

# Two-output Secure Computation With Malicious Adversaries

abhi shelat and Chih-hao Shen

University of Virginia, Charlottesville, VA 22904  
{shelat, shench}@virginia.edu

**Abstract.** We present a method to compile Yao’s two-player garbled circuit protocol into one that is secure against malicious adversaries that relies on witness indistinguishability. Our approach can enjoy lower communication and computation overhead than methods based on cut-and-choose [13] and lower overhead than methods based on zero-knowledge proofs [8] (or  $\Sigma$ -protocols [14]). To do so, we develop and analyze new solutions to issues arising with this transformation:

- How to guarantee the generator’s input consistency
- How to support different outputs for each player *without* adding extra gates to the circuit of the function  $f$  being computed
- How the evaluator can retrieve input keys but avoid selective failure attacks
- Challenging 3/5 of the circuits is near optimal for cut-and-choose (and better than challenging 1/2)

Our protocols require the existence of secure-OT and claw-free functions that have a weak malleability property. We discuss an experimental implementation of our protocol to validate our efficiency claims.

*Keywords:* Witness indistinguishability, Yao garbled circuits, signature schemes

## 1 Introduction

Yao [23] proposed a method that allows two *honest-but-curious* players—a *generator* (denoted by  $P_1$ ) with secret input  $x$ , and an *evaluator* (denoted by  $P_2$ ) with secret input  $y$ —to jointly compute a function  $f(x, y)$  such that  $P_1$  receives nothing and  $P_2$  receives  $f(x, y)$ .<sup>1</sup> In this paper, we propose an approach for transforming Yao’s garbled circuit protocol for honest-but-curious players into a protocol that is secure against *malicious* players. Our main goal is to improve the efficiency of this transformation and to do so using more general assumptions.

There are two well-known methods to achieve this transformation: the *commit-and-prove* and *cut-and-choose*. The commit-and-prove method suggested by Goldreich, Micali, and Wigderson [6] only requires the weak general assumption of zero-knowledge proofs of knowledge. However, this approach requires costly NP-reductions, which have never been implemented. On the other hand, an efficient

---

<sup>1</sup> A thorough description of this protocol can be found in Lindell and Pinkas [13].

transformation based on the cut-and-choose method was recently proposed by Lindell and Pinkas [13] and implemented by Pinkas et al. [20]. The general idea in cut-and-choose is for  $P_1$  to prepare multiple copies of the circuit to be evaluated. A randomly selected set of the circuits (called *check-circuits*) are then opened to show if they were constructed correctly. Finally, the unopened circuits (called *evaluation-circuits*) are evaluated by  $P_2$  and the majority of the results is taken as the final output. This approach has only constant round complexity, but the replication incurs both communicational and computational overhead.

The starting point for our work is the cut-and-choose method. *A natural question we aim to study is to understand the fundamental limitations (in terms of efficiency) of the cut-and-choose method.* This method does not require NP-reductions; however, it faces other efficiency problems stemming from the new security problems introduced by evaluating  $e$  out of  $s$  copies of the circuit. In this paper, we address several of these issues: (1) ensuring input consistency, (2) handling two-output functions, (3) preventing selective failure attacks, and (4) determining the optimal number of circuits to open versus evaluate. Moreover, we identify weak and generic properties that admit efficient solutions to these issues. In several of the cases, using witness indistinguishable protocols suffice. Thus, in the case of input consistency, we are able to use an extremely efficient protocol as long as claw-free functions with a minimal malleability property exist (they do under the standard algebraic assumptions). We will later demonstrate the benefits of our approach by both asymptotic analysis of complexity and experimental results from an implementation. We now give an overview of our contributions.

### 1.1 Generator’s input consistency

According to the cut-and-choose method,  $P_1$  needs to send  $e$  copies of her garbled input to  $P_2$ . Since the circuits are *garbled*,  $P_1$  could cheat by sending different inputs for the  $e$  copies of the garbled circuit. For certain functions, there are simple ways for  $P_1$  to extract information about  $P_2$ ’s input (§ 3 of [13]). Therefore, the protocol must ensure that all  $e$  copies of  $P_1$ ’s input are *consistent*.

*Related work* Let  $n$  be  $P_1$ ’s and  $P_2$ ’s input size, and let  $s$  be a statistical security parameter for the cut-and-choose method. Mohassel and Franklin [16] proposed the *equality-checker* scheme, which has  $O(ns^2)$  computation and communication complexity. Woodruff [22] later suggested an *expander-graph* framework to give a sharper bound to  $P_1$ ’s cheating probability. The asymptotic complexity is  $O(ns)$ , however, in practice, the constant needed to construct the expander graphs is prohibitively large. Lindell and Pinkas [13] 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. Jarecki and Shmatikov [8] presented an approach that is based on commit-and-prove method. Although only a single 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*. Nielsen and Orlandi [18] 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  $P_1$ 's input keys is needed for all the  $e$  copies of the garbled circuit. However, similar to Jarecki and Shmatikov's approach, each gate needs several group elements as commitments resulting both computational and communicational overhead. Lindell and Pinkas propose a Diffie-Hellman pseudorandom synthesizer technique in [14]; their approach relies on finding efficient zero-knowledge proofs for specifically chosen complexity assumptions, which is of complexity  $O(ns)$ .

*Our approach to consistency* We solve this problem not by explicitly using zero-knowledge protocols (or  $\Sigma$ -protocols) but by communicating merely  $O(ns)$  group elements. Our novel approach is to first observe that witness indistinguishable proofs suffice for consistency, and to then use *claw-free functions*<sup>2</sup> that have a weak malleability property to generate efficient instantiations of such proofs.

Intuitively,  $P_1$ 's input is encoded using elements from the domain of the claw-free collections which can later be used to prove their consistency among circuits. The elements are hashed into random bit-strings which  $P_1$  uses to construct keys for garbled input gates. The rest of the gates in the circuit use fast symmetric operations as per prior work. A concrete example is to instantiate the claw-free functions under the Discrete Logarithm assumption by letting  $f_b(m) = g^b h^m$  for some primes  $p$  and  $q$  such that  $p = 2q + 1$ , and distinct group elements  $g$  and  $h$  of  $\mathbb{Z}_p^*$  such that  $\langle g \rangle = \langle h \rangle = q$ . It is well-known that such a pair of functions have efficient zero-knowledge proofs. An example instantiation of our solution built on this pair of claw-free functions works as follows:  $P_1$  samples  $[m_{0,1}, \dots, m_{0,s}]$  and  $[m_{1,1}, \dots, m_{1,s}]$  from  $f_0$  and  $f_1$ 's domain  $\mathbb{Z}_q$ . The range elements  $[h^{m_{0,1}}, \dots, h^{m_{0,s}}]$  and  $[gh^{m_{1,1}}, \dots, gh^{m_{1,s}}]$  are then used to construct garbled circuits in the way that  $g^b h^{m_{b,j}}$  is associated with  $P_1$ 's input bit value  $b$  in the  $j$ -th garbled circuit. The cut-and-choose method verifies that the majority of the evaluation-circuits are correctly constructed. Let  $[j_1, \dots, j_e]$  be the indices of these evaluation-circuits. At the onset of the evaluation phase,  $P_1$  with input bit  $x$  reveals  $[g^x h^{m_{x,j_1}}, \dots, g^x h^{m_{x,j_e}}]$  to  $P_2$  and then proves that these range elements are the commitments of the same bit  $x$ . Intuitively, by the identical range distribution property,  $P_2$  with  $f_x(m_{x,i})$  at hand has no information about  $x$ . Furthermore, after  $P_1$  proves the knowledge of the pre-image of  $[f_x(m_{x,j_1}), \dots, f_x(m_{x,j_e})]$  under the same  $f_x$ , by the claw-free property,  $P_1$  proves the consistency of his input keys for all the evaluation-circuits.

Furthermore, in the course of developing our proof, we noticed that witness indistinguishable proofs suffice in place of zero-knowledge proofs. Even more generally, when the claw-free collection has a very weak malleability property

---

<sup>2</sup> Loosely speaking, a pair of functions  $(f_0, f_1)$  are said to be claw-free if they are (1) easy to evaluate, (2) identically distributed over the same range, and (3) hard to find a *claw*. A claw is a pair of elements, one from  $f_0$ 's domain and the other from  $f_1$ 's domain, that are mapped to the same range element.

(which holds for all known concrete instantiations), sending a simple function of the witness itself suffices. We will get into more details in §2.1.

It is noteworthy that both the committed-input scheme in [16] and Diffie-Hellman pseudorandom synthesizer technique in [14] 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 our case, which means that our approach is faster by a constant factor in practice.

## 1.2 Two-output Functions

It is not uncommon that *both*  $P_1$  and  $P_2$  need to receive outputs from a secure computation, that is, the goal function is  $f(x, y) = (f_1, f_2)$  such that  $P_1$  with input  $x$  gets output  $f_1$ , and  $P_2$  with input  $y$  gets  $f_2$ .<sup>3</sup> In this case, the security requires that *both* the input and output are hidden from the other player. When both players are honest-but-curious, a straightforward solution is to let  $P_1$  choose a random number  $c$  as an extra input, convert  $f(x, y) = (f_1, f_2)$  into a new function  $f^*((x, c), y) = (\lambda, (f_1 \oplus c, f_2))$ , run the original Yao protocol for  $f^*$ , and instruct  $P_2$  to pass the encrypted output  $f_1 \oplus c$  back to  $P_1$ , who can then retrieve her real output  $f_1$  with the secret input  $c$  chosen in the first place. However, the situation gets complicated when either of the players could potentially be malicious. Note that the two-output protocols we consider are not *fair* since  $P_2$  may always learn its own output and refuse to send  $P_1$ 's output. However, they can satisfy the notion that if  $P_1$  accepts output, it will be correctly computed.

