Efficient Two-Party Secure Computation on Committed Inputs

Stanisław Jarecki and Vitaly Shmatikov

University of California, Irvine
The University of Texas at Austin

Abstract. We present an efficient construction of Yao's "garbled circuits" protocol for securely computing any two-party circuit on committed inputs. The protocol is secure in a universally composable way in the presence of *malicious* adversaries under the decisional composite residuosity (DCR) and strong RSA assumptions, in the common reference string model. The protocol requires a constant number of rounds (four-five in the standard model, two-three in the random oracle model, depending on whether both parties receive the output), $O(|C|)$ modular exponentiations per player, and a bandwidth of $O(|C|)$ group elements, where $|C|$ is the size of the computed circuit.

Our technical tools are of independent interest. We propose a homomorphic, semantically secure variant of the Camenisch-Shoup verifiable cryptosystem, which uses shorter keys, is *unambiguous* (it is infeasible to generate two keys which successfully decrypt the same ciphertext), and allows efficient proofs that a committed plaintext is encrypted under a *committed key*.

Our second tool is a practical four-round (two-round in ROM) protocol for *committed* oblivious transfer on *strings* (string-COT) secure against malicious participants. The string-COT protocol takes a few exponentiations per player, and is UC-secure under the DCR assumption in the common reference string model. Previous protocols of comparable efficiency achieved either committed OT on *bits*, or standard (non-committed) OT on strings.

1 Introduction

Informally, a two-party protocol for computing a circuit is *secure* if participants do not learn anything from the protocol execution beyond what is revealed by the output of the circuit. In a seminal paper, Andrew Yao showed a "garbled circuit" protocol [Yao86] for secure two-party computation (2PC) of any circuit in the *semi-honest* model, *i.e.*, assuming that participants faithfully follow the protocol specification. Yao's protocol requires $O(|C|)$ symmetric-key operations, and its bandwidth is $O(|C|)$ symmetric-key ciphertexts, in addition to the cost of n instances of an oblivious transfer (OT) protocol, where n is the size of the circuit's inputs. Using a 2-round OT protocol, Yao's protocol takes only two communication rounds (assuming only one player receives the output).

The main contribution of this paper is a new variant of Yao's protocol, which replaces $O(|C|)$ symmetric-key operations with $O(|C|)$ public-key operations, and at this cost achieves security against *malicious* participants in the common reference string (CRS) model. Specifically, our protocol operates on a multiplicative group $\mathbb{Z}^*_{n^2}$ where n is a safe RSA modulus which satisfies DCR and strong RSA assumptions. The protocol requires $O(|C|)$ modular exponentiations, its bandwidth is $O(|C|)$ elements in $\mathbb{Z}_{n^2}^*$, and it takes four rounds in the standard model and two in ROM. Moreover, our protocol is universally composable, and securely computes any circuits on *committed* inputs.

A fundamental primitive in Yao's protocol is *oblivious transfer* (OT). Informally, OT is a two-party protocol in which the receiver (a.k.a. the "chooser") receives a value of his choice from among several values sent by the sender, while learning nothing about the other values. The sender does not learn anything from the protocol, and in particular he does not learn which of the values he sent was received by the chooser. *Committed oblivious transfer* (COT) is a variant of oblivious transfer, introduced by Crépeau [Cré89] as a "verifiable OT," in which both the sender and the chooser are committed to their inputs, and the oblivious transfer proceeds on the committed values. The second contribution of our paper is a new protocol for committed oblivious transfer on *strings* ("string-COT"). The protocol requires $O(1)$ exponentiations and has the bandwidth of $O(1)$ elements in $\mathbb{Z}_{n^2}^*$, which is comparable to the cost of previous protocols for standard (non-committed) OT on strings or previous COT protocols that operated only on bits. This new string-COT protocol is also universally composable in the CRS model.

A *committed* OT protocol secure against malicious players is a much more useful tool in a security protocol than a standard OT. For example, unless the OT protocol runs on committed inputs, it is fundamentally non-robust against network failures because re-running the protocol after a failure allows the cheating receiver to learn both of the sender's values. Similarly, secure *committed 2PC* protocol is a much more useful tool than a standard 2PC protocol. In general, universally composable string-OT and general 2PC *on committed data* makes it easy to ensure that multiple instances of these protocols are executed on consistent inputs, for example as prescribed by some larger protocol.

Technical roadmap. Both protocols we present in this paper, the protocol for secure two-party computation on committed inputs ("committed 2PC") and the string-COT protocol, rely on a modification of the verifiable encryption given by Camenisch and Shoup [CS03]. The efficiency of these two protocols is essentially due to the very strong properties that this encryption offers. We will refer to the original scheme of [CS03] as *CS encryption*, and we call our modification *sCS encryption*, where "s" stands for both "short" and "simplified," because the modification consists of (1) stripping off the chosen-ciphertext security check in the CS encryption, and (2) using significantly shorter private keys. Below we explain how several interesting properties of this encryption enable the efficient string-COT and committed 2PC protocols.

The sCS encryption scheme is *additively homomorphic*, *i.e.*, given ciphertexts of two values, one can obtain a ciphertext of their sum without decrypting the ciphertexts, and it is *verifiable*, *i.e.*, there is a very efficient ZK proof system due to [CS03] for showing that the encrypted message corresponds to a previously committed one. These two features together enable an efficient string-COT protocol. First, we use additive homomorphism of the sCS encryption to build an efficient protocol for OT on strings in a way that is similar to how Aiello *et al.* [AIR01] build a standard (*i.e.*, non-committed) OT on strings from the multiplicatively homomorphic ElGamal encryption. Then, by adapting the ZK proof systems given for the CS encryption in [CS03], we add efficient

ZK proofs for showing that the parties run this string-OT protocol on the previously committed inputs.

The sCS encryption has further useful properties which allow us to extend the string-COT protocol to an efficient committed 2PC protocol. First, it is *unambiguous*, in the sense that it is committing not only to the plaintext, but also to the encryption key: it is infeasible to produce a ciphertext that can be successfully decrypted, even to the same plaintext, under two different decryption keys. This property is crucial in the maliciously secure version of Yao's protocol. Otherwise, the player who creates the garbled circuit could embed all sorts of faults into the circuit. If the circuit evaluator encounters a fault which causes him to stop, the malicious player will learn information about the evaluator's inputs that he is not supposed to learn.

Second, we extend the Camenisch-Shoup ZK proof system to an efficient ZK proof that *a ciphertext encrypts a committed plaintext under a committed key*. (Technically, this proof system is defined for a symmetric-key version of the sCS encryption, where the key is both an encryption and a decryption key.) This proof system is a crucial component of proving that Yao's "garbled circuit" is formed correctly. Yao's construction of the garbled circuit involves encrypting, for every circuit gate, the keys corresponding to the output wires under the keys corresponding to the input wires. In our version of Yao's protocol, the sender commits to the keys he created for every circuit wire. For the wires corresponding to the receiver's inputs, the sender sends to the receiver the appropriate key values using our efficient string-COT protocol operating on these commitments. Furthermore, the sender must prove, for each gate, that the ciphertexts that are supposed to encrypt the appropriate output-wire keys under the appropriate input-wire keys are formed correctly. This is accomplished precisely by the above proof system, because the input-wire keys appear as *keys* in these ciphertexts, while the output-wires keys appear as *plaintexts*.

Giving an efficient ZK proof system for this statement for some version of the CS encryption scheme is an interesting technical challenge, because in the CS cryptosystem plaintexts and keys "live" in different groups (and are acted upon by different moduli). It is not immediately obvious how to encrypt one CS encryption key under another CS encryption key and have an efficient proof of correctness for this encryption, because the efficient proof systems given for the CS encryption require that the plaintext be significantly smaller than the encryption key. One solution is to extend these proof systems to handle larger plaintexts (namely, plaintexts of the same size as the key), using proofs of equality of elements of two different groups represented as integers (*e.g.*, [Bou00]). We propose a simpler solution based on the observation that, from the results of Håstad, Schrift and Shamir [HSS93] on simultaneous bit security of exponentiation in groups of unknown order, it follows that one can shorten the private keys used in the CS encryption to $\frac{|n|}{2}$ bits. This significantly speeds up the CS encryption, but, more importantly, this modification allows for a very efficient ZK proof that a ciphertext encrypts a *committed plaintext* under a *committed key*.

Organization of the paper. In Section 2 we discuss related work. In Section 3, we describe our cryptographic toolkit. In Section 4, we present the string-COT protocol, and in Section 5, the protocol for general two-party secure computation on committed inputs. All proofs have been delegated to the full version of the paper.

2 Related Work on Constant-Round 2PC and Committed OT

2PC protocols. The first constructions for secure two-party computation are Yao's "garbled circuits" protocol [Yao86] and the protocol of [GMW87]. Of the two, only Yao's protocol is constant-round, but secure only in the semi-honest model. Most subsequent constant-round protocols for secure computation in the malicious model, such as [Kil88, Lin03, KO04], employ generic zero knowledge proofs (*i.e.*, proofs for any NP statement). The overhead of this approach is likely to remain prohibitive for practical applications.

