Efficient Covert Two-Party Computation

Stanisław Jarecki

University of California, Irvine
sjarecki@uci.edu

Abstract. Covert computation (of general functions) strengthens the notion of secure computation so that the computation hides not only everything about the participants' inputs, except for what is revealed by the function output, but it also hides the very fact that the computation is taking place, by ensuring that protocol participants are indistinguishable from random beacons, except when the function output explicitly reveals the fact that a computation took place. General covert computation protocols proposed before have non-constant round complexity [19, 4] and their efficiency is orders of magnitude away from non-covert secure computation. Furthermore, [10] showed that constant-round covert computation of non-trivial functionalities with black-box simulation is impossible in the plain model.

However, the lower-bound of [10] does not disallow constant-round covert computation given some relaxation in the computation model. Indeed, we propose the first constant-round protocol for *Covert Two-Party Computation* (2PC) of general functions secure against malicious adversaries in the Common Reference String (CRS) model. Our protocol is a covert variant of a well-known paradigm in standard, i.e. non-covert, secure 2PC, using cut-and-choose technique over O(security parameter) copies of Yao's garbled circuit. Remarkably, the proposed protocol is efficiency-wise in the same ballpark as existing *non-covert* constant-round 2PC protocols secure against malicious players. As an added benefit, the protocol remains covert under concurrent composition.

An essential tool in the protocol is a concurrently secure covert zero-knowledge and simulation-sound Conditional KEM (CKEM) for arithmetic languages in prime-order groups. We show how to realize covert zero-knowledge and simulation-sound CKEM's for such languages in the Random Oracle Model, based on the covert CKEM's of [13], and in the CRS model, based on the Implicit Zero-Knowledge arguments of Benhamouda et al. [2]. The ROM-based covert CKEM's match the cost of known ROM-based NIZK's for the same languages, while the CRS-model CKEM's are 2-4 times more expensive.

1 Introduction

Covert computation addresses a security concern which is unusual for cryptography, namely how to hide the very fact that a (secure) protocol executes. Such hiding of a protocol instance is possible if the public channels connecting the communicating parties are *steganographic*, i.e. if they have some entropy. A protocol is covert if its messages can be efficiently injected into such channels in a way that the resulting communication cannot be distinguished from the a priori behavior of these channels. A standard example is a *random channel*, a.k.a. a *random beacon*, which can be implemented e.g. using protocol nonces, padding bits, time stamps, and various other communication (and cryptographic!) mechanisms which exhibit inherent (pseudo)entropy. Given a random channel, if protocol messages are indistinguishable from random bitstrings, such messages can be injected into the channel, and the protocol counterparty can interpret the information received on the channel as a protocol message. The participants must agree on the time they use such channels to run a protocol, so they know which bits to interpret as protocol messages, but this can be public information because if the protocol is covert then the exchanged messages cannot be distinguished from the a priori behavior of the random channel.[1]

Covert computation was formalized for the two-party setting by Von Ahn, Hopper and Langford in [19], and then generalized to the multi-party setting (and re-formulated) by Chandran et al. in [4], as a protocol that lets the participants securely compute the desired functionality on their joint inputs, with the additional property that each participant cannot distinguish the others from "random beacons", i.e. entities that send out random bitstrings of fixed length instead of prescribed protocol messages, unless the function output determines that it should be revealed. In other words, in covert computation the computed function outputs an additional *reveal* bit: If this bit is zero then each participant remains indistinguishable from a random beacon to the others, but if the reveal bit is one then the participants learn the function output, and in particular, learn that a computation took place, i.e. that they were interacting not with random beacons but with counterparties executing the same protocol (whose inputs into this protocol, moreover, made the reveal bit in the function output equal to 1).

Q & A on Covert Computation. *Motivation:* Who would care to compute a function while hiding this very fact from other potential protocol participants? One example given by Chandran et al. [4] is a company C that observes worrisome activity on its network: If C could determine that other companies observe similar activity, this information could help all of them fight against a hacking attack, but the very fact of engaging in such protocol reveals that company C observes some worrisome activity, and C might have business reasons to hide this. In another example, a military or intelligence agent A who detects some suspicious information could determine if similar information is detected by other agents, but no one, not even an active participant interacting with A can detect that A attempts to execute this protocol unless they have the information that "matches" A's knowledge. In general, covert computation can be used for any

[1] Works on steganographic *communication* [11] imply that random messages can be embedded into any non-uniform random channel with sufficient entropy, hence in particular once we know how to communicate and/or compute covertly assuming uniform random channels then we can also carry this communication/computation over any (non-uniform) steganographic channel with sufficient entropy.

form of authentication whose participants want to remain undetectable except to counter-parties whose inputs (certificates, permissions, passwords, environmental observations) matches their authentication policy. For example, if two spies want to authenticate one another in a foreign country, they would like to be able to do so in a way that prevents anyone from detecting that this authentication instance is taking place. If the spies authenticated each other using covert computation, the only way their presence can be detected is by an active attacker who owns authentication tokens that match their authentication policy.

Random Channels and Synchronization: If protocol parties were not communicating by default, it would always be possible to detect a protocol party by just observing that it sends out messages, and observing their number and size should normally suffice to conclude what protocol this party follows. Indeed, this is why we assume that protocol participants have default access to channels with some inherent entropy. A network entity cannot hide the fact that it sends out messages, but if the normal communication they emit exhibits some entropy (e.g. in protocol nonces, timing, padding, audio and video signals) then this entropy can be used to create a steganographic channel, i.e. convert any entropic channel into a random channel, and these random channels can carry covert MPC protocol messages. But even if agent A in the above example is connected by a random channel to potential counterparties in a covert MPC protocol, how would she know when to start the protocol? One answer is that the protocol start should be a public convention, e.g. the first message after time 12:00 on each day could contain a protocol instance. A participating party would interpret such messages as protocol messages, and if their sender was not engaging in the protocol, or it engaged but the reveal bit in the computation output is 0, this protocol instance would simply fail.

Covert MPC vs. Steganography: Covert MPC does not trivially follow by using steganography [11] to establish covert communication channels between potential protocol participants and running standard MPC over them. First, covert channels require prior key distribution which is not always possible, e.g. in the general authentication application above. Second, even if potential participants did have pre-shared keys, they might still want to hide whether or not they engage in a given protocol instance.

Covert MPC vs. Secure MPC: Secure computation is considered to be a blueprint for every security task: Whatever security property we want some network interaction to achieve, we can abstract it as a secure computation of an idealized functionality, and we can achieve it by MPC for this functionality. However, secure computation does leak one additional "bit" of information, namely it is not designed to hide whether or not some entity engages in the protocol, which in many applications (see examples above) is a very essential information. To give an extreme example: If CIA is the only organization whose agents follow some secure authentication protocol then a man-in-the-middle attacker will be trivially able to identify (potential) CIA agents. A *covert computation* strengthens secure computation to hide this one remaining bit, and allows protocol participation to be undetectable even to active protocol participants except (and this "escape clause" seems unavoidable) if the function output itself determines that the outputs, and hence also the fact of protocol participation, should be revealed to the participants. What we show in this paper is that, in contrast to the initial batch of works on covert computation [19, 4, 10] we discuss below, which showed only feasibility results for this notion, this strengthening of two-party secure computation to covertness can be achieved at costs which are comparable to those born by known standard, i.e. non-covert, constant-round secure computation protocols based on Yao's garbled circuits.[2] Moreover, in the process we show general tools for covert enforcement of honest protocol execution which can be re-used (and further improved) in efficient covert protocols for specific functions of interest.

Previous Works on Covert Computation. Von Ahn et al. [19] proposed the first covert two-party computation (2PC) protocol. Their protocol performed $O(\tau)$ repetitions, for τ a security parameter, of Yao's garbled circuit evaluation (with the circuit extended to compute an additional hash function), but this protocol guaranteed only secrecy against malicious participants, and not output correctness. The covert multi-party computation protocol of [4] realized a covert computation functionality against malicious participants (and in particular guaranteed output correctness), but it also used $O(\tau)$ rounds and its efficiency was several orders of magnitude away from known non-covert MPC protocols: Each party was covertly proving that it followed a GMW MPC protocol on committed input by casting it as an instance of a Hamiltonian Cycle problem, and that proof internally used Yao's garbled circuits for checking correctness of committed values. Moreover, Goyal and Jain subsequently showed that [10] the non-constant round protocol is necessary for achieving covert computation (with black-box simulation) against malicious adversaries, at least in the standard MPC model, i.e., without access to trusted parameters or public keys.

On the plus side, some constant-round covert protocols secure against malicious adversaries are known as well: Jarecki [13] showed a covert Authenticated Key Exchange (AKE) with $O(1)$ rounds and public key operations, but this protocol satisfied a game-based AKE definition, and in particular, it was not a covert secure computation of any function. Assuming a Random Oracle Model (ROM), Cho et al. [5] exhibited practical constant-round covert computation protocols, secure against active adversaries, for two non-trivial functionalities, namely for string equality and set intersection. (Moreover, Cho et al. strengthen the definition of covert computation of [4] to include concurrent self-composition, and we adopt this stronger notion of covert computation in this work.) However, in addition to relying on ROM, their constructions are custom-made for functionalities dealing with equality or set-membership checking, and it is not clear how they can be extended to computation of general functions.

Our Result: Efficient Covert Concurrent 2PC. This leaves a natural open question whether general two-party functions can be computed covertly by a

[2] In the work concurrent to ours, Couteau [6] shows essentially that the exact same statement applies also to MPC based on arithmetic circuits.

constant-round protocol, or even, better, by a protocol whose assumptions, the security guarantees, and efficiency, are all comparable to those of the currently known constant-round standard, i.e. non-covert, secure 2PC protocols. We answer all these questions affirmatively with a construction of a constant-round protocol for covert 2PC of general functions, secure against malicious adversaries. (We note that a corresponding result for GMW-style MPC on arithmetic circuits was developed concurrently to our work by Couteau [6].) Our protocol follows the well-known paradigm for non-covert constant-round secure 2PC, using the cut-and-choose technique over $O(\tau)$ copies of Yao's garbled circuit protocol, and its efficiency is in the same ballpark as the non-covert secure 2PC protocols. Concretely, for a function with n_I-bit inputs and n_O-bit outputs computable by a Boolean circuit with c gates, the proposed protocol requires 5 rounds, $O(n_I\tau + n_O\tau^2)$ exponentiations and $O(c\tau + n_O\tau^2)$ symmetric cipher operations.[3] The protocol works in the Common Reference String (CRS) model, it maintains covertness under concurrent composition, and it is secure under the Decisional Diffie-Hellman (DDH) assumption.

Enabling Tool: Covert Counterpart to Zero-Knowledge Proofs. Assuming random channels, covert *communication* is essentially as easy as secure communication: Since block ciphers are assumed to be pseudorandom functions many standard encryption (or even authenticated encryption) modes have ciphertexts which in addition to protecting the plaintext are also indistinguishable from random bitstrings. Several known public-key encryption schemes, e.g. Cramer-Shoup encryption [7], also have ciphertexts that are indistinguishable from a tuple of random group elements (assuming DDH), and random group elements, e.g. in a prime-order subgroup of modular residues, are easy to encode as random bitstrings. Assuming honest-but-curious participants, Von Ahn et al. [19] showed that general covert *computation* is also not more difficult than general secure computation: Given block ciphers whose outputs are indistinguishable from random strings, Yao's garbled circuit construction can be adjusted so that a garbled circuit for c-gates looks like $4c$ random ciphertexts even to the evaluator (except for whatever is revealed by the output, but that can be set to a random string if the "reveal bit" in the output evaluates to 0), and because ElGamal encryption is covert under DDH, an honest-but-curious secure OT based on ElGamal, like the Naor-Pinkas OT [17], also remains covert.

However, it is difficult to achieve covert 2PC/MPC protocols secure against *malicious* adversaries, and this is because of the lack of efficient covert counterparts to standard mechanisms for enforcing honest behavior in protocols. For example, the two chief tools used to enforce honest behavior in Yao's garbled circuit protocol are (1) Zero-Knowledge (ZK) proofs, e.g. to show that the sender entered consistent inputs into the Oblivious Transfer (OT) and into multiple garbled circuit instances, and (2) opening a commitment to show that the

[3] Both expressions can be reduced by factor of τ/κ where κ is the statistical security parameter, but the result would meet a reduced security goal, namely, with probability $2^{-\kappa}$ the adversary could make an honest party compute an adversarially chosen function on the joint inputs.

committed value is correctly formed, which is the basis of the general "cut-and-choose" method for enforcing honest behavior. Either of these tools would violate covertness because both are publicly verifiable, and in particular, the first party who sends a ZK proof or an opening of a commitment becomes distinguishable from random noise by the counterparty.

The enabling tool we use to enforce honest behavior in protocol are efficient realizations of a covert *Conditional Key Encapsulation Mechanism* (CKEM) for a wide class of discrete-log-based languages, including statements that the Cramer-Shoup encryption [7] ciphertexts or a response in the Oblivious Transfer (OT) of Aiello et al. [1], are computed correctly, or that an encrypted value is a bit, or that a commitment decommits to a given plaintext (with the decommitment as the prover's witness). A CKEM can be thought of as a language-based envelope, a KEM-version of Conditional Oblivious Transfer [8], or an interactive counterpart of Smooth Projective Hash Functions (SPHF): A CKEM for language L is a protocol which allows a sender S with input x to transmit a random key K to receiver R with input (x, w) if and only if w is a witness for x in L. A *covert* CKEM additionally assures that an interaction with either S or R is indistinguishable from an interaction with a random beacon. In particular, even knowing the witness w for x a malicious receiver cannot distinguish $\mathsf{S}(x)$ from a random beacon: It computes a key K which is the same as the key computed by the sender, but since the key is random, it still does not know if the sender was a real party or a source of randomness. Covert CKEMs can thus provide a covert counterpart to zero-knowledge proofs: Instead of asking party A in a 2PC protocol to prove that it creates its messages correctly, which makes A's presence publicly verifiable, parties B and A run a covert CKEM for the same language as resp. S and R, and then use key K to (covertly) encrypt subsequent protocol messages: If A's messages were not formed correctly, A will not be able to derive B's key K, which in particular will make all subsequent messages of B indistinguishable from random in A's view.

A Covert CKEM for language L was introduced as "Zero-Knowledge Send" by Chandran et al. [4] and achieved for general languages, but their construction reduced L to an NP-complete problem (Hamiltonian Cycle) and used garbled circuit evaluation at each step of a ZK proof for this problem. By contrast, we are looking for covert CKEM's for a class of languages, which we call *Linear Map Image* languages, which are discrete-log-based languages that have practical (HV)ZK proofs (and efficient SPHF systems), and we want the CKEM costs to be in a similar ballpark as the cost of these proofs. A natural starting point for a CKEM for a discrete-log-based language are well-known efficient SPHF schemes for such languages: If B uses an SPHF system on x and defines key K as the SPHF hash value then A can derive the same key K using its witness w for x in L. However, SPHF by itself cannot replace a zero-knowledge proof in a larger protocol, because it does not offer a way for the simulator playing the role of A and simulating A's message x, e.g. with random noise, to recover B's key K and continue the simulation on A's behalf in subsequent protocol rounds. Zero-knowledge works because the simulator can send $x \notin \mathsf{L}$ on A's behalf and then simulate A's ZK proof as if x was in L. By contrast, if $x \notin \mathsf{L}$ then an SPHF on x will hide B's key K in an information-theoretic sense.

This insufficiency of SPHF's to replace zero-knowledge proofs in MPC protocols was recognized by Benhamouda et al. [2] who added two essential properties to SPHF's for discrete-log-based languages: First, they added (concurrent) zero-knowledge, i.e. the ability for the simulator in position of the global trapdoor in the CRS to derive S's key K even on the wrong statement $x \notin \mathsf{L}$. Secondly, they added simulation-soundness, i.e. the assurance that a cheating receiver cannot recover S's key K for a protocol instance executing on a wrong statement $x \notin \mathsf{L}$ even if the adversary concurrently engages with the simulator who recovers keys corresponding to multiple protocol instances running on any *other* wrong statements $x' \notin \mathsf{L}$. These two properties, simulation-soundness and (concurrent) zero-knowledge are needed of zero-knowledge in the standard compiler from (concurrent) 2PC secure against honest-but-curious adversaries to (concurrent) 2PC secure malicious parties. Benhamouda et al. called this class of protocols (concurrent) simulation-sound *Implicit Zero-Knowledge* (IZK) Arguments, and they showed an efficient construction for such IZK's for a wide class of discrete-log-based languages (including the those listed above). The notion of IZK defined by [2] does not ask for the IZK protocol to be covert, because the goal of the Implicit ZK Arguments of [2] was reduction of round complexity in the honest-but-curious to malicious-security protocol compilation: The IZK schemes they show take only 2 rounds in the CRS model under DDH, compared to 3 rounds for zero-knowledge proofs of comparable efficiency for the same language class. Here we extend the IZK notion of [2] to include covertness, and we call the resulting notion a (concurrent and simulation-sound) *Covert CKEM*.

