Whitewash: Outsourcing Garbled Circuit Generation for Mobile Devices

Henry Carter
Georgia Institute of Technology
carterh@gatech.edu

Charles Lever
Georgia Institute of Technology
chazlever@gatech.edu

Patrick Traynor
Georgia Institute of Technology
traynor@cc.gatech.edu

Abstract

Garbled circuits offer a powerful primitive for computation on a user's personal data while keeping that data private. Despite recent improvements, constructing and evaluating circuits of any useful size remains expensive on the limited hardware resources of a smartphone, the primary computational device available to most users around the world. In this work, we develop a new technique for securely outsourcing the generation of garbled circuits to a Cloud provider. By outsourcing the circuit generation, we are able to eliminate the most costly operations from the mobile device, including oblivious transfers. After proving the security of our techniques in the malicious model, we experimentally demonstrate that our new protocol, built on this role reversal, decreases execution time by 98% and reduces network costs by as much as 92% compared to previous outsourcing protocols. In so doing, we demonstrate that the use of garbled circuits on mobile devices can be made nearly as practical as it is becoming for server-class machines.

1 Introduction

Mobile devices have become one of the dominant computing platforms, with approximately 57% market penetration in the United States alone [7]. These devices are capable of gathering and storing all of a user's personal data, from current location and social contacts to banking and electronic payment information. Because of the personal nature of these devices, it is critical that a user's information be protected at all times. Unfortunately, many smartphone applications that require users to send data to application servers make preserving the privacy of this data difficult.

To resolve this issue, a variety of secure multiparty computation techniques exist that could be leveraged to perform computation over encrypted inputs [26, 10, 9, 5]. Currently, the most practically efficient two-party technique is the Yao Garbled Circuit [39]. Despite recent improvements in the efficiency of garbled circuits [25, 37], this technique still requires significant computation and communication resources, rendering it impractical for most smartphones. One possible solution to this imbalance of resources is to blindly outsource the heavy computation to the Cloud. However, because of the untrusted nature of Cloud providers [38], such a solution fails to provide measurable guarantees for applications requiring high assurance.

In this work, we develop a new protocol for securely outsourcing garbled circuit generation to an untrusted Cloud. We construct a protocol that offloads the role of generating the garbled circuit from the mobile device to the Cloud without exposing any private inputs or outputs. By choosing to outsource this portion of the protocol, we eliminate a significant number of expensive public-key cryptography operations and rounds of communication used in oblivious transfer. The result is a more computationally and bandwidth efficient outsourcing protocol with improved security guarantees against malicious players [6, 21].

In this paper, we make the following contributions:

- **Develop a new outsourcing protocol:** We develop the Whitewash[1] outsourcing protocol, which allows a mobile device participating in a two-party secure function evaluation to outsource the generation of the garbled circuit.

[1] A reference to Tom Sawyer, who "outsourced" his chores to his friends without ever revealing the true nature of the work.

By reversing the roles of the two players in prior work [6, 21], we fully eliminate the requirement for any oblivious transfers, outsourced or otherwise, to or from the mobile device. This "simple" role reversal requires fundamentally redesigning the outsourcing techniques used in previous work, as well as new security proof formulations.

- **Formal verification and analysis:** We formally prove the security of our outsourcing techniques in the malicious model defined by Kamara et al. [21]. Unlike previous work [6, 21], our protocol provides security when the mobile device is colluding with its Cloud provider against the application server. We then provide an analysis of the reduction in operations between our work and the outsourced oblivious transfer of Carter et al. [6], as well as the Salus framework by Kamara et al. [21]. Specifically, our protocol requires more executions of a pseudorandom number generator in exchange for fewer algebraic group operations and zero-knowledge proofs. Moreover, we significantly reduce the number of rounds of communication required to the mobile device.
- **Implement and evaluate the performance of our protocol:** In our performance evaluation, we demonstrate a maximum improvement of 98% in execution time and 92% improvement in bandwidth overhead compared to Carter et al. [6]. For a different test application, when compared to performing computation directly on the mobile device [26], we demonstrated a 96% and 90% improvement in execution time and bandwidth, respectively. These improvements allow for the largest circuits evaluated on any platform to be computed from a mobile device efficiently and with equivalent security parameters to non-mobile protocols.

The rest of this work is organized as follows: Section 2 provides detail on related research; Section 3 describes our threat model and security definition; Section 4 provides a description of the Whitewash protocol; Section 5 compares the operations required in our protocol to the protocols by Carter et al. and Kamara et al.; Section 6 describes our empirical performance analysis; and Section 7 provides concluding remarks.

2 Related Work

Fairplay [33] provided the first practically efficient implementation of Yao's garbled circuit protocol [39], requiring only simple hash and symmetric key operations to securely evaluate an arbitrary function. Since then, a variety of garbled circuit-based secure function evaluation (SFE) protocols have been developed in the semi-honest adversarial model [4, 27, 18, 14, 19, 29, 32]. The latest of these, developed by Huang et al. [14], allows garbled circuits to be evaluated in stages, which makes it the most efficient semi-honest garbled circuit evaluation technique, both in computation and memory requirements. In recent work, several garbled circuit SFE protocols have been developed in the malicious security model, which require significantly more computational resources than semi-honest protocols, but are secure against arbitrary polynomial-time adversaries [31, 36, 35, 24, 26, 16, 28]. The protocol by shelat and Shen [37] provides a two-party garbled circuit protocol which uses only symmetric-key constructions outside of the oblivious transfer. When combined with Huang's pipelining approach and the PCF compiler by Kreuter et al. [25], their protocol is among the most efficient maliciously-secure garbled circuit protocols implemented to date. Some efforts have been made to improve the efficiency of these protocols by slightly reducing the adversary model. Many schemes have been developed in the covert adversary model, which allows for some efficiency gains at the cost of security [34, 8, 13, 2]. Huang et al. [15] developed a protocol that leaks only one bit of input to a malicious adversary through dual execution, which was later implemented on GPUs by Hustead et al. [17]. In order to further improve the efficiency of garbled circuit protocols, Gordon et al. [11] developed a protocol that combined Oblivious RAM with garbled circuits, allowing sub-linear amortized complexity. However, this protocol only allows this performance gain for functions that can be computed efficiently on a random-access machine.

To further improve the speed of cryptographic protocols on devices with minimal computational resources, the idea of outsourcing cryptographic operations has been explored for many years in the field of Server-assisted cryptography [3]. More recently, Green et al. [12] developed a technique for outsourcing the costly decryption of attribute-based encryption schemes to the cloud without revealing the contents of the ciphertext. Atallah and Frikken [1] developed a set of special-purpose protocols for securely outsourcing Linear Algebra computations to a single cloud server. For data mining applications, Kerschbaum recently developed an outsourced set intersection protocol using homomorphic encryption techniques [22]. While all of these applications provide significant performance gains for specific cryptographic applications, none of them address outsourcing of general secure computation.

In their Salus protocol, Kamara et al. [20, 21] developed two protocols for securely outsourcing the computation of arbitrary functions to the cloud. Following Salus, Carter et al. [6] developed an outsourcing protocol based on

the maliciously secure garbled circuit protocol by Kreuter et al. [26]. Carter's protocol outsources the evaluation of garbled circuits by adding in an Outsourced Oblivious Transfer primitive. Their participant configuration is the same configuration found in Kamara's maliciously secure protocol, where the cloud is made responsible for evaluating the garbled circuit. In this work, we choose to build on shelat and Shen's latest protocol [37] since the symmetric execution environment of Huang et al. [16] does not lend itself to outsourcing, and the bootstrapping technique used by Lindell [28] has not been implemented or evaluated in practice. Unlike previous work, we choose to fundamentally rearrange the roles of the participants, outsourcing the generation of the garbled circuits as in Kamara's covertly secure protocol. *Constructing a new protocol with this role configuration and the security guarantees of shelat and Shen's protocol allows us to develop a more efficient outsourcing protocol with stronger security guarantees than any previous outsourcing techniques.*

3 Overview and Security Definitions

3.1 Protocol Goals and Summary

The primary reason for developing an outsourcing protocol for secure function evaluation is to allow two parties of asymmetrical computing ability to securely compute some result. Current two-party computation protocols assume both parties are equipped with equivalent computing resources and so require both parties to perform comparable operations. However, when a mobile device is taking part in computation with an application server, some technique is necessary to reduce the complexity of the operation on the mobile device. Ideally, we can make the mobile device perform some small number of operations that is independent of the size of the circuit being evaluated.

In constructing such a protocol, there are three guarantees that we would like to provide. The first of these guarantees is correctness. It is necessary that an outsourcing protocol must produce correct output even in the face of malicious players attempting to corrupt the computation. The second desirable guarantee is security. SFE protocols frequently use a simulation-based approach to defining and proving security, which we outline in detail below. Essentially, the goal is to show that each party can learn the output of the computed function and nothing else. Finally, an ideal protocol would provide some guarantee of fair release. This guarantee ensures that either both parties receive their outputs from the computation, or neither party receives their output. Our protocol achieves this in all but one corruption scenario by treating the cloud as an arbiter, who will simultaneously and fairly release the outputs of the protocol using one-time pads. In the scenario where the mobile device and Cloud are colluding, it is possible for the Cloud to terminate the protocol after the mobile device receives output but before the application server receives output. However, this is inherently possible in most two-party garbled circuit protocols.

To achieve these guarantees, we first select the most efficient two-party garbled circuit computation protocol to date that provides guarantees of correctness and security in the malicious model. We assign the mobile device the role of circuit generator in this protocol, and the application server is assigned the role of circuit evaluator. To outsource the circuit generation operations from the mobile device, we allow the device to generate short random seeds and pass these values to a Cloud computation provider, which then generates the garbled circuits using these seeds to generate randomness. Thus, the mobile device's work is essentially reduced to (1) generating random strings on the order of a statistical security parameter, and (2) garbling and sending its input values to the evaluating party. In this way, we develop a secure computation protocol where the mobile device performs work that is independent of the size of the function being evaluated. We provide a formal proof of security for our protocol in Appendix A.

