2012 IEEE Symposium on Security and Privacy

Quid-Pro-Quo-tocols: Strengthening Semi-Honest Protocols with Dual Execution

Yan Huang
University of Virginia
yhuang@virginia.edu

Jonathan Katz
University of Maryland
jkatz@cs.umd.edu

David Evans
University of Virginia
evans@virginia.edu

***Abstract*—Known protocols for secure two-party computation that are designed to provide full security against malicious behavior are significantly less efficient than protocols intended only to thwart semi-honest adversaries. We present a concrete design and implementation of protocols achieving security guarantees that are much stronger than are possible with semi-honest protocols, at minimal extra cost. Specifically, we consider protocols in which a malicious adversary may learn a single (arbitrary) bit of additional information about the honest party's input. Correctness of the honest party's output is still guaranteed. Adapting prior work of Mohassel and Franklin, the basic idea in our protocols is to conduct two separate runs of a (specific) semi-honest, garbled-circuit protocol, with the parties swapping roles, followed by an inexpensive secure equality test. We provide a rigorous definition and prove that this protocol leaks no more than one additional bit against a malicious adversary. In addition, we propose some heuristic enhancements to reduce the overall information a cheating adversary learns. Our experiments show that protocols meeting this security level can be implemented at cost very close to that of protocols that only achieve semi-honest security. Our results indicate that this model enables the large-scale, practical applications possible within the semi-honest security model, while providing stronger security guarantees.**

***Keywords*-secure two-party computation, privacy-preserving protocols.**

I. Introduction

Protocols for secure two-party computation allow two mutually distrusting parties to compute a function that depends on both their inputs while ensuring correctness, privacy, and more, without relying on a trusted third party. Recent results [29, 28, 34, 14, 16] have shown that it is feasible to implement *generic* protocols for secure two-party computation (where by "generic" we mean protocols that can compute arbitrary functions specified as a boolean or arithmetic circuit) based on Yao's *garbled-circuit* approach [36], in some cases quite efficiently [17, 15]. To obtain reasonable performance, however, many of these works [29, 14, 16] and others that rely on them [17, 15] assume the *semi-honest* (or *honest-but-curious*) model in which the adversary is assumed to always follow the protocol but may try to learn information from the protocol transcript beyond what is allowed.

Several generic approaches are known for achieving security against *malicious* adversaries (i.e., adversaries who may arbitrarily deviate from the protocol specification). Some approaches rely on zero-knowledge proofs of correct behavior [11, 22]. Others rely on "cut-and-choose" techniques to detect dishonesty [25, 33, 35, 27, 24]. Another recent approach uses message-authentication techniques to ensure that parties use correct values throughout the computation [32]. Protocols produced by any of these approaches exhibit a slowdown of several orders of magnitude compared to protocols with semi-honest security. The slowdown is not only due to increased computation and communication; a (potentially) more significant issue is the memory usage required by some of the known protocols. As an example, the Lindell-Pinkas protocol [25] requires hundreds of copies of the garbled circuits to be transmitted before verification and evaluation, and so does not appear to be compatible with pipelined circuit execution (a technique that makes secure computation memory efficient [16]), at least not without introducing additional overhead. As one data point supporting this claim, a recent implementation of secure two-party computation of AES [35] (with security against malicious adversaries) required one of the parties to use about 190 MB of storage. This suggests that the resources required in order to achieve (full) security against malicious adversaries can be prohibitive, even for many small-scale computations.

Mohassel and Franklin [30] proposed a relaxed definition of security in which, informally, a malicious adversary may be able to learn a small number of additional bits of information about the honest party's input, beyond what is implied by the output (a formal definition is in Section V). This definition may suffice for many realistic scenarios in which secure computation would be used. Note that even fully secure protocols leak information about the honest party's input in the function output. Depending on the specific function and the control the adversary has over its own input this information leakage may be substantial and hard to characterize.

In addition to proposing the new security notion discussed above, Mohassel and Franklin also present a general approach to realizing that notion. At a high level, the idea is to run two independent executions of a (specific) semi-honest protocol, with the parties swapping roles in the two executions. This is followed by an equality test on particular values held by the two parties. See Section III for details.

Additional related work is discussed in Section II-C.

A. Contributions

In this work we flesh out and implement an optimized version of the Mohassel-Franklin protocol. We describe a specific, highly efficient mechanism for carrying out the equality test (Section III-B) in their approach, and implement

DOI 10.1109/SP.2012.43

their protocol, incorporating pipelined execution and other efficiency optimizations (Section VI). We provide a precise definition of their security notion (specialized for the case of 1-bit leakage), formally specify the details of the Mohassel-Franklin protocol, and give a rigorous proof that their "dual-execution" approach satisfies the given definition (Section V). Our experimental results (Section VI-C) show that it is possible to obtain performance competitive with state-of-the-art semi-honest protocols, while achieving meaningful (though not complete) protection against malicious behavior.

In Section VII we present two heuristic strategies for further limiting what an adversary can learn. In our first approach, the actual function output is revealed only *after* the equality test is completed successfully. This means the adversary may learn an unallowed bit of information, but only at the expense of failing to learn the actual output some of the time. The second approach ensures that the attacker learns at most one more bit of the output than the honest party. Both these enhancements are inexpensive, and their complexity depends only on the length of the output.

II. Background

The main cryptographic tools we use are garbled circuits and oblivious transfer, which we briefly introduce next. Section II-C summarizes previous work towards secure computation against stronger classes of adversaries.

A. Garbled Circuits

Garbled circuits [36] allow two semi-honest parties to compute an arbitrary function $f(x_0, x_1)$ that depends on their respective private inputs, x_0 and x_1, without leaking any information about their inputs beyond what is revealed by the function output itself. One party, acting as the circuit *generator*, produces a garbled circuit that is evaluated by the other party, known as the circuit *evaluator*. The result is an "encrypted" output, which can then be mapped to its actual value and revealed to either or both parties. We provide an overview here; for technical details and a proof of security, see Lindell and Pinkas [26].

The basic idea is to transform a boolean circuit into a garbled circuit that operates on labels (i.e., cryptographic keys) instead of bits. Any binary gate, g, which has two input wires W_0, W_1 and output wire W_2, can be converted into a garbled gate. First, generate random labels w_i^0 and w_i^1 to represent $\mathsf{0}$ and $\mathsf{1}$ on each wire W_i. Then, generate a truth table containing the four entries

$$\mathsf{Enc}_{w_0^{s_0}, w_1^{s_1}}(w_2^{g(s_0,s_1)})$$

for each $s_0, s_1 \in \{0, 1\}$ (where s_0, s_1 denote the 1-bit signals on wires W_0, W_1, respectively), and randomly permute the table. This truth table is called a *garbled gate*. Observe that given the garbled gate and labels $w_0^{s_0}$ and $w_1^{s_1}$, it is possible to recover $w_2^{g(s_0,s_1)}$. Thus, given the labels that correspond to some set of input values for the entire circuit, it is possible for the circuit evaluator to recover labels corresponding to the output of the circuit on those inputs. If the circuit generator provides a way to map those labels back to bits, the circuit evaluator can recover the actual output.

The only thing that remains is to provide a mechanism that allows the circuit evaluator to obtain input-wire labels for the bits corresponding to the inputs of the two parties. The circuit generator can simply send the appropriate labels for its own input to the circuit evaluator since these labels reveal nothing about the circuit generator's input but are merely randomly chosen labels. The circuit evaluator obtains the input-wire labels for its own input using *oblivious transfer*, described in Section II-B.

In summary, a garbled-circuit protocol involves parties agreeing to a circuit that computes the desired function, and then following these steps: (1) the circuit generator garbles each gate in the circuit; (2) the circuit generator sends the garbled gates, along with the wire labels corresponding to its own inputs; (3) the circuit evaluator obtains the wire labels corresponding to its inputs from the generator using an oblivious transfer protocol; (4) the circuit evaluator evaluates the circuit by successively decrypting entries of each garbled gate until reaching the output wires; and (5) the generator provides a way to map output-wire labels to bits, thus allowing the evaluator to compute the actual output.

The main bottleneck in garbled-circuit protocols is generating and evaluating each gate, since this requires four encryptions for the circuit generator and (as described) four decryptions for the circuit evaluator. Many techniques have been developed to reduce the costs of executing garbled circuits, including the *point-and-permute* technique that allows the circuit evaluator to decrypt only a single entry (rather than all four) in a garbled gate [29]; the *free-XOR* technique [23] that allows XOR gates to be executed without any encryption operations; *Garbled Row Reduction* (GRR) [34] that reduces the size of a garbled gate to three ciphertexts (thus saving 25% of network bandwidth); and *pipelined execution* that parallelizes circuit generation and evaluation [16]. Our implementation uses all of these techniques.