There are secure 2PC protocols that avoid generic zero-knowledge proofs (*e.g.*, see [JJ00, GMY04] and references therein), but the round complexity of these protocols is linear in the (boolean or arithmetic) circuit depth. On the other hand, Damgård and Ishai [DI05] showed the first constant-round *multi-party* protocol with $O(|C|n^2k)$ bandwidth and computation (here n is the number of parties, k is the security parameter), assuming a trusted preprocessing stage, but this protocol is secure only with an honest majority, and its techniques (*e.g.*, verifiable secret sharing) do not seem applicable to two-party computation.

2PC using verifiable encryption. Like our protocol, the constant-round 2PC protocol of Cachin and Camenisch [CC00] uses a verifiable public-key encryption scheme, but unlike in our scheme, their zero-knowledge proofs require s cut-and-choose repetitions where s is the statistical security parameter. Hence their 2PC protocol requires $O(s|C|)$ group elements in bandwidth and the same number of exponentiations (vs. $O(|C|)$ in our construction). It is worth mentioning, however, that our ciphertexts are elements of $\mathbb{Z}^*_{n^2}$, for n satisfying the DCR and strong RSA assumptions, while [CC00] can use any group where the Diffie-Hellman assumption holds.

2PC using cut-and-choose approach. A recent series of works on efficient constant-round 2PC protocols [Pin03, MF06, LP07, Woo07] shows that security in the malicious model can be achieved by cut-and-choose verification of the entire garbled circuit, at the cost of $O(s|C| + s^2n)$ [LP07] or $O(s|C|)$ [Woo07] symmetric-key operations, where s is the statistical security parameter of cut-and-choose and n is the input size. These cut-and-choose constructions probably require less computation than our protocol to achieve similar levels of security based on common assumptions, but our protocol may require less bandwidth, especially for small circuits whose size is comparable to the input size. Also, our protocol can be made non-interactive in the random oracle model at no extra cost, while the security parameter s in the cut-and-choose solutions increases if they are made non-interactive using the Fiat-Shamir heuristic.

COT. Committed OT (COT) was introduced by Crépeau [Cré89], where it was used to construct a general 2PC protocol (but not constant-round one) following the approach of [GMW87]. Crépeau constructed COT using black-box invocations of $\Omega(n^3)$ OTs. This was improved by [CvdGT95] to $O(n)$ OT's and $O(n^2)$ bit commitments. Both COT protocols, however, operate on bits rather than strings. Based on the concrete assumptions of Computational or Decisional Diffie-Hellman, Cramer and Damgård [CD97] and then Garay *et al.* [GMY04] give COT protocols which require $O(1)$ exponentiations but still operate only on bits, while Camenisch and Cachin [CC00] give a string-COT protocol, but it requires $O(k)$ modular exponentiations where k is the security parameter.

Lipmaa [Lip03] proposed to extend the (non-committed) string-OT protocol of Aiello *et al.* [AIR01] to a committed OT protocol on strings at the cost of $O(1)$ exponentiations. While this protocol does ensure that the *received* string is consistent with the sender's commitment, the sender can successfully cheat on the string that has *not* been transferred during the OT. This can be used to break chooser's privacy in any application (such as 2PC) where the sender can observe whether the chooser succesfully completed the protocol. Stronger verifiability can potentially be achieved by extending this protocol with zero-knowledge proofs, but the resulting protocol would not beat the $O(k)$ modular exponentiations bound because the commitment schemes (*e.g.*, [CGHGN01]) suggested in [Lip03] seem to have only cut-and-choose ZK proofs.

3 Cryptographic Tools

3.1 Camenisch-Shoup (CS) Encryption Scheme [CS03]

Common reference string. A trusted third party generates a safe RSA modulus $n = pq$, where $p = 2p' + 1$, $q = 2q' + 1$, $|p| = |q|$, $p \neq q$, and p, q, p', q' are all primes, a random element g' in $\mathbb{Z}^*_{n^2}$ and an element $g = (g')^{2n}$. The common reference string is (n, g), which also implicitly defines element $\alpha = 1 + n$. For standalone applications of CS encryption, pair (n, g) can be thought of as part of the public key. However, placing (n, g) in the CRS enables soundness of some very useful proof systems associated with this encryption scheme, *e.g.*, those used in our COT and 2PC protocols.

The group $\mathbb{Z}^*_{n^2}$ defined by the safe RSA modulus n can be decomposed into a cross-product of four subgroups: $\mathbb{Z}^*_{n^2} = G_n \times G_{n'} \times G_2 \times T$, where group G_n, generated by $\alpha = n + 1$, has order n, group $G_{n'}$ has order $n' = p'q'$, and G_2 and T are subgroups of order 2. As one consequence of this structure of $\mathbb{Z}^*_{n^2}$, the above procedure of picking g as a $2n$-power of a random element implies that, with an overwhelming probability, g is a generator of subgroup $G_{n'}$. In the following we treat all multiplications and exponentiations as operations in $\mathbb{Z}^*_{n^2}$, unless stated otherwise.

Key generation. The private key is a random triple x_1, x_2, x_3 chosen in $[0, \frac{n^2}{4}]$. The public key is $PK = (n, g, \mathfrak{g}, \mathfrak{h}, \mathfrak{f}, \mathsf{hk})$ where $\mathfrak{g} = g^{x_1}$, $\mathfrak{h} = g^{x_2}$, $\mathfrak{f} = g^{x_3}$, and hk is a key of a collision-resistant keyed hash function $\mathcal{H}$.

Encryption. Consider plaintext m as an integer in $[-\frac{n}{2}, \frac{n}{2}]$. (Note that one can encode elements m' in $\mathbb{Z}_n$ in this range as $m = m' \text{ rem } n$, *i.e.*, $m = m'$ if $m' \leq \frac{n}{2}$ and $m = m' - n$ if $m' > \frac{n}{2}$. Observe that $m = m' \mod n$.) A CS encryption of m under key PK with label L, denoted $\mathsf{CSenc}^L_{PK}(m)$, is a tuple (u, e, v) where $u = g^r$, $e = \alpha^m \mathfrak{g}^r$, and $v = \text{abs}((\mathfrak{h}\mathfrak{f}^{\mathcal{H}_{\mathsf{hk}}(u,e,L)})^r)$, for a randomly chosen $r \in [0, \frac{n}{4}]$. Operation $abs(a)$ returns a for $a < \frac{n}{2}$ and $n - a$ for $a \geq \frac{n}{2}$.

Decryption. Given a ciphertext (u, e, v), check $abs(v) = v$ and $u^{2(x_2 + \mathcal{H}_{\mathsf{hk}}(u,e,L)x_3)} = v^2$. If this holds, compute $\hat{m} = (e/u^{x_1})^2$. Note that $e/u^{x_1} = \alpha^m$ for correctly formed ciphertexts. If $\hat{m} \notin \langle\alpha\rangle$, *i.e.*, if n does not divide $\hat{m} - 1$, reject. Otherwise, set $\hat{m}' = \frac{\hat{m}-1}{n}$ (over the integers), $m' = \hat{m}'/2 \mod n$, and $m = m' \text{ rem } n$.

This encryption is CCA secure under the DCR assumption on safe RSA moduli [CS03]:

Assumption 1. (DCR) *[Pai99]: Given RSA modulus n, random elements of $\mathbb{Z}^*_{n^2}$ are computationally indistinguishable from elements of a subgroup formed by n-th powers of elements in $\mathbb{Z}^*_{n^2}$.*[1]

3.2 Simplified Camenisch-Shoup (sCS) Encryption Scheme

The group setting (n, g) is the same. Denote $k'' = \frac{|n|}{2}$, and let k, k' be parameters that control the quality of soundness and zero-knowledge of proof systems associated with the sCS encryption. We require that $2k + k' < k''$ and $k < p', q'$. For 80-bit security, one can take $k'' = 512$ and $k = k' = 80$.

Key generation. The private key is $x \in [0, 2^{k''}]$. The public key is $y = g^x$.

Encryption. The sCS encryption under key y of m, an integer in $[-\frac{n}{2}, \frac{n}{2}]$, denoted $\mathsf{sCSenc}_y(m)$, is (u, e) s.t. $e = \alpha^m y^r \bmod n^2$ and $u = g^r$ for a random r in $[0, \frac{n}{4}]$.

Decryption. Proceeds exactly like CS decryption, but omitting the CCA checks on v (since there's no v here), and using x instead of x_1 in decrypting (u, e).

Apart from stripping the CCA check, the only difference between CS and sCS encryption is the shortened private key. The fact that the scheme remains semantically secure with such modification follows from adapting the results of [HSS93] on simultaneous bit security of exponentiation modulo a Blum integer (and a safe RSA modulus is Blum integer) to exponentiation in $\mathbb{Z}^*_{n^2}$.[2] It follows that under the factoring assumption, the entire upper half of the bits of exponent x is simultaneously hidden under the exponentiation function $y = g^x \bmod n^2$, and therefore key $y = g^x$ for x random in $\mathbb{Z}_{n'}$ is indistinguishable from $y = g^x$ for x random in $[0, \frac{|n|}{2}]$. [3]

Theorem 1. *sCS encryption is semantically secure under DCR assumption on safe RSA moduli.*

Symmetric-key version of sCS encryption scheme. The sCS cryptosystem can also be used as a symmetric encryption scheme if the private key $x \in [0, 2^{k''}]$ is treated as a symmetric key. Encryption of m under key x is a pair (e, u), where $e = \alpha^m u^x \bmod n^2$, $u = g^r$ for random $r \in [0, \frac{n}{4}]$. The decryption procedure does not change, nor does the security of the encryption scheme.