*Related work* One straightforward solution is for the players to run the single-output protocol twice with roles reversed. Care must be taken to ensure that the same inputs are used in both executions. Also, this approach doubles the computation and communication cost. Other simple methods to handle two-output functions also have subtle problems. Suppose, for example,  $P_1$  encrypts all copies of her output and has  $P_2$  send these  $s$  random strings (or encryptions) in the last message. In a cut-and-choose framework, however, a cheating  $P_1$  can use these random strings to send back information about the internal state of the computation and thereby violate  $P_2$ 's privacy. As an example, the cheating  $P_1$  can make one bad circuit in which  $P_1$ 's output bit is equal to  $P_2$ 's first input bit. If  $P_2$  sends all copies of  $P_1$ 's output bit back to  $P_1$ , then with noticeable probability, the cheating  $P_1$  can learn  $P_2$ 's first input bit. The problem remains if instead of sending back all bits, only a randomly chosen output bit is sent. Besides,  $P_1$  should not be convinced by a cheating  $P_2$  with an arbitrary output.

As described in [13], the two-output case can be reduced to the single-output case as follows: (1)  $P_1$  randomly samples  $a, b, c \in \{0, 1\}^n$  as extra input; (2) the original function is converted into  $f^*((x, a, b, c), y) = (\lambda, (\alpha, \beta, f_2))$  where  $\alpha = f_1 \oplus c$  is an encryption of  $f_1$  and  $\beta = a \cdot \alpha + b$  is the Message Authentication code (MAC) of  $\alpha$ , and (3)  $P_2$  sends  $(\alpha, \beta)$  back to  $P_1$ , who can then check the

<sup>3</sup> Here  $f_1$  and  $f_2$  are abbreviations of  $f_1(x, y)$  and  $f_2(x, y)$  for simplicity purpose.

authenticity of the output  $\alpha = f_1 \oplus c$ . However, this transformation increases the size of  $P_1$ 's input from  $n$  bits to  $4n$  bits. As a result, the complexity of  $P_1$ 's input consistency check is also increased. A second drawback is that the circuit must also be modified to include extra gates for computing the encryption and MAC function. Although a recent technique [12] can be used to implement XOR gates “for free,” the MAC function  $a \cdot \alpha + b$  still requires approximately  $O(n^2)$  extra gates added to the circuit. Since all  $s$  copies of the circuit have to be modified, this results in additional communication of  $O(sn^2)$  encrypted gates. Indeed, for simple functions, the size of this overhead exceeds the size of the original circuit.

Kiraz and Schoenmakers [11] present a fair two-party computation protocol in which a similar issue for two-output functions arises. In their approach,  $P_2$  commits to  $P_1$ 's garbled output. Then  $P_1$  reveals the two output keys for each of her output wires, and  $P_2$  finds one circuit  $GC_r$  which agrees with “the majority output for  $P_1$ .” The index  $r$  is then revealed to  $P_1$ . However, informing  $P_1$  the index of the majority circuit could possibly leak information about  $P_2$ 's input. As an anonymous reviewer has brought to our attention an unpublished follow-up work from Kiraz [9], which elaborated this issue (in § 6.6 of [9]) and further fixed the problem without affecting the overall performance. Particularly, in the new solution, the dominant computational overhead is an OR-proof of size  $O(s)$ , and the dominant communicational overhead is the commitments to  $P_1$  output keys, where the number of such commitments is of order  $O(ns)$ . Their techniques favorably compare to our approach, but we do not have experimental data to make accurate comparisons with our implementation.

*Our approach to two-output functions* We present a method to evaluate two-output function  $f$  without adding non-XOR gates to the original circuit for  $f$ .

In order for  $P_2$  to choose one output that agrees with the majority, similar to Kiraz and Schoenmakers' approach in [11], we add extra bits to  $P_1$ 's input as a one-time pad encryption key by changing the function from  $f(x, y) = (f_1, f_2)$  to  $f^*(c, x, y) = (\lambda, (f_1 \oplus c, f_2))$ , where  $x, c, y, f_1, f_2 \in \{0, 1\}^n$ . With this extra random input  $c$  from  $P_1$ ,  $P_2$  is able to do the majority function on the evaluation output  $f_1 \oplus c$  without knowing  $P_1$ 's real output  $f_1$ . Next,  $P_2$  needs to prove the authenticity of the evaluation output  $f_1 \oplus c$  that she has given to  $P_1$ . Here, our idea is that  $P_1$ 's  $i$ -th output gate in the  $j$ -th garbled circuit is modified to output  $0 \parallel \sigma_{sk}(0, i, j)$  or  $1 \parallel \sigma_{sk}(1, i, j)$  instead of 0 or 1, where  $\sigma_{sk}(b, i, j)$  is a signature of the message  $(b, i, j)$  signed by  $P_1$  under the signing key  $sk$ . In other words, the garbled gate outputs  $P_1$ 's output bit  $b$  and a signature of  $b$ , bit index  $i$ , and circuit index  $j$ . Therefore, after the circuit evaluation,  $P_2$  hands  $f_1 \oplus c$  to  $P_1$  and proves the knowledge of the signature of each bit under the condition that the  $j$ -index for all signatures are the same and valid (among the indices of the evaluation-circuits). Naively, this proof would have been a proof of  $O(ns)$  group elements. However, we will show that a witness indistinguishable proof suffices, which reduces the complexity by a constant factor. Furthermore, by using the technique of Camenisch, Chaabouni, and Shelat for efficient set membership proof [4], we are able to reduce the complexity to  $O(n + s)$  group elements.

### 1.3 The problem of Selective Failure

Another problem with compiling garbled circuits occurs during the Oblivious Transfer (OT) phase, when  $P_2$  retrieves input keys for the garbled circuits. A malicious  $P_1$  can attack the protocol with *selective failure*, where the keys used to construct the garbled circuit might not be the ones used in the OT so that  $P_2$ 's input can be inferred according to her reaction after OT. For example, a cheating  $P_1$  could use  $(K_0, K_1)$  to construct a garbled circuit but use  $(K_0, K_1^*)$  instead in the corresponding OT, where  $K_1 \neq K_1^*$ . As a result, if  $P_2$ 's input bit is 1, she will get  $K_1^*$  after OT and cannot evaluate the garbled circuit properly. In contrast, if her input bit is 0,  $P_2$  will get  $K_0$  from OT and complete the evaluation without complaints.  $P_1$  can therefore infer  $P_2$ 's input. This issue is identified by both Mohassel and Franklin [16] and Kiraz and Schoenmakers [10].

*Related work* Lindell and Pinkas [13] replace each of  $P_2$ 's input bits with  $s$  additional input bits. These  $s$  new bits are XOR'ed together, and the result is used as the input to the original circuit. Such an approach makes the probability that  $P_2$  must abort due to selective failure independent of her input. This approach, however, increases the number of input bits for  $P_2$  from  $n$  to  $ns$ . Woodruff later pointed out that the use of clever coding system can reduce the overhead to  $\max(4n, 8s)$ . To be sure, Lindell, Pinkas, and Smart [15] implement the method described in [13] and empirically confirm the extra overhead from this step. In particular, a 16-bit comparison circuit that originally needs fifteen 3-to-1 gates and one 2-to-1 gate will be inflated to a circuit of several thousand gates after increasing the number of inputs. Since the number of inputs determines the number of OT operations, an approach that keeps the number of extra inputs small is preferable. In fact, we show that increasing the number of inputs and number of gates in the circuit for this problem is unnecessary.

Independent of our work, Lindell and Pinkas [14] propose to solve this problem by *cut-and-choose OT*. This new solution indeed provides a great improvement over [13] and shares roughly the same complexity with our solution. Furthermore, both the cut-and-choose OT and our solution can be built upon the efficient OT proposed by Naor and Pinkas [17] or Peikert, Vaikuntanathan, and Waters [19]. However, the particular use the latter OT in [14] needs two independently chosen common reference strings, while our solution needs only one.

*Our approach to selective failure* Inspired by the idea of *committing Oblivious Transfer* proposed by Kiraz and Schoenmakers [10], we solve the problem of selective failure by having the sender ( $P_1$  in Yao protocol) of the OT *post-facto* prove that she ran the OT correctly by revealing the randomness used in the OT. Normally, this would break the sender-security of the OT. However, in a cut-and-choose framework, the sender is already opening many circuits, so the keys used as inputs for the OT are no longer secret. Thus, the idea is that the sender can prove that he executed the OT correctly for all circuits that are opened by simply sending the random coins used in the OT protocol for those instances. We stress that not every OT can be used here. Intuitively, a committing OT

is the OT with the binding property so that it is hard for a cheating sender to produce random coins different from what she really used.

A critical point with this approach is that in order to simulate a malicious  $P_2$ , we need to use a coin-flipping protocol to pick which circuits to open. Consequently,  $P_1$  cannot open the circuits to  $P_2$  until the coin-flipping is over; yet the OT must be done before the coin-flipping in order to guarantee a proper *cut*. So the order of operations of the protocol is critical to security. An efficient committing OT based on Decisional Diffie-Hellman problem is presented in §2.3.

