Efficient and Universally Composable Committed Oblivious Transfer and Applications*

JUAN A. GARAY† PHILIP MACKENZIE† KE YANG‡

Abstract

Committed Oblivious Transfer (COT) is a useful cryptographic primitive that combines the functionalities of bit commitment and oblivious transfer. In this paper, we introduce an extended version of COT (ECOT) which additionally allows proofs of relations among committed bits, and we construct an efficient protocol that securely realizes an ECOT functionality in the universal-composability (UC) framework in the common reference string (CRS) model. Our construction is more efficient than previous (non-UC) constructions of COT, involving only a constant number of exponentiations and communication rounds. Using the ECOT functionality as a building block, we construct efficient UC protocols for general two-party and multi-party functionalities (in the CRS model), each gate requiring a constant number of ECOT's.

1 Introduction

Committed Oblivious Transfer (COT) was introduced by Crépeau [18] (under the name "Verifiable Oblivious Transfer") as a natural combination of $\binom{2}{1}$-Oblivious Transfer [22] and Bit Commitment. At the start of the computation Alice is committed to bits a_0 and a_1 and Bob is committed to bit b; at the end Bob is committed to a_b and knows nothing about $a_{\bar{b}}$, while Alice learns nothing about b. One can see that this allows each party engaged in an oblivious transfer to be certain that the other party is performing the oblivious transfer operation on their declared inputs.[1] This has been shown to be useful in [19], who construct a protocol for general secure multi-party computation in the model of [28] using COT.

In this paper we show how to improve on previous constructions of COT in the areas of efficiency and universal composability. In terms of efficiency, the protocol we construct for COT uses only a constant number of exponentiations and communication rounds per transfer.[2] In contrast, the most efficient previously known construction of COT [19] uses $O(k)$ invocations of OT (thus implying at least the same number of public-key operations using known constructions) and bit commitments, and $O(k)$ rounds, for k a security parameter. Furthermore, we show that our protocol securely realizes an ideal COT functionality in the recently-proposed *universal composability* (UC) framework by Canetti [10], in the *common reference string* (CRS) model. Recall that to define security in this framework, one first specifies an "ideal functionality" describing the desired behavior of the protocol using a trusted party,

*An extended abstract of this paper appeared in the *Proc. 1st Theory of Cryptography Conference*, Boston, MA, February 2004.

†Bell Labs, Lucent Technologies, 600 Mountain Ave, Murray Hill, NJ 07974. {garay,philmac}@research.bell-labs.com.

‡Computer Science Department, Carnegie Mellon University, 5000 Forbes Ave., Pittsburgh, PA 15213. yangke@cs.cmu.edu.

[1]This contrasts with standard oblivious transfer, where some other method (perhaps another cryptographic building block, or verification at some higher layer protocol) is required to guarantee that parties are performing their part of the transfer on their declared inputs.

[2]Security is proved under some standard number theoretic assumptions, discussed later.

and then one proves that a particular protocol operating in the real world "securely realizes" this ideal functionality, by showing that no "environment" would be able to distinguish (1) an adversary operating in the real world with parties running this protocol from (2) an "ideal adversary" operating in an ideal process consisting of dummy parties that simply call the ideal functionality. A main virtue of this framework is that the security of protocols thus defined is preserved under a general composition operation called "universal composition," which essentially means that protocols remain secure even when composed with other protocols that may be running concurrently in the same system. We give a more detailed review of the UC framework later in the paper. We note that a similar framework was independently proposed by Pfitzmann and Waidner [37, 38]. Intuitively, these two frameworks are similar, although there are a number of technical differences. We choose to use the UC framework in this paper.[3]

Our protocol actually realizes an enhanced COT functionality, which we call $ECOT$, where in addition to oblivious transfer, one can prove certain relations among committed bits (in particular, among three bits). To demonstrate the usefulness of this functionality, we show that using ECOT as a building block, any well-formed two-party and multi-party functionality can be securely realized *efficiently* in the universal composability framework. Plugging in our protocol for realizing the ECOT into this construction, we have an efficient protocol for any well-formed two-party and multi-party functionality in the CRS model.

Canetti *et al.* [12] were the first to show that such functionalities are indeed realizable in this model, even under general cryptographic assumptions and regardless of the number of corrupted parties. More specifically, [12] follows the general "two-phase" approach of [27] of first designing a solution for the case of honest-but-curious parties, and then turning it into a solution for the actively malicious adversary, using a "compiler." The compiler adds a zero-knowledge proof to every message, proving that it is consistent with the history and the (committed) private input and the randomness. Notice that since the "consistency" proofs are for relations involving the execution of Turing machines, they are quite complex and it is unlikely that they admit efficient protocols; rather, proofs for general NP statements are used (which involve a reduction to an NP-complete problem like Hamiltonian Cycle), making the compiler a major source of inefficiency. Canetti *et al.* make the protocol in [27] secure in the UC framework by replacing the basic primitives (namely, oblivious transfer, bit commitment, and zero-knowledge) with their universally composable counterparts. The resultant protocol becomes universally composable, but remains rather inefficient. In this paper we follow a different approach. By incorporating stronger security into the basic building block (i.e., ECOT), we are able to build protocols secure against adaptive and malicious adversaries *directly*, eliminating the need for a (normally inefficient) compiler. In this way, we are able to construct protocols that are efficient and at the same time enjoy a high level of security.

Our results. We now present a more detailed account of our results. We start by defining an ECOT functionality ($\mathcal{F}_{\mathrm{ECOT}}$), which, as mentioned above, additionally allows the sender to prove relations on three committed bits to the receiver. Then we construct a protocol to realize the ECOT functionality. The starting point for our construction is the standard Pedersen commitment scheme [35]. Then we build an OT protocol over these commitments that is loosely based on the (non-concurrent version of the) OT protocol of Garay and MacKenzie [24] (which in turn is based on the OT protocol of Bellare and Micali [4]). Zero-knowledge (ZK) proofs are required in this OT protocol, and thus we work in a hybrid model with ideal ZK functionalities. Naturally, the constructions for proving relations on three

[3]The ideal-process/real-world formulation of security and the simulator-based paradigm were initiated by Goldreich *et al.* [27]. From then on, there have been many definitions in this (now standard) paradigm, with emphasis on different aspects. For a number of examples, see Goldwasser and Levin [30], Micali and Rogaway [32], Beaver [2, 3], and Canetti [9], for the formulations that precede the UC framework.

committed bits also use these ideal ZK functionalities. Finally, to construct efficient protocols that securely realize these ZK functionalities, we construct a special type of honest-verifier ZK protocol for each desired relations, and then we use a result by Garay *et al.* [25] that shows how to convert this special type of honest-verifier ZK protocol into a universally-composable ZK protocol. These results are presented in Section 3.

The ECOT functionality can be naturally extended into one that performs $\binom{4}{1}$-transfers (instead of $\binom{2}{1}$) and proves relations on four committed bits (as opposed to three). We call this extended functionality $\mathcal{F}^4_{\mathrm{ECOT}}$, and show how to construct it using the original $\mathcal{F}_{\mathrm{ECOT}}$ functionality as a building block. Equipped with $\mathcal{F}^4_{\mathrm{ECOT}}$, we then show how to securely realize a two-party functionality that we call *Joint Gate Evaluation* ($\mathcal{F}_{\mathrm{JGE}}$), which, as its name indicates, allows two parties to securely compute any Boolean function over two bits shared between them. Essentially, the protocol realizing this functionality uses a construction similar to that of [27] for the computation of the multiplication gate. However, distinctive features of the protocol are that it deals directly with adaptively malicious parties, and its efficiency: only a constant number of exponentiations and communication rounds per gate evaluation. Joint Gate Evaluation is presented in Section 4.

Finally, we use $\mathcal{F}_{\mathrm{JGE}}$ to securely realize — efficiently — any adaptively well-formed two-party and multi-party functionality, which is expressed by a boolean circuit, in a universally-composable way. Again, since the realization is directly for the actively malicious adversary, and by means of an efficient building block, the overall computational complexity is a small constant times the number of gates in the representation of the functionality, and the number of rounds is a constant times the depth of the circuit. The treatment of two-party functionalities is presented in Section 5, while the case of multi-party functionalities, with the one-to-many extensions and realizations of the required building blocks, is discussed in Section 6. Putting everything together, we construct efficient and universally composable two-party and multi-party computation protocols that are secure against adaptive adversaries in the *erasing* model, where we allow parties to erase certain information reliably.

As a technical note, we use the gate-by-gate approach from [27], and make sure that each gate is computed efficiently. We do not use the "encrypted circuit" approach due to Yao [40], which yields constant-round protocols but is rather inefficient in terms of communication complexity, since one needs to prove in zero-knowledge that the encrypted circuit is correct and these proofs are unlikely to admit efficient protocols.

Related work. We already mentioned prior work on COT [18, 19]. Although the protocols presented there are generic and hence may be implemented with or without computational assumptions (e.g., using primitives based on quantum channels), they are less efficient by at least a factor of k, where k is the security parameter, and furthermore, they are not universally composable. (As a side note, a "stand-alone" version of our ECOT protocol would be substantially simpler, in particular with respect to the implementation of the necessary ZK proofs.)

We can also compare the ECOT functionality to the functionalities defined in [12], who use a "two-phase" approach to construct universally composable two-party/multiple-party computation protocols. In the first phase, where they construct a protocol secure against semi-honest adversaries, an important tool is the OT functionality. In the second phase, where they exhibit a "compiler" that turns protocols in the first phase into ones secure against malicious adversaries, an important tool is the "commit-and-prove" functionality, which proves general NP statements. In some sense, the ECOT functionality may be viewed as a "combination" of the OT functionality and the commit-and-prove functionality. However, we stress that since ECOT only needs to prove very simple relations (among three bits), it can be realized more efficiently.[4]

[4]We note that a commitment functionality with the capability to perform proofs on committed bits was also proposed

Recently, Damgård and Nielsen [21] presented efficient universally composable multi-party computation protocols using a different approach. Their construction is based on an efficient MPC protocol by Cramer *et al.* [15], which in turn is based on threshold homomorphic cryptosystems. Compared to our result, the Damgård-Nielsen construction works in a slightly stronger model, namely the *public key infrastructure* (PKI) model, where a trusted party not only generates a common reference string (which contains the public keys of all the paries), but also a private string for each party (as the party's secret key). On the other hand, their protocol is secure against adaptive adversaries in the so-called *non-erasing* model, where the parties are not allowed to erase any information, while our construction is secure in the erasing model only.

2 Preliminaries and Definitions

All our results are in the *common reference string* (CRS) model, which assumes that there is a string uniformly generated from some distribution and is available to all parties at the start of a protocol. This is a generalization of the *public random string* model, where a uniform distribution over fixed-length bit strings is assumed.

For a distribution Δ, we say $a \in \Delta$ to denote any element that has non-zero probability in Δ, i.e., any element in the support of Δ. We say $a \stackrel{R}{\leftarrow} \Delta$ to denote a is randomly chosen according to distribution Δ. For a set S, we say $a \stackrel{R}{\leftarrow} S$ to denote that a is uniformly drawn from S.

The universal composability framework. The universal composability framework was suggested by Canetti for defining the security and composition of protocols [10]. In this framework one first defines an "ideal functionality" of a protocol, and then proves that a particular implementation of this protocol operating in a given computational environment securely realizes this ideal functionality. The basic entities involved are n players $P_1, \ldots, P_n$, an adversary $\mathcal{A}$, and an environment $\mathcal{Z}$. The real execution of a protocol π, run by the players in the presence of $\mathcal{A}$ and an environment machine $\mathcal{Z}$, with input z, is modeled as a sequence of *activations* of the entities. The environment $\mathcal{Z}$ is activated first, generating in particular the inputs to the other players. Then the protocol proceeds by having $\mathcal{A}$ exchanging messages with the players and the environment. Finally, the environment outputs one bit, which is the output of the protocol.

The security of the protocols is defined by comparing the real execution of the protocol to an ideal process in which an additional entity, the ideal functionality $\mathcal{F}$, is introduced; essentially, $\mathcal{F}$ is an incorruptible trusted party that is programmed to produce the desired functionality of the given task. Let $\mathcal{S}$ denote the adversary in this idealized execution. The players are replaced by dummy players, who do not communicate with each other; whenever a dummy player is activated, its input is forwarded to $\mathcal{F}$ by $\mathcal{S}$, who can see the "public header" of the input.[5] As in the real-life execution, the output of the protocol execution is the one-bit output of $\mathcal{Z}$. Now a protocol π *securely realizes* an ideal functionality $\mathcal{F}$ if for any real-life adversary $\mathcal{A}$ there exists an ideal-execution adversary $\mathcal{S}$ such that no environment $\mathcal{Z}$, on any input, can tell with non-negligible probability whether it is interacting with $\mathcal{A}$ and players running π in the real-life execution, or with $\mathcal{S}$ and $\mathcal{F}$ in the ideal execution. More precisely, if the two binary distribution ensembles, $\mathrm{REAL}_{\pi,\mathcal{A},\mathcal{Z}}$ and $\mathrm{IDEAL}_{\mathcal{F},\mathcal{S},\mathcal{Z}}$, describing $\mathcal{Z}$'s output after interacting with adversary $\mathcal{A}$ and players running protocol π (resp., adversary $\mathcal{S}$ and ideal functionality $\mathcal{F}$), are computationally indistinguishable (denoted $\mathrm{REAL}_{\pi,\mathcal{A},\mathcal{Z}} \stackrel{c}{\approx} \mathrm{IDEAL}_{\mathcal{F},\mathcal{S},\mathcal{Z}}$).

by Damgård and Nielsen [20], along with efficient protocols realizing it under some specific number-theoretic assumptions. However, it was not shown that their functionality could be used in constructing protocols for general secure multi-party computation, and more specifically, oblivious transfer.

[5] This feature was added to the UC framework in [12].

Protocols typically invoke other sub-protocols. In this framework the *hybrid model* is like a real-life execution, except that some invocations of the sub-protocols are replaced by the invocation of an instance of an ideal functionality $\mathcal{F}$; this is called the "$\mathcal{F}$-hybrid model." Let $\mathrm{HYB}^{\mathcal{F}}_{\pi,\mathcal{A},\mathcal{Z}}$ denote the distribution ensemble of random variables describing the output of $\mathcal{Z}$, after interacting in the $\mathcal{F}$-hybrid model with protocol π. Let π be a protocol in the $\mathcal{F}$-hybrid model, and ρ a protocol that secures realizes $\mathcal{F}$. The composed protocol π^ρ is now constructed by replacing the first message to $\mathcal{F}$ in π by an invocation of a new copy of ρ, with fresh random input, the same *sid*, and with the contents of that message as input; each subsequent message to that copy of $\mathcal{F}$ is replaced with an activation of the corresponding copy of ρ, with the contents of that message as new input to ρ.

The UC composition theorem basically says that if ρ secure realizes $\mathcal{F}$ in the $\mathcal{G}$-hybrid model, for some functionality $\mathcal{G}$, then an execution of the composed protocol π^ρ, running in the $\mathcal{G}$-hybrid model, "emulates" an execution of protocol π in the $\mathcal{F}$-hybrid model. That is, no environment machine $\mathcal{Z}$ can distinguish whether it is interacting with $\mathcal{A}$ and π^ρ in the $\mathcal{G}$-hybrid model, or it is interacting with $\mathcal{S}$ and π in the $\mathcal{F}$-hybrid model. In other words, $\mathrm{HYB}^{\mathcal{G}}_{\pi^\rho,\mathcal{A},\mathcal{Z}} \stackrel{c}{\approx} \mathrm{HYB}^{\mathcal{F}}_{\pi,\mathcal{S},\mathcal{Z}}$.

We are designing and analyzing protocols in the CRS model, and so they will be operating in the $\mathcal{F}^{\mathcal{D}}_{\mathrm{CRS}}$-hybrid model, where $\mathcal{F}^{\mathcal{D}}_{\mathrm{CRS}}$ is the functionality that chooses a string from distribution $\mathcal{D}_k$ and hands it to all parties. Further, we will consider the "multi-session extension of $\mathcal{F}$" of Canetti and Rabin [13], denoted $\hat{\mathcal{F}}$, which runs multiple copies of $\mathcal{F}$ by identifying each copy by a special *sub-session identifier.*

The definition of $\hat{\mathcal{F}}^{R}_{\mathrm{ZK}}$, the multi-session extension of $\mathcal{F}^{R}_{\mathrm{ZK}}$, is shown below. Note the two types of indices: the *sid*, which differentiates messages to $\hat{\mathcal{F}}^{R}_{\mathrm{ZK}}$ from messages sent to other functionalities, and *ssid*, the sub-session identifier, which is unique per input message (or proof).

Functionality $\hat{\mathcal{F}}^{R}_{\mathrm{ZK}}$

$\hat{\mathcal{F}}^{R}_{\mathrm{ZK}}$ proceeds as follows, running with security parameter k, parties $P_1, \ldots, P_n$, and an adversary $\mathcal{S}$:

- Upon receiving $(\textsf{zk-prover}, sid, ssid, P_i, P_j, x, w)$ from P_i: If $R(x,w)$ then send $(\textsf{ZK-PROOF}, sid, ssid, P_i, P_j, x)$ to P_j and $\mathcal{S}$ and halt. Otherwise, ignore.

Refer to [10, 12] for further description of the UC framework.

2.1 Ω-protocols

Our constructions will use a special type of zero-knowledge protocols, namely, Ω-*protocols* [25], which are variants of the so-called Σ-protocols [16, 14].

We review some of the definitions and properties of these protocols.

Let $R = \{(x,w)\}$ be a binary relation and assume that for some given polynomial $p(\cdot)$ it holds that $|w| \leq p(|x|)$ for all $(x,w) \in R$. Furthermore, let R be testable in polynomial time. Let $L_R = \{x : (x,w) \in R\}$ be the *language* defined by the relation, and for all $x \in L_R$, let $W_R(x) = \{w : (x,w) \in R\}$ be the *witness set* for x. For any NP language L, note that there is a natural *witness relation* R containing pairs (x,w) where w is the witness for the membership of x in L, and that $L_R = L$.

A Σ-protocol (A,B) is a three move interactive protocol between a probabilistic polynomial-time prover A and a probabilistic polynomial-time verifier B, where the prover acts first. The verifier is only required to send random bits as a challenge to the prover. For some $(x,w) \in R$, the common input to both players is x while w is private input to the prover. For such given x, let (a,c,z) denote the conversation between the prover and the verifier. To compute the first and final messages, the prover invokes efficient algorithms $a(\cdot)$ and $z(\cdot)$, respectively, using (x,w) and random bits as input. Using an efficient predicate $\phi(\cdot)$, the verifier decides whether the conversation is accepting with respect to x. The relation R, the algorithms $a(\cdot)$, $z(\cdot)$ and $\phi(\cdot)$ are public. The length of the challenges is denoted t_B, and we assume that t_B only depends on the length of the common string x.

We will need to broaden this definition slightly, to deal with cheating provers. We will define $\hat{L}_R$ to be the input language, with the property that $L_R \subseteq \hat{L}_R$, and membership in $\hat{L}_R$ may be tested in polynomial time. We implicitly assume B only executes the protocol if the common input $x \in \hat{L}_R$.

All Σ-protocols presented here will satisfy the following security properties:

- *Weak special soundness*: Let (a, c, z) and (a, c', z') be two conversations, that are accepting for some given $x \in \hat{L}_R$. If $c \neq c'$, then $x \in L_R$. The pair of accepting conversations (a, c, z) and (a, c', z') with $c \neq c'$ is called a *collision* .

- *Special honest verifier zero knowledge* (SHVZK): There is a (probabilistic polynomial time) simulator M that on input $x \in L_R$ generates accepting conversations with the exact same distribution as when A and B execute the protocol on common input x (and A is given a witness w for x), and B indeed honestly chooses its challenges uniformly at random. The simulator is special in the sense that it can additionally take a random string c as input, and output an accepting conversation for x where c is the challenge. In fact, we will assume the simulator has this special property for not only $x \in L_R$, but also any $x \in \hat{L}_R$.

An Ω-protocol $(A, B)_{[\sigma]}$ for a relation $R = \{(x, w)\}$ and CRS σ, is a Σ-protocol for relation R except for the following.

