Fair Secure Two-Party Computation
Extended Abstract

Benny Pinkas*

HP Labs

Abstract. We demonstrate a transformation of Yao's protocol for secure two-party computation to a fair protocol in which neither party gains any substantial advantage by terminating the protocol prematurely. The transformation adds additional steps before and after the execution of the original protocol, but does not change it otherwise, and does not use a trusted third party. It is based on the use of gradual release timed commitments, which are a new variant of timed commitments, and on a novel use of blind signatures for verifying that the committed values are correct.

1 Introduction

Yao [30] introduced the concept of secure computation, and demonstrated a protocol that enables two parties to compute any function of their respective inputs without leaking to each other any additional information. Yao's protocol is both generic and efficient. The protocol is applied to a circuit that evaluates the function, and its overhead is linear in the size of the circuit (furthermore, the number of public key operations is only linear in the number of input bits). The actual running time of the protocol might be high if the circuit is very large, but it can be very reasonable if the circuit is of small to moderate size, as is the case for example with circuits that compute additions and comparisons (see e.g. [30, 24] for applications that use secure computation of small circuits). One of the disadvantages of this protocol, however, is that it does not guarantee fairness if both parties should learn an output of the protocol (be it the same output for both parties or a different output for each of them). Namely, one of the parties might learn her output before the other party does, and can then terminate the protocol prematurely, before the other party learns his output.

In Yao's protocol one party constructs a circuit that computes the function, and the other party evaluates the circuit. Let us denote the parties as the *Constructer* (known also as Charlie, or simply C), and the *Evaluator* (known also as Eve, or E). A malicious E can use the non-fairness "feature" and disguise the early termination of the protocol as a benign communication problem. The early termination gives E considerable illegitimate power. Consider for example the case where the parties run a protocol to decide on the terms of a sale of an item, owned by E, to C. E has a reserve price x, and C gives an offer y. The protocol

* Email: benny.pinkas@hp.com, benny@pinkas.net.

defines that if $y \geq x$ then C buys the item and pays $(x+y)/2$. Suppose that the parties run a secure protocol to compute the result of the protocol. The output of the protocol is the same for both parties, and is $(x+y)/2$ if $y \geq x$, or 0 otherwise. Now, if E is first to learn the result of the secure computation and if the result is higher than the reserve price x, she could terminate the communication between her and C. If C returns and tries to run the protocol again E changes her reserve price to be C's previous bid. This maximizes E's profit assuming that C uses the same bid. The protocol could be changed to enable C to learn his output before E does. In this case, though, a corrupt C could terminate the protocol once he learns his output, compute the value of E's reserve price, and rerun the protocol with a bid that is equal to the reserve price.

A possible approach for solving the fairness issue is for the output of the protocol to be *commitments* to E's and C's respective outputs. The commitment that becomes known to E is made using a key known to C, and vice versa. After exchanging the commitments the parties open them bit by bit, ensuring that none of them gains a significant advantage over the other party. This procedure can be performed by gradually revealing the bits of the keys that were used to generate the commitments, but this approach gives an unfair advantage to a more powerful party that can invest more resources in a parallel brute-force attack on the remaining bits of the key. This issue is solved by using the timed commitments of Boneh and Naor [8], or the subsequent work of Garay and Jakobsson [21]. In this paper we describe and use a new primitive, called a "gradual release timed commitment", which combines timed commitments with a gradual release of the commitment key (the gradual release property was not supported by [8, 21]).

The remaining problem: A protocol that uses gradual release timed commitments does not solve the fairness problem completely. Nothing ensures E that the commitment given to her by C is a commitment to the actual output of the circuit. A malicious C might as well provide E with a commitment to random data, and then follow the protocol and release the key that enables E to open this useless commitment. Likewise, E could take the same approach to cheat C. Previous solutions to the fairness problem (see Section 1.2 for details) solved this issue in a rather inefficient way: they changed the way each gate of the circuit is computed (and added costly proofs to gate evaluations), or changed the circuit so that it computed the result of a trapdoor function applied to one of the outputs. These constructions result in a fair protocol whose overhead is considerably higher than that of computing the original circuit. Our goal is to design a solution that does not add substantial overhead. This goal is important since Yao's basic protocol can be implemented rather efficiently if the circuit representation of the function is of reasonable size.

1.1 Our Contribution

The main technical focus of this work is the design of a "compiler" that transforms a secure two-party protocol into a fair and secure two-party protocol. The compiler solves the commitment verification problem, namely verifies that the commitments that are used at the last step of the protocol are indeed commitments to the actual outputs of the computed function. This can of course be guaranteed using zero-knowledge proofs, but this straightforward solution has considerable overhead. The protocol that is output by the compiler ensures the correctness of the commitments in an efficient way. It enjoys the following advantages:

– It does not require any third party to be involved.

– The computation of the function is performed by running the original two-party protocol. (More specifically, in order to reduce the probability of a fairness breach to be exponentially small in ℓ, it runs ℓ invocations of the circuit evaluation protocol.) This feature has two advantages: First, the resulting protocol is more efficient than previous suggestions for generic fair protocols, which required adding a proof to the computation of every gate of the circuit. Second, the construction can use any efficiency improvement to the original protocol (for example, use oblivious transfer protocols with low amortized complexity, as suggested in [28]).

On the other hand, our solution has the following disadvantages. First, the number of rounds in the final stage of the protocol is proportional to the security parameter of the commitment scheme (say, $k = 80$). This is essential for the gradual release of the committed values[1]. Second, the protocol requires the use of blind signatures. It is only known how to prove these to be secure assuming the use of a hash function that is modeled as a random function (i.e. in the random oracle model) [4]. The other tools we use do not depend on the random oracle assumption.