B. Oblivious Transfer

An oblivious-transfer (OT) protocol allows a *sender* to send one of a possible set of values to a *receiver*. The receiver selects and learns only one of the values, and the sender cannot learn which value the receiver selected. In particular, a 1-out-of-2 OT protocol [8] allows the sender, who has two strings b_0 and b_1, to transfer b_σ to the receiver, who holds a private bit σ. *Oblivious-transfer extension* [19] allows the realization of an unbounded number of oblivious transfers with minimal marginal cost per OT, starting from a small number of (expensive) "base" OTs.

In our implementation we use the OT-extension protocol of Ishai et al. [19] with security against malicious adversaries, that uses $O(k^2)$ "base" OTs for k a statistical security parameter. More efficient approaches, requiring only $O(k)$ base OTs, are known [12, 32], but are not used in our implementation. For the "base" OTs we use the Naor-Pinkas protocol [31], whose security is based on the computational

Diffie-Hellman assumption in the random-oracle model. This protocol achieves privacy against malicious adversaries, but *not* full (simulation-based) security against malicious adversaries (as required by our proof of security in Section V). Nevertheless, because we use oblivious-transfer extension, OT is a small fraction of the overall execution time, and changing the OT protocol used would not substantially impact our experimental results. The Naor-Pinkas protocol could easily be adapted to provide (simulation-based) security against malicious adversaries in the random-oracle model, with relatively little additional cost.

C. Threat Models

Most previous work in secure computation has assumed either a *semi-honest* or *malicious* threat model [10]. In the semi-honest (also known as *honest-but-curious*) model, the adversary is assumed to follow the protocol as specified, but may attempt to learn extra information from the protocol transcript. In contrast, a *malicious* adversary may arbitrarily deviate from the specified protocol as it attempts to compromise the privacy of the other party's inputs or correctness of the obtained result.

Most implementations of generic secure two-party computation have targeted the semi-honest threat model [29, 14, 16], and have used protocols based on Yao's garbled-circuit approach. The scalability and efficiency of garbled-circuit protocols have been improved by a series of optimizations including point-and-permute [29], free-XOR [23], garbled-row reduction [34], pipelining [16], and library-based modular circuit construction [16].

Several approaches have been proposed for achieving security against malicious adversaries [11, 22, 25, 33, 35, 27, 24, 32], some of which have been implemented [28, 34, 35, 32]. However, even the best known protocols are orders-of-magnitude slower than the best semi-honest protocols.

Aumann and Lindell et al. introduced the *covert* threat model [1]. In this model, a cheating adversary is "caught" with some constant probability, but with the remaining probability can (potentially) learn the honest party's entire input and arbitrarily bias the honest party's output. If an adversary is unwilling to take the risk of being caught, then such protocols will deter cheating altogether. Aumann and Lindell also show a two-party protocol with covert security that is only a small constant factor less efficient than the basic (semi-honest) garbled-circuit protocol.

The single-bit leakage model we consider here is incomparable to the covert model. On the one hand, the single-bit leakage model allows the adversary to *always* learn one additional bit about the honest user's input, without any risk of being caught. (See Sections IV-A and VII, however, for some discussion about mitigating what the adversary learns.) On the other hand, the covert model allows the adversary to learn the entire input of the honest party with constant probability. The covert model also allows the adversary to affect the correctness of the honest party's output (with constant probability), something prevented in the single-bit leakage model.

III. Dual-Execution Protocols

Informally, a secure-computation protocol needs to satisfy two properties: *privacy*, which ensures private inputs are not revealed improperly, and *correctness*, which guarantees the integrity of the final output. Yao's (semi-honest) garbled-circuit protocol is easily seen to provide *one-sided privacy* against a malicious circuit generator as long as an OT protocol secure against malicious adversaries is used, and the evaluator does not reveal the final output to the circuit generator. It thus only remains to provide correctness guarantees for an honest circuit evaluator against a possibly malicious generator, and to provide a way for both parties to receive the output while continuing to provide privacy guarantees against a malicious generator.

The *dual-execution* (DualEx) protocol proposed by Mohassel and Franklin [30] provides a mechanism to achieve these guarantees. The protocol involves two independent executions of the semi-honest garbled-circuit protocol, where each participant plays the role of circuit generator in one of the executions. The outputs obtained from the two executions are then compared to verify they are identical; if so, each party simply outputs the value it received. Intuitively, this may leak an extra bit of information to an adversary who runs the comparison protocol using an incorrect input.

We describe the protocol and method for comparing outputs in more detail next. Note that Mohassel and Franklin left some details of the protocol unspecified, and did not give a proof of security. They also did not provide any implementation of their approach.

A. Notation

We write a set of wire-label pairs as a matrix:

$$\mathbb{W} = \begin{pmatrix} w_1^0 & w_2^0 & \cdots & w_\ell^0 \\ w_1^1 & w_2^1 & \cdots & w_\ell^1 \end{pmatrix}.$$

A vector of wire labels is denoted as

$$\boldsymbol{w} = (w_1, w_2, \ldots, w_\ell).$$

If $v \in \{0,1\}^\ell$ is a string and $\mathbb{W}$ is a matrix as above, then we let

$$\mathbb{W}^v = (w_1^{v_1}, \ldots, w_\ell^{v_\ell})$$

be the corresponding vector of wire labels.

B. Protocol

Assume the parties wish to compute some function f, and (for simplicity) that each party holds an n-bit input and that f produces an ℓ-bit output.

Figure 1 depicts an overview of the basic DualEx protocol. This is essentially the protocol described in Section 4.1 of Mohassel and Franklin's paper [30]. The protocol consists of two separate runs of a particular semi-honest protocol plus a final stage for verifying that certain values computed during the course of the two semi-honest executions are identical.

A more detailed description of the DualEx protocol is shown in Figure 2. The protocol is conceptually divided into three stages: the first run, the second run, and the

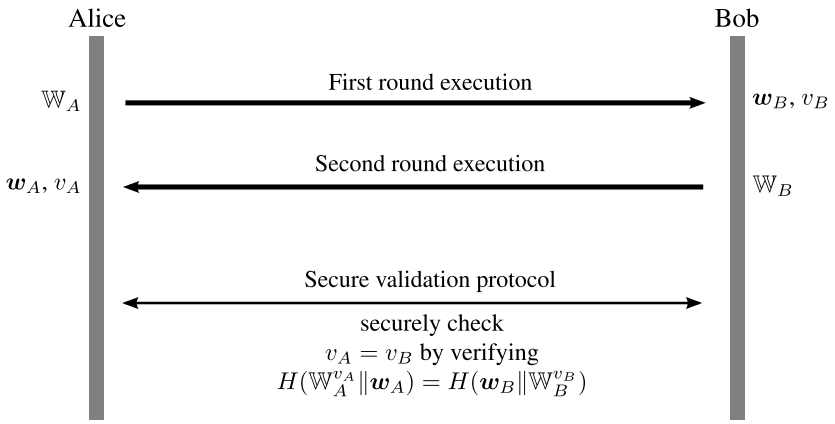

Figure 1. DualEx protocol overview (informal).

secure validation. For the sake of performance, however, our implementation executes the first two stages concurrently, using pipelining to overlap the circuit-generation and circuit-evaluation work for each party (see Section VI). (As long as the oblivious transfers are done sequentially, our security proof is unaffected by performing the two garbled-circuit executions in parallel. The reason is that our security proof holds even against a "worst-case" adversary who waits to receive the entire garbled circuit from the honest party before sending any of its own garbled gates.) We stress that the parties run each of the first two stages to completion — even if an error is encountered — so that no information is leaked about the presence or absence of errors in an execution. If an error is detected that prevents normal progress, the execution continues using random values.

The DualEx protocol uses a specific garbled-circuit protocol with an oblivious-transfer sub-protocol secure against malicious adversaries (see Figure 3). After an execution of this protocol, only P_2 learns the output (but is uncertain about its correctness), while P_1 learns nothing. In this version, the result $f(x, y)$ is revealed to P_2 (Bob in the first execution as defined in Figure 2) even if cheating by P_2 is detected during the equality-checking protocol. This does not violate our definition of security, however for many scenarios this property would be undesirable. Section VII presents some heuristic enhancements to the basic protocol that address this issue by limiting the amount of information either party can obtain during the protocol execution.

C. Secure Output Validation

The goal of the secure validation protocol is to verify the correctness of the outputs Alice and Bob obtained in the previous stages. The validation protocol consists of an equality test between certain *output-wire labels* held by each of the parties. Since half the output-wire labels chosen by the garbled-circuit generator are never learned by the circuit evaluator (and since output-wire labels are chosen at random) this has the effect of preventing an adversary from (usefully) changing their input to the equality test. In an honest execution, on the other hand, since both executions of the garbled-circuit sub-protocol are computing the same function on the same inputs, the inputs to the equality test will be equal.

