Invited paper in 7^{th} *International Conference on Information Systems Security*,
Kolkata, India, 15-19 December 2011

Efficient Secure Computation with Garbled Circuits

Yan Huang[1], Chih-hao Shen[1], David Evans[1], Jonathan Katz[2], and abhi shelat[1]

[1] University of Virginia, Department of Computer Science
[2] University of Maryland, Department of Computer Science

http://www.SecureComputation.org

Abstract. Secure two-party computation enables applications in which participants compute the output of a function that depends on their private inputs, without revealing those inputs or relying on any trusted third party. In this paper, we show the potential of building privacy-preserving applications using garbled circuits, a generic technique that until recently was believed to be too inefficient to scale to realistic problems. We present a Java-based framework that uses pipelining and circuit-level optimizations to build efficient and scalable privacy-preserving applications. Although the standard garbled circuit protocol assumes a very week, honest-but-curious adversary, techniques are available for converting such protocols to resist stronger adversaries, including fully malicious adversaries. We summarize approaches to producing malicious-resistant secure computations that reduce the costs of transforming a protocol to be secure against stronger adversaries. In addition, we summarize results on ensuring *fairness*, the property that either both parties receive the result or neither party does. Several open problems remain, but as theory and pragmatism advance, secure computation is approaching the point where it offers practical solutions for a wide variety of important problems.

1 Introduction

Data gains value when it can be used in computations with data from other sources. For example, my genetic information becomes much more valuable when I can use it in a computation to measure kinship with another individual's genetic data, contribute it to a scientific genome association study, or use it to analyze drug effectiveness by comparing it to the genomes of participants in a pharmaceutical study. All of those uses, however, seem to require exposing my private data to other parties (or the other parties being willing to provide their private data to me). This leaves individuals with a dilemma: either maintain privacy, but lose much of the potential value of their data; or give up on privacy and expose ones data to malicious uses.

Secure computation provides an attractive alternative. It enables data owners to keep their data private, while allowing it to be used in computations. Secure two-party computation allows two parties to cooperatively compute the output of a function, $f(a,b)$, without revealing any information about their private inputs, a and b respectively (other than what can be inferred from the function output). The idea of secure function evaluation was introduced by Andrew Yao in the 1980s [52, 53], but it has only recently

become realistic to imagine large-scale, important problems being solved by practical secure computations. Realizing such secure computations would enable many real world applications. For example, government agencies could use it to implement biometric security checks [29] (e.g., the no-fly list) and video criminal identification using street cameras [28, 47], without compromising the privacy of innocent citizens. If secure computation protocols can be inexpensive enough to execute on mobile devices, it could also enable applications using smartphones such as proximity-based voting, common interest and contacts matching, and real-time marketing [27].

In this paper, we focus on a generic approach to secure two-party computation known as *garbled circuits* or *Yao circuits* [52]. A garbled circuit protocol allows two semi-honest parties, a circuit *generator* and a circuit *evaluator*, to compute an arbitrary function $f(a,b)$, where a and b are private inputs from each party, without leaking any information about their respective secret inputs beyond what is revealed by the function output itself. We provide background on the garbled circuit protocol in Section 2. Although garbled circuit protocols have a reputation for being inefficient and requiring excessive amounts of memory, there are ways to implement garbled circuit protocols that take advantage of pipelining and circuit-level optimizations to enable much faster execution. Section 3 describes our framework for efficient garbled circuit protocols and reports on results using it for several applications.

An important parameter of any secure computation protocol is the threat model. The weakest commonly used threat model is the *semi-honest* threat model, where both parties are assumed to follow the protocol as specified but attempt to learn additional information about the other party's private inputs from the protocol transcript. This is the easiest model in which to build scalable applications, and the model most frequently used by implemented systems [9,26,32,37,47,51]). Although this model may be appropriate for some realistic situations in which both parties have limited ability to interfere with the execution or a vested interest in the correctness of the results, it assumes a very weak adversary so is insufficient for many important use scenarios.

The strongest model is called *malicious* adversary model, where an attacker can deviate arbitrarily from the protocol specification to pry on other parties privacy. Several techniques have been proposed for converting a protocol that is secure under the semi-honest model into a protocol that is secure against a malicious adversary, which we discuss in Section 4, along with our work on making these conversions less expensive.

A second important facet of secure computation is *fairness*, which requires the participating parties obtaining the results of the computation simultaneously. Although fairness seems impossible to achieve since one party could just abort the protocol after obtaining the result, surprisingly, it turns out that fairness is achievable for some functions, and that for other functions a relaxed definition of partial fairness may be useful. We discuss our results on ensuring *fairness* in Section 5.

2 Garbled Circuits Background

Garbled circuits were introduced by Yao [53] as a generic mechanism for secure computation. A standard garbled circuit protocol involves two parties who wish to cooperatively compute the output of a function that depends on private data from both parties

without revealing that private data. One party is known as the *generator*, who produces a garbled circuit for computing the function. The other party, the *evaluator*, evaluates that circuit to produce the (encrypted) result, which can then be revealed to either or both parties.

Any binary gate, f, which has two input wires W_0, W_1 and one output wire W_2, can be realized as a garbled gate. First, generate random nonces w_i^0 and w_i^1 to represent signal 0 and signal 1 on each wire W_i. Then, generate a truth table of four entries,

$$\mathrm{Enc}_{w_0^{s_0}, w_1^{s_1}}\left(w_2^{f(s_0,s_1)}\right)$$

where s_0, s_1 denote the 1-bit plain signal on wire W_0, W_1, respectively. The table entries are then randomly permuted so no information is revealed by the encrypted table. We call this encrypted and permuted truth table a *garbled table*.

Next, the garbled table and the wire labels, $w_0^{s_0}$, representing the generator's secret input, are sent to the evaluator. To obtain the appropriate wire label for her own input (without revealing the input), $w_1^{s_1}$, the evaluator and generator execute an *oblivious transfer* protocol (see Section 2.1). Thus, the evaluator can decrypt one and only one entry that corresponds exactly to their inputs. Following this construction strategy, an arbitrary number of binary gates can be assembled to accomplish general purpose computation using the output wire labels of one gate as the input labels of the next gate.

In summary, a garbled circuit protocol involves three main steps: (1) the circuit generator garbles the circuit's truth tables; (2) the circuit generator directly transfers the circuit and garbled truth tables, and obliviously transfers the appropriate input labels to the evaluator; and (3) the circuit evaluator evaluates the circuit by successively decrypting the entry of each garbled table corresponding to the available input wires to learn the output wire labels necessary to fully evaluate a single path through the circuit.

Garbled circuits can compute any function as a two-party secure computation, and can be easily extended to support multi-party computation. The protocol provides security in the *semi-honest* threat model, where each party is assumed to follow the protocol as specified but attempts to learn additional information about the other party's inputs by observing the protocol execution. In Section 4, we consider approaches to making garbled circuits protocols resilient to stronger adversaries.

Next, we provide background on the oblivious transfer protocols needed to exchange inputs at the beginning of a garbled circuit protocol. Section 2.2 describes some important improvements to the basic garbled circuit protocol that enable more efficient execution. Section 2.3 describes previous frameworks designed to make it easier to build garbled circuit protocols.

