OPFE: Outsourcing Computation for Private Function Evaluation

Henry Carter[1] and Patrick Traynor[2]

[1] Georgia Institute of Technology
carterh@gatech.edu
[2] University of Florida
traynor@cise.ufl.edu

Abstract. Outsourcing secure multiparty computation (SMC) protocols has allowed resource-constrained devices to take advantage of these developing cryptographic primitives with great efficiency. While the existing constructions for outsourced SMC guarantee input and output privacy, they require that all parties know the function being evaluated. Thus, stronger security guarantees are necessary in applications where the function itself needs to be kept private. We develop the first linear-complexity protocols for outsourcing private function evaluation (PFE), a subset of SMC protocols that provide both input and function privacy. Assuming a semi-honest function holder, we build on the most efficient two-party PFE constructions to develop outsourced protocols that are secure against a semi-honest, covert, or malicious Cloud server and malicious mobile devices providing input to the function. Our protocols require minimal symmetric key operations and only two rounds of communication from the mobile participants. As a secondary contribution, we develop a technique for combining public and private sub-circuits in a single computation called partially-circuit private (PCP) garbling. This novel garbling technique allows us to apply auxiliary circuits to check for malicious behavior using only free-XOR overhead gates rather than the significantly more costly PFE gate construction. These protocols demonstrate the feasibility of outsourced PFE and provide a first step towards developing privacy-preserving applications for use in Cloud computing.

Keywords: private function evaluation, garbled circuits, server-assisted cryptography

1 Introduction

Secure multiparty computation research has moved a set of theoretically novel protocols into feasible use over the past ten years. Since it was originally developed [39], garbled circuit protocols have been designed to run computation that is secure against a variety of adversaries [2,15,16,25], and optimized for a wide array of programs and computation settings, such as batched computation [27,17] or stateful processing [30]. More recent work has shown that these complex cryptographic constructions can even be applied efficiently to mobile applications by securely outsourcing computation to the Cloud [21,7,6,8,19]. These protocols have been shown to allow mobile devices to participate in complex privacy-preserving computations such as face recognition and navigation.

Unfortunately, these protocols require that all parties have knowledge of the application being executed. In a variety of applications for government agencies and private companies, the function itself may be confidential or proprietary information that should not be revealed to the mobile device or the Cloud. To remedy this problem, several protocols for private function evaluation (PFE) have been developed to allow for secure computation that maintains both input privacy and function privacy. However, it is currently unknown how these protocols could be applied in the outsourced setting.

1.1 Our Contributions

In this work, we develop the first outsourced PFE protocols secure against semi-honest, covert, and malicious Cloud adversaries. These protocols allow an application server to execute a private function over inputs provided by a set of mobile devices that are aided by an untrusted Cloud server. To make these protocols more efficient, we developed a new technique for combining public garbled circuit computations with the privately evaluated function, which we term a "partially-circuit private" garbling technique.

Outsourced PFE Setting We develop a new setting for outsourcing private function evaluation against a variety of adversaries. We build our setting on the outsourced model for SMC developed in a line of recent work [21,7,6,8]. However, our definitions treat the function being evaluated as a private input of a single party in the computation. Our protocol setting is composed of at least three players.

APPLICATION: We define this party as the application server that is hosting the application and providing the private function to be evaluated. Our setting assumes that this party will always provide the function and may or may not provide input and receive output from the computation. In addition, we assume this party to be semi-honest for all protocols.
MOBILE: We define this party (these parties) as one (or more) computationally restricted device(s) that provide input to the application server and may receive some output from the computation. We develop protocols that are secure in the presence of semi-honest and malicious mobile devices.
CLOUD: Our final participant is the Cloud server that assists the mobile device(s) in executing the costly operations in our PFE protocols. Since mobile users may be accessing variable Cloud infrastructure in any particular application scenario, we seek to develop protocols that remain secure against any adversary (semi-honest, covert, and malicious).

There are two reasons why we model the function holder as a semi-honest adversary. First, recent work in the general PFE space has produced protocols to defend against malicious function holders [28]. However, the practicality of such a guarantee is not universally applicable. If the function holder receives output from the computation, even a protocol that is secure against a malicious function holder cannot prevent that party from providing a function that reveals other parties' inputs.

Second, this setting accurately models a large number of real-world applications. A plethora of smartphone applications are built with the architecture of a client running on the mobile device contacting an application server for information and data processing. While the user may trust the application to provide useful and correct functionality, he may wish to limit the application server's access to his data in the event that the application server is later compromised. Given these two parties that can be trusted to follow a PFE protocol, it is reasonable to model the CLOUD as the strongest possible adversary while modeling the application server as semi-honest. This allows the mobile device to use *any* publicly available computing infrastructure to assist with the computation, from high-profile Cloud services to public access library terminals. For these reasons, as the first step in this space we prioritize security against the CLOUD, and add guarantees against other adversaries as a secondary goal.

Outsourced PFE Protocols Our primary contribution is a set of outsourced PFE protocols with security guarantees against semi-honest, covert, and malicious Cloud adversaries. To achieve this, we build on the garbled circuit-based PFE scheme of Mohassel and Sadeghian [29] and generalize their protocol according to the garbling scheme framework developed by Bellare et al. [4]. Given this technique for garbling circuits obliviously, we construct an outsourced PFE protocol that is secure against a semi-honest application server and both semi-honest and covert Cloud providers. Furthermore, we show that our covert secure protocol can also defend against malicious mobile parties. To achieve this, we leverage cut-&-choose techniques used in existing outsourced garbled circuit protocols [2] to ensure that all parties follow the protocol and learn nothing about the function being evaluated or the other inputs to the computation. Our proofs of security demonstrate that the Mohassel-Sadeghian garbling scheme, while originally proven secure against semi-honest adversaries, can be extended into stronger adversary models.

While outsourced PFE is theoretically possible using an outsourced SMC protocol executing a universal circuit, our protocols provide the first approach with linear complexity in the size of the private function. Furthermore, our protocols require the mobile parties to perform operations that are only dependent on

the input and output lengths. These operations can be implemented in practice using fast, symmetric-key operations.

PCP Garbling Our outsourced PFE protocol for a covert Cloud adversary provides efficient security for a variety of applications. However, to extend our results to all possible Cloud adversaries, we construct a third outsourced PFE protocol that is secure against a malicious Cloud and malicious mobile devices. Our chief technical innovation to make this protocol possible is the development of a partially-circuit private (PCP) garbling scheme, a technique which allows us to stitch together public and private functions in a single computation by alternating garbling techniques. This allows us to efficiently add public preprocessing input checks and post-processing output preparation using gate optimizations such as free XOR [23] and garbled row reduction [34] that were previously not applicable in the PFE setting. Using PCP garbling, we apply the input consistency checks developed by shelat and Shen [37] at the added overhead cost of $O(\lambda kn)$ free-XOR gates[3] rather than implementing the same checks inside the private function using $4O(\lambda kn)$ NAND gates without gate optimizations. In addition, this technique allows all parties to verify the correct construction of these consistency checks rather than just the party providing the private function as input. This is a critical first step towards strengthening our protocol against a more powerful function holder. While we apply this technique expressly for the purpose of adding consistency checks, the garbling technique stands as an independent contribution, as it is useful in practice to provide arbitrary public preprocessing and post-processing for any garbled circuit-based PFE protocol.

1.2 Organization

The remainder of this work is organized as follows: Section 2 provides an overview of related research; Section 3 describes the underlying techniques, security definitions, and notation used in our work; Section 4 defines our semi-honest secure protocol; Section 5 defines our covert secure protocol; Section 6 describes our novel PCP garbling construction; Section 7 applies our PCP garbling to define a malicious secure outsourcing protocol; and Section 8 provides concluding discussion.

2 Related Work

When secure multiparty computation (SMC) was first demonstrated in theory by Yao [39], it was largely believed to be infeasible in practical applications. However, recent developments and optimizations have shown that SMC can be used for a growing number of practical applications. While SMC protocols using homomorphic encryption [10,9] and oblivious transfers [12,31] have been shown to provide optimal performance in some cases, the garbled circuit construction has produced numerous protocols that are, in general, the most efficient SMC constructions for two-party computation [15]. These protocols have been developed for security against covert [2] and malicious [25,37,14] adversaries, as well as more specialized settings [16,17,27]. However, these protocols provide only one type of privacy guarantee: input privacy. This means that all parties involved know the function being evaluated. Because of the lack of function privacy, these protocols cannot be applied to applications where the function itself must remain hidden.

To provide function privacy, a specialized set of SMC protocols have been developed for private function evaluation (PFE). These protocols are designed to provide both input privacy and function privacy. The first construction to provide both of these privacy guarantees used existing SMC constructions to evaluate a universal circuit that accepts the desired function as an input parameter [38,1,24,35]. Because the specific function being evaluated is treated as an input to the universal circuit, the input privacy of the SMC

[3] Here, λ is the cut-&-choose statistical security parameter, k is a computational security parameter, and n is the bit length of the input.

protocol protects both the inputs and the function. However, the use of a universal circuit incurs significant overhead in the circuit size beyond the size of the function being computed. To provide a more efficient solution, many specialized PFE protocols were developed to provide function privacy for certain classes of functions [18,5,3,33]. The first general PFE protocol to provide linear asymptotic efficiency with respect to circuit size was developed by Katz and Malka [22]. Using techniques reminiscent of the LEGO SMC construction [32], they show how a generating party can generate a set of garbled gates that are then assembled and evaluated by the party possessing the function to be computed. In more recent work, Mohassel et al. [29] demonstrated how this idea can be generalized for various underlying SMC primitives, and later extended this concept for secret sharing SMC protocols to be secure against active adversaries [28]. However, all of these protocols assume a significant amount of computational power for all parties participating in the protocol.

To allow resource-constrained devices such as smartphones to participate efficiently in SMC protocols, several outsourcing techniques have been developed to securely offload the most costly operations to a Cloud server. The protocols developed by Kamara et al. [20,21] and Carter et al. [7,6] provide techniques for outsourcing both garbled circuit generation and evaluation in a variety of threat models. More recently, two works by Jakobsen et al. [19] and Carter et al. [8] produced protocols for outsourcing arbitrary SMC protocols in a black box manner. While these protocols provide an efficient means for mobile devices to take advantage of SMC input privacy, it is unclear how they could be expanded to provide function privacy. Our work develops a new setting and definitions of security for outsourcing PFE, as well as define the first protocols to allow resource-constrained mobile devices to execute these specialized SMC techniques.

3 Background

Our protocols combine a range of underlying SMC techniques for circuit garbling and for ensuring that adversarial behavior is caught during protocol execution. Here we provide a summary of these underlying constructions, and refer the reader to the original works for further discussion.

3.1 Garbling Scheme Definitions

To allow for a general approach to proving security in garbled circuit protocols, Bellare et al. [4] developed a cryptographic definition for a garbling scheme as well as several notions of security that are necessary to provide security in SMC protocols. They define a garbling scheme as a five-tuple $\mathcal{G} = (Gb, En, De, Ev, ev)$. Given the description f of a function that we wish to compute securely, the function $ev(f, \cdot) : \{0,1\}^n \rightarrow \{0,1\}^m$ executes f. $Gb(1^k, f) \rightarrow (F, e, d)$ is a probabilistic function that takes a function and security parameter as input and produces a garbled representation F, and the strings e and d which are used to describe the encode function $En(e, \cdot)$ and decode function $De(d, \cdot)$. These functions are used to encode the input to the garbled circuit, and decode the output after the garbled circuit has been evaluated. Finally, $Ev(F, \cdot)$ evaluates the garbled circuit with an encoded input. Correctness for the scheme is guaranteed by the requirement that for any $x \in \{0,1\}^{f.n}$ and $(F, e, d) \in [Gb(1^k, f)]$, then $ev(f, x) = De(d, Ev(F, En(e, x)))$.

Many practical garbled circuit protocols also require that a garbling scheme has a projective property. Bellare et al. define this property as follows: if, for all $x, x' \in \{0,1\}^{f.n}, k \in \mathbb{N}$, and $i \in [1, ..., n]$, when $(F, e, d) \in [Gb(1^k, f)], X' = En(e, x')$, and $X = En(e, x)$, then $X = (X_1, ..., X_n)$ and $X' = (X'_1, ..., X'_n)$ are n vectors, $|X_i| = |X'_i|$, and $X_i = X'_i$ if x and x' have the same i^{th} bit. For our partially-circuit private garbling scheme, all garbling schemes used must be projective garbling schemes.

