Two-Party Computing with Encrypted Data

Seung Geol Choi[1], Ariel Elbaz[1], Ari Juels[2], Tal Malkin[1], Moti Yung[1]

[1] Columbia University
{sgchoi, arielbaz, tal, moti}@cs.columbia.edu
[2] RSA Laboratories
ajuels@rsasecurity.com

Abstract. We consider a new model for online secure computation on encrypted inputs in the presence of malicious adversaries. The inputs are independent of the circuit computed in the sense that they can be contributed by separate third parties. The model attempts to emulate as closely as possible the model of "Computing with Encrypted Data" that was put forth in 1978 by Rivest, Adleman and Dertouzos which involved a single online message. In our model, two parties publish their public keys in an offline stage, after which any party (i.e., any of the two and any third party) can publish encryption of their local inputs. Then in an on-line stage, given any common input circuit C and its set of inputs from among the published encryptions, the first party sends a *single message* to the second party, who completes the computation.

Keywords: Computing with Encrypted Data, Secure Two-Party Computation,CryptoComputing, oblivious transfer

1 Introduction

In "Computing with Encrypted Data", first a public key is published by one party, followed by collection of data encrypted under this key (potentially from various sources and independent of the actual computation). Later, in an online stage, a computing party who possesses a circuit of a function acts on the encrypted data, and sends the result (a single message) to the owner of the public key for output decryption. This wishful single message scenario for secure computation, was put forth as early as 1978 by Rivest, Adleman and Dertouzos [24]. This model is highly attractive since it represents the case where a database is first collected and maintained and only later a computation on it is decided upon and executed (i.e., data mining and statistical database computation done over the encrypted database). However, in its most general form (and the way [24] envisioned it), the model requires an encryption function that is homomorphic over a complete base (sometimes called doubly homomorphic encryption), which is a construction that we do not have (finding such a scheme is a long standing open problem and would have far reaching consequences); further, we have indications such a scheme cannot be highly secure [3].

In this paper we put forth a relaxation of the above model, that relies on two party secure computations, yet retains much of the desired properties of the original model, namely, it allows computing of any feasible functions over encrypted data, it further allows the data to come from various sources, and it employs a single online message as well. Our proposed relaxation is to allow two parties (rather than one) to publish a shared pubic key, and both parties hold shares of the private key and use their shares of the secret key to do computations on data encrypted with the public key. Once the public key is published, data contributors publish encrypted (committed) data as before (this is called the off-line stage). Then, in the on-line stage, one of the two parties (the compiler) is sending a **single message** to the second party (the cryptocomputer), that contains a circuit for a function to compute, and a garbled circuit of the same function, allowing the second party to compute the result securely (i.e., while keeping the inputs private, and gaining no computational advantage beyond what it can compute from the result and the inputs it knows). Note that because of its essentially non-interactive nature, our model is also particularly suitable for applications involving low-latency remote executions, such as for mobile agent applications [26].

We give two protocols in this model, which differ only in the cryptographic assumptions and the communication complexity. Both protocols are secure even against malicious parties, and both allow computing any polynomial function (or sequence of functions) by a single on-line message exchange, in a sense satisfying the original vision of [24] for computing with encrypted data.

If we limit the input contribution to the two parties involved, our model matches naturally the theory of general secure two party computation (see [17, 32] and [20, 21] for some of the earliest and the latest works in this area). While it may be possible to turn many of the works on two party computations to single message protocols (based on random oracle or non-interactive proofs), we have not seen this mentioned explicitly (the closest being [4]) or a proof of security given for it. To the best of our knowledge none of the previous garbled-circuit-based two party secure computation results allows for data contribution by third parties (an issue that was not even modeled earlier).

In the general two party computation setting, two parties Alice and Bob have private inputs x_A and x_B respectively, and wish to compute a function $f(x_A, x_B)$ securely, without leaking any further information. A particularly useful setting is where Alice and Bob have published commitments s_A, s_B on their inputs, which allows secure computation to proceed more efficiently. Applying our results to this setting, we can have Alice and Bob encrypt their input during the off-line stage (independently of any computation); then the subsequent secure computation (or "cryptocomputing" [27]) only requires a single message per function to be computed. A similar result was previously known only for functions of restricted complexity classes (e.g., [27] show how to securely compute functions in NC^1), while we provide a protocol for any function in P.

The idea of minimizing the on-line stage in cryptographic primitives goes back to the notion of Off-line On-line Signature of Even, Goldreich and Micali where they minimized the amount of computations of a signature at the on-line stage (after a message is given as an input) [12].

1.1 Our Model and Results

As outlined above, we propose the off-line/on-line model for crypto-computing using a single message (and thus optimal round complexity) for the on-line stage. For $k \geq 2$, there are parties $P_1, \ldots, P_k$. We name P_1 as Alice and P_2 as Bob. The model consists of the following four stages.

1. Alice and Bob publish prospective shares of the public key, y_A and y_B.
2. Separable Data Collection: Parties $P_1, \ldots, P_k$ publish their data, encrypted by a shared public key y.
3. Communication: Given an input circuit C with m designated inputs bits, Alice sends to Bob a single message containing a set of indexes to the published encrypted inputs $\{idx_i\}_{i=1}^{m}$, and a garbled circuit $\hat{C}$.
4. Computation: Bob decides if the message is consistent with the input circuit and its inputs, verifying that the indices to the encrypted inputs are valid, and that $\hat{C}$ is a garbled version of C. If all these tests succeed, Bob computes $\hat{C}$ on the committed inputs.

Note that since we deal with any polynomial-size function (or circuit), we can have some of the data encode circuits and the on-line circuit be a universal one [31].

We give two protocols that are secure within this model. The first is based on the traditional and quite minimal DDH assumption and uses ElGamal encryption, and the other is based on the DCR assumption and uses the simplified Camenisch-Shoup encryption scheme (introduced by [20]). The latter protocol achieves better communication complexity, at the price of using a stronger more recent assumption and encryption method.

We use non-interactive zero-knowledge proofs (NIZK) for the malicious case, which can be achieved either in the common reference string model or in the random oracle model. Under the common reference string model, the NIZK PoK of De Santis and Persiano [28] can be used, assuming dense secure public-key encryption scheme. Under the random oracle model, the well-known Fiat-Shamir technique [14] can be used.

A main primitive our work relies upon is a conditional exposure primitive we call $\mathcal{CODE}$ (Conditional Oblivious Decryption Exposure). $\mathcal{CODE}$ is a two-party non-interactive protocol, which allows Bob to learn the plaintext of a cyphertext c, if two other cyphertexts a, b encrypt the same value. Unlike other conditional exposure primitives (e.g. Gertner et al [16] and Aeillo et al [1]), in CODE the three cyphertexts a, b, c are encrypted with a shared public key, such that third parties can contribute them, and neither Alice nor Bob alone know anything else about the result of $\mathcal{CODE}$. The conditional exposure primitive of Aeillo et al. [1] is a natural translation of a logical 'if a equals b' to arithmetics on cyphertexts using encryption that is homomorphic in the plaintext. The $\mathcal{CODE}$ primitive uses homomorphic properties of the keys and of the plaintexts and gives more freedom to design protocols that include inputs shared among the parties.

This allows for oblivious yet secure "input directed navigation" in a garbled circuit based on a single trigger, given encrypted inputs. The technique also allows efficient combination with zero-knowledge proofs to assure robustness of the overall protocol.