Unambiguity of sCS encryption. We introduce a very strong notion of *unambiguous encryption*, which applies to both public-key and symmetric schemes. It says that a ciphertext that passes a certain proof system, denoted $\mathsf{ZKUnEnc}$, cannot decrypt to two different plaintexts under two different private keys. Moreover, no two distinct decryption keys can decrypt a ciphertext even to the same plaintext. Therefore, in an unambiguous encryption scheme, the ciphertext is committing not only to the plaintext, but also to the decryption key. This notion of encryption unambiguity is essential for our

[1] For the safe RSA moduli n, the subgroup of n-th residues in $\mathbb{Z}^*_{n^2}$ is the subgroup $G_{n'} \times G_2 \times T$.

[2] Cf. similar observation in [CGHG01] for Paillier encryption, on which CS encryption is based.

[3] Note that in this way one can also shorten keys x_2, x_3 in CS encryption and the randomness r.

version of Yao's 2PC protocol, because otherwise a malicious creator of the garbled circuit could introduce errors in this circuit, and then learn something extra about the receiver's inputs by observing whether the receiver successfully completes his computation on this circuit.

Definition 1. *An encryption scheme is* unambiguous *if there exists a zero-knowledge proof system $\mathsf{ZKUnEnc}$ s.t. for every efficient probabilistic algorithm A, the following event has only negligible probability: (1) A outputs tuple (c, x_1, x_2) s.t. $x_1 \neq x_2$, (2) A passes the $\mathsf{ZKUnEnc}$ proof system on ciphertext c, (3) x_1, x_2 are valid private keys,* i.e., *they are accepted by the decryption procedure, and (4) both $Dec_{x_1}(c)$ and $Dec_{x_2}(c)$ output a valid message (or messages). In the CRS model, the probability is also taken over the randomness of the common reference string generation.*

Theorem 2. *sCS encryption is unambiguous under the factoring assumption on safe RSA moduli, in the CRS model.*

The ZK proof system $\mathsf{ZKUnEnc}$ for the sCS encryption is the proof that u^2 belongs to the group generated by g, *i.e.*, $\mathsf{ZKUnEnc}(u, e) = \mathsf{ZKDL}(g, u)$. (See section 3.4.)

3.3 CS Commitments and sCS Commitments

Our COT and 2PC protocols could be adapted to work with standard Pedersen-like commitment schemes of [Ped91, FO97, DF02] at the cost of additional mappings, via range proofs [CM99, Bou00, DF02], between commitments with different ranges of plaintexts. Instead, we use the full (*i.e.*, adaptive chosen-ciphertext secure) CS encryption as a commitment scheme, because it operates on the same group as the encryption we use, and hence is well-suited for both the COT and 2PC protocols of Sections 4 and 5.[4] Moreover, using a CCA-secure encryption as a commitment helps in showing that the COT and 2PC schemes are secure in the strong sense of universal composability.

An instance of a CS commitment scheme is a CS encryption public key $PK = (n, g, \mathfrak{g}, \mathfrak{h}, \mathfrak{f}, \mathsf{hk})$. The public key is chosen by a trusted third party, and security of this commitment scheme requires the CRS model. The CS commitment on message m, an integer in range $[-\frac{n}{2}, \frac{n}{2}]$ (with an obvious mapping to $\mathbb{Z}_n$), with label L, is the ciphertext $\mathsf{Com} = \mathsf{CSenc}^L_{PK}(m)$. For notational convenience of the COT and 2PC protocols, we denote the tuple forming commitment Com as (u, C, v), *i.e.*, $u = g^r$, $C = \alpha^m \mathfrak{g}^r$, and $v = \text{abs}((\mathfrak{h}\mathfrak{f}^{\mathcal{H}_{\mathsf{hk}}(u,C,L)})^r)$. The decommitment is the (r, m, L) tuple. In the COT and 2PC protocols, we often treat value C in the CS commitment as a commitment to m by itself. This shortened commitment is used very heavily in the 2PC protocol, thus we refer to value $C = \alpha^m \mathfrak{g}^r$ by itself as an *sCS commitment*. The corresponding decommitment is (m, r).

3.4 Efficient Concurrently Secure ZK Proof Systems in the CRS Model

All proof systems used in our COT and Committed 2PC protocols are concurrently secure ZK proofs in the CRS model. Specifically, each proof system is computationally

[4] Note that instances of other commitment schemes can be mapped to this one using the verifiable encryption proof system that accompanies the Camenisch-Shoup encryption [CS03].

sound and statistical zero-knowledge with a straight-line simulator. The latter is important for showing that the protocols are universally composable. Each of these proof systems is built from efficient HVZK proof systems for the languages listed below by a series of compilations which preserve the efficiency of the underlying HVZK protocols.

The compilations start from 3-round HVZK proof systems with the properties of *special honest-verifier zero-knowledge* and *(weak) special soundess* (we discuss these below). First, with the techniques of Cramer *et al.* [CDS94], HVZK systems of this class can be combined, at no extra cost, into HVZK proof systems of the same class for any (monotonic) disjunctive and/or conjuctive formula over statements proved in the component proof systems. Then, using Damgård [Dam02], the resulting HVZK proof system can be compiled into a three-round concurrently secure ZK proof systems with statistical zero-knowledge, computational soundness, and a straight-line simulator in the CRS model. This latter technique requires statistically hiding trapdoor commitments, and using Pedersen's commitment scheme it incurs a computational overhead of just one extra exponentiation per player. The computational soundness of the resulting ZK proof system is subject to the same assumption as the computational binding of the commitment scheme, which can be Strong RSA if Pedersen's trapdoor commitment is adapted to the $\mathbb{Z}^*_{n^2}$ setting, *e.g.*, as in Damgård-Fujisaki commitments [DF02]. Note that in ROM, using the Fiat-Shamir heuristic, the HVZK proof systems of this class can be converted at no extra cost to *non-interactive* ZKs with the same properties of computational soundness and statistical zero-knowledge with straight-line simulation.

We denote the statements being proved as X, Y, Z, and the corresponding "atomic" HVZK proof systems as HVZKX, HVZKY, HVZKZ. We use a notation derived from boolean formulas for the ZK proof systems resulting from this series of compilations. For example, the resulting ZK proof system for language $\mathsf{X} \wedge (\mathsf{Y} \vee \mathsf{Z})$ will be denoted $\mathsf{ZKX} \wedge (\mathsf{ZKY} \vee \mathsf{ZKZ})$. We catalog the proof systems used in the COT and 2PC protocols by the statements they prove, namely, membership in the languages DL, DLEQ, NotEq, Cot, Com, and $\mathsf{PlainEq}$. Each of these is parameterized by tuple $(n, g, \mathfrak{g}, \mathfrak{h}, \mathfrak{f}, \mathsf{hk})$, which forms an instance of the CS commitment scheme. Triple $(n, g, \mathfrak{g})$ also defines an instance of the sCS commitment. Parameters k, k', k'' are as in Section 3.2.

$\mathsf{DL} = \{(g, X) \mid \text{ there exists } x \text{ s.t. } X^2 = g^{2x}\}$.

$\mathsf{DLEQ} = \{(g, X, \tilde{g}, \tilde{X}) \mid \text{ there exists } x \text{ s.t. } X^2 = g^{2x}, \tilde{X}^2 = \tilde{g}^{2x}\}$.

$\mathsf{NotEq} = \{(C_a, C_b) \mid \text{ there exist } a, b, r_a, r_b \text{ s.t. } a \neq b \bmod n,\ C_a = \alpha^a \mathfrak{g}^{r_a}, \text{ and } C_b = \alpha^b \mathfrak{g}^{r_b}\}$. In other words, C_a and C_b are sCS commitments to two different values.

$\mathsf{Cot} = \{(i, e', u', e, u, y, C) \mid \text{ there exist } m, w, s, r \text{ s.t. } C^2 = \alpha^{2m} \mathfrak{g}^{2w},$ $e'^2 = e^{2s} \alpha^{2m - i * 2s} y^{2r}, \text{ and } u'^2 = u^{2s} g^{2r}\}$. In other words, $m \text{ rem } n$ is committed in sCS commitment C, and (u', e') is a correct "re-encryption" of m performed by the sender in the COT protocol, given the (y, u, e) tuple sent by the receiver.

$\mathsf{Com} = \{(\mathsf{Com}, \mathsf{ids}) \mid \text{ there exist } m, r \text{ s.t. } \mathsf{Com} = (u, C, v) \text{ where } u = g^r, C = \alpha^m \mathfrak{g}^r, \text{ and } v = \text{abs}((\mathfrak{h}\mathfrak{f}^{\mathcal{H}_{\mathsf{hk}}(u, C, \mathsf{ids})})^r)\}$. In other words, Com is a properly formed CS commitment to some message m with label ids.