3.2 Non-collusion

To maintain security, previous outsourcing protocols assume that neither party colludes with the cloud [21, 6]. The theoretical intuition for this constraint, outlined by Kamara et al. [21], is that the existence of an outsourcing protocol where parties can arbitrarily collude would imply a two-party secure multiparty computation protocol where one party performs sub linear work with respect to the size of the circuit. While this has been shown to be possible in some cases [11], it is not clear that these techniques can be efficiently applied to an outsourcing scheme. Because of this, previous work has left the more complex security model for future study. However, while previous protocols restrict collusion between the cloud and any party, the sub-linear work implication only applies to cases when the cloud is generating circuits and colludes with the evaluating party, or vice versa. In the Whitewash protocol, we prove security when the mobile device colludes with the cloud against the evaluating web application. While this collusion scenario removes the fair release guarantee of our protocol, it in no way compromises the security guarantees of confidentiality

of participant's inputs and outputs. Essentially, it reduces to the two-party computation scenario that the underlying protocol is proven to be secure in. Since the mobile device is paying the Cloud for computation services, we believe it is a more realistic assumption to assume that a cloud provider could collude maliciously with the paying customer, and note that our protocol is the first outsourcing protocol to provide any security guarantees in the face of collusion with the Cloud.

3.3 Security Constructions

In the two-party computation protocol underlying our work, shelat and Shen implement a number of new and efficient cryptographic checks to ensure that none of the parties participating in the computation behave maliciously. We provide an overview of these security checks in the following section. We refer the reader to shelat and Shen's work for more formal definitions and proofs [37].

3.3.1 k-probe-resistant input encoding

When working with garbled circuit protocols in the malicious model, the generator has the ability to learn information about the evaluator's input by corrupting the wire labels sent during the oblivious transfer. This attack, known as selective failure, was first proposed by Mohassel and Franklin [35] as well as Kiraz and Schoenmakers [23]. To prevent this attack, shelat and Shen [37] implement an improved version of the k-probe-resistant input encoding mechanism originally proposed by Lindell and Pinkas [30]. In their protocol, the evaluator Alice does not input her real input y to the computation, but chooses her input $\overline{y}$ such that $\mathbf{M} \cdot \overline{y} = y$ for a k-probe resistant matrix $\mathbf{M}$. Intuitively, the idea is that the generator would have to probe Alice's input approximately 2^k times before learning anything about her real input y.

3.3.2 2-Universal Hash Function

A second concern with garbled circuits in the malicious model is that the generator may send different input values for each of the evaluated circuits from the cut-&-choose. To ensure that the generator's inputs are consistent across evaluation circuits, shelat and Shen implement an efficient witness-indistinguishable proof, which computes a randomized, 2-universal hash of the input value using only arithmetic operations on matrices. Because of the regularity guarantees of a 2-universal hash, the outputs of these hash operations can be seen by the evaluator without revealing any information about the generator's inputs. However, if any of the hashed input values is inconsistent across evaluation circuits, the evaluator can infer that the generator provided inconsistent inputs, and can terminate the protocol.

3.3.3 Output proof of consistency

When a function being evaluated using garbled circuits has separate, private outputs for the generating and evaluating parties, it is necessary to ensure that the evaluating party does not tamper with the generating party's output. Since the output must be decoded from the garbled output wires for the majority check at the end of the protocol, if the output is only blinded with a one-time pad, this allows the evaluator the opportunity to change bits of the generator's output. Several techniques for preventing this kind of tampering have been proposed, but shelat and Shen's latest protocol [37] implements a witness-indistinguishable proof that uses only symmetric key cryptographic operations. After the evaluator sends the output of computation to the generator, the proof guarantees to the generator that the output value he received was actually generated by one of the garbled circuits he generated. However, it keeps the index of the circuit that produced the output hidden, as this could leak information to the generator.

3.4 Security Model and Definition

Our definition of security is based on the definition proposed by Kamara et al. [21], which we specify for the two-party setting as in Carter et al. [6]. We provide a brief description of the real/ideal world model here and direct readers to the previous work in this space for a more formal definition.

In the real world, each party participating in the computation provides an input to the computation and an auxiliary input of random coins, while the single party designated as the outsourcing party provides only random auxiliary input. The evaluating party in this computation is assumed to be non-colluding with the outsourcing party, as defined by Kamara et al. Some subset of these parties $A = (A_1, ..., A_m), m \leq 3$ are corrupted and can deviate arbitrarily from

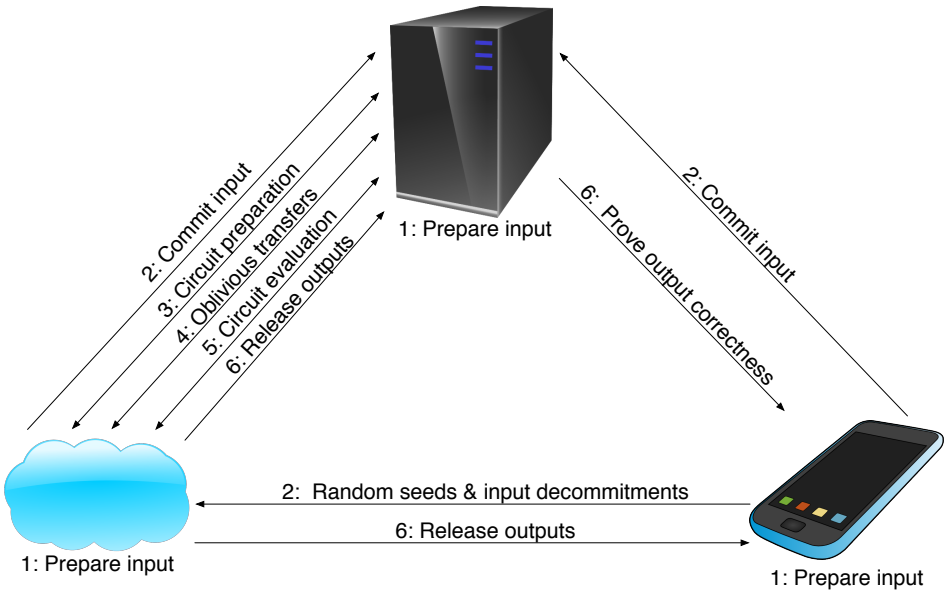

Figure 1: The complete whitewash protocol. Note that the mobile device (Bob) performs very little work compared to the web server (Alice) and the Cloud (Cloud).

the protocol. For the i^{th} honest party, let OUT_i be its output, and for the i^{th} corrupted party, let OUT_i be its view of the protocol execution. Then the i^{th} partial output of a real protocol execution with input x is defined as:

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

Where H is the set of honest parties, r is all random coins of all participants, and k is the security parameter.

In the ideal world, each party provides the same inputs as in the real world, however, they are sent to a trusted oracle which performs the secure computation. Once the trusted oracle completes the computation, it returns the output to the participating parties and no output to the outsourcing party. If any party aborts early or sends no input to the oracle, the oracle aborts and does not send the output to any party. For the i^{th} honest party, let OUT_i be its output to the oracle, and for the i^{th} corrupted party, let OUT_i be an arbitrary output value produced by the party. Then the i^{th} partial output of an ideal protocol execution in the presence of $m \leq 3$ independent malicious simulators $S = (S_1, S_2, S_3)$ is defined as:

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

Where H, r, and k are defined as before. Given this model, security is formally defined as:

Definition 1. *An outsourcing protocol securely computes the function f if there exists a set of probabilistic polynomial-time (PPT) simulators $\{Sim_i\}_{i\in[3]}$ such that for all PPT adversaries (A_1, A_2, A_3), inputs x, auxiliary inputs z, and for all $i \in [3]$:*

$$\{REAL^{(i)}(k, x; r)\}_{k\in N} \overset{c}{\approx} \{IDEAL^{(i)}(k, x; r)\}_{k\in N}$$

Where $S = (S_1, S_2, S_3)$, $S_i = Sim_i(A_i)$, and r is uniformly random.

4 Protocol

4.1 Participants

Given a mobile device and a web- or application- server who wish to jointly compute a function, there are three participating parties in the Whitewash protocol:

- **Alice:** We refer to the application server participating in the joint computation as "Alice." She is assumed to have large computational resources and is responsible for evaluating the garbled circuits.
- **Bob:** We refer to the mobile device participating in the joint computation as "Bob." He is assumed to have limited processing power, memory, and communication bandwidth.
- **Cloud:** We refer to the outsourcing party as "Cloud." Cloud is responsible for relieving Bob of the majority of his computational load, but is not trusted with knowing either party's input to or output from the joint computation.

4.2 Protocol

Common Inputs: Security parameters k (key length) and σ (the number of circuits generated for the cut-&-choose); a commitment scheme $com(x; c)$ with committed value x and commitment key c; and a function $f(x, y)$.
Private Inputs: Bob inputs x and Alice inputs y.
Outputs: Two outputs f_a, f_b for Alice and Bob, respectively.
Phase 1: Pre-computation

1. **Preparing inputs:** Bob randomly generates $r \in \{0,1\}^{2k+\log(k)}$ as his input to the 2-universal circuit. He also generates $e \in \{0,1\}^{|f_b|}$ as a one-time pad for his output. Alice computes her k-probe-resistant matrix $\mathbf{M}$ and $\overline{y}$ such that $\mathbf{M} \cdot \overline{y} = y$. Bob's input to the circuit will be $\overline{x} = x \| e \| r$ and Alice's input will be $\overline{y}$. We denote the set of indices $[m_a] = \{1, \cdots, |\overline{y}|\}$ and $[m_b] = \{1, \cdots, |\overline{x}|\}$.
2. **Preparing circuit randomness:** Bob generates random seeds $\{\rho^{(j)}\}_{j\in[\sigma]}$ for generating the circuits and sends them to Cloud.