We note that we concentrate on a single message computation and present the protocols with respect to the most efficient random oracle based proofs. Modifying the scheme to employ non-interactive proofs in the standard model and modifying the single message scheme to consider the universal composability model of security are possible as well.

1.2 Previous work

As mentioned above, Rivest, Adleman, and Dertouzos [24] offer perhaps the first proposal for the study of blind computation on cyphertexts, considering them as a primitive for private data manipulation. Feigenbaum and Merritt [13] subsequently urged more focused investigation on cryptosystems with algebraic homomorphisms. The term "CryptoComputing" and the first non-trivial instantiation originated with Sander, Young, and Yung [27], who present a CryptoComuting protocol for functions f in NC^1. In their model, Alice does not publish her input s_A, but instead sends it (hides it) within her transcript, and information theoretic security is achieved with respect to Bob. This is to say that Bob learns no information whatever about s_A apart from the output of f. Beaver [2] extends [27] to accommodate any function in NLOGSPACE. Other reduced round secure computations (two message constructions, in fact) have been suggested by Naor, Pinkas, and Sumner [22] and by Cachin, Camenisch, Kilian, and Müller [4]. Their approaches are based on the two-party secure function evaluation scheme of Yao [32] and Goldreich, Micali, and Wigderson [17].

Recently the area of robust two-party computations in constant rounds has gained some attention. Specifically, the works of Jarecki and Shmatikov [20], Lindell and Pinkas [21] and Horvitz and Katz [19] gave protocols for two-party computation using Yao's garbled circuit that are secure against malicious adversaries. [20] uses a modified Camenisch-Shoup verifiable encryption scheme [6] to allow the party that sends the garbled circuit to prove its correctness. Our simplified-Camenisch-Shoup based protocol was devised by combining the ideas of our first protocol with those from [20], in order to satisfy our model with better communication complexity. Lindell and Pinkas [21] use a cut-and-choose approach to proving security of Yao's garbled circuit against malicious adversaries and their method is more generic yet requires a few more rounds. Horvitz and Katz [19] showed a UC-secure protocol in two rounds (four messages) using the DDH assumption. In their protocol, the two parties essentially run two instances of Yao's protocol simultaneously.

2 Preliminaries

In the primitives we describe below, as well as in our main protocol, we assume that Alice and Bob agree in advance on some groups over which the computation is being done.

Let ℓ be a security parameter. In the constructions below, it is generally appropriate to let $\ell = \log q$. We say that a function $f(l)$ is *negligible* if for any polynomial *poly*, there exists a value d such that for any $l \geq d$, we have $f(l) < 1/|poly(l)|$. To achieve non-interactive proofs in the malicious case, we also assume a random oracle for the underlying hash function.

2.1 ElGamal Cryptosystem

We employ the ElGamal cryptosystem [11] in our first construction. ElGamal encryption takes place over the group $\mathcal{G}_q$ over which it is hard to compute discrete logarithms. Typically, $\mathcal{G}_q$ is taken to be a subgroup of Z_p^*, where $q \mid p-1$, for large primes p and q. We denote g as a published generator of $\mathcal{G}_q$.[3]

Let $y = g^x$ be the public key for the secret key x. The encryption of a message m (denoted $E_y(m)$) is $(g^r, m \cdot y^r)$ for $r \in_R [1, q]$. The decryption of a cyphertext (α, β) (denoted $D_x(\alpha, \beta)$ is β/α^x. The ElGamal cryptosystem is *semantically secure* [18] under the Decision Diffie-Hellman (DDH) assumption [10] over $\mathcal{G}_q$. We intensively use the multiplicative homomorphism of the ElGamal cryptosystem: $E_y(m_1) \cdot E_y(m_2) = E_y(m_1 \cdot m_2)$.

Our protocol makes use of a private/public keys $(x_A, y_A = g^{x_A})$ for Alice, as well as a private/public key $(x_B, y_B = g^{x_B})$ for Bob. We denote by y the shared public key $y_A \cdot y_B$, for which the corresponding private key is $x_A + x_B$. Note that y may be established implicitly by Alice on learning y_B and by Bob on learning y_A. In particular, there is no need for interaction between the parties to determine the shared key. Since the public keys are published, we assume all parties hold the joint public key y.

2.2 simplified-Camenisch-Shoup Cryptosystem

For sCS cryptosystem, Alice and Bob work over $Z_{n^2}^*$ for $n = pq$, where $p = 2p'+1, q = 2q'+1$, and p, q, p', q' are all primes, and $|p| = |q|$. Let $n' = p'qp'$, and $h = (1+n)$. The group $Z_{n^2}^*$ has unique (up to isomorphism) decomposition as the direct-product of four cyclic groups $Z_{n^2}^* = G_n \times G_{n'} \times G_2 \times T$, where G_n is generated by h and has order n, $G_{n'}$ has order n', and G_2 and T are of order 2. Let g' be a random element of $Z_{n^2}^*$. We know that the order of g' divides $\phi(n^2) = n \cdot \phi(n) = 4nn'$. With very high probability, the order of g' is a *multiple* of n', and $g = (g')^{2n}$ thus has order n' and is a generator of $G_{n'}$.

For the simplified-Camenisch-Shoup (as well as the original Camenisch-Shoup), all operations take place in $Z_{n^2}^*$. Note that h has order n and that $h^c = 1 + cn \pmod{n^2}$. The DCR assumption [23] is that given only n, random elements of $Z_{n^2}^*$ are hard to distinguish from random elements of P, which is the subgroup of $Z_{n^2}^*$ consisting of all nth powers of elements in $Z_{n^2}^*$.

The sCS encryption scheme, introduced by [20] (and based on the CS scheme of [6]), is semantically secure under the DCR assumption.

[3] In the settings where $p = 2q+1$ and $\mathcal{G}_q$ is the set of quadratic residues in Z_p^*, plaintexts not in $\mathcal{G}_q$ can be mapped onto $\mathcal{G}_q$ by appropriate forcing of the LeGendre symbol, e.g., through multiplication by a predetermined non-residue.

Key generation A private key is $x \in [0, \frac{n^2}{4}]$. A public key is (n, g, y) for $y = g^x$.

Encryption We map the message m to an integer in $(-\frac{n}{2}, \frac{n}{2}]$. The encryption $E_{PK}(m)$ is a pair $(u, e) = (g^r, h^m y^r)$ for a random integer $r \in [0, \frac{n}{4}]$.

Decryption Given a pair (u, e), if this is a valid cyphertext it is of the form $(g^r, h^m y^r)$. Let $\hat{m} = (\frac{e}{u^x})^2$. If $\hat{m}$ is valid, it is $(1+n)^m \equiv 1 + nm \pmod{n^2}$ for some m, so check that $n|\hat{m}-1$ and reject otherwise. Else, let $m' = (\hat{m}-1)/n$ (over the integers), let $m'' = m'/2 \pmod{n}$, and recover the message $m = m''\ rem\ n$, where $(a\ rem\ b)$ is a if $a \leq b/2$ and otherwise it is $b - a$.

2.3 $\mathcal{CODE}$ (Conditional Oblivious Decryption Exposure)

The linchpin of our construction is a protocol that we newly introduce in this paper. We refer it to as *Conditional Oblivious Decryption Exposure*. One of the main differences between $\mathcal{CODE}$ and previously suggested conditional exposure primitives is that $\mathcal{CODE}$ allows for third parties to contribute encryptions using a public key, and then Alice and Bob, who share the private key can perform the conditional exposure.