2.1 Oblivious Transfer

An oblivious transfer 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. For example, a 1-out-of-2 oblivious transfer protocol (denoted OT_1^2) allows the sender, who has two bits b_0 and b_1, to transfer b_σ to the receiver, where $\sigma \in \{0, 1\}$ is kept secret to the receiver throughout the protocol. OT_1^2 was first proposed by Even, Goldreich, and Lempel [12]. Naor and Pinkas developed an

efficient OT_1^2 protocol based on Decisional Diffie-Hellman (DDH) hardness assumption [44]. We use this technique in our implementation. Based on the random oracle assumption, Ishai et al. devised a novel technique to reduce the cost of doing m OT_1^2 transfers to k OT_1^2, where k, ($k \ll m$), serves as a configurable security parameter [30].

2.2 Improvements

Several techniques have been developed to improve garbled circuit protocols. This section describes a few of the most important enhancements.

The *point-and-permute* technique allows the circuit evaluator to identify the "right" entry in a garbled truth table to decrypt [41], saving the evaluator from decrypting more than one truth table entry. The basic idea is to assign an n-bit random string (say $p = p_1 p_2 \dots p_n$) for every garbled truth table with 2^n entries. The random string determines how the generator swaps the entries in the table (swapping every other 2^j consecutive entries if $p_j = 1$). For the i-th entry in the truth table, the generator sends an n-bit string $p' = p'_1 p'_2 \dots p'_n$ (where $p'_k = p_k \oplus b_k$ and $b_1 b_2 \dots b_n$ is the binary representation of the number i), which indexes the "right" entry for the evaluator to decrypt. Considering the full truth table, revealing p' does not leak anything about p, so the evaluator cannot learn any extra information about p.

The *free-XOR* technique [35, 36] realizes all XOR gates by just XOR-ing the input wire labels, without needing any encryption operations. Its security was originally proven in the random oracle model [36]. The idea is to select wire labels where $w_i^1 = w_i^0 \oplus R$ where R is a random nonce. This allows the XOR of two wire labels to be computed by simply XOR-ing the wire labels, without requiring any encryption or communication. Choi et al. proved that it is secure under a notion of *circular* correlation robustness (which is weaker than the random oracle assumption), but is not secure under a standard variant of the correlation robustness assumption [10].

The *Garbled Row Reduction* (GRR) technique reduces the size of a garbled table of binary gates to three entries (saving 25% of network bandwidth) for all non-free gates. This is achieved by simply assigning

$$w_{out}^0 = \mathrm{Enc}_{w_{in_1}^0, w_{in_2}^0}(0)$$

where w_{out}^0 denotes the wire label representing 0 on the output wire while w_{in_1} and w_{in_2} denote the two input wires. This eliminates the necessity to transmit the encryption of one particular entry in the table. This technique is composable with both the free-XOR and point-and-permute techniques.

2.3 Frameworks

Without appropriate programming tools, programmers would have to spend a great deal of tedious effort to build a privacy-preserving application using garbled circuits. Various programming tools have been developed to automate parts of building secure computations. The most widely used garbled circuits framework is *Fairplay* [41], developed by Malkhi et al. and extended by several subsequent papers. Fairplay is a compile-and-interpret framework that automates the production of secure two-party computation applications. The main interface Fairplay exposes to programmers is a simple Algol-like

programming language called *Secure Function Description Language* (SFDL) that supports very limited primitive data types (`boolean`, sized `int` and `enumerate`), expressions (addition, subtraction, comparison, and Boolean logical operations), and statements (non-recursive functions, branches, and constant number iterative loops). SFDL programs are compiled to a monolithic digital circuit which is interpreted by the server/-client runtime environments for protocol execution.

Several subsequent frameworks have built upon Fairplay. FairplayMP [8] extended the SFDL language to describe secure multi-party computations, using a circuit-based technique for multi-party computation [5]. TASTY [26] extended Fairplay's SFDL to allow the programmer to specify where in the digital circuit to integrate some arithmetic circuits (limited to addition and constant multiplication) that are realized by additive homomorphic encryption schemes.

Although Fairplay and similar previous secure computation frameworks demonstrated that it is feasible to automate building secure computing protocols, they also led to the false impression that such protocols are unlikely to be useful in practice because of their poor efficiency and scalability. Many researchers (e.g., [32, 47]) concluded that the generic garbled circuit technique is not suitable for solving real world computational problems, and instead developed custom solutions for particular problems [25, 32, 47]. In the next section, we argue that the efficiency and scalability problems attributed to garbled circuits are not inherent in the protocol, but can be overcome by a more efficient framework design.

3 Efficient Garbled Circuits Framework

There are two main reasons why secure computations implemented using Fairplay and similar frameworks tend to be slow and unscalable. The first is that the design of Fairplay requires that the entire circuit is constructed and stored in memory before evaluation can begin. These circuits are huge since each gate requires a garbled table (three encrypted values using the GRR technique) and gates may not be reused since that would leak information. Users of Fairplay have found that the memory required to store the garbled circuit prevents implementations from scaling to large inputs (as an example, Jha et al. failed to compute the edit distance of two 200-character sequences due to memory constraints thus concluded garbled circuit alone is not suitable for large circuits [32]). We address this problem by pipelining the generation and execution of the circuit so there is no need to ever have the entire circuit in memory (Section 3.1).

The other main problem with the Fairplay approach is also its main advantage: computations are represented as high-level, Algol-like programs. The problem with starting from a high-level representation is that it prevents many important optimization opportunities. Although it may one day be possible to automate these optimizations, important optimizations are well beyond the capabilities of current tools. Our approach is to adopt a circuit-level representation that enables both higher-level and lower-level optimizations to greatly improve the efficiency of generated protocols (Section 3.2).

Section 3.3 provides details on our implementation, and Section 3.4 reports on results for several applications.

3.1 Pipelined Circuit Execution

The primary limitation of previous garbled circuit implementations is the memory required to store the entire circuit in memory. For example, Pinkas et al.'s privacy-preserving AES implementation involved 11,500 non-free gates [48], each of which requires a garbled table of encrypted wire values. For problems like Levenshtein distance the size of the circuit scales with the size of the input, so only relatively small inputs can be handled.

There is no need, however, for either the circuit generator or circuit evaluator to ever hold the entire circuit in memory. The circuit generating and evaluating processes of different gates can actually be overlapped in time. In our framework, the processing of the garbled truth tables is *pipelined* to avoid the need to store the entire circuit and to save processing time. At the beginning of the evaluation, both the generator and evaluator instantiate the circuit structure, which is known to both and fairly small since it can reuse components just like a non-garbled circuit. Note that the process of generating and evaluating the circuit does not (indeed, it cannot, because of privacy) depend on the inputs, so there is no overhead required to keep the two parties synchronized.