In addition, Bellare et al. define a side-information function $\Phi(f)$ that defines what information is revealed about the function f by the garbled representation F. For the purpose of our protocols, we specify two specific side functions from their work. For a function f, $\Phi_{size}(f) = (n, m, q)$ reveals the size of the

circuit as having n input bits, m output bits, and q gates. If the circuit does not hide any information about the function, we say that $\Phi_{circ}(f) = f$.

Finally, Bellare et al. develop five different definitions of security for garbled circuits, providing three distinct guarantees: privacy (when the evaluating party receives output from the function), obliviousness (when the evaluating party does not receive output), and authenticity (to ensure that the evaluating party cannot tamper with the results of the computation). Our protocols require a garbling scheme to provide privacy under the prv.sim definition of privacy. This game-based definition specifies a game PrvSim that is given a garbling scheme, an adversary, a side-information function, and a security parameter. The adversary is allowed a single garbling query for a function f and input x of its choosing. The adversary is returned the tuple (F, X, d), where this tuple is, with uniform probability, either prepared according to the real garbling scheme $\mathcal{G}$, or is produced by a simulator $\mathcal{S}(1^k, y, \Phi(f))$ that only has access to the output $y \leftarrow ev(f, x)$ and the side-information function $\Phi(f)$. A garbling scheme is said to be prv.sim secure over a side information function $\Phi(f)$ if for every polynomial time adversary $\mathcal{B}$, there is a polynomial time simulator $\mathcal{S}$ such that the advantage of the adversary:

$$Adv_{\mathcal{G}}^{prv.sim,\Phi,\mathcal{S}}(\mathcal{B}, k) = 2Pr[PrvSim_{\mathcal{G},\Phi,\mathcal{S}}^{\mathcal{B}}(k)] - 1$$

is negligible.

3.2 PFE with Garbled Circuits

Mohassel and Sadeghian [29] define a framework for general PFE by breaking the protocol into two generic steps: hiding circuit topology and evaluating a single gate in a private manner. Using this framework, they develop a two-party, constant-round protocol for performing both steps using Yao's garbled circuits. Their construction requires a linear number of operations with respect to the size of the circuit being evaluated and is secure against semi-honest adversaries. Their protocol specifies the circuit generator as the party providing input, while the circuit evaluator provides the circuit to be computed and optionally an additional input value. To maintain privacy of the circuit topology, their protocol begins with the generating party defining random wire labels for every wire in the circuit. Then, using a construction termed the oblivious extended permutation (OEP), the evaluating party defines which gate output wires are connected as inputs to subsequent gates using blinding values to prevent the generating party from learning these mappings. Once the generating party receives the blinded input wires for each gate, he garbles the gates and returns the garbled circuit to the evaluator. The evaluator then evaluates the circuit following Yao's protocol by applying the blinding value at each gate to recover the correct output wire value. Rather than developing a special protocol for private gate evaluation, they simply define all functions using NAND gates, so that the generator cannot infer any information from the functionality of the gates.

To implement the OEP functionality, they define two constructions. The first uses an oblivious switching network (OSN) to blind and permute the wire values in an offline/online protocol. The second construction uses partially homomorphic encryption to allow the evaluator to add in blinding values for each input wire while not learning the actual garbled wire value. We build our protocol on the simpler construction using homomorphic encryption because it more readily allows for the circuit to be checked during our cut-&-choose, which is necessary for achieving security against any adversary stronger than a semi-honest adversary. However, if a cut-&-choose protocol were developed to commit and check the oblivious switching network and the oblivious transfers required to exchange the blinded inputs, our protocols could take advantage of the efficiency gains provided by using the OSN OEP construction.

For reference, Figures 1 and 2 define the semi-honest PFE scheme developed by Mohassel and Sadeghian [29] in the generalized garbled circuit notation developed by Bellare et al. [4]. For ease of comparison to our work, we refer to the garbling party as CLOUD and the evaluating party as APPLICATION. As an additional contribution, we prove the protocol is prv.sim secure in Appendix A. Given the Gb and CTH functions, the

Common inputs: a computational security parameter k and the side information function $\Phi_{size}(C) = (n, m, q)$ for APPLICATION's private function C. Here, n is the number of input wires, m is the number of output wires, and q is the number of gates in C.
Private inputs: APPLICATION inputs a circuit C that corresponds to her private function. CLOUD inputs a random seed r.
Outputs: APPLICATION receives the garbled function G, a vector of blinding values B, and a vector d that describes the decoding function $De(d, \cdot)$ that recovers the final output from the garbled output.

1. Using r to seed a pseudorandom generator, CLOUD generates $n + q$ sets of topologically ordered wire labels w_i^0, w_i^1, where the indices $i \in [1, ..., n]$ correspond to input wires, $i \in [n + 1, ..., n + q - m]$ correspond to gate output wires that are input to other gates, and $i \in [n + q - m + 1, ..., n + q]$, and circuit output wires.
2. CLOUD generates a random permutation string v from r where $|v| = n + q$ and permutes the wire labels as $w_i^{v_i}, w_i^{\overline{v}_i}$. He then appends each permutation bit to the wire label as $w_i^{v_i}||v_i, w_i^{\overline{v}_i}||v_i$.
3. CLOUD inputs $w_i^{v_i}||v_i, w_i^{\overline{v}_i}||v_i$ for $i \in [1, ..., n + q - m]$ into the $CTH(\cdot)$ functionality, while APPLICATION inputs the circuit C. CLOUD receives a vector of blinded input wires bw_j^0, bw_j^1 for $j \in [1, ..., 2q]$ that are topologically ordered input wires for the gates of C.
4. Given the ordered and blinded gate input wires bw_j^0, bw_j^1 for $j \in [1, ..., 2q]$ and the ordered gate output wires w_i^0, w_i^1 for $i \in [n + 1, ..., n + q]$, CLOUD garbles each gate following Yao's garbling protocol and using the NAND functionality for all gates.
5. CLOUD sends the garbled gates G and the hash of the output wire mappings $d = (H(w_i^0), H(w_i^1))$ for $i \in [n + q - m + 1, ..., n + q]$ to APPLICATION. The vector of blinding values B is retained by APPLICATION from the $CTH(\cdot)$ functionality.

Fig. 1. The private function garbling protocol by Mohassel and Sadeghian

remaining functions in the garbling scheme En, De, Ev, ev are trivially realized. En and De simply consist of mapping a real bit value b to a garbled wire value w^b and vice versa. Ev corresponds to the standard Yao evaluation protocol, with the exception that APPLICATION adds the appropriate blind (based on the permutation vector v') back to the previous output key before evaluating the next gate. ev is simply the function f that corresponds to the circuit C.

Intuitively, the combination of the OEP functionality and the use of NAND gates prevents the generator from learning anything about the circuit beyond the size and the number of input and output bits. The use of the homomorphic blinding or OSN prevents the evaluator from learning the garbled wire values, which preserves the privacy of the generator's inputs based on the guarantees of the garbled circuit construction. However, given a malicious generator, the security of the scheme does not hold. To protect against a malicious or covert adversary, a PFE protocol must handle the same malicious scenarios as any garbled circuit SMC protocol (i.e., ensure that the circuit is garbled consistently and that any oblivious transfers deliver correct input wire labels). In addition, we must ensure that a malicious generator cannot learn anything about the function during the partially homomorphic OEP protocol.

3.3 Outsourcing SMC Techniques

To develop an outsourced PFE protocol, we examine techniques from previous outsourced SMC protocols. However, many of these techniques do not directly apply to the PFE setting. For example, recent protocol developments have shown that any SMC protocol can be transformed into an outsourced protocol using black box techniques [19,8]. These protocols work by adding security checks into the evaluated circuit itself. In the PFE setting, such a protocol must *always* assume an honest function holder. If we hope to improve the security to defend against a malicious function holder, we cannot rely on black box techniques that provide security checks inside the private function.

Our approach uses techniques from Kamara et al. [21] and Carter et al. [6] to allow the mobile participants to jointly generate the randomness used to garble the circuit. This randomness is then passed to the Cloud for garbling a set of gates that can be assembled and checked using a cut-&-choose that is modified for the PFE setting to ensure the circuit is constructed correctly and that no information about the function

Private inputs: APPLICATION inputs a circuit C that corresponds to her private function. CLOUD inputs a set of permuted wire labels $w_i^{v_i}||v_i, w_i^{\overline{v}_i}||v_i$ for $i \in [1, ..., n+q-m]$.
Outputs: APPLICATION receives a vector of blinding values B and CLOUD receives a vector of blinded input wires bw_j^0, bw_j^1 for $j \in [1, ..., 2q]$ that are topologically ordered input wires for the gates of C.

1. CLOUD encrypts each wire label using his public key pk for a semantically secure, additive-homomorphic public key scheme, resulting in pairs $E_{pk}(w_i^{v_i}||v_i), E_{pk}(w_i^{\overline{v_i}}||v_i)$. CLOUD sends the permuted and encrypted labels to APPLICATION.
2. APPLICATION generates pairs of random blinding values b_j^0, b_j^1 for $j \in [1, ..., 2q]$ to blind the input labels for each gate. In addition, she generates a second permutation string v' where $|v'| = 2q$. For the ith gate in topological order, with input wires j and k, APPLICATION encrypts

$$E_{pk}(b_{2j-1}^{v'_{2j-1}}||v'_{2j-1}), E_{pk}(b_{2j-1}^{\overline{v'_{2j-1}}}||v_{2j-1}), E_{pk}(b_{2j}^{v'_{2j}}||v'_{2j}), E_{pk}(b_{2j}^{\overline{v'_{2j}}}||v_{2j})$$

3. APPLICATION then homomorphically adds the blind to the permuted keys by using the inverse permutation function to find $\pi^{-1}(2j) = k$ for $j \in [1, ..., 2q]$

$$E_{pk}(b_{2j-1}^{v'_{2j-1}}||v'_{2j-1}), E_{pk}(b_{2j-1}^{\overline{v'_{2j-1}}}||v_{2j-1}) + E_{pk}(w_{k-1}^{v_{k-1}}||v_{k-1}), E_{pk}(w_{\overline{k-1}}^{\overline{k-1}}||v_{k-1}),$$
$$E_{pk}(b_{2j}^{v'_{2j}}||v'_{2j}), E_{pk}(b_{2j}^{\overline{v'_{2j}}}||v'_{2j}) + E_{pk}(w_k^{v_k}||v_k), E_{pk}(w_{\overline{k}}^{v_{\overline{k}}}||v_{\overline{k}})$$

   APPLICATION then returns the resulting ciphertexts to CLOUD in topological ordering.
4. CLOUD decrypts the input wire labels, and for the ith gate in topological order, recovers the correct blinded input wire label by permuting based on the appended bit, returning the labels to their correct ordering (because, for the bit b, $b \oplus (v_j \oplus v'_{2j-1}) \oplus (v_j \oplus v'_{2j-1}) = b$).

Fig. 2. The $CTH(\cdot)$ functionality by Mohassel and Sadeghian

is leaked to the generator. As a secondary contribution, these security checks can be applied to the original two-party protocol of Mohassel and Sadeghian to provide security guarantees against a covert or malicious generator.

3.4 Security Definition

To capture the added security guarantee of function privacy in the outsourced setting, we slightly modify the definition for secure outsourced SMC proposed by Kamara et al. [20,21]. As in their work, we demonstrate security by showing that the real-world protocol execution can be simulated in an ideal setting with a trusted third party. However, rather than providing the function as a public parameter to all parties, we allow the simulator controlling the Application server to include the circuit as input to the trusted third party in the ideal world. By treating the circuit as an input to the computation, the input privacy captured by the existing SMC definition encompasses the function privacy that we wish to achieve in our PFE setting. We provide a summary of the definition from Kamara et al. [21] with our modifications:

Collusion assumption As in previous outsourcing protocols, we assume that CLOUD and APPLICATION do not collude against the other players in the computation. This comes from the inherently interactive nature of the underlying garbled circuit construction, which necessitates that these two parties not share information to maintain security. This requirement also implies that at least two parties in the computation must perform work that is linear with respect to the size of the circuit being evaluated. Kamara et al. [21] emphasize this point when they observe that an outsourced protocol that allows for arbitrary collusion implies a two-party SMC protocol where one party performs work that is sub-linear with respect to the size of the function being evaluated. While such protocols are possible using fully homomorphic encryption [11] or under other specific assumptions [13], it is not clear that these techniques can be efficiently applied to the outsourced setting.