Definition 1. Conditional Oblivious Decryption Exposure *Let (x_A, y_A) and (x_B, y_B) be two secret/public key pairs and E (resp. D) be the encryption (resp. decryption) function. Let c_1, c_2, c_3 be three cyphertexts encrypted under the joint key $y = y_A \cdot y_B$. The functionality $\mathcal{CODE}$ is defined by*

$$((c_1, c_2, c_3, x_A, y_B), (x_B, y_A)) \mapsto \begin{cases} (\perp, (c_1, c_2, c_3, D_x(c_3)) & \text{if } D_x(c_1) = D_x(c_2) \\ (\perp, (c_1, c_2, c_3, r)) & \text{otherwise.} \end{cases}$$

Where $x = x_A + x_B$ and $r \in_R \mathcal{G}_q$.

In this functionality, the decryption of c_3 is exposed to the second party conditioned on $c_1 \equiv c_2$ (i.e., if they encrypt the same message). Moreover the first party is oblivious of the outcome of the protocol.

We show protocols for secure implementations of $\mathcal{CODE}$ functionality using eitehr ElGamal and sCS encryptions. *impCODE* is a protocol for the $\mathcal{CODE}$ functionality secure in the honest-but-curious case.

impCODE. Let's call the first party Alice and the second party Bob. The $\mathcal{CODE}$ implementation consists of a single $\mathcal{CODE}$ transcript sent from Alice to Bob. Let $c_1 = (\alpha, \beta) = (g^{r_1}, m_1 y^{r_1})$, $c_2 = (\gamma, \delta) = (g^{r_2}, m_2 y^{r_2})$, and $c_3 = (\lambda, \mu) = (g^{r_3}, m_3 y^{r_3})$. Alice sends (ϵ, ζ, D) to Bob where

1. $\epsilon = (\alpha/\gamma)^e$ and $\zeta = (\beta/\delta)^e$, for $e \in_R Z_q$
2. $D = (\epsilon\lambda)^{x_A}$.

Bob computes $\tilde{m_3} = \frac{\zeta\mu}{D \cdot D'}$ where $D' = (\epsilon \cdot \lambda)^{x_B}$ and outputs $(c_1, c_2, c_3, \tilde{m_3})$. Note that

$$\tilde{m_3} = \frac{\left(\frac{m_1 y^{r_1}}{m_2 y^{r_2}}\right)^e \cdot m_3 y^{r_3}}{\left(\left(\frac{g^{r_1}}{g^{r_2}}\right)^e \cdot g^{r_3}\right)^x} = \left(\frac{m_1}{m_2}\right)^e \cdot m_3$$

that is, if $m_1 = m_2$, then $\tilde{m_3} = m_3$, as required.

Theorem 1. *The protocol impCODE securely implements the functionality $\mathcal{CODE}$ for the honest-but-curious two parties under DDH assumption.*

Proof. We show a simulator $S = (SIM_A, SIM_B)$ for *impCODE*. The case of corrupted Alice is easy; since Alice does not get any message from Bob, the simulator SIM_A is trivial. For a corrupted Bob, the simulator SIM_B has to simulate the view of Bob. Formally,

$$\{SIM_B((x_B, y_A), (c_1, c_2, c_3, d))\} \overset{c}{\approx} \{\text{view}_B((c_1, c_2, c_3, x_A, y_B), (x_B, y_A))\}$$

In other words, given the input and output of Bob, SIM_B has to simulate the *impCODE* transcript (ϵ, ζ, D) that Alice sends to Bob. The simulator SIM_B computes

$$e \in_R Z_q, \quad \epsilon = (\alpha/\gamma)^e, \quad \zeta = (\beta/\delta)^e, \quad D' = (\epsilon\lambda)^{x_B}, \quad D = \frac{\zeta\mu}{d \cdot D'} \ ,$$

and outputs (ϵ, ζ, D). The simulated (ϵ, ζ) have the same distribution as in the real protocol. Given (ϵ, ζ) and d, D is uniquely determined.

3 Honest-but-curious protocol

3.1 Intuition

In our one-message secure function evaluation scheme, Alice sends a garbled circuit to Bob, and Bob computes the function f using the garbled circuit. Let C be a circuit with gates $G_1, G_2, \ldots, G_m$ that computes the function f of interest, and let $T_1, T_2, \ldots, T_m$ be the corresponding truth tables. Sometimes we interchangeably use the term gates and tables.

First of all, Alice garbles each table by encrypting all the entries and then permuting the rows. See Table 1 for example, where Alice garbled an AND gate with shuffling permutation (1 2 3).

T_i	I^L (left input)	I^R (right input)	O (output)
1	0	0	0
2	0	1	0
3	1	0	0
4	1	1	1

T_i^a	I^L	I^R	O
1	$E[1]$	$E[0]$	$E[0]$
2	$E[0]$	$E[0]$	$E[0]$
3	$E[0]$	$E[1]$	$E[0]$
4	$E[1]$	$E[1]$	$E[1]$

In case of ElGamal encryption scheme, $E[0]$ and $E[1]$ actually mean $E[g^0]$ and $E[g^1]$ respectively. We still use the notation $E[0]$ and $E[1]$ to handle both ElGamal and sCS encryption schemes.

Fig. 1. Alice Garbles an AND gate T_i with permutation (1 2 3) and gets T_i^a.

As in Yao's garbled circuit, Bob's computation of a gate G_j depends on the computation of the two gates G_i, G_k associated with the inputs to G_j, where

T_i^b	I^L	I^R	O	$\mathsf{Plugs}_{[i\to j]}$
1	$E[1]$	$E[0]$	$E[0]$	(n,y,y,n)
2	$E[0]$	$E[0]$	$E[0]$	(n,y,y,n)
3	$E[0]$	$E[1]$	$E[0]$	(n,y,y,n)
4	$E[1]$	$E[1]$	$E[1]$	(y,n,n,y)

T_j^b	I^L	I^R	O	$\mathsf{Plugs}_{[j\to\cdot]}$
1	$E[1]$	$E[0]$	$E[0]$	$\cdots$
2	$E[0]$	$E[0]$	$E[0]$	$\cdots$
3	$E[0]$	$E[1]$	$E[0]$	$\cdots$
4	$E[1]$	$E[1]$	$E[1]$	$\cdots$

Fig. 2. Plugs are now added.

these gates' outputs are used in the decryption of the encrypted truth table T_j. One notable difference from Yao's technique, is that here we add another level of separation between these gates' (encrypted) outputs and the key for decrypting gate G_j - this is done using $\mathcal{CODE}$. We thank the annonymous referee for commenting that indeed, in the honest-but-curious case, it is enough for us to use $\mathcal{CODE}$ only in the input gates (where Yao's protocol uses Oblivious Transfer), improving our construction's efficiency and readability. However, using $\mathcal{CODE}$ is still required for assuring security in the malicious case.