$\mathsf{PlainEq} = \{((e,u), C_x, C_m) \mid$ there exist x, m, r_x, r_m s.t. $e = \alpha^m u^x$, $C_x = \alpha^x \mathfrak{g}^{r_x}$, and $C_m = \alpha^m \mathfrak{g}^{r_m}\}$. In other words, (e,u) is an sCS encryption of the plaintext m committed in (sCS commitment) C_m under the key x committed in C_x.

All of the above languages have efficient 3-round HVZK proof systems HVZKDL, $\mathsf{HVZKDLEQ}$, *etc.*, which unconditionally satisfy the two properties we need: (1) special HVZK, and (2) weak special soundness. The only exception is $\mathsf{HVZKPlainEq}$, for which we show that weak special soundness holds under the strong RSA assumption. All systems are efficient: the players make only a few exponentiations (between one and four) modulo n^2, and communication complexity ranges from $3|n|$ in HVZKDL to at most $20|n|$ bits in $\mathsf{HVZKPlainEq}$. We show the $\mathsf{HVZKPlainEq}$ proof system in Appendix A, because it has the most novelty. We delegate the other proof systems to the full version of the paper, but most of them are either standard, or simple modifications of the proofs that appear in [CS03]. The $\mathsf{HVZKPlainEq}$ proof system shown in Appendix A gives a good idea of how all of these HVZKs work.

Special HVZK and (weak) special soundness. Let (P_1, P_2, V) be a specification of a 3-round public coin proof system for language L. The prover's message in the first round on instance x, witness w for $x \in L$, and randomness r is computed as $a = P_1(x,w,r)$, its response in the third round is computed as $z = P_2(x,w,r,e)$ where e is the verifier's challenge, and the verifier accepts if and only if $V(x,a,e,z) = 1$. We call this proof system *special (statistical) HVZK* if there exists a simulator S s.t. for every challenge e and every witness (x,w) for $x \in L$, the tuple (a,z) output by $S(z,e)$ is distributed statistically close to tuple (a,z) where $a = P_1(x,w,r)$ and $z = P_2(x,w,r,e)$. The probability is over the coins of S and over r. We say that this proof system has *(weak) special soundness* if for every $x \notin L$, and for every PPT algorithm $\hat{P}$, the probability that $\hat{P}(x)$ outputs (a,e,z,e',z') s.t. $e \neq e'$ and $V(x,a,e,z) = V(x,a,e',z') = 1$, is negligible. Since the HVZK proof systems we use are parametrized by a reference string, the adversary $\hat{P}$ takes the CRS as an input and the probability is taken over the choice of the CRS and the adversary's coins. This notion of *(weak) special soundness* is weaker than the *special soundness* assumed by the compilers of [CDS94, Dam02], but it's easy to see that the same compilers still apply to this weaker class of HVZKs.

4 UC-Secure Committed Oblivious Transfer on Strings

Our protocol $\mathcal{P}_{\text{cot}}$ for 1-out-of-2 committed oblivious transfer (COT) on strings is similar to the 1-out-of-2 non-committed string-OT protocol of Aiello *et al.* [AIR01], but instead of multiplicatively homomorphic ElGamal encryption, $\mathcal{P}_{\text{cot}}$ uses *additively* homomorphic and *verifiable* sCS encryption, which enables succinct (constant number of exponentiations) proofs that receiver's and sender's inputs into OT match their previous commitments. Moreover, $\mathcal{P}_{\text{cot}}$ is universally composable in the CRS model.

We define the ideal functionality $\mathcal{F}_{\text{COT}}$ for a COT scheme, and show that $\mathcal{P}_{\text{cot}}$ securely realizes it. In contrast to the ideal COT functionality proposed by Garay *et al.* [GMY04], our functionality $\mathcal{F}_{\text{COT}}$ runs on *strings* rather than bits. However, $\mathcal{F}_{\text{COT}}$ is more restricted than the functionality of [GMY04] in that (1) the obliviously

Ideal functionality $\mathcal{F}_{\mathrm{COT}}$ for committed oblivious transfer on strings (COT)

Commit: Upon receiving a $\langle \mathsf{ComMsg}, (P_i, cid), m\rangle$ message from P_i, $\mathcal{F}_{\mathrm{COT}}$ records the $((P_i, cid), m)$ pair and broadcasts $\langle \mathsf{Committed}, (P_i, cid)\rangle$. Here m can be either a message in the prescribed message space or a special symbol $\perp$.

StartCOT: Upon receiving $msg = \langle \mathsf{StartCOT}, (P_S, P_R, sid, cid_R, cid_{S,0}, cid_{S,1})\rangle$ from P_R, $\mathcal{F}_{\mathrm{COT}}$ verifies that it has records $((P_R, cid_R), m_R)$, $((P_S, cid_{S,0}), m_{S,0})$, and $((P_S, cid_{S,1}), m_{S,1})$, and that $m_R \neq \perp$. If this fails, $\mathcal{F}_{\mathrm{COT}}$ ignores this message; otherwise, $\mathcal{F}_{\mathrm{COT}}$ records msg and forwards it to P_S.

CompleteCOT: Upon receiving $\langle \mathsf{CompleteCOT}, (P_S, P_R, sid, cid_R, cid_{S,0}, cid_{S,1})\rangle$ from P_S, $\mathcal{F}_{\mathrm{COT}}$ verifies that it has a record $\langle \mathsf{StartCOT}, \mathsf{ids}\rangle$, where $\mathsf{ids} = (P_S, P_R, sid, cid_R, cid_{S,0}, cid_{S,1})$. $\mathcal{F}_{\mathrm{COT}}$ looks up records $((P_S, cid_{S,0}), m_{S,0})$ and $((P_S, cid_{S,1}), m_{S,1})$, and checks if $m_{S,0} \neq \perp$ and $m_{S,1} \neq \perp$. If anything fails, $\mathcal{F}_{\mathrm{COT}}$ ignores this message.
Otherwise $\mathcal{F}_{\mathrm{COT}}$ looks up the record $((P_R, cid_R), m_R)$ (observe that such a record must exist). If $m_R \notin \{0,1\}$, $\mathcal{F}_{\mathrm{COT}}$ sends a special message $\langle \mathsf{COTFailed}, P_S, P_R, sid\rangle$ to P_R. Otherwise $\mathcal{F}_{\mathrm{COT}}$ sends $\langle \mathsf{CompleteCOT}, \mathsf{ids}, (m_{S,b}, b)\rangle$ to P_R for $b = m_R$.

Note: Additionally, $\mathcal{F}_{\mathrm{COT}}$ screens outs duplicates in commitment identifiers cid for every P_i, and in COT instance identifiers sid for every (P_S, P_R) pair.

Fig. 1. $\mathcal{F}_{\mathrm{COT}}$ ideal functionality

transferred values are the plaintexts of commitments, not full decommitments; and (2) $\mathcal{F}_{\mathrm{COT}}$ does not support opening of the committed values. Nevertheless, $\mathcal{F}_{\mathrm{COT}}$ can ensure that any combination of COT instances is executed on same committed inputs, and thus it can ensure that whenever COT is used as part of *any* security protocol, the parties' inputs into COT are consistent across multiple COT instances.

The COT protocol $\mathcal{P}_{\mathrm{cot}}$ is given in fig. 2. It assumes a common reference string picked by the trusted third party, which defines an instance PK of the CS commitment scheme. The message space for this COT scheme is $[-\frac{n}{2}, \frac{n}{2}]$, the message space of the CS commitment scheme. The commitment, identified as cid, of player P_i on message m is a CS commitment $\mathsf{Com} = \mathsf{CSenc}^{\mathsf{ids}}_{PK}(m)$ with label $\mathsf{ids} = (P_i, cid)$. As we will argue, $\mathcal{P}_{\mathrm{cot}}$ is a secure realization of $\mathcal{F}_{\mathrm{COT}}$; in particular, the receiver either outputs message m_σ committed in $\mathsf{Com}_{S,\sigma}$, or rejects.

The two proof systems used in $\mathcal{P}_{\mathrm{cot}}$ involve conjunctions of Com, DLEQ, and Cot statements. As explained in Section 3.4, such proofs are computationally sound ZK proofs which are concurrently secure in the CRS model. Each takes only a few exponentiations and three communication rounds. Moreover, the messages in both proofs (P_R to P_S and P_S to P_R) can be piggy-backed, with the statements proved by the two players delayed to the last messages, which results in a 4-round protocol. In the random oracle model these proofs are non-interactive and the protocol takes only 2 rounds.

Theorem 3. *Under the DCR assumption, protocol $\mathcal{P}_{\mathrm{cot}}$ is a UC-secure realization of the Committed-OT functionality $\mathcal{F}_{\mathrm{COT}}$ in the CRS model, if the proof systems involved*

Protocol $\mathcal{P}_{\mathrm{cot}}$ for committed oblivious transfer on strings

Common Reference String: CS commitment instance $PK = (n, g, \mathfrak{g}, \mathfrak{h}, \mathfrak{f}, \mathsf{hk})$.

