. Evans, “Privacy-Preserving Applications on Smartphones,” in *Proceedings of the USENIX Workshop on Hot Topics in Security*, 2011.

[10] M. Osadchy, B. Pinkas, A. Jarrous, and B. Moskovich, “Scifi-a system for secure face identification,” in *Proceedings of the IEEE Symposium on Security & Privacy*, 2010.

[11] L. Sweeney, “k-Anonymity: A Model For Protecting Privacy,” *International Journal on Uncertainty, Fuzziness and Knowledge-based Systems*, vol. 10, no. 5, pp. 557–570, 2002.

[12] M. Gruteser and D. Grunwald, “Anonymous Usage of Location-Based Services Through Spatial and Temporal Cloaking,” in *Proceedings of the ACM International Conference on Mobile Systems, Applications and Services (MobiSys)*, 2003.

[13] B. Gedik and L. Liu, “Protecting Location Privacy with Personalized k-Anonymity: Architecture and Algorithms,” *IEEE Transactions on Mobile Computing*, vol. 7, pp. 1–18, 2008.

[14] T. Xu and Y. Cai, “Feeling-based Location Privacy Protection for Location-based Services,” in *Proceedings of the ACM Conference on Computer and Communications Security (CCS)*, 2009.

[15] B. Bamba, L. Liu, P. Pesti, and T. Wang, “Supporting Anonymous Location Queries in Mobile Environments with PrivacyGrid,” in *Proceedings of the International World Wide Web Conference (WWW)*, 2008.

[16] J. Meyerowitz and R. R. Choudhury, “Hiding Stars with Fireworks: Location Privacy through Camouflage,” in *Proceedings of the ACM International Conference on Mobile Computing and Networking (Mobicom)*, 2009.

[17] J. Shi, R. Zhang, Y. Liu, and Y. Zhang, “PriSense: Privacy-Preserving Data Aggregation in People-Centric Urban Sensing Systems,” in *Proceedings of the IEEE International Conference on Computer Communications (INFOCOM)*, 2010.

[18] D. Quercia, I. Leontiadis, L. McNamara, C. Mascolo, and J. Crowcroft, “SpotME If You Can: Randomized Responses for Location Obfuscation on Mobile Phones,” in *Proceedings of the Int’l Conference on Distributed Computing Systems (ICDCS)*, 2011.

[19] H. Haddadi, P. Hui, and I. Brown, “MobiAd: private and scalable mobile advertising,” in *Proceedings of the ACM International Workshop on Mobility in the Evolving Internet Architecture (MobiArch)*, 2010.

[20] K. P. N. Puttaswamy and B. Y. Zhao, “Preserving Privacy in Location-based Mobile Social Applications,” in *Proceedings of the ACM Workshop on Mobile Computing Systems & Applications (HotMobile)*, 2010.

[21] A. Narayanan, N. Thiagarajan, M. Lakhani, M. Hamburg, and D. Boneh, “Location Privacy via Private Proximity Testing,” in *Proceedings of the ISOC Network and Distributed Systems Security (NDSS) Symposium*, 2011.

[22] F. Kerschbaum, A. Schropfer, D. Dahlmeier, and D. Biswas, “On the Practical Importance of Communication Complexity for Secure Multi-Party Computation Protocols,” in *Proceedings of the ACM Symposium on Applied Computing (SAC)*, 2009.

[23] T. Nakamura, S. Inenaga, D. Ikeda, K. Baba, and H. Yasuura, “Anonymous Authentication Systems Based on Private Information Retrieval,” in *International Conference on Networked Digital Technologies (NDT)*, 2009.

[24] J. Bethencourt, D. Song, and B. Waters, “Analysis-Resistant Malware,” in *Proceedings of the ISOC Network and Distributed Systems Security (NDSS) Symposium*, 2008.

[25] R. Ostrovsky and W. E. S. III, “Private Searching On Streaming Data.” *Journal of Cryptology*, vol. 20, no. 4, pp. 397–430, 2007.

[26] M. Hirt and K. Sako, “Efficient Reciept-free voting based on homomorphic encryption,” in *Proceedings of the International Conference on Theory and Application of Cryptographic Techniques (EUROCRYPT)*, 2000.

[27] G. Zhong, I. Goldberg, and U. Hengartner, “Louis, Lester and Pierre: Three Protocols for Location Privacy,” in *Privacy Enhancing Technologies Symposium*, 2007.

[28] C. Gentry, “A Fully Homomorphic Encryption Scheme,” Ph.D. dissertation, Stanford University, Sep. 2009.

[29] A. Ramachandran, Z. Zhou, and D. Huang, “Computing Cryptographic Algorithms in Portable and Embedded Devices,” in *Proceedings of the IEEE International Conference on Portable Information Devices (PORTABLE)*, 2007.

[30] J. Katz and Y. Lindell, *Introduction to Modern Cryptography*. Chapman and Hall/CRC, 2007.

[31] Y. Lindell and B. Pinkas, “A Proof of Yao’s Protocol for Secure Two-Party Computation,” in *Journal of Cryptology*, vol. 22, no. 2, 2009, pp. 161–188.

[32] T. Elgamal, “A public key cryptosystem and a signature scheme based on discrete logarithms,” *IEEE Transactions on Information Theory*, vol. 31, no. 4, pp. 469–472, Jul. 1985.

[33] B. Schneier, *Applied Cryptography*, 2nd ed. New York, New York, USA: John Wiley & Sons, Inc., 1996.

[34] Google, “Android project,” http://source.android.com, 2010.