With only isolated garbled tables, however, Alice cannot have Bob compute the function. She needs to give to him 'wiring information' between a row of a table (output) and a row of an upper-level table (input). The wiring information is hereafter called a plug. See Table 2 for example. Suppose that T_j^b is the upper-level table of T_i^b where T_i^b's output is propagated into T_j^b's left input. We denote the plugs in the v-th row of the table T_i^b by $\mathsf{Plugs}_{[i\to j]}(v)$, and, more specifically, $\mathsf{Plug}_{[i\to j]}(v,w)$ denotes the w-th element of $\mathsf{Plugs}_{[i\to j]}(v)$. For example, $\mathsf{Plugs}_{[i\to j]}(1) = (n,y,y,n)$ and $\mathsf{Plug}_{[i\to j]}(1,2) = y$. The plug $\mathsf{Plug}_{[i\to j]}(1,2) = y$ means that the output value on the first row of T_i^b is equal to the left-input value on the second row of T_j^b. On the other hand, from the plug $\mathsf{Plug}_{[i\to j]}(1,4) = n$, we know that the output value of the first row of T_i^b is different from the left-input value on the fourth row of T_j^b.

However, if Bob is honest-but-curious, he might be able to find out more than the output of the function by following other computation paths because all the plug information is exposed. For example, even if Bob determines that O_i is the correct output for table T_i^b, he can experiment and try computing another computation path using a different output O_i' on another row of the same table. Such an attack, if successful, can enable Bob to explore a rich set of different computational paths for f, potentially leaking information about the secret input.

The aim of our protocol is to restrict Bob's exploration exclusively to the correct computational path. Suppose that we have three tables T_i^b, T_j^b and T_k^b where the output of T_i^b and the left input of T_j^b are connected together and so are the output of T_k^b and the right input of T_j^b. Let O_{i,v_i} (resp. O_{k,v_k}) be the output for a given row v_i in T_i^b (resp. for a given row v_k in T_k^b), and I^L_{j,v_j} (resp. I^R_{j,v_j}) be the left (resp. right) input for a given row v_j in T_j^b. Now suppose that Bob has the plugs $\mathsf{Plug}_{[i\to j]}(v_i)$ and $\mathsf{Plug}_{[k\to j]}(v_k)$, and wants to retrieve plugs in the table T_j^b. We want to make sure that Bob obtains the plug for the v_j-th row of the T_j^b **only when** $O_{i,v_i} \equiv I^L_{j,v_j}$ and $O_{k,v_k} \equiv I^R_{j,v_j}$. Since the same will be true for all gates in C, Bob can only follow the correct computational path, and learns nothing about other paths.

In order to achieve our goal, for each row of a table Alice generates an encryption key pair (pk, sk), exposes the public key pk, and hides the secret key sk by encrypting it with the global encryption key (i.e., $y = y_A \cdot y_B$). She then encrypts the plug information with pk. She wants Bob to obtain the key sk and therefore get the plug information **only when** Bob follows correct computation path. The idea is using $\mathcal{CODE}$ transcript as a plug. Recall that $\mathcal{CODE}$, given three cyphertexts c_1, c_2 and c_3, outputs the decryption of c_3 when $c_1 \equiv c_2$. Here, c_1 and c_2 corresponds to O_{i,v_i} and I^L_{j,v_j} (or O_{k,v_k} and I^R_{j,v_j}), and and c_3 to the cyphertext of sk. Below, we describe our protocols in detail.

3.2 Protocol details: Publication of keys and inputs

Alice and Bob publish their keys y_A and y_B. Input contributors encrypt input bits using the public key $y = y_A \cdot y_B$. Let s be an n-bit input string that is contributed by input contributors. Denote the i-th bit of s by s_i. When ElGamal encryption scheme is used, s is encrypted as $\{(g^{r_i}, g^{s_i} \cdot y^{r_i})\}_{i=1}^n$, where $r_i \in_R Z_q$. When sCS scheme is used, s is encrypted as $\{(g^{r_i}, h^{s_i} \cdot y^{r_i})\}_{i=1}^n$, where $r_i \in_R [0, n/4]$.

3.3 Protocol details: Alice

Structure of the Table Alice reads Bob's published key and input cyphertexts and computes $y = y_A y_B$. Now, in order to incorporate $\mathcal{CODE}$ we must extend the underlying table structure to incorporate plugs and associated keys. To do so, we append two columns to the basic table T_i^b, and denote the resulting expanded table by $\overline{T}_i$. See Table 3.

$\overline{T}_i$	I^L	I^R	O	K^L (left key)	K^R (right key)	$\mathsf{Plugs}_{[i\to j]}$
1	$E_y(1)$	$E_y(0)$	$E_y(0)$	$E_y(k_{i,11}), g^{k_{i,11}}$	$E_y(k_{i,12}),\ g^{k_{i,12}}$	$E_{z_{i,1}}(\rho_{i,11}), \ldots, E_{z_{i,1}}(\rho_{i,14})$
2	$E_y(0)$	$E_y(0)$	$E_y(0)$	$E_y(k_{i,21}), g^{k_{i,21}}$	$E_y(k_{i,22}),\ g^{k_{i,22}}$	$E_{z_{i,2}}(\rho_{i,21}), \ldots, E_{z_{i,2}}(\rho_{i,24})$
3	$E_y(0)$	$E_y(1)$	$E_y(0)$	$E_y(k_{i,31}), g^{k_{i,31}}$	$E_y(k_{i,32}),\ g^{k_{i,32}}$	$E_{z_{i,3}}(\rho_{i,31}), \ldots, E_{z_{i,3}}(\rho_{i,34})$
4	$E_y(1)$	$E_y(1)$	$E_y(1)$	$E_y(k_{i,41}), g^{k_{i,41}}$	$E_y(k_{i,42}),\ g^{k_{i,42}}$	$E_{z_{i,4}}(\rho_{i,41}), \ldots, E_{z_{i,4}}(\rho_{i,44})$

1. The value $k_{i,vw}$ for $v \in [1,4], w \in [1,2]$ is chosen randomly from Z_q.
2. The public key $z_{i,v} = g^{k_{i,v1}} \cdot g^{k_{i,v2}}$ for $v \in [1,4]$ is used to encrypt plugs of the v-th row.
3. When we want to emphasize on the abstract view of the plug $\rho_{i,vw}$ (resp. $E_{z_{i,v}}(\rho_{i,vw})$), we use the notation $\mathsf{Plug}_{[i\to j]}(v,w)$ (resp. $\widehat{\mathsf{Plug}}_{[i\to j]}(v,w)$).

Fig. 3. Schematic depiction of table $\overline{T}_i$.

Here, Bob obtains a key $k_{i,v1}$ (resp. $k_{i,v2}$) from the plug of the lower-level table when he makes a successful match against the left (resp. right) input on the row v.

Construction of the Overall Garbled Circuit Alice has to construct three types of tables: input, output and intermediate gates. First, Alice constructs the set of intermediate tables $\{\overline{T}_i\}$ as follows.

1. Alice mixes each table T_i and encrypts all the entries to yield T_i^b.
2. For each table T_i^b and row v, Alice selects $k_{i,v1}, k_{i,v2} \in_R Z_q$ to construct two columns of K^L and K^R in $\overline{T}_i$.
3. Alice computes $\widehat{\mathsf{Plugs}}_{[i\to\cdot\,]}(v)$ for each row v.

Inputs to the circuits are plugs connecting input ciphertexts and the first-level intermediate gates. Again, plugs are constructed using $\mathcal{CODE}$.

Output gates have much the same structure as intermediate gates. The only difference is in the last column. Rather than providing encrypted plugs to enable the computation to be continued, Alice provides encrypted output bits for the function f.