1.2 Related Work

The "fairness" problem was introduced by Blum [6]. A detailed discussion of early solutions appears in Franklin's dissertation [19]. Early work concentrated on fair coin-tossing and bit-exchange [25, 14, 15], and Damgard [17] describes an efficient protocol for gradually and verifiably releasing any secret. Constructions that achieve fairness for general two-party and multi-party protocols were introduced by Yao [30], Galil, Haber and Yung [20], Brickell et al. [10], Ben-Or et al. [5], Beaver and Goldwasser [2], and Goldwasser and Levin [24]. These constructions concentrate on generic properties rather than on efficiency. Most of these constructions apply a transformation to the computation of every gate rather than taking our approach of leaving the circuit evaluation as it is. The constructions of [30, 20] do not change the computation of every gate, but rather work by (1) the parties generating a trapdoor function, (2) the circuit computing an output encrypted by that function, and (3) the parties gradually revealing the bits of the trapdoor key. This solution is not very efficient due to overhead of secure distributed generation and computation of the trapdoor function. A

[1] The large number of rounds is not surprising given the lower bound of Cleve [15] for the number of rounds for fair exchange of secrets, and the lower bound of Boneh and Naor [8] for the number of rounds in fair contract signing.

more efficient construction was given in [9] for a specific two-party function — testing the equality of two inputs. That construction is tailored for the equality function. All the constructions require a final phase in which the outputs are revealed bit by bit in order to prevent one party from obtaining an unfair advantage. Note that unlike our protocol these constructions do not use the more recent timed commitments of Boneh and Naor [8], or the variant suggested by Garay and Jakobsson [21], which prevent a more powerful adversary from running a parallel attack for breaking the commitments.

An alternative approach for achieving fairness is to use an "optimistic" protocol which involves an offline trusted third party. If E sends the required information to C then this third party does not participate in the protocol (this is the optimistic case). However, if C does not receive her outputs from E she can contact the third party and obtain this information. The idea of using optimistic fair exchange for implementing generic secure protocols was suggested by [1, 27]. Cachin and Camenisch [11] designed an efficient optimistic fair protocol for this task, using efficient proofs of knowledge. The advantage of the optimistic approach is that the number of rounds is constant and does not depend on the security parameter. The disadvantages are the requirement for a third party, and a major degradation in the efficiency of the original protocol (even in the protocol of [11] public key operations and zero-knowledge proofs are required for every *gate* in the circuit).

2 Two-Party Protocols and the Fairness Issue

2.1 Fairness

We consider two-party protocols with the following parameters:
Input: E's input is x, C's input is y. There are also two public functions, F_E and F_C, with two inputs. To simplify the notation we define $F(x, y) = \langle F_E(x, y), F_C(x, y)\rangle$ and refer to F throughout the paper.
Output: E should learn $F_E(x, y)$ while C learns $F_C(x, y)$. (Of course, it might be that $F_E \equiv F_C$, or that one of the parties should learn no output).

The definition of fairness that we use follows that of Boneh and Naor [8], and states that no party has a significant advantage over the other party in computing his or her output. Namely, the definition uses parameters c and ϵ and states that if a party that aborts the protocol can compute his or her output in time t with probability q, the other party can compute her or his output in time $c \cdot t$ with probability at least $q - \epsilon$.

Definition 1 ((c, ϵ)-Fairness). *A protocol is (c, ϵ)-fair if the following two conditions hold:*

For any time t smaller than some security parameter and any adversary E working in time t as E: let E choose a function F and input x and run the two-party protocol for computing F. At some point E aborts the protocol and attempts to recover $F_E(x, y)$. Denote by q_1 the difference between E's probability of success, and the probability that a party that knows only F and x succeeds in *computing $F_E(x, y)$. Then there is an algorithm that C can run in time $c \cdot t$ for computing $F_C(x, y)$ after the protocol is aborted, such that if q_2 is the difference between this algorithm's probability of success and the probability of computing $F_C(x, y)$ given only F and y, it holds that $q_1 - q_2 \leq \epsilon$.*

The same requirement holds with relation to C. Namely, where we exchange between the roles of E and C, x and y, $F_E(x, y)$ and $F_C(x, y)$, and q_1 and q_2.

Note that the definition does not compare the probabilities of E and C successfully computing their outputs of F , but rather compares the advantages they gain over their initial probabilities of computing the result. This is done since the a-priori (i.e. initial) probabilities of E and C successfully computing their outputs might not be equal, for example, if the outputs of the two parties are of different lengths, and are each uniformly distributed given the other party's input.

2.2 The Basic Secure Two-Party Protocol

Known protocols for secure two-party computation follow a structure similar to that of Yao's original protocol. Following is a high-level description of this protocol (for more details see [30, 20, 22, 19]).
Input: E's input is x, C's input is y.
Output: E should learn $F_E(x, y)$ while C learns $F_C(x, y)$.
Goal: E must not learn from her protocol interaction with C anything that cannot be directly computed from x and $F_E(x, y)$. Similarly, C must not learn anything which cannot be directly computed from y and $F_C(x, y)$.
The protocol:
(1) C prepares a combinatorial circuit (using gates such as "or" and "and") with binary wires, whose inputs are x and y and whose outputs are $F_E(x, y)$ and $F_C(x, y)$. The circuit contains gates, and also input wires, internal wires, and output wires.
(2) C "scrambles" the circuit: he generates for every wire two random values (which we denote as the "garbled" values) to replace the 0/1 values, and every gate is replaced by a table that enables the computation of the garbled value of the output wire of the gate as a function of the garbled values of the input wires, and otherwise gives no information. These tables do not divulge any other information. (The details of this construction are not important for our purpose and we refer the interested user to [30, 20, 22]).
(3) For every bit of E's input x, E and C run an oblivious transfer protocol in which E learns the garbled value that corresponds to the value of her input bit.
(4) C sends to E the tables and the wiring of the gates (namely, the assignment of output wires of gates to input wires of other gates), and the garbled values that correspond to C's input values. He also sends to E a translation table from the garbled values of the output wires of $F_E(x, y)$ to their actual 0/1 values.
(5) E uses the tables to "propagate" the garbled values of the computation through the circuit. Namely, for every gate for which knows the garbled values of the input wires she computes the garbled values of the output wire. At the

end of this process E computes the garbled values of her output, which she can translate to 0/1 values using the translation tables. She also computes the garbled output values of $F_C(x, y)$.
(6) E sends the garbled output values of $F_C(x, y)$ to C. C can translate these to the 0/1 values of his output.