The equality test will be done by first (*a*) computing a hash of the inputs at both sides, and then (*b*) comparing the hashes using an equality test that is secure against malicious adversaries. (See Figure 4.) If the hash used in the first step is modeled as a random oracle (with sufficiently large output length), then it is possible to show that this results in an equality test for the original inputs with security against malicious adversaries.

Input to Alice**:** the private input x.
Input to Bob**:** the private input y.

Output to both Alice **and** Bob**:** $f(x, y)$, or $\perp$ if cheating is detected.

Execution:

1) Alice and Bob run the semi-honest garbled-circuit protocol (Figure 3) where Alice plays the role of circuit generator (P_1), and Bob plays the role of circuit evaluator (P_2). Alice knows the 2ℓ output-wire labels she generated, $\mathbb{W}_A$, while Bob learns ℓ output-wire labels $\boldsymbol{w}_B$ and an output $v_B \in \{0,1\}^\ell$. (If both parties are honest, $\boldsymbol{w}_B = \mathbb{W}_A^{v_B}$.)
2) Alice and Bob invoke the semi-honest garbled circuit protocol again, swapping roles. Alice learns the output v_A along with labels $\boldsymbol{w}_A$, while Bob knows the label pairs $\mathbb{W}_B$. (If both parties are honest, then $\boldsymbol{w}_A = \mathbb{W}_B^{v_A}$, and also $v_A = v_B$.)
3) Alice and Bob run a "validation protocol" (i.e., an equality test), secure against malicious adversaries. (See Figures 4 and 5 for one possible instantiation.) Alice uses input $\mathbb{W}_A^{v_A} \| \boldsymbol{w}_A$ and Bob uses input $\boldsymbol{w}_B \| \mathbb{W}_B^{v_B}$. If the protocol outputs true, then Alice outputs v_A and Bob outputs v_B. Otherwise, the honest party has detected malicious behavior and outputs $\perp$.

Figure 2. DualEx protocol

Simply exchanging the hashes and doing local comparison is problematic, because this may reveal information on the outputs of the circuit evaluation which is supposed to be hidden from the generator unless the validation check passes. For example, already knowing all the output-wire label pairs, the generator who learns the evaluator's hash can test for candidate output values. Therefore, it is of vital importance to keep the hashes secret throughout the comparison protocol if the equality test fails.

The most straightforward realization of the equality test is to use a generic garbled-circuit protocol (with malicious security). The inputs to the circuit are the hashes $h_1 = H(\mathbb{W}_A^{v_A} \| \boldsymbol{w}_A)$ and $h_2 = H(\boldsymbol{w}_B \| \mathbb{W}_B^{v_B})$, while the circuit is simply a bunch of bitwise XORs (to compute $h_1 \oplus h_2$) followed by a many-to-1 OR circuit that tests if all bits of $h_1 \oplus h_2$ are zero. This still requires running a full garbled-circuit protocol with malicious security, however, which can be expensive.

An alternative is to view an equality test as computing the intersection of two singleton sets. Private set intersection has been widely studied in many contexts, including in the presence of malicious adversaries [9, 13, 5, 20, 21, 7]. We derive our secure equality-test protocol (Figure 5) by specializing the ideas of Freedman et al. [9] based on additively homomorphic encryption. The basic protocol enables P_2 to prove to P_1 that he holds an h_B that is equal to h_A in a privacy-preserving fashion. This basic protocol will be invoked twice with the parties swapping roles. We remark that the protocol that results does not appear to achieve the standard notion of (simulation-based) security against malicious adversaries. Nevertheless, under the assumption that h_1, h_2 are independent, random values (of sufficient length), it does, informally, satisfy the following properties even against a malicious adversary: (1) no information about the honest party's input is leaked to the malicious party, and (2) the malicious party can cause the honest party to output 1 with only negligible probability. We conjecture that our proof of security in Section V can be adapted for equality-testing protocols having these properties.

First of all, P_1 sends to P_2 $\alpha_0 = [\![-h_1]\!]$. Then, P_2 computes $e = [\![r \times (h_2 - h_1) + s]\!]$, using the homomorphic properties of the encryption scheme, as follows

$$r * \big((h_2 * \alpha_1) \ + \ \alpha_0\big) \ + \ s * \alpha_1$$

where $*$ and $+$ here denote homomorphic addition and constant multiplication, respectively. In addition, P_2 sends $h = H(s, h_2)$. P_1 decrypts the result to recover $\hat{s} = r \times (h_2 - h_1) + s$, which is equal to s in case $h_2 = h_1$ but a random value otherwise. Finally, P_1 checks whether $H(\hat{s}, h_1) = h$.

In contrast to the "malicious-client" protocol by Freedman et al. [9], it is unnecessary for P_1 to verify that P_2 followed the protocol (even though it could). The reason is a consequence of several facts:

(*a*) P_2 doesn't gain anything from seeing $(\alpha_0, \alpha_1, s_1)$;
(*b*) it is of P_2's own interest to convince P_1 that $h_1 = h_2$;
(*c*) the test only passes with negligible probability if P_2 cheats.

This three-round protocol satisfies the properties claimed earlier even when both Alice and Bob are malicious. The informal arguments are as follows. To see that Alice's privacy is guaranteed, note that $[\![-h_A]\!]$ hides $-h_A$ thanks to the semantic security offered by the homomorphic encryption scheme. Bob's privacy is also guaranteed by the semantic security of both the homomorphic encryption scheme and the cryptographic hash function (in the random-oracle model).

Input from P_1**:** private input x.
Input from P_2**:** private input y.

Output to P_1**:** the output wire key pairs
$$\mathbb{W}_A = \begin{pmatrix} w_{A1}^0 & w_{A2}^0 & \cdots & w_{A\ell}^0 \\ w_{A1}^1 & w_{A2}^1 & \cdots & w_{A\ell}^1 \end{pmatrix}.$$

Output to P_2**:** $v_B \in \{0,1\}^\ell$ representing the value of $f(x, y)$, and output-wire labels $\boldsymbol{w}_B = (w_{A1}^{v_{B1}}, w_{A2}^{v_{B2}}, \cdots, w_{A\ell}^{v_{B\ell}})$.

Execution:

1) P_1 and P_2 run a garbled-circuit protocol where P_1 plays the circuit *generator*'s role and P_2 the circuit *evaluator*'s role.
2) P_1 and P_2 execute a malicious OT protocol (with P_1 the *sender* and P_2 the receiver) to enable P_2 to learn the wire labels corresponding to P_2's input y. Then P_2 evaluates the garbled circuit to learn output-wire labels $\boldsymbol{w}_B$.
3) P_1 computes
$$\begin{pmatrix} H(w_{A1}^0) \cdots H(w_{A\ell}^0) \\ H(w_{A1}^1) \cdots H(w_{A\ell}^1) \end{pmatrix}$$
for H a random oracle, and sends it to P_2 so that it can use $\boldsymbol{w}_A$ to learn v_B.
4) If P_2 detects cheating at any point during the protocol, he does not complain but instead just outputs completely random v_B and $\boldsymbol{w}_B$.

Figure 3. Semi-honest garbled-circuit sub-protocol

Input to Alice**:** $\mathbb{W}_A^{v_A}, \boldsymbol{w}_A$.
Input to Bob**:** $\boldsymbol{w}_B, \mathbb{W}_B^{v_B}$.

Output to both Alice **and** Bob**:**
$$\begin{cases} \mathsf{true}, & \text{if } \mathbb{W}_A^{v_A} = \boldsymbol{w}_B \text{ and } \boldsymbol{w}_A = \mathbb{W}_B^{v_B}; \\ \mathsf{false}, & \text{otherwise.} \end{cases}$$

Execution:

1) Alice computes $h_1 = H(\mathbb{W}_A^{v_A} \| \boldsymbol{w}_A)$;
2) Bob computes $h_2 = H(\boldsymbol{w}_B \| \mathbb{W}_B^{v_B})$;
3) Alice and Bob uses an equality test (secure against malicious adversaries) to compare h_1 and h_2. If they are equal, Alice and Bob both output true; otherwise they output false.

Figure 4. An instantiation of the secure-validation protocol.

IV. Security Analysis

This section clarifies what it means to leak a single bit on average, and informally discusses possible attack strategies. In Section V, we provide a proof that the defined protocol satisfies the desired property of not leaking more than one bit on average beyond what can already be inferred from the outputs.