1. For a given distribution ensemble $\mathcal{D}$, a common reference string σ is drawn from $\mathcal{D}_k$ and each function $a(\cdot)$, $z(\cdot)$, and $\phi(\cdot)$ takes σ as an additional input.

2. It is *computational* SHVZK, meaning that the simulator on input $x \in L_R$ generates a conversation that is only computationally indistinguishable from one generated in a real execution between A and B on common input x[6]. Specifically, there is a negligible function $\alpha(k)$ and a polynomial-time simulator $\mathcal{S}$ such that for all non-uniform probabilistic polynomial-time adversaries $\mathcal{A} = (\mathcal{A}_1, \mathcal{A}_2)$, we have that $|\Pr[\mathsf{Expt}_{\mathcal{A}}(k) = 1] - \Pr[\mathsf{Expt}^{\mathcal{S}}_{\mathcal{A}}(k) = 1]| \leq \alpha(k)$, where the experiments $\mathsf{Expt}_{\mathcal{A}}(k)$ and $\mathsf{Expt}^{\mathcal{S}}_{\mathcal{A}}(k)$ are defined as follows:

$\mathsf{Expt}_{\mathcal{A}}(k)$:	$\mathsf{Expt}^{\mathcal{S}}_{\mathcal{A}}(k)$:
$\sigma \xleftarrow{R} \mathcal{D}_k$	$\sigma \xleftarrow{R} \mathcal{D}_k$
$(x, w, s) \leftarrow \mathcal{A}_1(\sigma)$	$(x, w, s) \leftarrow \mathcal{A}_1(\sigma)$
If $(x, w) \notin R$ return 0	If $(x, w) \notin R$ return 0
$r \xleftarrow{R} \{0,1\}^*$	$c \xleftarrow{R} \{0,1\}^k$
$a \leftarrow a(x, w, r, \sigma)$	Return $\mathcal{A}_2(s, \mathcal{S}(\sigma, c))$
$c \xleftarrow{R} \{0,1\}^k$	
Return $\mathcal{A}_2(s, (a, c, z(x, w, r, c, \sigma)))$	

3. There exists a polynomial-time extractor $\mathcal{E} = (\mathcal{E}_1, \mathcal{E}_2)$ such that the reference string output by $\mathcal{E}_1(1^k)$ is statistically indistinguishable from $\mathcal{D}_k$. Furthermore, given $(\sigma, \tau) \leftarrow \mathcal{E}_1(1^k)$, if there exists two accepting conversations (a, c, z) and (a, c', z') with $c \neq c'$ for some given $x \in \hat{L}_R$, then $\mathcal{E}_2(x, \tau, (a, c, z))$ outputs w such that $(x, w) \in R$.[7]

[6]Notice that the simulator does not generate the common reference string.

[7]Notice that this definition on extractor is similar to that of weak special soundness of Σ-protocols. Having two accepting conversations sharing the same a guarantees that a witness will be extracted, although the extractor only needs one conversation.

Intuitively, Ω-protocols are proof of knowledge protocols with a straight-line extractor.

In our results to follow, we need a particular, simple instance of the main theorem from [16]. Specifically, we use a slight generalization of a corollary in [16] which enables a prover, given two relations (R_1, R_2), values $(x_1, x_2) \in \hat{L}_{R_1} \times \hat{L}_{R_2}$, and corresponding 3-move Σ-protocols $((A_1, B_1), (A_2, B_2))$, to present a 3-move Σ-protocol (A_{or}, B_{or}) for proving the existence of a w such that either $(x_1, w) \in R_1$ or $(x_2, w) \in R_2$. We call this the "OR" protocol for $((A_1, B_1), (A_2, B_2))$,

We will describe the protocol assuming the challenges from (A_1, B_1) and (A_2, B_2) are of the same length. This can easily be generalized, as long as the challenge length in the combined protocol is at least as long as the challenges from either protocol. The protocol consists of (A_1, B_1) and (A_2, B_2) running in parallel, but with the verifier's challenge c split into $c = c_1 \oplus c_2$, with c_1 as the challenge for (A_1, B_1), and c_2 as the challenge for (A_2, B_2).

The protocol for A_{or} is as follows: Without loss of generality, say A_{or} knows w such that $(x_1, w) \in R_1$. Let M_2 be the simulator for S_2. Then A_{or} runs $M_2(x_2)$ to generate (m, e, z). It sends the first message of (A_1, B_1), along with m as the first message of (A_2, B_2). On challenge c, it chooses $c_2 = e$, and $c_1 = c \oplus c_2$. It is able to provide the final response in (A_1, B_1) because it knows w, and the final response in (A_2, B_2) is simply z. The final message of A_{or} includes c_1 along with the final responses for (A_1, B_1) and (A_2, B_2). We note that this construction works for Ω-protocols as well as Σ-protocols.

3 Universally Composable Committed Oblivious Transfer

In this section we present the $\mathcal{F}_{\text{ECOT}}$ functionality, an extension of COT where in addition to the oblivious transfer, the sender can prove to the receiver (Boolean) relations among the committed bits. We will later use this functionality to implement an efficient *Joint Gate Evaluation* functionality, which in turn will enable efficient and universally composable multi-party computation. The functionality $\mathcal{F}_{\text{ECOT}}$ is shown in Figure 1. Informally, a party P_i commits to a bit b by sending an ecot-commit message to the ideal functionality $\mathcal{F}_{\text{ECOT}}$, and P_i can later open this bit by sending an ecot-open message with appropriate commitment identifier (cid) value. For P_i to obliviously transfer a bit to P_j, P_i needs to commit two bits b_0 and b_1 and P_j needs to commit to one bit b_t; after sending an ecot-transfer to $\mathcal{F}_{\text{ECOT}}$, the bit b_{b_t} is transferred to P_j and automatically committed by $\mathcal{F}_{\text{ECOT}}$ on behalf of P_j. Meanwhile, P_i does not learn anything, except that a transfer took place. Furthermore, the functionality also allows a party P_i to prove to P_j that three bits b_0, b_1, and b_2 it committed to satisfy a particular binary relation by sending an ecot-prove message to $\mathcal{F}_{\text{ECOT}}$.

As a convention, we use $\mathsf{op}_m^{(2)}$ to denote a function on two bits, where $m \in \{0,1\}^4$ is the string of bits of the Boolean function's truth table (output column). We also often identify m with the integer whose binary representation is m. (For example, $m = 1$ represents the AND function, whose truth table is 0001.)

As a technical note, we note that the Open phase is not strictly necessary since it can be simulated by the Prove phase. Take $\mathsf{op}_{0000}^{(2)}$ and $\mathsf{op}_{1111}^{(2)}$, which correspond to the all-zero and all-one functions. Then, by proving that $\mathsf{op}_{0000}^{(2)}(b_0, b_1) = b_2$ for arbitrary bits b_0 and b_1, one essentially opens bit b_2 to 0; similarly, by proving that $\mathsf{op}_{1111}^{(2)}(b_0, b_1) = b_2$, one opens b_2 to 1. We choose to include the Open phase in the functionality for clarity and efficiency (the Open phase can be realized more efficiently than the simulated Prove phase).

Before presenting a protocol that securely realizes $\mathcal{F}_{\text{ECOT}}$, we first discuss some preliminary constructions that will be used as building blocks.

Functionality $\mathcal{F}_{\mathrm{ECOT}}$

$\mathcal{F}_{\mathrm{ECOT}}$ proceeds as follows, running with parties $P_1, ..., P_n$ and an adversary $\mathcal{S}$.

- **Commit phase:** When receiving from P_i a message $\langle \textsf{ecot-commit}, sid, cid, P_j, b\rangle$, record $\langle cid, P_i, P_j, b\rangle$, send message $\langle \textsf{ECOT-RECEIPT}, sid, cid, P_i, P_j\rangle$ to P_i, P_j and $\mathcal{S}$, and ignore all future messages of the form $\langle \textsf{ecot-commit}, sid, cid, P_j, *\rangle$ from P_i and $\langle \textsf{ecot-transfer}, sid, cid, *, *, *, P_i\rangle$ from P_j.
- **Prove phase:** When receiving from P_i a message $\langle \textsf{ecot-prove}, sid, ssid, cid_0, cid_1, cid_2, P_j, m\rangle$, if the following three tuples, $\langle cid_0, P_i, P_j, b_0\rangle$, $\langle cid_1, P_i, P_j, b_1\rangle$, $\langle cid_2, P_i, P_j, b_2\rangle$, are all recorded, and $\textsf{op}_m^{(2)}(b_0, b_1) = b_2$, then send message $\langle \textsf{ECOT-PROOF}, sid, ssid, cid_0, cid_1, cid_2, P_i, m\rangle$ to P_j and $\mathcal{S}$; otherwise do nothing.
- **Transfer phase:** When receiving from P_i a message $\langle \textsf{ecot-transfer}, sid, cid, cid_0, cid_1, tcid, P_j\rangle$, if the following three tuples $\langle cid_0, P_i, P_j, b_0\rangle$, $\langle cid_1, P_i, P_j, b_1\rangle$, and $\langle tcid, P_j, P_i, b_t\rangle$, are all recorded, send message $\langle \textsf{ECOT-DATA}, sid, cid, P_i, P_j, cid_0, cid_1, tcid, b_{b_t}\rangle$ to P_j, record tuple $\langle cid, P_j, P_i, b_{b_t}\rangle$, and send message $\langle \textsf{ECOT-RECEIPT}, sid, cid, P_i, P_j, cid_0, cid_1, tcid\rangle$ to P_i and $\mathcal{S}$, and ignore all future messages of the form $\langle \textsf{ecot-commit}, sid, cid, P_i, *\rangle$ from P_i and $\langle \textsf{ecot-transfer}, sid, cid, *, *, *, P_j\rangle$ from P_j. Otherwise, do nothing.
- **Open phase:** When receiving from P_i a message $\langle \textsf{ecot-open}, sid, cid, P_i, P_j\rangle$, if the tuple $\langle cid, P_i, P_j, b\rangle$ is recorded, send message $\langle \textsf{ECOT-DATA}, sid, cid, P_i, P_j, b\rangle$ to both $\mathcal{S}$ and P_j; otherwise, do nothing.

Figure 1: *The extended committed oblivious transfer functionality*

3.1 Building blocks

In particular, we shall discuss an Ω-protocol DL for discrete log, an Ω-protocol PEREP for partial equality of representation, and their compositions.

An Ω-protocol for discrete logarithm. In the full version of their paper, Garay *et al.* [25] presented an Ω-protocol for the discrete logarithm relation. More concretely, let (p, q, g) be public parameters, where q and p are primes satisfying $q|(p-1)$ and $g \in \mathbb{Z}_p^*$ with $\textsf{order}(g) = q$. Let R be the discrete logarithm relation $R = \{(y, x) : y \equiv g^x \bmod p\}$. They constructed an Ω-protocol for the relation R. We use a slight variant of this protocol for a different relation R', which differs from R by moving g from the public parameter to the input. In other words, $R_{\textsf{DL}} = \{((y, g), x) : y \equiv g^x \bmod p\}$. We call our protocol $\textsf{DL}(y, g)$ and present it in Appendix B for completeness.

Lemma 3.1 *The protocol* $\textsf{DL}(y, g)$ *is an* Ω*-protocol.* □

An Ω-protocol for partial equality of representation. The protocol DL can be extended to one for proving partial equality of representation. Let (p, q) be public parameters, where q and p are primes satisfying $q|(p-1)$. Let R be the following relation

$$R_{\textsf{PEREP}} = \{((x_0, g_0, g_1, x_1, g_2, g_3), (\alpha_0, \alpha_1, \alpha_2)) \mid x_0 \equiv g_0^{\alpha_0} g_1^{\alpha_1} \bmod p \ \wedge \ x_1 \equiv g_2^{\alpha_0} g_3^{\alpha_2} \bmod p\} \quad (1)$$

where g_0, g_1, g_2, and g_3 all have order q. We construct an Ω-protocol for this relation and we call it $\mathsf{PEREP}(x_0, g_0, g_1, x_1, g_2, g_3)$. Intuitively, this is a proof of knowledge of presentations of x_0 over bases (g_0, g_1) and x_1 over bases (g_2, g_3) such that the exponent of x_0 over g_0 equals the exponent of x_1 over g_2.

The intuition behind this protocol is very similar to that of $\mathsf{DL}(y, g)$. The common reference string consists a Paillier public key and additional RSA modulus with three generators. The prover sends over the the encryption of α, β, γ (the representations) using the Paillier encryption key and then prove that these encryptions are correct. The additional RSA modulus are used to resolve the different moduli problem. This is a standard technique [6, 8, 23, 31, 25]. We include the protocol in Appendix B.

Lemma 3.2 *The protocol* $\mathsf{PEREP}(x_0, g_0, g_1, x_1, g_2, g_3)$ *is an* Ω*-protocol.* □

We omit the proof of this lemma since the techniques are rather standard.

Composition of Ω-protocols and conversion to UCZK protocols. As we discussed in Section 2, since Ω-protocols are special Σ-protocols, they admit efficient monotone compositions. In particular, there exist efficient Ω-protocols for proving any "AND" and "OR" compositions of the discrete log relations R_{DL} and the partial equality of representation relations R_{PEREP}.

In [25], Garay *et al.* introduced a technique to transform any Ω-protocol into a universally composable protocol by using a digital signature scheme that is existentially unforgeable against adaptive chosen-message attacks. Their transformation is efficient, if the digital signature scheme admits an efficient proof of knowledge protocol. In particular, they proved the following result.

Theorem 3.3 ([25]) *Under the strong RSA assumption or the DSA assumption, for every relation* R *that admits an* Ω*-protocol* Π*, there exists a three-round protocol* $\mathsf{UC}[\Pi]$ *that securely realizes the* $\hat{\mathcal{F}}_{\mathrm{ZK}}^{R}$ *ideal functionality in the* $\mathcal{F}_{\mathrm{CRS}}$*-hybrid model against adaptive adversaries, assuming erasing. Furthermore, the (additive) overhead of* $\mathsf{UC}[\Pi]$ *to* Π *is constant number of exponentiations plus the generation of a signature.*

(See Appendix A for discussion on the Strong RSA assumption.)

Therefore, we can plug in the Ω-protocols for proving discrete logarithm, partial equality of representation, and their "AND"/"OR" compositions into Theorem 3.3 and obtain efficient UCZK protocols for these relations. In particular, we shall use UCZK protocols for proving the following relations.

1. **"OR" of two discrete logs:**

$$R_{\mathsf{OR\text{-}DL}}((y_0, g_0, y_1, g_1), (x_0, x_1)) = R_{\mathsf{DL}}((y_0, g_0), x_0) \ \vee \ R_{\mathsf{DL}}((y_1, g_1), x_1)$$

2. **"OR"/"AND" relation of six discrete logs:**

$$\begin{aligned} & R_{\mathsf{OR\text{-}N\text{-}DL}}((y_0, y_1, y_2, y_3, y_4, y_5, g), (x_0, x_1, x_2, x_3, x_4, x_5)) = \\ & \quad ((R_{\mathsf{DL}}((y_0, g), x_0) \vee R_{\mathsf{DL}}((y_1, g), x_1)) \wedge R_{\mathsf{DL}}((y_2, g), x_2)) \ \vee \\ & \quad (R_{\mathsf{DL}}((y_3, g), x_3) \wedge R_{\mathsf{DL}}((y_4, g), x_4) \wedge R_{\mathsf{DL}}((y_5, g), x_5)) \end{aligned}$$

3. **Partial equality of representations:**

$$R_{\mathsf{PEREP}}((x_0, g_0, g_1, x_1, g_2, g_3), (\alpha_0, \alpha_1, \alpha_2))$$

4. **"OR" of partial equality of representations:**

$$R_{\textsf{OR-PEREP}}((x_0, g_0, g_1, x_1, g_2, g_3, y_0, h_0, h_1, y_1, h_2, h_3), (\alpha_0, \alpha_1, \alpha_2, \beta_0, \beta_1, \beta_2)) = R_{\textsf{PEREP}}((x_0, g_0, g_1, x_1, g_2, g_3), (\alpha_0, \alpha_1, \alpha_2)) \lor R_{\textsf{PEREP}}((y_0, h_0, h_1, y_1, h_2, h_3), (\beta_0, \beta_1, \beta_2))$$

3.2 The UCECOT protocol

We now present UCECOT, a protocol that securely realizes the $\mathcal{F}_{\text{ECOT}}$ ideal functionality in the $(\mathcal{F}_{\text{CRS}}, \hat{\mathcal{F}}_{\text{ZK}}^{R_{\textsf{OR-DL}}}, \hat{\mathcal{F}}_{\text{ZK}}^{R_{\textsf{OR-N-DL}}}, \hat{\mathcal{F}}_{\text{ZK}}^{R_{\textsf{PEREP}}}, \hat{\mathcal{F}}_{\text{ZK}}^{R_{\textsf{OR-PEREP}}})$-hybrid model, where the CRS consists of (p, q, g, h) such that q and p are primes satisfying $q|(p-1)$ and $g, h \in \mathbb{Z}_p^*$ are random elements satisfying $\textsf{order}(g) = \textsf{order}(h) = q$. p and q will also serve as the public parameters in the relations $R_{\textsf{DL}}$, $R_{\textsf{PEREP}}$, and their compositions.

We first describe the protocol.

Commit phase: On receiving private input $\langle \textsf{ecot-commit}, sid, cid, P_j, b\rangle$, assuming that cid is not used before, party P_i picks a random $r \xleftarrow{R} \mathbb{Z}_q$, computes $B \leftarrow g^r \cdot h^b \mod p$, sends message $(\textsf{ucecot-commit}, sid, cid, B)$ to party P_j, message $(\textsf{zk-prover}, sid, cid, P_i, P_j, (B, g, B/h, g), (r, r))$ to $\hat{\mathcal{F}}_{\text{ZK}}^{R_{\textsf{OR-DL}}}$, and outputs $\langle \textsf{ECOT-RECEIPT}, sid, cid, P_i, P_j\rangle$. After receiving the messages from P_i and $\hat{\mathcal{F}}_{\text{ZK}}^{R_{\textsf{OR-DL}}}$ respectively, P_j outputs $\langle \textsf{ECOT-RECEIPT}, sid, cid, P_i, P_j\rangle$.

Essentially P_i sends a Pedersen commitment [35] of bit b to P_j and uses the $\hat{\mathcal{F}}_{\text{ZK}}^{R_{\textsf{OR-DL}}}$ ideal functionality to prove that he either knows the discrete log of B (in which case P_i is committing to bit 0) or the discrete log of B/h base g (in which case P_i is committing to bit 1).

Prove phase: Suppose P_i has committed bits b_0, b_1, and b_2 to P_j using cids cid_0, cid_1, and cid_2, respectively. Further assume that their corresponding Pedersen commitments are $B_0 = g^{r_0} \cdot h^{b_0}$, $B_1 = g^{r_1} \cdot h^{b_1}$, and $B_2 = g^{r_2} \cdot h^{b_2}$. Now, upon receiving private input $\langle \textsf{ecot-prove}, sid, ssid, cid_0, cid_1, cid_2, P_j, m\rangle$, P_i is to prove to P_j that $\textsf{op}_m^{(2)}(b_0, b_1) = b_2$, using sub-session id $ssid$. We first consider the situation where $m = 1110$, in which case $\textsf{op}_m^{(2)}$ is the NAND operation. In this situation, P_i sends message $(\textsf{ucecot-prove}, sid, ssid, cid_0, cid_1, cid_2, m)$ to P_j and sends message $(\textsf{zk-prover}, sid, ssid, P_i, P_j, (B_0, B_1, B_2/h, B_0/h, B_1/h, B_2, g), (r_0, r_1, r_2, r_0, r_1, r_2))$ to $\hat{\mathcal{F}}_{\text{ZK}}^{R_{\textsf{OR-N-DL}}}$. After receiving the corresponding message from $\hat{\mathcal{F}}_{\text{ZK}}^{R_{\textsf{OR-N-DL}}}$, P_j outputs $\langle \textsf{ECOT-PROOF}, sid, ssid, cid_0, cid_1, cid_2, P_i, m\rangle$.

Intuitively, P_i is proving that $(((b_0 = 0) \lor (b_1 = 0)) \land (b_2 = 1)) \lor ((b_0 = 1) \land (b_1 = 1) \land (b_2 = 0))$.