Phase 2: Input commitments

1. **Committing to Bob's inputs:** For each circuit $j \in [\sigma]$, input bit $i \in [m_b]$, and $b \in \{0,1\}$ Bob uses $\rho^{(j)}$ to generate commitment keys $\theta_{i,b}^{(j)}$. These keys will later be generated by Cloud to commit to the input wire labels corresponding to Bob's input. Bob then commits to his own inputs as $\{\Gamma^{(j)}\}_{j\in[\sigma]}$ as:
$$\Gamma^{(j)} = \{com(\theta_{i,\overline{x}_i}^{(j)}; \gamma_i^{(j)})\}_{i\in[m_b]}$$
using independently generated random commitment keys $\gamma_i^{(j)}$. Bob sends $\{\Gamma^{(j)}\}_{j\in[\sigma]}$ to Alice and the commitment keys $\{\gamma_i^{(j)}\}_{i\in[m_b],j\in[\sigma]}$ to Cloud.
2. **Committing to Cloud's inputs:** To allow for a fair release of the outputs, the cloud inputs one-time pads to blind both parties' outputs. Cloud randomly generates $p_a \in \{0,1\}^{|f_a|}$ and $p_b \in \{0,1\}^{|f_b|}$, as well as $r_c \in \{0,1\}^{2k+\log(k)}$ as its input to the 2-universal circuit. We denote Cloud's input as $z = p_a \| p_b \| r_c$, and the indices of Cloud's input wires as $[m_c] = \{1, \cdots, |z|\}$.
For each circuit $j \in [\sigma]$ and input bit $i \in [m_c]$, Cloud uses $\{\rho^{(j)}\}_{j\in[\sigma]}$ to generates the garbled input wire keys $(K_{i,0}^{(j)}, K_{i,1}^{(j)}, \pi_i^{(j)})$, where $K_{i,0}^{(j)}, K_{i,1}^{(j)} \in \{0,1\}^k$ and the permutation bit $\pi_i^{(j)} \in \{0,1\}$. To locate the correct key for bit b on input wire w_i of circuit j, we designate the label $W_{i,b}^{(j)} = (K_{i,b}^{(j)}, b \oplus \pi_i^{(j)})$.
Let $\{w_{m_a+i}\}_{i\in[m_c]}$ be the input wires for Cloud. Cloud then commits to the label pairs for its input wires as $\{\Psi^{(j)}\}_{j\in[\sigma]}$, where
$$\Psi^{(j)} = \{com(W^{(j)}_{m_a+i,0\oplus\pi_i^{(j)}}; \psi^{(j)}_{i,0\oplus\pi_i^{(j)}}), com(W^{(j)}_{m_a+i,1\oplus\pi_i^{(j)}}; \psi^{(j)}_{i,1\oplus\pi_i^{(j)}})\}_{i\in[m_c]}$$
using commitment keys $\psi_{i,b}^{(j)}$ generated with the random seed $\rho^{(j)}$. Cloud then commits to its inputs as $\{\Xi^{(j)}\}_{j\in[\sigma]}$ as:
$$\Xi^{(j)} = \{com(\psi_{i,z_i}^{(j)}; \xi_i^{(j)})\}_{i\in[m_c]}$$
using independently generated random commitment keys. Cloud sends $\{\Psi^{(j)}\}_{j\in[\sigma]}$ and $\{\Xi^{(j)}\}_{j\in[\sigma]}$ to Alice.

Phase 3: Circuit construction

1. **Constructing the objective circuit:** Alice sends $\mathbf{M}$ to Cloud, then Alice and Cloud run a coin flipping protocol to randomly determine the 2-universal hash matrix $\mathbf{H} \in \{0,1\}^{k \times m_b}$. These two matrices can be used to generate the new circuit C that computes the function $g : (\overline{x}, \overline{y}) \rightarrow (\perp, (h_b, h_c, c_a, c_b))$, where $h_b = \mathbf{H} \cdot \overline{x}, h_c = \mathbf{H} \cdot z, g_b = f_b(x, \mathbf{M} \cdot \overline{y}), c_b = g_b \oplus e \oplus p_b, g_a = f_a(x, \mathbf{M} \cdot \overline{y})$, and $c_a = g_a \oplus p_a$. Bob will need the values $h_c \| c_b$ to recover his output. We denote the set of indices corresponding to these values as $O_b = \{1, \cdots, |h_c| + |c_b|\}$.

2. **Committing to input and output wire label pairs:** Using the same method as in Step 2, Cloud uses $\{\rho^{(j)}\}_{j \in [\sigma]}$ to generate the input wire keys for both Alice and Bob's input as well as the output wire keys for Bob's output (these output keys must be committed for the witness indistinguishable proof of Bob's output correctness). Let $\{w_i\}_{i \in [m_b]}$ be the input wires for Bob, $\{w_{m_b+i}\}_{i \in [m_a]}$ be the input wires for Alice, and $\{w_i\}_{i \in O_b}$. Cloud then commits to the label pairs in Bob's input, Alice's input, and Bob's Output as $\{\Theta^{(j)}, \Omega^{(j)}, \Phi^{(j)}\}_{j \in [\sigma]}$, where

$$\begin{aligned}
\Theta^{(j)} =& \{com(W^{(j)}_{i,0 \oplus \pi_i^{(j)}}; \theta^{(j)}_{i,0 \oplus \pi_i^{(j)}}), \\
& \quad com(W^{(j)}_{i,1 \oplus \pi_i^{(j)}}; \theta^{(j)}_{i,1 \oplus \pi_i^{(j)}})\}_{i \in [m_b]} \\
\Omega^{(j)} =& \{com(W^{(j)}_{m_b+i,0}; \omega_i^{(j)}), com(W^{(j)}_{m_b+i,1}; \omega_i^{(j)})\}_{i \in [m_a]} \\
\Phi^{(j)} =& \{com(W^{(j)}_{i,0}; \phi_i^{(j)}), com(W^{(j)}_{i,1}; \phi_i^{(j)})\}_{i \in O_b}
\end{aligned}$$

   using commitment keys generated with the random seed $\rho^{(j)}$. Cloud then sends these commitments to Alice.

Phase 4: Oblivious transfers

1. **Oblivious transfers:** The Cloud and Alice execute m_a input oblivious transfers and σ circuit oblivious transfers as follows:

   (a) **Input:** For each $i \in [m_a]$, both parties run a $\binom{2}{1}$-OT where Cloud inputs

$$\left(\{(W^{(j)}_{m_b+i,0}; \omega_i^{(j)})\}_{j \in [\sigma]}, \{(W^{(j)}_{m_b+i,1}; \omega_i^{(j)})\}_{j \in [\sigma]}\right)$$

   while Alice inputs $\overline{y}_i$. Once Alice receives all of her garbled input wire labels, she uses the decommitment keys obtained in the OTs to check the committed wire values in $\{\Omega^{(j)}\}_{j \in [\sigma]}$. If any of the labels received in the OT do not match the committed wire labels, Alice terminates the protocol.

   (b) **Circuit:** Alice selects a set of circuits to be evaluated $S \subset [\sigma]$ such that $|S| = \frac{2\sigma}{5}$, as in shelat and Shen's protocol [36]. She represents this set with a bit string $s \in \{0,1\}^\sigma$ such that the j^{th} bit $s_j = 1$ if $j \in S$ and $s_j = 0$ otherwise. Alice and Cloud perform σ $\binom{2}{1}$-OTs where, for every $j \in [\sigma]$, Cloud inputs $(\rho^{(j)}, (\{\gamma_i^{(j)}\}_{i \in [m_b]} \| \{\Xi_i^{(j)}\}_{i \in [m_c]}))$, while Alice inputs s_j. This allows Alice to learn either the randomness used to generate the check circuits or Bob and Cloud's inputs for the evaluation circuits without the cloud knowing which circuits are being checked or evaluated.

Phase 5: Evaluation

1. **Circuit evaluation:** Using $\rho^{(j)}$, Cloud garbles the objective circuit C as $G(C)^{(j)}$ for all $j \in [\sigma]$ and pipelines these circuits to Alice using Huang's technique [14]. Depending on whether the circuit is a check circuit or an evaluation circuit, Alice performs one of two actions:

   (a) **Check:** For each $j \in [\sigma] \backslash S$, Alice checks to see if $\rho^{(j)}$ can correctly regenerate the committed wire values $\{\Theta^{(j)}, \Omega^{(j)}, \Phi^{(j)}, \Psi^{(j)}\}$ and the circuit $G(C)^{(j)}$.

   (b) **Evaluate:** For each $j \in S$, Alice checks that she can correctly decommit Bob's input by recovering half of $\Theta^{(j)}$ from the keys committed in $\Gamma^{(j)}$. She does the same for Cloud's input, recovering half of $\Psi^{(j)}$ from the keys committed in $\Xi^{(j)}$

   If any of the above checks fail, Alice aborts the protocol. Otherwise, she evaluates the circuits $\{G(C)^{(j)}\}_{j \in [\sigma] \backslash S}$. Each circuit outputs the values $(h_b^{(j)}, h_c^{(j)}, c_a^{(j)}, c_b^{(j)})$ for $j \in [\sigma] \backslash S$.

Protocol	Symmetric ops	Asymmetric/group ops	Oblivious transfers	coin toss
CMTB	$\|x\|$	$\frac{2\sigma}{5}(\|y\|+1)$	k	yes
Salus	$\frac{2\sigma}{5}(\|x\|+\|y\|+\|f(x,y)\|)$	-	-	yes
Whitewash	$\sigma(\|x\|+\frac{2}{5}\|f_b(x,y)\|)$	-	-	no

Table 1: Operations required on the mobile device by three outsourcing protocols. Recall that k is the security parameter, σ is the number of circuits generated, x is the mobile device's input, and y is the application server's input.

2. **Majority output selection and consistency check:** Let (h_b, h_c, c_a, c_b) be the output of the majority of the evaluated circuits. If no majority value exists, Alice aborts the protocol. Otherwise, she checks that $h_b^{(j)} = h_b$ and $h_c^{(j)} = h_c$ for all $j \in [\sigma]\backslash S$. If any of Bob or Cloud's hashed input values do not match, Alice aborts the protocol.

Phase 6: Output proof and release

1. **Proof of output authenticity:** Alice and Bob perform the proof of authenticity from shelat and Shen's protocol [37] using the commitments to Bob's output wires $\{\Phi^{(j)}\}_{j\in[\sigma]\backslash S}$ and the values $h_c\|c_b$ which are to be proven correct.

2. **Output release:** Cloud simultaneously releases the input one-time pads p_a and p_b to Alice and Bob. Alice and Bob then hash the pads and check to see if the hash values output by the circuit $h_c = H \cdot p_a\|p_b$. If the hashes do not match, Alice and Bob abort the protocol. Otherwise, Alice receives $c_a \oplus p_a$ as her output and Bob receives $c_b \oplus p_b \oplus e$ as his output.

5 Comparison with previous outsourcing protocols

In this section, we compare the asymptotic complexity and security guarantees of the Whitewash protocol to two previous outsourcing techniques: the protocol developed by Carter et al. [6], which we call "CMTB" for the remainder of this work, and the Salus framework developed by Kamara et al. [21].

5.1 Comparison to CMTB

The underlying two-party computation protocols of Whitewash and CMTB follow similar structures in terms of the security checks that are performed. However, the KSS protocol, which underlies CMTB, uses a number of algebraic operations to perform input consistency checks and output proofs of consistency. The protocol developed by shelat and Shen [37], which underlies Whitewash, removes these expensive cryptographic primitives in favor of constructions that use only efficient, symmetric-key operations. In addition to the improvements to the underlying protocol, Whitewash outsources the generation side of two-party computation, while CMTB outsources the evaluation side. In CMTB, since neither the mobile device or the cloud could garble inputs before computation, a specially designed Outsourced Oblivious Transfer (OOT) protocol is necessary to deliver the mobile device's inputs to the evaluating cloud in a secure, privacy-preserving manner. By swapping roles in the Whitewash protocol, we allow the mobile device to garble its own inputs, removing the need for any oblivious transfer protocol to be performed from the mobile device. While Whitewash still requires OTs between the cloud and the evaluating party, these operations can be parallelized, while the OOT protocol acts as a non-parallelizable bottleneck in computation.