This protocol is clearly unfair, as E can simply not send to C the values of his output wires.

Comment 1 *In this paper E can verify that the circuit that C constructs for computing F is correct. Namely, that it does indeed compute F and not some other function. The correctness of the circuit is verified using the cut-and-choose procedure suggested in Section 3.1, whose error probability is exponentially small.*

Perquisites for the generic transformation: The fairness transformation can be applied to any two-party protocol with a structure similar to that of Yao's protocol. The element of the protocol which is essential for the fairness transformation is that C can set the garbled output values of the circuit for both his output and E's, and that E learns both her own output values and those of C. To be more precise, the fairness transformation depends on the following properties of the protocol:

Structure: (1) The output of the protocol is defined as a set of output wires, each having a binary value. (2) C can set random values to represent the 0 and 1 values of every output wire of the circuit, and can construct a translation table from these values to the original 0/1 values. (3) At the end of the protocol E learns the values of the output wires of the circuit and sends to C the values of the wires that correspond to C's output.

Privacy: When the protocol is stopped (either at its defined end of execution, or prematurely) the parties do not learn more than the defined outputs of the function. Namely, anything that a party can compute given his or her input and output and given his or her view of the interaction in the protocol, he or she can also compute given the input and output alone.

2.3 Gradual Release Timed Commitments

For the fair exchange protocol the two parties use a variation of the "timed commitments" suggested by Boneh and Naor [8]. The timed commitment protocol is a "timed cryptography" protocol (see e.g. [18, 3, 29]) for generating commitments while ensuring the possibility of forced opening, which can be achieved using some "large but achievable" amount of computation. The commitments are also secure against parallel attacks, preventing a powerful adversary that has a large number of machines from achieving a considerable advantage over a party that can use only a single machine (the only previous timed construction supporting this property was that of [29]). The Boneh-Naor construction was later improved by Garay and Jakobsson [21], who showed how to reuse the commitments and reduce their amortized overhead (this improvement can also be applied to our gradual release timed commitments). Mao [26] showed how to improve the efficiency of the proof that the commitment is correctly constructed.

Timed commitments have the following properties: (1) *Binding:* the generator of a timed commitment (the "sender") cannot change the committed value. (2) *Forced opening and soundness:* The receiver of the commitment can find the committed value by running a computation of 2^k steps (denoted as "forced opening"), where a typical value for k is in the range $30, \ldots, 50$ and enables to open the commitment by applying a moderately hard computation. The details of the construction guarantee that this computation cannot be parallelized, and therefore a party that can invest in many processors is not much stronger than a party that can only use a single processor. Furthermore, at the end of the commit phase the receiver is convinced that the forced opening algorithm will produce the committed value. (3) *Privacy:* Given the transcript of the commit phase, it is impossible to distinguish, with any non-negligible advantage, the committed value from a random string.

Our construction requires an additional property: the commitment must require an *infeasible* amount of work for forced opening (say 2^{80} computation steps), and in addition supports *gradual opening.* Namely, the sender of the commitment should be able to send to the receiver a sequence of "hints", where each of these hints reduces the computation task of opening the commitment by a factor of 2. Moreover, the receiver can verify that each hint message indeed reduces the effort of opening the commitment by a factor of 2. The commitment is therefore essentially *gradually timed.*[2] We describe in Appendix A two constructions of a gradual release timed commitment protocol, which are based on the timed commitment protocol of [8].

The motivation for using gradual release *commitments:* One might expect that the protocol could use simple timed commitments. After all, they support forced opening that requires a moderately hard effort, and therefore if one of the parties aborts the invocation of Yao's protocol the other party could apply forced opening to the commitments made by the first party. This solution is wrong since there is a point in time, right after E computes the output of the circuit and before she sends the commitments to C, where E knows the commitments made by C but C has no information about the commitments made by E. At that point E can run a forced opening procedure and learn her outputs, while C cannot learn any information about his outputs.

In order to solve this problem we must use timed commitments that, unlike those of [8], require an *infeasible* computation for forced opening say, 2^k modular squarings where k equals 80 (compare this to the Boneh-Naor protocol where k is typically in the range of $30, \ldots, 50$). This property ensures that when E has the commitments but C does not, E has no advantage over C since she cannot open

[2] Note that the timed commitment construction of [8] does not enable gradual opening. That paper uses timed commitments in order to design a contract signing protocol that has the gradual release property. However, the commitment protocol itself does not have this property.

the commitments. Consequently, the parties must somehow gradually reduce the amount of computation that is needed in order to open the commitments, until they can be opened using a feasible computation. This is where the gradual release property is required.

2.4 The Basic Transformation – Randomizing the Outputs

The first transformation towards achieving fairness involves the following steps: (1) C randomizing the output values, (2) E and C running a protocol that lets E learn the randomized outputs of both C and herself, and (3) E and C performing a fair exchange in which E sends to C his outputs, and C sends to E a derandomization of her outputs.

Transformation 1: We now describe the transformation in more detail. Given a two-party protocol that follows the structure of Yao's original protocol, the transformation applies to it the following modifications:

— Before running the two-party protocol C randomly permutes the order of the output entries in the tables defining the 0/1 values of the output wires. For each translation table corresponding to a binary wire i, C chooses a random binary key $k_i \in \{0,1\}$. If $k_i = 0$ then the table is not changed. If $k_i = 1$ the order of the entries in the table is reversed. The resulting tables are sent to E instead of the real tables.

— At the end of the protocol E learns the permuted 0/1 values of all output wires. She should then exchange values with C by receiving from C the keys k_i of the wires that correspond to E's output, and sending the output bits of the wires that correspond to C's output. (E does not know the corresponding original values since she does not know the required k_i values.) C can use these values to compute his original output since he knows how he randomized each wire.