Commit: For player P_i, on commitment instance cid and message m: Player P_i sets $\mathsf{ids} = (P_i, cid)$, $\mathsf{Com} = \mathsf{CSenc}^{\mathsf{ids}}_{PK}(m)$, and broadcasts $\langle \mathsf{ComMsg}, \mathsf{ids}, \mathsf{Com} \rangle$.

Receiver P_R executes a COT instance sid with sender P_S. P_R's bit σ is committed in Com_R, P_S's messages m_0, m_1 are committed in $\mathsf{Com}_{S,0}, \mathsf{Com}_{S,1}$. Let $cid_R, cid_{S,0}, cid_{S,1}$ be the identifiers for these commitments.

COT Step 1: P_R sets $\mathsf{ids} = (P_S, P_R, sid, cid_R, cid_{S,0}, cid_{S,1})$, retrieves $\mathsf{Com}_R = (\tilde{u}, C, \tilde{v})$ and its decommitment $r \in [0, \frac{n}{4}]$. Note that $C = \alpha^\sigma \mathfrak{g}^r$. P_R picks $x \in [0, \frac{n}{4}]$, and computes

$$y = g^x, \quad u = g^r, \quad e = \alpha^\sigma y^r$$

P_R sends $\langle \mathsf{COTMsg1}, \mathsf{ids}, (u, e, y) \rangle$ to P_S, and performs as the prover in the proof system $\mathsf{ZKDLEQ}(g, u, \mathfrak{g}/y, C/e) \wedge \mathsf{ZKCom}(PK, \mathsf{Com}_R, (P_R, cid_R))$ with P_S.

COT Step 2: Upon receiving $\langle \mathsf{COTMsg1}, \mathsf{ids}, (u, e, y) \rangle$ from P_R, P_S retrieves messages m_0, m_1 committed in $\mathsf{Com}_{S_0} = (\tilde{u}_0, C_0, \tilde{v}_0)$ and $\mathsf{Com}_{S_1} = (\tilde{u}_1, C_1, \tilde{v}_1)$. Note that $C_i = \alpha^{m_i} \mathfrak{g}^{r_{m_i}}$ for some r_{m_i}. P_S creates two "COT-encryptions" for $i = 0, 1$:

$$e_i = e^{s_i} \alpha^{m_i - i * s_i} y^{r_i} \quad \text{and} \quad u_i = u^{s_i} g^{r_i}$$

for random *even* values $s_i \in [0, 2n]$ and $r_i \in [0, \frac{n}{2}]$. If P_R passed its proof in Step 1, P_S sends message $\langle \mathsf{COTMsg2}, \mathsf{ids}, (u_0, e_0, u_1, e_1) \rangle$ to P_S, and performs with P_R as the verifier a proof system $\mathsf{ZKCot}(0, e_0, u_0, e, u, y, C_0) \wedge \mathsf{ZKCot}(1, e_1, u_1, e, u, y, C_1) \wedge \mathsf{ZKCom}(\mathsf{Com}_{S,0}, (P_S, cid_{S_0})) \wedge \mathsf{ZKCom}(\mathsf{Com}_{S,1}, (P_S, cid_{S_1}))$.

COT Step 3: P_R decrypts the sCS ciphertext (u_σ, e_σ) and obtains m_σ. If P_S passed its proof in step 2, then P_R outputs m_σ; otherwise P_R rejects.

Note: Either player rejects if the values he receives are *visibly* not in $\mathbb{Z}^*_{n^2}$, *i.e.*, they are outside the $[1, n^2]$ range or are divisible by n.

Fig. 2. Protocol $\mathcal{P}_{\mathrm{cot}}$ for committed OT on strings

are computationally sound and statistically zero-knowledge with straight-line simulators in the CRS model.

Due to lack of space, we present only the crucial aspects of the proof.

Verifiability of inputs. By computational soundness of the proof systems, the players cannot, except with negligible probability, enter different values σ, m_0, m_1 into the OT protocol than those they previously committed. This is easy to see for the cheating receiver P_R. For the cheating sender P_S, by soundness of ZKCot, if P_R accepts, then, with overwhelming probability, for each i there exists a tuple (m_i, r_{m_i}, s_i, r_i) s.t. $(C_i)^2 = \alpha^{2m_i} \mathfrak{g}^{2r_{m_i}}$, $e_i^2 = e^{2s_i} \alpha^{2m_i - i*2s_i} y^{2r_i}$, and $u_i^2 = u^{2s_i} g^{2r_i}$, where $\mathsf{Com}_i = (\tilde{u}_i, C_i, \tilde{v}_i)$ is P_S's commitment whose id is $cid_{S,i}$. In particular, m_i is the message committed in Com_i. Since for honest P_R, $e = \alpha^\sigma y^r$ and $u = g^r$, it follows that for

$i = \sigma$ we have $e_\sigma^2 = \alpha^{2m_\sigma} y^{2r''}$ and $u_\sigma^2 = g^{2r''}$ where $r'' = s_\sigma r + r_\sigma$. Therefore, message m_σ decrypted by P_R from the ciphertext (u_σ, e_σ) is the message committed in Com_σ.

Receiver's and sender's privacy. Receiver's privacy follows from semantic security of CS encryption, while the sender's privacy relies on the fact that if P_R's commitment $\mathsf{Com}_R = (\tilde{u}, C, \tilde{v})$ and the tuple (u, e, y) in P_R's COT message are correctly formed (and they are, except for negligible probability, if P_S accepts P_R's ZKCom and ZKDLEQ proofs, and if the factoring assumption holds), and if σ is a value that satisfies $e^2 = \alpha^{2\sigma} g^{2r}$ for some r (there exists such σ for every $e \in \mathbb{Z}_{n^2}^*$), then the pairs (e_0, u_0) and (e_1, u_1) sent by P_S reveal m_σ, but information-theoretically hide m_i for $i \neq \sigma$. Observe first that if tuples $(\tilde{u}, C, \tilde{v})$ and (u, e, y) are accepted by the verifier (*i.e.*, each element is in $\mathbb{Z}_{n^2}$, but is not a multiple of n), then under the factoring assumption, which is implied by the DCR assumption, all these elements are also in $\mathbb{Z}_{n^2}^*$, except for negligible probability. Second, if P_R passes the ZKCom proof on Com_R and the ZKDLEQ proof on (u, e, y), then except for negligible probability we have $e = \omega_0 \alpha^\sigma g^r$, $u = \omega_1 g^r$, and $y = \omega_2 g^x$ for some (σ, r, x) and some elements $\omega_0, \omega_1, \omega_2$ of order 2 in $\mathbb{Z}_{n^2}^*$. Therefore, values (u_i, e_i) sent by P_S are equal to $e_i = \alpha^{m_i + s_i(\sigma - i)} y^{s_i r + r_i}$ and $u_i = g^{s_i r + r_i}$, because s_i is even. Note that for any σ, $gcd(\sigma - i, n) = 1$ for either $i = 0$ or $i = 1$ (or for both). Since the order of α is n, and $(s_i \bmod n)$ is distributed uniformly in $\mathbb{Z}_n$, value $\alpha^{m_i + s_i(\sigma - i)}$ is distributed uniformly in the subgroup generated by α in $\mathbb{Z}_{n^2}^*$. Because (1) the orders of g and y are both divisors of $2n'$, (2) $s_i r + r_i$ is even, and (3) $(r_i \bmod n')$ is distributed statistically close to uniform over $\mathbb{Z}_{n'}$, it follows that pair $(g^{s_i r + r_i}, y^{s_i r + r_i})$ is distributed statistically close to $(g^{2r'}, y^{2r'})$ for r' uniform in $\mathbb{Z}_{n'}$. Taken together, it follows that pair (e_i, u_i), for $i \neq \sigma$, is distributed statistically close to $(\alpha^{m'} y^{2r'}, g^{2r'})$ for random $(m', r') \in (\mathbb{Z}_n \times \mathbb{Z}_{n'})$, and thus it is statistically independent of m_i.

Construction of the straight-line simulator. The proof that protocol $\mathcal{P}_{\text{cot}}$ UC-realizes the COT functionality $\mathcal{F}_{\text{COT}}$ involves construction of a straight-line simulator, which pretends to follow the protocol on behalf of the uncorrupted parties by executing it on some fixed values unrelated to the real inputs of these parties, and simulates their proof systems using their straight-line simulators. Moreover, the simulator straight-line extracts the effective inputs contributed by the corrupted players by choosing the Camenisch-Shoup public key PK embedded in the CRS and decrypting these players' inputs from their commitments. The simulator submits these extracted inputs to the ideal functionality if the corrupted players pass the associated ZK proofs. CCA security of Camenisch-Shoup encryption implies that the ciphertexts contained in the commitments and COT messages created by the simulator remain indistinguishable from the corresponding ciphertexts created in the real protocol, even if the simulator accesses the decryption oracle (to extract the values committed by the corrupt players). Finally, the proof systems performed by the corrupted players are sound even if the simulator picks the CRS because as long as the adversary passes its proofs only on correct statements, the simulation is distributed statistically close to the real execution. Hence, by the standard soundness of the proof systems involved, the adversary has only negligible probability of passing some proof on an incorrect statement in the simulation.