Formally, Kamara et al. define non-collusion as non-cooperative adversaries [20]. We include the definition here and refer the reader to [20] for the discussion.

Definition 1. *An adversary $\mathcal{A}_i$ is non-cooperative with respect to adversary $\mathcal{A}_j$ if the messages $\mathcal{A}_i$ sends to $\mathcal{A}_j$ reveal no information about $\mathcal{A}_i$'s private values (i.e., its coins and input) to $\mathcal{A}_j$ beyond what can be inferred from $\mathcal{A}_j$'s output $f_j(x)$.*

Real-world execution Our real-world protocol executes between some number of participating parties $(P_1, ..., P_n)$ and an additional Cloud server P_{n+1}, with some number of adversaries $(\mathcal{A}_1, ..., \mathcal{A}_{m+1})$ with $m \leq n$ controlling a subset of the parties. At the beginning of the protocol, each party is run with a protocol input x_i, random coins r_i, and some auxiliary input z_i, while P_{n+1} receives only random coins r_{n+1} and auxiliary input z_{n+1}. Additionally, party P_1 is given a function f to provide as input to the protocol, which will be evaluated in a private manner. Each independent adversary $(\mathcal{A}_1, ..., \mathcal{A}_m)$ is given a single index i which represents the party it corrupts, while the monolithic adversary $\mathcal{A}_{m+1}$ receives a set of indices that represent the set of colluding parties.

For an honest party P_i, OUT_i is defined as P_i's output from the computation. For a corrupted party P_i, OUT_i is defined as P_i's view during the protocol execution. The ith partial output of the protocol execution is defined as

$$REAL^{(i)}(k, x, f; r) \stackrel{def}{=} \{OUT_j : j \in H\} \cup OUT_i$$

where k is a computational security parameter, H is the set of honest parties, and r is the set of all random coins given to all parties.

Ideal-world execution Following the real world, execution begins with the parties $(P_1, ..., P_n)$ receiving their inputs x_i, z_i and coins r_i, while P_{n+1} only receives z_i and r_i. Also, P_1 receives the function f and sends f to the trusted third party for evaluation. Each honest party sends $x'_i = x_i$ to the trusted party, while any cover, malicious, or non-cooperative party sends an arbitrary input x'_i. If any party fails to send an input, the trusted party aborts computation. If the server P_{n+1} is covert, it may send a $cheat$ message to the trusted third party. Then, with probability $1 - \epsilon$ (where ϵ is the covert security parameter), the trusted third party reveals that P_{n+1} cheated and the protocol terminates. Otherwise, with probability ϵ, the trusted third party continues. If P_{n+1} successfully cheats, the trusted third party sends $(x'_1, ..., x'_n)$ to the server, receives a set of output values $(y_1, ..., y_n)$ in return, and delivers these outputs to the respective party. If the server does not send the $cheat$ message, the trusted party returns $f_i(x'_1, ..., x'_n)$ to the respective party.

For an honest party P_i, OUT_i is defined as P_i's output from the computation. For a corrupted party P_i, OUT_i is defined as some value output by P_i. The ith partial output of the protocol execution in the presence of independent simulators $\mathcal{S} = (\mathcal{S}_1, ..., \mathcal{S}_{m+1})$ is defined as

$$IDEAL^{(i)}(k, x, f; r) \stackrel{def}{=} \{OUT_j : j \in H\} \cup OUT_i$$

where k is a computational security parameter, H is the set of honest parties, and r is the set of all random coins given to all parties.

Given these execution settings, Kamara et al. define security in terms of partial simulation with the following definition.

Definition 2. *An outsourced PFE protocol for n parties securely and privately computes a function f if there exists a set $\{Sim_i\}_{i\in[m+1]}$ of polynomial-time simulators such that for all polynomial-time adversaries $\mathcal{A} = (\mathcal{A}_1, ..., \mathcal{A}_{m+1})$*

$$\left\{REAL^{(i)}(k, x, f; r)\right\}_{k\in\mathbb{N}} \stackrel{c}{\approx} \left\{IDEAL^{(i)}(k, x, f; r)\right\}_{k\in\mathbb{N}}$$

We also use the following lemma from Kamara et al. [20] to prove the security of our protocols.

Lemma 1. *If a multi-party protocol between n parties $(P_1, ..., P_n)$ securely computes f in the presence of (1) independent and semi-honest adversaries and (2) a malicious $\mathcal{A}_i$ and honest $\{\mathcal{A}_j\}_{j \neq i}$; then it is also secure in the presence of an adversary $\mathcal{A}_i$ that is non-cooperative with respect to all other semi-honest adversaries.*

3.5 Notation

Here we define the notation used in the following sections. We describe the circuit hiding garbling protocol of Mohassel and Sadeghian [29] (which we later refer to as MS13) using a slight modification on the general notation of Bellare et al. [4] as $\mathcal{G} = (Gb_{MS}, En_{MS}, De_{MS}, Ev_{MS}, ev)$. The function $Gb_{MS}(1^k, f, r) \rightarrow (F, e, d)$ uses the random seed r to output a garbled circuit F using that is composed of two components (G, B). G is the garbled representation of the circuit, and is delivered to the generator, while B is a set of blinding values maintained by the evaluator during the OEP protocol. One critical note for the security of our protocol is that G reveals no information about the topology of the underlying circuit without possession of B. Finally, we also assume a projective garbling scheme $Gb(1^k, f, r)$ to be a Yao-based circuit garbling technique, with inputs defined as previously stated.

4 Semi-Honest Outsourced PFE

To illustrate our outsourcing techniques in a simplified setting, we first define a protocol for outsourcing PFE in the presence of semi-honest adversaries. Not only does this protocol serve as a warm up to stronger security settings, it could also be applied to applications requiring higher efficiency with less need for strong assurance.

4.1 Protocol Overview

For this protocol, we assume that CLOUD and APPLICATION are semi-honest and non-cooperative, and all MOBILE participants are semi-honest. At a high level, our protocol (described in detail in Figure 3) essentially splits the role of the circuit generator across all of the mobile devices participating in the computation. These devices must initiate the computation with a shared secret that will then be used by the Cloud as the randomness to generate the garbled wire values. This shared secret can be generated using a secure coin flip or a key exchange protocol. Next, the Cloud and the Application server engage in the MS13 circuit hiding garbling process to prepare the function for evaluation. Then, since each mobile device possesses the random seed used to garble the circuit, each can generate the garbled wire labels that correspond to its input and directly deliver their inputs to the Application server for evaluation. Finally, the Application server delivers the garbled output to each party and receives a table to ungarble her output wires (if she receives any output from the computation). Again, using the shared random seed, the mobile parties can then generate their own ungarbling table and recover their output.

Note that unlike other outsourced SMC protocols, this protocol does not achieve any sort of fairness, as the Application server receives her output before all other parties. However, fairness can be incorporated into this protocol by having CLOUD blind all of the output values with one-time pads, which it can release simultaneously at the end of the protocol.

4.2 Security

The security of our scheme essentially reduces to the underlying PFE scheme. CLOUD learns nothing about the private function based on the topology-hiding property of Mohassel and Sadeghian's protocol, while

Common inputs: a computational security parameter k and the side information function $\Phi_{size}(C) = (n, m, q)$ for APPLICATION's private function C. Here, n is the length of a single party's input, m is the length of a single party's output, and q is the number of gates in C. All MOBILE participants share a common random seed s where $|s| = k$.
Private inputs: For every $i \in [2, ..., N]$, MOBILE$_i$ inputs a string m^i. APPLICATION inputs a bit string a and a circuit C that evaluates her private function $f(a, m^2, ..., m^N)$.
Outputs: APPLICATION receives the output y_1 and MOBILE$_i$ receives the output y_i for $i \in [2, ..., N]$.

1. The MOBILE devices deliver s to CLOUD.
2. CLOUD and APPLICATION run the interactive protocol $Gb_{MS}(1^k, C, s) \rightarrow (G, B, e, d)$, with CLOUD as generator using the seed s and APPLICATION as the evaluator with the private circuit C. After the protocol, CLOUD receives the garbled circuit G and APPLICATION receives a vector of blinding values B. CLOUD delivers the garbled circuit G to APPLICATION along with output decoding function d for APPLICATION's output wires only.
3. APPLICATION and CLOUD run k instances of an oblivious transfer protocol (using OT-extensions). After the protocol, APPLICATION receives her garbled input $En_{MS}(e, a)$.
4. Every MOBILE participant delivers his garbled input $En_{MS}(e, m_i)$ to APPLICATION.
5. APPLICATION runs $Ev_{MS}([G, B], [En_{MS}(e, a), En_{MS}(e, m_2), ..., En_{MS}(e, m_N)])$ to evaluate the garbled circuit, producing a vector of output values $[Y_1, ..., Y_N]$.
6. APPLICATION recovers her output as $De_{MS}(d, Y_1) = y_1$. She then delivers the garbled output wires Y_i to the ith MOBILE participant for $i \in [2, ..., N]$.
7. The ith MOBILE participant recovers his output as $De_{MS}(e, Y_i) = y_i$.

Fig. 3. Semi-honest Outsourced PFE Protocol

APPLICATION learns nothing about any party's input based on the privacy of Yao's garbled circuit protocol. Finally, since CLOUD never observes any MOBILE input in any form and receives no output from the circuit, it cannot learn anything about the inputs to the computation. We provide a complete proof of the following theorem in Appendix B.

Theorem 1. *The outsourced PFE protocol securely and privately computes a circuit C when the Application server is semi-honest and non-cooperative, the Cloud is semi-honest and non-cooperative, and the mobile devices are semi-honest.*

4.3 Complexity

Our protocol essentially matches the complexity of Mohassel and Sadeghian for the Application server and Cloud, with the only computation for the mobile devices being the input and output preparation and recovery. For the mobile devices input m_i and output y_i, this amounts to $O(|m_i| + |y_i|)$ symmetric key operations. While Mohassel and Sadeghian provide a more thorough discussion of the complexity of their protocol, it essentially requires that the Cloud generate $O(g)$ wire values, encrypt these, decrypt the result, and garble the gates, yielding a complexity of $O(g)$ symmetric key operations and $O(g)$ asymmetric key encryptions for the additive homomorphic encryption. The Application server must then generate $O(g)$ blinding values, encrypt these, add them homomorphically to the wire values sent by the generator, then evaluate the circuit using these blinding values. This yields $O(g)$ symmetric key operations, $O(g)$ asymmetric key encryptions, and $O(g)$ homomorphic additions. Finally, using OT extensions, the Cloud and Application server must also perform $O(k)$ public key operations to garble the Application server's input. For a summary of each party's complexity, see Table 1.

Communication When developing protocols with mobile participants, another critical efficiency metric is communication costs. Transmitting data wirelessly tends to consume significantly more power than processing data, so minimizing the amount of data and the number of communication rounds is critical if a mobile application is to be implemented in a power efficient way. Because our protocol only requires the mobile

Mobile	$O(\|m_i\| + \|y_i\|)$
Application server	$O(g) + O(g \text{ HE}) + O(g \text{ HA}) + O(k)$
Cloud	$O(g) + O(g \text{ HE}) + O(k)$

Table 1. The computational complexity of our semi-honest outsourced PFE protocol. Note that HE signifies additively homomorphic encryptions, HA signifies homomorphic addition.

device to send and receive its inputs to the garbled circuit, the total communication cost for each mobile participant is $O(kn)$ bits sent and received over a single round of communication.