We offer two constructions of 2-round covert CKEM's, for two classes of languages commonly occurring in cryptographic protocols. First, we characterize the class of languages which includes all languages used in our covert 2PC protocol as *Linear Map Image* (LMI) languages, i.e. languages whose statements can be represented as pair (C, M) of a vector C and a matrix M of group elements s.t. C belongs to a range of a linear map $f_M(x) = M \cdot x$ defined by M. To construct covert CKEM's for such languages, we first show a construction in ROM for all languages with Σ-protocols, and we observe that all LMI languages in prime-order group have a Σ-protocol. This construction is a version of a compiler shown in [13], which converted a Σ-protocol into a weaker form of covert CKEM, which in particular did not include existence of a simulator. Here we show that the ROM-based version of that construction does allow for efficient simulation, and that it moreover preserves simulation soundness. The resulting ROM-based covert CKEM's are relevant in practice because they add only a small overhead to the cost of well-known ROM-based NIZK's for languages which commonly occur in standard, i.e. *non-covert*, secure protocols. Secondly, we show that a simple modification of the CRS-model IZK construction of [2] for any LMI language, also forms a covert CKEM for LMI languages characterized by matrix M which is full row rank. This is commonly the case, including all LMI languages used in our Covert 2PC protocol.

Organization. In Section 2 we introduce covertness-related notation. In Section 3 we specify our notion of concurrent covert 2PC for arbitrary functions. In Section 4 we go over several covert counterparts to standard protocol building blocks, including covert CCA encryption, commitment, Oblivious Transfer, and (HbC-secure) circuit garbling. In Section 5 we define *covert* Conditional KEM (CKEM) as a covert counterpart to concurrent and simulation-sound zero-knowledge proofs, and we introduce the class of so-called LMI languages for which we need covert CKEM's in the next section. In Section 6 we describe our concurrent covert 2PC protocol for covert computation of arbitrary functions. Finally, in Section 7 we exhibit constructions of efficient covert CKEM's for the LMI languages used in this covert 2PC protocol.

Revision Notes. The previous version of this paper (available on eprint) shows that the CKEM of [2] without any modifications meet relaxed conditions of covertness defined in that version. Here instead we show that a simple modification of this CKEM satisfies a more general covertness notion. The current version contains also more extensive overview of the design choices in the covert 2PC protocol in Section 6.

2 Preliminaries

Notation. If a, b are bitstrings then $|a|$ is the length of a, $a|b$ is the concatenation of stings a and b, and $a[i]$ is the i-th bit of a. If n is an integer then $[n] = \{1, ..., n\}$. We write $y \leftarrow \mathsf{P}(x)$ when y is an output of a (randomized) procedure P on input x, and $y \leftarrow \mathsf{S}$ when y is sampled from uniform distribution over set S. We write $y \in \mathsf{P}(x)$ if there is randomness r s.t. $\mathsf{P}(x; r)$ outputs y. We say $(a, b) \leftarrow [A(x), B(y)]$ if a, b are the local outputs of algorithms resp. A, B interacting on local inputs resp. x, y. If L is a language in NP then $\mathcal{R}[\mathsf{L}]$ is a relation s.t. $(x, w) \in \mathcal{R}[\mathsf{L}]$ if w is an efficiently verifiable witness for $x \in \mathsf{L}$. If $\mathsf{ks} = \{(k^{i,0}, k^{i,1})\}_{i\in[n],b\in\{0,1\}}$, i.e. ks is a sequence of n pairs of bitstrings, and $x \in \{0,1\}^n$, then $\mathsf{ks}[{:}\, x]$ denotes a *selection* of bitstrings from the n pairs in ks according to the n bits of x, namely $\mathsf{ks}[{:}\, x] = \{k^{i,x[i]}\}_{i\in[n]}$.

We call two-party protocol (A, B) *regular* if the number of rounds and length of all messages is a function of the security parameter, and not the parties' inputs. If P is an interactive algorithm in a regular two-party protocol then $\mathsf{P}^{\$(\tau)}$ denotes a random beacon corresponding to P, which sends random bitstrings of the same length as P's messages in every protocol round. If P is an interactive algorithm then $P_{\&\mathsf{Out}}(x)$ is a wrapper which runs $P(x)$ and includes P's final local output in its last message. For any algorithm Setup and oracles P_0, P_1 we say that $\{\mathcal{A}^{P_0(x_0)}(z)\} \approx \{\mathcal{A}^{P_1(x_1)}(z)\}$ *for* $(x_0, x_1, z) \leftarrow \mathsf{Setup}(1^\tau)$ if for every efficient $\mathcal{A}$ quantity $|p^0_{\mathcal{A}} - p^1_{\mathcal{A}}|$ is negligible where $p^b_{\mathcal{A}} = \Pr[1 \leftarrow \mathcal{A}(z)^{P_b(x_b)} \,|\, (x_0, x_1, z) \leftarrow \mathsf{Setup}(1^\tau)]$, where the probability goes over the coins of Setup, $\mathcal{A}$, and P_b.

Covert Encodings. In the protocols in this paper all communicated values are fixed-size bitstrings, or integers from some integer range $\mathbb{Z}_n$, or elements of a prime-order group G. In the latter two cases what is sent on the wire are the values themselves but their covert encodings. A covert encoding of domain D is a randomized function $\mathsf{EC} : D \rightarrow \{0,1\}^{p(\tau)}$ defined for some polynomial p, s.t. a random variable $\{\mathsf{EC}(a;\mathsf{r})\}$, induced by a random a in D and a random r, is statistically close to a random bitstring of length $p(\tau)$. Moreover, there must exist a corresponding decoding procedure DC s.t. $\mathsf{DC}(b) = a$ for all b output by $\mathsf{EC}(a)$. For example, for an integer range domain $D = \mathbb{Z}_n$, encoding $\mathsf{EC}(a)$ can pick $r \leftarrow \mathbb{Z}_R$ for $R = \lceil 2^{|n|+\tau}/n \rceil$ and output $a + n{\cdot}r$ (over integers), while $\mathsf{DC}(v)$ outputs $v \bmod n$. If the domain D is a subgroup G of order p in a group of residues modulo q s.t. $q = p \cdot t + 1$ for $gcd(p,t) = 1$, then $\mathsf{EC}(a)$ can pick $b \leftarrow \mathbb{Z}_q$, compute $v = (a \cdot (b)^p) \bmod q$, and then apply the encoding for integer range $\mathbb{Z}_q$ to v. The corresponding decoding first decodes v and then outputs $w^s \bmod q$ for $w = v^t \bmod q$ and $s = t^{-1} \bmod p$.

3 Concurrent Covert Two-Party Computation

We provide the definition of concurrent covert computation of two-party functions, which is a close variant of the definition which appeared recently in [5]. Intuitively, the differences between the covert computation of a two-party functionality F and the secure computation for F is that (1) F's inputs and outputs are extended to include a special sign $\perp$ designating non-participation; (2) F is restricted to output a non-participation symbol $\perp$ to each party if the input of either party is $\perp$; and (3) the real-world protocol of either party on the non-participation input $\perp$ is fixed as a "random beacon", i.e. a protocol which sends out random bitstrings of fixed length independently of the messages it receives.

The definition of concurrent covert computation of [5], which we recall (and refine) below, follows the definition of stand-alone (i.e. "single-shot") covert computation given by Chandran et al. [4], here restricted to the two-party case. The definition casts this notion in the framework of universal composability (UC) by Canetti [3], but the composability guarantee it implies is restricted to concurrent self-composition because it guarantees only self-composability of covert computation for *functions*, and not for general reactive functionalities as in the case of UC definition [3]. The reason for this restriction is two-fold: First, concurrent covert computation for arbitrary efficiently computable functions already provides a significant upgrade over the "single-shot" covert computation notion of [4], and achieving it efficiently presents sufficient technical challenges that justify focusing on this restricted notion. Secondly, composing functionally distinct covert protocols poses conceptual challenges: Consider a protocol Π implemented by a protocol Π_1 which runs Π_2 as a subroutine, and note that the outputs of subroutine Π_2 can reveal the participation of an honest party in Π before Π completes. Here we focus on concurrent composition of covert computation of two-party function, and leave development of a framework for fully composable covert computation for future work.

Ideal and Real Models. The definition of the *ideal model* is the UC analogue of the ideal model of Chandran et al. [4], except that composability guarantees

On input $(\mathsf{InputA}, \mathsf{sid}, B, x)$ **from party** A**:**
Record $(\mathsf{InputA}, \mathsf{sid}, A, B, x)$ and send $(\mathsf{InputA}, \mathsf{sid}, A, B)$ to $\mathcal{A}^*$.

On input $(\mathsf{InputB}, \mathsf{sid}, A, y)$ **from party** B**:**
Record $(\mathsf{InputB}, \mathsf{sid}, A, B, y)$ and send $(\mathsf{InputB}, \mathsf{sid}, A, B)$ to $\mathcal{A}^*$.

Given records $(\mathsf{InputA}, \mathsf{sid}, A, B, x)$ and $(\mathsf{InputB}, \mathsf{sid}, A, B, y)$ compute

$$(z, v) \leftarrow \begin{cases} (\perp, \perp) & \text{if} \quad x = \perp \ \vee \ y = \perp \ \vee \ g(x, y) = 0 \\ f(x, y) & \text{otherwise} \end{cases}$$

and record $(\mathsf{Output}, \mathsf{sid}, A, z)$ and $(\mathsf{Output}, \mathsf{sid}, B, v)$.
If A is corrupt, send $(\mathsf{Output}, \mathsf{sid}, z)$ to A.
If B is corrupt, send $(\mathsf{Output}, \mathsf{sid}, v)$ to B.

On input $(\mathsf{Output}, \mathsf{sid}, P, \mathsf{release?})$ **from** $\mathcal{A}^*$**:**
Retrieve record $(\mathsf{Output}, \mathsf{sid}, P, w)$ (ignore if record does not exist).
If $\mathsf{release?} = \mathsf{T}$ then send $(\mathsf{Output}, \mathsf{sid}, w)$ to P.
If $\mathsf{release?} = \mathsf{F}$ then send $(\mathsf{Output}, \mathsf{sid}, \perp)$ to P.

Fig. 1. Covert 2-Party Function Computation Functionality $\mathsf{F}_{\mathsf{C}(f,g)}$

are restricted to self-composition. Covert computation is defined by functionality $\mathsf{F}_{\mathsf{C}(f,g)}$ shown in Figure 1, where f, g are functions defined on pairs of bitstrings. We note that f and g can be randomized functions, in which case functionality $\mathsf{F}_{\mathsf{C}(f,g)}$ picks the randomness which is appended to input (x, y) before computing g and f. The ideal process involves functionality $\mathsf{F}_{\mathsf{C}(f,g)}$, an ideal process adversary $\mathcal{A}^*$, an environment $\mathcal{Z}$ with some auxiliary input z, and a set of dummy parties, any number of which can be (statically) corrupted. Each party can specify its input to some instance of $\mathsf{F}_{\mathsf{C}(f,g)}$, which is either a bitstring or a special symbol $\perp$ indicating that there is no party which will participate in a given role, e.g. a requester or responder in this protocol instance. The *real model* is exactly as in the standard UC security model, except that the protocol of each real-world uncorrupted party which runs on input $\perp$ is a-priori specified as a random beacon protocol, i.e. such party sends out random bitstrings of lengths appropriate for a given protocol round.

Let $\mathsf{Ideal}_{\mathsf{F},\mathcal{A}^*,\mathcal{Z}}(\tau, z, r)$ denote the output of environment $\mathcal{Z}$ after interacting in the ideal world with adversary $\mathcal{A}^*$ and functionality $\mathsf{F} = \mathsf{F}_{\mathsf{C}(f,g)}$, on security parameter τ, auxiliary input z, and random input $\mathsf{r} = (\mathsf{r}_{\mathcal{Z}}, \mathsf{r}_{\mathcal{A}^*}, \mathsf{r}_{\mathsf{F}})$, as described above. Let $\mathsf{Ideal}_{\mathsf{F},\mathcal{A}^*,\mathcal{Z}}\ (\tau, z)$ be the random variable $\mathsf{Ideal}_{\mathsf{F},\mathcal{A}^*,\mathcal{Z}}(\tau, z; \mathsf{r})$ when r is uniformly chosen. We denote the distribution ensemble of variable $\mathsf{Ideal}_{\mathsf{F},\mathcal{A}^*,\mathcal{Z}}(\tau, z)$ by $\{\mathsf{Ideal}_{\mathsf{F},\mathcal{A}^*,\mathcal{Z}}(\tau, z)\}_{\tau \in \mathbb{N}; z \in \{0,1\}^*}$. In the corresponding way we define $\mathsf{Real}_{\Pi,\mathsf{Adv},\mathcal{Z}}(\tau, z; \mathsf{r})$ as the output of $\mathcal{Z}$ after interacting with a real-world adversary Adv and parties running protocol Π on security parameter τ, input z, and random tapes $\mathsf{r} = (\mathsf{r}_{\mathcal{Z}}, \mathsf{r}_{\mathsf{Adv}}, \mathsf{r}_A, \mathsf{r}_B)$. In parallel to the ideal model, we define the corresponding distribution ensemble $\{\mathsf{Real}_{\Pi,\mathsf{Adv},\mathsf{F}}(\tau, z)\}_{\tau \in \mathbb{N}; z \in \{0,1\}^*}$.

Definition 1. *Protocol Π realizes the concurrent two-party covert computation functionality $\mathsf{F} = \mathsf{F}_{\mathsf{C}(f,g)}$ if for any efficient adversary Adv there exists an efficient ideal-world adversary $\mathcal{A}^*$ such that for any efficient environment $\mathcal{Z}$,*

$$\{\mathsf{Ideal}_{\mathsf{F},\mathcal{A}^*,\mathcal{Z}}(\tau, z)\}_{\tau\in\mathbb{N};z\in\{0,1\}^*} \overset{c}{\approx} \{\mathsf{Real}_{\Pi,\mathsf{Adv},\mathsf{F}}(\tau, z)\}_{\tau\in\mathbb{N};z\in\{0,1\}^*}$$

Notes on Functionality $\mathsf{F}_{\mathsf{C}(f,g)}$. Functionality $\mathsf{F}_{\mathsf{C}(f,g)}$ in Figure 1 is realizable only assuming secure channels. Without secure channels the adversary could hijack a protocol session an honest player wants to execute with some intended counterparty. However, the secure channel assumption does not substantially change the complexity of the protocol problem because the intended counterparty can itself be corrupted and follow an adversarial protocol. The second point we want to make is that functionality $\mathsf{F}_{\mathsf{C}(f,g)}$ always delivers the output first to a corrupted party, whether it is party A or B, and if this output is not a non-participation symbol $\perp$ then in both cases the corrupted party can decide if the correct computation output should also be delivered to its (honest) counterparty or the honest counterparty's output will be modified to $\perp$. (Note that if an output of a corrupt party, say A, is $\perp$ then B's output is also $\perp$, hence it does not matter in this case whether the adversary sends $(\mathsf{Output}, \mathsf{T}, \mathsf{sid})$ or $(\mathsf{Output}, \mathsf{F}, \mathsf{sid})$.) Any constant-round protocol without a trusted party *must be unfair* in the sense that the party which speaks last gets its output but can prevent the delivery of an output to its counterparty. However, functionality $\mathsf{F}_{\mathsf{C}(f,g)}$ affords this unfair advantage to both the corrupt requester and the corrupt responder. Indeed, a concrete protocol Π_{COMP} presented in Section 6 which realizes this functionality allows the corrupt party A to learn its output z and stop B from learning anything about its output v (simply by aborting before sending its last message to B). However, this protocol also allows the corrupt party B to prevent party A from being able to decide if its output z (learned in step A2 in Figure 2) is an output of $f(x, y)$ or a random value induced from an interaction with a random beacon: Only B's final message can confirm which is the case for A, but a corrupt B can send this message incorrectly, in which case an honest A will dismiss the tentative value z it computed and output $\perp$ instead. We leave achieving $O(1)$-round covert protocols with one-sided fairness, or two-side fairness, e.g. using an off-line escrow authority, to future work.

4 Covert Protocol Building Blocks

CCA-Covert Public Key Encryption. Covertness of a public key encryption scheme in a Chosen-Ciphertext Attack, or *CCA covertness* for short, is a generalization of CCA security: Instead of requiring that ciphertexts of two challenge messages are indistinguishable from each other, we require that a ciphertext on any (single) challenge message is indistinguishable from a random bitstring, even in the presence of a decryption oracle. For technical reasons it suffices if interaction with the real PKE scheme is indistinguishable from an interaction with a simulator who not only replaces a challenge ciphertext with a random string but also might follow an alternative key generation and decryption strategy.

Formally, we call a (labeled) PKE scheme $(\mathsf{Kg}, \mathsf{E}, \mathsf{D})$ *CCA covert* if there exist polynomial n s.t. for any efficient algorithm $\mathcal{A}$, quantity $\mathsf{Adv}_{\mathcal{A}}(\tau) = |p^0_{\mathcal{A}}(\tau) - p^1_{\mathcal{A}}(\tau)|$ is negligible, where $p^b_{\mathcal{A}}(\tau)$ is the probability that $b' = 1$ in the following game: Generate $(\mathsf{pk}, \mathsf{sk}) \leftarrow \mathsf{Kg}(1^\tau)$, and let $\mathcal{A}^{\mathsf{D}(\mathsf{sk},\cdot,\cdot)}(\mathsf{pk})$ output an encryption challenge (m^*, ℓ^*). If $b = 1$ then set $\mathsf{ct}^* \leftarrow \mathsf{E}(\mathsf{pk}, m^*, \ell^*)$, and if $b = 0$ then pick ct^* as a random string of length $n(\tau)$. In either case set $b' \leftarrow \mathcal{A}^{\mathsf{D}(\mathsf{sk},\cdot,\cdot)}$, where oracle $\mathsf{D}(\mathsf{sk}, \cdot, \cdot)$ returns $\mathsf{D}(\mathsf{sk}, \mathsf{ct}, \ell)$ on any ciphertext,label pair s.t. $(\mathsf{ct}, \ell) \neq (\mathsf{ct}^*, \ell^*)$.