— E and C use gradual release timed commitments in the following natural way, suggested in [8]. E sends to C a commitment to each of the values she should send him (namely the values of C's output wires), and C sends to E commitments to each of the values he should send her (namely the keys used to permute the values of E's output wires). Then the parties perform *gradual opening* of the commitments: E sends to C the first "hint" messages for each of the commitments she sent him. C verifies that these hints are valid and if so replies with the first hint messages for each of the commitments he sent to her. The parties continue sending and verifying alternate hint messages for k rounds, at the end of which both of them can open all the commitments they need to open.

Suppose that at some stage E defaults and does not send the required hint messages to C. Then the amount of work that C has to invest in order to open the commitments he received is at most twice the amount of work that E has to invest in order to open the commitments she received from C (since C received one less round of hint messages than E). On the other hand, if C defaults then the amount of work that he has to invest is at least the work that E has to do.

Claim 1 *If the commitments sent by the parties at the end of the protocol are for the correct values (namely E sends to C commitments to the values of each of C's output wires, and C sends to E commitments to each of the keys used to permute the values of E's output wires), then the protocol is $(2, \epsilon)$-fair, for a negligible ϵ. (Assuming that a computation of 2^k steps is infeasible for both parties, where k is the security parameter of the timed commitments.)*

Proof: (sketch) The privacy property of the two-party protocol ensures that with overwhelming probability the protocol execution reveals nothing to the parties until the step where E learns the values of the output wires. Since these values are randomly permuted by C, E learns nothing from observing them, too. Therefore no information is learned until the commitments are exchanged, and thus no party gains any advantage by terminating the protocol before that step.

The timed commitments ensure that the receiver of a commitment needs to invest a computation of 2^k steps, which cannot be parallelized, in order to find the committed value. The party that is the first to receive the commitments can open them in 2^k steps, while the other party (who at this stage does not have the commitments) might have to invest considerably more computation in order to break the privacy of the two-party protocol and learn his or her output. We therefore assume that a computation of 2^k steps is infeasible for both parties. (To get rid of this assumption we can set k to be such that 2^k is equal to the work needed in order to break the privacy property of the two-party protocol. In that case the party that receives the commitments first has no advantage over the other party.)

After both parties have their respective commitments, they run the gradual opening protocol. The largest advantage that one of the parties can gain by terminating the protocol is a factor of 2, achieved by the party that received one more "hint" than other party. □

The remaining challenge is enabling the parties to verify that the timed commitments they receive are commitments to the actual values that are defined in the protocol, which enable the computation of F.

3 Verifying the Commitments

The main technical focus of this paper is the design of a method that enables the two parties to verify that the commitments they receive commit to the values defined by the protocol. The verification of the commitments sent by C (the constructor) can be done using a cut-and-choose method with an exponentially small error probability. The verification of the commitments sent by E (the evaluator) is less straightforward and involves a novel use of blind signatures. The difficulty in verifying these signatures arises from the fact that E learns an output value of the circuit, which was defined by C, and must provide C with a commitment to this value. We must ensure that E sends a commitment to the right value, but simultaneously we should make sure that C, who generated the circuit, cannot recognize the commitment.

Section 3.1 describes how to transform the protocol into a protocol that lets E verify the commitments sent to her by C. Section 3.2 describes a transformation that lets C verify the commitments sent to him by E. Both transformations have an exponentially small error probability. Both transformations, as well as Transformation 1 (Section 2.4), must be applied in order to ensure fairness.

3.1 Verifying C's commitments

E should learn commitments to the random permutations that C applies to the wires of E's output bits, namely the k_i values that correspond to E's output wires. The protocol should be changed to let E verify that the commitments are to the correct values of the permutations. This is done using the following cut-and-choose technique:

Transformation 2:

— Let ℓ be a security parameter. Before the protocol is run C prepares 2ℓ different circuits that compute the required function. Let m be the number of output wires in the circuit. For every output wire in each circuit C chooses a random bit k_i that defines a permutation of the order of the output bits. I.e., every output bit of the new circuit is equal to the exclusive-or of the corresponding k_i and the original output bit.

— C sends to E the following information: (1) the tables that encode each circuit, (2) gradual release timed commitments of all k_i values corresponding to the output wires, and (3) commitments to the garbled values of the input wires, ordered in two sets. Namely, a set I_0 that includes the commitments to all the $2\ell m$ garbled values of "0" input values, and a set I_1 of the commitments to all the $2\ell m$ garbled values of "1" input values.

— E chooses a random subset of ℓ of the 2ℓ circuits. She asks C to open the commitments of the garbled values that correspond to both the 0 and 1 values of each of the input wires of these circuits, as well as open the commitments to all the permutations k_i of the output wires of these circuits. She uses this information to verify that all ℓ circuits compute the function F. If this is not the case E aborts the protocol.

– E and C run the two-party protocol for each of the remaining ℓ circuits. In the first step of the protocol, C sends to E the garbled values of the input wires that correspond to C's input in all the remaining ℓ circuits. C then proves to E, that C's input values are consistent in all the evaluated circuits. Namely, that for each of his input wires either *all* of the ℓ garbled values correspond to a 0 input, or all of them correspond to a 1 input. This can be done by proving that either all of them are taken from the set I_0 or they are all taken from the set I_1, and this proof can be done using the "proofs of partial knowledge" method of Cramer et al [16] (the full version of this paper contains a more elaborate discussion of this proof and more efficient constructions). E can then compute the circuit and learn the permuted output values of all output wires in each of the circuits, and commitments to the random permutations of these wires.

– E and C then run a gradual opening of their commitments. At the end of this stage E can compute her actual output in each of the circuits.

– For each output wire E finds which is the most common output value in the ℓ circuits that she computed (i.e. computes the majority of the ℓ bits that were output), and defines it to be her output value. (If not all ℓ circuits have the same output value then it is clear that C cheated. Still, since this is found *after* the commitments were open it is too late for E to abort the protocol, and therefore we must define an output for her.)