5 Covert Cloud Outsourced PFE

Common inputs: a computational security parameter k, a statistical security parameter λ, and the side information function $\Phi_{size}(C) = (n, m, q)$ for APPLICATION's private function C. Here, n is the length of a single party's input, m is the length of a single party's output, and q is the number of gates in C. All MOBILE participants share a common random seed s where $|s| = k$.
Private inputs: For every $i \in [2, ..., N]$, MOBILE$_i$ inputs a string m^i. APPLICATION inputs a bit string a and a circuit C that evaluates her private function $f(a, m^2, ..., m^N)$.
Outputs: APPLICATION receives the output y_1 and MOBILE$_i$ receives the output y_i for $i \in [2, ..., N]$.

1. The MOBILE devices deliver s to CLOUD.
2. CLOUD ensures that every copy of s received is identical. If any of the values differs, CLOUD aborts. Otherwise, CLOUD uses a pseudorandom generator seeded with s to produce random seeds r_i for $i \in [1, ..., \lambda]$.
3. For every $i \in [1, ..., \lambda]$, CLOUD and APPLICATION run the interactive protocol $Gb_{MS}(1^k, C, r_i) \rightarrow (G_i, B_i, e_i, d_i)$, with CLOUD as generator using the seed r_i and APPLICATION as the evaluator with the private circuit C. After the protocol, CLOUD receives the garbled circuit G_i and APPLICATION receives a vector of blinding values B_i. CLOUD delivers all garbled circuits to APPLICATION along with output decoding functions d_i for APPLICATION's output wires.
4. For every $i \in [1, ..., \lambda]$, APPLICATION and CLOUD run k instances of a covert-secure oblivious transfer protocol (using OT-extensions). After the protocol, APPLICATION receives her garbled input $En_{MS}(e_i, a)$.
5. APPLICATION selects a random and uniform index $E \in [1, ..., \lambda]$. For every $i \in [1, ..., \lambda] \setminus E$, CLOUD sends r_i to APPLICATION. APPLICATION runs $Gb_{MS}(1^k, C, r_i) \rightarrow (G'_i, B'_i, e'_i, d'_i)$ and checks that:
   (a) $G_i = G'_i$
   (b) $d_i = d'_i$
   (c) $En(e_i, a) = En(e'_i, a)$
   If any of these checks fails, APPLICATION aborts.
6. Every MOBILE participant $j \in [1, ..., N]$ delivers his garbled input $En_{MS}(e_E, m_j)$ to APPLICATION.
7. APPLICATION runs $Ev_{MS}(G_E, B_E, En_{MS}(e_E, a), En_{MS}(e_E, m_2), ..., En_{MS}(e_E, m_N))$ to evaluate the selected evaluation circuit, producing output values $[Y_1, ..., Y_N]$.
8. APPLICATION recovers her output as $De_{MS}(d_E, Y_1) = y_1$. She then delivers the garbled output wires Y_i to the jth MOBILE participant for $j \in [2, ..., N]$.
9. The jth MOBILE participant recovers his output as $En_{MS}(d_E, Y_j) = y_j$.

Fig. 4. Covert Server Outsourced PFE Protocol

While the semi-honest secure protocol in the previous section provides a foundation for outsourcing PFE, it does not capture the necessary security guarantees for many realistic applications. Our original motivation for developing protocols in this setting is to allow users to participate in secure computation with any available Cloud resource, trusted or untrusted. Because many Cloud providers wish to maintain a reputation as reliable and trustworthy, they may not cheat if there is some possibility of being caught. This behavior is modeled by the covert security model, allowing a balance between efficiency and security guarantees. To meet this application scenario, we developed an outsourced PFE protocol that is secure against a covert Cloud.

Mobile	$O(\|m_i\| + \|y_i\|)$
Application server	$\lambda[O(g) + O(g \text{ HE}) + O(g \text{ HA}) + O(k)]$
Cloud	$\lambda[O(g) + O(g \text{ HE}) + O(k)]$

Table 2. The computational complexity of our covert outsourced PFE protocol. Note that HE signifies additively homomorphic encryptions, HA signifies homomorphic addition.

5.1 Protocol Overview

In this protocol, we assume CLOUD is covert and non-cooperative, APPLICATION is semi-honest and non-cooperative, and all MOBILE parties are malicious. To achieve covert security, this protocol (Figure 4) replicates the semi-honest garbling process and adds a cut-&-choose operation to ensure that the Cloud has a low probability of successfully cheating. Given a statistical security parameter λ, the Cloud and Application server execute the garbling procedure λ times. They then execute λ sets of oblivious transfers to garble the Application server's input to the computation. After both of these sets of operations are complete, the Application server selects a single execution circuit, and for the remaining check circuits, she checks the correctness of both the circuit garbling and the oblivious transfers. This ensures that the Cloud will be caught with $\epsilon = \frac{1}{\lambda}$ probability. Once this procedure is complete, each mobile device delivers his input to the Application server for the selected execution circuit. The protocol concludes as in the semi-honest protocol, with the Application server evaluating the circuit and returning the output to each mobile device.

5.2 Security

The security of this protocol against a covert CLOUD is essentially achieved through the addition of a cut-&-choose. While it is possible for CLOUD to improperly garble a circuit, it will be caught with a very high probability. Furthermore, because the MOBILE devices perform so few operations (sending a random seed, garbling their input, and un-garbling their output), the checks required to prevent malicious behavior are trivially added in. Finally, while there are several additional operations performed in the cut-&-choose, it can be clearly observed from the proof of the semi-honest protocol that none of these operations reveal any additional information to a semi-honest APPLICATION. We have included a complete proof of the following theorem in Appendix C.

Theorem 2. *The outsourced PFE protocol securely and privately computes a circuit C with a covert security parameter of $\epsilon = \frac{1}{\lambda}$ when the Application server is semi-honest and non-cooperative, the Cloud is covert and non-cooperative, and the mobile devices are malicious.*

5.3 Complexity

The complexity of our covert secure protocol essentially multiplies the complexity of the semi-honest protocol by a factor of λ for both the Application Server and the Cloud. However, the complexity at the mobile parties is identical to the semi-honest model, since they still only garble input and ungarble output for a single circuit execution. These formulae are summarized in Table 2.

Communication The communication costs for the mobile participants in this protocol also match the semi-honest model. Although multiple circuits are generated and sent between the Application Server and Cloud, the mobile participants only transmit and receive input and output for a single circuit, totaling $O(kn)$ bits sent over a single round of communication.

6 Partially-Circuit Private Garbling

While the previous protocols provide relatively efficient and secure mechanisms for outsourcing PFE, the efficiency of the schemes is bound by the underlying requirement in MS13 that the evaluated function be represented entirely using NAND gates with independent wire labels. While this is necessary to maintain function privacy, it prohibits the application of modern circuit optimizations that significantly improve the performance of standard garbled circuit protocols. For example, the free-XOR technique [23] allows XOR gates to be evaluated using a single bitwise XOR, and to be transmitted over the network at no cost. Garbled row reduction [34] allows garbled gates to be transmitted at up to half the communication cost of a full garbled truth table. Unfortunately, because these optimizations require the garbled wire labels to be related, the OEP blinding values do not allow them to be applied to the private function.

As a first step towards combining these optimizations with PFE protocols, we designed a partially-circuit private (PCP) garbling technique where the private function receives input from some public preprocessing circuit, and outputs to another publicly available post-processing circuit. This allows us to incorporate standard circuit optimizations into the public portions of the evaluated circuit, reducing the efficiency limitations of the MS13 garbling scheme to a minimal private sub-circuit. As an example application, Section 7 describes our outsourced PFE protocol that is secured against a malicious Cloud through the use of public circuit based consistency checks that can be efficiently implemented using only free-XOR gates.

6.1 Protocol Overview

Our PCP garbling technique (Figure 5) is achieved by requiring the generator to alternate between different garbling techniques for each section of the circuit, then stitching these sections together by matching the appropriate garbled wire labels from the output of one circuit to the input of the next. Essentially, since the number of input and output wires of the private function are publicly known, they can be correlated to the output wires of a preprocessing function and the input wires to a post processing function. By simply garbling these circuits with the same wire labels, a circuit with a private central function can be generated by combining the MS13 private garbling technique with any Yao-based garbling technique as long as it has the projective property.

To evaluate the function, the evaluating party simply evaluates the preprocessing function using a typical garbled circuit evaluation protocol. Once she gets the output wire labels from this function, she can apply her blinding values used to garble the private function and continue evaluation according to MS13. Finally, since the output labels of the private circuit are not blinded, they can be directly input into the post-processing circuit, which is evaluated and output as in a typical garbled circuit protocol. Note that PCP garbling adds no additional overhead to either of the garbling techniques used. Since it relies only on correctly matching the output wire labels from the previous circuit to the input wires of the next circuit, each sub-circuit can be garbled and evaluated for the cost of garbling and evaluating that circuit outside of the combined protocol.

6.2 Security

To understand the security implications of combining two different garbled circuit primitives, we consider the fact that the wire labels for the private function are generated independently from each other as well as the wire labels used in the pre- and post-processing circuits. Thus, having knowledge of the input and output wires of the private section of the circuit, which are shared with one of the public circuits, reveals nothing to CLOUD about the topology of the private portion of the computation. Furthermore, during the evaluation of the circuit, we are guaranteed from both garbling schemes that privacy is maintained as long as APPLICATION only observes one wire label for each wire in the circuit. This holds for the public circuits, which may use correlated wire labels to achieve free-XOR and other optimizations, as well as the private

Common inputs: a computational security parameter k; circuit representations C_1 and C_2 of the preprocessing function $f_1(x_1, ..., x_n)$ and post processing function $f_2(x_1, ..., x_n)$; and the side information function $\Phi_{size}(C) = (n, m, q)$ for APPLICATION's private function C.
Private inputs: For every $i \in [2, ..., N]$, MOBILE$_i$ inputs a string m^i. APPLICATION inputs a circuit C_p that evaluates her private function $f_p(x_1, ..., x_n)$.
Outputs: APPLICATION receives a vector of blinding values B and CLOUD receives the garbled representation (G, e, d) of the composite function $f_2(f_p(f_1))$.
Circuit Garbling

1. Given random seeds s_1 and s_2, CLOUD garbled the pre and post processing circuits $Gb(1^k, C_1, s_1) \rightarrow (G_1, e_1, d_1)$ and $Gb(1^k, C_2, s_2) \rightarrow (G_2, e_2, d_2)$. For each of these circuits, there is an associated vector of input wire labels IW_i and a vector of output wire labels OW_i.
2. Using the random seed r_p, CLOUD and APPLICATION run the modified $Gb_{MS}(1^k, C_p, r_p, OW_1, IW_2) \rightarrow (G_p, B_p, e_p, d_p)$. Rather than generating new wire labels for the input and output wires of G_p, the modified garbling algorithm associates OW_1 with the input wires of G_p and IW_2 with the output wires of G_p. This sets the composite functions $En_{MS}(e_p, De(d_1, Y_1)) = Y_1$ and $En(e_2, De_{MS}(d_p, Y_p)) = Y_p$ to both be the *identity function*.
3. CLOUD receives the composite garbled circuit (G_c, e_1, d_2) and APPLICATION receives a vector of blinding values B_p for the private section of the function.

Circuit Evaluation

1. Given the garbled input values $[En(e_1, x_i)]$ for $i \in [1, ..., N]$ and G_c, APPLICATION evaluates $Ev(G_1, [En(e_1, x_i)])$ using the chosen garbled circuit evaluation technique.
2. Given the garbled wire values for $Ev(G_1, [En(e_1, x_i)])$, APPLICATION evaluates $Ev_{MS}(G_p, Ev(G_1, [En(e_1, x_i)]))$ using the MS13 evaluation algorithm.
3. Given the garbled wire values for $Ev_{MS}(G_p, Ev(G_1, [En(e_1, x_i)]))$, APPLICATION concludes by evaluating $Ev(G_2, Ev_{MS}(G_p, Ev(G_1, [En(e_1, x_i)])))$ again using the chosen evaluation algorithm.

Fig. 5. Outsourced PCP garbling and evaluation protocols

circuit, which simply requires the addition of the correct blinding pad to properly evaluate the circuit. We prove security of this construction applied to our malicious secure PFE outsourcing protocol in Appendix D, but these security properties can be shown to extend to more general applications as well.