Notice that by transitivity of indistinguishability if PKE is CCA-covert then it is also CCA-secure. The other direction does not hold in general, but many known CCA-secure PKE's are nevertheless also CCA-covert, including RSA OAEP and Cramer-Shoup PKE [7]. We will use here the latter scheme because its arithmetic structure can be utilized for efficient covert OT (see below) and efficient covert CKEM's on associated languages (e.g. that a ciphertext encrypts a given plaintext). In Appendix A we show that the proof of CCA security of Cramer-Shoup PKE under the DDH assumption [7] can be extended to imply its CCA covertness. For notational convenience we assume that the key generation Kg picks the group setting (g, G, p) as a *deterministic* function of security parameter τ, and we restrict the message space to group G, since this is how we use this PKE in our covert 2PC protocol, but it can be extended to general message space using covert symmetric encryption.

Cramer-Shoup PKE (for message space G) works as follows: $\mathsf{Kg}(1^\tau)$ chooses generator g of group G of prime order p of appropriate length, sets a collision-resistant hash function H, picks $(x_1, x_2, y_1, y_2, z) \leftarrow (\mathbb{Z}_p^*)^5$, $(g_1, g_2) \leftarrow (G \backslash 1)^2$, sets $(c, d, h) \leftarrow (g_1^{x_1} g_2^{x_2}, g_1^{y_1} g_2^{y_2}, g_1^z)$, and outputs $\mathsf{sk} = ((g, G, p, \mathsf{H}), x_1, x_2, y_1, y_2, z)$ and $\mathsf{pk} = ((g, G, p, \mathsf{H}), g_1, g_2, c, d, h)$. Encryption $\mathsf{E}_{\mathsf{pk}}(\mathsf{m}, \ell)$, for $m \in G$, picks $r \leftarrow \mathbb{Z}_p$, sets $(u_1, u_2, e) \leftarrow (g_1^r, g_2^r, \mathsf{m} \cdot h^r)$, $\xi \leftarrow \mathsf{H}(\ell, u_1, u_2, e)$, $v \leftarrow (cd^\xi)^r$, and outputs $\mathsf{ct} = (u_1, u_2, e, v)$. Decryption $\mathsf{D}_{\mathsf{sk}}((u_1, u_2, e, v), \ell)$ re-computes ξ, and outputs $\mathsf{m} = e \cdot u_1^z$ if $v = u_1^{x_1 + \xi \cdot y_1} u_2^{x_2 + \xi \cdot y_2}$ and $\perp$ otherwise.

Covert Non-Malleable Commitments. It is well-known that CCA-secure PKE implements non-malleable commitment. However, to stress that sometimes no one (including the simulator) needs to decrypt, we define commitment $\mathsf{Com}_{\mathsf{pk}}(m)$ as a syntactic sugar for $\mathsf{E}_{\mathsf{pk}}(H(m))$ where H is a collision-resistant hash onto G, but we will pass on defining a notion of covert commitment, relying instead directly on the fact that $\mathsf{Com}_{\mathsf{pk}}(m)$ stands for $\mathsf{E}_{\mathsf{pk}}(H(m))$.

Covert Oblivious Transfer. Von Ahn et al. [19] used a covert version of Naor-Pinkas OT [17] for their covert 2PC secure against honest-but-curious adversaries. Here we will use a covert version of the OT of Aiello et al. [1] instead because it is compatible with CCA-covert Cramer-Shoup encryption and covert CKEM's of Section 7. Let E be the Cramer-Shoup encryption and let $\mathsf{pk} = ((g, G, p, \mathsf{H}), g_1, g_2, c, d, h)$. Define a 2-message OT scheme $(\mathsf{E}, \mathsf{OTrsp}, \mathsf{OTfin})$ on Rec's input b, Snd's input $m_0, m_1 \in G$, and a public label ℓ as follows:
(1) Rec's first message to Snd is $\mathsf{ct} = (u_1, u_2, e, v) = \mathsf{E}_{\mathsf{pk}}(g^b, \ell; \mathsf{r})$ for $\mathsf{r} \leftarrow \mathbb{Z}_p$.

(2) Snd's response computation, denoted $\mathsf{OTrsp}_{\mathsf{pk}}(\mathsf{ct}, m_0, m_1; \mathsf{r}')$, outputs $\mathsf{otr} = \{s_i, t_i\}_{i=0,1}$ for $(s_i, t_i) = (g_1^{\alpha_i} h^{\beta_i}, u_1^{\alpha_i}(e/g^i)^{\beta_i} m_i)$ and $\mathsf{r}' = \{\alpha_i, \beta_i\}_{i=0,1} \leftarrow \mathbb{Z}_p^4$.
(3) Rec's output computation, denoted $\mathsf{OTfin}_{\mathsf{pk}}(b, \mathsf{r}, \mathsf{otr})$, outputs $m = t_b \cdot (s_b)^{-\mathsf{r}}$.

The above OT is covert for random payloads in the following sense: First, the Rec's message is indistinguishable from random even on access to the decryption oracle $\mathsf{D}_{\mathsf{sk}}(\cdot,\cdot)$; Secondly, Snd's message is indistinguishable from random for payloads (m_0, m_1) random in G^2. (Note that if (m_0, m_1) were non-random then the Rec's output would suffice to distinguish OTrsp and $\mathsf{OTrsp}^{\$(\tau)}$.)

Covert Garbled Circuits. Von Ahn et al. [19] shows a covert version of Yao's garbling $\mathsf{GCgen}(f)$ for any $f : \{0,1\}^n \to \{0,1\}^m$. Procedure $\mathsf{GCgen}(f)$ outputs (1) a vector of input wire keys $\mathsf{ks} = \{k^{w,b}\}_{w \in [n], b \in \{0,1\}}$ where n is the bitlength of arguments to f, and (2) a vector gc of $4|C|$ covert symmetric encryption ciphertexts, where $|C|$ is the number of gates in a Boolean circuit for f. The corresponding evaluation procedure Eval_f outputs $f(x)$ given gc and $\mathsf{ks}[:x]) = \{k^{i,x[i]}\}_{i \in [n]}$, for $(\mathsf{gc}, \mathsf{ks})$ output by $\mathsf{GCgen}(f)$ and $x \in \{0,1\}^n$. Let $m' = 4|C|\tau + n\tau$. The notion of a *covert garbling* defined by [19] and satisfied by their variant of Yao's garbling scheme, is that for any function f, any distribution D over f's inputs, and any efficient algorithm $\mathcal{A}$, there is an efficient algorithm $\mathcal{A}^*$ s.t. $|\mathsf{Adv}_{\mathcal{A}} - \mathsf{Adv}_{\mathcal{A}^*}|$ is negligible, where:

$$\mathsf{Adv}_{\mathcal{A}} = |\Pr[1 \leftarrow A(\{\mathsf{gc}, \mathsf{ks}[:x]\})]_{x \leftarrow D, (\mathsf{gc},\mathsf{ks}) \leftarrow \mathsf{GCgen}(f)} - \Pr[1 \leftarrow \mathcal{A}(r)]_{r \leftarrow \{0,1\}^{m'}}|$$
$$\mathsf{Adv}_{\mathcal{A}^*} = |\Pr[1 \leftarrow \mathcal{A}^*(f(x))]_{x \leftarrow D} - \Pr[1 \leftarrow \mathcal{A}^*(r)]_{r \leftarrow \{0,1\}^m}|$$

In other words, for any function f and distribution D over its inputs, the garbled circuit for f together with the set of wire keys $\mathsf{ks}[:x]$ defined for input x sampled from D, are (in)distinguishable from a random string *to the same degree* as function outputs $f(x)$ for $x \leftarrow D$. In particular, if f and D are such that $\{f(x)\}_{x \leftarrow D}$ is indistinguishable from random, then so is $\{\mathsf{gc}, \mathsf{ks}[:x]\}_{(\mathsf{gc},\mathsf{ks}) \leftarrow \mathsf{GCgen}(f), x \leftarrow D}$.

SPHF's. We define a Smooth Projective Hash Function (SPHF) for language family L parametrized by π as a tuple $(\mathsf{PG}, \mathsf{KG}, \mathsf{Hash}, \mathsf{PHash})$ s.t. $\mathsf{PG}(1^\tau)$ generates parameters π and a trapdoor td which allows for efficient testing of membership in $\mathsf{L}(\pi)$, $\mathsf{KG}(\pi, x)$ generates key hk together with a *key projection* hp (here we use the Gennaro-Lindell notion of SPHF's [18] where the hash key hk can depend on the statement x, in contrast to the SPHF's by Katz-Vaikuntanathan [14]), and $\mathsf{Hash}(\pi, x, \mathsf{hk})$ and $\mathsf{PHash}(\pi, x, w, \mathsf{hp})$ generate hash values denoted H and projH, respectively. SPHF *correctness* requires that $\mathsf{Hash}(\pi, x, \mathsf{hk}) = \mathsf{PHash}(\pi, x, w, \mathsf{hp})$ for all τ, all (π, td) output by $\mathsf{PG}(1^\tau)$, all $(x, w) \in \mathcal{R}[\mathsf{L}(\pi)]$, and all $(\mathsf{hk}, \mathsf{hp})$ output by $\mathsf{KG}(\pi, x)$. Since in the SPHF's we use in this paper the hash values are elements of group G which is uniquely defined for a given security parameter τ by the Cramer-Shoup key generation procedure, we define SPHF *smoothness* as the requirement that $(\mathsf{hp}, \mathsf{Hash}(\pi, x, \mathsf{hk}))$ is distributed identically to (hp, r) for $r \leftarrow G$ and $(\mathsf{hk}, \mathsf{hp}) \leftarrow \mathsf{KG}(\pi, x)$, for all π and $x \notin \mathsf{L}(\pi)$. However, in our applications of SPHF's we rely on a stronger notion of *covert smoothness*, namely that for some constant c, for all π and $x \notin \mathsf{L}(\pi)$, pair $(\mathsf{hp}, \mathsf{Hash}(\pi, x, \mathsf{hk}))$ for $(\mathsf{hk}, \mathsf{hp}) \leftarrow \mathsf{KG}(\pi, x)$ is uniform in $G^c \times G$.

5 Covert Simulation-Sound Conditional KEM

A *Conditional Key Encapsulation Mechanism* (CKEM) was introduced in [13] as a generalization of SPHF to interactive protocols, and as a KEM version of Conditional Oblivious Transfer [9]. A CKEM for language L is a protocol between two parties, a sender S and a receiver R, on S's input a statement x_S and R's input a (statement,witness) pair (x_R, w_R). The outputs of S and R are respectively K_S and K_R s.t. K_S is a random string of τ bits, and $\mathsf{K}_R = \mathsf{K}_S$ if and only if $x_S = x_R$ and $(x_R, w_R) \in \mathcal{R}[\mathsf{L}]$. A CKEM scheme is an encryption counterpart of a zero-knowledge proof, where rather than having R use its witness w_R to prove to S that $x_S \in \mathsf{L}$, here R establishes a session key K with S if and only if w_R is a witness for x_S in L. Because of this relation to zero-knowledge proofs we will use proof-system terminology to define security properties of a CKEM scheme. In particular, we will refer to the CKEM security property that if $x \notin \mathsf{L}$ then no efficient algorithm can compute K output by $\mathsf{S}(x)$ as the *soundness* property.

Benhamouda et al. [2] considered a stronger notion of *trapdoor CKEM*, which they called *Implicit Zero-Knowledge*. Namely, they extended the CKEM notion by a CRS generation procedure which together with public parameters generates a trapdoor td that allows an efficient simulator algorithm to compute the session key K_S output by a sender $\mathsf{S}(x)$ for any x, including $x \notin \mathsf{L}$. The existence of such simulator makes CKEM into a more versatile protocol building block. For example, trapdoor CKEM implies a zero-knowledge proof for the same language, if R simply returns the key K_R to S who accepts iff $\mathsf{K}_R = \mathsf{K}_S$. Indeed, following [2], we refer to the property that the simulator computes the same key as the honest receiver in the case $x \in \mathsf{L}$ as the *zero-knowledge* property of a CKEM.

As in the case of zero-knowledge proofs, if multiple parties perform CKEM instances then it is useful to strengthen CKEM security properties to *simulation-soundness*, which requires that all instances executed by the corrupt players remain sound even in the presence of a simulator $\mathcal{S}$ who uses its trapdoor to simulate the instances performed on behalf of the honest players. Simulation-soundness is closely related to non-malleability: If $\mathcal{S}$ simulates a CKEM instance Π on $x \notin \mathsf{L}$ then an efficient adversary must be unable to use protocol instance Π executed by $\mathcal{S}$ to successfully complete another instance Π' of CKEM executed by a corrupt party for any $x' \notin \mathsf{L}$.

Below we formally define these security properties, by extending the definitions of Benhamouda et al. [2] to assure the following *covertness* properties:

(I) First, for *covert (simulation) soundness* we require not only that the sender's session key K is indistinguishable from a random string if the protocol proceeds on $x \notin \mathsf{L}$ (even in the presence of a multiple simulated protocol instances), but that all messages the sender sends *and* the session key it outputs, are together indistinguishable from a random beacon.

(IIa) Secondly, for *covert zero-knowledge* we require not only that an interaction with an honest receiver is indistinguishable from the interaction with a simulator but also that both are indistinguishable from random beacons.

(IIb) Lastly, since in the CKEM applications both protocol parties need to be covert, we extend the *covert zero-knowledge* property so that an interaction with the sender is indistinguishable from an interaction with a random beacon on *all* statements x, i.e. not just $x \notin \mathsf{L}$ as above.

Note that in items (IIa) and (IIb) which define covert zero-knowledge, we only require the covertness (i.e. the indistinguishability from random) for protocol *messages* sent by an honest party, and not the key (or rejection) defined by the honest party's *local output.* Otherwise, i.e. if the honest party's local output was included in the adversary's view, then an adversary running the counterparty's protocol on the same x (and a corresponding witness w), would always be able to distinguish such party from a random beacon because it would compute the same output key. This is in contrast to the notion of covert simulation soundness, i.e. item (I), which includes honest sender's local output in adversary's view, and can do so because it considers only CKEM executions on $x \notin \mathsf{L}$.

To distinguish between different CKEM sessions the CKEM syntax must also be amended by *labels*, which play similar role as labels in CCA encryption. Formally, a CKEM scheme for language family L is a tuple of algorithms $(\mathsf{PG}, \mathsf{TPG}, \mathsf{Snd}, \mathsf{Rec}, \mathsf{TRec})$ s.t. parameter generation $\mathsf{PG}(1^\tau)$ generates CRS parameter π, trapdoor parameter generation $\mathsf{TPG}(1^\tau)$ generates π together with the simulation trapdoor td, and sender Snd, receiver Rec, and trapdoor receiver TRec are interactive algorithms which run on local inputs respectively (π, x, ℓ), (π, x, ℓ, w), and $(\pi, x, \ell, \mathsf{td})$, and each of them outputs a session key K as its local output. CKEM correctness requires that for all ℓ:

$$\forall (x, w) \in \mathcal{R}[\mathsf{L}],\ \ [\mathsf{K}_S, \mathsf{K}_R] \leftarrow [\mathsf{Snd}(\pi, x, \ell), \mathsf{Rec}(\pi, x, \ell, w)] \Rightarrow \mathsf{K}_S = \mathsf{K}_R \qquad (1)$$

$$\forall x,\ \ [\mathsf{K}_S, \mathsf{K}_R] \leftarrow [\mathsf{Snd}(\pi, x, \ell), \mathsf{TRec}(\pi, x, \ell, \mathsf{td})] \Rightarrow \mathsf{K}_S = \mathsf{K}_R \qquad (2)$$

where (1) holds for all π generated by $\mathsf{PG}(1^\tau)$ and (2) holds for all (π, td) generated by $\mathsf{TPG}(1^\tau)$. Crucially, property (2) holds for all x, and not just for $x \in \mathsf{L}$.