#### 1.4 Optimal Cut-and-Choose Strategy

We find that most cut-and-choose protocols open  $s/2$  out of the  $s$  copies of the garbled circuit to reduce the probability that  $P_1$  succeeds in cheating. We show that opening  $3s/5$ -out-of- $s$  is a better choice than  $s/2$ -out-of- $s$ . In particular, when  $s$  circuits are used, our strategy results in security level  $2^{-0.32s}$  in contrast to  $2^{-s/17}$  from [13] and  $2^{-0.31s}$  from [14]. Although the difference with the latter work is only 1% less, we show the optimal parameters for the cut-and-choose method in Appendix A, thereby establishing a close characterization of the limits of the cut-and-choose method.

#### 1.5 Comparison of Communication Complexity

We attempt to compare 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). To meaningfully do so, we consider asymptotic security under reasonable assumptions about the growth of various primitives with respect to the security parameter  $k$ . We assume that:

1. light cryptographic primitives have size  $\Theta(k)$ ;
2. heavy cryptographic operations that can be instantiated with elliptic curves or bilinear groups take size  $\tilde{o}(k^2)$ .
3. heavy cryptographic operations that require RSA or prime order groups over  $\mathbb{Z}$  take size  $\tilde{o}(k^3)$ .

The size assumption we make is quite conservative. It is based on the observation that in certain elliptic curve groups, known methods for computing discrete logarithms of size  $n$  run in time  $L_n(1, 1/2)$ . 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  $\log(k)$ . The computation bound follows from the running time analysis of point multiplication (or exponentiation in the case of  $\mathbb{Z}_p^*$ ) algorithms. As we discuss below, for reasonable security parameters, however, the hidden constants in this notation make the difference much smaller. Let  $k$  be a security parameter for cryptographic operations, let  $s$  be a statistical security parameter, and let  $|C|$  be the number of gates in the base circuit computing  $f : \{0, 1\}^n \times \{0, 1\}^n \rightarrow \{0, 1\}^n \times \{0, 1\}^n$ .

- Jarecki and Shmatikov [8]: For each gate, the number of the communicated group elements is at least 100, including the commitments of the garbled values for input wires, the commitments of the doubly-encrypted entries, and the ZK proof for the correctness of the gate. Moreover, for each input or output wires, a ZK proof for conjunction/disjunction is required. Each of the ZK proofs needs constant number of group elements. Finally, this protocol assumes the decisional composite residuosity problem in an RSA group; thus, each group element is of size  $\tilde{o}(k^3)$ .
- Kiraz [9]: This approach uses an equality-checker framework that requires  $O(ns^2)$  commitments for checking  $P_1$ 's input consistency. They solve the selective failure attack with committing OT as we do. Moreover, to deal with two-output functions, they add  $n$  extra bits to  $P_1$ 's input, commit to all of  $P_1$ 's output keys, which include  $2ns$  commitments and  $2ns$  decommitments, and a zero-knowledge OR-proof of size  $O(s)$ .
- Lindell and Pinkas [13]: Each of the garbled gates requires  $4k$  space for four doubly-encrypted entries. Thus, for this approach, the communication analysis is as follows: (1)  $s$  copies of the base circuit itself require  $s|C|$  gates; (2) each of  $P_1$ 's  $n$  input bits requires  $s^2$  light commitments for the consistency check; (3)  $P_2$ 's  $n$  input bits require  $\max(4n, 8s)$  OT's. Also, the MAC-based two-output function computation add additional  $O(n^2)$  gates to each of the  $s$  copies of the circuit and additional  $3n$  bits to  $P_1$ 's input. Thus, the overall communication cost to handle two-output function is  $O(n^2sk + ns^2k)$ .

	Base circuit	$P_1$ 's input	COMMUNICATION	
			$P_2$ 's input	Two-output
JS [8]	$ C  \cdot \tilde{o}(k^3)$	$n \cdot \tilde{o}(k^3)$	$n$ OT's	$n \cdot \tilde{o}(k^3)$
K [9]	$\Theta( C  \cdot sk)$	$\Theta(ns^2k)$	$n$ OT's	$\Theta(nsk) + \Theta(s) \cdot \tilde{o}(k^2)$
LP07 [13]	$\Theta( C  \cdot sk)$	$\Theta(ns^2k)$	$\max(4n, 8s)$ OT's	$\Theta(n^2sk + ns^2k)$
LP10 [14]	$\Theta( C  \cdot sk)$	$\Theta(ns) \cdot \tilde{o}(k^2)$	$n$ OT's	$\Theta(n^2sk + ns^2k)$
Our work	$\Theta( C  \cdot sk)$	$\Theta(ns) \cdot \tilde{o}(k^2)$	$n$ OT's	$\Theta(ns) \cdot \tilde{o}(k^2)$

Table 1: Asymptotic Analysis of various two-party secure computation.

The recent work of [14] also considers a more efficient way to implement two-party computation based on cut-and-choose OT and specific security assumptions. They report  $13sn$  exponentiations and communication of  $5sn + 14k + 7n$  group elements. (Note we count bits above to compare commitments versus other primitives.) Concretely, these parameters are similar to our parameters but rely on more specific assumptions, and do not consider two-party outputs.



## 2 Building Blocks

For clarity purpose, the standard checks that are required for security have been omitted. For example, in many cases, it is necessary to verify that an element that has been sent is indeed a member of the right group. In some cases, it is implicit that if a player detectably cheats in a sub-protocol, then the other player would immediately abort execution of the entire protocol.

### 2.1 Consistency Check for the Generator’s Input

The cut-and-choose approach to compiling Yao circuits ensures that  $P_1$  submits consistent input values for each copy of the evaluation-circuits. Recall that there are  $e$  copies of the circuit which must be evaluated. Thus, for each input wire,  $P_1$  must send  $e$  keys corresponding to an input bit 0 or 1. It has been well-documented [16,10,22,13] that in some circumstances,  $P_1$  can gain information about  $P_2$ ’s input if  $P_1$  is able to submit different input values for the  $e$  copies of this input wire. The main idea of our solution is inspired by the *claw-free collections*<sup>4</sup> defined as follows:

**Definition 1 (Claw-Free Collections in [7]).** *A three-tuple of algorithms  $(G, D, F)$  is called a claw-free collection if the following conditions hold*

1. **Easy to evaluate:** *Both the index selecting algorithm  $G$  and the domain sampling algorithm  $D$  are probabilistic polynomial-time, while the evaluating algorithm  $F$  is a deterministic polynomial-time.*
2. **Identical range distribution:** *Let  $f_I^b(x)$  denote the output of  $F$  on input  $(b, I, x)$ . For any  $I$  in the range of  $G$ , the random variable  $f_I^0(D(0, I))$  and  $f_I^1(D(1, I))$  are identically distributed.*
3. **Hard to form claws:** *For every non-uniform probabilistic polynomial-time algorithm  $A$ , every polynomial  $p(\cdot)$ , and every sufficiently large  $n$ ’s, it is true that  $\Pr[I \leftarrow G(1^n); (x, y) \leftarrow A(I) : f_I^0(x) = f_I^1(y)] < 1/p(n)$ .*

With the claw-free collections, our idea works as follows:  $P_2$  first generates  $I$  by invoking the index generating algorithm  $G(1^k)$ , where  $k$  is a security parameter. For each of her input bits,  $P_1$  invokes sampling algorithms  $D(I, 0)$  and  $D(I, 1)$  to pick  $[m_{0,1}, \dots, m_{0,s}]$  and  $[m_{1,1}, \dots, m_{1,s}]$ , respectively.  $P_1$  then constructs  $s$  copies of garbled circuit with range elements  $[f_I^0(m_{0,1}), \dots, f_I^0(m_{0,s})]$  and  $[f_I^1(m_{1,1}), \dots, f_I^1(m_{1,s})]$  by associating  $f_I^b(m_{b,j})$  with  $P_1$ ’s input wire of bit value  $b$  in the  $j$ -th garbled circuit. Let  $[j_1, \dots, j_e]$  denote the indices of the garbled circuits not checked in the cut-and-choose (evaluation-circuits). During the evaluation,  $P_1$  reveals  $[f_I^b(m_{b,j_1}), \dots, f_I^b(m_{b,j_e})]$  to  $P_2$  and proves in zero-knowledge that  $P_1$  gets  $f_I^b(m_{j_1}^b)$  and  $f_I^b(m_{j_i}^b)$  via the same function  $f_I^b$ , for  $2 \leq i \leq e$ .