3.4 Protocol details: Bob

Now let us consider how Bob evaluates the transcript sent by Alice. We assume, by recursion, that when Bob tries to evaluate the output of gate G_j, he has the plugs (i.e., $\mathsf{Plugs}_{[i\to j]}(v_i)$ and $\mathsf{Plugs}_{[k\to j]}(v_k)$) for these ciphertexts into $\overline{T}_j$.

1. For each $v \in \{1, 2, 3, 4\}$, Bob performs *impCODE* with the two plugs $\mathsf{Plug}_{[i\to j]}(v_i, v)$ and $\mathsf{Plug}_{[k\to j]}(v_k, v)$ trying to obtain keys $k_{j,v1}$ and $k_{j,v2}$. If he fails (by checking if $g^\eta \stackrel{?}{=} g^{k_{j,v1}}$, where η is the output of *impCODE*), he tries the next row.
2. If he succeeds, he decrypts $\widehat{\mathsf{Plugs}}_{[j\to\cdot\,]}(v)$ with the decryption key $(k_{j,v1} + k_{j,v2})$ and gets the plug information $\mathsf{Plugs}_{[j\to\cdot\,]}(v)$. Note that $z_{j,v} = g^{k_{j,v1}+k_{j,v2}}$.
3. He proceeds with the computation using the obtained plugs.

When Bob has obtained all outputs from output gates, and so he learns the output of the circuit.

3.5 Communication Complexity in the Honest-But-Curious Case

Consider a single truth table; Each row of the table has 12 values of Z_p^* except the plugs. Plugs of each row has $4 \cdot 4 = 16$ values of Z_p^*. Therefore, each (output or intermediate) table contains $4 \cdot (12 + 16) = 112 = \mathcal{O}(1)$ values of Z_p^*. Each input plug has 5 values for Alice, and 3 values for Bob. Thus, we need $8n$ values of Z_p^* for inputs of Alice and Bob. We need another n bits for Bob to send the result of the function back to Alice. Summing all the above, it is clear that the total communication complexity is $\mathcal{O}((m + n) \log p)$ bits.

4 Full protocol

4.1 Intuition

While the protocol described above is secure assuming honest-but-curious participants, it it not secure against active cheating on the part of Alice.

A corrupted party (either Alice or Bob) can publish a public key which is not chosen randomly. For example, Alice can wait for Bob to publish his public key y_B, pick a shared private key x of her choice, and send g^x/y_B as her public key y_B. This gives Alice knowledge of the shared private key and the power to decrypt any of the inputs (including Bobs: she just needs to re-encrypt Bob's input with y_B, and then she can decrypt them with x). To overcome this kind of attacks, the malicious case protocol requires that Alice and Bob publish non-malleable PoK for the knowledge of the discrete logs of their respective public keys, together with their public keys. We note that both in the common reference string model and in the random oracle model, adding non-malleability to NIZK PoK [28] is simple: In the CRS, we follow the technique of [25]; In the random oracle model, adding non-malleability to Fiat-Shamir style NIZK PoK [14] is simple: include the name of the publisher in hash function evalution.

A corrupt Alice may cheat in the construction of the gate. First, Alice may send encrypted truth tables that do not correspond to the gates of the circuit. In Section 4.2 we show how Alice can prove that the truth tables are correct. Second, Alice may fake the plugs. Specifically, Alice may use the fact that the plugs are encrypted, and encrypt random values instead of valid plugs at selected locations. If Bob does complete the protocol, Alice learns that these invalid plugs were not decrypted, thus learning about Bob's computation path.

Therefore, in our full protocol, Alice sends Bob not only the garbled circuit but also the proof of its correct construction. The proof comprises two parts: the proof of correct construction of basic gates, and the proof of correct construction of plugs.

4.2 Proof of Correct Construction of Basic Gates

In this section, we give a zero knowledge proof of knowledge for a correct construction of gates. We assume that the circuit consists of NAND gates.

Zero-knowledge Proof of Knowledge. Informally, a Proof of Knowledge is a proof for a relation R, in which the prover convinces the verifier that an instance is in the language, and also that *the prover knows a witness for this instance*, rather then just the existence of such a witness. In a (standard) proof of knowledge for the discrete log, the prover convinces the verifier that she knows the value of b, such that $a = g^b$, when a is known to both. We denote such proof by $PK\{b : a = g^b\}$. There are many variants on these proofs, such as in [30]. In this paper, we make use of variants in which Alice proves conjunctive statements, and statements regarding her knowledge of sets of discrete logs. See [9, 29, 5] for a description of how to achieve such variants in an efficient manner.

Proof of Boolean Plaintext. Let $\sigma^0 = 1$ and σ^1 represent boolean values 0 and 1, respectively. Specifically, we define $\sigma := g$ in ElGamal encryption while $\sigma := h$ in sCS encryption. Cramer et al. [8] showed how to prove that the plaintext of an ElGamal cyphertext $A = (\alpha, \beta)$ is Boolean, i.e.,

$$\mathsf{Bool}(A) \stackrel{def}{=} PK\{r : \alpha = g^r, (\beta = y^r \text{ or } \beta = \sigma \cdot y^r)\}.$$

Proof of Equality/Inequality of Boolean Plaintext. Using ZK PoK for the discrete log it is easy to prove equality/inequality of the plaintexts of two ElGamal/sCS cyphertexts. Given the two cyphertexts $A = (\alpha, \beta)$ and $A' = (\alpha', \beta')$, let $(\epsilon, \delta) = (\alpha/\alpha', \beta/\beta')$, and let $(\mu, \nu) = (\alpha\alpha', \beta\beta'/\sigma)$. To prove equality of $D_x(A) = D_x(A')$, we give $PK\{e : y = g^e, \delta = \epsilon^e\}$ and denote such proof by $\mathsf{Eq}(A, A')$. To prove inequality of $D_x(A) \neq D_x(A')$, we give $PK\{e : y = g^e, \mu = \nu^e\}$ and denote such proof by $\mathsf{Neq}(A, A')$.

Shuffling Lists of Cyphertexts We adopt a protocol of [15] for non-interactively proving that two lists of cyphertexts are equivalent, and that one is a permutation of the other. We denote this protocol $\mathsf{Shuffle}$ and note that the length of the transcript of the protocol is linear with the number of cyphertexts. While the protocol of [15] is originally designed for ElGamal encryptions, it can be easily applied to sCS encryptions too.

I_L	I_R	O
A_1	B_1	C_1
A_2	B_2	C_2
A_3	B_3	C_3
A_4	B_4	C_4

I_L	I_R	O
A'_1	B'_1	C'_1
A'_2	B'_2	C'_2
A'_3	B'_3	C'_3
A'_4	B'_4	C'_4

I	O
A_1	B_1
A_2	B_2

Fig. 4. base NAND gate, NAND gate, and OUTPUT gate.