Our framework automates the pipelined execution, so a user only needs to construct the desired circuit. When the protocol is executed, the generator transmits garbled truth tables over the network as they are produced, in an order defined by the circuit structure. As the client receives the garbled truth tables, it associates them with the corresponding gate. The client determines which gate to evaluate next based on the available output values and tables. Gate evaluation is triggered automatically when all the necessary inputs are ready. Once a gate has been evaluated it is immediately discarded, so the number of garbled tables stored in memory is minimal. Evaluating larger circuits does not substantially increase the memory load on the generator or evaluator, only the network bandwidth needed to transmit the garbled tables.

3.2 Generating Efficient Circuits

Since pipelined execution eliminates the memory bottleneck, the cost of evaluating a garbled circuit protocol scales linearly with the number of garbled gates. One way to reduce the number of gates is to identify parts of the computation that only require private inputs from one party. These components can be computed directly by that party so do not require any garbled circuits. By designing circuits at the circuit-level rather than using a high-level language like SFDL, users of our framework can take advantage of these opportunities (for example, by computing the key schedule for AES locally and transmitting it obliviously).

For the parts of the computation that involve private data from both parties so must be done cooperatively, we exploit several opportunities enabled by our framework for minimizing the number of non-free gates in our circuits.

Circuit Library. A programmer can create circuits using a library of basic circuits (e.g., comparator, adder, muxer, min) designed to make the best use of free-XOR techniques. This serves as one solution to the more general goal of replacing expensive AND and OR gates with XOR gates, which are free. Our goals are distinguished from conventional hardware circuit design in that the latter aims to optimize circuits under a completely

different set of criteria, such as total number of gates, area, and power consumption. Also, since each garbled gate can only be evaluated once, the reuse goals of hardware circuit design do not apply to garbled circuits.

Minimizing Width. In garbled circuits, every bit of computation requires very expensive operations. To improve performance, our circuits are constructed with the minimal width required for the correctness of the programs. For example, if one wants to count the number of 1's in a 900-bit number, as is encountered in a face recognition application, SFDL's simplicity encourages programmers to write code that leads to a circuit that uses 10-bit accumulators throughout the computation. However, narrower accumulators are sufficient for early stages. The Hamming distance, Levenshtein distance, and Smith-Waterman applications all take advantage of this technique. For example, this technique reduces the cost for our Levenshtein distance protocol by about 20%.

Propagating Known Values. Our framework automatically propagates known wire signals when the circuit is built. For example, given a circuit designed for Hamming distance of 1024×1024 vectors, we can immediately obtain a 900×900 Hamming distance circuit by fixing 248 of the 2048 input wires to 0. Because of the value propagation, this does not incur any significant evaluation cost. As another example, all the initial states (i.e., entries in the first row and the first column of the state matrix) in both the Levenshtein and Smith-Waterman protocols are filled with known values agreed upon by both parties. Hence, our circuit computes on known signals without any needing any encryption.

Exploiting Wire Labels. The wire labels obtained during a garbled circuit evaluation are normally treated as a worthless by-product of the evaluation, but can be used in subsequent computations. In the garbled circuit evaluator's perspective, the set of wire labels computed are meaningless numbers, conveying no semantic information until the last step. This property is bound to the rigorous definition of security for garbled circuit technique. We exploit this fact to avoid garbled execution for many binary gates, in two main ways:

- **Backtracking** We used the wire labels in conjunction with permuted data structures to perform additional computation without leaking any information. For example, the wire exiting each comparison sub-circuit in a tree-structured circuit for determining the minimum value of a large set encodes the information about each pairwise comparison. Thus, the wire labels obtained by the evaluator can be used to very efficiently evaluate a *backtracking tree* created by the generator to obliviously retrieve profile information associated with the minimum value found. In essence, the labels are used as encryption keys that are combined to reveal profile information. We use this technique in our fingerprint matching protocol [29] to obtain the identity record associated with the closest matching fingerprint.
- **Symbolic Execution** Recall that plain signals, occasionally or frequently, can appear in our hybrid circuit (especially when circuit executions of plain and garbled signals are combined). In addition, since wire labels are unique, we can treat them as ordinary distinct symbols. This insight allows us to do binary gate-level *symbolic execution* which is *free* compared to garbled execution. When combined in a large circuit, these gate-level simplfications can collapse many binary gates to

simple wire connections. For example, the initialization in Levenshtein and Smith-Waterman algorithms specifies that a fraction of wire ports (corresponding to the values of the first row and the first column) are bound to known signals, a fact that can be exploited to eliminate many gates. Zahur et al. describes this technique in more detail, as well as further opportunities for using higher-level symbolic execution techniques to speedup privacy-preserving applications when some data can be revealed [54].

3.3 Implementation

Our framework is designed to enable programmers to define secure computations using a high-level language while providing enough control over the circuit design to enable efficient implementation. Users of our framework write a combination of high-level (Java) code and code for constructing circuits. Users need not be cryptographic experts, but are expected to be familiar with digital circuit design. Our framework and applications are available under an open source license from http://www.MightBeEvil.com.

Our framework core is about 1500 lines of Java code. The utility circuits comprise an additional 700 lines for efficient implementations of common circuits (adders, muxers, comparators, etc.). The small size of the framework enables it to be reasonably verified to provide users with good confidence in the integrity of the protocol software implementation (although we have not yet attempted any formal verification of properties of the framework implementation). Since both parties involved in a privacy-preserving computation need to fully trust their software for protecting their secrets, this small code base is essential in practice.

Figure 1 depicts a UML class diagram of the core classes of our framework. Concrete circuits are constructed using their build() method. The hierarchy of circuits is organized following the *Composite* design pattern [14] with respect to the build() method. Circuits are constructed in a highly modularized way, using Wire objects to connect them all together. The relation between Wire and Circuit follows a variation of the *Observer* pattern (a kind of publish-subscribe) [14]. The main difference is that, when a wire w is *connected* to a circuit on a *port* p (represented as a position index to the inputWires array of the circuit), all the observers of the input port wire p are automatically become observers of w. Moreover, the wire-to-wire propagation is done once at circuit construction time (instead of circuit execution time), which yields about 15% speedup in our experiments.

Subclasses of the SimpleCircuit abstract class provide the functions commonly required by any binary gates such as 2-to-1 AND, OR, and XOR. The AND and OR gates are implemented using the standard garbled circuit technique, whereas the XOR gate is implemented using the free-XOR optimization [36]. Our framework automatically propagates known signals which saves the protocol run-time cost whenever any internal wires can be fixed to a known value. The binary circuits form the core of our framework. To build a new application, users only need to create new subclasses of CompositeCircuit.