However, in the course of developing our solution, we noticed that witness indistinguishable proofs suffice in place of zero-knowledge proofs. For example,

<sup>4</sup> It is well known that claw-free collections exist under either the Discrete Logarithm assumption or Integer Factorization assumption [7].

consider the claw-free collection instantiated from the Discrete Logarithm assumption, that is, let  $f_I^b(m) = g^b h^m$ , where  $I = (g, h, p, q)$  includes two primes  $p$  and  $q$  such that  $p = 2q + 1$ , and distinct generators  $g$  and  $h$  of  $\mathbb{Z}_p^*$  such that  $\langle g \rangle = \langle h \rangle = q$ . After revealing  $[g^b h^{m_{b,j_1}}, \dots, g^b h^{m_{b,j_e}}]$  to  $P_2$ , it is a natural solution that  $P_1$  proves in zero-knowledge to  $P_2$  the knowledge of  $(m_{b,j_i} - m_{b,j_1})$  given common input  $g^b h^{m_{b,j_i}} (g^b h^{m_{b,j_1}})^{-1} = h^{m_{b,j_i} - m_{b,j_1}}$ , for  $2 \leq i \leq e$ . The key insight here is that it is unnecessary for  $P_1$  to hide  $(m_{b,j_i} - m_{b,j_1})$  from  $P_2$  since  $[m_{b,j_1}, \dots, m_{b,j_e}]$  are new random variables introduced by  $P_1$  and  $b$  is the only secret needed to be hidden from  $P_2$ . Simply sending  $(m_{b,j_i} - m_{b,j_1})$  to  $P_2$  will suffice a proof of checking  $P_1$ 's input consistency without compromising  $P_1$ 's privacy. In other words, given  $[g^b h^{m_{b,j_1}}, \dots, g^b h^{m_{b,j_e}}, m'_2, \dots, m'_e]$ , if  $P_2$  confirms that  $g^b h^{m_{b,j_1}} = g^b h^{m_{b,j_i}} \cdot h^{m'_i}$  for  $2 \leq i \leq e$ , then either  $P_1$ 's input is consistent so that  $m'_i = m_{b,j_i} - m_{b,j_1}$ , or  $P_1$  is able to come up with a claw.

Note that extra work is only done for the input gates—and moreover, only those of  $P_1$ . All of the remaining gates in the circuit are generated as usual, that is, they do not incur extra commitments. So, unlike solutions with committed OT such as [8], asymmetric cryptography is only used for the input gates rather than the entire circuit. To generalize the idea, we introduce the following notion.

**Definition 2 (Malleable Claw-Free Collections).** *A four-tuple of algorithms  $(G, D, F, R)$  is a malleable claw-free collection if the following conditions hold.*

1. **A subset of claw-free collections:**  $(G, D, F)$  is a claw-free collection, and the range of  $D$  and  $F$  are groups, denoted by  $(\mathbb{G}_1, \star)$  and  $(\mathbb{G}_2, \diamond)$  respectively.
2. **Uniform domain sampling:** For any  $I$  in the range of  $G$ , random variable  $D(0, I)$  and  $D(1, I)$  are uniform over  $\mathbb{G}_1$ , and denoted by  $D(I)$  for simplicity.
3. **Malleability:**  $R : \mathbb{G}_1 \rightarrow \mathbb{G}_2$  runs in polynomial time, and for  $b \in \{0, 1\}$ , any  $I$  in the range of  $G$ , and any  $m_1, m_2 \in \mathbb{G}_1$ ,  $f_I^b(m_1 \star m_2) = f_I^b(m_1) \diamond R_I(m_2)$ .

Consider the claw-free collection constructed above under the Discrete Logarithm assumption, we know that it can become a malleable claw-free collection simply by letting  $\mathbb{G}_1 = \mathbb{Z}_q$ ,  $\mathbb{G}_2 = \mathbb{Z}_p^*$ , and  $R_I(m) = h^m$  for any  $m \in \mathbb{G}_1$ .

## 2.2 Two-Output Functions

To handle two-output functions, we want to satisfy the notion that it might be unfair in the sense that  $P_2$  could abort prematurely after circuit evaluation and she gets her output. However, if  $P_1$  accepts the output given from  $P_2$ , our approach guarantees that this output is genuine. Namely,  $P_2$  cannot provide an arbitrary value to be  $P_1$ 's output. In particular,  $P_2$  cannot learn  $P_1$ 's output more than those deduced from  $P_2$ 's own input and output.

Recall that it is a well-accepted solution to convert the garbled circuit computing  $f(x, y) = (f_1, f_2)$  into the one computing  $g((x, p, a, b), y) = ((\alpha, \beta), f_2)$ , where  $\alpha = f_1 + p$  as a ciphertext of  $f_1$  and  $\beta = a \cdot \alpha + b$  as a MAC for the ciphertext. Since  $P_2$  only gets the ciphertext of  $P_1$ 's output, she does not learn anything from the ciphertext. Also, given  $(\alpha, \beta)$ ,  $P_1$  can easily verify the authenticity of her output. However, we are not satisfied with the additional  $O(s^2)$

gates computing the MAC ( $s$  is the statistical security parameter) to each of the  $s$  copies of the garbled circuit, which results in  $O(s^3)$  extra garbled gates in total. Indeed, the number of extra gates can easily exceed the size of the original circuit when  $f$  is a simple function. Hence, we propose another approach to authenticate  $P_1$ 's output without the extra gates computing the MAC function.

While our approach also converts the circuit to output the ciphertext of  $P_1$ 's output, that is, from  $f(x, y) = (f_1, f_2)$  to  $f^*((c, x), y) = (\lambda, (f_1 \oplus c, f_2))$ , we solve the authentication problem by the use of the public-key signature scheme and its corresponding witness-indistinguishable proof. Each bit value of the output of  $P_1$ 's output gates is tied together with a signature specifying the value and the location of the bit. On one hand,  $P_2$  can easily verify the signature during the cut-and-choose phase (to confirm that the circuits are correctly constructed). On the other hand, after the evaluation and giving  $P_1$  the evaluation result  $(f_1 \oplus c)$ ,  $P_2$  can show the authenticity of each bit of the result by proving the knowledge of its signature, that is, the signature of the given bit value from the right bit location. Note that a bit location includes a bit index and a circuit index. In other words, a bit location  $(i, j)$  indicates  $P_1$ 's  $i$ -th output bit from the  $j$ -th garbled circuit. While the bit index is free to reveal (since  $P_1$  and  $P_2$  have to conduct the proof bit by bit anyway), the circuit index needs to be hidden from  $P_1$ ; otherwise,  $P_1$  can gain information about  $P_2$ 's input as we discussed above. We stress that it is critical for  $P_2$  to provide a signature from the right location. Since during the cut-and-choose phase, many properly signed signatures are revealed from the check-circuits, if those signatures do not contain location information, they can be used to convince  $P_1$  to accept arbitrary output.

Normally, an OR-proof will suffice the proof that the signature is from one of the evaluation-circuits. Nevertheless, an OR proof of size  $O(s)$  for each bit of  $P_1$ 's  $n$ -bit output will result in a zero-knowledge proof of size  $O(ns)$ . We therefore adopt the technique from [4] in order to reduce the size of the proof to  $O(n + s)$ . Let  $S = \{j_1, \dots, j_e\}$  be the indices of all the evaluation-circuits. The idea is for  $P_1$  to send a signature of every element in  $S$ , denoted by  $[\delta(j_1), \dots, \delta(j_e)]$ . By reusing these signatures,  $P_2$  is able to perform each OR proof in constant communication. More specifically, after the evaluation,  $P_2$  chooses one evaluation-circuit, say the  $j_l$ -th circuit, the result of which conforms with the majority of all the evaluation-circuits. Let  $\mathcal{M} = [M_1, \dots, M_n]$  be  $P_1$ 's output from the  $j_l$ -th circuit. Recall that  $P_2$  has both  $M_i$  and the signature to  $(M_i, i, j_l)$ , denoted by  $\sigma(M_i, i, j)$ , due to the way the garbled circuits were constructed. To prove the authenticity of  $M_i$ ,  $P_2$  sends  $M_i$  to  $P_1$ , blinds signature  $\delta(j_l)$  and  $\sigma(M_i, i, j_l)$ , and proves the knowledge of " $\sigma(M_i, i, j)$  for some  $j \in S$ ." In other words,  $P_2$  needs to prove the knowledge of  $\sigma(M_i, i, j)$  and  $\delta(j^*)$  such that  $j = j^*$  for  $i = 1, \dots, n$ . The complete proof is shown in Protocol 1. Due to the nonforgeability property of signature schemes,  $P_2$  proves the knowledge of the signature and thus the authenticity of  $\mathcal{M}$ .

One particular implementation of our protocol can use the Boneh-Boyen short signature scheme [2] which is briefly summarized here. The Boneh-Boyen

signature scheme requires the  $q$ -SDH (*Strong Diffie-Hellman*) assumption<sup>5</sup> and *bilinear maps*<sup>6</sup>. Based on these two objects, the Boneh-Boyen signature scheme includes a three-tuple of efficient algorithms  $(G, S, V)$  such that

1.  $G(1^k)$  generates key pair  $(sk, vk)$  such that  $sk = x \in \mathbb{Z}_p^*$  and  $vk = (p, g, \mathbb{G}_1, X)$ , where  $\mathbb{G}_1$  is a group of prime order  $p$ ,  $g$  is a generator of  $\mathbb{G}_1$ , and  $X = g^x$ .
2.  $S(sk, m)$  signs the message  $m$  with the signing key  $sk$  by  $\sigma(m) = g^{1/(x+m)}$ .
3.  $V(vk, m, \sigma)$  verifies the signature  $\sigma$  with  $vk$  by calculating  $e(\sigma, g^m X)$ . If the result equals  $e(g, g)$ ,  $V$  outputs **valid**; otherwise,  $V$  outputs **invalid**.