In the case of any other binary operations $\textsf{op}_m^{(2)}$, it can be written as a composition of NANDs and then proved step by step. P_i will need to commit to all the intermediate bits and prove each NAND operation is correct. For example, consider the case where $m = 0001$ is the AND operation. Notice that $x \land y = \overline{\overline{x \land y} \land \overline{x \land y}}$ Therefore, to prove that $b_2 = b_0 \land b_1$, P_i needs to commit to a new bit $b_3 = \overline{b_0 \land b_1}$ using the protocol in the Commit phase, and then prove that both $b_3 = \overline{b_0 \land b_1}$ and that $b_2 = \overline{b_3 \land b_3}$.

Transfer phase: Suppose P_i has committed bits b_0 and b_1, and P_j has committed bit b_t, using identifiers cid_0, cid_1, and $tcid$, respectively. Further assume that the corresponding Pedersen commitments are $B_0 = g^{r_0} \cdot h^{b_0}$, $B_1 = g^{r_1} \cdot h^{b_1}$, and $B_t = g^{r_t} \cdot h^{b_t}$. Now, upone receiving private input $\textsf{ecot-transfer}, sid, cid, cid_0, cid_1, tcid, P_j\rangle$, assuming that cid is not used before, P_i is to obliviously transfers bit b_{b_t} to P_j, using session id sid and the commitment id cid for the new bit b_{b_t}. Intuitively, P_i sends two Pedersen commitments, C_0 and C_1, where C_0 is a commitment to b_0 using base B_t, and C_1 is a commitment to b_1 using base B_t/h. It also sends A_0 and A_1 generated using the same randomness as C_0 and C_1. If $b_t = 0$, then P_j knows the discrete log of B_t and can check if C_0 is a commitment to zero or not, and if $b_t = 1$, then P_j knows the discrete log of B_t/h and can check if C_1 is a commitment to zero or not.

Now we proceed to the details. P_i randomly picks $a_0, a_1 \stackrel{R}{\leftarrow} \mathbb{Z}_q$ and computes $A_0 \leftarrow g^{a_0}$, $A_1 \leftarrow g^{a_1}$, $C_0 \leftarrow B_t^{a_0} \cdot h^{b_0}$, and $C_1 \leftarrow (B_t/h)^{a_1} \cdot h^{b_1}$. P_i then sends message $(\mathsf{ucecot\text{-}transfer}, sid, cid, cid_0, cid_1, tcid, A_0, A_1, C_0, C_1)$ to P_j and sends the following four messages to the ideal functionality $\hat{\mathcal{F}}_{\mathrm{ZK}}^{R_{\mathsf{PEREP}}}$.[8]

$$(\mathsf{zk\text{-}prover}, sid, cid \circ 00, P_i, P_j, (C_0, h, B_t, B_0, h, g), (b_0, a_0, r_0))$$
$$(\mathsf{zk\text{-}prover}, sid, cid \circ 01, P_i, P_j, (C_1, h, B_t/h, B_1, h, g), (b_1, a_1, r_1))$$
$$(\mathsf{zk\text{-}prover}, sid, cid \circ 10, P_i, P_j, (A_0, g, 1, C_0, B_t, h), (a_0, 0, b_0))$$
$$(\mathsf{zk\text{-}prover}, sid, cid \circ 11, P_i, P_j, (A_1, g, 1, C_1, B_t/h, h), (a_1, 0, b_1))$$

After this, P_i erases a_0 and a_1.

After receiving the message from P_i and four messages from the ideal functionality $\hat{\mathcal{F}}_{\mathrm{ZK}}^{R_{\mathsf{PEREP}}}$, P_j does the following (otherwise P_j aborts).

If $b_t = 0$, then check if $A_0^{r_t} = C_0 \bmod p$, and set $b \leftarrow 0$ if yes and $b \leftarrow 1$ otherwise; if $b_t = 1$, then check if $A_1^{r_t} = C_1 \bmod p$, and sets $b \leftarrow 0$ if yes and $b \leftarrow 1$ otherwise. Now b is the bit P_j receives.

Next, P_j picks a random $r \stackrel{R}{\leftarrow} \mathbb{Z}_q^*$ and sets $B \leftarrow g^r \cdot h^b \bmod p$, sends message $(\mathsf{ecot\text{-}commit}, sid, cid, B)$ to party P_i, sends message $(\mathsf{zk\text{-}prover}, sid, cid, P_j, P_i, (C_0, h, A_0, B, h, g, C_1, h, A_1, B, h, g), (b, r_t, r, b, r_t, r))$ to ideal functionality $\hat{\mathcal{F}}_{\mathrm{ZK}}^{R_{\mathsf{OR\text{-}PEREP}}}$, and outputs $\langle \mathsf{ECOT\text{-}DATA}, sid, cid, P_i, P_j, cid_0, cid_1, tcid, b\rangle$. Finally, after receiving messages from P_j and $\hat{\mathcal{F}}_{\mathrm{ZK}}^{R_{\mathsf{OR\text{-}PEREP}}}$, P_i outputs $\langle \mathsf{ECOT\text{-}RECEIPT}, sid, cid, P_i, P_j, cid_0, cid_1, tcid\rangle$.

Open phase: Suppose P_i has committed a bit b to P_j using session id sid, and commitment id cid. Further assume that the commitment is $B = g^r \cdot h^b \bmod p$. Now upon receiving private input $\langle \mathsf{ecot\text{-}open}, sid, cid, P_i, P_j\rangle$, P_i opens the bit b by sending message $(\mathsf{ucecot\text{-}open}, sid, cid, b, r)$ to P_j, who then verifies that $B = g^r \cdot h^b \bmod p$, and outputs $(\mathsf{ECOT\text{-}DATA}, sid, cid, P_i, P_j, b)$ if the verification is valid.

This is exactly the opening of a Pedersen commitment.

Theorem 3.4 *Under the DDH assumption, protocol* UCECOT *securely realizes the* $\mathcal{F}_{\mathrm{ECOT}}$ *ideal functionality in the* $(\mathcal{F}_{\mathrm{CRS}}, \hat{\mathcal{F}}_{\mathrm{ZK}}^{R_{\mathsf{OR\text{-}DL}}}, \hat{\mathcal{F}}_{\mathrm{ZK}}^{R_{\mathsf{OR\text{-}N\text{-}DL}}}, \hat{\mathcal{F}}_{\mathrm{ZK}}^{R_{\mathsf{PEREP}}}, \hat{\mathcal{F}}_{\mathrm{ZK}}^{R_{\mathsf{OR\text{-}PEREP}}})$*-hybrid model against adaptive, malicious adversaries, assuming erasing.*

Proof: Let $\mathcal{A}$ be an adversary that operates against protocol UCECOT. We construct an ideal process adversary $\mathcal{S}$ such that no environment $\mathcal{Z}$ can distinguish whether it is interacting with $\mathcal{A}$ and UCECOT in the $(\mathcal{F}_{\mathrm{CRS}}, \hat{\mathcal{F}}_{\mathrm{ZK}}^{R_{\mathsf{OR\text{-}DL}}}, \hat{\mathcal{F}}_{\mathrm{ZK}}^{R_{\mathsf{OR\text{-}N\text{-}DL}}}, \hat{\mathcal{F}}_{\mathrm{ZK}}^{R_{\mathsf{PEREP}}}, \hat{\mathcal{F}}_{\mathrm{ZK}}^{R_{\mathsf{OR\text{-}PEREP}}})$-hybrid model, or with $\mathcal{S}$ in the ideal process for $\mathcal{F}_{\mathrm{ECOT}}$.

For simplicity, we assume that only one copy of $\mathcal{F}_{\mathrm{ECOT}}$ is access by $\mathcal{Z}$, since otherwise we can easily duplicate the actions for $\mathcal{S}$ for each copy of $\mathcal{F}_{\mathrm{ECOT}}$.

At the beginning of the ideal process, the ideal adversary $\mathcal{S}$ simulates the common reference string $\mathcal{F}_{\mathrm{CRS}}$. First, $\mathcal{S}$ generate two primes p and q of appropriate size such that $q|(p-1)$. Next $\mathcal{S}$ generates

[8] We assume that all the id's are binary strings, and we use "$a \circ b$" to indicate the concatenation of string a with string b.

random $g \in \mathbb{Z}_p^*$ such that $\mathsf{order}(g) = q$, a random $t \xleftarrow{R} \mathbb{Z}_q$, and $h \leftarrow g^t$. Then $\mathcal{S}$ outputs (p, q, g, h) as the common reference string for $\mathcal{F}_{\mathrm{CRS}}$ and stores t. Notice that since $\mathcal{S}$ knows the discrete log of h base g, it can equivocate the Pedersen commitment.

During the ideal process, $\mathcal{S}$ runs a simulated copy of $\mathcal{A}$. Messages received from $\mathcal{Z}$ are forwarded to the simulated $\mathcal{A}$, and messages sent by the simulated $\mathcal{A}$ to its environment are forwarded to $\mathcal{Z}$. $\mathcal{S}$ also see the *public header* (see [12]) of all the messages from uncorrupted parties to $\mathcal{F}_{\mathrm{ECOT}}$ and may decide whether or not to forward these messages. In the case of $\mathcal{F}_{\mathrm{ECOT}}$, all messages to $\mathcal{F}_{\mathrm{ECOT}}$ are in the public header (meaning that $\mathcal{S}$ can see all these messages from uncorrupted parties to $\mathcal{F}_{\mathrm{ECOT}}$) except the bit b in the $\mathsf{ecot\text{-}commit}$ message. We use "$-$" to denote this private bit. Furthermore, $\mathcal{S}$ also plays the roles of the four UCZK functionalities $\hat{\mathcal{F}}_{\mathrm{ZK}}^{R_{\mathsf{OR\text{-}DL}}}$,$\hat{\mathcal{F}}_{\mathrm{ZK}}^{R_{\mathsf{OR\text{-}N\text{-}DL}}}$, $\hat{\mathcal{F}}_{\mathrm{ZK}}^{R_{\mathsf{PEREP}}}$, and $\hat{\mathcal{F}}_{\mathrm{ZK}}^{R_{\mathsf{OR\text{-}PEREP}}}$. Furthermore, $\mathcal{S}$ maintains a record of committed bits erased in $\mathcal{F}_{\mathrm{ECOT}}$. The record consists of tuples of the form (cid, P_i, P_j, B, r, b), indicating that the committed bit of id cid is from P_i to P_j, the bit is b and the Pedersen commitment is $B = h^b \cdot g^r$. If P_i is uncorrupted, then the bit b is set to "?," indicating that $\mathcal{S}$ can equivocate on this one and in this case, $B = g^r$.

Next, we describe the behavior of $\mathcal{S}$ in detail. $\mathcal{S}$ is "event-driven," in that its actions are triggered by the messages. We itemize the behavior of $\mathcal{S}$ according to the messages it sees.

1. If $\mathcal{S}$ sees a corrupted party P_i send a message $(\mathsf{ucecot\text{-}commit}, sid, cid, B)$ to party P_j, this means that P_i (which is controlled by $\mathcal{A}$) is committing a bit to P_j. Then $\mathcal{S}$ expects a message $(\mathsf{zk\text{-}prover}, sid, cid, P_i, P_j, (B, g, B/h, g), (r_0, r_1))$ from P_i to $\hat{\mathcal{F}}_{\mathrm{ZK}}^{R_{\mathsf{OR\text{-}DL}}}$. After receiving this message, $\mathcal{S}$ checks if $B = g^{r_0}$ and if $B/h = g^{r_1}$. If neither holds, $\mathcal{S}$ ignores this message, since P_i fails the proof. If both hold, then $\mathcal{A}$ manages to find out the discrete log of h (which is $r_0 - r_1$), and $\mathcal{S}$ in this case aborts and outputs an error message ``DL broken: $h = g^{r_0 - r_1}$.''

   If $B = g^{r_0}$ but $B/h \neq g^{r_1}$, then P_i is committing to the bit $b = 0$; if $B \neq g^{r_0}$ but $B/h = g^{r_1}$, then P_i is committing to the bit $b = 1$. In either case, $\mathcal{S}$ records $(cid, P_j, P_i, B, r_b, b)$ and sends message $\langle \mathsf{ecot\text{-}commit}, sid, cid, P_j, b \rangle$ to on behalf of P_i to $\mathcal{F}_{\mathrm{ECOT}}$.

2. If $\mathcal{S}$ sees a message $\langle \mathsf{ecot\text{-}commit}, sid, cid, P_j, - \rangle$ from an uncorrupted party P_i to $\mathcal{F}_{\mathrm{ECOT}}$, this means that P_i is committing a bit to P_j. Notice that $\mathcal{S}$ cannot see the actual bit b in the message, and the place of b is filled by "$-$." $\mathcal{S}$ then forwards this message to the $\mathcal{F}_{\mathrm{ECOT}}$, and obtains a receipt $\langle \mathsf{ECOT\text{-}RECEIPT}, sid, cid, P_i, P_j \rangle$ in return. Next, $\mathcal{S}$ fakes a Pedersen commitment by picking a random $r \xleftarrow{R} \mathbb{Z}_q$, setting $B \leftarrow g^r$, and sending message $(\mathsf{ucecot\text{-}commit}, sid, cid, B)$ to party P_j. and records $(cid, P_i, P_j, B, r, ?)$. $\mathcal{S}$ also forges a message $(\mathsf{ZK\text{-}PROOF}, sid, cid, P_i, P_j, (B, g, B/h, g))$ from $\hat{\mathcal{F}}_{\mathrm{ZK}}^{R_{\mathsf{OR\text{-}DL}}}$ to P_j. Notice that if P_j is corrupted, then the messages to P_j are in fact directed to $\mathcal{A}$.

3. If $\mathcal{S}$ sees a corrupted party P_i (controlled by $\mathcal{A}$) send a message $(\mathsf{ucecot\text{-}transfer}, sid, cid, cid_0, cid_1, tcid, A_0, A_1, C_0, C_1)$ to an uncorrupted party P_j, this means that P_i (controlled by $\mathcal{A}$) is performing an oblivious transfer to P_j. In this case, $\mathcal{S}$ first verifies the it has stored three tuples $(cid_0, P_i, P_j, B_0, r_0, b_0)$, $(cid_1, P_i, P_j, B_1, r_1, b_1)$, and $(tcid, P_j, P_i, B_t, r_t, ?)$, and does nothing if not. Next, $\mathcal{S}$ plays the ideal functionality $\hat{\mathcal{F}}_{\mathrm{ZK}}^{R_{\mathsf{PEREP}}}$ to receive four messages from P_i, and verifies that

$$\begin{aligned} C_0 = h^{\alpha_{00}} \cdot B_t^{\alpha_{01}} \quad &\wedge \quad B_0 = h^{\alpha_{00}} \cdot g^{\alpha_{02}} \\ C_1 = h^{\alpha_{10}} \cdot (B_t/h)^{\alpha_{11}} \quad &\wedge \quad B_1 = h^{\alpha_{10}} \cdot g^{\alpha_{12}} \\ A_0 = g^{\beta_{00}} \quad &\wedge \quad C_0 = B_t^{\beta_{00}} \cdot h^{\beta_{01}} \\ A_1 = g^{\beta_{10}} \quad &\wedge \quad C_1 = (B_t/h)^{\beta_{10}} \cdot h^{\beta_{11}} \end{aligned}$$

If $\alpha_{00} \neq b_0$, $\mathcal{S}$ aborts and outputs `` `DL broken: $h = g^{(\alpha_{02}-r_0)/(b_0-\alpha_{00})}$.'' ``

If $\alpha_{10} \neq b_1$, $\mathcal{S}$ aborts and outputs `` `DL broken: $h = g^{(\alpha_{12}-r_1)/(b_1-\alpha_{10})}$.'' ``

If $\beta_{01} \neq b_0$, $\mathcal{S}$ aborts and outputs `` `DL broken: $h = g^{r_2 \cdot (\alpha_{01}-\beta_{00})/(\beta_{01}-b_0)}$.'' ``

If $\beta_{11} \neq b_1$, $\mathcal{S}$ aborts and outputs `` `DL broken: $h = g^{r_2 \cdot (\alpha_{11}-\beta_{10})/(\alpha_{11}+\beta_{11}-\beta_{10}-b1)}$.'' ``

Otherwise, we have $\alpha_{00} = \beta_{01} = b_0$, $\alpha_{10} = \beta_{11} = b_1$, $\alpha_{02} = r_0$, $\alpha_{12} = r_1$, $\beta_{00} = \alpha_{01}$, and $\beta_{10} = \alpha_{11}$. Setting $a_0 = \alpha_{01}$ and $a_1 = \alpha_{11}$, we see that (A_0, A_1, C_0, C_1) is indeed of the form as prescribed in protocol UCECOT.

After this, $\mathcal{S}$ sends message $\langle$ecot-transfer$, sid, cid, cid_0, cid_1, tcid, P_j\rangle$ to $\mathcal{F}_{\text{ECOT}}$, and waits for the message $\langle$ECOT-RECEIPT$, sid, cid, P_i, P_j, cid_0, cid_1, tcid\rangle$ back from it. Next, $\mathcal{S}$ fakes a commitment by setting $r \stackrel{R}{\leftarrow} \mathbb{Z}_q$, $B \leftarrow g^r$, and sending the message (ecot-commit$, sid, cid, B)$ to P_i, and forges a message (ZK-PROOF$, sid, cid, P_j, P_i, (C_0, h, A_0, B, h, g, C_1, h, A_1, B, h, g))$ from $\hat{\mathcal{F}}_{\text{ZK}}^{R_{\text{OR-PEREP}}}$ to P_i.

4. If $\mathcal{S}$ sees a message $\langle$ecot-transfer$, sid, cid, cid_0, cid_1, tcid, P_j\rangle$ from an uncorrupted party P_i to $\mathcal{F}_{\text{ECOT}}$, this means that P_i is transferring a bit to P_j obliviously. If P_j is uncorrupted, then $\mathcal{S}$ can simulate the protocol easily. In particular, it forwards this message to $\mathcal{F}_{\text{ECOT}}$, obtains a message $\langle$ECOT-RECEIPT$, sid, cid, P_i, P_j, cid_0, cid_1, tcid\rangle$ in return from $\mathcal{F}_{\text{ECOT}}$, forges a message (ucecot-transfer$, sid, cid, cid_0, cid_1, tcid, A_0, A_1, C_0, C_1)$ from P_i to P_j and then a message (ucecot-commit$, sid, cid, B)$ from P_j to P_i, where A_0, A_1, C_0, C_1 are generated as in the protocol (using $b_0 = b_1 = 0$) and $B \leftarrow g^r$ for a random $r \stackrel{R}{\leftarrow} \mathbb{Z}_q$. $\mathcal{S}$ also simulates the appropriate messages to and from the UCZK ideal functionalities $\hat{\mathcal{F}}_{\text{ZK}}^{R_{\text{PEREP}}}$ and $\hat{\mathcal{F}}_{\text{ZK}}^{R_{\text{OR-PEREP}}}$.

   Now, consider the case that P_j is corrupted. Assume that $\mathcal{S}$ has internally stored tuples $(cid_0, P_i, P_j, B_0, r_0, ?)$, $(cid_1, P_i, P_j, B_1, r_1, ?)$ and $(tcid, P_j, P_i, B_t, r_t, b_t)$. If it is not the case, $\mathcal{S}$ does nothing. Otherwise $\mathcal{S}$ does the following. Intuitively, $\mathcal{S}$ can simulate the protocol correctly since it knows the bit b_t used by the corrupted receiver, and it learns from $\mathcal{F}_{\text{ECOT}}$ the transferred bit. It sets the data for the opposite case $(1 - b_t)$ randomly, since the receiver should not learn anything about the data corresponding to $1 - b_t$. Below is the detailed description.

   (a) Forward the message to $\mathcal{F}_{\text{ECOT}}$ and obtains the message $\langle$ECOT-DATA$, sid, cid, P_i, P_j, cid_0, cid_1, tcid, b\rangle$ in return.

   (b) Set $s_0, s_1, s_2 \stackrel{R}{\leftarrow} \mathbb{Z}_q$, $A_{b_t} \leftarrow g^{s_0}$, $C_{b_t} \leftarrow (B_t/h^{b_t})^{s_0} \cdot h^b$, $A_{1-b_t} \leftarrow g^{s_1}$, and $C_{1-b_t} \leftarrow g^{s_2}$.