To simplify building new composite circuits, the build() method of CompositeCircuit abstract class is designed with the *Factory Method* pattern [14]. The code below shows the general structure of the build() method:

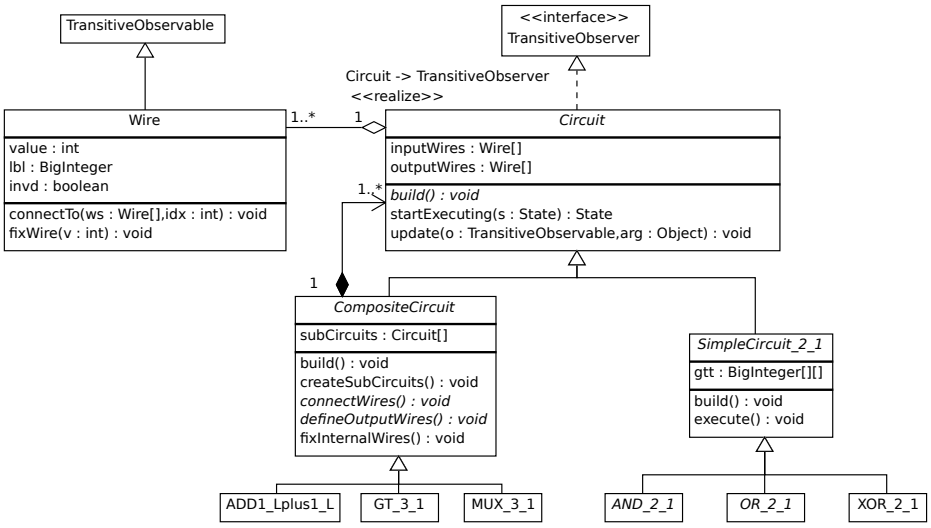

Fig. 1: Core Classes in Framework.

```
public void build() throws Exception {
    createInputWires();
    createSubCircuits();
    connectWires();
    defineOutputWires();
    fixInternalWires();
}
```

To define a new circuit, a user creates a new subclass of CompositeCircuit that overrides the createSubCircuits(), connectWires(), and defineOutputWires() methods to define a new circuit. In cases where internal wires have known values (e.g., the carry-in port of an adder is fixed to 0), better performance is obtained if the user also overrides the fixInternalWires() method.

3.4 Applications

We built several target applications using our garbled circuits framework to evaluate its scalability and efficiency. Table 1 summarizes the results.

In *privacy-preserving fingerprint matching* [29], a client has a scanned candidate fingerprint and the server has a database of fingerprint images with associated profile information. The system does not reveal any information about the candidate fingerprint to the server, or about the database to the client, except the identity of the closest match if there is a match within some threshold distance (or the non-existence of any close match). We designed a bit-width minimizing circuit for finding the minimum difference of the Euclidean distances (removing the random vector added in the homomorphic encryption phase), and used a backtracking strategy to obliviously obtain the associated profile information. Our fingerprint matching protocol combines an additive

Application	Best Previous	Measurement	Results	Our Results	Speedup
Fingerprint Matching	Barni et al. [2]	Closest Threshold Match[a]	16s	1.5s	10.6[a]
Face Recognition	SCiFI [47]	900-bit Hamming, Online	0.310s	0.019s	16.3
		900-bit Hamming, Total	213s	0.05s	4176
Levenshtein Distance	Jha et al. [32]	100×100[b]	92.4s	4.1s	22.4
		200×200[c]	534s	18.4s	29.0
Smith-Waterman	Jha et al. [32]	60×60	*d*	447s	*d*
AES Encryption	Henecka et al. [26]	Online Time (per block)	0.4s	0.008s	50
		Total Time (per block)	3.3s	0.2s	16.5

Table 1: Performance results for privacy-preserving applications.

All results are for 80-bit wire labels and the security parameters for the extended OT protocol [30] are $(80, 80)$. Our results are the average of 100 trials with the client and server each running on a Intel Core Duo E8400 3GHz; the comparisons are with results reported in the cited papers, using similar, but not identical, machines. *a*. Unlike our work, Barni et al. [2] cannot find the best match but instead identifies all entries within some threshold. *b*. Protocol 1, a garbled-circuit only implementation that is faster than Protocol 3, but does not scale to 200×200. *c*. Protocol 3, a hybrid protocol (the flagship protocol of [32]). *d*. No meaningful comparison is possible here, although our protocol is about twice as fast, since [32] implemented a simplified Smith-Waterman protocol [28].

homomorphic encryption phase used to compute Euclidean distances between fingerprint vectors, and a garbled circuit phase for finding the closest match within ε. For the other applications, we use only garbled circuit techniques.

The main operation in the *privacy-preserving face recognition* application is computing the Hamming distance between bit vectors representing face characteristics. Osadchy et al.'s results which use a combination of homomorphic encryption and 1-out-of-*n* oblivious transfer, but are far slower than our generic garbled circuit approach. The *Levenshtein Distance* (edit distance) and *Smith-Waterman* (genome alignment) applications use a dynamic programming algorithm. In the privacy-preserving setting, each party has one of the input strings and they wish to compute the distance between the two strings without revealing anything else about the input strings. In considering these problems, Jha et al. concluded that garbled circuits could not because of the memory blowup as the circuit size increases [32]. Our implementation is able to complete a $2000 \times 10{,}000$ Levenshtein distance problem on commodity PCs, evaluating more than 1.29 billion non-free gates in 223 minutes.

We also demonstrated generic garbled circuit based privacy-preserving applications are even possible for mobile platforms, where memory and computing resources are much more constrained [27].

4 Stronger Adversaries

The standard garbled circuits protocol, as implemented by the framework described in the previous section, is secure against semi-honest adversaries who are required to follow the protocol as specified but attempt to learn additional information about private inputs from the protocol execution. Although the semi-honest model is very weak, it is appropriate for some realistic scenarios such as when the result (including knowledge of whether or not the protocol completed successfully) is only made visible to the circuit evaluator and is not revealed to the circuit generator. In these scenarios, garbled circuit protocols can provide strong privacy (but not correctness) guarantees even against an arbitrary adversary so long as an oblivious transfer protocol that resists malicious adversaries is used. When correctness guarantees are also needed, or when confidentiality must be maintained even though the generator received the final output, the standard garbled circuit protocol is not sufficient. Instead, a protocol that tolerates malicious adversaries is required.

Section 4.1 discusses the possible attacks in the malicious model, and Section 4.2 surveys work on transforming semi-honest protocols to resist malicious adversaries. We describe our approach to constructing garbled circuit protocols that provably tolerate malicious adversaries in Section 4.3, and we compare the asymptotic complexity of our work and previous solutions in Section 4.4. Implementing protocols that can resist stronger adversaries remains much more expensive than protocols in the semi-honest model, but with improvements in both the underlying protocol implementations and approaches for strengthening semi-honest protocols, secure computations that are secure against strong adversaries are now feasible in practice for some problems.

4.1 Threats

The malicious model allows a protocol participant to deviate from the agreed protocol in arbitrary ways. Such an adversary may compromise the privacy of the other participant's data, or tamper with the correctness of the result.

A malicious generator might construct a *faulty circuit* that discloses the evaluator's private input. For example, instead of producing a circuit that computes the agreed upon function $f(a,b)$, a malicious generator could secretly construct a garbled circuit that computes $f'(a,b) \mapsto b$, thus learning the second participant's input directly.

A more subtle attack is *selective failure* [34, 42]. In this attack, a malicious generator uses inconsistent labels to construct the garbled gate and OT so that the evaluator's input can be inferred from whether or not the protocol completes. For example, a cheating generator may assign (w^0, w^1) to an input wire in the garbled circuit while using $(w^0, \hat{w}^1)$ instead in the corresponding OT where $w^1 \neq \hat{w}^1$. Consequently, if the evaluator's input is 0, she will get w^0 from OT and complete the evaluation without noticing any anomaly. In contrast, if her input is 1, she will get $\hat{w}^1$ and be unable to complete the evaluation properly. If the protocol expects the evaluator to share the result with the generator at the end, the generator learns if the evaluation failed and information about the evaluator's input is leaked.