### Protocol 1: Proof of $P_1$ 's output authenticity

---

**Common Input:** ciphertext of  $P_1$ 's output  $f_1 \oplus c = [M_1, \dots, M_n]$ , the indices of the evaluation-circuits  $S = \{j_1, \dots, j_e\}$  and the public key  $(p, \mathbb{G}, g, X, Y)$  of the Boneh-Boyen signature scheme. In particular,  $X = g^x$ , and  $Y = g^y$ .

$P_1$  **Input:** the corresponding private key  $(x, y)$  of the signature scheme.

$P_2$  **Input:** the signature vector  $[\sigma(b_1, 1, j_1), \dots, \sigma(b_n, n, j_i)]$  such that  $\sigma(b, i, j) = g^{1/(bx+iy+j)}$  and  $j_i \in S$ .

$P_1 \xrightarrow{Z, \{\delta(j)\}_{j \in S}} P_2$   $P_1$  picks another generator  $h$  of  $G$  and a random  $z \in \mathbb{Z}_p^*$ . Then  $P_1$  sends  $[Z, \delta(j_1), \dots, \delta(j_e)]$  to  $P_2$  such that

$$Z = h^z \text{ and } \delta(j) = h^{1/(z+j)}.$$

$P_1 \xleftarrow{U_1, \dots, U_n, V} P_2$   $P_2$  picks  $u_1, \dots, u_n, v \in \mathbb{Z}_p$  and computes  $U_i \leftarrow \sigma(b_i, i, j_i)^{u_i}$  and  $V \leftarrow \delta(j_i)^v$ . Then  $[U_1, \dots, U_n, V]$  is sent to  $P_1$ .

$P_1 \xleftarrow{a_1, \dots, a_n, b} P_2$   $P_2$  picks  $\alpha, \beta_1, \dots, \beta_n, \gamma \in \mathbb{Z}_p$  and sends  $[a_1, \dots, a_n, b]$  to  $P_1$ , where  $a_i \leftarrow e(U_i, g)^\alpha e(g, g)^{\beta_i}$  and  $b \leftarrow e(V, h)^\alpha e(h, h)^\gamma$ .

$P_1 \xrightarrow{c} P_2$   $P_1$  picks  $c \in \mathbb{Z}_p$  at random and sends it to  $P_2$ .

$P_1 \xleftarrow{z_\alpha, \{z_{\beta_i}\}, z_\gamma} P_2$   $P_2$  sends  $z_\alpha \leftarrow \alpha + c \cdot j_i$ ,  $z_{\beta_i} \leftarrow \beta_i - c \cdot u_i$ , and  $z_\gamma \leftarrow \gamma - c \cdot v$  back to  $P_1$ , who checks  $a_i \stackrel{?}{=} e(U_i, X^{M_i} Y^i)^c \cdot e(U_i, g)^{z_\alpha} \cdot e(g, g)^{z_{\beta_i}}$  for  $i = 1, \dots, n$  and  $b \stackrel{?}{=} e(V, Z)^c \cdot e(V, h)^{z_\alpha} \cdot e(h, h)^{z_\gamma}$ .  $P_1$  aborts if any of the checks fails.

---

## 2.3 Committing Oblivious Transfer

The oblivious transfer (OT) primitive, introduced by Rabin [21], and extended by Even, Goldreich, and Lempel [5] and Brassard, Crépeau and Robert [3] works

<sup>5</sup>  $q$ -SDF assumption in a group  $G$  of prime order  $p$  states that given  $g, g^x, g^{x^2}, \dots, g^{x^q}$ , it is infeasible to output a pair  $(c, g^{1/(x+c)})$  where  $c \in \mathbb{Z}_p^*$ .

<sup>6</sup> Let  $\mathbb{G}_1$  and  $\mathbb{G}_2$  be two groups of prime order  $p$ . A bilinear map is a map  $e : \mathbb{G}_1 \times \mathbb{G}_1 \mapsto \mathbb{G}_2$  with the following properties: (1) for any  $u, v \in \mathbb{G}_1$  and  $a, b \in \mathbb{Z}$ ,  $e(u^a, v^b) = e(u, v)^{ab}$ ; (2) for any generator  $g$  of  $\mathbb{G}_1$ ,  $e(g, g) \neq 1$ ; and (3) for any  $u, v \in \mathbb{G}_1$ , it is easy to compute  $e(u, v)$ .

as follows: there is a sender with messages  $[m_1, \dots, m_n]$  and a receiver with a selection value  $\sigma \in \{1, \dots, n\}$ . The receiver wishes to retrieve  $m_\sigma$  from the sender in such a way that (1) the sender does not “learn” anything about the receiver’s choice  $\sigma$  and (2) the receiver “learns” only  $m_\sigma$  and nothing about any other message  $m_i$  for  $i \neq \sigma$ . Kiraz and Schoenmakers [10] introduced another notion of OT called *committing OT* in which the receiver also receives a perfectly-hiding and computationally-binding commitment to the sender’s input messages, and the sender receives as output the values to open the commitment. Indeed, Kiraz and Schoenmakers introduced this notion specifically for use in a Yao circuit evaluation context. We adopt the idea behind their construction.

Formally, a one-out-of-two committing oblivious transfer  $OT_1^2$  is a pair of interactive probabilistic polynomial-time algorithms sender and receiver. During the protocol, the sender runs with input messages  $((m_0, r_0), (m_1, r_1))$ , while the receiver runs with input the index  $\sigma \in \{0, 1\}$  of the message it wishes to receive. At the end of the protocol, the receiver outputs the retrieved message  $m'_\sigma$  and two commitments  $\text{com}_H(m_0; r_0), \text{com}_H(m_1; r_1)$ , and the sender outputs the openings  $(r_0, r_1)$  to these commitments. Correctness requires that  $m'_\sigma = m_\sigma$  for all messages  $m_0, m_1$ , for all selections  $\sigma \in \{0, 1\}$  and for all coin tosses of the algorithms. Here, we use the standard notion of simulation security.

**Theorem 1.** [19] *If the Decisional Diffie-Hellman assumption holds in group  $G$ , there exists a protocol that securely computes the committing  $OT_1^2$ .*

Protocol 2 constructively proves Theorem 1. This protocol is a simple modification of the OT protocols designed by Peikert, Vaikuntanathan, and Waters [19] and later Lindell and Pinkas [14]. We simply add a ZK proof of knowledge in intermediate steps. Intuitively, the receiver-security is achieved due to the Decisional Diffie-Hellman assumption and the fact that the ZK proof of knowledge is independent of the receiver’s input. On the other hand, the sender security comes from the uniform distributions of  $X_{i,j}$  and  $Y_{i,j}$  over  $G$  given that  $r_{i,j}$  and  $s_{i,j}$  are uniformly chosen and that the ZK proof has an ideal-world simulator for the verifier (or the receiver in the OT). As described in [15], it is possible to batch the oblivious transfer operations so that all  $n$  input keys (one for each bit) to  $s$  copies of the garbled circuit are transferred in one execution.

### 3 Main Protocol

Here we put all the pieces together to form the complete protocol. Note that  $\text{com}_H(K; t)$  denotes a perfectly-hiding commitment to  $K$  with opening  $t$ , and  $\text{com}_B(K; t)$  denotes a perfectly-binding commitment to  $K$  with opening  $t$ .

**Common input:** a security parameters  $k$ , a statistical security parameter  $s$ , a malleable claw-free collection  $(G_{\text{CLW}}, D_{\text{CLW}}, F_{\text{CLW}}, R_{\text{CLW}})$ , a signature scheme  $(G_{\text{SIG}}, S_{\text{SIG}}, V_{\text{SIG}})$ , a two-universal hash function  $H : \{0, 1\}^* \rightarrow \{0, 1\}^k$ , and the description of a boolean circuit  $C$  computing  $f(x, y) = (f_1, f_2)$ , where  $|x| = 2n$  (including the extra  $n$ -bit random input) and  $|y| = |f_1| = |f_2| = n$ .

Protocol 2: Oblivious transfer for retrieving  $P_2$ 's input keys [14]

---

<b>Common:</b>	A statistical security parameter $s$ , a group $G$ of prime order $p$ , and $G$ 's generator $g_0$
$P_1$ <b>Input:</b>	Two $s$ -tuples $[K_{0,1}, \dots, K_{0,s}]$ and $[K_{1,1}, \dots, K_{1,s}]$ .
$P_2$ <b>Input:</b>	$\sigma \in \{0, 1\}$
$P_1$ <b>Output:</b>	Commitment openings $\{K_{i,j}, r_{i,j}, s_{i,j}\}_{i \in \{0,1\}, 1 \leq j \leq s}$
$P_2$ <b>Output:</b>	$[K_{\sigma,1}, \dots, K_{\sigma,s}]$ and $\{\text{com}_H(K_{i,j}; r_{i,j}, s_{i,j})\}_{i \in \{0,1\}, 1 \leq j \leq s}$
$P_1 \xleftarrow{h_0, g_1, h_1} P_2$	$P_2$ picks $y, a \in \mathbb{Z}_p$ and sends $(g_1, h_0, h_1) \leftarrow (g_0^y, g_0^a, g_1^{a+1})$ to $P_1$ .
$P_1 \xleftrightarrow{\text{ZK PoK}} P_2$	$P_2$ proves that $(h_0, g_1, h_1)$ satisfies $(h_0 = g_0^a) \wedge (\frac{h_1}{g_1} = g_1^a)$ .
$P_1 \xleftarrow{g, h} P_2$	$P_2$ picks $r \in \mathbb{Z}_p$ and sends $g \leftarrow g_\sigma^r$ and $h \leftarrow h_\sigma^r$ to $P_1$ .
$P_1 \xrightarrow{\{X_{i,j}, Y_{i,j}\}} P_2$	For $i \in \{0, 1\}, 1 \leq j \leq s$ , $P_1$ picks $r_{i,j}, s_{i,j} \in \mathbb{Z}_p$ and sends $X_{i,j}$ and $Y_{i,j}$ to $P_1$ , where $X_{i,j} = g_i^{r_{i,j}} h_i^{s_{i,j}}$ and $Y_{i,j} = g^{r_{i,j}} h^{s_{i,j}} \cdot K_{i,j}$ . $P_2$ gets $\text{com}_H(K_{i,j}; r_{i,j}, s_{i,j}) = (X_{i,j}, Y_{i,j})$ and computes key $K_{\sigma,j} \leftarrow Y_{\sigma,j} \cdot X_{\sigma,j}^{-r}$ .