Comment 2 *This transformation only verifies that a malicious C cannot prevent E from learning the output of F. The transformation does not ensure that a malicious C cannot learn an output different from F. This issue is discussed in Section 3.2.*

Claim 2 *E computes the right output value with probability at least* $1-(3/4)^{3\ell/2}$*.*

Proof: (sketch) E computes the wrong output value only if there are at least $\ell/2$ circuit evaluations that do not compute F correctly. A circuit evaluation does not compute F correctly if (1) the function encoded in the circuit itself is not F, or (2) the commitments to the permutation values are wrong, or (3) C did not provide the correct garbled input values for the computation of the circuit. Each of these events is detected by the test run at the beginning of the protocol. C also proves that the inputs he provides to all ℓ copies of the circuit are taken from the same set of inputs. Consequently, a wrong final output is computed if there are $\ell/2$ circuit evaluations that result in an incorrect evaluation of F, and *none* of them was chosen by E among the ℓ circuits she opens. This event happens with probability at most $\binom{3\ell/2}{\ell}/\binom{2\ell}{\ell} \approx (3/4)^{3\ell/2}$. □

Overhead: The transformation increases C's overhead of generating the circuit, and also the communication between the parties, by a factor of 2ℓ. The parties should compute ℓ circuits, but the computation overhead involved in that might not increase substantially since the oblivious transfer step (which is the major computation task and is done in the beginning of the circuit evaluation in Yao's protocol) can use only a single public key operation per bit of E's original input. This is done by E using a single oblivious transfer to learn, for each of her input bits, either *all* ℓ garbled values of a 0 input, or *all* those of a 1 input. The computational overhead of the partial proofs of knowledge protocol is $O(\ell m)$ exponentiations. The gradual opening of the circuits requires k steps, and k public key operations per output bit.

3.2 Verifying E's Commitments

At the end of the circuit evaluation E should provide C with commitments to the permuted values of C's output wires. This step seems hard to implement: On one hand if E generates the commitments from the permuted output values of the circuit, nothing prevents her from generating commitments to incorrect values. On the other hand if C generates the commitments when he constructs the circuit he might be able to identify, given a commitment, the value that is

Step	E		C
1		blinded commitments $\rightarrow$	
2			checks random subset
3			signs remaining commitments assigns to output tables
4		*circuit evaluation*	
	unblinds output		
5		signed commitments $\rightarrow$	
6			verifies signatures
7		*gradual opening*	
8			learns majority of results

Fig. 1. The verification of E's commitments.

committed to. This gives him an unfair advantage in the protocol since he does not need to wait for E to open the commitments for him.

The solution we suggest is based on the use of blind signatures (see e.g. [12]). Loosely speaking, the solution operates in the following way, described in Figure 1: (1) C prepares several copies of the circuit. For every wire corresponding to a bit of C's output, E generates a set S_0 of commitments to 0, and a set S_1 of commitments to 1. She asks C to blindly sign each of the commitments in these sets. (2) C then asks E to open a random sample of half of the commitments in each set, and verifies that the opened commitments from S_0 are to 0 values, and the opened commitments from S_1 are to 1 values. (3) If all commitments are correct then C signs each of the remaining values, and puts them in the permuted output tables of the circuit, replacing the 0/1 output values. (4) After E evaluates the circuit and obtains the output values she *unblinds* the signatures and obtains signatures of C to commitments of the *permuted* output values. (5) E then sends the signed commitments to C. (6) C can verify that he indeed signed the commitments, and therefore that he has placed them in the correct place in the output tables and that they are committing to the correct values. Since the signature was blind he cannot, however, know what are the values that are committed to in each commitment. (7) The parties run a gradual opening of their commitments. (8) For each of his output wires, C learns the majority of the output values (each output value is defined by applying the permutation to the committed value).

Blind signatures Our construction uses blind signatures. A blind signature protocol is run between a data owner A and a signer B. The input to the protocol is an input x known to A, and the output is a signature of x done by B. Blind signatures were introduced by Chaum [12], and a formal treatment of them was given in [4], where Chaum's signature was shown to be secure given the random oracle assumption and a slightly non-standard RSA assumption.

The public key in Chaum's blind signature scheme is a pair (N, e), and his secret key is d, as in the RSA scheme. The system also uses a public hash function

H whose range is Z_N^*. A signature of a message M is defined a $(H(M))^d \bmod N$, and a message-signature pair (M, Y) is considered valid if $H(M) = Y^e \bmod N$. The blind signature protocol is run in the following way:

– Given an input x, A picks a random value $r \in_R Z_N^*$, computes $\bar{x} = r^e \cdot H(x) \bmod N$, and sends it to B.
– B computes $\bar{y} = \bar{x}^d \bmod N$ and sends it back to A.
– A computes $y = \bar{y}/r$. The pair (x, y) is a valid message-signature pair.

To be concrete we describe our construction using Chaum's blind signatures, rather than referring to a generic blind signature scheme.

Blind MACs: The blind signatures in our protocol are generated and verified by the same party. This means that, at least potentially, message authentication codes (MACs) could be used instead of signatures. We do not know, however, how to construct a MAC which has the blindness property.

3.3 A Protocol with an Exponentially Small Cheating Probability

The basic idea of the transformation to a protocol that verifies E's commitments was described above. A subtle issue that must be addressed is ensuring that the only information that C learns is the result of computing the *majority* of the output values of the ℓ circuits. Otherwise, a corrupt C could, with high probability, learn additional information about E's input. See discussion below.

In more detail, the protocol operates in the following way: Let (N, e) be a public RSA modulus and an exponent, and let d be the inverse of e that is known only to C. Let H be a hash function whose range is Z_N^*, which is modeled in the analysis as a *random function.*

Transformation 3: Note that Transformation 2 required the parties to evaluate ℓ circuits, in order to ensure E that the output of the protocol is the correct value of F. We now describe a transformation that is applied to every output wire of each of these circuits.

Pre circuit evaluation stage: Before C constructs the output tables of the circuits he runs, for each of the circuits, the following protocol with E:

— Assume that C's output is m bits long. The basic step involves E generating, for every output wire i of C in circuit j, a commitment c_0 to the tuple $(i, j, 0)$, and a a commitment c_1 to the tuple $(i, j, 1)$. She then chooses random values $r \in_R Z_N^*$, and hashes and randomizes the commitments, to generate (modulo N) values of the form $\bar{c_0} = r^e \cdot H(c_0)$, or $\bar{c_1} = r^e \cdot H(c_1)$. Let n be a security parameter. E generates for every output wire a set S_0 of $2n$ randomized hashes of commitments to 0, and a similar set S_1 of commitments to 1. E assigns a different identification number to each set and sends them to C.