A. Leakage and Detection Probability

As argued at the beginning of Section III, there is no privacy loss for the evaluator when a semi-honest garbled circuit protocol is executed if the result is not disclosed to the circuit generator. This works in both directions for our DualEx protocol, so the only concern is how much information is leaked by the equality test. The output of the equality test is just a single bit.

In an information theoretic sense, the maximum average leakage is achieved when the adversary can partition the victim's possible private inputs into two subsets which are equally likely. This assumes the attacker know the a priori distribution of the victim's private inputs, and can design the malicious circuit to produce incorrect results on an arbitrary subset of inputs. Since the other party learns when the equality test fails, a malicious adversary can achieve the maximum expected information gain by designing a circuit that behaves correctly on the victim's private inputs half the time, and incorrectly (that is, it produces an output that will fail the equality test) on the other inputs.

This extra information does not come free to the adversary, however. If the equality test fails, the victim learns that the other party misbehaved. So, for the maximum average leakage circuit which divides the private inputs into two equally likely subsets the attacker learns one extra bit on every protocol execution but has a $1/2$ probably of getting caught.

An attacker who does not want to get caught, can (possibly) design a circuit that divides the other parties private inputs into two subsets where the first subset for which the equality test passes contains $0 < e \leq 1/2$ of the input distribution. (It makes no sense for $e > 1/2$ since then the equality test fails on more than half of the inputs, and the attacker is more likely to get caught than if the subsets are swapped.) In the extreme, an attacker who only cares about one particular private input, x^* could create a circuit that behaves correctly on all inputs except for x^*. The attacker would have no chance of getting caught except when the target input is matched, but would gain very little information otherwise. This suggests, unsurprisingly, that it is unwise to execute a secure computation many times since each execution leaks some information. This is true even for a maliciously secure protocol since the output itself leaks information, and, even with a maliciously secure protocol, an attacker can alter its own inputs to probe the victim's private inputs.

Input to P_1: h_1; decryption key sk_{P_1}.
Input to P_2: h_2.
Public inputs: Public key pk_{P_1}.

Output to P_1: true if $h_1 = h_2$; false otherwise.
Output to P_2: $\perp$.

Execution:

1) P_1 sends to P_2 $\alpha_0 = [\![-h_1]\!]$.
2) P_2 picks random r, s, computes $(e, h) = ([\![r \times (h_2 - h_1) + s]\!], H_2(s, h_2))$, and sends (e, h) to P_1.
3) P_1 computes $\hat{s} = \mathsf{Dec}(e)$. If $H(\hat{s}, h_1) = h$, P_1 outputs true; otherwise, it outputs false.

Figure 5. A one-sided equality-testing protocol. For a discussion of the security guarantees provided, see the text.

The implications of the single-bit leakage depend on the application, and in some scenarios leaking even a single bit may be unacceptable. On the other hand, for many applications this is much less information that an adversary can infer from the outputs, even if a maliciously secure protocol is used. Further, our analysis assumes the worst case where the adversary may implement an arbitrary partitioning function. It may be possible to take advantage of constraints in the circuit design to limit the possible partitioning functions that can be implemented, and to combine this with delayed revelation protocols (see Section VII) to further limit the actual information an adversary can obtain, although we have no yet found a principled way to provide meaningful constraints on the possible partitioning functions.

B. Attacker Strategies

There are several possible strategies a malicious attacker may use against a DualEx protocol. The attacks may be grouped into three main types[1]: *selective failure*, in which the attacker constructs a circuit that fails along some execution paths and attempts to learn about the other party's private inputs from the occurrence of failure, *false function*, in which the attacker constructs a circuit that implements function that is different from the agreed upon function, and *inconsistent inputs*, in which the attacker provides different inputs to the two executions. Note that the formal security proof presented in Section V is *independent* of any specific attack strategy.

Selective failure. In a *selective failure* attack, a malicious party engages in the protocol in a way that causes it to fail for a subset of the other party's possible inputs. This could be done by either putting bad entries into a garbled truth table, or by providing bad input wire labels in the OT for some values. If the protocol succeeds, in addition to knowing the secure computation outcome, the attacker also eliminates some possibilities of the peer's private inputs. If the protocol fails, the attacker still learns something about the peer's inputs, but the misbehavior will be detected by the peer.

One characteristic of our garbled circuit implementation is that the circuit evaluator can always proceed to the normal

[1]We do not consider side-channel attacks, which are possible if the protocol implementation is not done carefully, but only focus on protocol-level attacks here.

termination even when some intermediate wire labels are broken (possibly due to the selective failure attack) since the evaluator never checks the validity of those wire labels except for the final wires. This is desirable because it conceals the positions where the fault occurs, constraining the amount of leakage to merely a single bit on average.

False function. A malicious participant can generate a circuit that computes some function, g, that is different from the function f which the other party agreed to compute. The malicious circuit function g has the same output format as f, enabling the attacker to learn if $g(x,y) = f(x,y)$, in addition to knowing the desired output $f(x,y)$. However, the adversary has to risk being caught if $g(x,y) \neq f(x,y)$.

This attack is intrinsic to the design of dual execution protocols. However, the space of g can be limited by the structure of the circuit, which is already known to the evlauator. Although the other party cannot determine if the individual garbled tables perform the correct logical function, it can verify that (*a*) the circuit uses a predefined number of gates, (*b*) the gates are interconnected as presumed, (*c*) all XOR gates show up as presumed (since these are implemented using the free-XOR optimization), and (*d*) all non-free gates are positioned correctly. Limiting the set of functions g that could be implemented by the adversary, depends on understanding the possible functions that can be computed with a given circuit structure by changing the binary operations of non-free gates in the circuit. Analyzing a given circuit structure for more precise quantification of the leakage can be an interesting future work.

Inconsistent inputs. The adversary can also provide different inputs to the two protocol executions so that the equality test reveals if $f(x,y) = f(x',y)$ (where $x \neq x$ and are selected by the adversary) in the secure validation stage. For example, for any input wire corresponding to Alice's private input, Alice as a circuit generator could send to Bob a label representing 1, whereas as a circuit evaluator uses 0 as her input to the OT protocol to obtain a wire label representing 0 for evaluation.

These attacks appear to give the malicious adversary a great deal of power. However, as we prove in the next section, regardless of the adversary's strategy, the essential property of dual execution protocols is that the leakage is limited to the single bit leaked by the equality test.

V. Proof of Security

We give a rigorous proof of security for the DualEx protocol following the classic paradigm of comparing the real-world execution of the protocol to an ideal-world execution where a trusted third party evaluates the function on behalf of the parties [10]. The key difference is that here we consider a non-standard ideal world where the adversary is allowed to learn an additional bit of information about the honest party's input.

We remark that, for reasons described throughout the text, the proof here does not apply to our implementation per se, but instead refers to the DualEx protocol from in Figure 2, instantiated using the garbled-circuit protocol from Figure 3, where the equality test in Figure 2 and the oblivious transfers in Figure 3 are done using protocols that achieve the standard (simulation-based) notion of security against malicious adversaries, and the oblivious-transfer subprotocols are run sequentially.

A. Definitions

Preliminaries. We use n to denote the security parameter. A function $\mu(\cdot)$ is *negligible* if for every positive polynomial $p(\cdot)$ and all sufficiently large n it holds that $\mu(n) < 1/p(n)$.

A *distribution ensemble* $X = \{X(a,n)\}_{a\in\mathcal{D}_n,\, n\in\mathbb{N}}$ is an infinite sequence of random variables indexed by $a \in \mathcal{D}_n$ and $n \in \mathbb{N}$, where $\mathcal{D}_n$ may depend on n.

Distribution ensembles $X = \{X(a,n)\}_{a\in\mathcal{D}_n,\, n\in\mathbb{N}}$ and $Y = \{Y(a,n)\}_{a\in\mathcal{D}_n,\, n\in\mathbb{N}}$ are *computationally indistinguishable*, denoted $X \stackrel{c}{\equiv} Y$, if for every non-uniform polynomial-time algorithm D there exists a negligible function $\mu(\cdot)$ such that for every n and every $a \in \mathcal{D}_n$

$$\Big|\Pr[D(X(a,n)) = 1] - \Pr[D(Y(a,n)) = 1]\Big| \leq \mu(n).$$

We consider secure computation of *single-output, deterministic* functions where the two parties wish to compute some (deterministic) function f with Alice providing input x, Bob providing input y, and both parties learning the result $f(x,y)$. We assume f maps two n-bit inputs to an ℓ-bit output.

A two-party protocol for computing a function f is a protocol running in polynomial time and satisfying the following correctness requirement: if Alice begins by holding 1^n and input x, Bob holds 1^n and input y, and the parties run the protocol honestly, then with all but negligible probability each party outputs $f(x,y)$.