---

**Private input:**  $P_1$  has the original input  $x_1 \dots x_n$  and the extra random input  $x = x_{n+1} \dots x_{2n}$ , while  $P_2$  has input  $y = y_1 y_2 \dots y_n$ .

**Private output:**  $P_1$  receives output  $f_1(x, y)$ , while  $P_2$  receives output  $f_2(x, y)$ .

1.  $P_2$  runs the index selecting algorithm  $I \leftarrow G_{\text{CLW}}(1^k)$  and sends  $I$  to  $P_1$ .
2. **Committing OT for  $P_2$ 's input:** For every  $1 \leq i \leq n$  and every  $1 \leq j \leq s$ ,  $P_1$  picks a random pair of  $k$ -bit strings  $(K_{i,j}^0, K_{i,j}^1)$ , which is associated with  $P_2$ 's  $i$ -th input wire in the  $j$ -th circuit. Both parties then conduct  $n$  instances of committing OT in parallel. In the  $i$ -th instance,
  - (a)  $P_1$  uses input  $([K_{i,1}^0, \dots, K_{i,s}^0], [K_{i,1}^1, \dots, K_{i,s}^1])$ , whereas  $P_2$  uses input  $y_i$ .
  - (b)  $P_1$  gets the openings  $([t_{i,1}^0, \dots, t_{i,s}^0], [t_{i,1}^1, \dots, t_{i,s}^1])$  to both commitment vectors, whereas  $P_2$  gets the vector of her choice  $[K_{i,1}^{y_i}, \dots, K_{i,s}^{y_i}]$  and the commitments to both vectors, ie.,  $[\text{com}_H(K_{i,1}^0; t_{i,1}^0), \dots, \text{com}_H(K_{i,s}^0; t_{i,s}^0)]$  and  $[\text{com}_H(K_{i,1}^1; t_{i,1}^1), \dots, \text{com}_H(K_{i,s}^1; t_{i,s}^1)]$ .
3. **Garbled circuit construction:**  $P_1$  runs the key generating algorithm  $G_{\text{SIG}}(1^k)$  to generate a signature key pair  $(sk_1, pk_1)$  and the domain sampling algorithm  $D_{\text{CLW}}(I)$  to generate domain element  $m_{i,j}^b$ , for  $b \in \{0, 1\}, 1 \leq i \leq 2n, 1 \leq j \leq s$ . Next,  $P_1$  constructs  $s$  independent copies of garbled version of  $C$ , denoted by  $GC_1, \dots, GC_s$ . In addition to Yao's construction, circuit  $GC_j$  also satisfies the following:
  - (a)  $J_{i,j}^b$  is associated with value  $b$  to  $P_1$ 's  $i$ -th input wire, where  $J_{i,j}^b$  is extracted from group element  $F_{\text{CLW}}(b, I, m_{i,j}^b)$ , ie.,  $J_{i,j}^b = H(F_{\text{CLW}}(b, I, m_{i,j}^b))$ .
  - (b)  $K_{i,j}^b$  chosen in Step 2 is associated with value  $b$  to  $P_2$ 's  $i$ -th input wire.
  - (c)  $b || \mathcal{S}_{\text{SIG}}(sk_1, (b, i, j))$  is associated with bit value  $b$  to  $P_1$ 's  $i$ -th output wire.
4. For  $b \in \{0, 1\}, 1 \leq i \leq 2n, 1 \leq j \leq s$ ,  $P_1$  sends circuits  $GC_1, \dots, GC_s$  and the commitments to  $F_{\text{CLW}}(b, I, m_{i,j}^b)$ , denoted by  $\text{com}_B(F_{\text{CLW}}(b, I, m_{i,j}^b); r_{i,j}^b)$  to  $P_2$ .

5. **Cut-and-choose:**  $P_1$  and  $P_2$  conduct the coin flipping protocol to generate a random tape, by which they agree on a set of check-circuits. Let  $T$  be the resulting set, that is,  $T \subset \{1, \dots, s\}$  and  $|T| = 3s/5$ . For every  $j \in T$ ,  $P_1$  sends to  $P_2$   $P_1$ 's of garbled circuit  $GC_j$ , including  $[K_{1,j}^b, \dots, K_{n,j}^b]$ ,  $[t_{1,j}^b, \dots, t_{n,j}^b]$ ,  $[m_{1,j}^b, \dots, m_{2n,j}^b]$ ,  $[r_{1,j}^b, \dots, r_{2n,j}^b]$ , for  $b \in \{0, 1\}$ , and the random keys associated with each wire of  $GC_j$ .  $P_2$  check the following:
- (a) The commitment from Step 2 is revealed to  $K_{i,j}^b$  with  $t_{i,j}^b$ .
  - (b) The commitment from Step 4 is revealed to  $F_{\text{CLW}}(b, I, m_{i,j}^b)$  with  $r_{i,j}^b$ .
  - (c)  $GC_j$  is a garbled version of  $C^*$  that is correctly built. In particular,
    - $H(F_{\text{CLW}}(b, I, m_{i,j}^b))$  is associated with value  $b$  to  $P_1$ 's  $i$ -th input wire;
    - $K_{i,j}^b$  is associated with bit value  $b$  to  $P_2$ 's  $i$ -th input wire;
    - $V_{\text{SIG}}(pk_1, (b, i, j), \sigma(b, i, j)) = \text{valid}$ , where  $\sigma(b, i, j)$  is the signature comes along with bit value  $b$  from  $P_1$ 's  $i$ -th output wire;
    - the truth table of each boolean gate is correctly converted to the doubly-encrypted entries of the corresponding garbled gate.

If any of the above checks fails,  $P_2$  aborts.

6. **Consistency check for  $P_1$ 's inputs:** Let  $e = 2s/5$  and  $\{j_1, \dots, j_e\}$  be the indices of evaluation-circuits.  $P_1$  then decommits to her input keys for the evaluation-circuits by sending  $([r_{1,j_1}^{x_1}, \dots, r_{2n,j_1}^{x_{2n}}], \dots, [r_{1,j_e}^{x_1}, \dots, r_{2n,j_e}^{x_{2n}}])$  to  $P_2$ . Let  $[M_{1,j_1}, \dots, M_{2n,j_1}], \dots, [M_{1,j_e}, \dots, M_{2n,j_e}]$  be the resulting decommitments. Next,  $P_1$  proves the consistency of her  $i$ -th input bit by sending  $[m_{i,j_2}^{x_i} \star (m_{i,j_1}^{x_i})^{-1}, \dots, m_{i,j_e}^{x_i} \star (m_{i,j_1}^{x_i})^{-1}]$  to  $P_2$ , who then checks if

$$M_{i,j_l} = M_{i,j_1} \diamond R_{\text{CLW}}(I, m_{i,j_l}^{x_i} \star (m_{i,j_1}^{x_i})^{-1}), \text{ for } l = 2, \dots, e.$$

$P_2$  aborts if any of the checks fails. Otherwise, let  $J_{i,j_l}^{x_i} = H(M_{i,j_l})$ .