6.3 Additional Applications

In addition to the significant efficiency gains, PCP garbling allows for public portions of the evaluated circuit to be verified by all parties involved in the computation. In certain real-world scenarios, the parties participating in the computation may want some assurance that their private input or output is being prepared in a secure manner that can be audited. For example, in data mining applications, data must be aggregated to preserve privacy before it is processed by research or advertising agencies. By making the necessary preprocessing a public circuit, data owners can be assured that their data is properly anonymized before any proprietary or sensitive analytic functions are applied using the private circuit. In Section 7, we discuss how these publicly verifiable circuits could be applied to future protocols to achieve security against a malicious function holder. In summary, our PCP garbling technique can be used both to add gate optimizations or publicly verifiable subcircuits to PFE applications where only a portion of the circuit needs to remain private.

7 Malicious Cloud Outsourced PFE

To construct an outsourced PFE protocol that is secure against a malicious CLOUD, we use the PCP garbling to add the k-probe-resistant input encoding technique and the 2-Universal output hash, which strengthens our cut-&-choose as in the two-party protocol developed by shelat and Shen [37]. Using PCP garbling allows

us to reduce the additional gates from $4O(\lambda kn)$ NAND gates to $O(\lambda kn)$ free-XOR gates, and ensures that all parties participating in the computation will know and can verify that the necessary checks are being performed within the garbled circuit. We present our protocol and prove security in the presence of a semi-honest APPLICATION that is in possession of the private function.

7.1 Background constructions

Here we briefly describe the constructions we use to prevent selective failure attacks and to ensure input consistency. For more detailed analysis, see shelat and Shen's original publication [37].

***k*-probe-resistant matrix encoding** In the setting where CLOUD is malicious, it may attempt to learn bits from APPLICATION'S input by providing invalid wire labels during the oblivious transfer. If these invalid labels cause the circuit evaluation to fail, CLOUD can infer which labels were selected. To prevent this "selective failure" attack, we use the k-probe-resistant matrix construction developed by Lindell and Pinkas [26] and optimized by shelat and Shen [37]. Rather than input her real input a, APPLICATION encodes her input as $\overline{a}$ using the k-probe-resistant matrix $\mathbf{M}$ such that $\mathbf{M} \cdot \overline{a} = a$. Intuitively, since there are many possible encodings $\overline{a}$ that correspond to the same a, CLOUD would have to successfully probe k bits of $\overline{a}$ before learning anything about the real input a. While the reconstruction of APPLICATION'S input a must now be performed inside the circuit, it can be efficiently instantiated using only XOR gates.

2-universal hash input verification Since multiple circuits must be evaluated to ensure that a majority of the circuits are correctly constructed, a check is required to ensure all MOBILE parties use the same input across all evaluation circuits. To achieve this, shelat and Shen add an auxiliary circuit that outputs the 2-universal hash of each MOBILE input for each circuit. The regularity properties of the hash function ensure that no information is leaked to APPLICATION when the hashes are checked, while a discrepancy in the hashes indicates that one of the MOBILE parties used inconsistent inputs in some of the evaluated circuits. In their protocol, shelat and Shen demonstrate how to implement this hash inside of a circuit as a boolean matrix multiplication using only $O(kn)$ XOR gates.

7.2 Protocol Overview

In this protocol, we assume CLOUD is malicious and non-cooperative, APPLICATION is semi-honest and non-cooperative, and all MOBILE parties are malicious. Our final outsourced PFE protocol (Figure 6) begins by having all mobile participants commit to their inputs for a set of λ garbled preprocessing circuits, where λ is a statistical security parameter. Once these values are committed, the generating party will garble λ copies of the preprocessing function, post processing function, and private function, using the appropriate garbling protocol for each section. Once the Application server possesses the garbled circuits and has her input delivered for each via oblivious transfer, she selects a fraction of the circuits to be opened using cut-&-choose. Once these circuits are verified, the mobile parties decommit their inputs corresponding to the remaining evaluation circuits, and the Application server evaluates the remaining unopened garbled circuits. Finally, the Application server concludes by taking the majority output value from all of the evaluation circuits and delivering this value to the mobile devices to conclude the computation.

7.3 Security

The primary requirement for achieving malicious security over covert security is in modifying the cut-&-choose to include *multiple* evaluation circuits. This modification allows us to reduce the probability of

Common inputs: a computational security parameter k, a statistical security parameter λ, a commitment scheme $com_c(x)$ with key c and message x, and the side information function $\Phi_{size}(C) = (n, m, q)$ for APPLICATION's private function C. Here, n is the length of a single party's input, m is the length of a single party's output, and q is the number of gates in C. All MOBILE participants share a common random seed s where $|s| = k$.
Private inputs: For every $i \in [2, ..., N]$, MOBILE$_i$ inputs a string m^i. APPLICATION inputs a bit string a and a circuit C that evaluates her private function $f(a, m^2, ..., m^N)$.
Outputs: APPLICATION receives the output y_1 and MOBILE$_i$ receives the output y_i for $i \in [2, ..., N]$.

1. The MOBILE devices deliver s to CLOUD.
2. CLOUD ensures that every copy of s received is identical. If any of the values differs, CLOUD aborts. Otherwise, CLOUD uses a pseudorandom generator seeded with s to produce random seeds r_i for $i \in [1, ..., \lambda]$.
3. Every MOBILE participant generates the same set of seeds r_i for $i \in [1, ..., \lambda]$. The j^{th} MOBILE then generate a one-time pad $b_j \in \{0,1\}^m$ for $j \in [2, ..., N]$ to blind his output from the evaluated circuits. Each then garbles his input using each seed, so that the j^{th} mobile party produces $En(e_i, m^j||b_j)$ for $i \in [1, ..., \lambda]$. Finally, MOBILE generates commitment keys c_i for $i \in [1, ..., \lambda]$ and commits to each garbled input as $com_{c_i}(En(e_i, m^j||b_j))$ for $i \in [1, ..., \lambda]$. MOBILE delivers all of his commitments to APPLICATION.
4. CLOUD and APPLICATION agree on the preprocessing and post-processing functions. These parties jointly establish a 2-Universal hash matrix $\mathbf{H} \in \{0,1\}^{k \times n}$, and APPLICATION generates her k-probe-resistant matrix $\mathbf{M}$. She then defines an input vector $\overline{a}$ such that $\mathbf{M} \cdot \overline{a} = a$. They then define the preprocessing function $f_1(\overline{a}, m^2, ..., m^N) = \{\mathbf{M} \cdot \overline{a}, m^2, ..., m^N\}$ and the output function to be $f_2(f(a, m^2, ..., m^N), m^2||b_2, ..., m^N||b_N) = \{y_1, [y_2 \oplus b_2, \mathbf{H} \cdot m^2], ..., [y_N \oplus b_N, \mathbf{H} \cdot m^N]\}$. They set C_1 and C_2 to be the circuit representations of these functions.
5. For every index $[i \in 1, ..., \lambda]$, CLOUD and APPLICATION run $Gb_{PCP}(1^k, C_1, C_2, C, r_i) \rightarrow (G_i, B_i, e_i, d_i)$.
6. For every $i \in [1, ..., \lambda]$, APPLICATION and CLOUD run k instances of a malicious-secure oblivious transfer protocol (using OT-extensions). After the protocol, APPLICATION receives her garbled input $En(e_i, \overline{a})$.
7. APPLICATION selects a random set of indices $[E] \in [1, ..., \lambda]$ where $|[E]| = \frac{2\lambda}{5}$. For the indices $i \in [1, ..., \lambda] \setminus [E]$, CLOUD reveals r_i. APPLICATION runs $Gb_{PCP}(1^k, C_1, C_2, C, r_i) \rightarrow (G'_i, B'_i, e'_i, d'_i)$ and checks:
   (a) $G_i = G'_i$
   (b) $d_i = d'_i$
   (c) $En(e_i, \overline{a}) = En(e'_i, \overline{a})$
   If any of these checks fails for any index i, APPLICATION aborts.
8. Every MOBILE participant decommits its garbled input by delivering c_i for $i \in [E]$.
9. APPLICATION For every $i \in [E]$, APPLICATION runs $Ev_{PCP}(G_i, B_i, En(e_i, \overline{a}), En(e_i, m^2||b_2), ..., En(e_i, m^N||b_N)) = Y_1, Y_2, ..., Y_N$
10. For all $i \in [E]$, APPLICATION recovers $De(d_i, go_1) = y_1$ and $De(d_i, go_j) = y_j \oplus b_j, \mathbf{H} \cdot m^j$ for all $j \in [2, ..., N]$. For each $j \in [2, ..., N]$, APPLICATION ensures that the hash of the input $\mathbf{H} \cdot m^j$ is the same across every evaluated circuit in $[E]$ and aborts otherwise.
11. APPLICATION takes a majority vote from all evaluation circuits $i \in [E]$ to determine the output $y_1, y_2 \oplus b_2, ..., y_N \oplus b_N$ for all parties. If no majority exists, APPLICATION aborts. Otherwise, she delivers $y_j \oplus b_j$ to the j^{th} MOBILE party.
12. The j^{th} MOBILE party recovers his output $y_j = (y_j \oplus b_j) \oplus b_j$.

Fig. 6. Malicious Server Outsourced PFE Protocol

cheating to a computationally negligible probability. Our cut-&-choose technique is derived from the two-party garbled circuit protocol developed by shelat and Shen [36], which gives $2^{-0.32\lambda}$ probability of an adversary successfully cheating. Given our computational security parameter k, we use this bound to set $\lambda = \frac{k}{0.32}$.

This cut-&-choose technique requires additional verification to ensure that the MOBILE participants provide consistent inputs to all evaluation circuits, as well as taking the majority output from the evaluation circuits to prevent the CLOUD from learning anything about a party's input based on differing circuit outputs from the evaluation circuits. By applying our partial PFE garbling technique, we can incorporate existing techniques from traditional malicious secure garbled circuit protocols to combat these attacks. In particular, we ensure input consistency using shelat and Shen's 2-Universal hash construction [37], output privacy during the majority vote with additional one-time pads, and selective failure prevention using the k-probe-resistant encoding as implemented by shelat and Shen [37]. However, our protocol could easily be modified to incorporate new consistency checks as they are developed. We formally prove the following theorem in Appendix D.

Theorem 3. *The outsourced PFE protocol securely and privately computes a circuit C when the Application server is semi-honest and non-cooperative, the Cloud is malicious and non-cooperative, and the mobile devices are malicious.*

Recent work in two-party SMC has produced protocols with more efficient cut-&-choose techniques, such as Lindell's auxiliary circuit technique [25]. However, applying this techniques to the outsourced PFE setting remains an open challenge. In particular, Lindell's technique of punishing the circuit generator by exposing his outputs to the evaluator does not translate into a setting where the generator is responsible for other parties' inputs. In our setting, CLOUD could potentially cheat, which would result in an honest MOBILE party's input being revealed to APPLICATION, which is an undesirable property in any outsourced protocol.

Security against a stronger function holder While our protocol is only secure against a semi-honest APPLICATION, the use of PCP garbling provides us with publicly verifiable preprocessing and post-processing functions, which is one step towards strengthening the security of outsourced PFE. If we were to develop an aut-secure (defined by Bellare et al. [4]) PFE garbling scheme and an efficient zero-knowledge circuit commitment, it would allow this protocol to be secure even in the presence of a malicious function holder. This is because whatever arbitrary function is chosen by the Application server to be evaluated, to prove security the function must be recoverable by the simulator and sent to the trusted third party to be evaluated in the ideal world. If this function is committed and verified to have been executed by the Application server, then the output distributions between the real world and ideal world will be computationally indistinguishable. Furthermore, the function holder must not be able to tamper with the output of the garbled circuit, which is prevented by the aut-secure garbling scheme. While this adversary model appears flawed in practice when the function holder both defines the function *and* receives the output of that function, it provides a useful guarantee when *only* the mobile parties receive output from the computation.