Security of protocols. We consider static corruptions by *malicious* adversaries, who may deviate from the protocol in an arbitrary manner. We define security via the standard real/ideal paradigm, with the difference being that we use a weaker version of the usual ideal world. Specifically, in the standard formulation of the ideal world there is a trusted entity who receives inputs x and y from the two parties, respectively, and returns $f(x,y)$ to both parties. (We ignore for now the issue of fairness.) In contrast, here we consider an ideal world where a malicious party sends its input along with an *arbitrary* boolean function g, and learns $g(x,y)$ in addition to $f(x,y)$. (The honest party still learns only $f(x,y)$.) Note that in this weaker ideal model, *correctness* and *input independence* still hold: that is, the honest party's output still corresponds to $f(x,y)$ for some legitimate inputs x and y, and the adversary's input is independent of the honest party's input. *Privacy* of the honest party's input also holds, modulo a single additional bit that the adversary is allowed to learn.

Execution in the real model. We first consider the real model in which a two-party protocol Π is executed by Alice

and Bob (and there is no trusted party). In this case, the adversary $\mathcal{A}$ gets the inputs of the corrupted party and arbitrary auxiliary input aux and then starts running the protocol, sending all messages on behalf of the corrupted party using an arbitrary polynomial-time strategy. The honest party follows the instructions of Π.

Let Π be a two-party protocol computing f. Let $\mathcal{A}$ be a non-uniform probabilistic polynomial-time machine with auxiliary input aux. We let $\text{VIEW}_{\Pi,\mathcal{A}(\mathsf{aux})}(x,y,n)$ be the random variable denoting the entire view of the adversary following an execution of Π, where Alice holds input x and 1^n, and Bob holds input y and 1^n. Let $\text{OUT}_{\Pi,\mathcal{A}(\mathsf{aux})}(x,y,n)$ be the random variable denoting the output of the honest party after this execution of the protocol. Set

$$\text{REAL}_{\Pi,\mathcal{A}(\mathsf{aux})}(x,y,n) \stackrel{\text{def}}{=} \left(\text{VIEW}_{\Pi,\mathcal{A}(\mathsf{aux})}(x,y,n),\ \text{OUT}_{\Pi,\mathcal{A}(\mathsf{aux})}(x,y,n)\right).$$

Execution in our ideal model. Here we describe the ideal model where the adversary may obtain one additional bit of information about the honest party's input. The parties are Alice and Bob, and there is an adversary $\mathcal{A}$ who has corrupted one of them. An ideal execution for the computation of f proceeds as follows:

Inputs: Alice and Bob hold 1^n and inputs x and y, respectively; the adversary $\mathcal{A}$ receives an auxiliary input aux.

Send inputs to trusted party: The honest party sends its input to the trusted party. The corrupted party controlled by $\mathcal{A}$ may send any value of its choice. Denote the pair of inputs sent to the trusted party as (x', y'). (We assume that if x' or y' are invalid then the trusted party substitutes some default input.) In addition, the adversary sends an arbitrary boolean function g to the trusted party.

Trusted party sends output: The trusted party computes $f(x', y')$ and $g(x', y')$, and gives both these values to the adversary. The adversary may at this point tell the trusted party to stop, in which case the honest party is given $\perp$. Otherwise, the adversary may tell the trusted party to continue, in which case the honest party is given $f(x', y')$. (As usual for two-party computation with malicious adversaries, it is impossible to guarantee complete fairness and we follow the usual convention of giving up on fairness altogether in the ideal world.)

Outputs: The honest party outputs whatever it was sent by the trusted party; $\mathcal{A}$ outputs an arbitrary function of its view.

We let $\text{OUT}^{\mathcal{A}}_{f,\mathcal{A}(\mathsf{aux})}(x,y,n)$ (resp., $\text{OUT}^{\mathsf{hon}}_{f,\mathcal{A}(\mathsf{aux})}(x,y,n)$) be the random variable denoting the output of $\mathcal{A}$ (resp., the honest party) following an execution in the ideal model as described above. Set

$$\text{IDEAL}_{f,\mathcal{A}(\mathsf{aux})}(x,y,n) \stackrel{\text{def}}{=} \left(\text{OUT}^{\mathcal{A}}_{f,\mathcal{A}(\mathsf{aux})}(x,y,n),\ \text{OUT}^{\mathsf{hon}}_{f,\mathcal{A}(\mathsf{aux})}(x,y,n)\right).$$

Definition 1 Let f, Π be as above. Protocol Π is said to *securely compute f with 1-bit leakage* if for every non-uniform probabilistic polynomial-time adversary $\mathcal{A}$ in the real model, there exists a non-uniform probabilistic polynomial-time adversary $\mathcal{S}$ in the ideal model such that

$$\left\{\text{IDEAL}_{f,\mathcal{S}(\mathsf{aux})}(x,y,n)\right\}_{x,y,\mathsf{aux}\in\{0,1\}^*} \stackrel{\text{c}}{\equiv} \left\{\text{REAL}_{\Pi,\mathcal{A}(\mathsf{aux})}(x,y,n)\right\}_{x,y,\mathsf{aux}\in\{0,1\}^*}$$

Remark. In the proof of security for our protocol, we consider a slight modification of the ideal model described above: namely, the adversary is allowed to *adaptively* choose g after learning $f(x', y')$. Although this may appear to be weaker than the ideal model described above (in that the adversary is stronger), in fact the models are identical. To see this, fix some adversary $\mathcal{A} = (\mathcal{A}_1, \mathcal{A}_2)$ in the "adaptive" ideal world, where $\mathcal{A}_1$ denotes the initial phase of the adversary (where the adversary decides what input to send to the trusted party) and $\mathcal{A}_2$ denotes the second phase of the adversary (where, after observing $f(x', y')$, the adversary specifies g). We can construct an adversary $\mathcal{A}'$ in the "non-adaptive" ideal world who learns the same information: $\mathcal{A}'$ runs $\mathcal{A}_1$ to determine what input to send, and also submits a boolean function g' defined as follows: $g'(x', y')$ runs $\mathcal{A}_2(f(x', y'))$ to obtain a boolean function g; the output is $g(x', y')$.

B. Proof of Security

We assume the DualEx protocol runs in a *hybrid world* where the parties are given access to trusted entities computing two functions: oblivious transfer and equality testing. (These trusted entities operate according to the usual ideal-world model where there is no additional one-bit leakage.) We show that the DualEx protocol securely computes f with one-bit leakage in this hybrid model. It follows from standard composition theorems [4] that the DualEx protocol securely computes f with one-bit leakage if the trusted entities are replaced by secure protocols (achieving the standard security definition against malicious adversaries).

Theorem 1. *If the garbled-circuit construction is secure against semi-honest adversaries and H is modeled as a random oracle, then the DualEx protocol securely computes f with one-bit leakage in the hybrid world described above.*

Proof: Let $\mathcal{A}$ denote an adversary attacking the protocol in a hybrid world where the parties have access to trusted entities computing oblivious transfer and equality testing. We assume that $\mathcal{A}$ corrupts Bob, though the proof is symmetric in the other case. We show that we can construct an adversary $\mathcal{S}$, running in our ideal world where the parties have access to a trusted entity computing f, that has the same effect as $\mathcal{A}$ in the hybrid world.

Construct $\mathcal{S}$ as follows:

1) $\mathcal{S}$, given inputs y and aux, runs $\mathcal{A}$ on the same inputs. It then simulates the first-stage oblivious transfers as follows: for the i^{th} oblivious transfer, $\mathcal{S}$ receives $\mathcal{A}$'s input bit y'_i and returns a random "input-wire label" w_i to $\mathcal{A}$.
2) $\mathcal{S}$ sets $y' = y'_1 \cdots y'_n$ and sends y' to the trusted entity computing f. It receives in return an output v_B.

3) $\mathcal{S}$ chooses random output-wire labels

$$\mathbb{W}_A^{v_B} \stackrel{\text{def}}{=} (w_{A1}^{v_{B1}}, \cdots, w_{A\ell}^{v_{B\ell}}).$$

Then, in the usual way (e.g., [26]), $\mathcal{S}$ gives to $\mathcal{A}$ a simulated garbled circuit constructed in such a way that the output-wire labels learned by $\mathcal{A}$ (given the input-wire labels chosen by $\mathcal{S}$ in the first step) will be precisely $\mathbb{W}_A^{v_B}$. Additionally, $\mathcal{S}$ chooses random $w_{A1}^{\bar{v}_{B1}}, \ldots, w_{A\ell}^{\bar{v}_{B\ell}}$, defines

$$\mathbb{W}_A = \begin{pmatrix} w_{A1}^0 & \cdots & w_{A\ell}^0 \\ w_{A1}^1 & \cdots & w_{A\ell}^1 \end{pmatrix},$$

and gives

$$\begin{pmatrix} H(w_{A1}^0) & \cdots & H(w_{A\ell}^0) \\ H(w_{A1}^1) & \cdots & H(w_{A\ell}^1) \end{pmatrix}$$

to $\mathcal{A}$. (The notation $H(\cdot)$ just means that $\mathcal{S}$ simulates a random function on the given inputs.) This completes the simulation of the first stage of the protocol.