4.2 Previous Work

Many approaches have been proposed to transform a garbled circuit protocol that provides security in the semi-honest model into a protocol that provides security guarantees in the malicious model. The two main approaches are *cut-and-choose* and *commit-and-prove*.

Cut-and-choose. One approach is to enforce honest behavior from malicious adversaries is for the generator to prepare multiple copies of the garbled circuit with independent randomness, and the evaluator randomly chooses a fraction of the circuits, whose randomness is then revealed. The evaluator aborts if any of the chosen circuits (called *check-circuits*) are inconsistent with the revealed randomness. Otherwise, she evaluates the remaining circuits (called *evaluation-circuits*) and takes the majority of the output from evaluation-circuits as the final output.

The intuition is that in order to pass the check, a malicious generator will need to keep the number of faulty circuits low, and the minority faulty circuits will be fixed by the majority operation in the end. In other words, if a malicious generator wants to manipulate the final output, she needs to construct faulty majority among evaluation-circuits, and then the chance that none of the faulty circuits is checked will be negligible.

The cut-and-choose technique requires that the evaluator has a way to ensure that the generator provides the same input for each evaluation circuit. Recall that the generator also sends multiple copies of her input labels to the evaluator. A malicious generator may provide altered inputs to different evaluation-circuits, and it has been shown that for some functions, there are simple ways for the generator to extract information about the evaluator's input [38]. For example, suppose both parties agree to compute the inner-product of their input, that is, $f(a,b) \mapsto \sum_{i=1}^{n} a_i b_i$, where a_i and b_i is the generator's and evaluator's i-th input bit, respectively. Instead of providing $[a_1,\ldots,a_n]$ to all evaluation-circuits, the generator might send $[d_1^j,\ldots,d_n^j]$ to the j-th copy of the evaluation-circuits where $d_j^j = 1$, and $d_i^j = 0$ if $i \neq j$. The malicious generator then learns the majority bit in the evaluator's input, which is not what the evaluator agreed to reveal in advance. As a result, we must ensure *the generator's input consistency.*

Mohassel and Franklin [42] proposed the *equality-checker* scheme, which needs $O(ns^2)$ commitments to be computed and exchanged to ensure the generator's input consistency, where n is the input size and s is a statistical security parameter that is the number of copies of the garbled circuit. Lindell and Pinkas [38] develop an elegant cut-and-choose based construction that enjoys the simulation-based security against malicious players. This approach requires $O(ns^2)$ commitments to be computed and exchanged between the participants. Although these commitments can be implemented using lightweight primitives such as collision-resistant hash functions, communication complexity is still an issue. Nielsen and Orlandi [45] proposed an approach with Lego-like garbled gates. Although it is also based on the cut-and-choose method, via an alignment technique only a single copy of the generator's input keys is needed for all the evaluation-circuits. However, each gate needs several group elements as commitments resulting both computational and communicational overhead. Lindell and Pinkas propose a Diffie-Hellman pseudorandom synthesizer technique [39]. Their approach relies

on finding efficient zero-knowledge proofs for specifically chosen complexity assumptions, which has complexity $O(ns)$.

In summary, to enforce honest behavior from malicious adversaries based on the cut-and-choose technique, we need to deter the faulty circuit, selective failure, and the generator's input inconsistency attacks.

Commit-and-prove. Another well-known category to enforce honest behavior is called *commit-and-prove*. This approach is first suggested by Goldreich, Micali, and Widgerson [16], and only requires the weak general assumption of zero-knowledge proofs of knowledge. However, it has never been implemented since it requires costly NP-reductions.

Following the same idea, Jarecki and Shmatikov [31] presented an approach, in which the generator is asked to prove the correctness of the garbled circuit in zero knowledge before the evaluation starts. Although only one copy of the garbled circuit is constructed, their protocol requires hundreds of heavy cryptographic operations *per gate*, whereas approaches based on the cut-and-choose method require only such expensive operations for the *input gates*.

Recently, Nielsen et al. proposed a solution based on a variation of the commit-and-prove approach [46]. They extended the GMW [16] protocol with a technique called *authenticated bits* that have the property that only if the participants follow the agreed protocol do the results remain authenticated. In other words, if a malicious participant deviates from the agreed protocol, the other party will notice and then abort. Therefore, if the final result remains authenticated, it constitutes a proof that both parties behaved honestly. Although the GMW protocol requires many expensive OTs, Neilson et al. manage to conquer this issue with OT extensions, and thus, their solution has good amortized efficiency.

4.3 Our Approach

We tackle the selective failure attack by using a stronger notion of OT called *committing OT* [34]. This notion requires that in addition to getting exactly one message of her choice, the receiver of the OT (the evaluator of the garbled circuit protocol) also gets the commitments to *both* of the sender's messages. Later, the sender of the OT can *post-facto* prove that she ran the OT correctly by revealing the randomness used in the OT only for those OT instances corresponding to circuits that are opened for verification.

We solve the input consistency problem in an efficient manner by designing a way to use weak witness indistinguishable proofs instead of zero-knowledge protocols (or Σ-protocols). A witness-indistinguishable proof only makes the guarantee that the verifier cannot learn which witness the prover used during a proof. We design a special witness-indistinguishable proof for an operation concerning *claw-free functions* that have a weak malleability property to generate efficient instantiations of input consistency proofs. Shelat and Shen provide details on the full protocol and its security proof [50].

We note that both the committed-input scheme [42] and Diffie-Hellman pseudorandom synthesizer technique [39] are special cases of our approach, and thus, have similar complexity. However, the committed-input scheme is not known to enjoy simulation-

based security, and the pseudorandom synthesizer technique requires zero-knowledge proofs that are unnecessary in this case. Our approach is faster than these works by a constant factor.

4.4 Communication Complexity

To understand the costs of hardening a garbled circuit protocol against malicious adversaries, we compare the communication efficiency between protocols that use a mix of light cryptographic primitives (such as commitments instantiated with collision-resistant hash functions) and heavy ones (such as group operations that rely on algebraic assumptions like discrete logarithm). We consider asymptotic complexity under reasonable assumptions about the growth of various primitives with respect to the security parameter k:

1. light cryptographic primitives have size $\Theta(k)$;
2. heavy cryptographic operations, like elliptic curve operations, have size $\tilde{o}(k^2)$; and
3. heavy cryptographic operations, like RSA or group operations over $\mathbb{Z}$, have size $\tilde{o}(k^3)$.

We make the assumption since in certain elliptic curve groups, known methods for computing discrete logarithms of size n run in time $L_n(1, \frac{1}{2})$. Thus, to achieve security of 2^k, it suffices to use operands of size $\tilde{o}(k^2)$, by which we mean a value that is asymptotically smaller than k^2 by factors of $\log(k)$.