5 UC-Secure Two-Party Computation on Committed Inputs

We present an efficient version of Yao's "garbled circuits" protocol for secure two-party computation (2PC). The protocol operates on committed inputs and is universally composable (in the CRS model). In addition to any two-party secure computation in the malicious model, our protocol can be used, for example, to ensure that multiple instances of secure computation are executed on consistent inputs.

The ideal functionality $\mathcal{F}_{\mathrm{2PC}}$ for secure two-party computation on committed inputs in shown in fig. 3. Abstracting from the bookkeeping details, $\mathcal{F}_{\mathrm{2PC}}$ is a simple generalization of the standard secure computation functionality where two players send their respective inputs x and y to the trusted third party $\mathcal{F}$, who returns the result of evaluating some circuit $C(x, y)$ to one or both players.

The *committed 2PC* functionality $\mathcal{F}_{\mathrm{2PC}}$ accepts any number of commitments from parties $P_1, \ldots, P_n$, which are intended to represent the commitments to the bits encoding these parties' inputs into some two-party computation protocols. For every commitment, $\mathcal{F}_{\mathrm{2PC}}$ records the committed bit. If some party P_R requests secure computation of some circuit C with another party P_S, the request specifies C and a vector of commitments to P_R's and P_S's inputs into this circuit. If party P_S accedes to this request, $\mathcal{F}_{\mathrm{2PC}}$ sends to P_R the output of circuit C computed on the inputs committed in the specified commitments. Note that our $\mathcal{F}_{\mathrm{2PC}}$ sends the output only to P_R, but since this is a *committed* 2PC functionality, the players can simply reverse the roles and request that the same C be computed on the same vector of commitments, in order to enable P_S to receive the output. (Our actual 2PC protocol allows P_S to receive the output with no computational overhead and one extra communication round.)

We assume that the circuit $\mathcal{C}$ consists of binary two-input gates $G = \{g_1, \ldots, g_c\}$ with unbounded fan-out but no cycles, connected by wires $W = \{w_1, \ldots, w_m\}$. Some subset W_S of n_s input wires are designated as P_S's inputs, and n_r input wires form the set W_R of P_R's inputs. Some subset W_O of the output wires is designed as outputs for P_R. (Optionally, some output wires can also be designated as outputs for P_S.)

The Committed 2PC protocol is in fig. 4. It is similar to the COT protocol of Section 4, and uses the same commitments and same message pattern, requiring 4 rounds in CRS and 2 rounds in ROM. In the first message, the receiver uses the proof systems of the $\mathcal{P}_{\mathrm{cot}}$ protocol and an additional proof system $\mathsf{ZKBit}(C) = (\mathsf{ZKDL}(\mathfrak{g}, C) \vee \mathsf{ZKDL}(\mathfrak{g}, C/\alpha))$ for proving that the CS commitment $\mathsf{Com} = (u, C, v)$ or the sCS commitment C are commitments to a bit. In the second message, the sender creates the garbled circuit and uses the $\mathsf{CorrectYao}$ proof system to prove that it has been formed correctly. This step encompasses the entire Yao's construction and is discussed below. In the following, we denote sender P_S as S and receiver P_R as R.

Wire keys and commitments: S picks two random (symmetric) sCS private keys x_0^w, x_1^w for every wire $w \in W$, and for each x_i^w computes an sCS commitment C_i^w to x_i^w. Also, S makes a set of wire keys corresponding to his inputs, $\{x_{b_w}^w\}_{w \in W_S}$, where b_w is S's input bit on $w \in W_S$.

COTs on receiver's wire keys: S completes n_r instances of the COT protocol on the wire keys corresponding to receiver's wires: for each $i = 1, .., n_r$, S enters keys $(x_0^{w_i}, x_1^{w_i})$ as a sender in the COT protocol, where w_i designates the receiver's i^{th} input

Ideal functionality $\mathcal{F}_{\mathrm{2PC}}$ for two-party secure computation on committed inputs

Commit: Upon receiving a $\langle \mathsf{ComMsg}, (P_i, cid), m \rangle$ message from P_i, $\mathcal{F}_{\mathrm{2PC}}$ verifies that this cid has not been used by P_i before, records the $((P_i, cid), m)$ pair and broadcasts a $\langle \mathsf{Committed}, (P_i, cid) \rangle$ message. Message m is either a message in the prescribed message space, or a special symbol $\perp$.

Start2PC: Upon receiving

$$msg = \langle \mathsf{Start2PC}, (P_S, P_R, sid, cid_{S1}, \ldots, cid_{Sn_s}, cid_{R1}, \ldots, cid_{Rn_r}, C) \rangle$$

from P_R, $\mathcal{F}_{\mathrm{2PC}}$ verifies that (i) this sid has not been used by P_S and P_R before; (ii) for every index k such that $1 \le k \le n_s$, $\mathcal{F}_{\mathrm{2PC}}$ has a unique record $((P_S, cid_{Sk}), m_{Sk})$ (these commitments correspond to P_S's inputs into the protocol); (iii) for every index l such that $1 \le l \le n_r$, $\mathcal{F}_{\mathrm{2PC}}$ has a unique record $((P_R, cid_{Rl}), m_{Rl})$ and that $m_{Rl} \in \{0, 1\}$ (these commitments correspond to P_R's inputs into the protocol), and (iv) C is a description of a circuit that takes $n_s + n_r$ bits as inputs. If this fails, $\mathcal{F}_{\mathrm{2PC}}$ ignores this message; otherwise, it records msg and forwards it to P_S.

Complete2PC: Upon receiving

$$msg = \langle \mathsf{Complete2PC}, (P_S, P_R, sid, cid_{S1}, \ldots, cid_{Sn_s}, cid_{R1}, \ldots, cid_{Rn_r}, C) \rangle$$

from P_S, $\mathcal{F}_{\mathrm{2PC}}$ verifies that it has a record $\langle \mathsf{Start2PC}, \mathsf{ids} \rangle$, where $\mathsf{ids} = (P_S, P_R, sid, cid_{S1}, \ldots, cid_{R1}, \ldots, C)$. If not, $\mathcal{F}_{\mathrm{2PC}}$ ignores this message.
$\mathcal{F}_{\mathrm{2PC}}$ looks up the records $((P_S, cid_{S1}), m_{S1}), \ldots, ((P_S, cid_{Sn_s}), m_{Sn_s})$ and $((P_R, cid_{R1}), m_{R1}), \ldots, ((P_R, cid_{Rn_r}), m_{Rn_r})$. If $m_{Sk} \notin \{0, 1\}$ for some index k, $\mathcal{F}_{\mathrm{2PC}}$ ignores this instance of the 2PC protocol.
Otherwise, $\mathcal{F}_{\mathrm{2PC}}$ evaluates circuit C on inputs $m_{S1}, \ldots, m_{Sn_s}, m_{R1}, \ldots, m_{rn_r}$. $\mathcal{F}_{\mathrm{2PC}}$ sends $\langle \mathsf{Complete2PC}, \mathsf{ids}, \boldsymbol{b}) \rangle$ to P_R, where $\boldsymbol{b}$ is the output of the circuit.

Note: This is a functionality for *one-directional* two-party computation, where only the receiver P_R learns the output. Because both parties are committed to their inputs, they can run another instance of the same protocol with the roles of P_S and P_R reversed.

Fig. 3. $\mathcal{F}_{\mathrm{2PC}}$ ideal functionality

wire. This way, for every $w \in W_r$, the receiver obtains the wire key $x^w_{b_w}$ where b_w is his input bit on wire w. Technically, S computes tuple $({u_0}^{(w_i)}, {e_0}^{(w_i)}, {u_1}^{(w_i)}, {e_1}^{(w_i)})$ by following the sender's algorithm in Step 2 of $\mathcal{P}_{\mathrm{cot}}$ on tuple $(u^{(i)}, e^{(i)}, y^{(i)})$ and a pair of messages $({x_0}^{w_i}, {x_1}^{w_i})$, and their corresponding sCS commitments $({C_0}^{w_i}, {C_1}^{w_i})$.

Receiver's output wires: For every receiver's output wire $w \in W_0$, S creates a pair of ciphertexts E^w_0, E^w_1 that enables R to interpret the corresponding wire keys. Namely, $E^w_0 = \mathsf{sCSenc}_{x^w_0}(0)$ and $E^w_1 = \mathsf{sCSenc}_{x^w_1}(1)$.

Forming the garbled truth tables: The following process is repeated for every gate $g \in G$. Let A and B be the input wires of g, and C the output wire. Let $C^A_{0,1}, C^B_{0,1}, C^C_{0,1}$ be the six sCS commitments to the respective wire keys (two per wire). These commitments form the truth table for the gate g in which the input bits b_A, b_B and the output bit $b_C = g(b_A, b_B)$ are replaced by commitments to the corresponding wire keys. As in the

Committed 2PC Protocol

Common Reference String: CS commitment instance $PK = (n, g, \mathfrak{g}, \mathfrak{h}, \mathfrak{f}, \mathsf{hk})$.