4) Next, $\mathcal{S}$ simulates the second-stage oblivious transfers by simply recording, for all i, the "input-wire labels" (w_i^0, w_i^1) used by $\mathcal{A}$ in the i^{th} oblivious transfer. $\mathcal{A}$ then sends its second-phase message (which contains a garbled circuit, input-wire labels corresponding to its own input, and hashes of the output-wire labels).
5) Finally, $\mathcal{A}$ submits some input $\boldsymbol{w}_B \| \boldsymbol{w}'_B$ for the equality test. $\mathcal{S}$ then defines the following boolean function g (that depends on several values defined above):
   a) On input $x, y \in \{0,1\}^n$, use the bits of x as selector bits to define "input-wire labels" $w_1^{x_1}, \ldots, w_n^{x_n}$. Then, run stage 2 of the protocol exactly as an honest Alice would to obtain $v_A \in \{0,1\}^\ell$ and $\boldsymbol{w}_A$. (In particular, if some error is detected then random values are used for v_A and $\boldsymbol{w}_A$. These random values can be chosen by $\mathcal{S}$ in advance and "hard coded" into g.)
   b) Return 1 if $\mathbb{W}_A^{v_A} \| \boldsymbol{w}_A$ is equal to $\boldsymbol{w}_B \| \boldsymbol{w}'_B$; otherwise, return 0.

   $\mathcal{S}$ sends g to the trusted party, receives a bit z in return, and gives z to $\mathcal{A}$.
6) If $z = 0$ or $\mathcal{A}$ aborts, then $\mathcal{S}$ sends stop to the trusted entity. Otherwise, $\mathcal{S}$ sends continue. In either case, $\mathcal{S}$ then outputs the entire view of $\mathcal{A}$ and halts.

To complete the proof, we need to show that

$$\left\{\text{IDEAL}_{f,\mathcal{S}(\mathsf{aux})}(x,y,n)\right\}_{x,y,\mathsf{aux}\in\{0,1\}^*} \stackrel{\text{c}}{\equiv} \left\{\text{REAL}_{\Pi,\mathcal{A}(\mathsf{aux})}(x,y,n)\right\}_{x,y,\mathsf{aux}\in\{0,1\}^*}$$

(where, above, the second distribution refers to the execution of DualEx in the hybrid world where the parties have access to trusted entities computing oblivious transfer and equality). This is fairly straightforward since there are only two differences between the distribution ensembles:

1) In the real world the garbled circuit sent by Alice to $\mathcal{A}$ is constructed correctly based on Alice's input x, while in the ideal world the garbled circuit is simulated based on the input-wire values given to $\mathcal{A}$ and the output v_B obtained from the trusted entity computing f.
2) In the real world the output of the honest Alice when the equality test succeeds is v_A, whereas in the ideal world it is v_B (since v_B is the value sent to Alice by the trusted entity computing f).

Computational indistinguishability of the first change follows from standard security proofs for Yao's garbled-circuit construction [26]. For the second difference, the probability (in the ideal world) that the equality test succeeds and $v_B \neq v_A$ is negligible. The only way this could occur is if $\mathcal{A}$ is able to guess at least one value $w_{Ai}^{\bar{v}_{Bi}}$; but, the only information $\mathcal{A}$ has about any such value is $H(w_{Ai}^{\bar{v}_{Bi}})$. Thus, $\mathcal{A}$ cannot guess any such value except with negligible probability. ■

VI. Evaluation

A. Implementation

Since there is no need for any party to keep the circuit locally as is required for cut-and-choose, the execution of the garbled circuit sub-protocol (Figure 3) can be pipelined as for ordinary semi-honest secure computing protocols. We implement this protocol using the framework of Huang et al. [16]. This framework provides the most efficient known implementation of semi-honest garbled circuit protocols by incorporating circuit-level optimizations (including bit width minimization and extensive use of the free-XOR technique) and scalability by using a pipelined execution process where garbled gates are transmitted to the evaluator and evaluated as they are ready, thereby reducing latency and avoiding the need for either the generator or evaluator to ever keep the entire garbled circuit in memory.

In adapting this framework to support dual execution protocols, we observe that Stage 1 and Stage 2 of the dual execution protocol are actually two independent executions of the same semi-honest protocol. Their executions can be overlapped, with both parties simultaneously running the execution where they are the generator and the one where they are the evaluator as two separate threads executing in parallel. Since the workload for the different roles is different, this has additional benefits. Because the generator must perform four encryptions to generate each garbled table, while the evaluator only has to perform a single decryption, the workload for the party acting as the generator is approximately four times that of the evaluator. During normal pipelined execution, this means the circuit evaluator is idle most of the time. With simultaneous dual execution, however, both parties have the same total amount of work to do, and nearly all the previously idle time can be used usefully.

B. Experimental Setup

Hardware & Software. The experiments are done on two standard Dell boxes, each equipped with an Intel® Core™ 2 Duo E8400 3GHz processor, 8 GB memory. They are connected with a 100 Mbps LAN. Both boxes are running Linux 3.0.0-12 (64 bit). The JVM version we used is

Sun®Java™1.6 SE. Our implementation is available under an open source license from http://www.MightBeEvil.com.

Security Parameters. We use 80-bit nonces to represent wire labels. In our implementation of the Naor-Pinkas OT protocol, we use an order-q cyclic subgroup of $\mathbb{Z}_p^*$ where $|p| = 1024$ and $|q| = 160$. For the implementation of OT extension, we used $k = 80$ and 80-bit symmetric keys. Our security parameters conform to the ultra-short security level recommended by NIST [2].

Applications. We demonstrate the effectiveness of the DualEx protocol with several secure two-party computation applications including private set intersection (PSI), which enables computing the intersection of two secret sets without revealing them, secure edit distance (ED), which computes the edit distance between two secret strings, and private AES encryption, where the key and message are supplied by different entities and kept secret throughout the ciphering. These applications are representative of commonly studied privacy-preserving applications in the literature and were selected for easy performance comparison. Our implementations are based on the framework of Huang et al. [16, 15].

C. Results

Figure 6 summarizes the running time for the three applications running under different settings. The PSI instance is computed over two sets each containing 4096 32-bit numbers using the *Sort-Compare-Shuffle with Waksman Network* (SCS-WN) protocol [15]. The edit distance is calculated from two strings each having 200 8-bit characters. The AES instance is executed in 128-bit key size mode, with 100 iterations. The measurements are the average time over 20 runs of each protocol with randomly generated private inputs (of course, in a secure computation protocol, the running time cannot depend on the actual input values since all operations must be data-independent). We compare our results for DualEx protocols with the results for the best known semi-honest protocols [16, 15], which uses a single garbled circuit execution using the same framework upon which our DualEx protocols are built.

The measurements include time spent on direct transfer of wire labels, the online phase of oblivious transfer, circuit generation and evaluation, and secure validity test. The time used to initialize the circuit structure and oblivious transfer is not included since these are one-time costs that can be performed off-line.

For symmetric input applications (PSI and ED), we observe the bandwidth cost of dual execution protocols is exactly twice of that for semi-honest protocols. The running time of DualEx protocols running on a dual-core hardware is only slightly higher than that for the corresponding semi-honest protocol. All of the work required for the second execution is essentially done simultaneously with the first execution using the otherwise idle core. The only additional overhead is the very inexpensive equality test at the end of the protocol.

On the other hand, for asymmetric input applications like AES, the dual execution protocol appears to be slower. The reason is that in the semi-honest settings the party holding the message is always designated the circuit generator such that the more expensive oblivious transfers need only to be used for the encryption key (which is shorter than the message). In the DualEx protocol every input bit needs to be obliviously transferred once. Thus, it runs slower than its semi-honest version deployed in favor of using less OTs.