Table 2 summarizes our asymptotic analysis. Let k be a security parameter and s be a statistical security parameter, and let $|C|$ be the number of gates in the base circuit. The other protocols are:

– Jarecki and Shmatikov [31]: This is a commit-and-prove approach, so it does not need to ensure input consistency. However, it requires hundreds of group operations per gate in order to defend faulty circuit and selective failure attacks (with ZK proofs). Since this protocol assumes the decisional composite residuosity problem in an RSA group, each group element is of size $\tilde{o}(k^3)$.
– Kiraz [33]: This approach is based on the cut-and-choose technique. So, it uses s copies of the garbled circuit, each circuit has $|C|$ gates, and each gate needs $O(k)$

	Communication		
	Base Circuit	Generator's Input	Evaluator's Input
JS07 [31]	$\|C\| \cdot \tilde{o}(k^3)$	–	n (committed) OT's
Ki08 [33]	$\Theta(\|C\| \cdot sk)$	$\Theta(ns^2k)$	n (committing) OT's
LP07 [38]	$\Theta(\|C\| \cdot sk)$	$\Theta(ns^2k)$	$\max(4n, 8s)$ OT's
LP10 [39]	$\Theta(\|C\| \cdot sk)$	$\Theta(ns) \cdot \tilde{o}(k^2)$	n OT's
Our work [50]	$\Theta(\|C\| \cdot sk)$	$\Theta(ns) \cdot \tilde{o}(k^2)$	n (committing) OT's

Table 2: Analysis of two-party secure computation against malicious adversaries.

for garbled truth table. Also, they use an equality-checker framework that requires $O(ns^2)$ commitments to enforce the generator's input consistency. As with our approach, they thwart the selective failure attack by using committing OTs.
- Lindell and Pinkas [38]: This is also a cut-and-choose-based approach. Each of the generator's input bits requires $O(s^2)$ light commitment for the consistency check. To defend against the selective failure attack, it requires $\max(4n, 8s)$ OT's.
- Lindell and Pinkas [39]: This approach is similar to ours in finding a good balance between approaches using many but lightweight primitives and approaches using a few expensive group operations. This uses a technique called cut-and-choose OT to handle the generator's input inconsistency and selective failure attacks. However, they rely on more specific cryptographic assumptions, and the unnecessary zero-knowledge proofs incur constant factor overhead.

5 Fairness

Fairness is the property that either *all parties* receive the output, or *no one* does.[3] None of the protocols we have described so far provide any fairness properties. Fairness is desirable in many circumstances:

- *Coin-tossing* protocols can be used to generate an unbiased coin. This can be viewed as secure computation of a probabilistic functionality taking no inputs. If fairness is not guaranteed, then one party might learn the value of the coin first and then decide whether to abort the protocol based on the coin's value.
- In a protocol for *exchanging digital goods* (such as using digital currency to purchase a song) it would be unacceptable for the buyer to obtain the digital good without also making the payment (or for the seller to receive the payment without providing the digital good). Similarly, in *exchange of digital signatures* it is considered undesirable for one party to obtain the second party's signature on a contract without the second party simultaneously getting a copy of the first party's signature on the same contract.

More generally, we can imagine any scenario where learning the output — while preventing the other party from learning the output — provides a competitive advantage. Without a guarantee of fairness, parties in such a situation may be unwilling to even participate in the protocol.

In fact, fairness *can* be ensured in the multi-party setting in the case when a majority of the parties are guaranteed to remain honest [3, 16, 49]. (This assumes a broadcast channel, or a mechanism such as a PKI that allows broadcast to be implemented.) In case no honest majority can be assumed, it may seem obvious that fairness cannot be ensured by the following argument (specialized to the setting of two-party computation): as the parties alternate sending messages of the protocol, surely one party must learn their output first. If that party aborts immediately upon learning its output, then the other party clearly does not learn its output.

[3] We assume for simplicity in our discussion that all parties are *supposed* to receive the same output, but everything we say generalizes to the case where different parties are supposed to receive different outputs.

Cleve [11] formalized this intuition, and showed that fair coin tossing is impossible in both the two-party and multi-party settings (when an honest majority is not assumed). This implies an analogous impossibility result for fair computation of the 1-bit XOR function (since fair computation of XOR would immediately imply fair coin tossing).

Thus, secure computation without an honest majority (for malicious adversaries) is typically defined relative to an ideal world in which fairness is not ensured at all. Specifically, the ideal world is taken to be one in which the parties send their inputs to a trusted entity who computes the result *and sends it back to the adversary only*; the adversary then sends either abort or continue to the trusted party. If she sends abort, the honest parties receive nothing from the trusted party, while in the second case the trusted party sends the honest parties the correct result.

Faced with Cleve's impossibility result, researchers have explored several ways to obtain some form of fairness:

1. Cleve's impossibility result rules out fairness for one specific function, but does not rule out fairness for *every* function. Might there be any non-trivial functions for which fairness is possible? For over twenty years after Cleve's result the answer was assumed to be "no," especially given how convincing the informal argument against fairness seemed to be. It therefore came as somewhat of a surprise when it was recently shown that there *do* exist non-trivial functions for which fairness is possible. Section 5.1 surveys this work.
2. The standard definition of secure computation without an honest majority gives up on fairness entirely. Instead, it seems preferable to define some notion of *partial fairness* and design protocols achieving at least that. Section 5.2 discusses several different notions of partial fairness.
3. Cleve's impossibility result holds in the usual cryptographic model where the adversary may behave in an arbitrarily malicious way. In some settings, however, it may be reasonable to assume a *rational* adversary whose cheating is motivated by some explicit utility function that the adversary is trying to maximize. Section 5.3 describes work on protocols that are designed to provide fairness against a rational, but not malicious, adversary.

5.1 Complete Fairness for Specific Functions

Cleve showed one function for which fair secure two-party computation is impossible without an honest majority, but did not show that fairness is impossible for *all* functions. Indeed, functions that depend on only one of the parties' inputs can be computed with complete fairness. This class includes some interesting functionalities — *zero-knowledge* among them — but still seems to miss the main difficulty of fairness in the first place. Thus, the question becomes whether there are any functions that can be computed with complete fairness that depend on more than one party's inputs. The answer, surprisingly, turns out to be *yes*. This has been shown in both the two-party [19] and multi-party [21] settings.

Instead of presenting the technical details of the protocols here (see [19, 21]), we explain informally how the two types of protocols shown by Gordon et al. [19] in the two-party setting manage to circumvent the convincing intuition that "one party must

learn its output first, and can abort immediately after doing so to prevent the other party from learning its output". In the first type of protocol presented in [19], *the round in which a party learns its output depends on that party's input*. (This is in contrast to standard protocols for secure computation where the output is always learned, by both parties, in some fixed round.) In particular, simplifying slightly, if we number the parties' possible inputs $x_1, \ldots, x_\ell$ then a party holding input x_i learns the output in round i. Thus, depending on the parties' respective inputs, either party might learn the output first. Moreover, on an intuitive level, an abort by a party (say, P_1) in round i can be viewed as a "signal" to the other party P_2 that P_1's input was in fact x_i; thus, even after an abort by P_1, party P_2 can still compute the function using x_i as P_1's input. That this works is not immediate: for one thing, a malicious P_1 can try to "fool" P_2 by aborting in round $i+1$, say. Nevertheless, it can be shown that this protocol does achieve complete fairness for certain carefully constructed functions.