   (c) Send message (ucecot-transfer$, sid, cid, cid_0, cid_1, tcid, A_0, A_1, C_0, C_1)$ to P_j and simulate the four messages from $\hat{\mathcal{F}}_{\text{ZK}}^{R_{\text{PEREP}}}$ to P_j.

   (d) Receive message (ecot-commit$, sid, cid, B)$ from P_j and message (zk-prover$, sid, cid, P_j, P_i, (C_0, h, A_0, B, h, g, C_1, h, A_1, B, h, g), (\alpha_0, \alpha_1, \alpha_2, \beta_0, \beta_1, \beta_2))$ from P_j to $\hat{\mathcal{F}}_{\text{ZK}}^{R_{\text{OR-PEREP}}}$.

   (e) Check if $C_0 = h^{\alpha_0} \cdot A_0^{\alpha_1} \wedge B = h^{\alpha_0} \cdot g^{\alpha_2}$ (we call it "event 0"), or $C_1 = h^{\beta_0} \cdot A_1^{\beta_1} \wedge B = h^{\beta_0} \cdot g^{\beta_2}$ (we call it "event 1"). If none of the events happens, then stop and furthermore, ignore all ucecot-commit and ucecot-transfer messages from P_j with cid as commitment id.

   (f) If event $(1 - b_t)$ happens, abort and output certain error messages. In particular, assume that $b_t = 0$ and event 1 happens, then we have $A_1 = g^{s_1}$, $C_1 = g^{s_2}$, and $C_1 = h^{\beta_0} \cdot A_1^{\beta_1}$. If $\beta_0 = 0$, then output `` `DL broken: $C_1 = A_1^{\beta_1}$,'' ``, else output `` `DL broken: $h = g^{(s_2 - s_1 \cdot \beta_1)/\beta_0}$.'' ``. The situation for $b_2 = 1$ is similar.

   (g) Now assume that event b_t happens. For simplicity we assume that $b_t = 0$ (the situation for $b_t = 1$ is similar). In this case check if $\alpha_0 = b$. If not, we have that $A_0 = g^{s_0}$,

$C_0 = B_t^{s_0} \cdot h^b = h^b \cdot g^{r_2}$, and $C_0 = h^{\alpha_0} \cdot A_0^{\alpha_1} = h^{\alpha_0} \cdot g^{s_0 \cdot \alpha_1}$. Then, $\mathcal{S}$ aborts and outputs ``DL broken: $h = g^{(s_0 \cdot \alpha_1 - r_t)/(b - \alpha_0)}$.''

(h) Otherwise, if $b_t = 0$, then store tuple $(cid, P_j, P_i, B, \alpha_2, b)$; otherwise store $(cid, P_j, P_i, B, \beta_2, b)$.

5. If $\mathcal{S}$ sees a message $\langle \mathsf{ecot\text{-}prove}, sid, cid_0, cid_1, cid_2, P_j, m\rangle$ from an uncorrupted party P_i to $\mathcal{F}_{\text{ECOT}}$, this means that P_i is proving to P_j that $\mathsf{op}_m^{(2)}(b_0, b_1) = b_2$. $\mathcal{S}$ then forwards this message to the $\mathcal{F}_{\text{ECOT}}$. Next, after receiving message $\langle \mathsf{ECOT\text{-}PROOF}, sid, ssid, cid_0, cid_1, cid_2, P_i, m\rangle$ from $\mathcal{F}_{\text{ECOT}}$, $\mathcal{S}$ forges the various messages from the ZK functionality $\hat{\mathcal{F}}_{\text{ZK}}^{R_{\mathsf{OR\text{-}N\text{-}DL}}}$ to P_j. For example, if $m = 1110$ (the NAND function), then $\mathcal{S}$ sends message $(\mathsf{ZK\text{-}PROOF}, sid, ssid, P_i, P_j,$ $(B_0, B_1, B_2/h, B_0/h, B_1/h, B_2, g)$ to P_j, assuming that $\mathcal{S}$ has recorded tuples $(cid_0, P_i, P_j, B_0, r_0, ?)$, $(cid_1, P_i, P_j, B_1, r_1, ?)$ and $(cid_2, P_i, P_j, B_2, r_2, ?)$. For other m, multiple messages would be sent accordingly, in order to simulate the prove phase in protocol UCECOT. Again, if P_j is corrupted, then the messages to P_j are in fact directed to $\mathcal{A}$.

6. If $\mathcal{S}$ sees a corrupted party P_i send a message $(\mathsf{ucecot\text{-}prove}, sid, ssid, , cid_0, cid_1, cid_2, m)$ to P_j, this means that P_i (controlled by $\mathcal{A}$) is proving to party P_j about a binary relation for three bits. In this case, $\mathcal{S}$ checks if the tuples $(cid_0, P_i, P_j, B_0, r_0, b_0)$, $(cid_1, P_i, P_j, B_1, r_1, b_1)$ and $(cid_2, P_i, P_j, B_2, r_2, b_2)$ are all stored, and stops otherwise. Next, $\mathcal{S}$ receives further messages from P_i to the ideal functionality $\hat{\mathcal{F}}_{\text{ZK}}^{R_{\mathsf{OR\text{-}N\text{-}DL}}}$, verifies them. If all the proofs to $\hat{\mathcal{F}}_{\text{ZK}}^{R_{\mathsf{OR\text{-}N\text{-}DL}}}$ are valid, which implies that $\mathsf{op}_m^{(2)}(b_0, b_1) = b_2$, $\mathcal{S}$ then sends message $\langle \mathsf{ecot\text{-}prove}, sid, ssid, cid_0, cid_1, cid_2, P_j, m\rangle$ to $\mathcal{F}_{\text{ECOT}}$; otherwise do nothing.

7. If $\mathcal{S}$ sees a message $\langle \mathsf{ecot\text{-}open}, sid, cid, P_i, P_j\rangle$ from an uncorrupted party P_i to $\mathcal{F}_{\text{ECOT}}$, this means that P_i is opening a committed bit to P_j. $\mathcal{S}$ checks for the stored tuple $(cid, P_i, P_j, B, r, ?)$ and does nothing no such tuple is tored. Then $\mathcal{S}$ forwards this message to $\mathcal{F}_{\text{ECOT}}$ and wait to receive the message $\langle \mathsf{ECOT\text{-}DATA}, sid, cid, P_i, P_j, b\rangle$ back from it. Next, $\mathcal{S}$ sends message $(\mathsf{ucecot\text{-}open}, sid, cid, b, r - b \cdot t)$ to P_j. Here t is the discrete log of h base g. Again, if P_j is corrupted, this message in fact goes to $\mathcal{A}$.

8. If $\mathcal{S}$ sees that a corrupted party P_i send a message $(\mathsf{ucecot\text{-}open}, sid, cid, b, r)$ to an uncorrupted party P_j, this means that P_i (controlled by $\mathcal{A}$) is opening a bit to P_j. In this situation, $\mathcal{S}$ checks the stored tuple $(cid, P_i, P_j, B, r', b')$. If $B \neq h^b \cdot g^r$, then this is an invalid opening and $\mathcal{S}$ does nothing. Otherwise $\mathcal{S}$ checks if $b = b'$. If it is not the case, $\mathcal{S}$ aborts and outputs an error message ``DL broken: $h = g^{(r'-r)/(b-b')}$.'' If $b = b'$, $\mathcal{S}$ then sends message $\langle \mathsf{ecot\text{-}open}, sid, cid, P_i, P_j\rangle$ to $\mathcal{F}_{\text{ECOT}}$.

9. If $\mathcal{A}$ corrupts a party P_i, $\mathcal{S}$ needs to simulate its internal state, which consists solely of the information to all the unopened commitments. When P_i is corrupted, these committed bits are given to $\mathcal{S}$. Now, since $\mathcal{S}$ knows the discrete log of h, it can open these commitments to any value. So $\mathcal{S}$ simply generates these openings by finding all the tuples of the form $(cid, P_i, *, B, r, ?)$, for each such tuple finding out the corresponding bit b from the corruption information, and producing $r' = r - b \cdot t$ as the opening information. $\mathcal{S}$ also change the tuple $(cid, P_i, *, B, r, ?)$ to $(cid, P_i, *, B, r', b)$.

This finishes the description of $\mathcal{S}$.

Now, we show that

$$\text{HYB}_{\mathsf{UCECOT}, \mathcal{A}, \mathcal{Z}}^{\mathcal{F}_{\text{CRS}}, \hat{\mathcal{F}}_{\text{ZK}}^{R_{\mathsf{OR\text{-}DL}}}, \hat{\mathcal{F}}_{\text{ZK}}^{R_{\mathsf{OR\text{-}N\text{-}DL}}}, \hat{\mathcal{F}}_{\text{ZK}}^{R_{\mathsf{PEREP}}}, \hat{\mathcal{F}}_{\text{ZK}}^{R_{\mathsf{OR\text{-}PEREP}}}} \overset{c}{\approx} \text{IDEAL}_{\mathcal{F}_{\text{ECOT}}, \mathcal{S}, \mathcal{Z}},$$

which implies our theorem. We first sketch the structure of the proof. We constructs a sequence of "mix" experiments between the experiment HYB and the experiment IDEAL, described in more detail below, such that the experiments on the two extremes are HYB and IDEAL Then, we prove that every two adjacent experiments are indistinguishable.

Mix_0: All parties run the real protocol, and the ideal functionalities are run as specified. This is identical to experiment $\mathrm{HYB}^{\mathcal{F}_{\mathrm{CRS}},\hat{\mathcal{F}}_{\mathrm{ZK}}^{R_{\mathsf{OR\text{-}DL}}},\hat{\mathcal{F}}_{\mathrm{ZK}}^{R_{\mathsf{OR\text{-}N\text{-}DL}}},\hat{\mathcal{F}}_{\mathrm{ZK}}^{R_{\mathsf{PEREP}}},\hat{\mathcal{F}}_{\mathrm{ZK}}^{R_{\mathsf{OR\text{-}PEREP}}}}_{\mathsf{UCECOT},\mathcal{A},\mathcal{Z}}$.

Mix_1: This experiment is the same as Mix_0, except that when a corrupted sender performs a commitment or opening to an uncorrupted receiver, the tests from $\mathcal{S}$ are run. In particular, the experiment does the following: (1) using the values sent in the ZK proof of the commitment phase to either extract the committed bit or else abort with a ``DL broken'' error message, and (2) to verify that the same bit is used in the open phase, or else aborts with a ``DL broken'' error message.

Mix_2: This experiment is the same as Mix_1, except that when a corrupted sender performs a transfer to an uncorrupted receiver, the tests from $\mathcal{S}$ are run. In particular, the experiment does the following: using the values sent in the ZK proof of the transfer phase to either verify that the bits transferred correspond to the previously committed bits, or else abort with a ``DL broken'' error message.

Mix_3: This experiment is the same as Mix_2, except that when an uncorrupted sender performs a transfer to a corrupted receiver, it is simulated as in $\mathcal{S}$, where the bit b being transferred is not taken from $\mathcal{F}_{\mathrm{ECOT}}$, but instead is computed as b_{b_t}, where b_t is the (previously extracted) bit of the receiver, and b_0 and b_1 are the bits of the sender. Also, note that in this simulation, using the values sent in the ZK proof of the receiver's commitment to b, it either extracts the committed bit or else aborts with a ``DL broken'' error message.

Mix_4: This experiment is the same as Mix_3, except that (1) when an uncorrupted sender performs a proof to a corrupted receiver, the uncorrupted sender does not provide the verification information to the ideal ZK functionality, and $\mathcal{S}$ is used to forge the ZK-PROOF messages from the ZK functionality, and (2) when a corrupted sender performs a proof to an uncorrupted receiver, it verifies that the values submitted correspond to the values previously extracted for the commitments.

Mix_5: This experiment is the same as Mix_4, except that (1) $\mathcal{F}_{\mathrm{CRS}}$ is simulated as in $\mathcal{S}$, so the value of $\log_g h$ is known, and (2) when an uncorrupted sender makes a commitment (either in a commitment phase or as a receiver in the transfer phase) or opening, or is corrupted, it is simulated as in $\mathcal{S}$ (i.e., making commitments to zero, and then equivocating on an opening or a corruption). One can verify that this is now equivalent to the experiment $\mathrm{IDEAL}_{\mathcal{F}_{\mathrm{ECOT}},\mathcal{S},\mathcal{Z}}$, at least from the view of the adversary $\mathcal{A}$. Notice that only in this final experiment does $\mathcal{S}$ needs to know $\log_g h$ in order to equivocate the Pedersen commitment.

We prove a sequence of lemmas. We only sketch some of the proofs if they are standard.

Lemma 3.5 *Under the discrete log assumption,* $\mathsf{Mix}_0 \stackrel{c}{\approx} \mathsf{Mix}_1$.

Proof: The difference between Mix_0 and Mix_1 is that in Mix_1, $\mathcal{S}$ extracts the committed bits. So the only possibility to distinguish Mix_0 from Mix_1 is when a corrupted party opens a commitment in two different ways. In this case Mix_1 aborts and outputs the message ``DL broken'' along with the value of $\log_g h$. If this does not happens with negligible probability, we can convert this experiment into an algorithm $\mathcal{B}$ that breaks the discrete log assumption. The conversion is standard: the algorithm $\mathcal{B}$ with input (X, Y) runs the entire experiment, replacing the (g, h) in the common reference string by (X, Y). Notice that in experiment Mix_1, the knowledge of $\log_g h$ is not needed. Therefore, $\mathcal{B}$ can carry out the experiment Mix_1, and when Mix_1 aborts, $\mathcal{B}$ breaks the discrete log assumption. ∎

Lemma 3.6 *Under the discrete log assumption,* $\mathsf{Mix}_1 \stackrel{c}{\approx} \mathsf{Mix}_2$.

Proof: The difference between Mix_1 and Mix_2 is that when a corrupted party P_i transfers a bit to an uncorrupted one P_j, this happens in the real life model in Mix_1, while in Mix_2 $\mathcal{S}$ is run to verify if the bits transfered correspond t the previous committed ones. Therefore, the only possibility to distinguish these two experiments is when Mix_2 aborts. As in the previous proof, any time $\mathcal{S}$ aborts, it finds out $\log_g h$, and thus if Mix_2 aborts with non-negligible probability, we can construct an adversary $\mathcal{B}$ that breaks the discrete log assumption by running Mix_2. Also as in the previous proof, knowledge of $\log_g h$ is not needed by Mix_2. ∎

Lemma 3.7 *Under the DDH assumption,* $\mathsf{Mix}_2 \stackrel{c}{\approx} \mathsf{Mix}_3$.

Proof: First, with the same reasoning as in the previous proof, we may assume that the probability that $\mathcal{S}$ aborts in Mix_3 is negligible, under the discrete log assumption, which is implied by DDH.

Now, assuming that $\mathcal{S}$ never aborts, the only difference is that the tuple (A_0, C_0, A_1, C_1) is generated differently in two cases. However, we shall prove that assuming DDH, these two cases are indistinguishable. From now on, we assume that $b_t = 0$ for simplicity (the case for $b_t = 1$ is similar). Then we have $B_t = g^{r_t}$ and $b_0 = b$.

In Mix_2, the (A_0, C_0, A_1, C_1) are generated as

$$a_0, a_1 \leftarrow \mathbb{Z}_q: \; A_0 \leftarrow g^{a_0}, C_0 \leftarrow B_t^{a_0} \cdot h^{b_0} = (B_t)^{a_0} \cdot h^b, A_1 \leftarrow g^{a_1}, C_1 \leftarrow (B_t/h)^{a_1} \cdot h^{b_1} = (B_t)^{a_1} \cdot h^{b_1 - a_1},$$

In Mix_3, they are generated as

$$s_0, s_1, s_2 \leftarrow \mathbb{Z}_q: \; A_0 \leftarrow g^{s_0}, C_0 \leftarrow (B_t)^{s_0} \cdot h^b, A_1 \leftarrow g^{s_1}, C_1 \leftarrow g^{s_2}.$$

It is easy to see that in both cases, (A_0, C_0) are distributed identically, and are independent from (A_1, C_1). So we now focus on (A_1, C_1). In Mix_2, the tuple $(g, h, A_1, (B_t)^{a_1} \cdot h^{b_1}/C_1) = (g, h, g^{a_1}, h^{a_1})$ is a random DH-tuple. In Mix_3, the tuple $(g, h, A_1, (B_t)^{a_1} \cdot h^{b_1}/C_1) = (g, h, g^{s_1}, g^{-s_2} \cdot (B_t)^{a_1} \cdot h^{b_1})$ is a random 4-tuple.

Thus, if an environment $\mathcal{Z}$ can distinguish Mix_2 from Mix_3, we can use it to build another adversary $\mathcal{B}$ that breaks the DDH assumption. This is a rather standard conversion: $\mathcal{B}$ receives a tuple (x_0, x_1, x_2, x_3) as input. It generates a random $t \in \mathbb{Z}_q$, use (x_0, x_1) in the place of (g, h) in the common reference string, and carries out the experiment Mix_3. In particular, $\mathcal{B}$ runs the environment $\mathcal{Z}$ and thus can see all the bits committed to; $\mathcal{B}$ also runs $\mathcal{S}$ and thus can see all the messages transmitted, as well as the internal random bits used by $\mathcal{S}$.

Then $\mathcal{B}$ picks a random session where an uncorrupted party P_i transfers a bit to a corrupted one P_j. We use the notations in $\mathcal{S}$, and so the bit committed by P_j is b_t with Pedersen commitment B_t. The $\mathcal{B}$ replaces the tuple (A_0, A_1, C_0, C_1) sent by the $\mathcal{S}$ to P_j by $(A_0, x_2, C_0, (B_t)^{a_1} \cdot h^{b_1}/x_3)$. If the tuple (x_0, x_1, x_2, x_3), is a random DH tuple, then the experiment is identical to Mix_2; if (x_0, x_1, x_2, x_3) is a random 4-tuple, then the experiment is identical to Mix_3. Thus if the environment $\mathcal{Z}$ can distinguish Mix_2 from Mix_3, $\mathcal{B}$ can break the DDH assumption. ∎

Lemma 3.8 $\mathsf{Mix}_3 \overset{c}{\approx} \mathsf{Mix}_4$.

Proof: This is straightforward. In fact the only difference between Mix_3 and Mix_4 is that in Mix_3, the $\mathcal{F}_{\mathrm{ECOT}}$ verifies the proofs, while in Mix_4, it is the functionalities $\hat{\mathcal{F}}_{\mathrm{ZK}}^{R_{\mathsf{PEREP}}}$ and $\hat{\mathcal{F}}_{\mathrm{ZK}}^{R_{\mathsf{OR\text{-}PEREP}}}$ that do the verification. Thus these two distributions are in fact *identical.* ∎

Lemma 3.9 $\mathsf{Mix}_4 \overset{c}{\approx} \mathsf{Mix}_5$.

Proof: In Mix_4, the commitments by uncorrupted parties are carried by themselves. In Mix_5, $\mathcal{S}$ simulates these commitments without the knowledge of the actual bits committed to. In fact, $\mathcal{S}$ learns these bits only later, at the opening or when a party is corrupted. However, since $\mathcal{S}$ knows the discrete log, it can equivocate and open the commitment to arbitrary values. Thus, the lemma follows directly the fact that the Pedersen commitment is perfect hiding. ∎

Summing everything up, we know that $\mathcal{S}$ is a valid ideal process adversary and therefore the UCECOT protocol securely realizes the $\mathcal{F}_{\mathrm{ECOT}}$ ideal functionality in the $(\mathcal{F}_{\mathrm{CRS}}, \hat{\mathcal{F}}_{\mathrm{ZK}}^{R_{\mathsf{OR\text{-}DL}}}, \hat{\mathcal{F}}_{\mathrm{ZK}}^{R_{\mathsf{OR\text{-}N\text{-}DL}}}, \hat{\mathcal{F}}_{\mathrm{ZK}}^{R_{\mathsf{PEREP}}}, \hat{\mathcal{F}}_{\mathrm{ZK}}^{R_{\mathsf{OR\text{-}PEREP}}})$-hybrid model.