Covert Zero-Knowledge. We say that a CKEM for language (family) L is *covert zero-knowledge* if the following properties hold:

1. *Setup Indistinguishability:* Parameters π generated by $\mathsf{PG}(1^\tau)$ and $\mathsf{TPG}(1^\tau)$ are computationally indistinguishable.
2. *Zero Knowledge:* For every efficient $\mathcal{A} = (\mathcal{A}_1, \mathcal{A}_2)$ we have

$$\{\mathcal{A}_2^{\mathsf{Rec}_{\&\mathsf{Out}}(\pi, x, \ell, w)}(\mathsf{st})\} \approx \{\mathcal{A}_2^{\mathsf{TRec}_{\&\mathsf{Out}}(\pi, x, \ell, \mathsf{td})}(\mathsf{st})\}$$

   for $(\pi, \mathsf{td}) \leftarrow \mathsf{TPG}(1^\tau)$ and $(\mathsf{st}, x, w, \ell) \leftarrow \mathcal{A}_1(\pi, \mathsf{td})$ s.t. $(x, w) \in \mathcal{R}[\mathsf{L}]$.[4]
3. *Trapdoor-Receiver Covertness:* For every efficient $\mathcal{A} = (\mathcal{A}_1, \mathcal{A}_2)$ we have

$$\{\mathcal{A}_2^{\mathsf{TRec}(\pi, x, \ell, \mathsf{td})}(\mathsf{st})\} \approx \{\mathcal{A}_2^{\mathsf{TRec}^{\$(\tau)}}(\mathsf{st})\}$$

   for $(\pi, \mathsf{td}) \leftarrow \mathsf{TPG}(1^\tau)$ and $(\mathsf{st}, x, \ell) \leftarrow \mathcal{A}_1(\pi, \mathsf{td})$.

[4] If $\mathcal{A}_1$ outputs $(x, w) \notin \mathcal{R}[\mathsf{L}]$ we override $\mathcal{A}_2$'s output by an arbitrary constant.

4. *Sender Covertness:* For every efficient $\mathcal{A} = (\mathcal{A}_1, \mathcal{A}_2)$ we have

$$\{\mathcal{A}_2^{\mathsf{Snd}(\pi,x,\ell)}(\mathsf{st})\} \approx \{\mathcal{A}_2^{\mathsf{Snd}^{\$(\tau)}}(\mathsf{st})\}$$

for $(\pi, \mathsf{td}) \leftarrow \mathsf{TPG}(1^\tau)$ and $(\mathsf{st}, x, \ell) \leftarrow \mathcal{A}_1(\pi, \mathsf{td})$.

Note that the *Zero-Knowledge* and *Trapdoor-Receiver Covertness* properties above imply the *Receiver Covertness* property, which says that for every efficient $\mathcal{A} = (\mathcal{A}_1, \mathcal{A}_2)$ we have

$$\{\mathcal{A}_2^{\mathsf{Rec}(\pi,x,\ell,w)}(\mathsf{st})\} \approx \{\mathcal{A}_2^{\mathsf{Rec}^{\$(\tau)}}(\mathsf{st})\}$$

for $(\pi, \mathsf{td}) \leftarrow \mathsf{TPG}(1^\tau)$ and $(\mathsf{st}, x, w, \ell) \leftarrow \mathcal{A}_1(\pi, \mathsf{td})$ s.t. $(x, w) \in \mathcal{R}[\mathsf{L}]$. This holds because an interaction with $\mathsf{Rec}(\pi, x, \ell, w)$ for any $(x, w) \in \mathcal{R}[\mathsf{L}]$ is, by Zero-Knowledge, indistinguishable from an interaction with $\mathsf{TRec}(\pi, x, \ell, \mathsf{td})$, which by Trapdoor-Receiver Covertness is indistinguishable from an interaction with $\mathsf{TRec}^{\$(\tau)}$, which is in turn identical to an interaction with $\mathsf{Rec}^{\$(\tau)}$, because Zero-Knowledge implies that Rec and TRec output equal-sized messages.

Discussion. CKEM zero-knowledge [2] says that an interaction with Rec on any $x \in \mathsf{L}$ followed by Rec's local output K_R, can be simulated by TRec without knowledge of the witness for x. Receiver and Trapdoor-Receiver *covertness* mean that, in addition, the adversary $\mathcal{A}$ who interacts with resp. Rec and TRec, but does not see their local outputs, cannot tell them from random beacons. In the case of TRec we ask for this to hold for any x and not only for $x \in \mathsf{L}$ because a simulator of a higher-level protocol will typically create incorrect statements and then it will simulate the Receiver algorithm on them. Note that we cannot include the output K_R of either Rec or TRec in $\mathcal{A}$'s view in the (trapdoor) receiver covertness game because $\mathcal{A}$ can compute it by running $\mathsf{Snd}(x)$. Sender covertness means that an interaction with the Snd is indistinguishable from an interaction with a random beacon for any x. Here too we cannot include Snd's local output K_S in $\mathcal{A}$'s view because if $(x, w) \in \mathcal{R}[\mathsf{L}]$ then $\mathcal{A}$ who holds w can compute it running $\mathsf{Rec}(x, w)$. Note that $\mathcal{A}$'s view includes both the public parameters π and the simulator's trapdoor td, which implies that all the properties will hold in the presence of multiple CKEM instances simulated by TRec using td.

Sender Covertness Relaxation. The standard-model CKEM of Section 7.2 does not assure Sender Covertness against an adversary who holds the simulation trapdoor td. However, it assures sender covertness for adversary who gets parameters π and has access to the Trapdoor Receiver oracle for any (x', ℓ') which differs from (x, ℓ) that defines the sender covertness challenge. We say that CKEM for L is *Sender Simulation-Covert* if it satisfies:

4'. *Sender Simulation-Covertness:* For every efficient $\mathcal{A} = (\mathcal{A}_1, \mathcal{A}_2)$ we have

$$\{\mathcal{A}_2^{\mathsf{Snd}(\pi,x,\ell),\mathsf{TRec}_{\mathsf{Block}(x,\ell)}(\mathsf{td},\cdot)}(\mathsf{st})\} \approx \{\mathcal{A}_2^{\mathsf{Snd}^{\$(\tau)},\mathsf{TRec}_{\mathsf{Block}(x,\ell)}(\mathsf{td},\cdot)}(\mathsf{st})\}$$

for $(\pi, \mathsf{td}) \leftarrow \mathsf{TPG}(1^\tau)$ and $(\mathsf{st}, x, \ell) \leftarrow \mathcal{A}_1^{\mathsf{TRec}(\mathsf{td},\cdot)}(\pi)$ s.t. $\mathsf{TRec}(\mathsf{td}, \cdot)$ was not queried on (x, ℓ).

We say that a CKEM is *Covert Zero-Knowledge with Sender Simulation-Covertness* if it satisfies all the Covert Zero-Knowledge properties except with the *Sender Covertness* property replaced by *Sender Simulation-Covertness.*

Covert Soundness and Simulation-Soundness. A CKEM is covert sound if interaction with Snd on $x \notin \mathsf{L}$ followed by Snd's local output K_S is indistinguishable from interaction with a random beacon. Recall that CKEM soundness [2] requires pseudorandomness of only Snd's output K_S on $x \notin \mathsf{L}$, while here we require it also of the transcript produced by Snd. Covert simulation-soundness requires that this holds even if the adversary has access to the Trapdoor-Receiver for any (x', ℓ') which differs from the pair (x, ℓ) that defines the soundness challenge. To that end we use notation $P_{\mathsf{Block}(x)}$ for a wrapper over (interactive) algorithm P which outputs $\perp$ on input $x' = x$ and runs $P(x')$ for $x' \neq x$:
CKEM is *Covert Sound* if for every efficient algorithm $\mathcal{A} = (\mathcal{A}_1, \mathcal{A}_2)$ we have:

$$\{\mathcal{A}_2^{\mathsf{Snd}_{\&\mathsf{Out}}(\pi,x,\ell)}(\mathsf{st})\} \approx \{\mathcal{A}_2^{\mathsf{Snd}_{\&\mathsf{Out}}^{\$(\tau)}}(\mathsf{st})\}$$

for $(\pi, \mathsf{td}) \leftarrow \mathsf{TPG}(1^\tau)$ and $(\mathsf{st}, x, \ell) \leftarrow \mathcal{A}_1(\pi)$ s.t. $x \notin \mathsf{L}$.
CKEM is *Covert Simulation-Sound* if for every efficient algorithm $\mathcal{A} = (\mathcal{A}_1, \mathcal{A}_2)$ we have:

$$\{\mathcal{A}_2^{\mathsf{Snd}_{\&\mathsf{Out}}(\pi,x,\ell), \mathsf{TRec}_{\mathsf{Block}(x,\ell)}(\mathsf{td},\cdot)}(\mathsf{st})\} \approx \{\mathcal{A}_2^{\mathsf{Snd}_{\&\mathsf{Out}}^{\$(\tau)}, \mathsf{TRec}_{\mathsf{Block}(x,\ell)}(\mathsf{td},\cdot)}(\mathsf{st})\}$$

for $(\pi, \mathsf{td}) \leftarrow \mathsf{TPG}(1^\tau)$ and $(\mathsf{st}, x, \ell) \leftarrow \mathcal{A}_1^{\mathsf{TRec}(\mathsf{td},\cdot)}(\pi)$ s.t. $x \notin \mathsf{L}$ and $\mathsf{TRec}(\mathsf{td}, \cdot)$ was not queried on (x, ℓ).

Note that sender simulation-covertness together with standard, i.e. non-covert, simulation-soundness, imply covert simulation-soundness of a CKEM:

Lemma 1. *If a CKEM scheme is simulation-sound [2] and sender simulation-covert, then it is also covert simulation-sound.*

Proof. Consider the simulation-soundness game where adversary $\mathcal{A}$ on input π for $(\pi, \mathsf{td}) \leftarrow \mathsf{TPG}(1^\tau)$ interacts with $\mathsf{TRec}(\mathsf{td}, \cdot)$, generates (x, ℓ) s.t. $x \notin \mathsf{L}$, and interacts with oracles $\mathsf{Snd}_{\&\mathsf{Out}}(\pi, x, \ell)$ and $\mathsf{TRec}_{\mathsf{Block}(x,\ell)}(\mathsf{td}, \cdot)$. The standard (i.e. non-covert) simulation soundness of this CKEM [2] implies that this game is indistinguishable from a modification in which key K_S output by $\mathsf{Snd}_{\&\mathsf{Out}}(\pi, x, \ell)$ is chosen at random. Once K_S is independently random, sender simulation-covertness, which holds for all x, implies that this game is indstinguishable from a modification where the *messages* sent by Snd are replaced by uniformly random strings. Since these two moves together replace oracle $\mathsf{Snd}_{\&\mathsf{Out}}(\pi, x, \ell)$ with $\mathsf{Snd}_{\&\mathsf{Out}}^{\$(\tau)}$, it follows that the CKEM is covert simulation-sound.

Linear Map Image Languages. The Covert 2PC protocol of Section 6 relies on covert zero-knowledge and simulation-sound CKEM's for what we call *Linear Map Image* languages. A linear map image language $\mathsf{LMI}_{n,m}$ for group G of prime order p contains pairs $(C, M) \in G^n \times G^{n \times m}$ s.t. there exists a vector

$w \in \mathbb{Z}_p^m$ s.t. $C = w \cdot M$, where the vector dot product denotes component-wise exponentiation, i.e. $[w_1, ..., w_m] \cdot [g_{i1}, ..., g_{im}] = \prod_{j=1}^{m} (g_{ij})^{w_j}$, In other words, $(C, M) \in \mathsf{LMI}_{n,m}$ if C is in the image of a linear map $f_M : \mathbb{Z}_p^m \to G^n$ defined as $f_M(w) = w \cdot M$. Using an additive notation for operations in group G we can equivalently say that $(C, M) \in \mathsf{LMI}_{n,m}$ if C is in the subspace of G^n spanned by the rows of M, which we denote $\mathsf{span}(M)$.

We extend the notion of a Linear Map Image language to a *class* of languages, denoted LMI, which includes all languages L for which there exist two efficiently computable functions $\phi : U_x \to (G \times G^{n\times m})$ and $\gamma : U_w \to \mathbb{Z}_p^m$ for some n, m, where U_x, U_w are the implicit universes of respectively statements in L and their witnesses, s.t. for all $(x, w) \in U_x \times U_w$, w is a witness for $x \in \mathsf{L}$ if and only if $\gamma(w)$ is a witness for $\phi(x) \in \mathsf{LMI}_{n,m}$. We will sometimes abuse notation by treating set $\{\phi(x)\}_{x\in\mathsf{L}}$, i.e. L mapped onto (some subset of) $\mathsf{LMI}_{n,m}$, replaceably with L itself. Observe that LMI is closed under conjunction, i.e.

$$[(C_1, M_1) \in \mathsf{LMI}_{n_1,m_1} \wedge (C_2, M_2) \in \mathsf{LMI}_{n_2,m_2}] \Leftrightarrow (C, M) \in \mathsf{LMI}_{n_1+n_2,m_1+m_2}$$

for $C = (C_1, C_2)$ and M formed by placing M_1 in the upper-left corner, M_2 in the lower-right corner, and all-one matrices in the remaining quadrants.

6 Covert Computation of General 2-Party Functions

We describe a protocol Π_{COMP}, in Figure 2, which realizes the concurrent 2-party covert computation functionality $\mathsf{F}_{\mathsf{C}(f,g)}$ in the CRS model. Protocol Π_{COMP} uses a covert variant of Yao's garbled circuit protocol [19] and a variant of the cut-and-choose approach (see e.g. [16]) to enforce security against malicious players. Note that a standard way of implementing a cut-and-choose involves protocol tools which are inherently non-covert: The first tool is that the circuit garbling party, B, sends commitments to n copies of the garbled circuit and then decommits a randomly chosen half of them, so that party A can verify that the opened circuits are formed correctly *and* that they were committed in B's first message. Clearly, if B sends a commitment followed by a decommitment, such decommitment can be verified publicly, at which point A would distinguish a protocol-participating party B from a random beacon regardless of the inputs which A or B enter into the computation. Secondly, a cut-and-choose protocol also use zero-knowledge proofs, e.g. to prove that the OT's are performed correctly, or that the keys opened for different circuit copies correspond to the same inputs, and zero-knowledge proofs are similarly inherently non-covert. We show that (concurrent and simulation-sound) covert CKEM's can be effectively used in both cases.

First, we use CKEM's in place of all zero-knowledge proofs, i.e. instead of party P_1 proving statement x to party P_2, we will have P_2 encrypt its future messages under a key derived by CKEM on statement x. By covert concurrent zero-knowledge, the simulator can derive the CKEM keys and simulate subsequent interaction of each protocol instance even if the statements it makes on behalf of honest players are incorrect (e.g. because the simulator does not know these players' real inputs). And by covert simulation-soundness, the CKEM's made by corrupted players are still sound, i.e. the CKEM keys created by the simulator on behalf of honest parties are indistinguishable from random unless the statement made by a corrupted player is correct. Moreover, CKEM messages sent by either party are indistinguishable from random strings.

Secondly, we replace a commit/decommit sequence with a covert commitment c, release of the committed plaintext m (which must be pseudorandom), and a covert CKEM performed on a statement that there exists decommitment d (the CKEM receiver's witness) s.t. d decommits c to m. We use a perfectly binding commitment so that the notion of language membership suffices to define this problem. Specifically, we implement the commitment scheme using covert Cramer-Shoup encryption, which plays two additional roles in the protocol construction: First, it assures non-malleability of each commitment/ciphertext. Secondly, it allows for straight-line extraction of player's inputs using the decryption keys as a trapdoor for the CRS which contains a Cramer-Soup encryption public key, which allows for security across concurrently executed protocol instances. Finally, the arithmetic structure of Cramer-Shoup encryption enables an efficient covert OT and efficient CKEM's on statements on committed/encrypted values.

These are the basic guidelines we follow, but assuring (concurrent) simulatability of each party in a secure two-party computation, doing so efficiently, and doing so in the *covert* setting where the protocol view of each party must look like a random beacon except (if the admission function evaluates to true) when the functionality reveals computation outputs, requires numerous adjustments we must make. We will attempt to explain most of these adjustments in the technical protocol overview below.

Defining the Garbled Circuit. We first explain how we use the covert garbling procedure GCgen of [19], see Section 4, to enable covert computation of functionality $\mathsf{F}_{\mathsf{C}(f,g)}$ assuming the simplified case where the party that garbles the circuit is *Honest but Curious.* Our basic design follows the standard Yao's two-party computation protocol but instantiates it using covert building blocks, i.e. party B will use *covert garbling* on a circuit that corresponds to functionality $\mathsf{F}_{\mathsf{C}(f,g)}$ (more on this below), it will send the garbled circuit together with the input wire keys to A, either directly, for wires corresponding to B's inputs, or via a *covert OT*, for wires corresponding to A's inputs, and A will evalute the garbled circuit to compute the output. This will work if the circuit garbled by B is appropriately chosen, as we explain here.