In the second type of protocol, parties also do not learn their outputs in some fixed round. Instead, the round in which the output is revealed is chosen according to a geometric distribution; moreover, the parties do not learn definitively in which round the output is revealed until the end of the protocol. (Slightly more formally, by the end of the protocol they are guaranteed that, with all but negligible probability, they hold the correct output.) Probabilities are balanced in such a way that even if one party aborts early, and thus "knows more information" about the correct output than the other party does, it doesn't *know* how to use this extra knowledge to violate fairness. (The formal proof of security shows that an aborting adversary can "learn the same information" in an ideal world where early abort is not possible, possibly by changing its input.)

What is especially interesting about both protocols described above, is that the proofs of fairness in each case boil down to information-theoretic arguments that do not rely on cryptography. It is thus natural to conjecture that development of the right information-theoretic tools will enable better analysis of fairness, and might help to resolve the main open question that remains: to characterize those functions for which complete fairness is possible.

5.2 Partial Fairness

Even given the work described in the previous section, we know that for certain functions complete fairness is simply not possible. The standard approach is to give up on fairness altogether. An better alternative is to instead define some notion of *partial* fairness and attempt to design protocols that achieve that weaker notion.

There are at least two general approaches to partial fairness that date back to the early 1980s (see [18, 22] for more detailed discussion). In one approach, a protocol is constructed such that at every round both parties can recover their output using a "similar" amount of work [12, 13, 15]. An unsatisfying feature of this approach is that the decision of whether an honest party should invest the necessary work to recover the output is *not* specified as part of the protocol but is instead decided "externally". Such protocols raise the risk of denial-of-service attacks by an adversary who aborts the protocol early (if that would cause the honest party to then invest a significant amount of work to recover the answer). Protocols of this sort also seem to require very strong cryptographic assumptions.

A second approach [4, 17] can be viewed as designing a protocol in which both parties gradually increase their "confidence" in the output (for example, by learning an independent noisy version of the output in each round). It seems difficult to extend such protocols to multi-bit outputs, or the case where parties are supposed to learn different outputs. More problematic is that such protocols allow the adversary to significantly bias the output of the honest party, thus violating correctness.

More recently, Gordon and Katz suggested another definitional approach that has the advantage of remaining within the simulation-based framework of standard security definitions [22]. The idea is to define partial fairness using the *same* ideal-world model used to define complete fairness. However, rather than require that the real world and ideal world be completely (computationally) indistinguishable, partial fairness is instead defined by allowing the real and ideal worlds to be distinguishable by at most $1/p$, for an arbitrary polynomial p. Such a protocol is called $1/p$-*secure*. The way to think about this is that a $1/p$-secure protocol is secure up to a (possible) $1/p$ "defect". In particular, fairness is guaranteed to hold except with probability (at most) $1/p$.

Moran et al. showed how to construct a protocol for $1/p$-secure coin tossing [43]. Gordon and Katz [22] showed how to construct $1/p$-secure protocols in the two-party setting for any function with polynomial-size domain or range. They also proved a general impossibility result for functions without one of these requirements. Both of these works have since been extended to the multi-party setting [6, 7].

5.3 Fairness with Rational Parties

Another approach for dealing with fairness is to explicitly model the adversary as *rational*, rather than arbitrary malicious as in most work in cryptography. In the context of fairness, it is most natural to consider an adversary with the following utilities (specialized to the two-party case):

- The adversary prefers to learn the (correct) output of the function above all else.
- Assuming it learns the output of the function, the adversary prefers that the other party does *not* learn the output.

Protocols that remain fair in the presence of adversaries with the above utilities were first considered in the context of *rational secret sharing* [20, 24, 40]. More recently, the model has been extended to fair secure two-party computation of general functionalities. Asharov et al. [1] propose several equivalent definitions of the problem and show a negative result, giving a function that cannot be computed fairly even if a rational adversary is assumed. Subsequently, Groce and Katz [23] showed broad *positive* results for this setting along with a partial characterization of when rational fair computation is possible.

6 Conclusion

General secure computation techniques offer the promise of strong privacy guarantees without the the need for a trusted third party. Until recently, however, such techniques were largely viewed as a theoretical curiosity because of the high cost of implementing

them for real applications. Recent advances in both the theory and implementation of generic garbled circuit protocols, however, make large-scale privacy-preserving applications a realistic possibility. Many challenges remain, especially to provide efficient solutions against stronger adversaries and to provide fairness guarantees when needed, but the promise secure computation offers to perform computation with private data without compromising that data appears to be in reach.

Acknowledgments

The work described in this paper was partly supported by grants from the National Science Foundation, DARPA, and a MURI award from the Air Force Office of Scientific Research. The contents of this paper do not necessarily reflect the position or the policy of the US Government, and no official endorsement should be inferred. The authors thank Peter Chapman, Jiamin Chen, Yikan Chen, Michael Hicks, Sang Koo, Benjamin Kreuter, Aaron Mackey, Steven Myers, Mona Sergi, and Samee Zahur for their contributions to this project.

References

1. Asharov, G., Canetti, R., Hazay, C.: Towards a Game Theoretic View of Secure Computation. In: Eurocrypt (2011)
2. Barni, M., Bianchi, T., Catalano, D., Raimondo, M.D., Labati, R.D., Faillia, P., Fiore, D., Lazzeretti, R., Piuri, V., Scotti, F., Piva, A.: Privacy-preserving Fingercode Authentication. In: ACM Multimedia and Security Workshop (2010)
3. Beaver, D.: Secure Multiparty Protocols and Zero-Knowledge Proof Systems Tolerating a Faulty Minority. Journal of Cryptology (1991)
4. Beaver, D., Goldwasser, S.: Multiparty Computation with Faulty Majority. In: 30th Symposium on Foundations of Computer Science (1989)
5. Beaver, D., Micali, S., Rogaway, P.: The Round Complexity of Secure Protocols. In: ACM Symposium on Theory of Computing (1990)
6. Beimel, A., Lindell, Y., Omri, E., Orlov, I.: $1/p$-secure Multiparty Computation without Honest Majority and the Best of Both Worlds. In: Crypto (2011)
7. Beimel, A., Omri, E., Orlov, I.: Protocols for Multiparty Coin Toss with Dishonest Majority. In: Crypto (2010)
8. Ben-David, A., Nisan, N., Pinkas, B.: FairplayMP: A System for Secure Multi-party Computation. In: ACM Conference on Computer and Communications Security (2008)
9. Brickell, J., Shmatikov, V.: Privacy-preserving Graph Algorithms in the Semi-honest Model. In: Asiacrypt (2005)
10. Choi, S.G., Katz, J., Kumaresan, R., Zhou, H.S.: On the Security of the "Free-XOR" Technique. http://eprint.iacr.org/2011/510 (2011)
11. Cleve, R.: Limits on the Security of Coin Flips when Half the Processors Are Faulty. In: 18th Symposium on Theory of Computing (1986)
12. Even, S., Goldreich, O., Lempel, A.: A Randomized Protocol for Signing Contracts. Communications of the ACM (1985)
13. Galil, Z., Haber, S., Yung, M.: Cryptographic Computation: Secure Fault-Tolerant Protocols and the Public-Key Model. In: Crypto (1988)