7.4 Complexity

This protocol magnifies the complexity of our covert secure protocol with respect to three parameters: the size of the circuit representations of f_1 and f_2, as well as the increase in λ to ensure security against a malicious CLOUD ($\lambda = 16$ is common in practice for covert security, with $\lambda = 256$ for malicious security under our cut-&-choose parameters). In this protocol, CLOUD is tasked with garbling λ copies of the private circuit C and both public functions C_1 and C_2. APPLICATION is then responsible for checking $\frac{3\lambda}{5}$ complete

Mobile	$\lambda O(\|m_i\| + \|y_i\|)$
Application server	$\lambda[O(\|C_1\| + g + \|C_2\|) + O(g \text{ HE}) + O(g \text{ HA}) + O(k)]$
Cloud	$\lambda[O(\|C_1\| + g + \|C_2\|) + O(g \text{ HE}) + O(k)]$

Table 3. The computational complexity of our malicious outsourced PFE protocol. Note that HE signifies additively homomorphic encryptions, HA signifies homomorphic addition.

circuits (preprocessing, private, and postprocessing), as well as evaluating $\frac{2\lambda}{5}$ remaining circuits, according to the optimal cut-&-choose parameters used by shelat and Shen [36]. However, the use of PCP garbling significantly reduces the complexity of the preprocessing and post-processing functions. Since it requires 4 NAND gates to represent an XOR gate, applying the free-XOR optimization to these functions allows us to reduce the number of gates from $4O(\lambda kn)$ NAND gates to $O(\lambda kn)$ free-XOR gates, which can be transmitted at no cost and evaluated with a single bitwise XOR operation. For a detailed representation of these operations, see Table 3.

Communication To achieve security against a malicious Cloud, the mobile participants must commit to inputs for all of the generated circuits, then decommit a fraction of those inputs that will be used in the evaluation circuits. Because of this, the communication cost for the mobile devices in this protocol is multiplied by the cut-&-choose parameter λ, totaling $O(\lambda kn)$ bits over two rounds of communication.

7.5 Black Box PFE Outsourcing

Recent work has demonstrated that the concept of auxiliary circuits that we apply here can be extended for efficiently outsourcing SMC protocols in a completely black-box manner [19,8]. However, additional research into varied adversary models and computation techniques is necessary before black-box outsourcing can be applied to PFE protocols. Our partial PFE garbling technique offers a useful first step in developing a completely black-box approach to outsourcing PFE. However, it can only be applied to garbled circuit-based PFE schemes. To achieve security against malicious adversaries using this generic technique, a two-party PFE protocol using garbled circuits must first be developed. The only existing PFE protocol that has been demonstrated secure against malicious adversaries is the secret-sharing based scheme of Mohassel et al. [28]. Our outsourcing protocol can be reduced to a two-party PFE protocol, but is only secure against a malicious circuit generator, while the function holder must be semi-honest.

8 Conclusion

Garbled circuit protocols have been developed and optimized for a range of privacy-preserving applications. The outsourced SMC model has shown that these protocols can be efficiently applied to some mobile applications as well. However, these outsourcing protocols require that the function being evaluated be public to all participants, which prohibits their use in a variety of government and commercial applications. This work develops the first efficient outsourced PFE protocols that allow for private applications to be evaluated securely using mobile input sources. By combining existing outsourcing techniques with a novel partially-circuit private garbling technique, we show that security can be maintained against malicious mobile devices and covert or malicious Cloud providers. These protocols allow for a range of real-world privacy-preserving applications that were not feasible using existing outsourcing protocols.

References

1. Abadi, M., Feigenbaum, J.: Secure circuit evaluation. Journal of Cryptology 2(1), 1–12 (1990)

2. Aumann, Y., Lindell, Y.: Security against covert adversaries: Efficient protocols for realistic adversaries. In: Theory of Cryptography (TCC) (2007)
3. Barni, M., Failla, P., Kolesnikov, V., Lazzeretti, R., Sadeghi, A.R., Schneider, T.: Secure evaluation of private linear branching programs with medical applications. In: ESORICS (2009)
4. Bellare, M., Hoang, V.T., Rogaway, P.: Foundations of garbled circuits. In: Proceedings of the ACM Conference on Computer and Communications Security (CCS) (2012)
5. Brickell, J., Porter, D.E., Shmatikov, V., Witchel, E.: Privacy-preserving remote diagnostics. In: Proceedings of the ACM Conference on Computer and Communications Security (CCS) (2007)
6. Carter, H., Lever, C., Traynor, P.: Whitewash: Outsourcing garbled circuit generation for mobile devices. In: Proceedings of the Annual Computer Security Applications Conference (ACSAC) (2014)
7. Carter, H., Mood, B., Traynor, P., Butler, K.: Secure outsourced garbled circuit evaluation for mobile devices. In: Proceedings of the USENIX Security Symposium (SECURITY) (2013)
8. Carter, H., Mood, B., Traynor, P., Butler, K.: Outsourcing secure two-party computation as a black box. In: Proceedings of the International Conference on Cryptology and Network Security (CANS) (2015)
9. Damgård, I., Keller, M., Larraia, E., Pastro, V., Scholl, P., Smart, N.P.: Practical covertly secure MPC for dishonest majority - or: Breaking the SPDZ limits. In: ESORICS (2013)
10. Damgård, I., Pastro, V., Smart, N., Zakarias, S.: Multiparty computation from somewhat homomorphic encryption. In: Advances in Cryptology–CRYPTO (2012)
11. Gentry, C.: Fully homomorphic encryption using ideal lattices. In: Proceedings of the Annual ACM Symposium on Theory of Computing (STOC) (2009)
12. Goldreich, O., Micali, S., Wigderson, A.: How to play any mental game. In: Proceedings of the Annual ACM Symposium on Theory of Computing (STOC) (1987)
13. Gordon, S.D., Katz, J., Kolesnikov, V., Labs, A.l.B., Krell, F., Raykova, M.: Secure two-party computation in sublinear (amortized) time. In: Proceedings of the ACM conference on Computer and communications security (CCS) (2012)
14. Huang, Y., Katz, J., Evans, D.: Efficient secure two-party computation using symmetric cut-and-choose. In: Advances in Cryptology–CRYPTO (2013)
15. Huang, Y., Evans, D., Katz, J., Malka, L.: Faster secure two-party computation using garbled circuits. In: Proceedings of the USENIX Security Symposium (SECURITY) (2011)
16. Huang, Y., Katz, J., Evans, D.: Quid-pro-quo-tocols: Strengthening semi-honest protocols with dual execution. In: Proceedings of the IEEE Symposium on Security and Privacy (S & P) (2012)
17. Huang, Y., Katz, J., Kolesnikov, V.: Amortizing garbled circuits. In: Advances in Cryptology–CRYPTO (2014)
18. Ishai, Y., Paskin, A.: Evaluating branching programs on encrypted data. In: Theory of Cryptography (TCC) (2007)
19. Jakobsen, T.P., Nielsen, J.B., Orlandi, C.: A framework for outsourcing of secure computation. In: Proceedings of the ACM Workshop on Cloud Computing Security (CCSW) (2014)
20. Kamara, S., Mohassel, P., Raykova, M.: Outsourcing multi-party computation. Cryptology ePrint Archive, Report 2011/272 (2011), `http://eprint.iacr.org/`
21. Kamara, S., Mohassel, P., Riva, B.: Salus: A system for server-aided secure function evaluation. In: Proceedings of the ACM conference on Computer and communications security (CCS) (2012)
22. Katz, J., Malka, L.: Constant-round private function evaluation with linear complexity. In: Advances in Cryptology–ASIACRYPT (2011)
23. Kolesnikov, V., Schneider, T.: Improved garbled circuit: Free XOR gates and applications. In: Proceedings of the international colloquium on Automata, Languages and Programming, Part II (2008)
24. Kolesnikov, V., Schneider, T.: A practical universal circuit construction and secure evaluation of private functions. In: Financial Cryptography and Data Security (FC) (2008)
25. Lindell, Y.: Fast cut-and-choose based protocols for malicious and covert adversaries. In: Advances in Cryptology–CRYPTO (2013)
26. Lindell, Y., Pinkas, B.: An efficient protocol for secure two-party computation in the presence of malicious adversaries. In: Advances in Cryptology–EUROCRYPT (2007)
27. Lindell, Y., Riva, B.: Cut-and-choose yao-based secure computation in the online/offline and batch settings. In: Advances in Cryptology–CRYPTO (2014)
28. Mohassel, P., Sadeghian, S., Smart, N.P.: Actively secure private function evaluation. In: Advances in Cryptology–ASIACRYPT (2014)
29. Mohassel, P., Sadeghian, S.: How to hide circuits in MPC an efficient framework for private function evaluation. In: Advances in Cryptology–EUROCRYPT (2013)
30. Mood, B., Gupta, D., Butler, K., Feigenbaum, J.: Reuse it or lose it: More efficient secure computation through reuse of encrypted values. In: Proceedings of the ACM conference on Computer and communications security (CCS) (2014)
31. Nielsen, J.B., Nordholt, P.S., Orlandi, C., Burra, S.S.: A new approach to practical active-secure two-party computation. In: Advances in Cryptology–CRYPTO (2012)
32. Nielsen, J., Orlandi, C.: LEGO for two-party secure computation. In: Theory of Cryptography (TCC) (2009)

33. Paus, A., Sadeghi, A.R., Schneider, T.: Practical secure evaluation of semi-private functions. In: Applied Cryptography and Network Security (ACNS) (2009)
34. Pinkas, B., Schneider, T., Smart, N.P., Williams, S.: Secure two-party computation is practical. In: Advances in Cryptology–ASIACRYPT (2009)
35. Sadeghi, A.R., Schneider, T.: Generalized universal circuits for secure evaluation of private functions with application to data classification. In: Proceedings of the International Conference on Information Security and Cryptology (ICISC) (2009)
36. shelat, a., Shen, C.H.: Two-output secure computation with malicious adversaries. In: Advances in Cryptology–EUROCRYPT (2011)
37. shelat, a., Shen, C.H.: Fast two-party secure computation with minimal assumptions. In: Proceedings of the ACM conference on Computer and communications security (CCS) (2013)
38. Valiant, L.G.: Universal circuits (preliminary report). In: Proceedings of the Annual ACM Symposium on Theory of Computing (STOC) (1976)
39. Yao, A.C.: Protocols for secure computations. In: Proceedings of the Annual Symposium on Foundations of Computer Science (FOCS) (1982)

A Proof of MS13

Here we prove the prv.sim security of Mohassel and Sadeghian's garbling scheme.

Theorem 4. *The scheme Gb_{MS} is prv.sim secure over the size-information function Φ_{circ}.*

We set an adversary A such that the tuple of garbled circuit, garbled input, and decoding function (F, X, d) is indistinguishable between the real protocol and a simulated view where the simulator is given $\Phi_{circ}(f)$ and $f(x)$. To do this, we build a simulator S through the following series of hybrid experiments. We slightly modify the notation of Bellare et al. [4] and notate the prv.sim game as $PrvSim^{A}_{MS,\Phi_{circ},S(k,b)}$, where $b = 1$ denotes the real protocol and $b = 0$ denotes the simulated view. Note that for this garbling scheme, the garbled function $F = (G, B)$, where G is the set of garbled gates and B is the set of blinding values generated during garbling.

$Hyb1^{A}_{MS,\Phi_{circ},S(k)}$: This experiment is identical to $PrvSim^{A}_{MS,\Phi_{circ},S(k,1)}$ except that the experiment garbles the output wires of G to produce the fixed output $y = f(x)$.

Lemma 2. $PrvSim^{A}_{MS,\Phi_{circ},S(k,1)} \overset{c}{\approx} Hyb1^{A}_{MS,\Phi_{circ},S(k)}$

Proof. This follows from the original proof of Yao's protocol. Since the adversary is only given a single garbled input X, they can only decrypt a single entry at each garbled gate, guaranteed by the semantic security of the underlying encryption scheme. Because they cannot decrypt any *other* output wire labels, a real circuit garbling and a fixed output garbling are indistinguishable.

$Hyb2^{A}_{MS,\Phi_{circ},S(k)}$: This experiment is identical to $Hyb1^{A}_{MS,\Phi_{circ},S(k)}$ except that the experiment provides a random garbled input X' instead of the real garbled input X.