7. **Circuit evaluation:** For every  $j \in \{j_1, \dots, j_e\}$ ,  $P_2$  now has key vectors  $[J_{1,j}^{x_1}, \dots, J_{2n,j}^{x_{2n}}]$  (from Step 6) representing  $P_1$ 's input  $x$  and  $[K_{1,j}^{y_1}, \dots, K_{n,j}^{y_n}]$  (from Step 2) representing  $P_2$ 's input  $y$ . So  $P_2$  is able to do the evaluation on circuit  $GC_j$  and get  $P_1$ 's output  $[M_{1,j} \parallel \sigma(M_{1,j}), \dots, M_{n,j} \parallel \sigma(M_{n,j})]$  and  $P_2$ 's output  $[N_{1,j}, \dots, N_{n,j}]$ , where  $M_{i,j}, N_{i,j} \in \{0, 1\}$ . Let  $\mathcal{M}_j = [M_{1,j}, \dots, M_{n,j}]$  and  $\mathcal{N}_j = [N_{1,j}, \dots, N_{n,j}]$  be the  $n$ -bit outputs for  $P_1$  and  $P_2$ , respectively.  $P_2$  then chooses index  $j_l$  such that  $\mathcal{M}_{j_l}$  and  $\mathcal{N}_{j_l}$  appear more than  $e/2$  times in vectors  $[\mathcal{M}_{j_1}, \dots, \mathcal{M}_{j_e}]$  and  $[\mathcal{N}_{j_1}, \dots, \mathcal{N}_{j_e}]$ , respectively.  $P_2$  sends  $\mathcal{M}_{j_l}$  to  $P_1$  and takes  $\mathcal{N}_{j_l}$  as her final output. If no such  $j_l$  exists,  $P_2$  aborts.
8. **Verification to  $P_1$ 's output:** To convince  $P_1$  the authenticity of  $\mathcal{M}_{j_l}$  without revealing  $j_l$ ,  $P_1$  generates another signature key pair  $(sk_2, pk_2)$ . Then  $P_1$  signs the indices of all the evaluation-circuits and sends the results to  $P_2$ . In particular,  $P_1$  sends to  $P_2$  the public key  $pk_2$  and a signature vector  $[\delta(j_1), \dots, \delta(j_e)]$ , where  $\delta(j) = S_{\text{SIG}}(sk_2, j)$ . The signature is verified by  $P_2$  by checking  $V_{\text{SIG}}(pk_2, j, \delta(j)) = \text{valid}$ , for every  $j \in \{j_1, \dots, j_e\}$ . Next,  $P_2$  proves to  $P_1$  in witness-indistinguishable sense the knowledge of  $\sigma(M_{i,j_l}, i, j)$  (a signature signed with  $sk_1$ ) and  $\delta(j^*)$  (a signature signed with  $sk_2$ ) such that  $j$  and  $j^*$  are equivalent, for  $1 \leq i \leq n$ .  $P_1$  aborts if the proof is not valid; otherwise,  $P_1$  takes  $\mathcal{M}_{j_l} \oplus (x_{n+1}, \dots, x_{2n})$  as her final output.

**Theorem 2.** *Let  $f : \{0, 1\}^n \times \{0, 1\}^n \rightarrow \{0, 1\}^n \times \{0, 1\}^n$  be any function. Given a secure committing oblivious transfer protocol, a perfectly-hiding commitment scheme, a perfectly-binding commitment scheme, a malleable claw-free family, and a pseudo-random function family, the Main protocol securely computes  $f$ .*

We have omitted the standard simulation-based definition of “securely computes  $f$ ” for space. Roughly, this definition requires a simulator for the corrupted evaluator, and a simulator for the corrupted generator that is able to generate transcripts given only oracle access to either the evaluator or generator (respectively) that are indistinguishable from the transcripts produced in real interactions between the corrupted generator and honest evaluator or honest generator and corrupted evaluator. (A simulator for when both parties are corrupted is also required but trivial.) The proof of Theorem 2 is omitted for space.

## 4 Experimental Results

We produced an implementation of our protocol to demonstrate its practical benefits. Our implementation takes the boolean circuit generated by Fairplay compiler as input. The encryption function used to construct garbled gates is defined as  $\text{Enc}_{J,K}(m) = (m \oplus \text{SHA-256}(J) \oplus \text{SHA-256}(K))_{1\dots k}$ , where  $|J| = |K| = |m| = k$ , and  $S_{1\dots k}$  denotes the least significant  $k$  bits of  $S$ . Here SHA-256 is modeled as a pseudorandom function. The choice of SHA-256 is to make a fair comparison as it is used in [20].

	# Gates			Time (s)			Totals	
	Base	Overhead	Non-XOR	Precomp	OT	Calc	Time (s)	KBytes
$(f_1, f_2)$	531	2,250	278	117	16	39	172	140,265
Ours (on <b>slower</b> )	531	6	237	35	15	21	<b>71</b>	<b>5,513</b>
Ours (on <b>fast</b> )	531	6	237	27	11	15	53	5,513
$(\lambda, \text{AES}_x(y))$	33,880	12,080	11,490	483	34	361	878	406,010
Ours (on <b>slower</b> )	33,880	0	11,286	138	58	69	<b>265</b>	<b>190,122</b>
Ours (on <b>fast</b> )	33,880	0	11,286	98	44	50	192	190,122

Table 2: The performance comparison with [20].

Following Pinkas et. al [20], we set the security level to  $2^{-40}$  and the security parameter  $k$  (key length) to 128-bit. In the first experiment,  $P_1$  and  $P_2$  hold a 32-bit input  $x = (x_{31}x_{30} \dots x_0)_2$  and  $y = (y_{31}y_{30} \dots y_0)_2$ , respectively. They want to compute  $f(x, y) = (f_1, f_2)$  such that after the secure computation,  $P_1$  receives  $f_1 = \sum_{i=0}^{31} x_i \oplus y_i$ , and  $P_2$  receives  $f_2$  as the result of comparison between  $x$  and  $y$ . The 6 gates of overhead we incur in the first experiment relate to our method for two-output functions. In the second experiment,  $P_2$  has a 128-bit message block while  $P_1$  has a 128-bit encryption key. They want to securely compute the AES encryption, and only  $P_2$  gets the ciphertext.



We ran our experiments on two machines: `slower` and `fast`, where `slower` runs OS X 10.5 with Intel Core 2 Duo 2.8 GHz and 2GB RAM, and `fast` runs CentOS with Intel Xeon Quad Core E5506 2.13 GHz and 8GB RAM. `slower` is not as powerful as the machine used in [20] (Intel Core 2 Duo 3.0 GHz, 4GB RAM), and `fast` is the next closest machine that we have.

Table 2 reports the *best* numbers from [20]. We note that [20] applies the Garbled Row Reduction technique so that even non-XOR gates can save 25% of the communication overhead. A future version of our protocol can also reap this 25% reduction since the technique is compatible with our protocol.

Our implementation involves a program for  $P_1$  and one for  $P_2$ . For the purpose of timing, we wrote another program that encapsulates both of these programs and feeds the output of one as the input of the other and vice versa. Timing routines are added around each major step of the protocol and tabulated in Table 3. This timing method eliminates any overhead due to network transmission, which we cannot reliably compare. The reported values are the averages from 5 runs.

We implemented our solution with the PBC (Pairing Based Cryptography) library [1] for testing. The components of our protocol, including the claw-free collections, the generator’s input consistency check, and the generator’s output validity check, are built on top of the elliptic curve  $y^2 = x^3 + 3$  over the field  $\mathbb{F}_q$  for some 80-bit prime  $q$ . We have made systems-level modifications to the random bit sampling function of the PBC library (essentially to cache file handles and eliminate unnecessary systems calls).

In Table 4, we list the results of the MAC-based two-output function handling and ours. The MAC approach introduces extra 16,384 ( $128^2$ ) non-XOR gates to the AES circuit, whereas the original AES circuit has only 11,286 non-XOR gates. Since the number of non-XOR gates is almost doubled in the MAC-based approach, their circuit construction and evaluation need time about twice as much as ours. Moreover, the MAC-based approach has twice as many input bits as ours so that the time for  $P_1$ ’s input consistency has doubled.

	$f(x, y) = (f_1, f_2)$			$f(x, y) = (\lambda, \text{AES}_x(y))$		
	$P_1$	$P_2$	Sum (s)	$P_1$	$P_2$	Sum (s)
Precomp Time	35.4	0.0	35.4	137.7	0.0	137.7
OT Time	7.9	6.7	14.6	31.9	26.3	58.2
Cut-and-Choose	0.0	14.7	14.7	0.0	44.4	44.4
Input Check	0.0	3.0	3.0	0.0	10.0	10.0
Eval Time	0.0	3.4	3.4	0.0	14.1	14.1
Two-output	0.1	0.0	0.1	0.0	0.0	0.0
Total (s)	43.4	27.8	71.2	169.6	94.8	264.4

Table 3: The running time (in seconds) of two experiments on machine `slower`.

COMM. FOR EACH STAGE (KBYTES)			SEMI-HONEST ADVERSARIES		
Circuit construction	2,945	53.42%	This work		[20]
Oblivious transfer	675	12.25%	No. of gates	531	531
Cut-and-choose	1,813	32.89%	Comm. (KBytes)	23	22
$P_1$ 's input consistency	76	1.38%	MALICIOUS ADVERSARIES		
$P_1$ 's output validity	3	0.01%	No. of gates	537	2,781
Total communication	5,513	100.00%	Comm. (KBytes)	5,513	167,276

(a)

(b)

Fig. 3: (a) Communication cost for Experiment 1 by stages for our solution given statistical security parameter  $s = 125$  and security parameter  $k = 128$ . (b) The circuit size and communication cost comparison with [20] (which also ensures the cheating probability is limited below  $2^{-40}$ ).

COMM. FOR EACH STAGE (KBYTES)			SEMI-HONEST ADVERSARIES		
Circuit construction	99,408	52.29%	This work		[20]
Oblivious transfer	2,699	1.42%	No. of gates	33,880	33,880
Cut-and-choose	87,585	46.16%	Comm. (KBytes)	795	503
$P_1$ 's input consistency	256	0.13%	MALICIOUS ADVERSARIES		
$P_1$ 's output validity	0	0.00%	No. of gates	33,880	45,960
Total communication	190,122	100.00%	Comm. (KBytes)	190,122	406,010

(a)

(b)

Fig. 4: (a) Communication cost for Experiment 2 by stages for our solution given statistical security parameter  $s = 125$  and security parameter  $k = 128$ .