∎

4 Joint Gate Evaluation

In this section we show how to securely realize a two-party functionality that we call *Joint Gate Evaluation* ($\mathcal{F}_{\mathrm{JGE}}$) in the $\mathcal{F}_{\mathrm{ECOT}}$-hybrid model in the presence of a malicious, adaptive adversary. Informally, $\mathcal{F}_{\mathrm{JGE}}$ allows two parties to jointly evaluate any binary operation on two bits, and this will allow us to construct general two-party computation protocols on top of $\mathcal{F}_{\mathrm{JGE}}$. We first present the functionality, shown in Figure 2.

Functionality $\mathcal{F}_{\mathrm{JGE}}$

$\mathcal{F}_{\mathrm{JGE}}$ proceeds as follows, running with parties $P_1, ..., P_n$, and adversary $\mathcal{S}$.

- **Commit phase:** When receiving from P_i a message $\langle \mathsf{commit}, sid, cid, P_j, b\rangle$, record $\langle cid, \{P_i, P_j\}, b\rangle$, send message $\langle \mathsf{RECEIPT}, sid, cid, P_i, P_j\rangle$ to P_i, P_j and $\mathcal{S}$, and ignore all further messages of the form $\langle \mathsf{commit}, sid, cid, x, *\rangle$ and $\langle \mathsf{eval}, sid, cid, *, *, x, *\rangle$ from P_j or P_j, where $x \in \{P_i, P_j\}$.
- **Evaluate phase:** When receiving from P_i a message $\langle \mathsf{eval}, sid, cid, cid_0, cid_1, P_j, m\rangle$, if both $\langle cid_0, \{P_i, P_j\}, b_0\rangle$ and $\langle cid_1, \{P_i, P_j\}, b_1\rangle$ are recorded, then compute $b = \mathsf{op}_m^{(2)}(b_0, b_1)$, record $\langle cid, \{P_i, P_j\}, b\rangle$, send message $\langle \mathsf{EVAL\text{-}RECEIPT}, sid, cid, cid_0, cid_1, P_i, P_j, m\rangle$ to P_i, P_j and $\mathcal{S}$, and ignore all further messages of the form $\langle \mathsf{commit}, sid, cid, x, *\rangle$ and $\langle \mathsf{eval}, sid, cid, *, *, x, *\rangle$ from P_i or P_j, where $x \in \{P_i, P_j\}$. Otherwise, do nothing.
- **Open phase:** When receiving from P_i a message $\langle \mathsf{open}, sid, cid, P_j\rangle$, if the tuple $\langle cid, \{P_i, P_j\}, b\rangle$ is recorded, then send message $\langle \mathsf{DATA}, sid, cid, P_i, P_j, b\rangle$ to P_j; otherwise, do nothing.

Figure 2: *The joint gate evaluation functionality.*

At a high level, the approach we will use to realize functionality $\mathcal{F}_{\rm JGE}$ is similar to that in [26, 12]. In particular, each bit stored in $\mathcal{F}_{\rm JGE}$ will be XOR-shared by P_i and P_j, and each gate evaluation will be done by a $\binom{4}{1}$-oblivious transfer. However, our resulting construction is directly secure against a malicious, adaptive adversary, and therefore we do not need the "compiler" used in [26, 12]. This "direct" (as opposed to the "two-phase") approach makes our protocol much more efficient.

In particular, we will realize the $\mathcal{F}_{\rm JGE}$ functionality using a further generalization of the $\mathcal{F}_{\rm ECOT}$ functionality, which we call $\mathcal{F}^4_{\rm ECOT}$. The Commit and Open phases of $\mathcal{F}^4_{\rm ECOT}$ are identical to those of $\mathcal{F}_{\rm ECOT}$, but the Transfer phase performs a $\binom{4}{1}$-transfer (instead of $\binom{2}{1}$), while the Prove phase proves relations consisting of Boolean functions of *three bits* (as opposed to two).

We now present $\mathcal{F}^4_{\rm ECOT}$ in Figure 3. We use $\mathsf{op}^{(3)}_m$ to express a Boolean function of three bits, where $m \in \{0,1\}^8$ denotes the output bits of the Boolean function's truth table.

Functionality $\mathcal{F}^4_{\rm ECOT}$

$\mathcal{F}^4_{\rm ECOT}$ proceeds as follows, running with parties $P_1, ..., P_n$ and an adversary $\mathcal{S}$:

- **Commit phase:** When receiving from P_i a message $\langle \mathsf{ecot\text{-}commit}, sid, cid, P_j, b\rangle$, record $\langle cid, P_i, P_j, b\rangle$, send message $\langle \mathsf{ECOT\text{-}RECEIPT}, sid, cid, P_i, P_j\rangle$ to P_i, P_j and $\mathcal{S}$, and ignore all future messages of the form $\langle \mathsf{ecot\text{-}commit}, sid, cid, P_j, *\rangle$ from P_i and $\langle \mathsf{ecot\text{-}transfer}, sid, cid, *, *, *, *, *, P_i\rangle$ from P_j.
- **Prove phase:** When receiving from P_i a message $\langle \mathsf{ecot\text{-}prove}, sid, cid_0, cid_1, cid_2, cid_3, P_j, m\rangle$, if the following four tuples $\langle cid_0, P_i, P_j, b_0\rangle$, $\langle cid_1, P_i, P_j, b_1\rangle$, $\langle cid_2, P_i, P_j, b_2\rangle$, and $\langle cid_3, P_i, P_j, b_3\rangle$, are all recorded, and $\mathsf{op}^{(3)}_m(b_0, b_1, b_2) = b_3$, then send message $\langle \mathsf{ECOT\text{-}PROOF}, sid, cid_0, cid_1, cid_2, cid_3, P_i, m\rangle$ to P_j and $\mathcal{S}$; otherwise do nothing.
- **Transfer phase:** When receiving from P_i a message $\langle \mathsf{ecot\text{-}transfer}, sid, cid, cid_0, cid_1, cid_2, cid_3, tcid_0, tcid_1, P_j\rangle$, if the following six tuples $\langle cid_0, P_i, P_j, b_0\rangle$, $\langle cid_1, P_i, P_j, b_1\rangle$, $\langle cid_2, P_i, P_j, b_2\rangle$, $\langle cid_3, P_i, P_j, b_3\rangle$, $\langle tcid_0, P_j, P_i, b_4\rangle$, $\langle tcid_1, P_j, P_i, b_5\rangle$, are all recorded, then record $\langle cid, P_j, P_i, b_{b_t}\rangle$, $t \in \{0, 1, 2, 3\}$, where t's binary representation equals $b_4 b_5$, send messages $\langle \mathsf{ECOT\text{-}DATA}, sid, cid, P_i, P_j, cid_0, cid_1, cid_2, cid_3, tcid_0, tcid_1, b_{b_t}\rangle$ to P_j and $\langle \mathsf{ECOT\text{-}RECEIPT}, sid, cid, P_i, P_j, cid_0, cid_1, cid_2, cid_3, tcid_0, tcid_1\rangle$ to P_i and $\mathcal{S}$, and ignore all future messages of the form $\langle \mathsf{ecot\text{-}commit}, sid, cid, P_i, *\rangle$ from P_j and $\langle \mathsf{ecot\text{-}transfer}, sid, cid, *, *, *, *, *, *, P_j\rangle$ from P_i. Otherwise, do nothing.
- **Open phase:** When receiving from P_i a message $\langle \mathsf{ecot\text{-}open}, sid, cid, P_i, P_j\rangle$, if the tuple $\langle cid, P_i, P_j, b\rangle$ is recorded, send message $\langle \mathsf{ECOT\text{-}DATA}, sid, cid, P_i, P_j, b\rangle$ to both P_j and $\mathcal{S}$; otherwise, do nothing.

Figure 3: *A further generalization of the extended committed oblivious transfer functionality*

Here we sketch how the $\mathcal{F}^4_{\rm ECOT}$ functionality can be implemented in the $\mathcal{F}_{\rm ECOT}$-hybrid model. Implementation of the Commit and Open phases is straightforward. The Prove phase is based on the following construction of a three-bit Boolean gate g using five two-bit Boolean gates.[9] Let x, y, z be the inputs to g, and let $g_{x=b}$ denote the two-bit Boolean gate corresponding to g when x is set to b.

[9]Note that we use a gate that takes two inputs a, b and computes $\bar{a} \wedge b$.

Then

$$g(x,y,z) = (\bar{x} \wedge g_{x=0}(y,z)) \vee (x \wedge g_{x=1}(y,z)).$$

Using this construction, one can implement the Prove phase of $\mathcal{F}^4_{\mathrm{ECOT}}$ with five $\mathcal{F}_{\mathrm{ECOT}}$ invocations of Prove and four of Commit (to commit to the intermediate bits). The Transfer phase of $\mathcal{F}^4_{\mathrm{ECOT}}$ can be realized using $\binom{2}{1}$-transfers together with auxiliary commitments. Assume the sender wants to perform the $\binom{4}{1}$-transfer of bits b_i, $i \in \{0,1,2,3\}$. The sender commits to auxiliary bits c_i, $i \in \{0,1,2,3\}$, and d_0, d_1, and proves to the receiver that

$$\begin{aligned} b_0 &= c_0 \oplus d_0, \\ b_1 &= c_1 \oplus d_0, \\ b_2 &= c_2 \oplus d_1, \\ b_3 &= c_3 \oplus d_1. \end{aligned}$$

The $\binom{4}{1}$-transfer of (b_0, b_1, b_2, b_3) can now be realized by three $\binom{2}{1}$-transfers of the pairs (d_0, d_1), (c_0, c_1) and (c_2, c_3). Leaving further details for the interested reader, we simply state the following lemma.

Lemma 4.1 *There exists an efficient protocol that securely realizes the $\mathcal{F}^4_{\mathrm{ECOT}}$ functionality in the $\mathcal{F}_{\mathrm{ECOT}}$-hybrid model against malicious, adaptive adversaries.*

We now turn our attention to the realization of the $\mathcal{F}_{\mathrm{JGE}}$ functionality in the $\mathcal{F}^4_{\mathrm{ECOT}}$-hybrid model. As we mentioned above, each bit b (associated with an identifier *cid*) stored in $\mathcal{F}_{\mathrm{JGE}}$ is shared between P_i and P_j. More precisely, P_i has a bit b_1 and P_j has a bit b_2 such that $b = b_1 \oplus b_2$. Furthermore, each of b_1 and b_2 is a random bit by itself. Both P_i and P_j will commit to their bits to each other using identifier *cid*. To open this bit to P_i, P_j opens its share, b_2, to P_i, who then computes $b = b_1 \oplus b_2$.

In order to evaluate $c = \mathsf{op}^{(2)}_m(a,b)$, suppose P_i holds a_1 and b_1 (shares of a and b, respectively), and P_j holds a_2 and b_2. Then, P_i generates a random bit $c_1 \stackrel{R}{\leftarrow} \{0,1\}$ and computes four bits $o_{00}, o_{01}, o_{10}, o_{11}$, which are the "candidate bits" for c_2, P_j's share of bit c. Which bit is c_2 depends on P_j's shares a_2 and b_2. The actual bits are computed as in the table below.

(a_2, b_2)	P_j's output c_2
$(0,0)$	$o_{00} = c_1 \oplus \mathsf{op}^{(2)}_m(a_1, b_1)$
$(0,1)$	$o_{01} = c_1 \oplus \mathsf{op}^{(2)}_m(a_1, (b_1 \oplus 1))$
$(1,0)$	$o_{10} = c_1 \oplus \mathsf{op}^{(2)}_m((a_1 \oplus 1), b_1)$
$(1,1)$	$o_{11} = c_1 \oplus \mathsf{op}^{(2)}_m((a_1 \oplus 1), (b_1 \oplus 1))$

P_i then commits to the bits $c_1, o_{00}, o_{01}, o_{10}, o_{11}$ and proves to P_j the relations in the table using the $\mathcal{F}^4_{\mathrm{ECOT}}$ Prove phase. We use m_0, m_1, m_2, m_3 to denote the encodings of the functions associated with these relations. Next, P_i and P_j engage in a $\binom{4}{1}$-oblivious transfer so that P_j receives bit $o_{a_2 b_2}$, which is P_j's share of bit c.

The full protocol UCJGE for securely realizing $\mathcal{F}_{\mathrm{JGE}}$ in the $\mathcal{F}^4_{\mathrm{ECOT}}$-hybrid model is shown in Figure 4, with the checks for reused *cid*'s omitted.

Theorem 4.2 *Protocol* UCJGE *securely realizes the $\mathcal{F}_{\mathrm{JGE}}$ functionality in the $\mathcal{F}^4_{\mathrm{ECOT}}$-hybrid model against malicious, adaptive adversaries.*

Proof: Let $\mathcal{A}$ be an adaptive adversary that operates against protocol UCJGE in the $\mathcal{F}^4_{\mathrm{ECOT}}$-hybrid model. We construct an ideal adversary $\mathcal{S}$ such that no environment $\mathcal{Z}$ can tell with non-negligible

Protocol UCJGE

- **Commit phase:**

1. On input $\langle \mathsf{commit}, sid, cid, P_j, b\rangle$, P_i chooses $b_1 \xleftarrow{R} \{0,1\}$, sets $b_2 = b_1 \oplus b$, and sends messages $\langle \mathsf{ecot\text{-}commit}, sid, cid, P_j, b_1\rangle$ to $\mathcal{F}^4_{\mathrm{ECOT}}$, and $(\mathsf{jge\text{-}share}, sid, cid, b_2)$ to P_j.
2. Upon receiving $(\mathsf{jge\text{-}share}, sid, cid, b_2)$ from P_i and $\langle \mathsf{ECOT\text{-}RECEIPT}, sid, cid, P_i, P_j\rangle$ from $\mathcal{F}^4_{\mathrm{ECOT}}$, P_j sends $\langle \mathsf{ecot\text{-}commit}, sid, cid, P_i, b_2\rangle$ and $\langle \mathsf{open}, sid, cid, P_j, P_i\rangle$ to $\mathcal{F}^4_{\mathrm{ECOT}}$, and upon receiving $\langle \mathsf{ECOT\text{-}RECEIPT}, sid, cid, P_j, P_i\rangle$ from $\mathcal{F}^4_{\mathrm{ECOT}}$, P_j outputs $\langle \mathsf{RECEIPT}, sid, cid, P_j, P_i\rangle$
3. Upon receiving $\langle \mathsf{ECOT\text{-}RECEIPT}, sid, cid, P_i, P_j\rangle$, $\langle \mathsf{ECOT\text{-}RECEIPT}, sid, cid, P_j, P_i\rangle$ and $\langle \mathsf{ECOT\text{-}DATA}, sid, cid, P_j, P_i, b\rangle$ from $\mathcal{F}^4_{\mathrm{ECOT}}$, where $b = b_2$, P_i outputs $\langle \mathsf{RECEIPT}, sid, cid, P_i, P_j\rangle$.

- **Evaluate phase:**

1. On input $\langle \mathsf{eval}, sid, cid, cid_0, cid_1, P_j, m\rangle$, P_i chooses $c_1 \xleftarrow{R} \{0,1\}$, sets
$$o_{00} = c_1 \oplus \mathsf{op}^{(2)}_m(a_1, b_1), \qquad o_{01} = c_1 \oplus \mathsf{op}^{(2)}_m(a_1, (b_1 \oplus 1)),$$
$$o_{10} = c_1 \oplus \mathsf{op}^{(2)}_m((a_1 \oplus 1), b_1), \quad o_{11} = c_1 \oplus \mathsf{op}^{(2)}_m((a_1 \oplus 1), (b_1 \oplus 1)),$$
where a_1, b_1 are P_i's shares of bits of identifiers cid_0 and cid_1, respectively, and sends the following messages to $\mathcal{F}^4_{\mathrm{ECOT}}$: $\langle \mathsf{ecot\text{-}commit}, sid, cid, P_j, c_1\rangle$
$\langle \mathsf{ecot\text{-}commit}, sid, cid \circ 00, P_j, o_{00}\rangle$, $\langle \mathsf{ecot\text{-}commit}, sid, cid \circ 01, P_j, o_{01}\rangle$
$\langle \mathsf{ecot\text{-}commit}, sid, cid \circ 10, P_j, o_{10}\rangle$, $\langle \mathsf{ecot\text{-}commit}, sid, cid \circ 11, P_j, o_{11}\rangle$
and $(\mathsf{jge\text{-}eval}, sid, cid, cid_0, cid_1, m)$ to P_j.
2. P_i sends the following messages to $\mathcal{F}^4_{\mathrm{ECOT}}$:
$\langle \mathsf{ecot\text{-}prove}, sid, cid_0, cid_1, cid, cid \circ 00, P_j, m_0\rangle$,
$\langle \mathsf{ecot\text{-}prove}, sid, cid_0, cid_1, cid, cid \circ 01, P_j, m_1\rangle$,
$\langle \mathsf{ecot\text{-}prove}, sid, cid_0, cid_1, cid, cid \circ 10, P_j, m_2\rangle$, and
$\langle \mathsf{ecot\text{-}prove}, sid, cid_0, cid_1, cid, cid \circ 11, P_j, m_3\rangle$.
3. P_i sends $\langle \mathsf{ecot\text{-}transfer}, sid, cid, cid \circ 00, cid \circ 01, cid \circ 10, cid \circ 11, cid_0, cid_1, P_j\rangle$ to $\mathcal{F}^4_{\mathrm{ECOT}}$, and upon receiving $\langle \mathsf{ECOT\text{-}RECEIPT}, sid, cid, P_i, P_j cid \circ 00, cid \circ 01, cid \circ 10, cid \circ 11, cid_0, cid_1\rangle$, outputs $\langle \mathsf{EVAL\text{-}RECEIPT}, sid, cid, cid_0, cid_1, P_i, P_j, m\rangle$.
4. Upon receiving receipts for all the commitments and proof verification messages for all the proofs, along with $\langle \mathsf{ECOT\text{-}DATA}, sid, cid, P_i, P_j, cid \circ 00, cid \circ 01, cid \circ 10, cid \circ 11, cid_0, cid_1, o_{a_2 b_2}\rangle$ from $\mathcal{F}^4_{\mathrm{ECOT}}$, where a_2, b_2 are P_j's shares of bits of identifiers cid_0 and cid_1, respectively, P_j outputs $\langle \mathsf{EVAL\text{-}RECEIPT}, sid, cid, cid_0, cid_1, P_i, P_j, m\rangle$.

- **Open phase:**

1. On input $\langle \mathsf{open}, sid, cid, P_j\rangle$, P_i sends $\langle \mathsf{open}, sid, cid, P_i, P_j\rangle$ to $\mathcal{F}^4_{\mathrm{ECOT}}$.
2. Upon receiving $\langle \mathsf{ECOT\text{-}DATA}, sid, cid, P_i, P_j, b_1\rangle$ from $\mathcal{F}^4_{\mathrm{ECOT}}$, P_j computes $b = b_1 \oplus b_2$ and outputs $\langle \mathsf{DATA}, sid, cid, P_i, P_j, b\rangle$. Here b_2 is P_j's share of the bit of identifier cid.

Figure 4: *Protocol* UCJGE

advantage whether it is interacting with $\mathcal{A}$ and parties running protocol UCJGE in the $\mathcal{F}^4_{\mathrm{ECOT}}$-hybrid model, or with $\mathcal{S}$ in the ideal process with $\mathcal{F}_{\mathrm{JGE}}$.

$\mathcal{S}$ runs a copy of $\mathcal{A}$, simulating uncorrupted parties and $\mathcal{F}^4_{\mathrm{ECOT}}$ for $\mathcal{A}$. Messages received from $\mathcal{Z}$ are forwarded to the simulated $\mathcal{A}$, and messages sent by the simulated $\mathcal{A}$ to its environment are forwarded to $\mathcal{Z}$. $\mathcal{S}$ also sees the public header of all the messages from the uncorrupted parties to $\mathcal{F}_{\mathrm{JGE}}$, and may decide when, if ever, to forward these messages. Further, $\mathcal{S}$ maintains a record of the shares of committed bits to $\mathcal{F}_{\mathrm{JGE}}$ held by corrupted parties. For each share of a bit of identifier cid, the record contains a tuples of the form (cid, P_i, P_j, b). This indicates that b is P_i's share of bit b, which is shared with P_j.