Step 1: Encoding B's Output. Note that functionality $\mathsf{F}_{\mathsf{C}(f,g)}$ has two-sided output, so we must include an encoding of B's output in the outputs of the garbled circuit in such a way that (1) this encoding looks random to A, and (2) A cannot modify this encoding to cause B to output any other value (except $\perp$). Let h be the two-sided output function at the heart of functionality $\mathsf{F}_{\mathsf{C}(f,g)}$, namely $h(x,y) = (z,v)$ s.t. $(z,v) = f(x,y)$ if $g(x,y) = 1$ and $(z,v) = (\perp,\perp)$ if $g(x,y) = 0$. Let n_x, n_y, n_z, n_v define resp. the length of input x of party A, input y of party B, output z of A, and output v of B. Let f_z, f_v satisfy $f(x,y) = (f_z(x,y), f_v(x,y))$. We will encode B's output in the outputs of the garbled circuit evaluated by A using the standard way for converting the garbled circuit technique into secure computation of a two-sided function: If $\mathsf{ts} = \{t_i^0, t_i^1\}_{i \in [n_v]}$ is the set of garbled circuit keys on the wires encoding B's output v in the garbled circuit for h, then the garbled circuit evaluator A computes $(z, \mathsf{ts}[:v])$ where $(z, v) = f(x, y)$ (if $g(x, y) = 1$). Note that $\mathsf{ts}[:v]$ is an encoding of v which satisfies the above two conditions, and if A sends it to B, B can decode it to v using set ts. Even though this encoding of B's output is implicit in the garbled circuit technique, we will add ts to the inputs and $\mathsf{ts}[:v]$ to the outputs of the function $f|_g$ we will garble, because this simplifies our notation and lets us use the covert garbling procedure GCgen of [19] as a black-box. In other words, we modify h to h' which on input $(x, (y, \mathsf{ts}))$ outputs $(z, \mathsf{ts}[:v])$ for $(z, v) = f(x, y)$ if $g(x, y) = 1$ and $(\perp, \perp)$ if $g(x, y) = 0$.

Step 2: Making $\perp$ Output Random. Next, note that if B garbles the circuit for h' then for any x, y s.t. $g(x, y) = 0$, party A on input x will distinguish between a random beacon and an honest party B which executes the protocol on input y. (This would not be a covert computation of $\mathsf{F}_{\mathsf{C}(f,g)}$ because $\mathsf{F}_{\mathsf{C}(f,g)}$ assures that $A(x)$ cannot distinguish $B(y)$ for y s.t. $(y \neq \perp \wedge g(x, y) = 0)$, from a random beacon $B(\perp)$.) This is because in the 2nd case the garbled circuit evaluates to $h'(x, y, \mathsf{ts}) = (\perp, \perp)$, and in the 1st case A will interpret random strings as a garbled circuit and the input wire keys, and those will evaluate to random outputs. To make the circuit evaluate to random outputs in the case $g(x, y) = 0$, we add $(n_z + n_v\tau)$-bit strings c and d to respectively A's and B's input, we define $h''((x, c), (y, d, \mathsf{ts}))$ as $(z, \mathsf{ts}[:v])$ for $(z, v) = f(x, y)$ if $g(x, y) = 1$, and as $c \oplus d$ if $g(x, y) = 0$, and we specify that both A and B set input random c and d strings into the computation. Note that if B is honest then setting the output to d instead of $c \oplus d$ in the $g(x, y) = 0$ case would suffice, but a malicious B would be then able to set A's output in the $g(x, y) = 0$ case, because A treats the first n_z bits of the circuit output as its local output z.

Step 3: Adding Simulation Trapdoor. Finally, we add a "simulator escape" input bit u to B's inputs, and the final circuit we garble, function $f|_g$ defined below, is like h'' but with condition $(g(x, y) \wedge u)$, in place of condition $g(x, y)$, for deciding between output $(z, \mathsf{ts}[:v])$ for $(z, v) = f(x, y)$ and output $c \oplus d$:

$$f|_g((x, c), (y, d, \mathsf{ts}, u)) = \begin{cases} (f_z(x, y)\ ,\ \mathsf{ts}[:v]) \quad \text{if} \quad g(x, y) = 1 \wedge u = 1 \\ \qquad \text{where } v = f_v(x, y) \text{ and } \mathsf{ts}[:v] = [t_1^{v[1]}, ..., t_{n_v}^{v[n_v]}] \\ c \oplus d \quad \text{otherwise,} \end{cases}$$

Here is how we will use this "escape bit" in the $g(x, y) = 1$ clause in the simulation: An honest real-world party B will set $u = 1$, in which case circuit $f|_g$ is identical to h''. However, a simulator $\mathcal{A}^*$ for the case of corrupt party A, will use the $u = 0$ escape clause to aid in its simulation as follows: $\mathcal{A}^*$ will send to A a garbled circuit for $f|_g$ as B would, but before it sends the wire input keys corresponding to its inputs, it needs to *extract* inputs (x, c) which A contributes to the covert OT. (This is why we base the covert OT of Section 4 on CCA(-covert) PKE of Cramer-Shoup: The receiver's first message will be a vector of Cramer-Shoup encryptions of the bits in string $x|c$, which the simulator will straight-line extract using the decryption key as a trapdoor.) Having extracted (x, c) from the covert OT, the simulator $\mathcal{A}^*$, playing the role of an ideal-world adversary $\mathsf{F}_{\mathsf{C}(f,g)}$'s instance identified by sid, sends x to $\mathsf{F}_{\mathsf{C}(f,g)}$ and receives $\mathsf{F}_{\mathsf{C}(f,g)}$'s reply z. Note that if $\mathcal{A}^*$ sets $u = 0$ then the only part of its input that matters is d, because $f|_g$ will outputs $c \oplus d$ to A. Simulator $\mathcal{A}^*$ will then prepare d as follows: If $z \neq \perp$, i.e. the input y to the ideal-world party B must be such that $g(x, y) = 1$, simulator $\mathcal{A}^*$ picks t' as a random $n_v\tau$ string and sets $d = c \oplus (z|t')$. In this way the garbled circuit will output $c \oplus d = z|t'$. Since t' is a sequence of n_v random bitstrings of length τ, string $z|t'$ is distributed identically to the circuit output $z|\mathsf{ts}[{:}\,v]$ which A would see in an interaction with the real-world party $B(y)$. Moreover, $\mathcal{A}^*$ can detect if A tries to cheat the real-world party B by sending a modified encoding of B's output: If A sends back the same t' which $\mathcal{A}^*$ embedded in the circuit output, then $\mathcal{A}^*$ sends $(\mathsf{Output}, \mathsf{sid}, B, \mathsf{T})$ to $\mathsf{F}_{\mathsf{C}(f,g)}$, and if B sends any other value, in which case the real-world B would reject, $\mathcal{A}^*$ sends $(\mathsf{Output}, \mathsf{sid}, B, \mathsf{F})$ to $\mathsf{F}_{\mathsf{C}(f,g)}$.

Notation for Garbled Circuit Wires. We will find it useful to fix a notation for groups of wires in the garbled circuit $f|_g$ depending on the part of the input they encode. Note that $f|_g$ takes input $((x, c), (y, d, \mathsf{ts}, u))$. We will use W to denote all the input wires, and we will use X, C, Y, D, T, U to denote the sets of wires encoding the bits of respectively $x, c, y, d, \mathsf{ts}, u$, where $|X| = n_x, |Y| = n_y, |T| = 2n_v\tau, |C| = |D| = n_z + n_v\tau, |U| = 1$. We denote the set of wires for A's inputs as $\overline{X} = X \cup C$ and the set of wires for B's inputs as $\overline{Y} = Y \cup D \cup T \cup U$. If bitstring s is formed as concatenation of any of the circuit inputs x, c, y, d, t, u and $w \in W$ then $s[w]$ denotes the bit of s corresponding to input wire w.

Fully Malicious Case. In a simple usage of the cut-and-choose technique for garbled circuits, party B would use $\mathsf{GCgen}(f|_g)$ to prepare $n = O(\tau)$ garbled circuit instances $(\mathsf{gc}_1, \mathsf{ks}_1), ..., (\mathsf{gc}_n, \mathsf{ks}_n)$, would send $(\mathsf{gc}_1, ..., \mathsf{gc}_n)$ to A, who would choose a random subset $S \in [n]$ of $n/2$ elements, send it to B, who would then open the coins it used in preparing gc_i's for $i \in S$, and proceed with the OT's and sending its input wire keys for all gc_i's for $i \notin S$. Party A would then check that each gc_i for $i \in S$ is formed correctly, and it would evaluate each gc_i for $i \notin S$. If at least $n/4$ of these returned the same value w, A would interpret this as the correct output $w = (z, \mathsf{ts}[{:}\,v])$ of $f|_g$, output z locally and send $\mathsf{ts}[{:}\,v]$ to B, who would decode it to its output v. In order to enforce consistency of the inputs which both parties provide to circuit instances $\{\mathsf{gc}_i\}_{i \notin S}$, we would have each party to commit to their inputs to $f|_g$, and then use efficient ZK proofs that the keys B sends and the bits A chooses in the OT's for the evaluated gc_i instances correspond to these committed inputs. Further, B would need to commit to each key in the wire key sets $\{\mathsf{ks}_i\}_{i \in [n]}$, and show in a ZK proof that the keys it sends and enters into the OT's for $i \notin S$ are the committed keys. Our protocol uses each of the elements of this sketch, but with several modifications.

Step 1: ZK→CKEM, Com/Decom→CKEM. First, we follow the above method using covert commitments, covert circuit garbling, and covert OT. Second, we replace all the ZK proofs with covert simulation-sound CKEM's. Next, note that circuits $\{\mathsf{gc}_i\}_{i \in [n]}$ in themselves are indistinguishable from random by covertness of the garbling procedure, but if gc_i's were sent in the clear then B could not then open the coins used in the preparation of gc_i's for $i \in S$, because coin $\mathsf{r}_i^{\mathsf{gc}}$ together with gc_i s.t. $(\mathsf{gc}_i, \mathsf{ks}_i) = \mathsf{GCgen}(f|_g; \mathsf{r}_i^{\mathsf{gc}})$ forms a publicly verifiable (commitment,decommitment) pair. We deal with it roughly the way we deal with general (commitment,decommitment) sequence. In this specific case, we replace gc_i's in B's first message with covert commitments to both the circuits, $\mathsf{cgc}_i \leftarrow \mathsf{Com}_{\mathsf{pk}}(\mathsf{gc}_i; \mathsf{r}_i^{\mathsf{cgc}})$, and to all the input wire keys, $\mathsf{ck}_i^{w,b} \leftarrow \mathsf{E}_{\mathsf{pk}}(k_i^{w,b}; \mathsf{r}_{i,w,b}^{\mathsf{ck}})$. When B sends $\mathsf{r}_i^{\mathsf{gc}}$ for $i \in S$, A can derive $(\mathsf{gc}_i, \{k_i^{w,b}\}_{w,b}) \leftarrow \mathsf{GCgen}(f|_g; \mathsf{r}_i^{\mathsf{gc}})$, and now A has a (commitment,message) pair $(c, m) = (\mathsf{cgc}_i, \mathsf{gc}_i)$ and (encryption,message) pairs $(c, m) = (\mathsf{ck}_i^{w,b}, k_i^{w,b})$, while B has the randomness r s.t. $c = \mathsf{Com}_{\mathsf{pk}}(m; r)$ or $c = \mathsf{E}_{\mathsf{pk}}(m; r)$. Since we implement $\mathsf{Com}(m)$ as $\mathsf{E}(H(m))$, both instances can be dealt with a covert CKEM, with sender A and receiver B, for the statement that (c, m) is in the language of correct (ciphertext,plaintext) pairs for Cramer-Shoup encryption, i.e. $\mathsf{Le}^{\ell}(\mathsf{pk})$. Finally, to covertly encode the random $n/2$-element subset S chosen by A, we have A send to B not the set S but the coins r^{SG} which A uses in the subset-generation procedure SG which generates a random $n/2$-element subset on n-element set.

Let us list the CKEM's which the above procedure includes so far. A has to prove that it inputs into the OT's for all $i \notin S$ the same bits which A (covertly) committed in its first message. Recall that in the covert OT based on the Cramer-Shoup encryption (see Section 4) the receiver's first message is the encryption of its bit. We will have A then commit to its bits by encrypting them, and so the proof we need is that the corresponding plaintexts are bits, and for that purpose we will use a CKEM for language $\mathsf{Lbit}^{\ell}(\mathsf{pk})$ (see language LA below). Party B has more to prove: First, it needs to prove all the $\mathsf{Le}^{\ell}(\mathsf{pk})$ statements as explained above (these correspond to items #1, #2, and #3 in the specification of LB below). Second, B proves that its committed inputs on wires $\overline{Y} \setminus D$ are bits, using CKEM for $\mathsf{Lbit}^{\ell}(\mathsf{pk})$, and that for $i \notin S$ it reveals keys consistent with these committed inputs. Both statements will be handled by CKEM for language Ldis, see below, which subsumes language $\mathsf{Lbit}^{\ell}(\mathsf{pk})$ (see item #4 in the specification of LB). Third, for reasons we explain below, B will not prove the consistency of inputs d it enters into n instances of garbled circuit $f|_g$, hence for $i \notin S$ and $w \in D$ it needs only to prove that the revealed key $k_i^{w,b}$ is committed either in $\mathsf{ck}_i^{w,0}$ or $\mathsf{ck}_i^{w,1}$, which is done via CKEM for Ldis' (see below, and item #5 in the specification of LB). Finally, B proves that it computes the OT responses otr_i^w correctly, for $i \notin S$ and $w \in \overline{X}$, on A's first OT message ct_i^w using the keys committed in $\mathsf{ck}_i^{w,0}, \mathsf{ck}_i^{w,1}$, which is done via CKEM for Lotr (see below, and item #6 in the specification of LB).

Step 2: Input Consistency Across Garbled Circuit Copies. We must ensure that A and B input the same x and y into each instance of the garbled circuit $f|_g$. However, the decision process in our cut-and-choose approach is that A decides whether the outputs w_i of $n/2$ garbled circuits $i \notin S$ it evaluates correspond to $(z, \mathsf{ts}[:v])$ for $(z, v) = f(x, y)$ or to random bits, is that it decides on the former if $n/4$ of the w_i's are the same. Hence, to prevent B from getting honest A into that case if $g(x, y) = 0$ (or $u = 0$), A chooses each c_i at random, so in that case the (correctly formed) circuits in $[n] \setminus S$ output $w_i = c_i \oplus d_i$ values which

B cannot control. Consequently, B must also choose each d_i independently at random, which is why B does not have to commit to the inputs on wires in D.

Step 3: Straight-Line Extraction of Inputs. As we sketched before, we get concurrent security by using CCA-covert encryption as, effectively, a non-malleable and straight-line extractable covert commitment. Each player commits to its input bits by encrypting them (except B does not encrypt its inputs on D wires), and the simulator decrypts the bits effectively contributed by a malicious party. However, for the sake of efficient CKEM's we need these bit encryptions to use a "shifted" form, i.e. an encryption of bit b will be $\mathsf{E}_{\mathsf{pk}}(g^b)$ and not $\mathsf{E}_{\mathsf{pk}}(b)$. This is because the Cramer-Shoup encryption E has group G as a message space. Also, if bit b is encrypted in this way then we can cast the language Lbit (and languages Ldis and Ldis') as an LMI language with an efficient covert CKEM.

Step 4: Encoding and Encrypting Wire Keys. To enable efficient covert CKEM's for the languages we need we also modify the garbling procedure GCgen of [19] so it chooses wire keys $k^{w,b}$ corresponding to A's input wires, i.e. for $w \in \overline{X}$, as random elements in G, but keys $k^{w,b}$ corresponding to B's input wires, i.e. for $w \in \overline{Y}$, as random elements in $\mathbb{Z}_p$. Note that either value can be used to derive a standard symmetric key, e.g. using a strong extractor with a seed in the CRS. We use the same encryption E to commit to these wire keys, but we commit them differently depending on whose wires they correspond to, namely as $\mathsf{ck}^{w,b} = \mathsf{E}_{\mathsf{pk}}(k^{w,b})$ for $w \in \overline{X}$, because $k^{w,b} \in G$ for $w \in \overline{X}$, and as $\mathsf{ck}^{w,b} = \mathsf{E}_{\mathsf{pk}}(g^{k^{w,b}})$ for $w \in \overline{Y}$, because $k^{w,b} \in \mathbb{Z}_p$ for $w \in \overline{Y}$. The reason we do this is that values $\mathsf{ck}^{w,b}$ for $w \in \overline{X}$ take part in the CKEM for correct OT response language Lotr, and since in OT the encrypted messages (which are the two wires keys $k^{w,0}$ and $k^{w,1}$) will be in the base group, hence we need the same keys to be in the base group in commitments $\mathsf{ck}^{w,0}, \mathsf{ck}^{w,1}$. By contrast, values $\mathsf{ck}^{w,b}$ for $w \in \overline{Y}$ take part in the CKEM of language Ldis, for proving consistency of key k^w opened by B with B's commitment ct^w to bit b on wire w. Bit b is in the exponent in ct^w, and using homomorphism of exponentiation, this allows us to cast language Ldis as an LMI language provided that k^w is also in the exponent in $\mathsf{ck}^{w,0}$ and $\mathsf{ck}^{w,1}$.

Step 5: Using CKEM Keys to Encrypt and/or Authenticate. We will run two CKEM's: After A's first message, containing A's input commitments, we run a covert CKEM for language LA for correctness of A's messages, with sender B and receiver A, denoting the keys this CKEM establishes as K_B for B and K'_B for A. Subsequently, B will encrypt its messages under key K_B, using covert encryption $(\mathsf{SE}, \mathsf{SD})$ implemented as $\mathsf{SE}^0_K(\mathsf{m}) = \mathsf{G}^{|\mathsf{m}|}(\mathsf{F}(K, 0)) \oplus \mathsf{m}$ and $\mathsf{SD}^0_K(\mathsf{ct}) = \mathsf{G}^{|\mathsf{ct}|}(\mathsf{F}(K, 0)) \oplus \mathsf{ct}$, where F is a PRF with τ-bit keys, arguments, and outputs, G^ℓ is a PRG with τ-bit inputs and ℓ-bit outputs. Similarly when B responds as described above given A's chosen set $S \subset [n]$, we run a covert CKEM for language LB for correctness of B's messages, with sender A and receiver B, establishing keys K_A for A and K'_A for B, and A will encrypt its subsequent messages using the same covert encryption. In the last two messages we will use values $\mathsf{F}(\mathsf{K}_B, 1)$, $\mathsf{F}(\mathsf{K}_B, 2)$, and $\mathsf{F}(\mathsf{K}_A, 1)$ derived from the same CKEM keys as, resp. a one-time authenticator for A's message m^2_A, an encryption key for B's final message m^3_B, and a one-time authenticator for that same message.