— C chooses a random subset of n commitments from each set and asks E to open them. He verifies that all commitments to 0 (1) indeed commit to a 0 (1) value, and include the correct circuit and wire identifiers. (Namely, given a value $\bar{c_0}$, E should present to C a commitment c_0 to 0 and to the circuit and wire indenifiers, and a value r such that $\bar{c_0} = r^e \cdot H(c_0)$. E should also open the commitment c_0. If one of the tests fails then C aborts the protocol.)

— C signs each of the remaining $2 \times m \times n$ $\bar{c}$ values by raising them to the dth power modulo N, generating for each of his output wires n signed commitments for the 0 entry, and n signed commitments for the 1 entry.

— As in the basic protocol, C chooses random bit permutations k_i for the order of the values of each of his output wires. C then maps the n values of S_0, and this set's identification number, to the 0 entry of the permuted table, and the n values of S_1, and this set's identification number, to the 1 entry of the permuted table.

Post circuit evaluation stage:

— The parties evaluate all the circuits. There are ℓ circuits, each of them has m wires that correspond to C's output, and for each of these wires E learns n values, as well as the corresponding identification number.

— E uses the identification number to locate the random r values that she used to blind the commitments, and divides each one of them by the corresponding r value. The result is a set of n signatures of commitments per wire.

Comment: The protocol computes the result of ℓ circuits, which were constructed by C. As we explain below, it is insecure to let C learn the outputs of all these circuits, since in this case an adversarial C can construct circuits that disclose illegitimate information to him, even after the cut-and-choose test done by E in Transformation 2 (which only assures, with high probability, that the majority of the ℓ circuits that are computed are legitimate). The following step is done in order to prevent a corrupt C from learning illegitimate information. (This also means that in the analysis of this step we should mostly be concerned of adversarial behavior of C, and not of E.)

— Before the parties run a gradual opening of their commitments they should compute the majority of the output values for each of C's output bits, and find a circuit whose output agrees with this majority decision. In order to do that the parties run the following procedure:

- E generates a new circuit with $2m\ell$ inputs. For each output wire of the original circuit there are ℓ inputs, corresponding to the output values of this wire in the copies of the original circuit that are evaluated by E. There are also ℓ inputs that correspond to the permutations (k_i values) that C applied to this wire in each of these circuits.
- For each output wire, the new circuit computes the exclusive-or of each pair of E's and C's inputs corresponding to the wire. It then computes the majority of these ℓ results. Finally, it outputs an index $1 \leq i \leq \ell$ such that the output of the ith copy of the original circuit agrees with all the majority results computed above. (Such a circuit exists with high probability, since, with high probability, at least $\ell/2$ circuits compute the correct function.)
- Since we are worried that C might cheat in this protocol, we must ensure that the input values that he uses are indeed the permutations that were used in constructing the circuits. To achieve this, C prepares *in advance* the encoding of his input to the majority circuit (i.e. his messages for the oblivious transfer step of computing this circuit), and stores commitments to these messages together with the other commitments that he provides to

E in the protocol described in Transformation 2. When E chooses a set of ℓ circuits to examine, she also verifies that the commitments to C's input to the majority protocol corresponding to these circuits, are the same as the permutations he used in constructing the circuits.
- The parties evaluate the majority computation circuit. The output is the index of one of the original circuits, that all of its m output values correspond to the majority results. Note that the size of the circuit computing the majority function is small, $O(m\ell \log \ell)$.

The parties now run the gradual opening of the commitments, where E only opens the nm commitments that are output from the circuit chosen by the above procedure. At the end of this stage C has a set of n opened commitments for each of his m output wires. He takes the majority value of each of these sets as the output value of the corresponding wire.

Claim 3 *An honest C computes a wrong output with prob. at most* $m \cdot (\frac{3}{4})^{3n/2}$.

Proof: (sketch) The decoding of a specific output wire in a specific circuit fails only if E provided at least $n/2$ corrupt values, and none of them is checked by C in the initialization stage. The probability that this event happens is $\binom{3n/2}{n} / \binom{2n}{n} \approx (3/4)^{3n/2}$. The probability that in a specific circuit there are one or more undetected corrupt output wires is therefore at most $m \cdot (\frac{3}{4})^{3n/2}$. Now, if C is honest then he can compute his correct output given the output of any of the ℓ copies of the original circuit. Even if E inputs an incorrect input to the majority computation circuit, the output of this circuit would still be an index of one of the ℓ copies of the original circuit. The gradual opening phase therefore enables C to compute the output of this copy, which is the correct output with probability $1 - m \cdot (\frac{3}{4})^{3n/2}$.

Claim 4 *C cannot learn any of the committed values before the beginning of the gradual release stage.*

Claim 5 *If E is honest then with probability at least* $1 - (3/4)^{3\ell/2}$ *C cannot compute anything but the output of* F.

Proof: (sketch) As described in Section 3.1, E verifies that a in randomly selected sample subset of ℓ of the original 2ℓ circuits F is evaluated correctly, and the majority circuit computation receives a correct input from C. In addition, C proves that she uses the same input in the evaluation of the remaining ℓ circuits. Therefore, with probability $1 - (3/4)^{3\ell/2}$, at least $\ell/2$ of the computed circuits compute C's output as defined by F (we are assured that E's input to the circuits is correct, since we assume E to be honest). When this event happens, the result of the majority circuit that is run at the post-processing stage is an index of one of these circuits, and C's output is the output of this circuit.