Commit: As in $\mathcal{P}_{\text{cot}}$ of fig. 2, player P_i on commitment instance cid and message m broadcasts $\langle \mathsf{ComMsg}, \mathsf{ids}, \mathsf{Com}\rangle$ where $\mathsf{Com} = \mathsf{CSenc}^{\mathsf{ids}}_{PK}(m)$ for $\mathsf{ids} = (P_i, cid)$.

2PC Step 1: To trigger instance sid of the protocol in order to compute circuit C on commitment instances $cid_{S1}, \ldots, cid_{Sn_s}$ made by P_S and commitments $cid_{R1}, \ldots, cid_{Rn_r}$ made by P_R, the *receiver* P_R prepares n_r messages, each computed as in Step 1 of $\mathcal{P}_{\text{cot}}$ (fig. 2): for each $i = 1, .., n_r$, P_S computes a tuple $(y^{(i)}, u^{(i)}, e^{(i)})$ on bit σ_i committed in $\mathsf{Com}_{cid_{Ri}} = (\tilde{u}^{(i)}, C^{(i)}, \tilde{v}^{(i)})$ and its decommitment $r^{(i)}$. P_S sends to P_R message

$$\langle \mathsf{Start2PC}, \mathsf{ids}, C, \{y^{(i)}, u^{(i)}, e^{(i)}\}_{i=1..n_r}\rangle$$

where ids is the above vector of commitment ids. P_R then performs the ZK proof system $\mathsf{ZKR}_{\mathsf{2PC}}$, which is a conjunction of n_r instances of the $\mathsf{ZKDLEQ}(\ldots) \wedge \mathsf{ZKCom}(\ldots)$ proof system used in Step 1 of $\mathcal{P}_{\text{cot}}$, one per each tuple $(r^i, C^{(i)}, y^{(i)}, u^{(i)}, e^{(i)})$, and n_r instances of the $\mathsf{ZKBit}(C^{(i)})$ proof.

2PC Step 2: On receiving the $\langle \mathsf{Start2PC}, \mathsf{ids}, C, ...\rangle$ message and verifying the ZK proofs, P_S retrieves its commitments $\mathsf{Com}_{cid_{S1}}, ..., \mathsf{Com}_{cid_{Sn_s}}$ specified in the ids string, and sends to P_R a garbled version of circuit C computed on these inputs:

$$\mathsf{Complete2PC}\langle\ \mathsf{ids}, \{C^w_b\}_{b\in\{0,1\},\ w\in W}, \{E^g_{\alpha\beta}\}_{\alpha\beta\in\{00,01,10,11\},\ g\in G},$$
$$\{x^w_{b_w}\}_{w\in W_S}, \{E^w_0, E^w_1\}_{w\in W_0}, \{u^{(w)}_0, e^{(w)}_0, u^{(w)}_1, e^{(w)}_1\}_{w\in W_R}\ \rangle$$

These values are defined in Section 5. P_S also performs the ZK proof $\mathsf{CorrectYao}$.

2PC Step 3: P_R verifies the ZK proof $\mathsf{CorrectYao}$, evaluates the garbled circuit and outputs its result. (Optionally, P_R can send back to P_S the wire keys corresponding to P_S's output wires.)

Fig. 4. Committed 2PC Protocol

original Yao's protocol, S creates a ciphertext for each row of the truth table, encrypting the output-wire key corresponding to this row's output bit under the two input-wire keys corresponding to this row's input bits. The ciphertexts must be randomly shuffled to prevent R from learning which row $(b_A, b_B, g(b_A, b_B))$ of the truth table he succeeds in decrypting. S picks two random bits, σ_A and σ_B, which determine, intuitively, if the values corresponding to the A and B wires are "switched" or not. (If w is S's input wire, than σ_w is equal to S's input bit on that wire.) If the rows are denoted in binary as $00, 01, 10, 11$, then the first ciphertext received by R corresponds to row $\sigma_A\sigma_B$, the second to row $\bar{\sigma}_A\sigma_B$, the third to row $\sigma_A\bar{\sigma}_B$, and the fourth to row $\bar{\sigma}_A\bar{\sigma}_B$.

S creates the ciphertext list (E_{00}, E_{01}, E_{10}, and E_{11}) using a two-key encryption scheme $E_{\alpha\beta} = \mathsf{2KEnc}_{x_1, x_2}(x)$, where for each α, β, $x_1 = x^A_{\alpha\oplus\sigma_A}$, $x_2 = x^B_{\beta\oplus\sigma_B}$, and $x = x^C_{g(\alpha\oplus\sigma_A, \beta\oplus\sigma_B)}$. For example, if $\sigma_A = \sigma_B = 0$, then each $E_{\alpha\beta}$ is a two-key encryption under keys x^A_α and x^B_β of the output-wire key $x^C_{g(\alpha,\beta)}$. If $\sigma_A = 1, \sigma_B = 0$, then each $E_{\alpha\beta}$ is a two-key encryption under keys $x^A_{\overline{\alpha}}$ and x^B_β of key $x^C_{g(\overline{\alpha},\beta)}$, and so on. Note that tuple $(\sigma_A, \sigma_B, \alpha, \beta)$ uniquely defines the commitments C_1, C_2, C that correspond to the above keys x_1, x_2, x: $C_1 = C^A_{\alpha\oplus\sigma_A}$, $C_2 = C^B_{\beta\oplus\sigma_B}$, and $C = C^C_{g(\alpha\oplus\sigma_A, \beta\oplus\sigma_B)}$.

The two-key encryption $\mathsf{2KEnc}_{x_1,x_2}(x)$ is created as follows. The key $x \in [0, 2^{k''}]$ is split in two parts, x'_1 and x'_2, by choosing x'_1 at random in $[-2^{k''+k}, 2^{k''+k}]$ (recall that k'', k are security parameters, where $k'' = \frac{|n|}{2}$ and k can be 80), and setting $x'_2 = x - x'_1$ (over integers). S also computes an sCS commitment D to x'_1. Observe that if C is an sCS commitment to x, then C/D is an sCS commitment to x'_2. The ciphertext E is a triple $\langle D, F^{(1)}, F^{(2)}\rangle$, where $F^{(i)} = \mathsf{sCSenc}_{x_i}(x'_i)$. Let $E_{\alpha\beta}$ denote $\langle D_{\alpha\beta}, F^{(1)}_{\alpha\beta}, F^{(2)}_{\alpha\beta}\rangle$.

Proving circuit correctness: CorrectYao is a (concurrent ZK, with straight-line simulator) proof system formed by *conjunction* of the following proof systems:

$$\begin{aligned}\textstyle\bigwedge_{g\in G} \mathsf{CorrectGarble}_g \wedge \bigwedge_{w\in W} \mathsf{GoodKeys}_w &\wedge \textstyle\bigwedge_{w\in W_S} \mathsf{CorrectInput}_w \\ \wedge \textstyle\bigwedge_{w\in W_R} \mathsf{ZKS}_w &\wedge \textstyle\bigwedge_{w\in W_O} \mathsf{CorrectOutput}_w\end{aligned}$$

where

$$\begin{aligned}\mathsf{GoodKeys}_w &= \mathsf{ZKNotEq}(C^w_0, C^w_1)\\ \mathsf{CorrectInput}_w &= (\mathsf{ZKDL}(\mathfrak{g}, C^w_0/\alpha^{x^w_{b_w}}) \wedge \mathsf{ZKDL}(\mathfrak{g}, C_b)) \vee \\ &\quad\ (\mathsf{ZKDL}(\mathfrak{g}, C^w_1/\alpha^{x^w_{b_w}}) \wedge \mathsf{ZKDL}(\mathfrak{g}, C_b/\alpha)), \text{ where } C_b \text{ is the}\\ &\quad\ \text{sCS commitment inside } \mathsf{Com}_{cid_{Si}} \text{ if } w \text{ is the } i^{th} \text{ input wire of} S\\ \mathsf{CorrectOutput}_w &= \mathsf{ZKPlainEq2}(E^w_0, C^w_0, 0) \ \wedge\ \mathsf{ZKPlainEq2}(E^w_1, C^w_1, 1)\end{aligned}$$

Here ZKS_w refers to the proof performed by the sender in the instance of the COT protocol that corresponds to receiver's wire $w \in W_R$. $\mathsf{ZKPlainEq2}(E, C_k, m)$ is the proof system for showing that E is an sCS encryption of plaintext m under key k committed in C_k, and is a trivial simplification of the $\mathsf{ZKPlainEq}(E, C_k, C_m)$ proof system for proving the same about commitment C_m to m. Finally, $\mathsf{CorrectGarble}_g$ proves that the ciphertext table $E_{00}, E_{01}, E_{10}, E_{11}$ corresponding to garbled gate g is formed correctly, where $E_{\alpha\beta} = (D_{\alpha\beta}, F^{(1)}_{\alpha\beta}, F^{(2)}_{\alpha\beta})$:

$$\begin{aligned}\mathsf{CorrectGarble}_g = \mathsf{CorrectShuffle}(0,0) &\vee \mathsf{CorrectShuffle}(0,1) \vee\\ \mathsf{CorrectShuffle}(1,0) &\vee \mathsf{CorrectShuffle}(1,1)\end{aligned}$$

$$\begin{aligned}\mathsf{CorrectShuffle}(\alpha,\beta) = \mathsf{CorrectCipher}(0,0,\alpha,\beta) &\wedge \mathsf{CorrectCipher}(0,1,\alpha,\beta) \ \wedge\\ \mathsf{CorrectCipher}(1,0,\alpha,\beta) &\wedge \mathsf{CorrectCipher}(1,1,\alpha,\beta)\end{aligned}$$

$$\begin{aligned}\mathsf{CorrectCipher}(\sigma_A, \sigma_B, \alpha, \beta) = &\ \mathsf{ZKPlainEq}(F^{(1)}_{\alpha\beta}, C^A_{\alpha\oplus\sigma_A}, D_{\alpha\beta}) \ \wedge\\ &\ \mathsf{ZKPlainEq}(F^{(2)}_{\alpha\beta}, C^B_{\beta\oplus\sigma_B}, (C^C_{g(\alpha\oplus\sigma_A, \beta\oplus\sigma_B)}/D_{\alpha\beta}))\end{aligned}$$

Circuit evaluation: R obtains his input-wire keys via COT and evaluates the entire circuit gate by gate. Unambiguity of sCS encryption and soundness of the proof systems ensures that for each gate, R decrypts exactly one of the four ciphertexts forming that gate's garbled truth table and obtains the key corresponding to the gate's output wire.