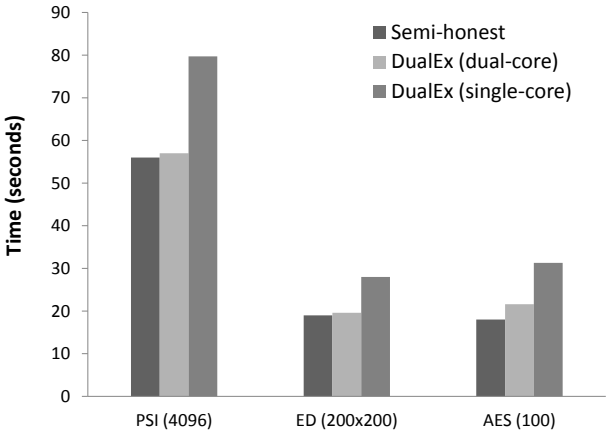

Figure 6. Time costs comparing to semi-honest protocols.

We do not include the time required to compute the 80 "base" OTs (about 12 seconds) in the timing measurements, since this is a one-time, fixed cost that can be pre-computed independent of the actual function to be computed.

Although our implementation is programmed explicitly in two Java threads, we have also run it using a single core for fair comparisons. We used the same software and hardware setup but the processes are confined to be run on a single core using the `taskset` utility command. The corresponding results are shown as the third column in Figure 6. Note that the added overhead is only 42%–47% than a semi-honest run even if two semi-honest runs are included in the dual execution protocol. Recall that in the semi-honest garbled circuit protocol, the point-and-permute [29] technique sets the workload ratio between the generator and the evaluator to four to one, because the evaluator needs to decrypt only one of the four entries in a garbled truth table. Moreover, garbled-row-reduction [34] optimization brings this ratio down to about 3, since only 3 out of 4 entries in a garbled truth table needs to be transmitted. Therefore, should the overhead of thread-level context switch and interferences are ignored, the slowdown factor of dual execution will be about of 33%. (We assume the network bandwidth is not the bottleneck, which is true on a wired LAN.) Our experimental results actually show that about another 15% of time is lost due to the interferences between the two threads.

The scale of the circuits used in our experiments above is already well beyond what has been achieved by state-of-art maliciously-secure secure two-party computation prototypes. However, to fully demonstrate the memory efficiency of

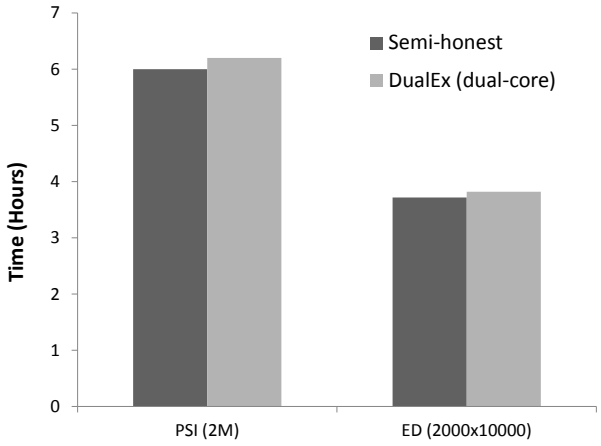

Figure 7. Time costs for large scale problems.

the dual execution approach, we also report results from running the PSI and edit distance applications on larger problem sizes. The timing results are shown in Figure 7 for performing private set intersection on two sets of one million 32-bit values each, and for an edit-distance computation with input DNA sequences (2-bit character) of 2000 and 10000. The performance of DualEx protocols remains very competitive with semi-honest secure computation protocols even for large inputs.

VII. Enhancements

One problem with the basic DualEx protocol is that it allows the attacker to learn the output of $f(x, y)$ even when cheating since the output is revealed before the equality test. Consequently, this advantage for adversaries could actually encourage participants to cheat and would be unacceptable in many scenarios.

In this section, we present two heuristic enhancements that aim at mitigating the problem. (We leave formal definitions and proofs of security for future work.) The first enhancement, called *progressive revelation*, is the most straightforward and guarantees that the adversary has can only learn one more bit of the output than the honest party. The second enhancement, we call *DualEx-based equality test*, ensures the outputs are revealed only after the equality check passes. Note that since the two enhancements are orthogonal, they can be combined to construct a improved DualEx protocol that benefits from both.

In both enhancements, to prevent early revelation of outputs we change the output revelation process in the basic DualEx protocol by replacing the final step in the garbled circuit sub-protocol (execution step 3 from Figure 3) with a step that just outputs the wire labels without decoding their semantic values:

3) P_1 outputs $\mathbb{W}_1$ that it produced when generating the circuit, while P_2 outputs $\boldsymbol{w}_1^{\boldsymbol{v}_2}$ that it obtains from circuit evaluation.

This changes the output P_2 receives to only include the output-wire labels (and not the underlying bits to which they map).

The delayed revelation modification prevents the semantic values from being learned at the end of each semi-honest protocol execution, and supports the two protocol variations discussed next for revealing the semantic values in a way that ensures a limited notion of fairness.

A. DualEx-based Equality Test

Our goal is to prevent an adversary from learning the output if it is caught cheating by the equality test. To achieve this, we introduce a pre-emptive secure equality-test protocol that is done before output-wire label interpretation. In addition, compared to the secure equality test used in the basic DualEx protocol (Section III), the test here has to start from output-wire labels (as opposed to be able to use the previously-revealed outputs).

The goal of the pre-emptive equality test is to compute in a privacy-preserving way the predicate $Equal = \mathsf{AND}(Equal_1, Equal_2, \ldots, Equal_\ell)$ in which $Equal_i$ is defined as follows,

$$Equal_i = \begin{cases} 1, & \text{if } \exists \sigma, \text{s.t. } w_{Ai} = w_{Ai}^{\sigma} \textbf{ and } w_{Bi} = w_{Bi}^{\sigma}; \\ 0, & \text{otherwise.} \end{cases}$$

where w_{Ai} (respectively w_{Bi}) is the i^{th} output-wire label Bob (respectively Alice) obtained from circuit evaluation.

The basic idea is to implement *Equal* with a garbled circuit, which ANDs all ℓ wires from ℓ Equal_i circuits. The circuit Equal_i can be implemented as shown in Figure 8. The cost of $Equal_i$ is 2σ non-free gates (where σ is the length of a wire label), while the *Equal* circuit requires $2\ell\sigma$ non-free gates. Thus, its cost does not grow with the length of f's inputs nor the f's circuit size (which can be very large).

We could execute the *Equal* circuit with any generic protocol secure against malicious adversaries. Alternatively, the basic DualEx protocol can be employed to keep the overhead low. Note that on average one-bit could be leaked using the DualEx protocol here, but it is a bit about the random nonces used as wire labels, hence does not expose the original private inputs.

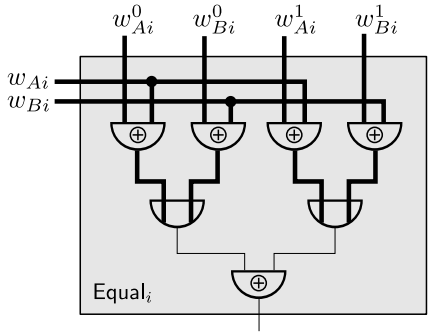

Figure 8. Circuit realization of $Equal_i$.

B. Progressive Revelation

The goal of this variation is to reveal the output wires to both parties in a bitwise fashion, until cheating (if there is any) is detected on one output wire. Hence, if the outputs match exactly, both parties will receive the full output at the end of the protocol. If the outputs do not match, both parties will receive the same matching output bits until the first mismatch and the adversary receives at most a single additional mismatched output bit.

The idea resembles that of the gradual release protocols used for exchanging secret keys [3, 6], signing contracts [8], and secure computation [18]. In our scenario, to reveal the i^{th} bit of the output, the parties can securely evaluate a circuit $\mathsf{EqualRev}_i$ (Figure 9), which tests equality (indicated by v_i) and reveals the output bit (denoted by o_i) at the same time. This circuit looks exactly the same as Equal_i except it has an extra o_i bit which set to 0 if and only if $w_{Ai} = w^0_{Ai}$ and $w_{Bi} = w^0_{Bi}$. The $v_i = 1$ bit implies the o_i bit is indeed valid.

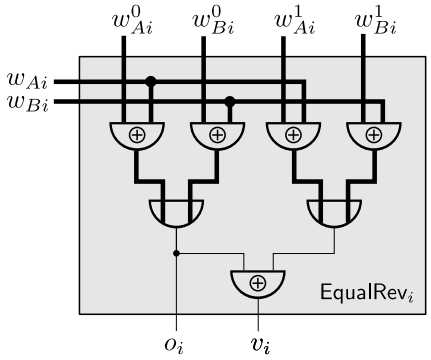

Figure 9. Circuit realization of *EqualRev*$_i$.

To further limit an adversary's advantage, we can require the output-wire labels interpretation process to be done in an order that is collectively determined by both parties. For example, let p_a and p_b denote two random permutations solely determined by Alice and Bob, respectively. The two parties will reveal the output bits in the order determined by the permutation $p = p_a \oplus p_b$. Note that to make sure each party samples its random permutation independent of the other's permutation, p_a and p_b need to be committed before they are revealed.