14. Gamma, E., Helm, R., Johnson, R.E., Vlissides, J.: Design Patterns — Elements of Reusable Object-Oriented Software. Addison-Wesley (March 1995)
15. Garay, J.A., MacKenzie, P.D., Prabhakaran, M., Yang, K.: Resource Fairness and Composability of Cryptographic Protocols. In: 3rd Theory of Cryptography Conference (2006)
16. Goldreich, O., Micali, S., Wigderson, A.: How to Play Any Mental Game, or a Completeness Theorem for Protocols with Honest Majority. In: 19th Symposium on Theory of Computing (1987)
17. Goldwasser, S., Levin, L.A.: Fair Computation of General Functions in Presence of Immoral Majority. In: Crypto (1991)
18. Gordon, S.D.: Fairness in Secure Computation. Ph.D. thesis, University of Maryland (2010)
19. Gordon, S.D., Hazay, C., Katz, J., Lindell, Y.: Complete Fairness in Secure Two-Party Computation. In: 40th Symposium on Theory of Computing (2008)
20. Gordon, S.D., Katz, J.: Rational Secret Sharing, Revisited. In: 5th International Conference on Security and Cryptography for Networks (2006)
21. Gordon, S.D., Katz, J.: Complete Fairness in Multi-Party Computation without an Honest Majority. In: 6th Theory of Cryptography Conference (2009)
22. Gordon, S.D., Katz, J.: Partial Fairness in Secure Two-Party Computation. In: Eurocrypt (2010)
23. Groce, A., Katz, J.: Fair Computation with Rational Players. http://eprint.iacr.org/2011/396 (2011)
24. Halpern, J., Teague, V.: Rational Secret Sharing and Multiparty Computation. In: 36th Symposium on Theory of Computing (2004)
25. Hazay, C., Lindell, Y.: Efficient Protocols for Set Intersection and Pattern Matching with Security Against Malicious and Covert Adversaries. In: Theory of Cryptography Conference (2008)
26. Henecka, W., K ogl, S., Sadeghi, A.R., Schneider, T., Wehrenberg, I.: TASTY: Tool for Automating Secure Two-partY computations. In: ACM Conference on Computer and Communications Security (2010)
27. Huang, Y., Chapman, P., Evans, D.: Privacy-Preserving Applications on Smartphones. In: USENIX Workshop on Hot Topics in Security (2011)
28. Huang, Y., Evans, D., Katz, J., Malka, L.: Faster Secure Two-Party Computation Using Garbled Circuits. In: USENIX Security Symposium (2011)
29. Huang, Y., Malka, L., Evans, D., Katz, J.: Efficient Privacy-Preserving Biometric Identification. In: Network and Distributed System Security Symposium (2011)
30. Ishai, Y., Kilian, J., Nissim, K., Petrank, E.: Extending Oblivious Transfers Efficiently. In: Crypto (2003)
31. Jarecki, S., Shmatikov, V.: Efficient Two-Party Secure Computation on Committed Inputs. In: Eurocrypt (2007)
32. Jha, S., Kruger, L., Shmatikov, V.: Towards Practical Privacy for Genomic Computation. In: IEEE Symposium on Security and Privacy (2008)
33. Kiraz, M.: Secure and Fair Two-Party Computation. Ph.D. thesis, Technische Universiteit Eindhoven (2008)
34. Kiraz, M., Schoenmakers, B.: A Protocol Issue for The Malicious Case of Yao's Garbled Circuit Construction. In: 27th Symposium on Information Theory in the Benelux (2006)
35. Kolesnikov, V., Sadeghi, A.R., Schneider, T.: Improved Garbled Circuit Building Blocks and Applications to Auctions and Computing Minima. In: International Conference on Cryptology and Network Security (2009)
36. Kolesnikov, V., Schneider, T.: Improved Garbled Circuit: Free XOR Gates and Applications. In: International Colloquium on Automata, Languages and Programming (2008)
37. Lindell, Y., Pinkas, B.: Privacy Preserving Data Mining. Journal of Cryptology 15(3) (2002)

38. Lindell, Y., Pinkas, B.: An Efficient Protocol for Secure Two-Party Computation in the Presence of Malicious Adversaries. In: Eurocrypt (2007)
39. Lindell, Y., Pinkas, B.: Secure Two-Party Computation Via Cut-and-Choose Oblivious Transfer. Crypto ePrint Archive (2010), http://eprint.iacr.org/2010/284
40. Lysyanskaya, A., Triandopoulos, N.: Rationality and Adversarial Behavior in Multi-Party Computation. In: Crypto (2006)
41. Malkhi, D., Nisan, N., Pinkas, B., Sella, Y.: Fairplay — A Secure Two-Party Computation System. In: USENIX Security Symposium (2004)
42. Mohassel, P., Franklin, M.: Efficiency Tradeoffs for Malicious Two-Party Computation. In: Public Key Cryptography (2006)
43. Moran, T., Naor, M., Segev, G.: An Optimally Fair Coin Toss. In: 6th Theory of Cryptography Conference (2009)
44. Naor, M., Pinkas, B.: Efficient Oblivious Transfer Protocols. In: ACM-SIAM Symposium on Discrete Algorithms (2001)
45. Nielsen, J., Orlandi, C.: LEGO for Two-Party Secure Computation. In: Reingold, O. (ed.) TCC 2009. LNCS, vol. 5444, pp. 368–386. Springer (2009)
46. Nielsen, J.B., Nordholt, P.S., Orlandi, C., Burra, S.S.: A New Approach to Practical Active-Secure Two-Party Computation. Crypto ePrint Archive (2011), http://eprint.iacr.org/2011/091
47. Osadchy, M., Pinkas, B., Jarrous, A., Moskovich, B.: SCiFI: A System for Secure Face Identification. In: IEEE Symposium on Security and Privacy (2010)
48. Pinkas, B., Schneider, T., Smart, N.P., Williams, S.C.: Secure Two-Party Computation Is Practical. In: Asiacrypt (2009)
49. Rabin, T., Ben-Or, M.: Verifiable Secret Sharing and Multiparty Protocols with Honest Majority. In: 21st Symposium on Theory of Computing (1989)
50. shelat, a., Shen, C.h.: Two-Output Secure Computation with Malicious Adversaries. In: Eurocrypt (2011)
51. Yang, Z., Zhong, S., Wright, R.: Privacy-preserving Classification of Customer Data without Loss of Accuracy. In: SIAM International Conference on Data Mining (2005)
52. Yao, A.C.: Protocols for Secure Computations. In: Symposium on Foundations of Computer Science (1982)
53. Yao, A.C.: How to Generate and Exchange Secrets. In: Symposium on Foundations of Computer Science (1986)
54. Zahur, S., Huang, Y., Evans, D.: Efficient Secure Computation over Partially-Secret Inputs. Unpublished manuscript (2011)