$\mathcal{S}$ proceeds by reacting to the messages it sees.

Simulating the Commit phase: In this phase $\mathcal{F}^4_{\mathrm{ECOT}}$ is simulated as normal. We break this phase into cases:

1. Say $\mathcal{S}$ sees message $\langle \mathsf{commit}, sid, cid, P_j, - \rangle$ on the outgoing communication tape of P_i destined for $\mathcal{F}_{\mathrm{JGE}}$. Then $\mathcal{S}$ chooses a bit $b_2 \stackrel{R}{\leftarrow} \{0,1\}$ and sends messages $(\mathsf{jge\text{-}share}, sid, cid, b_2)$ from (simulated) P_i to P_j and $\langle \mathsf{ecot\text{-}commit}, sid, cid, P_j, 0 \rangle$ from P_i to $\mathcal{F}^4_{\mathrm{ECOT}}$. If P_j is corrupted $\mathcal{S}$ stores (cid, P_j, P_i, b_2) and forwards the commit message to $\mathcal{F}_{\mathrm{JGE}}$. (If P_j is uncorrupted, $\mathcal{S}$ will store the tuple and forward the message in the simulation of P_j.)

   Upon P_i receiving $\langle \mathsf{ECOT\text{-}RECEIPT}, sid, cid, P_i, P_j \rangle$, $\langle \mathsf{ECOT\text{-}RECEIPT}, sid, cid, P_j, P_i \rangle$ and $\langle \mathsf{ECOT\text{-}DATA}, sid, cid, P_j, P_i, b \rangle$ from $\mathcal{F}^4_{\mathrm{ECOT}}$, where $b = b_2$, $\mathcal{S}$ forwards $\langle \mathsf{RECEIPT}, sid, cid, P_i, P_j \rangle$ from $\mathcal{F}_{\mathrm{JGE}}$ to P_j. (Note that if P_j is uncorrupted, the commit message will have been forwarded to $\mathcal{F}_{\mathrm{JGE}}$ and (cid, P_j, P_i, b_2) will have been stored by $\mathcal{S}$.)

2. Say $\mathcal{S}$ sees an uncorrupted P_j receive messages $\langle \mathsf{ECOT\text{-}RECEIPT}, sid, cid, P_i, P_j, \rangle$ from $\mathcal{F}^4_{\mathrm{ECOT}}$ and $(\mathsf{jge\text{-}share}, sid, cid, b_2)$ from party P_i. If a corrupted P_i sent the message $\langle \mathsf{ecot\text{-}commit}, sid, cid, P_j, b_1 \rangle$ to $\mathcal{F}^4_{\mathrm{ECOT}}$, $\mathcal{S}$ sends $\langle \mathsf{commit}, sid, cid, P_j, b \rangle$ from P_i to $\mathcal{F}_{\mathrm{JGE}}$, where $b = b_1 \oplus b_2$, and stores tuple (cid, P_i, P_j, b_1). If not, $\mathcal{S}$ forwards the commit message from the uncorrupted P_i to $\mathcal{F}_{\mathrm{JGE}}$ (see above). In either case it stores tuple (cid, P_j, P_i, b_2). Then $\mathcal{S}$ sends $\langle \mathsf{ecot\text{-}commit}, sid, cid, P_j, P_i, b_2 \rangle$ and $\langle \mathsf{open}, sid, cid, P_j, P_i \rangle$ to $\mathcal{F}^4_{\mathrm{ECOT}}$.

   Upon P_j receiving $\langle \mathsf{ECOT\text{-}RECEIPT}, sid, cid, P_j, P_i \rangle$ from $\mathcal{F}^4_{\mathrm{ECOT}}$, $\mathcal{S}$ forwards the message $\langle \mathsf{RECEIPT}, sid, cid, P_j, P_i \rangle$ from $\mathcal{F}_{\mathrm{JGE}}$ to P_j.

Simulating the Evaluate phase: Again in this phase $\mathcal{F}^4_{\mathrm{ECOT}}$ is simulated as normal, except that when it receives $\mathsf{ecot\text{-}prove}$ messages from uncorrupted parties the $\mathsf{ECOT\text{-}PROOF}$ messages get sent without verifying the relation, and a transfer causes a zero bit to be committed for the receiver, without actually performing any transfer.

We break this phase into cases:

1. Say $\mathcal{S}$ sees P_j receive five $\langle \mathsf{ECOT\text{-}RECEIPT}, \cdots \rangle$ messages from $\mathcal{F}^4_{\mathrm{ECOT}}$, committing, say, to bits $c_1, o_{00}, o_{01}, o_{10}, o_{11}$ from P_i, a message $(\mathsf{jge\text{-}eval}, sid, cid, cid_0, cid_1, m)$ from P_i, four $\langle \mathsf{ECOT\text{-}PROOF}, \cdots P_i, m_\ell \rangle$ messages from $\mathcal{F}^4_{\mathrm{ECOT}}$ (with each m_ℓ corresponding to m as in the protocol) and a message $\langle \mathsf{ECOT\text{-}DATA}, sid, cid, P_i, P_j cid \circ 00, cid \circ 01, cid \circ 10, cid \circ 11, cid_0, cid_1, c_2 \rangle$ from $\mathcal{F}^4_{\mathrm{ECOT}}$. (Assume the cid's are consistent in all these messages.) If P_i is corrupted, $\mathcal{S}$ stores the tuple (cid, P_i, P_j, c_1). If the message $\langle \mathsf{eval}, sid, cid, cid_0, cid_1, P_j \rangle$ has not been sent to $\mathcal{F}_{\mathrm{JGE}}$, $\mathcal{S}$ sends this message (or simply forwards it if P_i is uncorrupted). The message $\langle \mathsf{EVAL\text{-}RECEIPT}, sid, cid, cid_0, cid_1 P_i, P_j, m \rangle$ is then forwarded from $\mathcal{F}_{\mathrm{JGE}}$ to P_j.

2. Say $\mathcal{S}$ sees message $\langle \mathsf{eval}, sid, cid, cid_0, cid_1, P_j, m \rangle$ from an uncorrupted party P_i to $\mathcal{F}_{\mathrm{JGE}}$. $\mathcal{S}$ sends five $\langle \mathsf{ecot\text{-}commit}, \cdots \rangle$ messages to $\mathcal{F}^4_{\mathrm{ECOT}}$ with values equal to zero, and the ap-

propriate *cid*'s (corresponding to the commitments to $c_1, o_{00}, o_{01}, o_{10}, o_{11}$), sends the four $\langle \mathsf{ecot\text{-}prove}, \cdots \rangle$ messages to $\mathcal{F}^4_{\mathrm{ECOT}}$ (which are not checked by the simulated $\mathcal{F}^4_{\mathrm{ECOT}}$), and sends the message $\langle \mathsf{ecot\text{-}transfer}, sid, cid, \cdots, P_j \rangle$ to $\mathcal{F}^4_{\mathrm{ECOT}}$. When P_i receives all the receipt messages, $\mathcal{S}$ forwards the original message to $\mathcal{F}_{\mathrm{JGE}}$ (if it has not been forwarded already, see above) and forwards the message $\langle \mathsf{EVAL\text{-}RECEIPT}, sid, cid, cid_0, cid_1 P_i, P_j, m \rangle$ from $\mathcal{F}_{\mathrm{JGE}}$ to P_i.

Simulating the Open phase: We break this into cases:

1. Say $\mathcal{S}$ sees an uncorrupted P_j receive message $\langle \mathsf{ECOT\text{-}DATA}, sid, cid, P_i, P_j, - \rangle$ from $\mathcal{F}^4_{\mathrm{ECOT}}$. If there is not already a message $\langle \mathsf{DATA}, sid, cid, P_i, P_j, - \rangle$ on the outgoing communication tape of $\mathcal{F}_{\mathrm{JGE}}$, then P_i must be corrupted, so $\mathcal{S}$ sends $\langle \mathsf{open}, sid, cid, P_j \rangle$ from P_i to $\mathcal{F}_{\mathrm{JGE}}$. In either case, $\mathcal{S}$ then forwards the message $\langle \mathsf{DATA}, sid, cid, P_i, P_j, - \rangle$ from $\mathcal{F}_{\mathrm{JGE}}$ to P_j.

2. Say $\mathcal{S}$ sees message $\langle \mathsf{open}, sid, cid, P_j \rangle$ on the outgoing communication tape of an uncorrupted P_i destined for $\mathcal{F}_{\mathrm{JGE}}$. Then $\mathcal{S}$ sends a message $\langle \mathsf{ecot\text{-}open}, sid, cid, P_i, P_j \rangle$ from (simulated) P_i to $\mathcal{F}^4_{\mathrm{ECOT}}$. Once that message is forwarded to $\mathcal{F}^4_{\mathrm{ECOT}}$, $\mathcal{S}$ forwards the message $\langle \mathsf{open}, sid, cid, P_j \rangle$ to $\mathcal{F}_{\mathrm{JGE}}$. Now say a message $\langle \mathsf{DATA}, sid, cid, P_i, P_j, - \rangle$ appears on the outgoing communication tape of $\mathcal{F}_{\mathrm{JGE}}$. If P_j is corrupted, $\mathcal{S}$ will see the bit b in this message, and will have stored a tuple (cid, P_j, P_i, b_2). Then it can compute b_1 (and store (cid, P_i, P_j, b_1)) if it is not already stored, and send the message $\langle \mathsf{ECOT\text{-}DATA}, sid, cid, P_i, P_j, b_1 \rangle$ from $\mathcal{F}^4_{\mathrm{ECOT}}$ to P_j. If P_j is uncorrupted, $\mathcal{S}$ simply sends the message $\langle \mathsf{ECOT\text{-}DATA}, sid, cid, P_i, P_j, 0 \rangle$ from $\mathcal{F}^4_{\mathrm{ECOT}}$ to P_j.

Simulating corruptions: If $\mathcal{A}$ corrupts a (simulated) party P_i, $\mathcal{S}$ corrupts P_i and needs to simulate its internal state, which consists of the shares of all unopened committed and evaluated bits. For every bit b of identifier *cid* shared by P_i and P_j, $\mathcal{S}$ does the following.

- If (cid, P_i, P_j, b_1) is stored, $\mathcal{S}$ uses the share b_1.
- If (cid, P_j, P_i, b_2) is stored, and $\mathcal{S}$ learns b upon corruption (because this bit was committed by P_i or opened by P_j, or because P_j is also corrupted, and thus $\mathcal{S}$ can compute all bits shared by P_i and P_j), $\mathcal{S}$ computes $b_1 = b \oplus b_2$, stores (cid, P_i, P_j, b_1) and uses the share b_1.
- If neither tuple is stored (because this is an unopened bit from an evaluation and P_j is uncorrupted), $\mathcal{S}$ generates $b_1 \xleftarrow{R} \{0,1\}$, stores (cid, P_i, P_j, b_1), and uses the share b_1.

This concludes the description of $\mathcal{S}$. It is straightforward to see that the simulation is *perfect*, i.e., $\mathrm{HYB}^{\mathcal{F}^4_{\mathrm{ECOT}}}_{\mathsf{UCJGE},\mathcal{A},\mathcal{Z}}$ and $\mathrm{IDEAL}_{\mathcal{F}_{\mathrm{JGE}},\mathcal{S},\mathcal{Z}}$ are identically distributed. ◻

5 Efficient and Universally Composable Two-Party Computation

In this section we show how to securely realize any adaptively well-formed two-party functionality in the presence of malicious adaptive adversaries in the $\mathcal{F}_{\mathrm{JGE}}$-hybrid model. Our construction is similar to the constructions in [27, 26, 12] for semi-honest adversaries. However, since our $\mathcal{F}_{\mathrm{JGE}}$ functionality is secure in the presence of malicious adversaries, we are able to obtain a two-party protocol secure against malicious adversaries directly.

We first review some of the assumptions about two-party functionalities we use in our paper, which are also used in [12]. We let $\mathcal{F}$ be an ideal two-party functionality, and we let P_1 and P_2 be the participating parties. We assume that $\mathcal{F}$ may be represented via a family $\mathcal{C}_{\mathcal{F}}$ of Boolean circuits, the

kth circuit representing an activation of $\mathcal{F}$ with security parameter k. Without loss of generality, we assume the circuits are composed entirely of NAND gates.[10]

For simplicity, we assume that in each activation, (1) at most one party has an input to $\mathcal{F}$ with at most k bits, (2) each party may receive at most k bits as output from $\mathcal{F}$, (3) $\mathcal{F}$ is a deterministic function, and (4) the local state of $\mathcal{F}$ after each activation can be described by at most k bits. The initial state of $\mathcal{F}$ is described by k zero bits. We assume that messages sent from $\mathcal{A}$ to $\mathcal{F}$ are ignored, and there are no messages from $\mathcal{F}$ to $\mathcal{A}$.

We note that the "deterministic function" assumption about $\mathcal{F}$ is without loss of generality, since we can always realize a probabilistic functionality $\mathcal{F}$ using a deterministic one $\mathcal{F}'$ as follows. Assuming that $\mathcal{F}$ needs k random bits, then $\mathcal{F}'$ receives a k-bit string as auxiliary input from each participating party upon the first activation, and then runs $\mathcal{F}$ using the XOR of these strings as the random bit string. It is easy to see that the simple protocol where each party sends a random k-bit string as the auxiliary input to $\mathcal{F}'$ securely realizes the ideal functionality $\mathcal{F}$ in the $\mathcal{F}'$-hybrid model.[11]

The following protocol $\Pi_{\mathcal{F}}$ realizes an activation of $\mathcal{F}$ when P_1 sends a message to $\mathcal{F}$. (The case for P_2 is analogous.) We assume that both P_1 and P_2 hold an *sid* as auxiliary input. When P_1 is activated with input (sid, v), it initiate a protocol with P_2 to perform a joint gate-by-gate evaluation of the appropriate circuit in $\mathcal{C}_{\mathcal{F}}$.

Formally, they carry out the following protocol.

Initialization: When P_1 receives (sid, v), it checks if this is the first activation of $\mathcal{F}$, and if so it sets up the internal state. Then it commits to its private input.

> **Setting up the internal state:** For $i = 1, 2, ..., k$, P_1 sends messages $\langle \mathsf{commit}, sid, cid_i, P_2, 0\rangle$ and then $\langle \mathsf{open}, sid, cid_i, P_2\rangle$ to $\mathcal{F}_{\rm JGE}$. P_1 waits to receive the appropriate receipts. P_2 aborts if any of the bits are not zero. Effectively, P_1 commits to the initial internal state of $\mathcal{F}$ (which is all zeros), and by opening them immediately, it proves to P_2 that these bits are indeed all-zero.[12]
>
> **Committing to the private input:** For $i = 1, 2, ..., k$, P_1 sends messages $\langle \mathsf{commit}, sid, cid_i, P_1, v_i\rangle$ to $\mathcal{F}_{\rm JGE}$ and waits to receive the appropriate receipts. Here we assume that $v = v_1 v_2 \cdots v_k$.[13] P_2 simply records the receipts received from $\mathcal{F}_{\rm JGE}$.

Gate-by-gate evaluation: For each NAND gate in the circuit, P_1 determines the commitment identifiers associated with the inputs to that NAND gate, say cid_0 and cid_1, creates a new unique commitment identifier cid, sends message $\langle \mathsf{eval}, sid, cid, cid_0, cid_1, P_2, 1110\rangle$ to $\mathcal{F}_{\rm JGE}$, and waits for the appropriate receipt. Here $m = 1110$ is the encoding of the NAND operation. P_2 simply records the receipts received from $\mathcal{F}_{\rm JGE}$.

Output: P_2 verifies from all its receipts that P_1 had $\mathcal{F}_{\rm JGE}$ perform the correct computation on the appropriate bits. Then for each output bit of $\mathcal{F}$, it is either an internal state bit, or a bit addressed to either P_1 or P_2 (we have assumed that $\mathcal{F}$ does not communicate with $\mathcal{A}$). In the former case, P_1 and P_2 do not need to do anything. They simply store the identifier of this bit,

[10] This is entirely for simplicity. Note that the $\mathcal{F}_{\rm JGE}$ functionality can be used to evaluate any gate of fan-in two.

[11] Note that if adaptive corruptions are allowed, then this is actually only true for adaptively well-formed functionalities. See [12] for a discussion on this point, and the modifications necessary for an ideal adversary in the case of probabilistic functions.

[12] Note that the *cid*'s used here and elsewhere in the protocol must all be unique bit strings that indicate the bit's use in the circuit. For instance, the cid_i here could be the bit encoding of $\langle \mathsf{state}, i\rangle$.

[13] To indicate the use of each of these bits in the circuit, one could, for instance, set cid_i to be the bit encoding of $\langle \mathsf{input}, P_1, a, i\rangle$, where a is the activation number.

so that they can use it in the next activation. In the latter case, assuming that this bit, with identifier cid, is addressed to P_2, P_1 sends a message $\langle \mathsf{open}, sid, cid, P_2\rangle$ to $\mathcal{F}_{\mathrm{JGE}}$ and P_2 extracts the bit b from the message $\langle \mathsf{DATA}, sid, cid, \{P_1, P_2\}, b\rangle$ received from $\mathcal{F}_{\mathrm{JGE}}$. The protocol for the case for a bit addressed to P_1 is the same, but with P_1 and P_2 switched.

Messages that are out of order are dealt with using tagging, as in [12].

Theorem 5.1 *Let $\mathcal{F}$ be a two-party adaptively well-formed functionality. Then $\Pi_{\mathcal{F}}$ securely realizes $\mathcal{F}$ in the $\mathcal{F}_{\mathrm{JGE}}$-hybrid model, in the presence of malicious adaptive adversaries.*

Proof: The proof follows the proof of Claim 4.4 in [12].

Let $\mathcal{A}$ be a malicious, adaptive adversary that interacts with parties running $\Pi_{\mathcal{F}}$ in the $\mathcal{F}_{\mathrm{JGE}}$-hybrid model. We will construct an adversary $\mathcal{S}$ in the ideal process for $\mathcal{F}$ such that no environment $\mathcal{Z}$ can distinguish whether it is interacting with $\mathcal{A}$ and $\Pi_{\mathcal{F}}$ in the $\mathcal{F}_{\mathrm{JGE}}$-hybrid model, or with $\mathcal{S}$ and $\mathcal{F}$ in the ideal process. $\mathcal{S}$ runs $\mathcal{A}$ (simulating the uncorrupted parties and $\mathcal{F}_{\mathrm{JGE}}$ for it), and proceeds as follows.

Simulating the communication with $\mathcal{Z}$: $\mathcal{S}$ directly forwards any messages between $\mathcal{Z}$ and $\mathcal{A}$.

Simulating the input stage: In this stage, $\mathcal{F}_{\mathrm{JGE}}$ is simulated normally. Say P_1 is uncorrupted. Then when P_1 receives an input from $\mathcal{Z}$, $\mathcal{S}$ sees a message on the outgoing communication tape of P_1 for $\mathcal{F}$, but can only view the header. $\mathcal{S}$ does not forward this message directly, since $\mathcal{A}$ may corrupt P_1 at any time. $\mathcal{S}$ simulates P_1 for $\mathcal{A}$ by placing k messages $\langle \mathsf{commit}, sid, cid_i, P_2, 0\rangle$ on its (simulated) outgoing communication tape for $\mathcal{F}_{\mathrm{JGE}}$. If $\mathcal{A}$ forwards all these messages to $\mathcal{F}_{\mathrm{JGE}}$ before corrupting P_1, $\mathcal{S}$ forwards the original message from P_1 to $\mathcal{F}$. $\mathcal{S}$ also waits until P_1 receives the appropriate receipts.

If P_1 is corrupted, $\mathcal{S}$ simply waits for $\mathcal{A}$ to deliver the messages $\langle \mathsf{commit}, sid, cid_i, P_2, v_i\rangle$, for $i = 1, 2, ..., k$, from P_1 to $\mathcal{F}_{\mathrm{JGE}}$, and then delivers a message (sid, v) from P_1 to $\mathcal{F}$, where $v = v_1 v_2 \cdots v_k$.

If P_2 is uncorrupted, $\mathcal{S}$ simply records the receipts the simulated P_2 received from from $\mathcal{F}_{\mathrm{JGE}}$.