Covert CKEM's for Linear Map Image Languages. Protocol Π_{COMP} uses CKEM's for two languages: Language LA which contains correctly formed wire-bit ciphertexts sent by A, and language LB which contains correctly formed messages sent by B. Both are formed as conjunctions of LMI languages, hence both are LMI languages as well. Let $(\mathsf{Kg}, \mathsf{E}, \mathsf{D})$ be the CCA-covert Cramer-Shoup PKE. All languages below are implicitly parametrized by the public key pk output by $\mathsf{Kg}(1^\tau)$ and some label ℓ. (Formally, pk, ℓ is a part of each statement in the given language.) Recall that the public key pk specifies the prime-order group setting (g, G, p).

We first list all the component languages we need to define LA and LB. We defer to Appendix B for the specification of the mapping between the instances of each language to instance (C, M) of $\mathsf{LMI}_{n,m}$ for some n, m.

Language Le of correct (ciphertext,label,plaintext) tuples for plaintext $\mathsf{m} \in G$:

$$\mathsf{Le}^\ell(\mathsf{pk}) = \{(\mathsf{ct}, \mathsf{m}) \text{ s.t. } \mathsf{ct} \in \mathsf{E}^\ell_{\mathsf{pk}}(\mathsf{m})\}$$

Language Lbit of "shifted" encryptions of a bit:

$$\mathsf{Lbit}^\ell(\mathsf{pk}) = \{\mathsf{ct} \text{ s.t. } \exists b \in \{0,1\}\ (\mathsf{ct}, g^b) \in \mathsf{Le}^\ell(\mathsf{pk})\}$$

Language Ldis is used for proving that a key corresponding to some *sender's wire* in Yao's garbled circuit is consistent with the two key values the sender committed in $\mathsf{ck}_0, \mathsf{ck}_1$ and with the bit the sender committed in ct. To cast this language as a (simple) LMI language we use the "shifted" version of Cramer-Shoup encryption in these statements, i.e. we encrypt $g^{\mathsf{m}} \in G$ instead of $\mathsf{m} \in \mathbb{Z}_p$. In other words, Ldis consists of tuples $(\mathsf{m}, \mathsf{ct}, \mathsf{ck}_0, \mathsf{ck}_1)$ s.t. either (ct encrypts g^0 and ck_0 encrypts g^{m}) or (ct encrypts g^1 and ck_1 encrypts g^{m}):

$$\mathsf{Ldis}^{\ell,i}(\mathsf{pk}) = \{(\mathsf{ct}, \mathsf{m}, \mathsf{ck}_0, \mathsf{ck}_1) \text{ s.t. } \exists b{\in}\{0,1\}\ (\mathsf{ct}, g^b){\in}\mathsf{Le}^\ell(\mathsf{pk}){\wedge}(\mathsf{ck}_b, g^{\mathsf{m}}){\in}\mathsf{Le}^{[\ell|i|b]}(\mathsf{pk})\}$$

Language Ldis' is a simplification of Ldis which omits checking the constraint imposed by ciphertext ct.

Language Lotr is used for proving correctness of a response in an Oblivious Transfer of Aiello et al. [1], formed using procedure OTrsp (see Section 4), which the sender uses in Yao's protocol to send keys corresponding to *receiver's wires*:

$$\begin{aligned}\mathsf{Lotr}^\ell(\mathsf{pk}) \;=\; \{\; & (\mathsf{otr}, \mathsf{ct}, \mathsf{ck}_0, \mathsf{ck}_1) \quad \text{s.t.} \quad \exists k_0, k_1, \mathsf{r} \\ & (\mathsf{ck}_0, k_0) \in \mathsf{Le}^{[\ell|0]}(\mathsf{pk}) \wedge (\mathsf{ck}_1, k_1) \in \mathsf{Le}^{[\ell|1]}(\mathsf{pk}) \wedge \mathsf{otr} = \mathsf{OTrsp}_{\mathsf{pk}}(\mathsf{ct}, k_0, k_1; \mathsf{r}) \;\; \}\end{aligned}$$

We use the above component languages to define languages LA and LB as follows:

$$\begin{aligned}\mathsf{LA}^{\ell_A}(\mathsf{pk}) \;=\; \{\; & \big(\;\{\mathsf{ct}^w\}_{w\in X}, \{\mathsf{ct}^w_i\}_{i\in[n],w\in C}\;\big) \\ & \text{s.t.} \quad \mathsf{ct}^w \in \mathsf{Lbit}^{[\ell_A|w]}(\mathsf{pk}) \qquad \text{for all } w \in X \\ & \text{and} \quad \mathsf{ct}^w_i \in \mathsf{Lbit}^{[\ell_A|w|i]}(\mathsf{pk}) \quad \text{for all } i \in [n],\, w \in X \quad \}\end{aligned}$$

$$
\begin{aligned}
\mathsf{LB}^{\ell_B}(\mathsf{pk}) \;=\; \{\; \big(\; & \{(\mathsf{cgc}_i, H(\mathsf{gc}_i))\}_{i\in[n]} \\
& \{(k_i^{w,b}, \mathsf{ck}_i^{w,b})\}_{i\in S, w\in\overline{X}, b\in\{0,1\}} \\
& \{(g^{k_i^{w,b}}, \mathsf{ck}_i^{w,b})\}_{i\in S, w\in\overline{Y}, b\in\{0,1\}} \\
& \{(k_i^{w}, \mathsf{ct}^{w}, \mathsf{ck}_i^{w,0}, \mathsf{ck}_i^{w,1})\}_{i\notin S,\, w\in\overline{Y}\setminus D} \\
& \{(k_i^{w}, \mathsf{ck}_i^{w,0}, \mathsf{ck}_i^{w,1})\}_{i\notin S,\, w\in D} \\
& \{(\mathsf{otr}_i^{w}, \mathsf{ct}_i^{w}, \mathsf{ck}_i^{w,0}, \mathsf{ck}_i^{w,1})\}_{i\notin S, w\in\overline{X}} \;\;\big)
\end{aligned}
$$

$$
\begin{array}{lll}
\text{s.t.} & & \\
(1) & (\mathsf{cgc}_i, H(\mathsf{gc}_i)) \in \mathsf{Le}^{[\ell_B|i]}(\mathsf{pk}) & \text{for } i\in[n] \\
(2) & (\mathsf{ck}_i^{w,b}, k_i^{w,b}) \in \mathsf{Le}^{[\ell_B|w|i|b]}(\mathsf{pk}) & \text{for } i\in S,\ w\in\overline{X},\ b\in\{0,1\} \\
(3) & (\mathsf{ck}_i^{w,b}, g^{k_i^{w,b}}) \in \mathsf{Le}^{[\ell_B|w|i|b]}(\mathsf{pk}) & \text{for } i\in S,\ w\in\overline{Y},\ b\in\{0,1\} \\
(4) & (\mathsf{ct}^{w}, k_i^{w}, \mathsf{ck}_i^{w,0}, \mathsf{ck}_i^{w,1}) \in \mathsf{Ldis}^{[\ell_B|w],i}(\mathsf{pk}) & \text{for } i\notin S,\ w\in\overline{Y}\setminus D \\
(5) & (k_i^{w}, \mathsf{ck}_i^{w,0}, \mathsf{ck}_i^{w,1}) \in \mathsf{Ldis'}^{[\ell_B|w],i}(\mathsf{pk}) & \text{for } i\notin S,\ w\in D \\
(6) & (\mathsf{otr}_i^{w}, \mathsf{ct}_i^{w}, \mathsf{ck}_i^{w,0}, \mathsf{ck}_i^{w,1}) \in \mathsf{Lotr}^{[\ell_B|w]}(\mathsf{pk}) & \text{for } i\notin S,\ w\in\overline{X} \quad \}
\end{array}
$$

Notation in Figure 2. Procedures $(\mathsf{Kg}, \mathsf{E}, \mathsf{D})$, $(\mathsf{GCgen}, \mathsf{GCev})$, Com, SG, $(\mathsf{OTrsp}, \mathsf{OTfin})$, CKEM, $(\mathsf{F}, G, \mathsf{SE}, \mathsf{SD})$ are as explained above. If P is a randomized algorithm we sometimes explicitly denote its randomness as r^{P}, with the implicit assumption that it is a random string. Expression $\{x_i \leftarrow \mathsf{P}\}_{i\in R}$ denotes *either* a loop "perform $x_i \leftarrow \mathsf{P}$ for each i in R", *or* a set of values $\{x_i\}_{i\in R}$ resulting from executing such a loop. Letter b always stands for a bit, and expressions $\{...\}_b$ stand for $\{...\}_{b\in\{0,1\}}$.

Cost Discussions. Since the Covert CKEM's in ROM of Section 7.1 have virtually the same computation and bandwidth costs as standard ZK proofs for the same languages, protocol Π_{COMP} realizes the concurrent *covert* 2PC functionality $\mathsf{F}_{\mathsf{C}(f,g)}$ with $O(1)$ rounds, $O(\tau|C|)$ bandwidth, $O(\tau|C|)$ symmetric cipher operations, and $O(\tau|W|)$ exponentiations, where $|C|$ is the number of gates and $|W|$ is the size of the input in the circuit for function $f|_g$, and τ is the security parameter. This places covert computation in the same efficiency ballpark as existing $O(1)$-round secure (but *not* covert) "cut-and-choose over garbled circuits" 2PC protocols. Of course there remains plenty of room for further improvements: Protocol Π_{COMP} uses $2.4\cdot\tau$ garbled circuit copies instead of τ as the 2PC protocols of [12, 15], it does not use an OT extension, and it does not use many other bandwidth and cost-saving techniques that were developed over the last decade to increase the efficiency of standard, i.e. non-covert, 2PC protocols. However, we see no inherent reasons why, using the techniques we employed here, many of the same cost-saving techniques cannot be adopted to covert computation.

Here we single out two particular sources of an "efficiency gap" between our covert 2PC and current secure 2PC protocols that perhaps stand out. First,

$\mathsf{PG}(1^\tau)$ picks $(\mathsf{pk},\mathsf{sk}) \leftarrow \mathsf{Kg}(1^\tau)$ and sets $\pi \leftarrow \mathsf{pk}$.

B1: on input $(\mathsf{InputB}, A, y, \mathsf{sid})$ and $\ell_B = (B, A, \mathsf{sid})$:
set $\{(\mathsf{gc}_i, \{k_i^{w,b}\}_{w\in W,b}) \leftarrow \mathsf{GCgen}(f|_g; \mathsf{r}_i^{\mathsf{gc}})\}_{i\in[n]}$ and $\{\mathsf{cgc}_i \leftarrow \mathsf{Com}_{\mathsf{pk}}^{[\ell_B|i]}(\mathsf{gc}_i)\}_{i\in[n]}$;
set $\{\mathsf{ck}_i^{w,b} \leftarrow \mathsf{E}_{\mathsf{pk}}^{[\ell_B|w|i|b]}(k_i^{w,b})\}_{w\in\overline{X},i\in[n],b}$ and $\{\mathsf{ck}_i^{w,b} \leftarrow \mathsf{E}_{\mathsf{pk}}^{[\ell_B|w|i|b]}(g^{k_i^{w,b}})\}_{w\in\overline{Y},i\in[n],b}$;
send $\mathsf{m}_B^1 = (\{\mathsf{cgc}_i\}_{i\in[n]}, \{\mathsf{ck}_i^{w,b}\}_{i\in[n],w\in W,b})$ to A.

A1: on input $(\mathsf{InputA}, B, x, \mathsf{sid})$ and $\ell_A = (A, B, \mathsf{sid})$, and message m_B^1 from B:
sample $S \leftarrow \mathsf{SG}(n; \mathsf{r}^{\mathsf{SG}})$, pick $c_i \leftarrow \{0,1\}^{n_z+n_v\tau}$ for $i\in[n]$;
set $\mathsf{x}_A \leftarrow (\{\mathsf{ct}^w \leftarrow \mathsf{E}_{\mathsf{pk}}^{[\ell_A|w]}(g^{x[w]}; \mathsf{r}_w^{\mathsf{ct}})\}_{w\in X}, \{\mathsf{ct}_i^w \leftarrow \mathsf{E}_{\mathsf{pk}}^{[\ell_A|w|i]}(g^{c_i[w]}; \mathsf{r}_{w,i}^{\mathsf{ct}})\}_{w\in C,i\in[n]})$;
set $\mathsf{w}_A \leftarrow (x, \{c_i\}_{i\in[n]}, \{\mathsf{r}_w^{\mathsf{ct}}\}_{w\in X}, \{\mathsf{r}_{w,i}^{\mathsf{ct}}\}_{w\in C,i\in[n]})$, send $\mathsf{m}_A^1 = (\mathsf{r}^{\mathsf{SG}}, \mathsf{x}_A)$ to B.

A,**B** run $\mathsf{CKEM}_{\mathsf{LA}^{(\ell_A)}(\mathsf{pk})}$ on x_A and A's input w_A; let B output K_B and A output K'_B.

B2: on K_B and r^{SG} received in m_A^1 from A:
re-generate $S \leftarrow \mathsf{SG}(n; \mathsf{r}^{\mathsf{SG}})$, set $u = 1$, pick $t \leftarrow \{0,1\}^{2n_v\tau}$ and $\{d_i \leftarrow \{0,1\}^{n_z+n_v\tau}\}_{i\in[n]}$;
set $\mathsf{ct}^{\mathsf{B}} \leftarrow \{\mathsf{ct}^w \leftarrow \mathsf{E}_{\mathsf{pk}}^{[\ell_B|w]}(g^{\overline{y}[w]})\}_{w\in\overline{Y}\setminus D}$ where $\overline{y} = y|t|u$;
set $\{k_i^w \leftarrow k_i^{w,\overline{y}_i[w]}\}_{w\in\overline{Y},i\notin S}$ where $\overline{y}_i = y|t|u|d_i$, and $\{\mathsf{ks}_i^B \leftarrow \{k_i^w\}_{w\in\overline{Y}}\}_{i\notin S}$;
set $\{\mathsf{otr}_i^w \leftarrow \mathsf{OTrsp}_{\mathsf{pk}}(\mathsf{ct}_i^w, k_i^{w,0}, k_i^{w,1})\}_{w\in\overline{X},i\notin S}$, where $\mathsf{ct}_i^w = \mathsf{ct}^w$ for $w\in X$;
send $\mathsf{m}_B^2 = \mathsf{SE}^0_{\mathsf{K}_B}[\mathsf{ct}^{\mathsf{B}}, \{\mathsf{r}_i^{\mathsf{gc}}\}_{i\in S}, \{\mathsf{gc}_i, \mathsf{ks}_i^B, \{\mathsf{otr}_i^w\}_{w\in\overline{X}}\}_{i\notin S}]$ to A.

A2: on K'_B and m_B^2:
set $(\mathsf{ct}^{\mathsf{B}}, \{\mathsf{r}_i^{\mathsf{gc}}\}_{i\in S}, \{\mathsf{gc}_i, \mathsf{ks}_i^B, \{\mathsf{otr}_i^w\}_{w\in\overline{X}}\}_{i\notin S}) \leftarrow \mathsf{SD}^0_{\mathsf{K}'_B}(\mathsf{m}_B^2)$;
set $\{\mathsf{ks}_i^A \leftarrow \{k_i^w \leftarrow \mathsf{OTfin}_{\mathsf{pk}}(\overline{x}_i[w], \mathsf{r}_{w,i}^{\mathsf{ct}}, \mathsf{otr}_i^w)\}_{w\in\overline{X}}\}_{i\notin S}$, for $\overline{x}_i = x|c_i$ and $\mathsf{r}_{w,i}^{\mathsf{ct}} = \mathsf{r}_w^{\mathsf{ct}}$ for $w\in X$;
set $(\mathsf{gc}_i, \{k_i^{w,b}\}_{w,b}) \leftarrow \mathsf{GCgen}(f|_g; \mathsf{r}_i^{\mathsf{gc}})$ for $i\in S$ and $w_i \leftarrow \mathsf{GCev}(\mathsf{gc}_i, (\mathsf{ks}_i^A \cup \mathsf{ks}_i^B))$ for $i\notin S$.