5.1.1 Asymptotic Complexity

When examining the complexity of each protocol, our main goal is to optimize the efficiency on the mobile device. Thus, we examine the number of operations each protocol requires on the mobile device itself. Table 1 shows this complexity for both Whitewash and CMTB. Note that for the mobile device, Whitewash requires significantly more symmetric key operations for garbling its own input and verifying the correctness of its output. By contrast, the OOT protocol in CMTB requires very few symmetric key operations, but requires several instantiations of an oblivious transfer. In addition, CMTB requires that the mobile device check the application server's input consistency and verify the correctness of the output using algebraic operations (e.g., modular exponentiations and homomorphic operations).

Considering the fact that modular exponentiation is significantly more costly than symmetric key operations, removing these public key operations from the phone is a significant efficiency improvement for Whitewash. We also note that CMTB requires a two-party fair coin toss at the mobile device, which is not required by Whitewash.

5.1.2 Security Guarantees

The removal of the OOT protocol in Whitewash not only increases its efficiency when compared to CMTB, it also allows for stronger security guarantees. In CMTB, security was only possible if none of the parties collude, since the mobile device possessed information that would allow the Cloud to recover both input wire labels for all of the mobile input wires after the OOT. If the mobile device and cloud collude in the Whitewash protocol, it simply removes the guarantee of fair release and makes the protocol equivalent to the underlying two-party computation protocol. Thus, the only guarantee lost is that of fair release at the end of the protocol, since a colluding mobile device and cloud may not release the one-time pad used to blind the evaluating party's output. We believe that this represents a more realistic security setting, since the mobile device is paying for the assistance of the Cloud and may collude.

5.2 Comparison to Salus

When considering the operations performed on the mobile device, the Salus protocol and the Whitewash protocol both make the mobile device responsible for generating circuit randomness and garbling its own inputs. However, the Whitewash protocol requires an added proof of output consistency that is not included in Salus. While this proof adds some complexity to the protocol, it allows Whitewash to handle functions where both parties get different output values, while Salus is designed to handle functions with a single, shared output value. In addition, the Whitewash protocol outsources the generation of the garbled circuit, while the malicious secure Salus protocol outsources the evaluation. By swapping the roles of the outsourced task and adding in consistency checks at the evaluating party, the Whitewash protocol guarantees security in a stronger adversarial model.

5.2.1 Asymptotic Complexity

Table 1 shows the number of operations performed on the mobile device for both Salus and Whitewash. Both protocols use only efficient, symmetric key operations, but there is a slight tradeoff in the number of operations required. Salus only requires operations for the $\frac{2\sigma}{5}$ evaluated circuits, but requires those operations for each bit of both party's inputs and the shared output. By contrast, Whitewash requires that the mobile device's input be committed for all σ circuits generated, but then only requires correctness proof of the output wires on the $\frac{2\sigma}{5}$ evaluated circuits. When the application server's input is significantly longer than the mobile device's, this will cause the Salus protocol to be less efficient than Whitewash. However, in the average case where both inputs are approximately the same length, this will mean that Whitewash requires more operations. This small tradeoff in efficiency is justified by the fact that Whitewash provides security in a stronger adversarial model than Salus. We also note that Salus requires a two-party fair coin toss before the protocol begins, which is not required by Whitewash.

5.2.2 Security Guarantees

The Salus protocol provides equivalent security guarantees to CMTB, guaranteeing security when none of the parties are colluding. This is a result of outsourcing the evaluation to the Cloud while allowing the mobile device to generate circuit randomness. If the mobile device colludes with the cloud, they can trivially recover all of the other party's inputs. By outsourcing the generation of the garbled circuit and adding in additional consistency checks at the evaluating party, Whitewash guarantees security under this type of collusion. As stated above, the only guarantee lost is that of fair output release, which ultimately reduces Whitewash to the security of the underlying two-party computation protocol.

6 Performance Evaluation

Our protocol significantly expands upon the implementations of the PCF garbled circuit generation technique [25] and shelat and Shen's garbled circuit evaluation protocol [37]. For experimental comparison to previous protocols, we acquired the code implementation of the outsourcing protocol by Carter et al. [6] directly from the authors, as

Circuit	Input Size (Bits)	Total Gates		Non-XOR Gates	
		KSS	PCF	KSS	PCF
Hamming (1600)	1,600	24,379	32,912	17,234	6,375
Hamming (16384)	16,384	262,771	376,176	186,326	101,083
Matrix (3x3)	288	424,748	92,961	263,511	27,369
Matrix (5x5)	800	1,968,452	433,475	1,221,475	127,225
Matrix (8x8)	2,048	8,067,458	1,782,656	5,006,656	522,304
Matrix (16x16)	8,192	64,570,969	14,308,864	40,076,631	4,186,368
Dijkstra's 10	112/1,040	259,232	530,354	118,357	291,490
Dijkstra's 20	192/2,080	1,653,380	2,171,088	757,197	1,192,704
Dijkstra's 50	432/5,200	22,109,330	13,741,514	10,170,407	7,549,370
RSA-256	256/512	934,092,960	673,105,990	602,006,981	235,925,023

Table 2: Input size and circuit size for all test circuits evaluated.

well as an Android port of the two-party garbled circuit protocol developed by Kreuter, shelat, and Shen [26]. For the remainder of the work, we refer to these protocols as CMTB and KSS respectively. We refer to our Whitewash protocol as WW. We would like to thank the authors of [6, 26, 25, 37] for making their code available and for assisting us in running this performance evaluation[2].

6.1 Test Environment

For evaluating our test circuits, we perform our experiments with a single server performing the role of Cloud and Application server, communicating with a mobile device over an 802.11g wireless connection. The server is equipped with 64 cores and 1TB of memory, and we partition the work between cores into parallel processing nodes using MPI. The mobile device used is a Samsung Galaxy Nexus with a 1.2 GHz dual-core ARM Cortex-A9 processor and 1 GB of RAM, running Android 4.0.

The large input sizes examined in the Hamming Distance trials required us to use a different testbed. For inputs as large as 16,384 bits, the phone provided by the above computing facility would overheat and fail to complete computation. Because the gate counts for Hamming Distance are significantly smaller than the other test circuits, we were able to run these experiments on a local testbed. We used two servers with Dual Intel Xeon E5620 processors, each with 4 hyper-threaded cores at 2.4 GHz each for the Cloud and the application server. Each server is running the Linux kernel version 2.6, and is connected by a VLAN through a 1 Gbps switch. Our mobile device is a Samsung Galaxy Note II with a 1.6 GHz quad-core processor with Arm Cortex A9 cores and 2 GB of RAM, running the Android operating system at version 4.1. The phone connects to the two servers through a Linksys 802.11g wireless router with a maximum data rate of 54 Mbps. While this test environment represents optimistic connection speeds that may not always be available in practice, it allows us to consider the performance of the protocol without interference from variable network conditions, and mirrors the test environments used in previous work [26, 6, 37]. For all experiments except RSA-256, we take the average execution time over ten test runs, with a confidence interval of 95%. For RSA-256, we ran 3 executions.

6.2 Experimental Circuits

To evaluate the performance of our protocol, we run tests over the following functions. We selected the following test circuits because they exercise a range of the two major variables that affect the speed of garbled circuit protocols: input size and gate counts. In addition, these programs are becoming somewhat standard test applications, having been used as benchmarks in a large amount of the related literature [26, 6, 37, 25]. All of the programs are implemented with the algorithms used by Kreuter et al. [25] except for Dijkstra's algorithm, which matches the implementation used by Carter et al. [6]:

[2]We contacted the authors of the Salus protocol [21] in an attempt to acquire their framework to compare the actual performance of their scheme with ours. Because they were unable to release their code, no sound comparison to their work beyond an asymptotic analysis was possible. Our code will be made available immediately on publication.

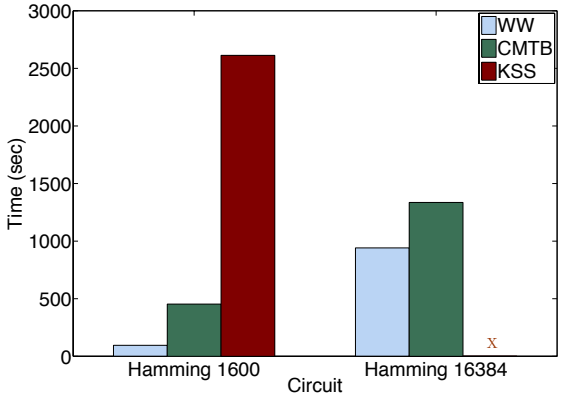

Figure 2: Execution time (ms) for Hamming Distance with input sizes of 1,600 and 16,384 bits for $\sigma = 256$ (note: log scale). Note that without outsourcing, only very small inputs can be computed over. Additionally, even for a large number of input bits, performing OTs on the servers still produces a faster execution time.

1. **Hamming Distance:** The Hamming Distance circuit accepts binary string inputs from both parties and outputs the number of locations at which those strings differ. This circuit demonstrates performance for a small number of gates over a wide range of input sizes. We consider input strings of length 1,600 bits and 16,384 bits.
2. **Matrix Multiplication:** Matrix multiplication takes an $n \times n$ matrix of 32-bit integer entries from each party and outputs the result of multiplying the matrices together. This circuit demonstrates performance when both input size and gate count vary widely. We consider square matrix inputs where $n = 3, 5, 8$, and 16.
3. **Dijksrta's Algorithm:** This version of Dijkstra's algorithm takes an undirected weighted graph with a grid structure and a maximum node degree of four from the first party, and a start and end node from the second party. The circuit outputs the shortest path from the start node to the end node to the second party, and nothing to the first. For an n node graph, the graph description from the first party requires $104n$ input bits, while the start and end node descriptions require $8n + 32$ bits. We consider graphs with $n = 10, 20$, and 50 nodes. Due to an error in the PCF compiler, we were unable to compile a program for graphs larger than 50 nodes.
4. **RSA Function:** The RSA function (i.e., modular exponentiation) accepts an RSA message from one party and an RSA public key from the other party and outputs the encryption of the input plaintext under the input public key. Specifically, one party inputs the modulus $n = pq$ for primes p and q, as well as the encryption key $e \in \mathbb{Z}_{\phi(n)}$. The other party inputs a message $x \in \mathbb{Z}_n^*$, and the circuit computes $x^e \pmod n$. This circuit demonstrates performance for small input sizes over very large gate counts. We consider the case where the input values x, n, and e are 256 bits each.