If this is the first activation, the internal state for $\mathcal{F}$ must also be set up. If P_1 is uncorrupted, $\mathcal{S}$ runs the actual protocol for the simulated P_1. If P_1 is corrupted, $\mathcal{S}$ waits for $\mathcal{A}$ to deliver the commit messages from P_1 to $\mathcal{F}_{\mathrm{JGE}}$. If P_2 is uncorrupted, $\mathcal{S}$ runs the actual protocol for the simulated P_2.

Simulating the circuit evaluation stage: In this stage, $\mathcal{F}_{\mathrm{JGE}}$ is simulated normally. Each NAND gate is computed by P_1 sending an eval message to $\mathcal{F}_{\mathrm{JGE}}$.

If P_1 is uncorrupted, $\mathcal{S}$ sends the eval message specified in the protocol from the simulated P_1 to $\mathcal{F}_{\mathrm{JGE}}$, and waits for the appropriate receipt. If P_1 is corrupted, $\mathcal{S}$ waits for $\mathcal{A}$ to deliver a message $\langle \mathsf{eval}, sid, cid, cid_0, cid_1, P_1, m\rangle$ from P_1 to $\mathcal{F}_{\mathrm{JGE}}$.

If P_2 is uncorrupted, $\mathcal{S}$ simply records the receipts the simulated P_2 received from from $\mathcal{F}_{\mathrm{JGE}}$.

Simulating the output stage: At this point, $\mathcal{F}$ should have sent DATA messages to P_1 and P_2, but $\mathcal{S}$ has not delivered any DATA message to an uncorrupted party yet.

If P_2 is uncorrupted and has not received the appropriate receipts in the preveious stages, $\mathcal{S}$ simulates P_2 aborting.

Now we break the simulation into cases.

1. P_1 and P_2 are uncorrupted: $\mathcal{S}$ sends the appropriate open messages from the simulated P_1 and P_2 to $\mathcal{F}_{\text{JGE}}$, and once an open message is received by (the simulated) $\mathcal{F}_{\text{JGE}}$, $\mathcal{S}$ sends the appropriate DATA message from $\mathcal{F}_{\text{JGE}}$, but with a data value of zero. (Note that the values in these messages are not seen by $\mathcal{A}$ and may be set arbitrarily.) Once all DATA messages have been received by (the simulated) P_1 (resp., P_2), $\mathcal{S}$ forwards the DATA message from $\mathcal{F}$ to P_1 (resp., P_2).
2. P_1 is corrupted, but P_2 is uncorrupted: $\mathcal{S}$ sends the appropriate open messages from the simulated P_2. Since P_1 is corrupted, $\mathcal{S}$ has received the DATA message from $\mathcal{F}$ to P_1, and thus knows the output bits for P_1. So $\mathcal{S}$ sends a message $\langle \mathsf{DATA}, sid, cid, \{P_1, P_2\}, b\rangle$ to P_1 for each output bit. Also for each open message deliver to $\mathcal{F}_{\text{JGE}}$ from P_1, $\mathcal{S}$ sends the appropriate DATA message from $\mathcal{F}_{\text{JGE}}$ to P_2 but with a data value of zero. (Again note that the values in these messages are not seen by $\mathcal{A}$ and may be set arbitrarily.) Once all DATA messages have been received by (the simulated) P_2 $\mathcal{S}$ forwards the DATA message from $\mathcal{F}$ to P_2.
3. P_1 is uncorrupted, but P_2 is corrupted: Analogous to the previous case.
4. Both P_1 and P_2 are corrupted: $\mathcal{S}$ simulates $\mathcal{F}_{\text{JGE}}$ normally.

Simulation of corruptions: If a (simulated) party P_1 is corrupted by $\mathcal{A}$, $\mathcal{S}$ corrupts P_1 and needs to simulate the internal state of (the simulated) P_1 for $\mathcal{A}$ to match the messages sent to and received from $\mathcal{F}$, and in particular, the input and output bits of the computation for P_1. But this is straightforward, since upon corrupting P_1, $\mathcal{S}$ obtains the messages sent to and received from $\mathcal{F}$, and can patch the state of P_1 appropriately. The same is true for P_2.

It is straightforward to see that the simulation is *perfect*, i.e., that even a computationally unbounded environment cannot distinguish the ideal process with ideal functionality $\mathcal{F}$ from the $\mathcal{F}_{\text{JGE}}$-hybrid model with the protocol $\Pi_{\mathcal{F}}$. ∎

6 Efficient and Universally Composable Multi-Party Computation

In this section we show how to extend the results from previous sections to securely realize any well-formed multi-party functionality in the presence of malicious adaptive adversaries corrupting an arbitrary number of parties. Our construction is similar to that in [12] for semi-honest adversaries. But, again as in the two-party case, we are able to construct building blocks that can withstand malicious adversaries, and therefore our construction is secure against malicious, adaptive adversaries directly.

In particular, we assume that there are n parties participating in the computation, which are denoted by $P_1, P_2, ..., P_n$. We assume that the functionality $\mathcal{F}$ is represented by a family $\mathcal{C}_{\mathcal{F}}$ of Boolean circuits, the kth circuit representing an activation of $\mathcal{F}$ with security parameter k. Without loss of generality, we assume the circuits are composed entirely of AND and XOR gates. As before, we assume that in each activation, (1) at most one party has an input to $\mathcal{F}$ with at most k bits, (2) each party may receive at most k bits as output from $\mathcal{F}$, (3) $\mathcal{F}$ is a deterministic function, and (4) the local state of $\mathcal{F}$ after each activation can be described by at most k bits. The initial state of $\mathcal{F}$ is described by k zero bits. We assume that messages sent from the $\mathcal{A}$ to $\mathcal{F}$ are ignored, and there are no messages from $\mathcal{F}$ to $\mathcal{A}$.

In order to securely realize $\mathcal{F}$, we basically follow the same approach as in the two-party case. However, we first need to extend some of our constructions from previous sections to suit the multiple-party case.

6.1 Broadcast and the one-to-many ZK functionalities

We assume an *authenticated broadcast* channel available to all participating parties. The channel is modeled by the broadcast functionality $\mathcal{F}_{\mathrm{BC}}$ in Figure 5. The functionality guarantees the authenticity of a message, i.e., that no party P_i can fake a message from P_j. This is also the assumption used in [12], and we refer the readers to [12, 29] for more in-depth discussions.

Functionality $\mathcal{F}_{\mathrm{BC}}$

$\mathcal{F}_{\mathrm{BC}}$ proceeds as follow, running with parties $P_1, ..., P_n$ and an adversary $\mathcal{S}$.

- Upon receiving a message $(\mathsf{broadcast}, sid, \mathcal{P}, x)$ from P_i, where $\mathcal{P}$ is a set of parties, send $(\mathsf{BCAST\text{-}MSG}, sid, P_i, \mathcal{P}, x)$ to all parties in $\mathcal{P}$ and $\mathcal{S}$, and halt.

Figure 5: *The $\mathcal{F}_{\mathrm{BC}}$ broadcast functionality*

We also need an extension of the ZK functionality, namely the one-to-many ZK functionality, denoted by $\mathcal{F}_{\mathrm{mZK}}$. Intuitively, this functionality allows a single prover to prove a theorem to multiple verifiers simultaneously. See Figure 6 for a formal definition.

Functionality $\mathcal{F}^R_{\mathrm{mZK}}$

$\mathcal{F}^R_{\mathrm{mZK}}$ proceeds as follow, running with parties $P_1, ..., P_n$ and an adversary $\mathcal{S}$, and parametrized with a relation R:

- Upon receiving a message $(\mathsf{zk\text{-}prover}, sid, \mathcal{P}, x, w)$ from P_i, where $\mathcal{P}$ is a set of parties, if $R(x, w) = 1$, then send $(\mathsf{ZK\text{-}PROOF}, sid, P_i, \mathcal{P}, x)$ to all parties in $\mathcal{P}$ and $\mathcal{S}$ and halt; otherwise halt.

Figure 6: *The $\mathcal{F}^R_{\mathrm{mZK}}$ broadcast functionality*

We observe that the UCZK construction by Garay *et al.* [25] can be naturally extended to a one-to-many UCZK protocol with the additional broadcast functionality. Roughly speaking, P_i (the prover) runs an independent copy of the two-party UCZK protocol with every party $P_j \in \mathcal{P}$ using a unique sid, and all messages are broadcast. Each P_j accepts if and only if all the conversations are accepting. It is straightforward to construct an ideal adversary $\mathcal{S}$. If the prover is uncorrupted, $\mathcal{S}$ simply runs a multi-party UCZK simulator for every copy of the UCZK protocol. If the prover is corrupted and there is at least one uncorrupted verifier, $\mathcal{S}$ can extract the witness. If all parties are corrupted, the simulation is straightforward. The conversion remains efficient. Therefore we have the following theorem.

Theorem 6.1 *Under the strong RSA assumption or the DSA assumption, for every relation R that admits an Ω-protocol Π, there exists a three-round protocol $\mathsf{UC}[\Pi]$ that securely realizes the $\mathcal{F}^R_{\mathrm{mZK}}$ ideal functionality in the $(\mathcal{F}_{\mathrm{CRS}}, \mathcal{F}_{\mathrm{BC}})$-hybrid model against adaptive adversaries, assuming erasing. Furthermore, the computation complexity of $\mathsf{UC}[\Pi]$ is that of Π plus constant number of exponentiations and the generation of a signature, times the number of receiving parties.*

6.2 Multi-party ECOT

We also extend the $\mathcal{F}_{\mathrm{ECOT}}$ functionality to the multi-party case, where the proof phase is replaced by a one-to-many proof and the receipts are sent to all participating parties. The functionality is denoted by $\mathcal{F}_{\mathrm{mECOT}}$ and is given in Figure 7.

Functionality $\mathcal{F}_{\mathrm{mECOT}}$

$\mathcal{F}_{\mathrm{mECOT}}$ proceeds as follows, running with parties $P_1, ..., P_n$ and an adversary $\mathcal{S}$.

- **Commit phase:** When receiving from P_i a message $\langle \mathsf{ecot\text{-}commit}, sid, cid, \mathcal{P}, b\rangle$, record $\langle cid, P_i, \mathcal{P}, b\rangle$, send message $\langle \mathsf{ECOT\text{-}RECEIPT}, sid, cid, P_i, \mathcal{P}\rangle$ to all parties in $\mathcal{P}$ and $\mathcal{S}$, and ignore all future messages of the form $\langle \mathsf{ecot\text{-}commit}, sid, cid, \mathcal{P}, *\rangle$ from P_i and all furture messages of the form $\langle \mathsf{ecot\text{-}transfer}, sid, cid, *, *, *, P_i, \mathcal{P}\rangle$ from any party in $\mathcal{P}$.

- **Prove phase:** When receiving from P_i a message $\langle \mathsf{ecot\text{-}prove}, sid, ssid, cid_0, cid_1, cid_2, \mathcal{P}, m\rangle$, if the following three tuples, $\langle cid_0, P_i, \mathcal{P}, b_0\rangle$, $\langle cid_1, P_i, \mathcal{P}, b_1\rangle$, and $\langle cid_2, P_i, \mathcal{P}, b_2\rangle$ are all recorded, and $\mathsf{op}_m^{(2)}(b_0, b_1) = b_2$, then send message $\langle \mathsf{ECOT\text{-}PROOF}, sid, ssid, cid_0, cid_1, cid_2, P_i, \mathcal{P}, m\rangle$ to all parties in $\mathcal{P}$ and $\mathcal{S}$; otherwise do nothing.

- **Transfer phase:** When receiving from P_i a message $\langle \mathsf{ecot\text{-}transfer}, sid, cid, cid_0, cid_1, tcid, P_j, \mathcal{P}\rangle$, if the following three tuples, $\langle cid_0, P_i, \mathcal{P}, b_0\rangle$, $\langle cid_1, P_i, \mathcal{P}, b_1\rangle$, and $\langle tcid, P_j, \mathcal{P}, b_t\rangle$ are all recorded, send message $\langle \mathsf{ECOT\text{-}DATA}, sid, cid, P_i, P_j, cid_0, cid_1, tcid, b_{b_t}\rangle$ to P_j, record tuple $\langle cid, P_j, \mathcal{P}, b_{b_t}\rangle$, and send message $\langle \mathsf{ECOT\text{-}RECEIPT}, sid, cid, P_i, P_j, cid_0, cid_1, tcid, \mathcal{P}\rangle$ to all parties in $\mathcal{P}$, P_i, and $\mathcal{S}$; and ignore all future messages of the form $\langle \mathsf{ecot\text{-}commit}, sid, cid, \mathcal{P}, *\rangle$ from P_j and all future messages of the form $\langle \mathsf{ecot\text{-}transfer}, sid, cid, *, *, *, P_j, \mathcal{P}\rangle$ from any party in $\mathcal{P}$. Otherwise, do nothing.

- **Open phase:** When receiving from P_i a message $\langle \mathsf{ecot\text{-}open}, sid, cid, P_i, \mathcal{P}\rangle$, if the tuple $\langle cid, P_i, \mathcal{P}, b\rangle$ is recorded, send message $\langle \mathsf{ECOT\text{-}DATA}, sid, cid, P_i, \mathcal{P}, b\rangle$ to $\mathcal{S}$ and all parties in $\mathcal{P}$; otherwise, do nothing.

Figure 7: *The multiparty extended committed oblivious transfer functionality*

It is straightforward to extend the UCECOT protocol to the multiple-party case. One simply replaces the $\mathcal{F}_{\mathrm{ZK}}$ functionalities by the $\mathcal{F}_{\mathrm{mZK}}$ functionalities and replaces the point-to-point messages by broadcast messages. We denote the extended protocol by $\mathsf{UCmECOT}$, and we have the following theorem.

Theorem 6.2 *Under the DDH assumption, protocol* $\mathsf{UCmECOT}$ *securely realizes the* $\mathcal{F}_{\mathrm{mECOT}}$ *ideal functionality in the* $(\mathcal{F}_{\mathrm{CRS}}, \mathcal{F}_{\mathrm{BC}}, \hat{\mathcal{F}}_{\mathrm{mZK}}^{R_{\mathsf{OR\text{-}DL}}}, \hat{\mathcal{F}}_{\mathrm{mZK}}^{R_{\mathsf{OR\text{-}N\text{-}DL}}}, \hat{\mathcal{F}}_{\mathrm{mZK}}^{R_{\mathsf{PEREP}}}, \hat{\mathcal{F}}_{\mathrm{mZK}}^{R_{\mathsf{OR\text{-}PEREP}}})$*-hybrid model against adaptive adversaries, assuming erasing.*

6.3 Multi-party joint gate evaluation

We extend the joint gate evaluation functionality to the multi-party case. Functionality $\mathcal{F}_{\mathrm{mJGE}}$ is shown in Figure 8. The only changes with respect to the two-party case are that the receipts are sent to all participating parties, and that all parties have to agree for the opening to take place.

Functionality $\mathcal{F}_{\mathrm{mJGE}}$

$\mathcal{F}_{\mathrm{JGE}}$ proceeds as follows, running with parties $P_1, ..., P_n$, and adversary $\mathcal{S}$.

- **Commit phase:** When receiving from P_i a message $\langle \mathsf{commit}, sid, cid, \mathcal{P}, b\rangle$, if $P_i \in \mathcal{P}$, then record $\langle cid, \mathcal{P}, b\rangle$, send message $\langle \mathsf{RECEIPT}, sid, cid\rangle$ to all parties in $\mathcal{P}$ and $\mathcal{S}$, and ignore all further messages of the form $\langle \mathsf{commit}, sid, cid, \mathcal{P}, *\rangle$ and $\langle \mathsf{eval}, sid, cid, *, *, \mathcal{P}, *\rangle$ from any party in $\mathcal{P}$.

- **Evaluate phase:** When receiving from P_i a message $\langle \mathsf{eval}, sid, cid, cid_0, cid_1, \mathcal{P}, m\rangle$, if $P_i \in \mathcal{P}$ and both $\langle cid_0, \mathcal{P}, b_0\rangle$ and $\langle cid_1, \mathcal{P}, b_1\rangle$ are recorded, then compute $b = \mathsf{op}_m^{(2)}(b_0, b_1)$, record $\langle cid, \mathcal{P}, b\rangle$, send message $\langle \mathsf{EVAL\text{-}RECEIPT}, sid, cid, cid_0, cid_1, P_i, \mathcal{P}, m\rangle$ to all parties in $\mathcal{P}$ and $\mathcal{S}$, and ignore all further messages of the form $\langle \mathsf{commit}, sid, cid, \mathcal{P}, *\rangle$ and $\langle \mathsf{eval}, sid, cid, *, *, \mathcal{P}, *\rangle$ from any party in $\mathcal{P}$; otherwise, do nothing.

- **Open phase:** When receiving from P_i a message $\langle \mathsf{open}, sid, cid, P_j\rangle$, if $P_i \in \mathcal{P}$, $P_j \in \mathcal{P}$, and the tuple $\langle cid, \mathcal{P}, b\rangle$ is recorded, then record tuple $\langle \mathsf{openreq}, sid, cid, P_j\rangle$. When a tuple $\langle \mathsf{openreq}, sid, cid, P_i, P_j\rangle$ is recorded for every $P_j \in \mathcal{P}$, then send message $\langle \mathsf{DATA}, sid, cid, b\rangle$ to P_i.

Figure 8: *The multi-party joint gate evaluation functionality*

In fact, we only need a "weakened" version of the $\mathcal{F}_{\mathrm{mJGE}}$ functionality for general multi-party computation. The weakened version, denoted by $\mathcal{F}_{\mathrm{wmJGE}}$, has the additional constraint that only XOR and AND operations are allowed in the evaluation phase. It is obvious that since $\{\mathrm{XOR}, \mathrm{AND}\}$ is a complete set of Boolean operations, $\mathcal{F}_{\mathrm{wmJGE}}$ is powerful enough to realize any multi-party functionality.

As in the two-party case, we also need an extension of the $\mathcal{F}_{\mathrm{mECOT}}$ functionality, denoted by $\mathcal{F}^4_{\mathrm{mECOT}}$, that performs $\binom{4}{1}$-oblivious transfer and proves relations among four bits. We only state the following theorem and omit the details.

Theorem 6.3 *There exists an efficient protocol that securely realizes the $\mathcal{F}^4_{\mathrm{mECOT}}$ functionality in the $\mathcal{F}_{\mathrm{mECOT}}$-hybrid model against malicious, adaptive adversaries.*

Next, we briefly sketch a protocol UCmJGE that securely realizes $\mathcal{F}_{\mathrm{wmJGE}}$ in the $\mathcal{F}^4_{\mathrm{mECOT}}$-hybrid model. This protocol is essentially a multi-party extension to the UCJGE protocol. In UCmJGE, a bit b is now shared among all participating parties: party P_i has bit b_i such that $\sum_{i=1}^n b_i = b \mod 2$. In the following description, we omit some details in the protocol such as the format of the messages and the identifiers of the bits. These details should be clear from the context.

Commit phase: For party P_i to commit to a bit b, it generates random bits $b_1, b_2, ..., b_{n-1} \stackrel{R}{\leftarrow} \{0,1\}$ and $b_n \leftarrow b \oplus b_1 \oplus \cdots \oplus b_{n-1}$. Then P_i commits to b_i through $\mathcal{F}^4_{\mathrm{mECOT}}$, sends bits b_j to party P_j for all $j \neq i$. Then each P_j commits to b_j through the $\mathcal{F}^4_{\mathrm{mECOT}}$ and opens it to P_i immediately.

Evaluate phase: Assume the two bits to be computed are a and b, and P_i holds bits a_i, b_i as their shares. Naturally we have $a = \sum a_i \bmod 2$ and $b = \sum b_i \bmod 2$. We assume the result bit is c and each party should hold a share c_i at the end of this phase. We consider two cases according to the operation performed.

XOR: To compute the XOR of bit a and b, each party simply computes $c_i = a_i \oplus b_i$. No messages are needed.