Lemma 3. $Hyb1^{A}_{MS,\Phi_{circ},S(k)} \overset{c}{\approx} Hyb2^{A}_{MS,\Phi_{circ},S(k)}$

Proof. This follows from the garbling function $En_{MS}(e, \cdot)$, which produces a uniformly random string to represent each garbled input bit. Since the distribution of these strings is indistinguishable for any input x, then the garbled inputs $X = En_{MS}(e, x)$ and $X' = En_{MS}(e, x')$ are indistinguishable.

Lemma 4. $Hyb2^{A}_{MS,\Phi_{circ},S(k)}$ *runs in polynomial time.*

Proof. This lemma follows trivially since the real world garbling protocol runs in polynomial time and each intermediate hybrid adds a constant number of operations.

We conclude the proof by noting that $Hyb2^{A}_{MS,\Phi_{circ},S(k)} \equiv PrvSim^{A}_{MS,\Phi_{circ},S(k,0)}$. S follows the real world protocol where the hybrid experiment does not differ, garbling the function f given $\Phi_{circ}(f)$ and $y = f(x)$ (with the exception of the output wire modification in Lemma 2). From Lemma 2-4, S proves Theorem 4.

B Semi-honest Protocol Proof

Here we prove the security of our outsourced PFE protocol against semi-honest adversaries according to Theorem 1.

B.1 Semi-honest MOBILE

Here we construct a simulator S_M that can simulate the view of a semi-honest MOBILE adversary M^*. Without loss of generality, this party represents any number of colluding MOBILE devices executing the protocol. We construct this simulator through a set of hybrid experiments.

$Hyb1^{(M)}(k, x, f; r)$: This experiment is identical to $REAL^{(M)}(k, x, f; r)$ except that the experiment sends M^*s input m to the trusted third party instead of entering it into the real world protocol.

Lemma 5. $REAL^{(M)}(k, x, f; r) \overset{c}{\approx} Hyb1^{(M)}(k, x, f; r)$

Proof. Since all parties are behaving semi-honestly, the experiment can send the input it used to run M^* to the trusted third party. The trusted party will then return the result of computation $y = f(x)$ to the experiment, which will be identical to the result output by the circuit executed in the real world.

$Hyb2^{(M)}(k, x, f; r)$: This experiment is identical to $Hyb1^{(M)}(k, x, f; r)$ except that the experiment uses s, received from M^* at the beginning of the protocol, to generate the output wire labels used in the real world protocol. It then garbles the value y received from the trusted third party and returns go_{M^*} to M^*

Lemma 6. $Hyb1^{(M)}(k, x, f; r) \overset{c}{\approx} Hyb2^{(M)}(k, x, f; r)$

Proof. This hybrid holds since the garbled wire values in the real world protocol are generated deterministically based on s. In addition, since all parties are semi-honest, the resulting output y will be identical in both experiments.

Lemma 7. $Hyb2^{(M)}(k, x, f; r)$ *runs in polynomial time.*

Proof. This lemma follows trivially since a real world execution of the protocol runs in polynomial time and each intermediate hybrid adds only constant time operations.

We conclude the proof by letting S_M execute $Hyb2^{(M)}(k, x, f; r)$, following the real world protocol where the hybrid experiment does not differ. S_M runs M^* and controls CLOUD and APPLICATION. S_M simulates the real world protocol and outputs whatever M^* outputs at the end of the simulation. From Lemma 5-7, S_M proves Theorem 1 when the MOBILE parties are semi-honest.

B.2 Semi-honest APPLICATION

Here we construct a simulator S_A that can simulate the view of a semi-honest APPLICATION adversary A^*. We construct this simulator through a set of hybrid experiments.

$Hyb1^{(A)}(k, x, f; r)$: This experiment is identical to $REAL^{(A)}(k, x, f; r)$ except that the experiment sends A^*s inputs $f(\cdot), x$ to the trusted third party.

Lemma 8. $REAL^{(A)}(k, x, f; r) \overset{c}{\approx} Hyb1^{(A)}(k, x, f; r)$

Proof. Since A^* is semi-honest, the experiment invokes the trusted party using the input function $f(\cdot)$ and the input data x that it provided to A^* at the start of the experiment. The value returned by the trusted third party $y = f(x)$ will be identical to the value output by the circuit evaluated in the real world.

$Hyb2^{(A)}(k, x, f; r)$: This experiment is identical to $Hyb1^{(A)}(k, x, f; r)$ except that during the $Garble_{CTH}()$ execution, the experiment runs the simulator S_{MS} to simulate A^*s view of the garbled circuit.

Lemma 9. $Hyb1^{(A)}(k, x, f; r) \stackrel{c}{\approx} Hyb2^{(A)}(k, x, f; r)$

Proof. Since the experiment knows topologically which gates feed into output wires of the private function $f(\cdot)$, it can garble those gates to always output the same value y. Based on Theorem 4, we know that S_{MS} exists and can indistinguishably simulate the view of the garbling scheme.

Lemma 10. $Hyb2^{(A)}(k, x, f; r)$ *runs in polynomial time.*

Proof. This lemma follows trivially since the experiment runs the real world protocol and the simulator S_{MS}, both of which run in polynomial time.

We conclude the proof by letting S_A execute $Hyb2^{(A)}(k, x, f; r)$, following the real world protocol where the hybrid experiment does not differ. S_A runs A^* and controls CLOUD and MOBILE. S_A simulates the real world protocol and outputs whatever A^* outputs at the end of the simulation. From Lemma 8-10, S_A proves Theorem 1 when the APPLICATION party is semi-honest.

B.3 Semi-honest CLOUD

Here we construct a simulator S_C that can simulate the view of a semi-honest CLOUD adversary C^*. We construct this simulator through a set of hybrid experiments.

$Hyb1^{(C)}(k, x, f; r)$: This experiment is identical to $REAL^{(C)}(k, x, f; r)$ except that the experiment garbles a random function $f'(\cdot)$ during the $Gb_{MS}()$ invocation instead of the function $f(x)$ used in the real world invocation, such that $\Phi_{size}(f') = \Phi_{size}(f)$

Lemma 11. $REAL^{(C)}(k, x, f; r) \stackrel{c}{\approx} Hyb1^{(C)}(k, x, f; r)$

Proof. Based on the proof of security from Mohassel and Sadeghian [29], a simulator S_{CTH} exists that can simulate the view of C^* given only $\Phi_{size}(f)$.

$Hyb2^{(C)}(k, x, f; r)$: This experiment is identical to $Hyb1^{(C)}(k, x, f; r)$ except that during the oblivious transfer execution, the experiment provides a random input x' instead of APPLICATION's real input x.

Lemma 12. $Hyb1^{(C)}(k, x, f; r) \stackrel{c}{\approx} Hyb2^{(C)}(k, x, f; r)$

Proof. Based on the security guarantees of the OT protocol used, a simulator S_{OT} can simulate C^* view of a real protocol invocation using a random input x'.

Lemma 13. $Hyb2^{(C)}(k, x, f; r)$ *runs in polynomial time.*

Proof. This lemma follows trivially since the experiment runs the real world protocol with two polynomial time simulators.

We conclude the proof by letting S_C execute $Hyb2^{(C)}(k, x, f; r)$, following the real world protocol where the hybrid experiment does not differ. S_C runs C^* and controls APPLICATION and MOBILE. S_C simulates the real world protocol and outputs whatever C^* outputs at the end of the simulation. From Lemma 11-13, S_C proves Theorem 1 when the CLOUD party is semi-honest.

C Covert Protocol Proof

Here we prove the security of our outsourced PFE protocol against a covert Cloud according to Theorem 2.

C.1 Malicious MOBILE

Here we construct a simulator S_M that can simulate the view and output of a malicious MOBILE adversary M^*. Without loss of generality, this party represents any number of colluding MOBILE devices executing the protocol. We construct this simulator through a set of hybrid experiments.

$Hyb1^{(M)}(k, x, f; r)$: This experiment is identical to $REAL^{(M)}(k, x, f; r)$ except that the experiment terminates if any of the parties controlled by M^*s send an inconsistent seed s to CLOUD.

Lemma 14. $REAL^{(M)}(k, x, f; r) \stackrel{c}{\approx} Hyb1^{(M)}(k, x, f; r)$

Proof. In the real world protocol, CLOUD will terminate upon receiving inconsistent seeds from any MOBILE participant, thus, these two experiments are identical.

$Hyb2^{(M)}(k, x, f; r)$: This experiment is identical to $Hyb1^{(M)}(k, x, f; r)$ except that the experiment uses s to generate the input wire labels and recover M^*s actual input m^* to the computation. If any of the inputs are malformed, the experiment terminates. Otherwise, the experiment delivers m^* to the trusted third party and receives $y^* = f(m^*)$ as a result.

Lemma 15. $Hyb1^{(M)}(k, x, f; r) \stackrel{c}{\approx} Hyb2^{(M)}(k, x, f; r)$

Proof. In the real world protocol, if any party delivers a malformed garbled input, the circuit evaluation will fail and the protocol will terminate except with a negligible probability. Otherwise, the garbled input delivered here will be the input value used in computing the garbled circuit, so the output will match the ideal world output y^*.

$Hyb3^{(M)}(k, x, f; r)$: This experiment is identical to $Hyb2^{(M)}(k, x, f; r)$ except that the experiment uses s to generate the output wire labels and garbles y^* before returning the result to M^*.

Lemma 16. $Hyb2^{(M)}(k, x, f; r) \stackrel{c}{\approx} Hyb3^{(M)}(k, x, f; r)$

Proof. This hybrid holds since the garbled wire values in the real world protocol are generated deterministically based on s. In addition, since the arbitrary malicious input m^* was recovered and used in the ideal world computation, the output y^* will match in both the real and ideal execution.

Lemma 17. $Hyb3^{(M)}(k, x, f; r)$ *runs in polynomial time.*

Proof. This lemma follows trivially since a real world execution of the protocol runs in polynomial time and each intermediate hybrid adds operations that are polynomial with respect to input and output lengths.

We conclude the proof by letting S_M execute $Hyb3^{(M)}(k, x, f; r)$, following the real world protocol where the hybrid experiment does not differ. S_M runs M^* and controls CLOUD and APPLICATION. S_M simulates the real world protocol and outputs whatever M^* outputs at the end of the simulation. From Lemma 14-17, S_M proves Theorem 2 when the MOBILE parties are malicious.

C.2 Covert Cloud

Here we construct a simulator S_C that can simulate the view and output of a covert CLOUD adversary C^*. We construct this simulator through a set of hybrid experiments.

$Hyb1^{(C)}(k, x, f; r)$: This experiment is identical to $REAL^{(C)}(k, x, f; r)$ except that rather than garble the private function $f(\cdot)$ during the $Garble_{CTH}()$ functions, the experiment garbles a random function $f'(\cdot)$ such that $\Phi_{size}(f) = \Phi_{size}(f')$.

Lemma 18. $REAL^{(C)}(k, x, f; r) \overset{c}{\approx} Hyb1^{(C)}(k, x, f; r)$

Proof. This lemma follows from the proof of Mohassel and Sadeghian [29], which states that the view of the adversary C^* can be simulated indistinguishably by a simulator S_{CTH}. Even though the garbling party is covert, the output of the oblivious extended permutation (OEP) is always a set of uniformly random strings based on the blinding values used by the experiment to hide the circuit topology. Thus, C^* cannot distinguish between garbling $f(\cdot)$ and garbling $f'(\cdot)$.

$Hyb2^{(C)}(k, x, f; r)$: This experiment is identical to $Hyb1^{(C)}(k, x, f; r)$ except that the experiment invokes a simulator S_{OT} with a random input x' to simulate C^*s view of the oblivious transfer.

Lemma 19. $Hyb1^{(C)}(k, x, f; r) \overset{c}{\approx} Hyb2^{(C)}(k, x, f; r)$

Proof. This lemma holds based on the covert security of the OT protocol used. This guarantees that a simulator S_{OT} exists and can indistinguishably simulate C^*s view for any input value.