For each test circuit, we consider the time required to execute and the bandwidth overhead to the mobile device. Table 2 shows the input size and gate counts for each test circuit, showing the exact range of values tested for these two circuit variables.

6.3 Execution Time

In all experiments, the efficiency gains of removing oblivious transfers and public key operations are immediately apparent. To examine how Whitewash compares to generating garbled circuits directly on the mobile device, we considered Hamming Distance as a simple problem (Figure 2). Even with a relatively small gate count, garbling the circuit directly on the mobile device is only possible for the small input size of 1,600 bits. Whitewash is capable of executing this protocol in 96 seconds, while running the computation directly on the mobile device takes 2,613 seconds, representing a 96% performance improvement through our outsourcing scheme. For the very large input size of 16,384 bits, computation directly on the mobile device ceases to be possible. When comparing to CMTB, this circuit further illustrates the cost of oblivious transfers on the mobile device. Even with the significantly reduced number of OTs allowed by the OOT protocol in CMTB (80 OTs), performing 16,384 malicious secure oblivious transfers between two servers in Whitewash still runs 30% faster than CMTB.

The matrix-multiplication circuit provides a good overview of average-case garbled circuit performance, as it

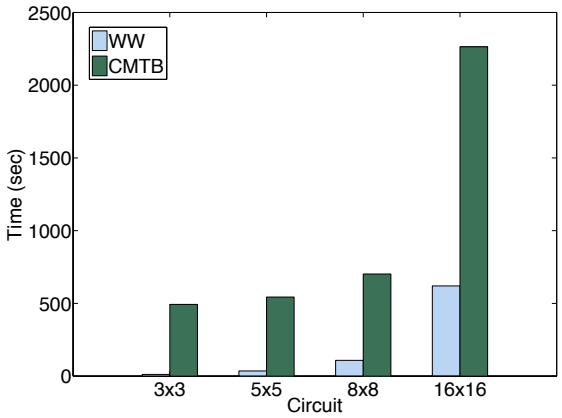

Figure 3: Execution time (ms) for the Matrix-Multiplication problem with input size varying between 3×3 matrices and 16×16 matrices for $\sigma = 256$ (note: log scale). This figure clearly shows that the oblivious transfers, consistency checks, and larger circuit representations of CMTB add up to a significant overhead as input size and gate count increase. By contrast, Whitewash requires less overhead and increases more slowly in execution time as gate counts and input size grow.

represents a large range of both gate counts and size of inputs. For the input size of a 3×3 matrix, the Whitewash protocol runs in an average of 12 seconds, while CMTB requires 493 seconds, representing a 98% improvement (see Figure 3). Upon inspecting the micro benchmarking breakdown of each protocol's execution in Figure 4, we observe a significant speedup simply by moving oblivious transfers off of the mobile device. Even though the number of OTs required by CMTB is essentially constant based on their application of the Ishai OT extension, performing standard malicious secure oblivious transfers in parallel between the servers is much more efficient than requiring that the phone perform these costly operations. In addition, if we examine the amount of execution time where the phone participates in Whitewash, we see that the mobile device ("MOBI" in Figure 4), takes around 1 second, and is idle during the majority of computation. By contrast, both the OT and consistency check phases of CMTB require the mobile device to participate in a significant capacity, totaling to almost 8 minutes of the computation. Having the phone perform as little computation as possible means that the Whitewash protocol performance is nearly equivalent to performing the same computation between two server-class machines.

To examine the performance of Whitewash for a more practical application, we considered the Dijkstra's algorithm circuit used to implement privacy-preserving navigation by Carter et al. [6]. They point out that this application, which has uses from military convoys to industrial shipping routes, is a significant first step in providing privacy for the growing genre of location-based mobile applications (see an example in Figure 5). Unfortunately, the PCF compiler does not optimize the Dijkstra's circuit as well as the previous experimental programs, which is evident in Table 2. In the 10 and 20 node graphs, the PCF compiler even produces a *larger* circuit than the compiler used by KSS. However, despite evaluating larger circuits, the Whitewash protocol still outperforms CMTB in execution time, running 88%,76%, and 51% faster in the $10, 20$, and 50 node cases respectively (shown in Figure 6). As circuit compilers continue to improve and produce smaller circuits, the performance gains of the Whitewash protocol will be even larger. In this experiment, we also noticed that because Whitewash evaluates and checks circuits simultaneously, it created contention for the network stack in our test server. In a truly distributed environment where each server node has dedicated network resources, the highly parallelizable structure of shelat and Shen's protocol would allow Whitewash to execute faster. Given that Whitewash can execute Dijkstra's algorithm obliviously on the order of minutes, it allows computation considered only feasible for previous schemes to be performed in a nearly practical execution time.

The previous experiments clearly show that outsourcing is necessary to run circuits of any practical size. For our final test circuit, we consider an extremely complex problem to demonstrate the ability of outsourcing protocols in the worst-case. The RSA-256 circuit evaluated by Kreuter et al. in [25] and shelat and Shen in [37] represents one of the largest garbled circuits ever evaluated by a malicious secure protocol. For the RSA-256 problem, Whitewash completed the computation in 515 minutes. CMTB was unable to complete one execution of the protocol. A large

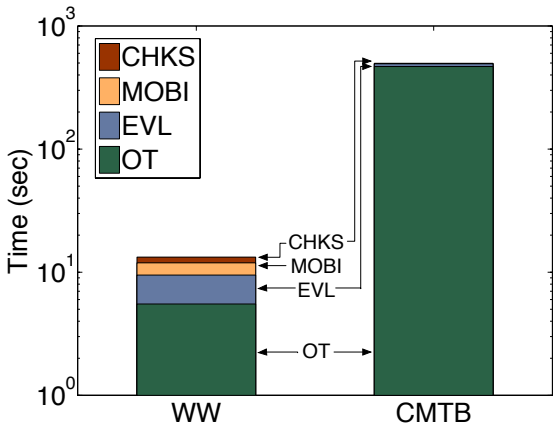

Figure 4: Microbenchmarking execution times (ms) for Whitewash and CMTB over the Matrix-Multiplication problem. We denote the total time spent in computation for Whitewash as "MOBI". Since the mobile device is linked with "CHKS" and "OT" in CMTB, we do not separate out the mobile time for that protocol. Notice the dominating amount of time required to perform oblivious transfers. Moving these operations off the mobile device removes a significant computation bottleneck.

	Bandwidth (MB)			Reduction Over	
Circuit	WW	CMTB	KSS	CMTB	KSS
Hamming (1600)	23.56	41.05	240.33	42.62%	90.20%
Hamming (16384)	241.02	374.03	x	35.56%	x
Matrix (3x3)	4.26	11.50	x	62.97%	x
Matrix (5x5)	11.79	23.04	x	48.82%	x
Matrix (8x8)	30.15	51.14	x	41.05%	x
Matrix (16x16)	120.52	189.52	x	36.41%	x
Dijkstra's 10	1.67	20.21	x	91.73%	x
Dijkstra's 20	2.85	35.28	x	91.93%	x
Dijkstra's 50	6.38	80.49	x	92.08%	x
RSA-256	3.97	x	x	x	x

Table 3: Bandwidth measures for all experiment circuits. Note that there is as much as a 84% reduction in bandwidth when using the Whitewash protocol.

part of this efficiency improvement results from the underlying protocol of Whitewash, which uses only symmetric-key operations outside of the oblivious transfers between the servers. The reduced non-XOR gate counts and more compact circuit representation of the PCF compiler also contribute to this improvement. Ultimately, because Whitewash ensures that the phone participates minimally in the protocol, it no longer acts as a bottleneck on computation. We essentially reduce performance of our outsourcing protocol to that of the underlying two-party protocol, allowing this technique for outsourcing to benefit as more improvements are made in non-outsourced garbled circuit protocols. In addition, this minimal level of interactivity allows us to run these protocols with 256 circuits, equivalent to a security parameter of approximately 80-bit security, which is agreed by the research community to be an adequate security parameter. Finally, since the phone is active for mere seconds during this large computation, its system resources are free for other user applications while the servers complete the computation. *This shows that Whitewash is capable of evaluating the same circuits as the most efficient desktop-based garbled circuit protocols with a minimal overhead cost.* For full experimental results, see Appendix B.

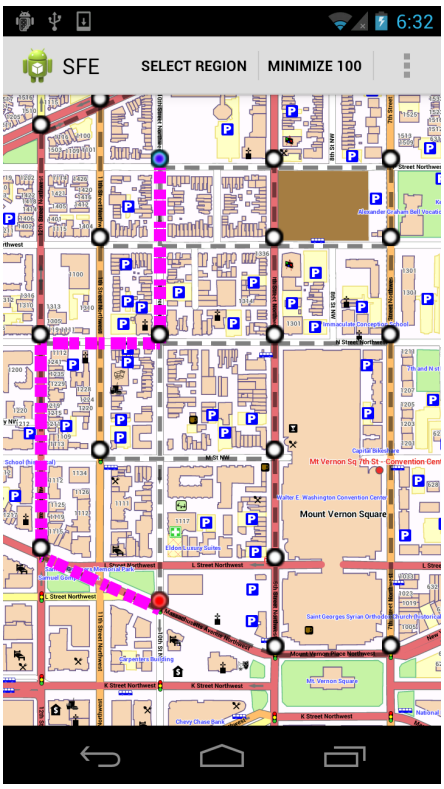

Figure 5: An example of the privacy-preserving navigation application of CMTB [6] retrofitted to use the Whitewash protocol. Such an application would allow a mobile user to compute the fastest route between two points using a mapping service that possesses potentially proprietary or secret geographic information. This application has many uses in military (troop movement), government (dignitary motorcade), and commercial (supply chain) settings.

6.4 Network Bandwidth

The Whitewash protocol not only improves the speed of execution when outsourcing garbled circuit computation, it also significantly reduces the amount of bandwidth required by the mobile device to participate in the computation. Table 3 shows the bandwidth used by the mobile device for each test circuit. In the best case, for Dijkstra's algorithm over 50 node graphs, we observed a 92% reduction in bandwidth between Whitewash and CMTB. This is a result of the mobile device not performing OTs and only sending relatively small symmetric-key values instead of algebraic elements for consistency checks. For all test circuits, we observed a small decrease in the amount of improvement between the two protocols as the input size increased. This is because the number of commitments sent by the phone in Whitewash increases as the size of the input grows, while CMTB performs a fixed number of OTs as the input size increases. However, the obvious transfers still require a significant enough amount of bandwidth to make removing them the most efficient option. When comparing to not outsourcing garbled circuit generation, the cost of oblivious transfers and sending several copies of the garbled circuit to the evaluator quickly adds up to a significant bandwidth cost. For the smallest circuit evaluated, outsourcing the circuit garbling reduces the required amount of bandwidth by 90%. The importance of these bandwidth reductions is further highlighted when considering mobile power savings. With data transmission costing roughly 100 times as much power as computation on the same amount of data, any reduction in the bandwidth required by a protocol implies a critical improvement in practicality.