VIII. Conclusion

Previous work in secure computation has left an enormous efficiency gap between protocols in the semi-honest and malicious models. This work demonstrates the potential of an alternate approach for security against a malicious adversary, which relaxes the security properties by allowing a single bit of extra information to leak. This relaxation allows us to implement privacy-preserving applications with much stronger security guarantees than semi-honest protocols, but with minimal extra cost. The applications scale to large inputs on commodity machines, including million-input private set intersection.

Acknowledgments

The authors thank Peter Chapman, Greg Morrisett, Abhi Shelat, David Wagner, and Samee Zahur for useful comments on this work. This work was supported by grants from the National Science Foundation, Air Force Office of Scientific Research, and DARPA. The contents of this paper, however, do not necessarily reflect the views of the US Government.

References

[1] Y. Aumann and Y. Lindell. Security against covert adversaries: Efficient protocols for realistic adversaries. *Journal of Cryptology*, 23(2):281–343, 2010.

[2] E. Barker, W. Barker, W. Burr, W. Polk, and M. Smid. NIST special publication 800-57: Recommendation for key management — part 1, March 2007.

[3] M. Blum. How to exchange (secret) keys. *ACM Transactions on Computer Systems*, 1(2):175–193, 1983.

[4] R. Canetti. Security and composition of multiparty cryptographic protocols. *Journal of Cryptology*, 13(1):143–202, 2000.

[5] D. Dachman-Soled, T. Malkin, M. Raykova, and M. Yung. Efficient robust private set intersection. In *7th Intl. Conference on Applied Cryptography and Network Security (ACNS)*, volume 5536 of *LNCS*, pages 125–142. Springer, 2009.

[6] I. Damgård. Practical and provably secure release of a secret and exchange of signatures. *Journal of Cryptology*, 8(4):201–222, 1995.

[7] E. De Cristofaro, J. Kim, and G. Tsudik. Linear-complexity private set intersection protocols secure in malicious model. In *Advances in Cryptology — Asiacrypt 2010*, volume 6477 of *LNCS*, pages 213–231. Springer, 2010.

[8] S. Even, O. Goldreich, and A. Lempel. A randomized protocol for signing contracts. *Communications of the ACM*, 28(6):637–647, 1985.

[9] M. J. Freedman, K. Nissim, and B. Pinkas. Efficient private matching and set intersection. In *Advances in Cryptology — Eurocrypt 2004*, volume 3027 of *LNCS*, pages 1–19. Springer, 2004.

[10] O. Goldreich. *Foundations of Cryptography, vol. 2: Basic Applications*. Cambridge University Press, Cambridge, UK, 2004.

[11] O. Goldreich, S. Micali, and A. Wigderson. How to play any mental game, or a completeness theorem for protocols with honest majority. In *19th Annual ACM Symposium on Theory of Computing (STOC)*, pages 218–229. ACM Press, 1987.

[12] D. Harnik, Y. Ishai, E. Kushilevitz, and J. B. Nielsen. OT-combiners via secure computation. In *5th Theory of*

Cryptography Conference — TCC 2008, volume 4948 of *LNCS*, pages 393–411. Springer, 2008.

[13] C. Hazay and Y. Lindell. Efficient protocols for set intersection and pattern matching with security against malicious and covert adversaries. In *5th Theory of Cryptography Conference — TCC 2008*, volume 4948 of *LNCS*, pages 155–175. Springer, 2008.

[14] W. Henecka, S. Kögl, A.-R. Sadeghi, T. Schneider, and I. Wehrenberg. TASTY: tool for automating secure two-party computations. In *17th ACM Conf. on Computer and Communications Security (CCS)*, pages 451–462. ACM Press, 2010.

[15] Y. Huang, D. Evans, and J. Katz. Private set intersection: Are garbled circuits better than custom protocols? In *Network and Distributed System Security Symposium (NDSS)*. The Internet Society, 2012.

[16] Y. Huang, D. Evans, J. Katz, and L. Malka. Faster secure two-party computation using garbled circuits. In *20th USENIX Security Symposium*, 2011.

[17] Y. Huang, L. Malka, D. Evans, and J. Katz. Efficient privacy-preserving biometric identification. In *Network and Distributed System Security Symposium (NDSS)*, pages 421–434. The Internet Society, 2011.

[18] R. Impagliazzo and M. Yung. Direct minimum-knowledge computations. In *Advances in Cryptology — Crypto '87*, volume 293 of *LNCS*, pages 40–51. Springer, 1988.

[19] Y. Ishai, J. Kilian, K. Nissim, and E. Petrank. Extending oblivious transfers efficiently. In *Advances in Cryptology — Crypto 2003*, volume 2729 of *LNCS*, pages 145–161. Springer, 2003.

[20] S. Jarecki and X. Liu. Efficient oblivious pseudorandom function with applications to adaptive OT and secure computation of set intersection. In *6th Theory of Cryptography Conference — TCC 2009*, volume 5444 of *LNCS*, pages 577–594. Springer, 2009.

[21] S. Jarecki and X. Liu. Fast secure computation of set intersection. In *7th Intl. Conf. on Security and Cryptography for Networks*, volume 6280 of *LNCS*, pages 418–435. Springer, 2010.

[22] S. Jarecki and V. Shmatikov. Efficient two-party secure computation on committed inputs. In *Advances in Cryptology — Eurocrypt 2007*, volume 4515 of *LNCS*, pages 97–114. Springer, 2007.

[23] V. Kolesnikov and T. Schneider. Improved garbled circuit: Free XOR gates and applications. In *35th Intl. Colloquium on Automata, Languages, and Programming (ICALP), Part II*, volume 5126 of *LNCS*, pages 486–498. Springer, 2008.

[24] Y. Lindell, E. Oxman, and B. Pinkas. The IPS compiler: Optimizations, variants and concrete efficiency. In *Advances in Cryptology — Crypto 2011*, volume 6841 of *LNCS*, pages 259–276. Springer, 2011.

[25] Y. Lindell and B. Pinkas. An efficient protocol for secure two-party computation in the presence of malicious adversaries. In *Advances in Cryptology — Eurocrypt 2007*, volume 4515 of *LNCS*, pages 52–78. Springer, 2007.

[26] Y. Lindell and B. Pinkas. A proof of security of Yao's protocol for two-party computation. *Journal of Cryptology*, 22(2):161–188, 2009.

[27] Y. Lindell and B. Pinkas. Secure two-party computation via cut-and-choose oblivious transfer. In *8th Theory of Cryptography Conference — TCC 2011*, volume 6597 of *LNCS*, pages 329–346. Springer, 2011.

[28] Y. Lindell, B. Pinkas, and N. Smart. Implementing two-party computation efficiently with security against malicious adversaries. In *6th Intl. Conf. on Security and Cryptography for Networks (SCN '08)*, volume 5229 of *LNCS*, pages 2–20. Springer, 2008.

[29] D. Malkhi, N. Nisan, B. Pinkas, and Y. Sella. Fairplay — a secure two-party computation system. In *Proc. 13th USENIX Security Symposium*, 2004.

[30] P. Mohassel and M. Franklin. Efficiency tradeoffs for malicious two-party computation. In *9th Intl. Conference on Theory and Practice of Public Key Cryptography(PKC 2006)*, volume 3958 of *LNCS*, pages 458–473. Springer, 2006.

[31] M. Naor and B. Pinkas. Efficient oblivious transfer protocols. In *ACM-SIAM Symposium on Discrete Algorithms (SODA)*, 2001.

[32] J. Nielsen, P. Nordholt, C. Orlandi, and S. Burra. A new approach to practical active-secure two-party computation. Available at `http://eprint.iacr.org/2011/091`.

[33] J. B. Nielsen and C. Orlandi. LEGO for two-party secure computation. In *6th Theory of Cryptography Conference — TCC 2009*, volume 5444 of *LNCS*, pages 368–386. Springer, 2009.

[34] B. Pinkas, T. Schneider, N. Smart, and S. Williams. Secure two-party computation is practical. In *Advances in Cryptology — Asiacrypt 2009*, volume 5912 of *LNCS*, pages 250–267. Springer, 2009.

[35] A. Shelat and C.-H. Shen. Two-output secure computation with malicious adversaries. In *Advances in Cryptology — Eurocrypt 2011*, volume 6632 of *LNCS*, pages 386–405. Springer, 2011.

[36] A. C.-C. Yao. How to generate and exchange secrets. In *27th Annual Symposium on Foundations of Computer Science (FOCS)*, pages 162–167. IEEE, 1986.