A,**B** run $\mathsf{CKEM}_{\mathsf{LB}^{(\ell_B)}(\mathsf{pk})}$ on x_B and B's input w_B for $\mathsf{x}_B = (\{\mathsf{ct}^w, k_i^w, \mathsf{ck}_i^{w,0}, \mathsf{ck}_i^{w,1}\}_{i\notin S,w\in\overline{Y}\setminus D}$, $\{k_i^w, \mathsf{ck}_i^{w,0}, \mathsf{ck}_i^{w,1}\}_{i\notin S,w\in D}, \{\mathsf{gc}_i, \mathsf{cgc}_i\}_{i\in[n]}, \{\mathsf{ck}_i^{w,b}, k_i^{w,b}\}_{i\in S,w\in W,b}, \{\mathsf{otr}_i^w, \mathsf{ct}_i^w, \mathsf{ck}_i^{w,b}\}_{i\notin S,w\in\overline{X},b})$, and w_B containing input y and randomness of B. Let A output K_A and B output K'_A.

A3: If $\exists R\subset[n]$ s.t. $|R|=n/4$ and $\exists w$ s.t. $\forall_{i\in R}\ w_i = w$ then set $(z|t_1|...|t_{n_v}){:=}w$ and $\mathsf{m}_A^2 \leftarrow \mathsf{SE}^0_{\mathsf{K}_A}(\mathsf{F}(\mathsf{K}'_B, 1), t_1, ..., t_{n_v})$; Otherwise set $z{:=}\perp$ and $\mathsf{m}_A^2 \leftarrow \{0,1\}^{\tau(n_v+1)}$. Send m_A^2 to B.

B3: Set $(\tau, t_1, ..., t_{n_v}) \leftarrow \mathsf{SD}^0_{\mathsf{K}'_A}(\mathsf{m}_A^2)$. Parse t as $[t_1^0|t_1^1|...|t_{n_v}^0|t_{n_v}^1]$. If $\tau{=}\mathsf{F}(\mathsf{K}_B, 1)$ and $t_j \in \{t_j^0, t_j^1\}$ for all $j\in[n_v]$ then set $\mathsf{m}_B^3 \leftarrow \mathsf{F}(\mathsf{K}_B, 2) \oplus \mathsf{F}(\mathsf{K}'_A, 1)$ and $\forall_j$ set $v[j] := b$ s.t. $t_j = t_j^b$; Otherwise set $v{:=}\perp$ and $\mathsf{m}_B^3 \leftarrow \{0,1\}^\tau$. Send m_B^3 to A and output (Output, v).

A4: If $\mathsf{m}_B^3 \neq \mathsf{F}(\mathsf{K}'_B, 2) \oplus \mathsf{F}(\mathsf{K}_A, 1)$ then set $z{:=}\perp$. Output (Output, z).

Fig. 2. Protocol Π_{COMP} for Concurrent 2-Party Covert Function Computation $\mathsf{F}_{\mathsf{C}(f,g)}$

protocol Π_{COMP} exchanges $O(\tau)$ garbled circuits instead of $O(\kappa)$ where κ is the statistical security parameter. We could do the latter as well, but the result would realize a weaker functionality than $\mathsf{F}_{\mathsf{C}(f,g)}$ defined in Section 3. Namely, with probability $2^{-\kappa}$ the functionality would allow the adversary to specify any function on the joint inputs, and this function would be computed by the honest party. Secondly, circuit $f|_g$ which is garbled in protocol Π_{COMP} increases the number of input wires of the underlying circuit for $\mathsf{F}_{\mathsf{C}(f,g)}$ by $O(n\tau)$ where n is the bitsize of the *output* of function f. However, since this extension in the input wire count was done for conceptual simplicity (see a note on *Encoding B's Output* on page 19), it might be avoidable with a more careful analysis.

Theorem 1. *Protocol* Π_{COMP} *in Figure 2 realizes the concurrent 2-party covert computation functionality* $\mathsf{F}_{\mathsf{C}(f,g)}$ *in the CRS model, assuming* $(\mathsf{Kg}, \mathsf{E}, \mathsf{D})$ *is a covert CCA public key encryption,* F *is a PRF,* G *is a PRG,* $(\mathsf{GCgen}, \mathsf{GCev})$ *is a covert garbling scheme,* $(\mathsf{OTreq}, \mathsf{OTrsp}, \mathsf{OTfin})$ *is a covert OT, and* $\mathsf{CKEM}_{\mathsf{LA}(\mathsf{pk})}$ *and* $\mathsf{CKEM}_{\mathsf{LB}(\mathsf{pk})}$ *are covert zero-knowledge and simulation-sound CKEM's for languages resp.* LA *and* LB.

Proof. In Figures 3 and 4 we specify the algorithm of the ideal-model adversary $\mathcal{A}^*$, i.e. the simulator, which interacts with functionality $\mathsf{F}_{\mathsf{C}(f,g)}$ and the real-world adversary Adv in such a way that for every efficient Adv and environment $\mathcal{Z}$, the environment's view of an interaction with Adv and honest players participating in protocol Π_{COMP} is indistinguishable from $\mathcal{Z}$'s view of an interaction with $\mathcal{A}^*$ (who locally interacts with a copy of Adv) and honest dummy players interacting via functionality $\mathsf{F}_{\mathsf{C}(f,g)}$. We split $\mathcal{A}^*$'s into two figures for presentation convenience, but these are two parts of the same algorithm: Figure 3 describes how $\mathcal{A}^*$ handles an interaction with $\mathsf{F}_{\mathsf{C}(f,g)}$ which pertain to honest ideal-world party B, while Figure 4 describes how $\mathcal{A}^*$ handles an interaction with $\mathsf{F}_{\mathsf{C}(f,g)}$ which pertain to honest ideal-world party A. The $\mathcal{A}^*$ code in these figures assumes that the CRS is chosen by setting $(\mathsf{pk}, \mathsf{sk}) \leftarrow \mathsf{Kg}(1^\tau)$ and $\pi \leftarrow \mathsf{pk}$, and that value sk is given to $\mathcal{A}^*$ as the simulation trapdoor.

Below we split the security argument, i.e. that $\mathcal{Z}$'s interaction with real-world honest players and Adv is indistinguishable from an interaction with dummy ideal-world honest players and $\mathcal{A}^*$, into four parts, distinguishing by two types of honest parties, A and B, and breaking each of these cases into two subcases depending on whether their inputs, resp. x and y, are bitstrings in $\{0,1\}^{n_x}$ resp. $\{0,1\}^{n_y}$, or the non-participation signals $\perp$. In each case we show that the interaction between Adv and an honest party following Π_{COMP}'s session with identifier sid is indistinguishable from the corresponding session executed by $\mathcal{A}^*$ interacting with $\mathsf{F}_{\mathsf{C}(f,g)}$ and a dummy (ideal-world) honest party. In the arguments below we call an interaction of $\mathcal{Z}$ (for fixed Adv) with the algorithms we specify a *security game*, denoted G_i, and we write $G_i \approx G_j$ to denote that $\mathcal{Z}$'s output in G_i is indistinguishable of $\mathcal{Z}$'s output in G_j.

Case 1A, honest B on input $y^* \in \{0,1\}^{n_y}$: Let G_0 be the interaction of $\mathcal{Z}$ and Adv in the real world where player B gets input $(\mathsf{InputB}, A, y^*, \mathsf{sid})$ from $\mathcal{Z}$

On behalf of <u>honest B</u>, on trapdoor sk and input $(\mathsf{InputB}, \mathsf{sid}, A, B)$ from $\mathsf{F}_{\mathsf{C}(f,g)}$:

(1) compute $\{\mathsf{cgc}_i, \{\mathsf{ck}_i^{w,b}\}_{w,b}\}_{i\in[n]}$ and send to A as B does in step B1;

(2) on $\mathsf{m}_A^1 = (\mathsf{r}^{\mathsf{SG}}, \mathsf{x}_A)$ for $\mathsf{x}_A = (\{\mathsf{ct}^w\}_{w\in X}, \{\mathsf{ct}_i^w\}_{i,w\in C})$ from A in A1, decrypt all ct's using sk to obtain $x, c_1, ..., c_n$, overwrite $x{:}= \perp$ if any decryption returns $\perp$ or not a bit, and send $(\mathsf{InputA}, B, x, \mathsf{sid})$ to $\mathsf{F}_{\mathsf{C}(f,g)}$ and receive $(\mathsf{Output}, z, \mathsf{sid})$ from $\mathsf{F}_{\mathsf{C}(f,g)}$;
If $x =\perp$ then in steps (4,6) below pick $\mathsf{m}_B^2, \mathsf{m}_B^3$ at random and in step (5) run $\mathsf{Rec}^{\$(\tau)}$.

(3) run $\mathsf{Snd}(\pi, (\mathsf{x}_A, \ell_A))$ in $\mathsf{CKEM}_{\mathsf{LA}^{(\ell_A)}(\mathsf{pk})}$, let K_B be Snd's output;

(4) set $y{:=}0^{n_y}$, $u{:=}0$, $t \leftarrow \{0,1\}^{2n_v\tau}$; if $z =\perp$ then $\forall i$ pick $d_i \leftarrow \{0,1\}^{n_z+n_v\tau}$; o/w set $t' \leftarrow \{0,1\}^{n_v\tau}$ and $d_i \leftarrow c_i \oplus (z|t')$ $\forall i$; compute ct^{B} and $\{\mathsf{ks}_i^B, \{\mathsf{otr}_i^w\}_w\}_{i\notin S}$ as in step B2, and encrypt them under K_B to A with $\{\mathsf{r}_i^{\mathsf{gc}}\}_{i\in S}$ and $\{\mathsf{gc}_i\}_{i\notin S}$ from step (1);

(5) run $\mathsf{Rec}(\pi, (\mathsf{x}_B, \ell_B), \mathsf{w}_B)$ in $\mathsf{CKEM}_{\mathsf{LB}^{(\ell_B)}(\mathsf{pk})}$ on inputs set as in Π_{COMP}, output K'_A;

(6) on m_A^2 from A, decrypt it as $\tau|t_1|...|t_{n_v}$ using K'_A; if $\tau = \mathsf{F}(\mathsf{K}_B, 1)$ and $t_1|...|t_{n_v} = t'$ then send $\mathsf{m}_B^3 = \mathsf{F}(\mathsf{K}_B, 2) \oplus \mathsf{F}(\mathsf{K}'_A, 1)$ to A and $(\mathsf{Output}, \mathsf{sid}, B, \mathsf{T})$ to $\mathsf{F}_{\mathsf{C}(f,g)}$, and otherwise send random m_B^3 in $\{0,1\}^\tau$ to A and $(\mathsf{Output}, \mathsf{sid}, B, \mathsf{F})$ to $\mathsf{F}_{\mathsf{C}(f,g)}$.

Fig. 3. Part 1 of simulator $\mathcal{A}^*$, for Π_{COMP} sessions with honest party B.

for $y^* \neq\perp$. Let G_1 be a modification of G_0, which uses the private key sk corresponding to the key pk in the CRS, to decrypt wire-input ciphertexts $\{\mathsf{ct}^w\}_{w\in X}$, $\{\mathsf{ct}_i^w\}_{i\in[n],w\in C}$ which Adv sends in message m_A^1 to B on this session, and if any of these ciphertexts does not decrypt to a bit then G_1 sets $x{:}= \perp$, at the end of $\mathsf{CKEM}_{\mathsf{LA}(\mathsf{pk})}$ it picks K_B as an independent random string, and then it continues computation as B in protocol Π_{COMP}. $G_1 \approx G_0$ by the simulation soundness of $\mathsf{CKEM}_{\mathsf{LA}(\mathsf{pk})}$: The reduction uses sk, in particular to test whether x_A defined by m_A^1 on this session is in LA. If $\mathsf{x}_A \notin \mathsf{LA}$, which is the only case G_1 differs from G_0, then if $\mathcal{Z}$ distinguishes between G_0 and G_1 then the reduction distinguishes between the real and the random key $\mathsf{K}_B = \mathsf{K}_S$ output by $\mathsf{Snd}(\pi, (\mathsf{x}_A, \ell_A))$.

Let G_2 modify G_1 s.t. B acts like a random beacon on a session where $x =\perp$ as above, i.e. it sends random strings as messages $\mathsf{m}_B^2, \mathsf{m}_B^3$ and runs $\mathsf{Rec}^{\$(\tau)}$ in $\mathsf{CKEM}_{\mathsf{LB}(\mathsf{pk})}$. $G_2 \approx G_1$ because, (1) by the PRF property of F, the key used in encryption SE in m_B^2 is indistinguishable from value $\mathsf{F}(\mathsf{K}_B, 2)$ which is used as a one-time pad in m_B^3, so we can replace them by independent random strings, which in particular means that m_B^3 can be a random string, (2) by the PRG property of G, m_B^2 can be replaced by a random string, and (3) since G_2 doesn't use Rec's output K'_A from $\mathsf{CKEM}_{\mathsf{LB}(\mathsf{pk})}$, by covert zero-knowledge of this CKEM we can replace Rec with $\mathsf{Rec}^{\$(\tau)}$. Note that G_2 acts like $\mathcal{A}^*$ if $x = \perp$.

We will consider a modification which changes values $y, \{d_i\}, t, v$ which B inputs into garbled circuit computation. Let D denotes the distribution of the circuit inputs $(\overline{x}, \overline{y}) = ((x, \{c_i\}), (y, \{d_i\}, t, v))$ in G_2 as a function of the randomness of $\mathcal{Z}$, Adv and G_2: The (Adv, env) together specify $x, \{c_i\}, y^*$, we assume that $x \neq\perp$, and G_2 sets $y{:=}y^*$, $v{:=}1$, and t and all d_i's are random strings. Let D' be an alternative distribution defined as follows: Given $(x, \{c_i\})$ decrypted from m_A^1 and y^* specified by $\mathcal{Z}$ (and assuming $x \neq \perp$), set $y{:=}0^{n_y}$, $v{:=}0$, pick t at random in $\{0,1\}^{2n_v\tau}$, and if $g(x,y^*) = 0$ then pick each d_i at random in $\{0,1\}^{n_z+n_v\tau}$, but if $g(x,y^*) = 1$ then let $(z,v){:=}f(x,y^*)$, define $t'_j{:=}t_j^{v[j]}$ for each $j{\in}[n_v]$ where $t = t_1^0|t_1^1|...|t_{n_v}^0|t_{n_v}^1$ (in other words t'_j's are chosen to encode the bits of B's output v given the authenticator values B chose in t), and let $d_i = c_i \oplus (z|t'_1|...|t'_{n_v})$ for each i.

Consider game G_3 which is like G_2 except the ciphertexts $\mathsf{ct}^{\mathsf{B}} = \{\mathsf{ct}^w\}_{w\in\overline{Y}\setminus D}$ are computed as encryptions of (y,t,v) chosen according to D' instead of D, but the keys in ks_i^B are still chosen according to $\overline{y}$ chosen as in G_2. Since B in G_3 will not have correct witnesses in $\mathsf{CKEM}_{\mathsf{LB(pk)}}$, it will run TRec using trapdoor sk in that step. $G_3 \approx G_2$ because, (1) by the simulation soundness of $\mathsf{CKEM}_{\mathsf{LB(pk)}}$, we can replace Rec by TRec in game G_2, at which point the game does not use the randomness in ciphertexts in ct^{B}, and (2) by the CCA security of cPKE, we can change the domain in which the plaintexts (y,t,v) in these ciphertexts are drawn, from D to D': The reduction will use the decryption oracle to decrypt ciphertexts sent by Adv in m_A^1, but these have different labels than the ciphertexts in ct^{B}, and it will also use this decryption oracle to implement TRec, but again on ciphertexts with different labels. (The reduction will also use sk to implement the code of $\mathcal{A}^*$ on protocol sessions with other sid's, but these ciphertexts will therefore also pertain to different labels.)

Consider G_4 in which not only the plaintexts in ct^{B} are chosen according to D', but also B picks the key sets $\mathsf{ks}_i^B = \{k_i^w\}_{w\in\overline{Y}}$ using values $(y,\{d_i\},t,v)$ chosen in D' instead of in D. $G_4 \approx G_3$ by reduction to B's security in Yao's garbled circuit procedure GCgen, because the distribution of outputs of the n copies of circuit $f|_g$ on inputs defined by $\overline{x}, \overline{y}$ (conditioned on $x \neq \perp$) is the same for $\overline{x}, \overline{y}$ sourced in D and in D': Let $w_i = f|_g(x, c_i, y, d_i, t, v)$ for $i{\in}[n]$. In the case case $g(x,y^*) = 0$ in G_3 (i.e. circuit inputs are drown from D) we have $w_i = c_i \oplus d_i$ for each i, but in the same case in G_4 we have $w_i = c_i \oplus d_i$ for each i as well because v is set to 0. (Note that it does not matter if $g(x, 0^{n_y}) = 0$, hence we can set $y{:=}0^{n_y}$ in G_4: This is why we add the "simulation bit" v to circuit $f|_g$.) In the case $g(x,y^*) = 1$, in G_3 we have $w_i = (z|t_1|...|t_{n_v})$ for each i where $t_1|...|t_{n_v}$ is a random string which encodes B's output v for $(z,v) = f(x,y^*)$. But the same happens in G'' in this case: Since $v = 0$, we have $w_i = c_i \oplus d_i$ for each i, but since $d_i = c_i \oplus (z|t'_1|...|t'_{n_v})$, we have $w_i = z|t'_1|...|t'_{n_v}$ for each i, where $t'_1|...|t'_{n_v}$ are also random strings which encode v for $(z,v) = f(x,y^*)$.