One challenge encountered during the implementation of the Whitewash protocol was the extensive use of hardware-specific functions used to implement commitment schemes in shelat and Shen's code. Rather than try to port this code over to Android, which would require significant development of hardware-specific libraries, we chose to implement the protocol in an equivalently secure manner by having the Cloud generate part of the commitments (which requires these functions) and send them to the mobile device. The mobile device then finishes generating the commitments that match its input and forwards them to the evaluator. If we were to implement these machine-specific instruction, we

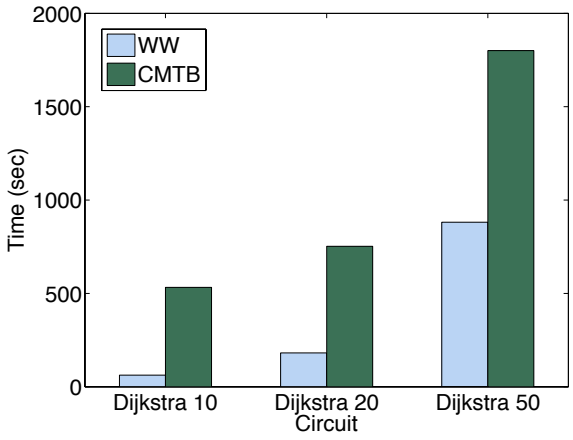

Figure 6: Execution time (s) for Dijkstra's algorithm with input sizes of $10, 20$, and 50 node graphs for $\sigma = 256$. This figure shows that the Whitewash protocol allows for computation that was only feasible to be executed in a close to practically useful time frame.

could further reduce the measured bandwidth values by over 60%. With already significant bandwidth reductions from previous outsourcing schemes, our protocol will see further improvements as mobile hardware begins to incorporate these machine-specific libraries in the next few years.

7 Conclusion

With the increasingly pervasive and personal nature of mobile computing, garbled circuits provide a solution that preserves both user privacy and application functionality. However, to make these computationally expensive protocols usable on mobile devices, secure outsourcing to the cloud is necessary. We develop a new scheme that eliminates the most costly operations, including oblivious transfers, from the mobile device. By requiring that the mobile device instead produce the randomness required for circuit generation, we significantly reduce the number of algebraic group operations and communication rounds for the mobile device. Our performance evaluation shows performance gains as high as 98% for execution time and 92% for bandwidth over the previous outsourcing protocol. These improvements allow large circuits representing practical applications to be computed efficiently from a mobile device. As a result, we show that the use of garbled circuits can be made nearly as efficient for mobile devices as it is becoming for server-class machines.

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

References

[1] ATALLAH, M. J., AND FRIKKEN, K. B. Securely outsourcing linear algebra computations. In *Proceedings of the ACM Symposium on Information, Computer and Communications Security (ASIACCS)* (2010).

[2] AUMANN, Y. Security Against Covert Adversaries: Efficient Protocols for Realistic Adversaries. *Journal of Cryptology 18*, 3 (2010), 554–343.

[3] BEAVER, D. Server-assisted cryptography. In *Proceedings of the workshop on New security paradigms (NSPW)* (1998).

[4] BRICKELL, J., AND SHMATIKOV, V. Privacy-preserving graph algorithms in the semi-honest model. In *Proceedings of the international conference on Theory and Application of Cryptology and Information Security* (2005).

[5] CARTER, H., AMRUTKAR, C., DACOSTA, I., AND TRAYNOR, P. For your phone only: custom protocols for efficient secure function evaluation on mobile devices. *Journal of Security and Communication Networks (SCN)* (To appear 2014).

[6] CARTER, H., MOOD, B., TRAYNOR, P., AND BUTLER, K. Secure Outsourced Garbled Circuit Evaluation for Mobile Devices. In *Proceedings of the USENIX Security Symposium* (2013).

[7] COMSCORE. comScore Reports February 2013 U.S. Smartphone Subscriber Market Share. `http://www.comscore.com/Insights/Press_Releases/2013/4/comScore_Reports_February_2013_U.S._Smartphone_Subscriber_Market_Share`, 2013.

[8] DAMGÅRD, I., GEISLER, M., AND NIELSEN, J. B. From passive to covert security at low cost. In *Proceedings of the 7th international conference on Theory of Cryptography* (2010).

[9] DAMGARD, I., PASTRO, V., SMART, N., AND ZAKARIAS, S. Multiparty computation from somewhat homomorphic encryption. In *Proceedings of the Annual International Cryptology Conference on Advances in Cryptology* (2012).

[10] GENTRY, C., HALEVI, S., AND SMART, N. P. Homomorphic evaluation of the AES circuit. In *Advances in Cryptology - CRYPTO* (2012).

[11] GORDON, S. D., KATZ, J., KOLESNIKOV, V., LABS, A.-L. B., KRELL, F., AND RAYKOVA, M. Secure Two-Party Computation in Sublinear (Amortized) Time. In *Proceedings of the ACM conference on Computer and communications security (CCS)* (2012).

[12] GREEN, M., HOHENBERGER, S., AND WATERS, B. Outsourcing the Decryption of ABE Ciphertexts. In *Proceedings of the USENIX Security Symposium* (2011).

[13] HAZAY, C., AND LINDELL, Y. Efficient Protocols for Set Intersection and Pattern Matching with Security Against Malicious and Covert Adversaries. *Journal of Cryptology 23*, 3 (2008), 422–456.

[14] HUANG, Y., EVANS, D., KATZ, J., AND MALKA, L. Faster Secure Two-Party Computation Using Garbled Circuits. In *Proceedings of the USENIX Security Symposium* (2011).

[15] HUANG, Y., KATZ, J., AND EVANS, D. Quid-pro-quo-tocols: Strengthening semi-honest protocols with dual execution. In *Proceedings of the IEEE Symposium on Security and Privacy* (2012).

[16] HUANG, Y., KATZ, J., AND EVANS, D. Efficient secure two-party computation using symmetric cut-and-choose. In *Advances in Cryptology–CRYPTO* (2013).

[17] HUSTEAD, N., MYERS, S., ABHI SHELAT, AND GRUBBS, P. GPU and CPU parallelization of honest-but-curious secure two-party computation. In *Proceedings of the Annual Computer Security Applications Conference (ACSAC)* (2013).

[18] ILIEV, A., AND SMITH, S. W. Small, Stupid, and Scalable: Secure Computing with Faerieplay. In *The ACM Workshop on Scalable Trusted Computing* (2010).

[19] JHA, S., KRUGER, L., AND SHMATIKOV, V. Towards practical privacy for genomic computation. In *Proceedings of the IEEE Symposium on Security and Privacy* (2008).

[20] KAMARA, S., MOHASSEL, P., AND RAYKOVA, M. Outsourcing multi-party computation. Cryptology ePrint Archive, Report 2011/272, 2011. `http://eprint.iacr.org/`.

[21] KAMARA, S., MOHASSEL, P., AND 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] KERSCHBAUM, F. Collusion-resistant outsourcing of private set intersection. In *Proceedings of the ACM Symposium on Applied Computing* (2012).

[23] KIRAZ, M., AND SCHOENMAKERS, B. A Protocol Issue for The Malicious Case of Yao's Garbled Circuit Construction. In *Proceedings of the Symposium on Information Theory in the Benelux* (2006).

[24] KIRAZ, M. S. *Secure and Fair Two-Party Computation*. PhD thesis, Technische Universiteit Eindhoven, 2008.

[25] KREUTER, B., SHELAT, A., MOOD, B., AND BUTLER, K. PCF: A portable circuit format for scalable two-party secure computation. In *Proceedings of the USENIX Security Symposium* (2013).

[26] KREUTER, B., SHELAT, A., AND SHEN, C. Billion-Gate Secure Computation with Malicious Adversaries. In *Proceedings of the USENIX Security Symposium* (2012).

[27] KRUGER, L., JHA, S., GOH, E.-J., AND BONEH, D. Secure Function Evaluation with Ordered Binary Decision Diagrams. In *Proceedings of the ACM conference on Computer and communications security (CCS)* (2006).

[28] LINDELL, Y. Fast cut-and-choose based protocols for malicious and covert adversaries. In *Advances in Cryptology–CRYPTO* (2013).

[29] LINDELL, Y., AND PINKAS, B. Privacy preserving data mining. In *Proceedings of the Annual International Cryptology Conference on Advances in Cryptology* (2000).

[30] LINDELL, Y., AND PINKAS, B. An efficient protocol for secure two-party computation in the presence of malicious adversaries. In *Proceedings of the annual international conference on Advances in Cryptology* (2007).

[31] LINDELL, Y., AND PINKAS, B. Secure two-party computation via cut-and-choose oblivious transfer. In *Proceedings of the conference on Theory of cryptography* (2011).

[32] MALKA, L. Vmcrypt: modular software architecture for scalable secure computation. In *Proceedings of the 18th ACM conference on Computer and communications security* (2011).

[33] MALKHI, D., NISAN, N., PINKAS, B., AND SELLA, Y. Fairplay–a secure two-party computation system. In *Proceedings of the USENIX Security Symposium* (2004).

[34] MIYAJI, A., AND RAHMAN, M. S. Privacy-preserving data mining in presence of covert adversaries. In *Proceedings of the international conference on Advanced data mining and applications: Part I* (2010).

[35] MOHASSEL, P., AND FRANKLIN, M. Efficiency tradeoffs for malicious two-party computation. In *Proceedings of the Public Key Cryptography conference* (2006).

[36] SHELAT, A., AND SHEN, C.-H. Two-output secure computation with malicious adversaries. In *Proceedings of the Annual international conference on Theory and applications of cryptographic techniques* (2011).

[37] SHELAT, A., AND 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] TALBOT, D. Security in the ether. `http://www.technologyreview.com/featuredstory/416804/security-in-the-ether/`, 2009.

[39] YAO, A. C. Protocols for secure computations. In *Proceedings of the Annual Symposium on Foundations of Computer Science* (1982).

A Proof of Security

Following the security definition from Section 3, we prove the following theorem.