Theorem 4. *Under the strong RSA and DCR assumptions, the 2PC protocol of fig. 4 is a UC-secure realization of the Committed 2PC functionality $\mathcal{F}_{\mathrm{2PC}}$ in the CRS model.*

References

[AIR01] W. Aiello, Y. Ishai, and O. Reingold. Priced oblivious transfer: How to sell digital goods. In *Proc. EUROCRYPT*, pages 119–135, 2001.

[Bou00] F. Boudot. Efficient proofs that a committed number lies in an interval. In *Proc. EUROCRYPT*, pages 431–444, 2000.

[CC00] J. Camenisch and C. Cachin. Optimistic fair secure computation. In *Proc. CRYPTO*, pages 93–111, 2000.

[CD97] R. Cramer and I. Damgård. Linear zero-knowledge – a note on efficient zero-knowledge proofs and arguments. In *Proc. STOC*, pages 436–445, 1997.

[CDS94] R. Cramer, I. Damgård, and B. Schoenmakers. Proofs of partial knowledge and simplified design of witness hiding protocols. In *Proc. CRYPTO*, pages 174–187, 1994.

[CGHG01] D. Catalano, R. Gennaro, and N. Howgrave-Graham. The bit security of Paillier's encryption scheme and its applications. In *Proc. EUROCRYPT*, pages 229–243, 2001.

[CGHGN01] D. Catalano, R. Gennaro, N. Howgrave-Graham, and P. Nguyen. Paillier's cryptosystem revisited. In *Proc. CCS*, pages 206–214, 2001.

[CM99] J. Camenisch and M. Michels. Proving in zero-knowledge that a number is a product of two safe primes. In *Proc. EUROCRYPT*, pages 107–122, 1999.

[Cré89] C. Crépeau. Verifiable disclosure of secrets and applications. In *Proc. EUROCRYPT*, pages 181–191, 1989.

[CS03] J. Camenisch and V. Shoup. Practical verifiable encryption and decryption of discrete logarithms. In *Proc. CRYPTO*, pages 126–144, 2003.

[CvdGT95] C. Crépeau, J. van de Graaf, and A. Tapp. Committed oblivious transfer and private multiparty computation. In *Proc. CRYPTO*, pages 110–123, 1995.

[Dam02] I. Damgård. Efficient concurrent zero-knowledge in the auxiliary string model. In *Proc. EUROCRYPT*, pages 418–430, 2002.

[DF02] I. Damgård and E. Fujisaki. A statistically hiding integer commitment scheme based on groups with hidden order. In *Proc. ASIACRYPT*, pages 125–142, 2002.

[DI05] I. Damgård and Y. Ishai. Constant-round multiparty computation using a black-box pseudorandom generator. In *Proc. CRYPTO*, pages 378–394, 2005.

[FO97] E. Fujisaki and T. Okamoto. Statistical zero knowledge protocols to prove modular polynomial relations. In *Proc. CRYPTO*, pages 16–30, 1997.

[GMW87] O. Goldreich, S. Micali, and A. Wigderson. How to play any mental game. In *Proc. STOC*, pages 218–229. ACM, 1987.

[GMY04] J. Garay, P. MacKenzie, and K. Yang. Efficient and universally composable oblivious transfer and applications. In *Proc. TCC*, pages 297–316, 2004.

[HSS93] J. Håstad, A. Schrift, and A. Shamir. The discrete logarithm modulo a composite hides $o(n)$ bits. *J. Comput. Syst. Sci.*, 47:850–864, 1993.

[JJ00] M. Jakobsson and A. Juels. Mix and match: Secure function evaluation via ciphertexts. In *Proc. ASIACRYPT*, pages 162–177, 2000.

[Kil88] J. Kilian. Founding cryptography on oblivious transfer. In *Proc. STOC*, pages 20–31, 1988.

[KO04] J. Katz and R. Ostrovsky. Rount-optimal secure two-party computation. In *Proc. CRYPTO*, pages 335–354, 2004.

[Lin03] Y. Lindell. Parallel coin-tossing and constant-round secure two-party computation. *J. Cryptology*, 16(3):143–184, 2003.

[Lip03] H. Lipmaa. Verifiable homomorphic oblivious transfer and private equality test. In *Proc. ASIACRYPT*, pages 416–433, 2003.

[LP07] Y. Lindell and B. Pinkas. An efficient protocol for secure two-party computation in the presence of malicious adversaries. In *Proc. EUROCRYPT*, 2007.

[MF06] P. Mohassel and M. Franklin. Efficiency tradeoffs for malicious two-party computation. In *Proc. PKC*, pages 458–473, 2006.

[Pai99] P. Paillier. Public-key cryptosystems based on composite degree residuosity classes. In *Proc. EUROCRYPT*, pages 223–238, 1999.

[Ped91] T. P. Pedersen. Non-interactive and information-theoretic secure verifiable secret sharing. In *Proc. CRYPTO*, pages 129–140, 1991.

[Pin03] B. Pinkas. Fair secure two-party computation. In *Proc. EUROCRYPT*, pages 87–105, 2003.

[Woo07] D. Woodruff. Revisiting the efficiency of malicious two-party computation. In *Proc. EUROCRYPT*, 2007.

[Yao86] A. Yao. How to generate and exchange secrets. In *Proc. FOCS*, pages 162–167, 1986.

A HVZK Proof System for Statement PlainEq

This is an HVZK proof system for language $\mathsf{PlainEq} = \{((e,u), C_x, C_m) \mid$ there exist x, m, r_x, r_m s.t. $e = \alpha^m u^x$, $C_x = \alpha^x \mathfrak{g}^{r_x}$, and $C_m = \alpha^m \mathfrak{g}^{r_m}$ $\}$, *i.e.*, for the language of tuples $((e,u), C_x, C_m)$ s.t. (e,u) is an sCS encryption of the plaintext m committed in sCS commitment C_m under the key x committed in sCS commitment C_x. It is special HVZK with weak special soundness under the strong RSA assumption. All the parameters are as in section 3.4, except for two additional elements G, H which are assumed to be random in $\mathbb{Z}^*_{n^2}$ and can be included in the CRS.

1. The private inputs of the prover are
$$m \in [-2^{k''+k}, 2^{k''+k}], \quad x \in [0, 2^{k''}], \quad r_m, r_x \in [0, \frac{n}{4}]$$
2. The prover picks $t_x \in [0, \frac{n}{4}]$ and sends $T_x = G^x H^{t_x}$ to the verifier. He also picks
$$m', r'_m, x', r'_x, t'_x \in [0, 2^{k+k'+2k''}]$$
and sends the following commitments to the verifier:
$$e' = \alpha^{2m'} u^{2x'}, \;\; C'_x = \alpha^{2x'} \mathfrak{g}^{2r'_x}, \;\; C'_m = \alpha^{2m'} \mathfrak{g}^{2r'_m}, \;\; T'_x = G^{x'} H^{t'_x}$$
3. Verifier responds with a random challenge $c \in \{0,1\}^k$
4. Prover sends the following responses, all computed over integers:
$$\tilde{m} = m' - cm, \; \tilde{r}_m = r'_m - cr_m, \; \tilde{x} = x' - cx, \; \tilde{r}_x = r'_x - cr_x, \; \tilde{t}_x = t'_x - ct_x$$
5. Verifies accepts if $\tilde{x} \in [-\frac{n}{4}, \frac{n}{4}]$ and if the following equations hold:
$$e' = e^{2c} \alpha^{2\tilde{m}} u^{2\tilde{x}},$$
$$C'_m = (C_m)^{2c} \alpha^{2\tilde{m}} \mathfrak{g}^{2\tilde{r}_m}, \;\; C'_x = (C_x)^{2c} \alpha^{2\tilde{x}} \mathfrak{g}^{2\tilde{r}_x},$$
$$T'_x = (T_x)^c G^{\tilde{x}} H^{\tilde{t}_x}$$