AND: To compute the AND of bit a and b, we follow the approach in [26, 12]. Observe that AND is the multiplication modulo 2, and we have the following equality.

$$\left(\sum_{i=1}^{n} a_i\right) \cdot \left(\sum_{i=1}^{n} b_i\right) = n \cdot \sum_{i=1}^{n} a_i \cdot b_i + \sum_{1 \le i < j \le n} (a_i + a_j) \cdot (b_i + b_j) \bmod 2$$

(see [26] for the justification of this equality). Therefore, each party P_i can compute $n \cdot a_i \cdot b_i$ by itself, and each pair P_i and P_j can jointly compute $(a_i + a_j) \cdot (b_i + b_j)$ as in the two-party case, by invoking multiple transfer phases of the $\mathcal{F}^4_{\mathrm{mECOT}}$ functionality.

Open phase: To open a bit b, shared as $b = \sum_{i=1}^{n} b_i$, to party P_i, every P_j opens its share b_j through $\mathcal{F}^4_{\mathrm{mECOT}}$. Then P_i sums up all the shares to obtain b.

Abort: In case any party aborts and/or deviates from the protocol, all parties abort the protocol.

Theorem 6.4 *Protocol* UCmJGE *securely realizes functionality $\mathcal{F}_{\mathrm{mJGE}}$ in the $\mathcal{F}^4_{\mathrm{mECOT}}$-hybrid model against malicious, adaptive adversaries.*

The proof is very similar to that of Theorem 4.2. Next, for any multi-party functionality $\mathcal{F}$, we construct a protocol $\Pi_{\mathcal{F}}$ that securely realizes $\mathcal{F}$ in the $\mathcal{F}_{\mathrm{mJGE}}$-hybrid model. The construction is almost identical to the two-party case, except that since we assume the circuit computing $\mathcal{F}$ consists of AND and XOR gates, instead of NAND gates, the gate-by-gate evaluation will invoke the $\mathcal{F}_{\mathrm{mJGE}}$ functionality with different encodings of functions.

Theorem 6.5 *Let $\mathcal{F}$ be a multi-party adaptively well-formed functionality. Then $\Pi_{\mathcal{F}}$ securely realizes $\mathcal{F}$ in the $\mathcal{F}_{\mathrm{mJGE}}$-hybrid model, in the presence of malicious adaptive adversaries.*

Acknowledgements

The authors thank the anonymous reviewers for TCC'04 for the many valuable comments.

References

[1] N. Barić and B. Pfitzmann. Collision-free accumulators and fail-stop signature schemes without trees. In *Advances in Cryptology–Eurocrypt '97*, pp.480–494, 1997.

[2] D. Beaver. Foundations of Secure Interactive Computing. In *Advances in Cryptology – CRYPTO '91*, pp. 377–391, 1991.

[3] D. Beaver. Secure Multiparty Protocols and Zero-Knowledge Proof Systems Tolerating a Faulty Minority. In *Journal of Cryptology* 4(2), pp. 75–122, 1991.

[4] M. Bellare and S. Micali. Non-interactive oblivious transfer and applications. *Advances in Cryptology—CRYPTO '89*, pp. 547–557, volume 435 of *Lecture Notes in Computer Science*, Springer-Verlag, 1990.

[5] D. Boneh. The decision Diffie-Hellman problem. In *Proceedings of the Third Algorithmic Number Theory Symposium* (LNCS 1423), pp. 48–63, 1998.

[6] F. Boudot. Efficient Proofs that a Committed Number Lies in an Interval. In *EUROCRYPT 2000*, pp. 431–444, 2000.

[7] F. Boudot and J. Traoré. Efficient Publicly Verifiable Secret Sharing Schemes with Fast or Delayed Recovery. In *Information and Communication Security, Second International Conference, ICICS'99*, pp. 87–102.

[8] J. Camenisch and M. Michels. Separability and efficiency for generic group signature schemes. In *Advances in Cryptology—CRYPTO '99*, volume 1666 of *Lecture Notes in Computer Science*, pages 414–430, Springer-Verlag, 1999.

[9] R. Canetti. Security and Composition of Multiparty Cryptographic Protocols. In *Journal of Cryptology* 13(1), pp. 143–202, 2000.

[10] R. Canetti. Universally composable security: A new paradigm for cryptographic protocols. In *42nd FOCS*, pp. 136–145, 2001.

[11] R. Canetti and M. Fischlin. Universally composable commitments. In *CRYPTO 2001* (LNCS 2139), pp. 19–40, 2001.

[12] R. Canetti, Y. Lindell, R. Ostrovsky and A. Sahai. Universally composable two-party computation. In *STOC '02*, pp. 494–503, 2002. Full version in *Cryptology ePrint Archive*, Report 2002/140.

[13] R. Canetti and T. Rabin. Universal Composition with Joint State In *Cryptology ePrint Archive*, Report 2002/047, `http://eprint.iacr.org/`, 2002.

[14] R. Cramer. Modular Design of Secure yet Practical Cryptographic Protocols. Ph.D. Thesis. CWI and University of Amsterdam, 1997.

[15] R. Cramer, I. Damgård, and J. Nielsen. Multiparty Computation from Threshold Homomorphic Encryption In *Advances in Cryptology–EuroCrypt '01*, volume 2045 of *Lecture Notes in Computer Science*, pp. 280–300, Springer-Verlag, 2001.

[16] R. Cramer, I. Damgård, and B. Schoenmakers. Proofs of partial knowledge and simplified design of witness hiding protocols. In *Advances in Cryptology–CRYPTO '94* (LNCS 839), pages 174–187, 1994.

[17] R. Cramer and V. Shoup. Signature scheme based on the strong RSA assumption. In *ACM Transactions on Information and System Security (ACM TISSEC)* 3(3):161-185, 2000.

[18] C. Crépeau. Verifiable disclosure of secrets and applications. In *Eurocrypt '89*, LNCS 434, pp 181 – 191, 1990.

[19] C. Crépeau, J. van de Graaf, and A. Tapp. Committed oblivious transfer and private multi-party computation. In *Advances in Cryptology—CRYPTO '95*, volume 963 of *Lecture Notes in Computer Science*, pages 110–123. Springer-Verlag, 27–31 Aug. 1995.

[20] I. Damgård and J. Nielsen. Perfect hiding and perfect binding universally composable commitment schemes with constant expansion factor. In *CRYPTO 2002* (LNCS 2442), pp. 581–596, 2002. A later version in *Cryptology ePrint Archive*, Report 2001/091. `http://eprint.iacr.org/`, 2001.

[21] I. Damgård, and J. Nielsen. Universally Composable Efficient Multiparty Computation from Threshold Homomorphic Encryption. In *Advances in Cryptology–CRYPTO '03*, 2003.

[22] S. Even, O. Goldreich, and A. lempel, A Randomized Protocol for Signing Contracts. In *Communications of the ACM*, 28:637-647 (1985).

[23] E. Fujisaki and T. Okamoto. Statistical zero knowledge protocols to prove modular polynomial relations. In *Advances in Cryptology–CRYPTO '97*, pp. 16-30, 1997.

[24] J. Garay and P. MacKenzie. Concurrent Oblivious Transfer. In *FOCS 2000*, pp. 314-324, Redondo Beach, CA, November 2000.

[25] J. Garay, P. MacKenzie and K. Yang. Strengthening Zero-Knowledge Protocols using Signatures. In *Advances in Cryptology–Eurocrypt 2003*, Warsaw, Poland, LNCS 2656, pp.177-194, 2003. Full version available from *Cryptology ePrint Archive*, Report 2003/037, `http://eprint.iacr.org/2003/037`, 2003.

[26] O. Goldreich. Secure Multi-Party Computation (Working Draft, Version 1.2), March 2000. Available from `http://www.wisdom.weizmann.ac.il/~oded/pp.html`.

[27] O. Goldreich, S. Micali and A. Wigderson. How to play any mental game or a completeness theorem for protocols with honest majority. In *19th ACM Symposium on the Theory of Computing*, pp. 218–229, 1987.

[28] S. Goldwasser and L. Levin. Fair computation of general functions in presence of immoral majority, In *Advances in Cryptology-CRYPTO '90*, pp. 77-93, Springer-Verlag, 1991.

[29] S. Goldwasser and Y. Lindell. Secure Computation Without Agreement. In 16th *DISC*, Springer-Verlag (LNCS 2508), pages 17-32, 2002.

[30] S. Goldwasser and L. Levin. Fair computation of general functions in presence of immoral majority, In *Advances in Cryptology–CRYPTO '90*, pp. 77-93, Springer-Verlag, 1991.

[31] P. MacKenzie and M. Reiter. Two-Party Generation of DSA Signatures. In *Advances in Cryptology–CRYPTO '01*, pp 137–154, 2001.

[32] S. Micali and P. Rogaway. Secure Computation (Abstract). In *Advances in Cryptology – CRYPTO '91*, pp. 392–404, 2001.

[33] T. Osamoto and S. Uchiyama. A new public-key cryptosystem as secure as factoring. In *Advances in Cryptology–Eurocrypt '98*, pp.380–318, 1998.

[34] P. Paillier. Public-key cryptosystems based on composite degree residue classes. In *Advances in Cryptology–Eurocrypt '99*, pp.223–238, 1999.

[35] T. P. Pedersen. Non-Interactive and Information-Theoretic Secure Verifiable Secret Sharing. In *CRYPTO 1991*, pp. 129–140, 1991.

[36] B. Pfitzmann, M. Schunter, and M. Waidner. Provably Secure Certified Mail. In *IBM Research Report RZ 3207 (#93253) 02/14/00*, IBM Research Division, Zürich, August 2000.

[37] B. Pfitzmann and M. Waidner. Composition and integrity preservation of secure reactive systems. In *ACM Conference on Computer and Communications Security (CCS 2000)*, pp. 245–254, 2000.

[38] B. Pfitzmann and M. Waidner. A Model for Asynchronous Reactive Systems and its Application to Secure Message Transmission. In *IEEE Symposium on Security and Privacy*, pp. 184–200, 2001.

[39] P. Rogaway. The round complexity of secure protocols. Ph.D. Thesis, MIT, June 1991.

[40] A. Yao. Protocols for Secure Computation. In *FOCS 1982*, pages 160–164, 1982.

A Number-Theoretic Assumptions

We review some of the number-theoretic assumptions used in this paper.

The Strong RSA assumption. The Strong RSA assumption is a generalization of the standard RSA assumption which (informally) states that given an RSA modulus N and an exponent e, it is computationally infeasible to find the e-th root of a random x. Informally, the strong-RSA assumption states that it is infeasible to find an *arbitrary* non-trivial root of a random x.

More formally, we say that p is a *safe prime* if both p and $(p-1)/2$ are prime. Then let $\mathsf{RSA\text{-}Gen}(1^k)$ be a probabilistic polynomial-time algorithm that generates two random $k/2$-bit safe primes p and q, and outputs $N \leftarrow pq$.

Assumption A.1 (Strong-RSA) *For any non-uniform probabilistic polynomial-size circuit $\mathcal{A}$, the following probability is negligible in k:*

$$\Pr[N \leftarrow \mathsf{RSA\text{-}Gen}(1^k); x \leftarrow \mathbb{Z}_N^*; (y,e) \leftarrow \mathcal{A}(1^k, x, N) : y^e \equiv x \bmod N \wedge e \geq 2]$$

The Strong RSA assumption we introduced by Barić and Pfitzmann [1], and has been used in several applications (see [23, 17]). It is a stronger assumption than the "standard" RSA assumption, yet no method is known for breaking it other than factoring N.

The Paillier cryptosystem and the Decision Composite Residuosity assumption. The Paillier encryption scheme [34] is defined as follows, where $\lambda(N)$ is the Carmichael function of N, and L is a function that takes input elements from the set $\{u < N^2 | u \equiv 1 \bmod N\}$ and returns $L(u) = \frac{u-1}{N}$. This definition differs from that in [34] only in that we define the message space for public key $pk = \langle N, g \rangle$ as $[-(N-1)/2, (N-1)/2]$ (versus $\mathbb{Z}_N$ in [34]), and we restrict h to be $1+N$. The security of this cryptosystem relies on the *Decision Composite Residuosity Assumption*, DCRA.

For key generation, choose random $k/2$-bit primes p, q, set $N = pq$, and set $h \leftarrow 1 + N$. The public key is $\langle N, h \rangle$ and the private key is $\langle N, h, \lambda(N) \rangle$. To encrypt a message m with public key $\langle N, h \rangle$, select a random $\alpha \in \mathbb{Z}_N^*$ and compute $c \leftarrow g^m \alpha^N \bmod N^2$. To decrypt a ciphertext c with secret key $\langle N, h, \lambda(N) \rangle$, compute $m = \frac{L(c^{\lambda(N)} \bmod N^2)}{L(g^{\lambda(N)} \bmod N^2)} \bmod N$, and the decryption is m if $m \leq (N-1)/2$, and otherwise the decryption is $m - N$. Paillier [34] shows that both $c^{\lambda(N)} \bmod N^2$ and $g^{\lambda(N)} \bmod N^2$ are elements of the form $(1+N)^d \equiv_{N^2} 1 + dN$, and thus the L function can be easily computed for decryption.

B Constructing Ω-protocols

We present the detailed constructions of some Ω-protocols in this paper. We first describe the detailed construction of the Ω-protocol for proving knowledge of discrete logarithm by Garay *et al.* [25]. We present an efficient Ω-protocol for R in Figure 9. The common reference string consists of two parts: (1) a Paillier public key $pk = \langle N, h \rangle$ where N is an RSA modulus and $h \in \mathbb{Z}_{N^2}^*$ with $N | \mathsf{order}(h)$, and (2) another RSA modulus with 2 generators $\langle \tilde{N}, h_1, h_2 \rangle$. The prover and the verifier share a common input y, while the prover also knows x, such that $g^x = y$. In the first message, the prover sends an encryption of x using the Paillier encryption key pk. Then a Σ-protocol is used to prove that the plain-text in the Paillier encryption is indeed the discrete log of y. A technical difficulty is that the discrete logarithm and the Paillier encryption work in different moduli. To overcome this, a known technique of adding a commitment to x using two generators (h_1, h_2) over a third modulus $\tilde{N}$ of unknown factorization [6, 8, 23, 31, 25] was used.

Since the techniques used in the protocol is rather standard, we omit the proof of security for simplicity.

Next, we present an Ω-protocol for proving knowledge of partial equality of representation, $\mathsf{PEREP}(x_0, g_0, g_1, x_1, g_2, g_3)$. See Figure 10.

prover		verifier
$\alpha, \beta \xleftarrow{R} \mathbb{Z}_N^*$		
$r \xleftarrow{R} \mathbb{Z}_{q^3}$		
$a \xleftarrow{R} \mathbb{Z}_{q\tilde{N}}$		
$b \xleftarrow{R} \mathbb{Z}_{q^3\tilde{N}}$		
$y' \leftarrow g^r \bmod p$		
$e \leftarrow h^x \alpha^N \bmod N^2$		
$e' \leftarrow h^r \beta^N \bmod N^2$		
$s \leftarrow (h_1)^x (h_2)^a \bmod \tilde{N}$		
$s' \leftarrow (h_1)^r (h_2)^b \bmod \tilde{N}$		
	$\xrightarrow{y', e, e', s, s'}$	
		$c \xleftarrow{R} \mathbb{Z}_q$
	$\xleftarrow{c}$	
$z_1 \leftarrow cx + r$		
$z_2 \leftarrow \alpha^c \beta \bmod N$		
$z_3 \leftarrow ca + b$		
	$\xrightarrow{z_1, z_2, z_3}$	
		$z_1 \stackrel{?}{\in} \mathbb{Z}_{q^3}$
		$y^c y' \stackrel{?}{\equiv} g^{z_1} \bmod p$
		$e^c e' \stackrel{?}{\equiv} h^{z_1} (z_2)^N \bmod N^2$
		$s^c s' \stackrel{?}{\equiv} (h_1)^{z_1} h_2^{z_2} \bmod \tilde{N}$

Figure 9: Ω-protocol for the discrete log relation $R_{\mathsf{DL}} = \{((y, g), x) : y \equiv g^x \bmod p\}$. Common reference string is a Paillier public key and a Strong RSA modulus along with two generators $((N, h), (\tilde{N}, h_1, h_2))$.

$$
\begin{array}{rcl}
\textsf{prover} & & \textsf{verifier} \\
 & & \\
v_0, v_1, v_2, v_0', v_1' v_2' \leftarrow Z_N^* & & \\
\beta_0, \beta_1, \beta_2 \leftarrow \mathbb{Z}_{q^3} & & \\
r \leftarrow \mathbb{Z}_{q\tilde{N}} & & \\
r' \leftarrow \mathbb{Z}_{q^3\tilde{N}} & & \\
w_0 := g^{\alpha_0} \cdot v_0^N \bmod N^2 & & \\
w_1 := g^{\alpha_1} \cdot v_1^N \bmod N^2 & & \\
w_2 := g^{\alpha_2} \cdot v_2^N \bmod N^2 & & \\
w_0' := g^{\beta_0} \cdot (v_0')^N \bmod N^2 & & \\
w_1' := g^{\beta_1} \cdot (v_1')^N \bmod N^2 & & \\
w_2' := g^{\beta_2} \cdot (v_2')^N \bmod N^2 & & \\
y_0 := g_0^{\beta_0} \cdot g_1^{\beta_1} \bmod p & & \\
y_1 := g_2^{\beta_0} \cdot g_3^{\beta_2} \bmod p & & \\
z := h_0^{\alpha_0} \cdot h_1^{\alpha_1} \cdot h_2^{\alpha_2} \cdot h_3^{r} \bmod \tilde{N} & & \\
z' := h_0^{\beta_0} \cdot h_1^{\beta_1} \cdot h_2^{\beta_2} \cdot h_3^{r'} \bmod \tilde{N} & & \\
 & \xrightarrow{w_0, w_1, w_2, w_0', w_1', w_2', y_0, y_1, z, z'} & \\
 & & c \leftarrow \mathbb{Z}_q \\
 & \xleftarrow{c} & \\
s_0 := c \cdot \alpha_0 + \beta_0 & & \\
s_1 := c \cdot \alpha_1 + \beta_1 & & \\
s_2 := c \cdot \alpha_2 + \beta_2 & & \\
s_3 := c \cdot r + r' & & \\
t_0 := v_0^c \cdot v_0' \bmod N^2 & & \\
t_1 := v_1^c \cdot v_1' \bmod N^2 & & \\
t_2 := v_2^c \cdot v_2' \bmod N^2 & & \\
 & \xrightarrow{s_0, s_1, s_2, t_0, t_1, t_2} & \\
 & & s_0 \overset{?}{\in} \mathbb{Z}_{q^3} \\
 & & s_1 \overset{?}{\in} \mathbb{Z}_{q^3} \\
 & & s_2 \overset{?}{\in} \mathbb{Z}_{q^3} \\
 & & s_3 \overset{?}{\in} \mathbb{Z}_{q^3\tilde{N}} \\
 & & g^{s_0} \cdot t_0^N \overset{?}{\equiv} w_0^c \cdot w_0' \bmod N^2 \\
 & & g^{s_1} \cdot t_1^N \overset{?}{\equiv} w_1^c \cdot w_1' \bmod N^2 \\
 & & g^{s_2} \cdot t_2^N \overset{?}{\equiv} w_2^c \cdot w_2' \bmod N^2 \\
 & & g_0^{s_0} \cdot g_1^{s_1} \overset{?}{\equiv} x_0^c \cdot y_0 \bmod p \\
 & & g_2^{s_0} \cdot g_3^{s_2} \overset{?}{\equiv} x_1^c \cdot y_1 \bmod p \\
 & & h_0^{s_0} \cdot h_1^{s_1} \cdot h_2^{s_2} \cdot h_3^{s_3} \overset{?}{\equiv} z^c \cdot z' \bmod \tilde{N}
\end{array}
$$

Figure 10: Ω-protocol for the partial equality of representation relation $R_{\mathsf{PEREP}} = \{((x_0, g_0, g_1, x_1, g_2, g_3), (\alpha_0, \alpha_1, \alpha_2)) \mid x_0 \equiv g_0^{\alpha_0} \cdot g_1^{\alpha_1} \bmod p \;\wedge\; x_1 \equiv g_2^{\alpha_0} \cdot g_3^{\alpha_2} \bmod p\}$. Common reference string is a Paillier public key and a Strong RSA modulus along with four generators $((N, g), (\tilde{N}, h_0, h_1, h_2, h_3))$.