Correct Construction of NAND Gate. For an NAND gate, we give a two-part proof; the first part shows the structure of the gate. However this part leaks information on the truth table, thus the second part shuffles and re-encrypts the truth table entries. For two ElGamal/sCS cyphertexts $Y = (\alpha, \beta)$ and $Y' = (\alpha', \beta')$, denote $Y \oplus Y' = (\alpha\alpha', \beta\beta')$. Let $X = (1, 1/\sigma^2)$ be a trivial encryption of $1/\sigma^2$. We use the following fact to construct the base gate:

$$c = a \text{ NAND } b \iff a + b + 2(c - 1) \in \{0, 1\}.$$

The base NAND gate:

1. $\mathsf{Bool}(A_1), \ldots, \mathsf{Bool}(C_4)$
2. $\mathsf{Eq}(A_1, A_2)$, $\mathsf{Eq}(A_3, A_4)$, $\mathsf{Neq}(A_1, A_3)$

$\overline{T}_j$	I^L	I^R	O	K^L	K^R	$\mathsf{Plugs}_{[j\to\cdot]}$
1	$(\alpha_{j,11}, \beta_{j,11})$	$(\alpha_{j,12}, \beta_{j,12})$	$(\gamma_{j,1}, \delta_{j,1})$	$(\lambda_{j,11}, \mu_{j,11}), \kappa_{j,11}$	$(\lambda_{j,12}, \mu_{j,12}), \kappa_{j,12}$	$\widehat{\mathsf{Plugs}}_{[j\to\cdot]}(1)$
2	$(\alpha_{j,21}, \beta_{j,21})$	$(\alpha_{j,22}, \beta_{j,22})$	$(\gamma_{j,2}, \delta_{j,2})$	$(\lambda_{j,21}, \mu_{j,21}), \kappa_{j,21}$	$(\lambda_{j,22}, \mu_{j,22}), \kappa_{j,22}$	$\widehat{\mathsf{Plugs}}_{[j\to\cdot]}(2)$
3	$(\alpha_{j,31}, \beta_{j,31})$	$(\alpha_{j,32}, \beta_{j,32})$	$(\gamma_{j,3}, \delta_{j,3})$	$(\lambda_{j,31}, \mu_{j,31}), \kappa_{j,31}$	$(\lambda_{j,32}, \mu_{j,32}), \kappa_{j,32}$	$\widehat{\mathsf{Plugs}}_{[j\to\cdot]}(3)$
4	$(\alpha_{j,41}, \beta_{j,41})$	$(\alpha_{j,42}, \beta_{j,42})$	$(\gamma_{j,4}, \delta_{j,4})$	$(\lambda_{j,41}, \mu_{j,41}), \kappa_{j,41}$	$(\lambda_{j,42}, \mu_{j,42}), \kappa_{j,42}$	$\widehat{\mathsf{Plugs}}_{[j\to\cdot]}(4)$

$\overline{T}_i$	I^L	I^R	O	K^L	K^R	$\mathsf{Plugs}_{[i\to j]}$
1	$(\alpha_{i,11}, \beta_{i,11})$	$(\alpha_{i,12}, \beta_{i,12})$	$(\gamma_{i,1}, \delta_{i,1})$	$(\lambda_{i,11}, \mu_{i,11}), \kappa_{i,11}$	$(\lambda_{i,12}, \mu_{i,12}), \kappa_{i,12}$	$\widehat{\mathsf{Plugs}}_{[i\to j]}(1)$
2	$(\alpha_{i,21}, \beta_{i,21})$	$(\alpha_{i,22}, \beta_{i,22})$	$(\gamma_{i,2}, \delta_{i,2})$	$(\lambda_{i,21}, \mu_{i,21}), \kappa_{i,21}$	$(\lambda_{i,22}, \mu_{i,22}), \kappa_{i,22}$	$\widehat{\mathsf{Plugs}}_{[i\to j]}(2)$
3	$(\alpha_{i,31}, \beta_{i,31})$	$(\alpha_{i,32}, \beta_{i,32})$	$(\gamma_{i,3}, \delta_{i,3})$	$(\lambda_{i,31}, \mu_{i,31}), \kappa_{i,31}$	$(\lambda_{i,32}, \mu_{i,32}), \kappa_{i,32}$	$\widehat{\mathsf{Plugs}}_{[i\to j]}(3)$
4	$(\alpha_{i,41}, \beta_{i,41})$	$(\alpha_{i,42}, \beta_{i,42})$	$(\gamma_{i,4}, \delta_{i,4})$	$(\lambda_{i,41}, \mu_{i,41}), \kappa_{i,41}$	$(\lambda_{i,42}, \mu_{i,42}), \kappa_{i,42}$	$\widehat{\mathsf{Plugs}}_{[i\to j]}(4)$

Fig. 5. Variable-based representation of table $\overline{T}_i$ and $\overline{T}_j$.

3. $\mathsf{Eq}(B_1, B_3),\ \mathsf{Eq}(B_2, B_4),\ \mathsf{Neq}(B_1, B_2)$
4. $\mathsf{Bool}(A_i \oplus B_i \oplus C_i \oplus C_i \oplus X)$ for $i \in \{1, \ldots, 4\}$

The second and the third items show the input columns are valid. The last item shows the output columns are valid. Note that the proof in this step reveals some information such as equality of cyphertexts in the same column. Hence, the second part: $\mathsf{Shuffle}(\langle A_i, B_i, C_i\rangle_{i=1}^4, \langle A'_i, B'_i, C'_i\rangle_{i=1}^4)$.

Correct Construction of an OUTPUT Gate. The proof for the correct construction is as follows:

The OUTPUT gate:
1. $\mathsf{Bool}(A_1),\ \mathsf{Bool}(A_2),\ \mathsf{Bool}(B_1),\ \mathsf{Bool}(B_2)$
2. $\mathsf{Neq}(A_1, A_2),\ \mathsf{Eq}(A_1, B_1),\ \mathsf{Eq}(A_2, B_2)$

4.3 Correct Construction of Plugs

Structure of the Plug. We modify the structure of $\widehat{\mathsf{Plug}}_{[i\to j]}(v, w)$. a little bit in the full protocol. We assume that the output of the gate G_i and the left input of the gate G_j are connected together. See Figure 5 for the representation of the two tables $\overline{T}_i$ and $\overline{T}_j$. The plug is an encryption of *impCODE* transcript[4] for $c_1 = (\alpha_{j,w1}, \beta_{j,w1}),\quad c_2 = (\gamma_{i,v}, \delta_{i,v}),\quad c_3 = (\lambda_{j,w1}, \mu_{j,w1})$. The actual transcript will be of the following form $\mathsf{Plug}_{[i\to j]}(v, w) = \langle \epsilon, \zeta, D\rangle$, where

$$e \in_R Z_q,\quad \epsilon = \left(\frac{\alpha_{j,w1}}{\gamma_{i,v}}\right)^e,\ \zeta = \left(\frac{\beta_{j,w1}}{\delta_{i,v}}\right)^e,\ D = (\epsilon \cdot \lambda_{j,w1})^{x_A}\,.$$

Note that we don't have to encrypt ϵ or ζ; the exponent e for ϵ and ζ is already hard to find due to the hardness of DLP. So we only have to apply

[4] If the output of G_i were the right input of G_j, it would be $c_1 = (\alpha_{j,w2}, \beta_{j,w2}),\ c_2 = (\gamma_{i,v}, \delta_{i,v}),\ c_3 = (\lambda_{j,w2}, \mu_{j,w2})$.

ElGamal encryption to D. The plug now looks as follows:

$$\widehat{\mathsf{Plug}}_{[i\to j]}(v,w) = \langle \epsilon,\ \zeta,\ (g^r, D\cdot z_{i,v}^r)\rangle, \quad \text{where } r \in_R Z_q.$$

We denote the $(g^r, D\cdot z_{i,v}^r)$ by $(\eta, \tilde{D})$.