Theorem 1. *The Whitewash outsourced two-party SFE protocol securely computes a function $f(a, b)$ in the following three corruption scenarios: (1) The cloud is malicious and non-cooperative with respect to the rest of the parties, while all other parties are semi-honest; (2) All but one party providing input is malicious, while the cloud is semi-honest; or (3) The cloud and mobile device are malicious and colluding, while the evaluator is semi-honest.*

Note that existing outsourcing schemes [21, 6] are only secure in corruption scenarios (1) and (2).

A.1 Malicious application server A^*

Consider when Alice can perform arbitrarily malicious actions while Bob and Cloud follow the protocol in a semi-honest manner. We note that the operations performed by A^* and the messages received by A^* are nearly identical to the malicious evaluator P_2^* from shelat and Shen's proof of their two-party computation scheme [37]. We note here four slight alterations necessary to their simulator S_2, none of which change their proof of security. We call the modified simulator S_A

1. **Input generation:** When S_A generates a random input $\overline{x}'$ for Bob, it also generates a random input $\overline{z}'$ for Cloud. Because this input is chosen from a uniform distribution in both the real and the ideal world, it is statistically indistinguishable.
2. **Input commitments:** When S_A generates the input commitments $\{\Gamma^{(j)}\}_{j\in\sigma}$, it also generates commitments $\{\Xi^{(j)}\}_{j\in\sigma}$ to commit to Cloud's input.
3. **Wire label commitments:** When S_A generates the commitments to its input wires $\{\Theta^{(j)}\}_{j\in\sigma}$, it also generates commitments to Cloud's input wire labels $\{\Psi^{(j)}\}_{j\in\sigma}$.
4. **Output proof:** If A^* successfully proves the correctness of Bob's output, the simulator S_A delivers the random input $\overline{z}'$ to A^*. As stated above, this input is statistically indistinguishable from Cloud's input in the real world.

Given the existence of the simulator S_A, this proves security when the evaluating party A^* is malicious (scenario 2).

A.2 Malicious mobile device B^*

Consider when Bob can perform arbitrarily malicious actions while Alice and Cloud follow the protocol in a semi-honest manner. We construct a simulator S_B in the ideal world to simulate Bob's view of a real execution of the protocol. Note that the simulator does not have the other parties' inputs, nor does it know what input the malicious B^* will use. Thus, Bob's inputs and commitments must be checked, and the output proof and result of computation must be simulated. Consider the following hybrid of experiments.

$\boldsymbol{Hybrid1^{(B)}(k,x;r)}$: This experiment is identical to the experiment $REAL^{(B)}(k,x;r)$ except that the experiment receives the values $\{\rho^{(j)}\}_{j\in\sigma}, \{\gamma^{(j)}\}_{j\in\sigma}$, and $\{\Gamma^{(j)}\}_{j\in\sigma}$ from B^* and uses them to recover B^*s input. If for any $j \in S$, the decommitment $\Gamma^{(j)}$ cannot reveal B^*s input $\overline{x}^{*(j)}$, the simulator aborts.

Lemma 1. $REAL^{(B)}(k,x;r) \overset{c}{\approx} Hybrid1^{(B)}(k,x;r)$

Proof. Because the experiment is in control of Alice and Cloud, for any $j \in \sigma$ we know that the commitment $\Theta^{(j)}$ is constructed correctly using $\rho^{(j)}$. Thus, the only possible way that the experiment will not uncover the value for some $\overline{x}^{*(j)}$ is if $\{\theta^{(j)}_{i,\overline{x}^*_i}\}_{i\in[m_b]}$, when decommitted from $\Gamma^{(j)}$ using $\gamma^{(j)}$ correctly decommits the $i \oplus 1$ half of $\Theta^{(j)}$, which happens with negligible probability based on the binding property of the commitment. Otherwise, at least one of the two commitments $\Gamma^{(j)}$ or $\Theta^{(j)}_i$ must fail to decommit, in which case both experiments abort. □

$\boldsymbol{Hybrid2^{(B)}(k,x;r)}$: This experiment is identical to the experiment $Hybrid1^{(B)}(k,x;r)$ except that if the extracted inputs are inconsistent, the experiment aborts.

Lemma 2. $Hybrid1^{(B)}(k,x;r) \overset{c}{\approx} Hybrid2^{(B)}(k,x;r)$

Proof. This follows from the 2-universal hash check of consistency. Since all of the circuits are generated by the experiment as cloud, they are all constructed correctly. Following from Lemma G.10 in shelat and Shen's proof [37], indistinguishability holds here. □

$\boldsymbol{Hybrid3^{(B)}(k,x;r)}$: This experiment is identical to the experiment $Hybrid2^{(B)}(k,x;r)$ except that the experiment passes x^* to the trusted third party and receives $f_b(x^*,y)$ in return. It then randomly selects an evaluated circuit $G(C)^{(j)}$ and uses the output keys from that circuit to run the output proof of correctness for $f_b(x^*,y) \oplus e^* \oplus p_b$ with B^*.

Lemma 3. $Hybrid2^{(B)}(k, x; r) \overset{c}{\approx} Hybrid3^{(B)}(k, x; r)$

Proof. This follows from the witness-indistinguishability property of the output proof, which guarantees that the index of the circuit output being sent remains hidden. Indistinguishability follows directly from Lemma G.12 in shelat and Shen's proof [37]. □

$\boldsymbol{Hybrid4^{(B)}(k, x; r)}$: This experiment is identical to the experiment $Hybrid3^{(B)}(k, x; r)$ except that the experiment selects random inputs for Alice a' and Cloud z' following the parameters of the protocol.

Lemma 4. $Hybrid3^{(B)}(k, x; r) \overset{c}{\approx} Hybrid4^{(B)}(k, x; r)$

Proof. This follows from the security guarantees of the garbled circuit itself. Since the output of the circuit in the real world matches the output of the trusted third party in the ideal world, B^* learns nothing from the output received. Since the output produced by replacing Alice's input with random inputs is never returned to B^*, he cannot distinguish between Alice's real inputs and the random inputs. Finally, since Cloud's input is two pseudorandom strings in the real protocol, it is statistically indistinguishable from the experiment's choice of z'. Thus, indistinguishability holds even when z' is revealed in the output release phase. □

Lemma 5. $Hybrid4^{(B)}(k, x; r)$ *runs in polynomial time.*

Proof. This follows trivially from the fact that the main protocol runs in polynomial time. Since the experiment does not perform any additional actions beyond the main protocol, it also runs in polynomial time. □

$Hybrid4^{(B)}(k, x; r)$ is identical to the simulator S_B running in the ideal world. The simulator runs B^* and controls Alice and Cloud. If any of the consistency checks fails, S_B terminates the protocol. Otherwise, it delivers B^*s input to the trusted third party in $Hybrid3^{(B)}(k, x; r)$, and outputs whatever B^* outputs. By Lemma 1-5, this simulator proves Theorem 1 when the mobile device is malicious (scenario 2).

A.3 Malicious cloud C^*

Consider when Cloud can perform arbitrary malicious actions while Alice and Bob follow the protocol in a semi-honest manner. We construct a simulator S_C in the ideal world to simulate Cloud's view of a real execution of the protocol. Note that since the simulator does not have the other parties' inputs, nor does it know what input the malicious C^* will use. Thus, the inputs, commitments, and circuits generated by the cloud must be checked, and the oblivious transfers must be simulated. Consider the following hybrid of experiments.

$\boldsymbol{Hybrid1^{(C)}(k, x; r)}$: This experiment is identical to the experiment $REAL^{(C)}(k, x; r)$ except that instead of running the circuit oblivious transfers, the experiment invokes the simulator S_{OT}, which recovers both of C^*s inputs to the oblivious transfer (i.e., the random coins $\{\rho^{(j)}\}_{j\in\sigma}$ and the commitment keys $\{\xi^{(j)}\}_{j\in\sigma}$).

Lemma 6. $REAL^{(C)}(k, x; r) \overset{c}{\approx} Hybrid1^{(C)}(k, x; r)$

Proof. Based on the malicious security of the oblivious transfer primitive, we know that S_{OT} exists. The proof of this lemma follows directly from Lemma G.7 in shelat and Shen's security proof [37]. □

$\boldsymbol{Hybrid2^{(C)}(k, x; r)}$: This experiment is identical to the experiment $Hybrid1^{(C)}(k, x; r)$ except that if more than $\sigma/5$ circuits are incorrectly constructed, then the experiment aborts.

Lemma 7. $Hybrid1^{(C)}(k, x; r) \overset{c}{\approx} Hybrid2^{(C)}(k, x; r)$

Proof. For a circuit to be incorrectly constructed means that the commitments $\{\Theta^{(j)}, \Omega^{(j)}, \Phi^{(j)}, \Psi^{(j)}\}$ and the circuit $G(C)^{(j)}$, for $j \in \sigma$, cannot be reconstructed given the objective circuit C and the randomness $\rho^{(j)}$. Again, this lemma follows directly from Lemma G.8 in shelat and Shen's proof [37]. □

$\boldsymbol{Hybrid3^{(C)}(k, x; r)}$: This experiment is identical to the experiment $Hybrid2^{(C)}(k, x; r)$ except that the experiment will abort if C^*s private inputs cannot be recovered for at least $\sigma/5$ of the evaluation circuits.

Lemma 8. $Hybrid2^{(C)}(k, x; r) \overset{c}{\approx} Hybrid3^{(C)}(k, x; r)$

Proof. From the previous lemma, we know that $4\sigma/5$ of the circuits are correctly constructed. This implies that of the $2\sigma/5$ circuits chosen to be evaluated, at least $\sigma/5$ are "good" circuits. Let these "good" circuits be denoted as G, where $|G| \geq \sigma/5$. Assume for contradiction that there is some $j \in G$ where $\overline{z}^{*(j)}$ cannot be recovered. The only possible way that the experiment will not uncover the value for some $\overline{z}^{*(j)}$ is if $\{\psi^{(j)}_{i,\overline{z}^*_i}\}_{i\in[m_c]}$, when decommitted from $\Xi^{(j)}$ using $\xi^{(j)}$ correctly decommits the $i \oplus 1$ half of $\Psi^{(j)}$, which happens with negligible probability based on the binding property of the commitment. Otherwise, at least one of the two commitments $\Xi^{(j)}$ or $\Psi^{(j)}_i$ must fail to decommit, in which case both experiments abort.

□

$\boldsymbol{Hybrid4^{(C)}(k, x; r)}$: This experiment is identical to the experiment $Hybrid3^{(C)}(k, x; r)$ except that the experiment aborts if any of C^*s inputs to the good circuits in G is inconsistent.