Note that if C were learning the output of all ℓ circuits then he was able to cheat with high probability. For example, he could have generated $2\ell - 1$ legitimate circuits and a single circuit that computes a function different from

F (say, a circuit that outputs E's input). This circuit has a probability of 1/2 of avoiding examination by E. Furthermore, C could have learned information even if the protocol was comparing the outputs of the circuits and refusing to provide C with any output in the case that not all circuits have the same output. For example, C could have generated one circuit whose output agrees with that of the other circuits if E's input has some property, and disagrees with the other circuits if this property does not hold. C could then learn this property based on observing whether he receives any output at all at the end of the protocol. We avoid these problems by making sure that the only information that C learns is the majority the outputs of the ℓ circuits.

Overhead: In the initialization stage E sends to C $2nm\ell$ commitments. C performs $n\ell$ blind signatures for each of his m output wires. The size of the tables of the output wires increases by a factor of n, but, unless the ratio between the number of gates and the number of input wires is small, this has little effect on the overall communication overhead of sending the tables, since the internal tables of the circuit do not change. The unblinding and signature verification stages have negligible complexity compared to the signature generation.

4 Open Problems

As noted in Section 3.2 the transformation can use blind message authentication codes (MACs) instead of blind signatures. It would be interesting to come up with a construction of a blind MAC. Another interesting problem is to find whether the number of rounds at the end of the protocol can be $o(k)$, where k is the security parameter of the commitment scheme.

5 Acknowledgments

We would like to thank Stuart Haber and Moni Naor, as well as the anonymous referees, for providing us with many helpful comments.

References

1. B. Baum-Waidner and M. Waidner, *Optimistic asynchronous multi-party contract signing*, Research report RZ 3078 (# 93124), IBM Research, Nov. 1998.
2. D. Beaver and S. Goldwasser, *Multiparty computation with faulty majority*, Proc. 30th FOCS, pp. 468-473, 1989.
3. M. Bellare and S. Goldwasser, *Verifiable partial key escrow*, 4th ACM CCS conference, pp. 78-91, 1997.
4. M. Bellare, C. Namprempre, D. Pointcheval and M. Semanko, *The power of RSA inversion oracles and the security of Chaum's RSA-based blind signature scheme*, in proc. of Financial Crypto '01, 2001.
5. M. Ben-Or, O. Goldreich. S. Micali and R. L. Rivest, *A fair protocol for signing contracts*, IEEE Trans. on Information Theory, vol. 36, 40-46, Jan. 1990.

6. M. Blum, *How to exchange (secret) keys*, ACM Transactions on Computer Systems, 1(2):175-193, May 1983.
7. L. Blum, M. Blum, and M. Shub, *A Simple Unpredictable Pseudo-Random Number Generator*, SIAM Journal on Computing, Vol. 15, pp. 364-383, May 1986.
8. D. Boneh and M. Naor, *Timed commitments*, Advances in Cryptology – Crypto '2000, Springer-Verlag LNCS 1880, 236–254, 2000.
9. F. Boudot, B. Schoenmakers and J. Traore, *A Fair and Efficient Solution to the Socialist Millionaires' Problem*, Discrete App. Math. 111, pp. 23-36, July 2001.
10. E. Brickell, D. Chaum, I. Damgard and J. van de Graaf, *Gradual and verifiable release of a secret*, Adv. in Crypt. – Crypto '87, Springer-Verlag LNCS 293, 1988.
11. C. Cachin and J. Camenish, *Optimistic fair secure computation*, Advances in Cryptology – Crypto '2000, Springer-Verlag LNCS 1880, 94–112, 2000.
12. D. Chaum, *Blind signatures for untraceable payments*, Advances in Cryptology – Crypto '82, pp. 199-203, 1982.
13. D. Chaum and T. Pedersen, *Wallet databases with observers*, Advances in Cryptology Crypto '92, Springer-Verlag, pp. 89-105, 1992.
14. R. Cleve, *Limits on the security of coin flips when half the processors are faulty*, STOC '86, 364–369, 1986.
15. R. Cleve, *Controlled gradual disclosure schemes for random bits and their applications*, Adv. in Crypt. – Crypto '89, Springer-Verlag, LNCS 435, 573-588, 1990.
16. R. Cramer, I. Damgard and B. Schoenmakers, *Proofs of Partial Knowledge and Simplified Design of Witness Hiding Protocols*, Advances in Cryptology Crypto '94, Springer Verlag LNCS, vol. 839, pp. 174-187, 1994.
17. I. Damgard, *Practical and provably secure release of a secret and exchange of signatures*, J. Cryptology, 8(4):201–222, 1995.
18. C. Dwork and M. Naor, *Pricing via processing, or combatting junk email*, Advances in Cryptology – Crypto '92, Springer-Verlag, 139-147, 1990.
19. M. Franklin, *Complexity and security of distributed protocols*, PhD dissertation, Columbia University, 1993.
20. Z. Galil, S. Haber and M. Yung, *Cryptographic Computation: Secure Fault-tolerant Protocols and the Public-Key Model*, Advances in Cryptology – Crypto '87, Springer-Verlag LNCS 293, 135-155, 1988.
21. J. Garay and M. Jakobsson, *Timed Release of Standard Digital Signatures* , Proc. Financial Cryptography 2002, March 2002.
22. O. Goldreich, *Foundations of Cryptography (Fragments of a Book)*, 1995. Available at `http://www.wisdom.weizmann.ac.il/~oded/frag.html`.
23. O. Goldreich and L.A. Levin, *A hard-core predicate for all one-way functions*, Proc. of the 21st ACM Symposium on Theory of Computing (STOC), pp. 25-32, 1989.
24. S. Goldwasser and L. Levin, *Fair computation of general functions in presence of immoral majority*, Adv. in Crypt. – Crypto '90, Springer-Verlag LNCS 537, 1991.
25. M. Luby, S. Micali and C. Rackoff, *How to simultaneously exchange secret bit by flipping a symmetrically-biased coin*, Proceedings of FOCS '83, 23-30, 1983.
26. W. Mao, *Timed-Release Cryptography*, Selected Areas in Cryptography VIII (SAC '01), Springer-Verlag LNCS 2259, pp. 342-357, 2001.
27. S. Micali, *Secure protocols with invisible trusted parties*, presented at the Workshop for Multi-Party Secure Protocols, Weizmann Inst. of Science, June 1998.
28. M. Naor and B. Pinkas, *Efficient Oblivious Transfer Protocols*, Proceedings of SODA 2001 (SIAM Symposium on Discrete Algorithms), January 7-9 2001.
29. R. Rivest, A. Shamir and D. Wagner, *Timed lock puzzles and timed release cryptography*, TR MIT/LC/TR-684, 1996.