When Bob obtains the (decrypted) plug $\mathsf{Plug}_{[i\to j]}(v,w)$, he executes *imp-CODE* scheme and gets an output $\hat{k}$ by computing

$$\hat{k} = \frac{\zeta\cdot\mu_{j,w1}}{D\cdot D'}, \quad \text{where } D' = (\epsilon\cdot\lambda_{j,w1})^{x_B}\,.$$

He checks if $g^{\hat{k}} = \kappa_{j,w1}$ holds; if it holds, he decides that $(\alpha_{j,w1}, \beta_{j,w1}) \equiv (\gamma_{i,v}, \delta_{i,v})$.

$\mathsf{ZKVerify}$ **: Proof of Correct Plug Construction.** The goal of the $\mathsf{ZKVerify}$ proof is for Alice to prove that the encrypted $\mathcal{CODE}$ transcripts are valid. Specifically we show how to generate the proof for the plug $\widehat{\mathsf{Plug}}_{[i\to j]}(v,w)$. The plug is encrypted using a key $z_{i,v} = \kappa_{i,v1}\cdot\kappa_{i,v2} = g^{k_{i,v1}}\cdot g^{k_{i,v2}}$ (See Table 3 and 5 for notations), and the corresponding secret key is obtained by Bob only if he learns correctly $k_{i,v1}$ and $k_{j,v2}$ (this limits his computation to a single computational path in the circuit). $\mathsf{ZKVerify}$ proves two things: (1) given two *ciphertexts* $E_y(k_{i,v1}), E_y(k_{i,v2})$, the encrypted part of the plug, i.e., $(g^r, D\cdot z_{i,v}^r)$ is actually encrypted using the public key $z_{i,v}$; (2) she knows the discrete-log used in the rest part of the plug:

$$PK\left\{e : \epsilon = \left(\frac{\alpha_{j,w1}}{\gamma_{i,v}}\right)^e, \zeta = \left(\frac{\beta_{j,w1}}{\delta_{i,v}}\right)^e\right\}.$$

In the ElGamal based construction, we assume that both p and q are safe primes such that $p = 2q+1$ and $q = 2q'+1$ (i.e., p is a double decker). It is claimed that there are infinitely many such tuples of primes, and they are easy to find. We let $k_{i,v1} = f^{\tau_1}$, and $k_{i,v2} = f^{\tau_2}$, where f is a generator in $\mathcal{G}_{q'}$. The proof $\mathsf{ZKVerify}\left((\lambda_{i,v1}\ ,\ \mu_{i,v1}\ ,\ g^{\kappa_{i,v1}})\ ,\ (\lambda_{i,v2}\ ,\ \mu_{i,v2}\ ,\ g^{\kappa_{i,v2}})\ ,\ \widehat{\mathsf{Plugs}}_{[i\to j]}(v,w)\right)$ is as follows:

$$\begin{aligned} PK\Big\{(r_1,\tau_1,r_2,\tau_2,e,r_3,x_A):\ & \lambda_{i,v1} = g^{r_1},\ \mu_{i,v1} = f^{\tau_1}\cdot y^{r_1},\ \kappa_{i,v1} = g^{f^{\tau_1}}, \\ & \lambda_{i,v2} = g^{r_2},\ \mu_{i,v2} = f^{\tau_2}\cdot y^{r_2},\ \kappa_{i,v2} = g^{f^{\tau_2}}, \\ & \epsilon = (\alpha_{j,w1}/\gamma_{i,v})^e\ ,\ \zeta = (\beta_{j,w1}/\delta_{i,v})^e\,, \\ & y_A = g^{x_A},\ \eta = g^{r_3},\ \tilde{D} = z_{i,v}^{r_3}\cdot(\epsilon\cdot\lambda_{j,w1})^{x_A}\Big\}. \end{aligned}$$

The above proof uses proofs of knowledge of the double discrete log, which can be constructed by using Camenisch and Stadler [7]. They showed how to construct such proof in their paper, and this costs $\Theta(\ell)$ communication complexity (ℓ is security parameter).

For the sCS based protocol, $\mathsf{ZKVerify}$ is simpler, and we do not need to construct k_1, k_2 in a special form. The proof shows directly that $\kappa_{i,v1} = g^{k_{i,v1}}$. The proof $\mathsf{ZKVerify}$ is as follows:

$$PK\Big\{(r_1, \tau_1, r_2, \tau_2, e, r_3, x_A) : \lambda_{i,v1} = g^{r_1},\ \mu_{i,v1} = h^{k_{i,v1}} \cdot y^{r_1},\ \kappa_{i,v1} = g^{k_{i,v1}},$$
$$\lambda_{i,v2} = g^{r_2},\ \mu_{i,v2} = h^{k_{i,v2}} \cdot y^{r_2},\ \kappa_{i,v2} = g^{k_{i,v2}},$$
$$\epsilon = (\alpha_{j,w1}/\gamma_{i,v})^e,\ \zeta = (\beta_{j,w1}/\delta_{i,v})^e,$$
$$y_A = g^{x_A},\ \eta = g^{r_3},\ \tilde{D} = z_{i,v}^{r_3} \cdot (\epsilon \cdot \lambda_{j,w1})^{x_A}\Big\}.$$

Note, in the sCS based protocol, the $\mathsf{ZKVerify}$ proof does not include a double discrete log proof.

4.4 Protocol Details: Input Contribution

The parties contibuting inputs might be malicious. For example, an input contributed may generate a committed input by mauling other committed input. To avoid this kind of attack, the input contributors add non-malleable zero-knowledge proofs of knowledge to each of their committed input bits. In addition, the parties who manage the public directory that stores the committed inputs check the committed inputs and reject any inputs that have the same proofs.

4.5 Protocol Details: Alice

Alice sends the tables as in the honest-but-curious case, and in addition, for each gate G_i, she sends a proof of correct construction of the gate and of the plugs $\mathsf{ZKVerify}\left((\lambda_{i,v1}\ ,\ \mu_{i,v1}\ ,\ g^{\kappa_{i,v1}})\ ,\ (\lambda_{i,v2}\ ,\ \mu_{i,v2}\ ,\ g^{\kappa_{i,v2}})\ ,\ \widehat{\mathsf{Plugs}}_{[i \to j]}(v)\right)$, where by $\widehat{\mathsf{Plugs}}_{[i \to j]}(v)$ we mean the four encrypted pairs, one for each $\mathcal{CODE}$ transcript, which are all encrypted using $z_{i,v} = g^{\kappa_{i,v1} + \kappa_{i,v2}}$.

4.6 Protocol Details: Bob

In the full version of the protocol, Bob first verifies that all the proofs Alice sent are valid. That is, for each gate G_i Bob verifies that the proof of correct construction of the gate is valid. For each row v of table $\overline{T}_i$, Bob verifies that the proof for correct encryption of the plugs $\widehat{\mathsf{Plugs}}_{[i \to j]}(v)$ is valid. If any of the proofs is invalid, Bob aborts the protocol. Otherwise (if all proofs are valid), Bob continues as described in Section 3.