$Hyb3^{(C)}(k, x, f; r)$: This experiment is identical to $Hyb2^{(C)}(k, x, f; r)$ except that the experiment uses s, which it generated according to the protocol, to check all of the garbled circuits sent by C^* according to the checking procedure in the real protocol. The experiment does one of three things:

1. If all circuits, output tables, and the result of the OT are consistent and correct, the experiment continues.
2. If exactly one circuit is incorrectly garbled (or the output table is malformed, or the output of the OT for one circuit is incorrectly generated), the experiment sends `cheat` to the trusted third party. If the trusted party returns `caught`, the experiment terminates. Otherwise, it continues.
3. If more than one circuit is incorrectly garbled (or the output tables are malformed, or the output of the OT for more than one circuit is incorrectly generated), the experiment terminates.

Lemma 20. $Hyb2^{(C)}(k, x, f; r) \overset{c}{\approx} Hyb3^{(C)}(k, x, f; r)$

Proof. This hybrid essentially matches the probability of failure during the cut-&-choose for each scenario:

1. If C^* runs the protocol correctly, the cut-&-choose will pass and the protocol will continue.
2. If C^* corrupts only one circuit (or output table or OT result), there is a $\frac{1}{\lambda}$ probability that it will not be caught in the cut-&-choose. This matches the probability that the trusted third party will not return `caught` if we assume $\epsilon = \frac{1}{\lambda}$.
3. If C^* corrupts multiple circuits, it will always be caught during the cut-&-choose, and so the protocol will terminate in both the real and ideal execution.

Lemma 21. $Hyb3^{(C)}(k, x, f; r)$ *runs in polynomial time.*

Proof. This lemma follows since a real world execution of the protocol runs in polynomial time; the simulator S_{OT} runs in polynomial time and is invoked a fixed number of times with respect to input length; the simulator S_{CTH} runs in polynomial time; and checking an additional circuit incurs a polynomial addition of operations with respect to circuit size.

We conclude the proof by letting S_C execute $Hyb3^{(C)}(k, x, f; r)$, following the real world protocol where the hybrid experiment does not differ. S_C runs C^* and controls MOBILE and APPLICATION. S_C simulates the real world protocol and outputs whatever C^* outputs at the end of the simulation. From Lemma 18-21, S_C proves Theorem 2 when the CLOUD is covert.

C.3 Semi-honest Application server

The only difference between this protocol and our semi-honest protocol is the addition of the cut-&-choose, which essentially repeats the semi-honest garbling protocol multiple times. Thus, simulating the Application server's view in this protocol reduces easily to the proof in Appendix B.

D Malicious Protocol Proof

Here we prove the security of our outsourced PFE protocol against a malicious Cloud according to Theorem 3.

D.1 Malicious Mobile device

Here we construct a simulator S_M that can simulate the view and output of a malicious MOBILE adversary M^*. Without loss of generality, this party represents any number of colluding MOBILE devices executing the protocol. We construct this simulator through a set of hybrid experiments.

$Hyb1^{(M)}(k, x, f; r)$: This experiment is identical to $REAL^{(M)}(k, x, f; r)$ except that the experiment terminates if any of the parties controlled by M^*s send an inconsistent seed s to CLOUD

Lemma 22. $REAL^{(M)}(k, x, f; r) \stackrel{c}{\approx} Hyb1^{(M)}(k, x, f; r)$

Proof. In the real world protocol, CLOUD will terminate upon receiving inconsistent seeds from any MOBILE participant, thus, these two experiments are identical.

$Hyb2^{(M)}(k, x, f; r)$: This experiment is identical to $Hyb1^{(M)}(k, x, f; r)$ except that after M^* decommits his inputs, the experiment uses s to generate the input wire labels and recover M^*s actual input m^* to the computation. If any of the decommitted inputs are inconsistent or malformed, the experiment terminates. Otherwise, the experiment delivers m^* to the trusted third party and receives $y^* = f(m^*)$ as a result.

Lemma 23. $Hyb1^{(M)}(k, x, f; r) \stackrel{c}{\approx} Hyb2^{(M)}(k, x, f; r)$

Proof. In the real world protocol, if any party delivers inconsistent input, the 2-Universal hash will reveal the inconsistency with all but negligible probability. In addition, malformed garbled input will cause the circuit evaluation to fail and the protocol will again terminate except with a negligible probability. Otherwise, the garbled input delivered here will be the input value used in computing the garbled circuit, so the output will match the ideal world output y^*.

$Hyb3^{(M)}(k, x, f; r)$: This experiment is identical to $Hyb2^{(M)}(k, x, f; r)$ except that the experiment uses s to generate the output wire labels and garbles y^* before returning the result to M^*.

Lemma 24. $Hyb2^{(M)}(k, x, f; r) \stackrel{c}{\approx} Hyb3^{(M)}(k, x, f; r)$

Proof. This hybrid holds since the garbled wire values in the real world protocol are generated deterministically based on s. In addition, since the arbitrary malicious input m^* was recovered and used in the ideal world computation, the output y^* will match in both the real and ideal execution.

Lemma 25. *$Hyb3^{(M)}(k, x, f; r)$ runs in polynomial time.*

Proof. This lemma follows trivially since a real world execution of the protocol runs in polynomial time and each intermediate hybrid adds operations that are polynomial with respect to input and output lengths.

We conclude the proof by letting S_M execute $Hyb3^{(M)}(k, x, f; r)$, following the real world protocol where the hybrid experiment does not differ. S_M runs M^* and controls CLOUD and APPLICATION. S_M simulates the real world protocol and outputs whatever M^* outputs at the end of the simulation. From Lemma 22-25, S_M proves Theorem 3 when the MOBILE parties are malicious.

D.2 Malicious Cloud

Here we construct a simulator S_C that can simulate the view and output of a malicious CLOUD adversary C^*. We construct this simulator through a set of hybrid experiments.

$Hyb1^{(C)}(k, x, f; r)$: This experiment is identical to $REAL^{(C)}(k, x, f; r)$ except that rather than garble the private function $f(\cdot)$ during the $Gb_{MS}()$ functions, the experiment garbles a random function $f'(\cdot)$ where $\Phi_{size}(f') = \Phi_{size}(f)$.

Lemma 26. $REAL^{(C)}(k, x, f; r) \overset{c}{\approx} Hyb1^{(C)}(k, x, f; r)$

Proof. This lemma follows from the proof of Mohassel and Sadeghian [29], which constructs a simulator S_{CTH} that can simulate C^*s view of the real execution given only $\Phi_{size}(f)$. The output of S_{CTH} after the oblivious extended permutation (OEP) is a set of uniformly random strings based on the blinding values used by the experiment to hide the circuit topology. Thus, C^* cannot distinguish between garbling $f(\cdot)$ and garbling $f'(\cdot)$.

$Hyb2^{(C)}(k, x, f; r)$: This experiment is identical to $Hyb1^{(C)}(k, x, f; r)$ except that the experiment invokes a simulator S_{OT} to simulate C^*s view of the oblivious transfer.

Lemma 27. $Hyb1^{(C)}(k, x, f; r) \overset{c}{\approx} Hyb2^{(C)}(k, x, f; r)$

Proof. This lemma holds based on the malicious security of the OT protocol used. This guarantees that a simulator S_{OT} exists, can indistinguishably simulate C^*s view of a real execution, and can recover C^*s input to the OT w_i^0, w_i^1 for $i \in [1, ..., n]$.

$Hyb3^{(C)}(k, x, f; r)$: This experiment is identical to $Hyb2^{(C)}(k, x, f; r)$ except that the experiment uses s, which it generated according to the protocol, to check the garbled circuits sent by C^*. The experiment does one of three things:

1. If any of the circuits G_j, output tables d_j, and the result of the OT $w_{i,j}^0, w_{i,j}^1$ for $i \in [1, ..., n]$ and for $j \in [1, ..., \lambda] \setminus [E]$ are not consistent and correct, the experiment terminates.
2. If a majority of the circuits G_j, output tables d_j, and the result of the OT $w_{i,j}^0, w_{i,j}^1$ for $i \in [1, ..., n]$ and for $j \in [E]$ are not consistent and correct, the experiment terminates.
3. Otherwise, the experiment continues.

Lemma 28. $Hyb2^{(C)}(k, x, f; r) \overset{c}{\approx} Hyb3^{(C)}(k, x, f; r)$

Proof. This hybrid essentially matches the probability of failure during the cut-&-choose for each scenario:

1. If C^* corrupts a circuit in the check circuits $[1, ..., \lambda] \setminus [E]$, the experiment terminates.

2. If C^* corrupts a majority of the evaluation circuits in $[E]$, the experiment terminates. Based on the proof by shelat and Shen [36], this happens with negligible probability.
3. Otherwise, execution continues with a majority of evaluation circuits correctly constructed.

Lemma 29. *$Hyb3^{(C)}(k, x, f; r)$ runs in polynomial time.*

Proof. This lemma follows since a real world execution of the protocol runs in polynomial time; the simulator S_{OT} runs in polynomial time and is invoked a fixed number of times with respect to input length; S_{CTH} runs in polynomial time; and checking the additional circuits incurs a polynomial addition of operations with respect to circuit size.

We conclude the proof by letting S_C execute $Hyb3^{(C)}(k, x, f; r)$, following the real world protocol where the hybrid experiment does not differ. S_C runs C^* and controls MOBILE and APPLICATION. S_C simulates the real world protocol and outputs whatever C^* outputs at the end of the simulation. From Lemma 26-29, S_C proves Theorem 3 when the CLOUD is malicious.

D.3 Semi-honest Application server

Here we construct a simulator S_A that can simulate the view of a semi-honest APPLICATION adversary A^*. We construct this simulator through a set of hybrid experiments.

$Hyb1^{(A)}(k, x, f; r)$: This experiment is identical to $REAL^{(A)}(k, x, f; r)$ except that the experiment sends A^*s inputs $f(\cdot), x$ to the trusted third party.

Lemma 30. $REAL^{(A)}(k, x, f; r) \stackrel{c}{\approx} Hyb1^{(A)}(k, x, f; r)$

Proof. Since A^* is semi-honest, the experiment invokes the trusted party using the input function $f(\cdot)$ and the input data x that it provided to A^* at the start of the experiment. The value returned by the trusted third party $y = f(x)$ will be identical to the value output by the circuit evaluated in the real world.

$Hyb2^{(A)}(k, x, f; r)$: This experiment is identical to $Hyb1^{(A)}(k, x, f; r)$ except that the experiment commits to a random input m' rather than the real MOBILE input m.

Lemma 31. $Hyb1^{(A)}(k, x, f; r) \stackrel{c}{\approx} Hyb2^{(A)}(k, x, f; r)$

Proof. Indistinguishability here is based on two properties. First, the hiding commitment scheme, guarantees the commitments are indistinguishable between the ideal and real world. Second, the 2-Universal hash construction of shelat and Shen [37] guarantees the hashes output by the circuit for consistency checking are indistinguishable for any two inputs.

$Hyb3^{(A)}(k, x, f; r)$: This experiment is identical to $Hyb2^{(A)}(k, x, f; r)$ except that the experiment uses the random coins given to A^* to determine $[E]$. During the $Gb_{PCP}()$ execution, for all circuits $F \in [E]$, the experiment simulates the view of A^* using simulators S_1, S_{MS}, S_2.

Lemma 32. $Hyb2^{(A)}(k, x, f; r) \stackrel{c}{\approx} Hyb3^{(A)}(k, x, f; r)$

Proof. This follows from the prv.sim security of the preprocessing and post processing garbling schemes, as well as the prv.sim security of Gb_{MS} proven in Theorem 4, all of which guarantee a simulator S_1, S_{CTH}, S_2 that can simulate the adversary's view of a real circuit and garbled inputs given only Φ_{circ} and $y = f_2(f(f_1(x)))$.

Lemma 33. *$Hyb3^{(A)}(k, x, f; r)$ runs in polynomial time.*

Proof. This lemma follows trivially since the experiment runs the real world protocol and a set of simulators S_1, S_2, S_{CTH} that all run in polynomial time.

We conclude the proof by letting S_A execute $Hyb3^{(A)}(k, x, f; r)$, following the real world protocol where the hybrid experiment does not differ. S_A runs A^* and controls CLOUD and MOBILE. S_A simulates the real world protocol and outputs whatever A^* outputs at the end of the simulation. From Lemma 30-33, S_A proves Theorem 3 when the APPLICATION party is semi-honest.