Lemma 9. $Hybrid3^{(C)}(k, x; r) \overset{c}{\approx} Hybrid4^{(C)}(k, x; r)$

Proof. Informally, this proof follows from the 2-universal hash check used in the circuit. This lemma follows directly from Lemma G.10 in shelat and Shen's proof [37]. □

$\boldsymbol{Hybrid5^{(C)}(k, x; r)}$: This experiment is identical to the experiment $Hybrid4^{(C)}(k, x; r)$ except that the experiment chooses a random input for Alice y' and computes $\overline{y}'$ such that $\mathbf{M} \cdot \overline{y}' = y'$ and uses that as input to the input oblivious transfers.

Lemma 10. $Hybrid4^{(C)}(k, x; r) \overset{c}{\approx} Hybrid5^{(C)}(k, x; r)$

Proof. Informally, this proof follows from the choose security of the OT primitive. This lemma follows directly from Lemma G.14 in shelat and Shen's proof [37]. □

$\boldsymbol{Hybrid6^{(C)}(k, x; r)}$: This experiment is identical to the experiment $Hybrid5^{(C)}(k, x; r)$ except that the experiment chooses a random input for Bob x' and computes $\overline{x}'$ by concatenating random strings e' and r' to x' and uses these inputs as input to the computation.

Lemma 11. $Hybrid5^{(C)}(k, x; r) \overset{c}{\approx} Hybrid6^{(C)}(k, x; r)$

Proof. Because C^* never receives Bob's inputs in any form, or output from the computation, he cannot distinguish between using Bob's real inputs and random inputs chosen by the experiment. Since C^* only ever sees Bob's input commitment keys $\{\gamma^{(j)}\}_{j\in\sigma}$, which are pseudorandom strings in both experiments, these strings are statistically indistinguishable as well. □

$\boldsymbol{Hybrid7^{(C)}(k, x; r)}$: This experiment is identical to the experiment $Hybrid6^{(C)}(k, x; r)$ except that the simulator runs the function $f(x', y')$ for the inputs randomly chosen in the previous lemma. If $f_1(x', y') \oplus e^* \oplus p_b$ and $f_2(x', y') \oplus p_a$ do not match a majority of the evaluation outputs, the experiment aborts.

Lemma 12. $Hybrid6^{(C)}(k, x; r) \overset{c}{\approx} Hybrid7^{(C)}(k, x; r)$

Proof. This follows trivially from the fact that at least $\sigma/5$ circuits are correctly constructed and that C^*s inputs to those circuits are consistent. Thus, the majority output will be exactly $f_1(x', y') \oplus e^* \oplus p_b$ and $f_2(x', y') \oplus p_a$. □

$\boldsymbol{Hybrid8^{(C)}(k, x; r)}$: This experiment is identical to the experiment $Hybrid7^{(C)}(k, x; r)$ except that when C^* returns the one-time pads w^* in the output release, the experiment aborts if $w^* \neq z^*$, where z^* is the consistent input to the good circuits in G. Otherwise, the experiment sends z^* to the trusted third party as C^*s input.

Lemma 13. $Hybrid7^{(C)}(k, x; r) \overset{c}{\approx} Hybrid8^{(C)}(k, x; r)$

Proof. This follows from the collision-resistance of the 2-universal hash family. C^* can only return a value w^* such that $\mathbf{H} \cdot w^* = \mathbf{H} \cdot z^*$ with negligible probability, and so the experiments are indistinguishable. □

Lemma 14. $Hybrid8^{(C)}(k, x; r)$ *runs in polynomial time.*

Proof. This follows trivially from the fact that the main protocol runs in polynomial time. Since the experiment only evaluates $f(\cdot, \cdot)$ (which is also polynomial time) in addition to the main protocol, it also runs in polynomial time. □

$Hybrid8^{(C)}(k, x; r)$ is identical to the simulator S_C running in the ideal world. The simulator runs C^* and controls Alice and Bob. If any of the consistency checks fails, S_C terminates the protocol. Otherwise, it delivers C^*s input to the trusted third party when it is completes $Hybrid8^{(B)}(k, x; r)$, and outputs whatever C^* outputs. By Lemma 6-14, this simulator proves Theorem 1 when Cloud is malicious (scenario 1).

A.4 Malicious and colluding mobile device and cloud BC^*

Consider when Bob and Cloud can perform arbitrary malicious actions and share arbitrary information while Alice follows the protocol in a semi-honest manner. We observe that this scenario is equivalent to a malicious generator P_1^* in shelat and Shen's proof of security [37], with some modifications to the lemmas to account for communicating with two parties and to account for Cloud's added input. We also note that in this scenario, the malicious and colluding BC^* may terminate the protocol early, preventing Alice from receiving her output. However, this is possible on the evaluator's side in shelat and Shen's protocol, so we consider fair release a separate guarantee from security. We describe the changes to each hybrid experiment in shelat and Shen's proof below, as well as noting slight changes to the proofs of each lemma.

$\boldsymbol{Hybrid1^{(BC)}(k, x; r)}$: This experiment is identical to the experiment $REAL^{(BC)}(k, x; r)$ except that instead of running the circuit oblivious transfers, the experiment invokes the simulator S_{OT}, which recovers both of C^*s inputs to the oblivious transfer (i.e., the random coins $\{\rho^{(j)}\}_{j\in\sigma}$ and the commitment keys $\{\gamma^{(j)}\}_{j\in\sigma}, \{\xi^{(j)}\}_{j\in\sigma}$).

The proof of this hybrid follows directly from shelat and Shen, only it is extended to recover the commitments to C^*s input as well as B^*s.
$\boldsymbol{Hybrid2^{(BC)}(k, x; r)}$: This experiment is identical to the experiment $Hybrid1^{(BC)}(k, x; r)$ except that if more than $\sigma/5$ circuits are incorrectly constructed, then the experiment aborts.

Again, this follows directly from shelat and Shen. However, the commitments to the Cloud's input wires $\{\Psi^{(j)}\}_{j\in\sigma}$ must also be checked.
$\boldsymbol{Hybrid3^{(BC)}(k, x; r)}$: This experiment is identical to the experiment $Hybrid2^{(BC)}(k, x; r)$ except that the experiment will abort if both B^* and C^*s private inputs cannot be recovered for at least $\sigma/5$ of the evaluation circuits.

Lemma 15. $Hybrid2^{(BC)}(k, x; r) \overset{c}{\approx} Hybrid3^{(BC)}(k, x; r)$

Proof. This lemma holds in the same manner as Lemma 8 when only the Cloud is malicious. Since the commitments $\Theta^{(j)}$ and $\Psi^{(j)}$ are constructed correctly, the only way that the input of either B^* or C^* cannot be recovered is if the decommited values from $\Gamma^{(j)}$ and $\Xi^{(j)}$ decommitted the wrong halves of the commitments $\Theta^{(j)}$ and $\Psi^{(j)}$ respectively. This would imply that B^* or C^* was able to break the binding property of the commitment, which can only happen with negligible probability. □

$\boldsymbol{Hybrid4^{(BC)}(k, x; r)}$: This experiment is identical to the experiment $Hybrid3^{(BC)}(k, x; r)$ except that the experiment aborts if any of B^* or C^*s inputs to the good circuits in G are inconsistent.

Again, this follows directly from shelat and Shen, expanded to handle the inputs of both malicious parties.
$\boldsymbol{Hybrid5^{(BC)}(k, x; r)}$: This experiment is identical to the experiment $Hybrid4^{(BC)}(k, x; r)$ except that the input recovered in the previous hybrid, $\overline{x}^* = x^*||r^*||e^*$ and $\overline{z}^* = p_a^*||p_b^*$ are forwarded to the trusted third party, which returns $f(x^*, y, z^*)$. The experiment aborts if the majority output of the computation does not match $f_b(x^*, y) \oplus e^* \oplus p_b$.

Here we extend shelat and Shen to include the input from Cloud, which is added in as a blind to the output of computation in the real protocol.
$\boldsymbol{Hybrid6^{(BC)}(k, x; r)}$, $\boldsymbol{Hybrid7^{(BC)}(k, x; r)}$,
$\boldsymbol{Hybrid8^{(BC)}(k, x; r)}$: These hybrid experiments are identical to the hybrids in shelat and Shen's proof. So, we invoke them directly and the proofs follow as they are in the two-party case.

Finally, we demonstrate that in the final step of the Whitewash protocol, the output release, that early termination by B^* or C^* is functionally the same as C^* returning an incorrect value for p_a^*. That is, Alice will always detect an early termination, and will always detect an incorrect value of p_a^* except for a negligible probability.

	Timing (Seconds)			Reduction Over	
Circuit	WW	CMTB	KSS	CMTB	KSS
Hamming (1600)	95.57 ± 0.48	453.36 ± 2.32	2614.04 ± 7.08	78.92%	96.34%
Hamming (16384)	941.15 ± 11.07	1,335.75 ± 3.69	x	29.54%	x
Matrix (3x3)	12.04 ± 0.26	493.28 ± 2.90	x	97.56%	x
Matrix (5x5)	35.62 ± 1.01	543.67 ± 4.23	x	93.45%	x
Matrix (8x8)	108.54 ± 3.55	702.13 ± 6.14	x	84.54%	x
Matrix (16x16)	620.34 ± 13.83	2,263.86 ± 108.27	x	72.60%	x
Dijkstra's 10	62.53 ± 2.69	532.55 ± 5.18	x	88.26%	x
Dijkstra's 20	181.55 ± 10.06	752.40 ± 3.58	x	75.87%	x
Dijkstra's 50	881.19 ± 37.97	1,800.47 ± 50.27	x	51.06%	x
RSA-256	30,872.58 ± 1,148.69	x	x	x	x

Table 4: Execution time (in seconds) for all tested circuits. Results with an 'x' indicate that a protocol was not able to evaluate that circuit.

Lemma 16. *The probability of catching a malformed p_a^* is computationally indistinguishable from catching early termination.*

Proof. Since the output $f_a(x^*, y) \oplus p_a^*$ was generated by a good circuit and C^*s input to that circuit was consistent, then the output of the hash $\mathbf{H} \cdot p_a^*$ is correctly computed. Thus, let $\overline{p}_a$ be the value that C^* returns during the output release. By the guarantees of a 2-universal hash, the probability that $\mathbf{H} \cdot p_a^* = \mathbf{H} \cdot \overline{p}_a$ is negligible. □

Given these changes to the simulator S_1 in shelat and Shen's proof, the modified simulator S_{BC} proves Theorem 1 (without the fair release guarantee) when the mobile device and Cloud are malicious and colluding (scenario 3).

B Experiment Results

In Table 4 we provide full experimental timings for all three of the evaluated test circuits with 95% confidence intervals. For all Hamming Distance and Matrix Multiplication circuits, the execution times are averaged over 10 trials. For RSA-256, we ran 3 executions.