4.7 Communication Complexity in the Malicious Case

In addition to the communication costs of the garbled circuit, the malicious case incurs the complexity of sending the additional proofs. When ElGamal encryption is used, the total communication complexity costs are $\mathcal{O}((m \cdot \ell + n) \log p$ bits mainly due to proof of double discrete log. When sCS encryption is used, the total communication complexity costs are $\mathcal{O}((m + n) \log p)$ bits.

References

1. W. Aiello, Y. Ishai, and O. Reingold. Priced oblivious transfer: How to sell digital goods. *Advances in Cryptology — (EUROCRYPT 2001)*, 2045:119–135, 2001.
2. D. Beaver. Minimal-latency secure function evaluation. In B. Preneel, editor, *Advances in Cryptology — (EUROCRYPT 2000)*, volume 1807 of *Lecture Notes in Computer Science*, pages 335–350. Springer-Verlag, 2000.
3. D. Boneh and R. Lipton. Algorithms for black-box fields and their application to cryptography. In *Advances in Cryptology — (CRYPTO 1996)*, pages 283–297, 1996.
4. C. Cachin, J. Camensich, J. Kilian, , and J. Müller. One-round secure computation and secure autonomous mobile agents. In *Proc. 27th International Colloquium on Automata, Languages and Programming (ICALP)*, 2000.
5. J. Camenisch and M. Michels. Proving that a number is the product of two safe primes. In J. Stern, editor, *Advances in Cryptology — (EUROCRYPT 1999)*, volume 1592 of *Lecture Notes in Computer Science*, pages 107–122. Springer-Verlag, 1999.
6. J. Camenisch and V. Shoup. Practical verifiable encryption and decryption of discrete logarithms. In *Advances in Cryptology — (CRYPTO 2003)*, pages 126–144, 2003.
7. Jan Camenisch and Markus Stadler. Efficient group signature schemes for large groups. In Burt Kaliski, editor, *Advances in Cryptology — (CRYPTO 1997)*, volume 1296 of *Lecture Notes in Computer Science*, pages 410–424. Springer-Verlag, 1997.
8. R. Cramer, R. Genaro, and B. Schoenmakers. A secure and optimally efficient multi-authority election scheme. In W. Fumy, editor, *Advances in Cryptology — (EUROCRYPT 1997)*, volume 1233 of *Lecture Notes in Computer Science*, pages 103–118. Springer-Verlag, 1997.
9. R. Cramer, I. Damgård, and B. Schoenmakers. Proofs of partial knowledge and simplified design of witness hiding protocols. In Y.G. Desmedt, editor, *Advances in Cryptology — (CRYPTO 1994)*, volume 839 of *Lecture Notes in Computer Science*, pages 174–187. Springer-Verlag, 1994.
10. Whitfield Diffie and Martin E Hellman. New directions in cryptography. *IEEE Trans. on Information Theory*, IT-22(6):644–654, November 1976.
11. Taher ElGamal. A public key cryptosystem and a signature scheme based on discrete logarithms. In *IEEE Transactions on Information Theory*, volume 31, pages 469–472, 1985.
12. S. Even, O. Goldreich, and S. Micali. On-line/off-line digital schemes. In *Advances in Cryptology — (CRYPTO 1989)*, pages 263–275, 1989.
13. J. Feigenbaum and M. Merritt. Open questions, talk abstracts, and summary of discussions. In *DIMACS Series in Discrete Mathematics and Theoretical Computer Science*, pages 1–45, 1991.
14. A. Fiat and A. Shamir. How to prove yourself: Practical solutions to identification and signature problems. In J. L. Massey, editor, *Advances in Cryptology — (CRYPTO 1986)*, volume 263 of *Lecture Notes in Computer Science*, pages 186–194. Springer-Verlag, 1986.
15. J. Furukawa and K. Sako. An efficient scheme for proving a shuffle. In *Advances in Cryptology — (CRYPTO 2001)*, volume 2139 of *Lecture Notes in Computer Science*, pages 368–387, 2001.

16. Y. Gertner, Y. Ishai, E. Kushilevitz, and T. Malkin. Protecting data privacy in private information retrieval schemes. *Proceedings of the thirtieth annual ACM symposium on Theory of computing*, pages 151–160, 1998.
17. O. Goldreich, S. Micali, and A. Wigderson. How to play any mental game. In *Proc. 19th Annual ACM Symposium on Theory of Computing (STOC)*, pages 218–229. ACM Press, 1987.
18. Shafi Goldwasser and Silvio Micali. Probabilistic encryption. *Journal of Computer and System Sciences*, 28(2):270–299, April 1984.
19. Omer Horvitz and Jonathan Katz. Universally-composable two-party computation in two rounds. In *Advances in Cryptology — (CRYPTO 2007)*, pages 111–129, 2007.
20. S. Jarecki and V. Shmatikov. Efficient two-party secure computation on committed inputs. In *Advances in Cryptology — (EUROCRYPT 2007)*, 2007.
21. Y. Lindell and B. Pinkas. An efficient protocol for secure two-party computation in the presence of malicious adversaries. In *Advances in Cryptology — (EUROCRYPT 2007)*, 2007.
22. M. Naor, B. Pinkas, and R. Sumner. Privacy preserving auctions and mechanism design. In *1st ACM Conference on Electronic Commerce*, 1999.
23. Pascal Paillier. Public-key cryptosystems based on composite degree residuosity classes. In Jacques Stern, editor, *Advances in Cryptology — (EUROCRYPT 1999)*, volume 1592 of *Lecture Notes in Computer Science*, pages 107–122. Springer-Verlag, 1999.
24. R. Rivest, L. Adelman, and M.L. Dertouzos. On data banks and privacy homomorphisms. In R.A. DeMillo, D.P. Dobkin, A.K. Jones, and R.J. Lipto, editors, *Foundations of Secure Computation*, pages 169–17. Academic Press, 1978.
25. Amit Sahai. Non-malleable non-interactive zero knowledge and adaptive chosen-ciphertext security. In *Proc. 40th IEEE Symposium on Foundations of Computer Science (FOCS)*, pages 543–553, 1999.
26. T. Sander and C.F. Tschudin. Protecting mobile agents against malicious hosts. In G. Vigna, editor, *Mobile Agents and Security*, volume 1419 of *Lecture Notes in Computer Science*, pages 44–61. Springer-Verlag, 1998.
27. T. Sander, A. Young, and M. Yung. Non-interactive cryptocomputing for NC^1. In *Proc. 40th IEEE Symposium on Foundations of Computer Science (FOCS)*, pages 554–567, 1999.
28. A. De Santis and G. Persiano. Zero-knowledge proofs of knowledge without interaction. In *Proc. 33rd IEEE Symposium on Foundations of Computer Science (FOCS)*, pages 427–437, 1992.
29. Alfredo De Santis, Giovanni Di Crescenzo, Giuseppe Persiano, and Moti Yung. On monotone formula closure of SZK. In *Proc. 35th IEEE Symposium on Foundations of Computer Science (FOCS)*, pages 454–465. IEEE Computer Society Press, 1994.
30. C.P. Schnorr. Efficient signature generation by smart cards. In *Journal of Cryptology*, volume 4, pages 161–174, 1991.
31. L. Valiant. Universal circuits. In *Proc. 8th Annual ACM Symposium on Theory of Computing (STOC)*, pages 196–203, 1976.
32. A.C. Yao. How to generate an exchange secrets. In *Proc. 27th IEEE Symposium on Foundations of Computer Science (FOCS)*, pages 162–167, 1986.