30. A. Yao, *Protocols for secure computation*, Annual Symposium on Foundations of Computer Science (FOCS), 162-167, 1986.

A Gradual Release Timed Commitments

We describe a variant of the Boneh-Naor timed commitment scheme, which enables the committer to perform a gradual release of the comitted value.

A.1 The Boneh-Naor construction

The timed commitments construction of [8] is defined in the following way:
Setup phase: Let n be a security parameter. The committer chooses two primes p_1, p_2 of length n, such that they both equal 3 mod 4. He publishes $N = p_1 p_2$, and keeps p_1 and p_2 private. Let k be an integer, typically in the range $(30, \ldots, 50)$.
Commit phase:

- The committer picks a random $h \in Z_N$, computes $g = h^{\Pi_{i=1}^{r}(q_i)^n} \bmod N$, where the q_is are all the primes smaller than some bound B. He sends g and h to the receiver, who verifies the computation of g.
- The committer computes $u = g^{2^{2^k}} \bmod N$, using her knowledge of $\phi(N)$.
- The committer hides the committed value M, using a BBS pseudo-random generator [7] whose tail is u. Namely, he xors the bits of M with the least significat bits of the successive square roots of u. In other words, he creates a sequence $S = S_1, \ldots, S_{|M|}$ where $S_i = M_i \oplus lsb(g^{2^{2^k - i}})$ for $i = 1, \ldots, |M|$. The commitment is $\langle h, g, u, S \rangle$.
- In order to prove that u is constructed correctly, the committer constructs the following vector W of length k and sends it to the receiver

$$W = \langle b_0, \ldots, b_k \rangle = \langle g^2, g^4, g^{16}, g^{256}, \ldots, g^{2^{2^i}}, \ldots, g^{2^{2^k}} \rangle \pmod N.$$

  For every i the committer proves in zero-knowlede that the tuple $\langle g, b_{i-1}, b_i \rangle$ is of the form $\langle g, g^x, g^{x^2} \rangle$. These proofs, together with a verification that $b_k = u$, convince the reciever that $u = g^{2^{2^k}}$. The proofs themselves are based on the zero-knowledge proof that a Diffie-Hellman tuple $\langle g, g^a, g^b, g^{ab} \rangle$ is correctly constructed, and are very efficient, see [13, 8] for details.

Open and forced open phases: We only describe here the main ideas of these phases, and refer the reader to [8] for more details. In the open phase the committer provides the reciever with a value $h^{2^{2^k - |M|}}$ from which the receiver can easily compute $g^{2^{2^k - |M|}}$, reveal the BBS sequence, compare the last element to u and decrypt M.

The forced open phase is run by the receiver if the commiter refuses to open the commitment. The receiver computes $g^{2^{2^k - |M|}}$ from g, or better still, from $b_{k-1} = g^{2^{2^{k-1}}}$. This computation requires $2^{k-1} - |M|$ squarings modulo N

(recall that k is typically at most 50). The security is based on the *generalized BBS assumption*, which essentially states that given the first $k-1$ elements of W the task of distinguishing between $g^{2^{2^k}}$ and a random residue is not significantly easier for a parallel algorithm than it is for a sequential algorithm.

A.2 Gradual Release

We describe two methods that enable gradual release, i.e. enable the committer to send "hints" to the receiver, such that each hint reduces by a factor of two the amount of work required for forced opening.

The halving method This method is based on using the same commitments as in the Boneh-Naor scheme, and defining a sequence of hints, each halving the distance between the last known element of the BBS sequence and $g^{2^{2^k-|M|}}$.

Before the first hint is sent, the receiver knows $b_{k-1} = g^{2^{2^{k-1}}}$, and can compute $g^{2^{2^k-|M|}}$ using $d_0 = 2^{k-1} - |M|$ squarings modulo N. The first hint is therefore the value $h_1 = g^{2^{2^{k-1}+2^{k-2}}}$ that reduces the number of squarings to $d_1 = 2^{k-1} - 2^{k-2} - |M| = 2^{k-2} - |M| \approx d_0/2$ (we assume that $|M| \ll 2^{k-1}$). In the same way, the following hints are defined as $h_i = g^{2^{2^{k-1}+2^{k-2}+\cdots+2^{k-i-1}}}$. The required work of forced opening after the ith hint is $d_i = 2^{k-i-1} - |M| \approx d_{i-1}/2$

The committer must also prove to the receiver that each of the hints is correctly constructed. This can be easily done using the Diffie-Hellman tuple zero-knowledge proofs that were used in constructing the sequence W.

A method based on the square roots of the sequence *W* This method is based on the seqeunce $W = \langle b_0, \ldots, b_k \rangle$ itself (which is also used in the signature scheme suggested in [8]). The gradual release timed commitment is defined in the following way:

- Define, $r_i = \sqrt{b_i} \pmod N$, for $i = 1, \ldots, k$. Namely, this is a seqeunce of the square roots modulo N of the elements of W.
- Define a bit c_i, for $i = 1, \ldots, k$, to be a hard-core predicate of r_i (for a discussion and construction of hard-core predicates see [23]).
- Define a k bit key as $c = c_1, \ldots, c_k$. Use this as a key to a committment scheme, and commit to the value M (note that the bits that are used here as the key of the commitment scheme are different than those used in [8]).

In the opening phase, the committer reveals the values r_i, which can be easily verified by the receiver using simple squarings. The key c can then be computed from the hard-core predicates. The forced opening phase is identical to the forced opening phase in the Boneh-Naor protocol.

The gradual release is performed by revealing the sequence of r_i values starting from r_k. Each value is verified by the receiver by comparing its square to the corresponding b_i value from W. Each value r_i reduces the computational overhead of forced opening by a factor of 2.