## References

1. Pairing-Based Cryptography Library (2006), <http://crypto.stanford.edu/abc/>
2. Boneh, D., Boyen, X.: Short Signatures Without Random Oracles and the SDH Assumption in Bilinear Groups. *Journal of Cryptology* 21, 149–177 (2008)
3. Brassard, G., Crépeau, C., Robert, J.M.: All-or-Nothing Disclosure of Secrets. In: Odlyzko, A. (ed.) CRYPTO 1986. LNCS, vol. 263, pp. 234–238. Springer (1987)
4. Camenisch, J., Chaabouni, R., Shelat, A.: Efficient Protocols for Set Membership and Range Proofs. In: Pieprzyk, J. (ed.) ASIACRYPT 2008. LNCS, vol. 5350, pp. 234–252. Springer (2008)
5. Even, S., Goldreich, O., Lempel, A.: A Randomized Protocol for Signing Contracts. *Communications of ACM* 28, 637–647 (1985)
6. Goldreich, O., Micali, S., Wigderson, A.: How to Play ANY Mental Game. In: 19th Annual ACM Symposium on Theory of Computing. pp. 218–229. ACM (1987)
7. Goldreich, O., Kahan, A.: How to Construct Constant-Round Zero-Knowledge Proof Systems for NP. *Journal of Cryptology* 9, 167–189 (1996)

	MAC two-output approach			Our two-output approach		
	$P_1$	$P_2$	Subtotal	$P_1$	$P_2$	Subtotal
Precomp Time	498.9	0.0	498.9	294.1	0.0	294.1
OT Time	32.0	26.3	58.3	31.9	26.2	58.1
Cut-and-Choose	0.0	158.6	158.6	0.0	185.3	185.3
Input Check	0.0	40.4	40.4	0.0	19.8	19.8
Eval Time	0.0	50.6	50.6	0.0	24.4	24.4
Two-output	0.0	0.0	0.0	0.7	0.6	1.3
Total	530.9	275.9	806.8	326.7	256.3	583.0

Table 4: Computation time (in seconds) of  $f(x, y) = (\text{AES}_x(y), \lambda)$  running on machine `slower` under different two-output handling methods.

8. Jarecki, S., Shmatikov, V.: Efficient Two-Party Secure Computation on Committed Inputs. In: Naor, M. (ed.) EUROCRYPT 2007. LNCS, vol. 4515, pp. 97–114. Springer (2007)
9. Kiraz, M.: Secure and Fair Two-Party Computation. Ph.D. thesis, Technische Universiteit Eindhoven (2008)
10. 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. pp. 283–290 (2006)
11. Kiraz, M., Schoenmakers, B.: An Efficient Protocol for Fair Secure Two-Party Computation. In: Malkin, T. (ed.) CT-RSA 2008. LNCS, vol. 4964, pp. 88–105. Springer (2008)
12. Kolesnikov, V., Schneider, T.: Improved Garbled Circuit: Free XOR Gates and Applications. In: Aceto, L., Damgård, I., Goldberg, L., Halldórsson, M., Ingólfssdóttir, A., Walukiewicz, I. (eds.) ALP 2008. LNCS, vol. 5126, pp. 486–498. Springer (2008)
13. Lindell, Y., Pinkas, B.: An Efficient Protocol for Secure Two-Party Computation in the Presence of Malicious Adversaries. In: Naor, M. (ed.) EUROCRYPT 2007. LNCS, vol. 4515, pp. 52–78. Springer (2007)
14. Lindell, Y., Pinkas, B.: Secure Two-Party Computation Via Cut-and-Choose Oblivious Transfer. Crypto ePrint Archive (2010), <http://eprint.iacr.org/2010/284>
15. Lindell, Y., Pinkas, B., Smart, N.: Implementing Two-Party Computation Efficiently with Security Against Malicious Adversaries. In: Ostrovsky, R., De Prisco, R., Visconti, I. (eds.) SCN 2008, LNCS, vol. 5229, pp. 2–20. Springer (2008)
16. Mohassel, P., Franklin, M.: Efficiency Tradeoffs for Malicious Two-Party Computation. In: Yung, M., Dodis, Y., Kiayias, A., Malkin, T. (eds.) PKC 2006. LNCS, vol. 3958, pp. 458–473. Springer (2006)
17. Naor, M., Pinkas, B.: Oblivious transfer with adaptive queries. In: Wiener, M. (ed.) CRYPTO 1999, LNCS, vol. 1666, pp. 791–791. Springer (1999)
18. Nielsen, J., Orlandi, C.: LEGO for Two-Party Secure Computation. In: Reingold, O. (ed.) TCC 2009. LNCS, vol. 5444, pp. 368–386. Springer (2009)
19. Peikert, C., Vaikuntanathan, V., Waters, B.: A framework for efficient and composable oblivious transfer. In: Wagner, D. (ed.) CRYPTO 2008, LNCS, vol. 5157, pp. 554–571. Springer (2008)
20. Pinkas, B., Schneider, T., Smart, N., Williams, S.: Secure Two-Party Computation Is Practical. In: Matsui, M. (ed.) ASIACRYPT 2009. LNCS, vol. 5912, pp. 250–267. Springer (2009)

21. Rabin, M.: How to Exchange Secrets by Oblivious Transfer. Tech. Rep. TR-81, Harvard Aiken Computation Laboratory (1981)
22. Woodruff, D.: Revisiting the Efficiency of Malicious Two-Party Computation. In: Naor, M. (ed.) EUROCRYPT 2007. LNCS, vol. 4515, pp. 79–96. Springer (2007)
23. Yao, A.: Protocols for Secure Computations. In: 23rd Annual Symposium on Foundations of Computer Science. pp. 160–164. IEEE Computer Society (1982)

## A Optimal Choice in Cut-and-Choose Strategy

According to the cut-and-choose strategy,  $P_2$  chooses  $e$  copies of the garbled circuits and asks  $P_1$  to open the rest ( $s - e$ ). After the verification,  $P_2$  evaluates the rest  $e$  copies of the circuits and takes the majority output as her output. A natural question is: *Under the assumption that  $P_1$ 's inputs are consistent, how many circuits does  $P_2$  evaluate in order to minimize the probability for  $P_1$ 's best cheating strategy to succeed?*

The assumption is valid due to the consistency check on  $P_1$ 's input. Given that  $s$  and  $e$  are fixed and known to  $P_1$ , let  $b$  be the number of bad circuits created by  $P_1$ . A circuit is *bad* if either the circuit is wrongly constructed or  $P_2$ 's inputs are selectively failed via OT. The goal is to find  $e$  and  $b$  such that the probability that  $P_1$  cheats without getting caught  $\binom{s-b}{s-e} / \binom{s}{s-e}$  is minimized.

We first claim that  $P_1$ 's best cheating strategy is to produce  $b = \lfloor e/2 \rfloor + 1$  bad circuits. Indeed, if  $b \leq \lfloor e/2 \rfloor$ ,  $P_2$ 's output will not get affected since the faulty outputs will be overwhelmed by majority good ones. Also, the more bad circuits, the more likely that  $P_1$  will get caught since  $\binom{s-(b-1)}{s-e} > \binom{s-b}{s-e}$ . So the best strategy for  $P_1$  to succeed in cheating is to construct as few bad circuits as possible while the majority of evaluation circuits are bad, which justifies the choice of  $b$ .

Our next goal is to find the  $e$  that minimizes  $\Pr(e) = \binom{s-\lfloor \frac{e}{2} \rfloor - 1}{s-e} / \binom{s}{s-e}$ . To get rid of the troublesome floor function, we will consider the case when  $e$  is even and odd separately. When  $e = 2k$  for some  $k \in \mathbb{N}$  such that  $k \leq \frac{s}{2}$ , let  $\Pr_{\text{even}}(k) = \binom{s-k-1}{s-2k} / \binom{s}{s-2k}$ . Observe that  $\frac{\Pr_{\text{even}}(k+1)}{\Pr_{\text{even}}(k)} = \frac{(2k+1)(2k+2)}{(s-k-1)k}$ . It is not hard to solve the quadratic inequality and come to the result that

$$\frac{\Pr_{\text{even}}(k+1)}{\Pr_{\text{even}}(k)} \leq 1 \text{ when } 0 < k \leq \frac{1}{10} \left( s - 7 + \sqrt{(s-7)^2 - 40} \right) \stackrel{\text{def}}{=} \alpha.$$

In other words,  $\Pr_{\text{even}}(k) \geq \Pr_{\text{even}}(k+1)$  when  $0 < k \leq \alpha$ ; and  $\Pr_{\text{even}}(k) < \Pr_{\text{even}}(k+1)$  when  $\alpha < k \leq \frac{s}{2}$ . Therefore,  $\Pr_{\text{even}}$  is minimal when  $k = \lceil \alpha \rceil$ . Similarly, when  $e = 2k + 1$ , the probability  $\Pr_{\text{odd}}(k) = \binom{s-k-1}{s-2k-1} / \binom{s}{s-2k-1}$  is minimal when  $k = \lceil \beta \rceil$ , where  $\beta = \frac{1}{5}(s-7)$ . In summary,

$$\begin{cases} \Pr_{\text{even}}(e) \text{ is minimal when } e = 2\lceil \alpha \rceil; \\ \Pr_{\text{odd}}(e) \text{ is minimal when } e = 2\lceil \beta \rceil + 1, \end{cases}$$

and  $\Pr(e)$ 's minimum is one of them.