Let G_5 be like G_4 except it runs Rec instead of TRec in $\mathsf{CKEM}_{\mathsf{LB(pk)}}$. Note that G_5 has all the witnesses it needs because G_5 forms ciphertexts ct^{B} and key sets ks_i^B in a consistent way, but using values $\overline{y}$ chosen from D^2 rather than D^1. $G_5 \approx G_4$ because of covert zero-knowledge of $\mathsf{CKEM}_{\mathsf{LB(pk)}}$.

Consider G_6 which modifies G_5 by changing how B computes its output in step B3 as follows: If m_A^2 decrypts under K_A to $\tau|t'_1|...|t'_{n_v}$ for $\tau = \mathsf{F}(\mathsf{K}_B, 1)$ then output v, otherwise output $\perp$. $G_6 \approx G_5$ by security of the Yao's garbled circuit procedure GCgen and the OT scheme $(\mathsf{OTreq}, \mathsf{OTrsp}, \mathsf{OTfin})$, because garbled circuit evaluation hides everything about B's inputs except the circuit output, and that contains only values $t'_1, ..., t'_{n_v}$ in t. In particular Adv cannot return any other value in t, except for negligible probability, and thus B's output is either v or $\perp$ except for negligible probability

Finally, note that G_6 proceeds like the simulator $\mathcal{A}^*$ interacting with $\mathsf{F}_{\mathsf{C}(f,g)}$ and ideal-world honest B who receives $(\mathsf{InputB}, A, y^*, \mathsf{sid})$ from $\mathcal{Z}$: The only difference is that G_5 evaluates $g(x, y^*)$ and $f(x, y^*)$ locally, while in the simulation $\mathcal{A}^*$ sends x to $\mathsf{F}_{\mathsf{C}(f,g)}$ and both functions are evaluated by $\mathsf{F}_{\mathsf{C}(f,g)}$. However, note that $\mathsf{F}_{\mathsf{C}(f,g)}$ replies to $\mathcal{A}^*$ with $z =\perp$ if and only if $g(x, y^*) = 0$, and that (1) $\mathcal{A}^*$ follows the same procedure on $z =\perp$ as G_6 does if $g(x, y*) = 0$, and (2) $\mathcal{A}^*$ follows the same procedure on $z = f_z(x, y^*) \neq\perp$ as G_6 does using $z = f_z(x, y^*)$ when $g(x, y) = 1$. Moreover, in both cases B can output only $v = f_v(x, y^*)$ or $\perp$, and the condition which controls which is the case is the same in both interactions, i.e. B outputs v if and only m^3_A decrypts to $\mathsf{F}(\mathsf{K}_B, 1)|t'_1|...|t'_{n_v}$. Hence we conclude that $\mathcal{Z}$'s view of an interaction with honest B on input $y^* \neq\perp$ is indistinguishable from its view of an interaction with $\mathcal{A}^*$ and an ideal dummy player B communicating through $\mathsf{F}_{\mathsf{C}(f,g)}$.

Case 1B, honest B on input $y^* =\perp$: Let G_0 be the interaction of $\mathcal{Z}$ and Adv in the real world where player B gets input $(\mathsf{InputB}, A, \perp, \mathsf{sid})$ from $\mathcal{Z}$. Note that in this case B is a random beacon, i.e. its messages $\mathsf{m}^1_B, \mathsf{m}^2_B, \mathsf{m}^3_B$ are random strings of the appropriate length, and we will assume for notational convenience that B follows $\mathsf{Snd}^{\$(\tau)}$ in $\mathsf{CKEM}_{\mathsf{LA(pk)}}$ and $\mathsf{TRec}^{\$(\tau)}$ in $\mathsf{CKEM}_{\mathsf{LB(pk)}}$. Also B always sends $\perp$ back to $\mathcal{Z}$ as its computation "output".

Let G_1 be a modification of G_0 in which B forms m^1_B as in the case $y^* \neq\perp$, i.e. it forms $\{\mathsf{gc}_i, \{k^{w,b}_i\}_{w,b}\}_i$ using $\mathsf{GCgen}(f|_g)$ and forms commitments $\{\mathsf{cgc}_i\}$ and wire-key encryptions $\{\mathsf{ck}^{w,b}_i\}$ correctly. $G_1 \approx G_0$ because the rest of G_1's computation does not involve these values, this follows from the covertness of the symmetric encryption used in ciphertexts tables by GCgen and by the covertness of encryption E (commitment Com is an instance of E).

Let G_2 be a further modification that runs Snd on statement in m^1_A instead of $\mathsf{Snd}^{\$(\tau)}$ in $\mathsf{CKEM}_{\mathsf{LA(pk)}}$. $G_2 \approx G_1$ by covert soundness of $\mathsf{CKEM}_{\mathsf{LA(pk)}}$.

Let G_3 be a further modification which in step B2 sets $y{:=}0^{n_y}$, $v{:=}0$, and picks t and each d_i as random bitstrings and forms ct^B in m^2_B as encryptions of y, t, u instead of as random strings. $G_3 \approx G_2$ by covertness of encryption cPKE.

Let G_4 be a modification where m^2_B includes the correct sets $\{\mathsf{r}^{\mathsf{gc}}_i\}_{i\in S}$ and $\{\mathsf{ks}^B_i, \{\mathsf{otr}^w_i\}_w\}_{i\notin S}$. $G_4 \approx G_3$ because $\mathsf{r}^{\mathsf{gc}}_i$'s and the wire keys in ks^B_i's are random (given commitments cgc_i and ciphertexts ck_i), and because OT is sender-covert, given that the OTrsp payloads $k^{w,0}_i, k^{w_1}_i$ for $w \in \overline{X}$ are random in G.

The next modification, G_5, computes everything in m^2_B correctly, including the garbled circuits $\{\mathsf{gc}_i\}_{i\notin S}$. By covertness of Yao's garbled circuit generation GCgen (and sender-security of the OT scheme), each circuit gc_i and a set of keys $\mathsf{ks}^B_i, \mathsf{ks}^A_i$ where ks^A_i is implied by $\{\mathsf{otr}^w_i\}_w$, adversary's advantage in distinguishing $(\mathsf{gc}_i, \mathsf{ks}^B_i, \mathsf{ks}^A_i)$ from random is at most negligibly greater than an advantage in distinguishing the circuit output $f|_g(x, c_i, y, d_i, t, u)$ from random. However, $f|_g(x, c_i, y, d_i, t, u) = c_i \oplus d_i$ because $u = 0$, and since each d_i is chosen at random by G_5, these outputs are random strings, hence it follows that $G_5 \approx G_4$.

In the next modification, G_6, we replace $\mathsf{TRec}^{\$(\tau)}$ with TRec in $\mathsf{CKEM}_{\mathsf{LA(pk)}}$, and $G_6 \approx G_5$ by covert zero-knowledge of $\mathsf{CKEM}_{\mathsf{LA(pk)}}$.

Next, in G_7 we will follow algorithm B in step B3 (note that at this point we have all the inputs used in this step, including key K'_A output by TRec in $\mathsf{CKEM}_{\mathsf{LA(pk)}}$), except that G_7 always outputs m_B^3 chosen at random, i.e. even if the t_i's decrypted from m_A^2 satisfy the constraint $t_j \in \{t_j^0, t_j^1\}$ where t_j^0, t_j^1 are parts of t which was chosen in step B2 (note that this is the only case where B would output a non-rejection and form m_B^3 in a non-random way). We have that $G_7 \approx G_6$ by security of the Yao's garbling circuit procedure, because the outputs of $f|_g(x, c_i, y, d_i, t, u)$ in the case $u = 0$ contain no information about t (and neither is t used in any other computation of G_7), the probability that this non-rejection constraint is satisfied is negligible.

In modification, G_8, we replace TRec in $\mathsf{CKEM}_{\mathsf{LB(pk)}}$ with Rec running on witnesses created in steps B1 and B2: Note that at this point B computes all the values in $\mathsf{m}_B^1, \mathsf{m}_B^2$ correctly, except that it picks its circuit inputs $y, \{d_i\}, t, u$ as the simulator $\mathcal{A}^*$ does in the case $\mathsf{F}_{\mathsf{C}(f,g)}$ returns $z = \perp$. We have $G_8 \approx G_7$ by covert zero-knowledge of $\mathsf{CKEM}_{\mathsf{LB(pk)}}$, since covert zero-knowledge implies indistinguishability of an interaction with TRec and Rec *and* the keys K they output.

Next, let G_9 be a modification of G_8 where in step B3 we remove the clause introduced in game G_7 i.e. in game G_9 message m_B^3 is formed as in protocol Π_{COMP} in case t_i's decrypted from m_A^2 satisfy the constraint $t_j \in \{t_j^0, t_j^1\}$ where pairs t_j^0, t_j^1 form string t. However, by the same argument from the security of Yao's garbling procedure, we have that t is indistinguishable to Adv, hence the probability of this clause acting is negligible, and consequently $G_9 \approx G_8$.

Note that G_9 proceeds like the simulator $\mathcal{A}^*$ interacting with $\mathsf{F}_{\mathsf{C}(f,g)}$ and ideal-world honest B who receives $(\mathsf{InputB}, A, \perp, \mathsf{sid})$ from $\mathcal{Z}$: Game G_9 simply assumes that $z = \perp$, but $\mathsf{F}_{\mathsf{C}(f,g)}$ sends $z = \perp$ to $\mathcal{A}^*$ in case the input of the ideal-world player B is $\perp$. Also, in G_9 player B always outputs $\perp$, but this is also the output that the ideal-world player B outputs in interaction with $\mathsf{F}_{\mathsf{C}(f,g)}$ if its computation input is $\perp$.

Case 2A, honest A on input $x^* \in \{0,1\}^{n_x}$: Let G_0 be the interaction of $\mathcal{Z}$ and Adv in the real world where A gets input $(\mathsf{InputA}, B, x^*, \mathsf{sid})$ for $x^* \neq \perp$.

Let G_1 be a modification of G_0, which in step A2 executes as follows: Use sk corresponding to pk in the CRS to decrypt each wire-input ciphertext ct^w in ct^B sent by Adv in m_B^2, and to obtain each bit of values y, t, u which Adv effectively inputs into each garbled circuit computation. Overwrite $y := \perp$ if any decryption output is not a bit or if $u = 0$. Also, do not use GCev to compute $\{w_i\}_{i \notin S}$, but instead, pick each w_i at random if $g(x^*, y) = 0$ or $u = 0$, and otherwise set each w_i to $w = z|t_1|...|t_{n_v}$ where $z = f_z(x^*, y)$ and $t_1, ..., t_{n_v}$ are the authenticator values in t which encode $v = f_v(x^*, y)$. Otherwise G_1 performs all other steps as G_0. $G_1 \approx G_0$ by the security of the Yao's garbling circuit procedure and by the covert simulation-soundness of $\mathsf{CKEM}_{\mathsf{LB(pk)}}$: To see this consider first Adv which implants errors in at least $n/4$ of the circuits, i.e. in $(\mathsf{gc}_i, \{k_i^{w,b}\}_{w,b})$ committed in m_B^1 for all $i \in S'$ for $S' \subset [n]$ s.t. $|S'| \geq n/4$. By the simulation soundness

On behalf of <u>honest A</u>, on trapdoor sk and input $(\mathsf{InputA}, \mathsf{sid}, A, B)$ from $\mathsf{F}_{\mathsf{C}(f,g)}$:

(1) on $\mathsf{m}_B^1 = \{\mathsf{cgc}_i, \{\mathsf{ck}_i^{w,b}\}_{w,b}\}_{i\in[n]}$ from B, set $x{:=}0^{n_x}$ and $c_i \leftarrow \{0,1\}^{n_z+n_v\tau}$ $\forall i$, compute $\mathsf{x}_A{:=}(\{\mathsf{ct}^w\}_{w\in X}, \{\mathsf{ct}_i^w\}_{i,w\in C})$, w_A and message m_A^1 as A does in step A1;

(2) run $\mathsf{Rec}(\pi, (\mathsf{x}_A, \mathsf{w}_A, \ell_A))$ in $\mathsf{CKEM}_{\mathsf{LA}^{(\ell_A)}(\mathsf{pk})}$, let K'_B be Rec's output;

(3) on m_B^2 from B, decrypt it using K'_B to get ct^B, $\{\mathsf{r}_i^{\mathsf{gc}}\}_{i\in S}$, $\{\mathsf{gc}_i, \mathsf{ks}_i^B, \{\mathsf{otr}_i^w\}_{w\in\overline{X}}\}_{i\notin S}$; decrypt each ct^w in ct^B using sk to obtain each bit of y, t, u, overwrite $y{:=}\perp$ if any decryption output is not a bit or $u = 0$; and send $(\mathsf{InputB}, A, y, \mathsf{sid})$ to $\mathsf{F}_{\mathsf{C}(f,g)}$ and receive $(\mathsf{Output}, v, \mathsf{sid})$ back;

(4) compute $(\mathsf{gc}_i, \{k_i^{w,b}\}) \leftarrow \mathsf{GCgen}(f|_g; \mathsf{r}_i^{\mathsf{gc}})$ for $i{\in}S$ to complete statement x_B, and run $\mathsf{Snd}(\pi, (\mathsf{x}_B, \ell_B))$ in $\mathsf{CKEM}_{\mathsf{LB}^{(\ell_B)}(\mathsf{pk})}$ on x_B as in Π_{COMP}, let K_A be Snd's output;

(5) if $v = \perp$ then set $\mathsf{release?}{:=}\mathsf{F}$ and set m_A^2 as a random string, otherwise set $\mathsf{release?}{:=}\mathsf{T}$ and $\mathsf{m}_A^2 \leftarrow \mathsf{SE}^0_{\mathsf{K}_A}(\mathsf{F}(\mathsf{K}'_B, 1), t_1, ..., t_{n_v})$ where t_i's encode v received from $\mathsf{F}_{\mathsf{C}(f,g)}$ given t decrypted from ct^B above; send m_A^2 to B.

(6) given m_B^3, if $\mathsf{m}_B^3 \neq \mathsf{F}(\mathsf{K}_B, 2) \oplus \mathsf{F}(\mathsf{K}'_A, 1)$ then (re)set $\mathsf{release?}{:=}\mathsf{F}$; send $(\mathsf{Output}, \mathsf{sid}, A, \mathsf{release?})$to $\mathsf{F}_{\mathsf{C}(f,g)}$.

Fig. 4. Part 2 of simulator $\mathcal{A}^*$, for Π_{COMP} sessions with honest party A.

of $\mathsf{CKEM}_{\mathsf{LB}(\mathsf{pk})}$ (note that the reduction testing the soundness of $\mathsf{CKEM}_{\mathsf{LB}(\mathsf{pk})}$ must have access to the trapdoor sk corresponding to the public parameters $\pi = (\mathsf{pk}, R)$) and by binding of commitment Com, we have that except for negligible probability key K_A output by this CKEM is indistinguishable for Adv, and therefore in this case $G_1 \approx G_0$ because m_A^2 is pseudorandom to Adv and m_B^3 can satisfy A's non-abort constraint with at most negligible probability. We are left with the case that there are errors in fewer than $n/4$ garbled circuits, in which case for every S the majority of circuits evaluated in step A2 are correct. By a similar argument as above we can discount the case that B forms any ciphertext in ct^B incorrectly or cheats in any other part of messages $\mathsf{m}_B^1, \mathsf{m}_B^2$, because otherwise $G_1 \approx G_0$ because K_A would be pseudorandom to Adv. We are left with the case that keys ks_i^B and OT response vectors otr_i encrypted in m_B^2 are formed correctly for $i \notin S$, in which case for at least $n/4$ indexes $i \notin S$ we have that value w_i which A computes in G_0 is equal to $f|_g(x^*, c_i, y, d_i, t, u)$ for some d_i and y, t, u encrypted in ct^B. Now, note that if $g(x^*, y) = 0$ or $u = 0$ then $w_i = c_i \oplus d_i$. Since each c_i is random and visible to Adv only in ciphertexts in $\{\mathsf{ct}_i^w\}_{w\in C}$, which are non-malleable, hence in particular encryption of y in ct^B cannot be related to c_i's, values w_i are indistinguishable from random in this case. If, on the other hand, $g(x^*, y) = 1$ and $u = 1$ then $w_i = (z|t_1|...|t_{n_v})$ for t_i's which encode v (given t encrypted in ct^B) for $(z, v) = f(x^*, y)$. Finally, note that G_2 forms w_i's exactly the same way in both cases.

Consider a modification G_2 of G_1 which replaces Rec in $\mathsf{CKEM}_{\mathsf{LA}(\mathsf{pk})}$ with TRec, and uses K'_A it outputs in the subsequent steps. We have that $G_2 \approx G_1$ by covert zero-knowledge of $\mathsf{CKEM}_{\mathsf{LA}(\mathsf{pk})}$.

Consider a modification G_3 of G_2 where ciphertexts $\{\mathsf{ct}^w\}_{w\in X}$ in m^1_A are formed by encrypting $x = 0^{n_x}$. Since G_2 doesn't use witnesses in this encryption in $\mathsf{CKEM}_{\mathsf{LA(pk)}}$, it follows that $G_3 \approx G_2$ by the reduction to CCA security of encryption cPKE. (Note that this reduction needs to decrypt the ciphertexts sent by Adv in m^2_B, but that these ciphertexts pertain to different labels than those in m^1_A, as well as the ciphertexts pertaining to sessions with other sid's, but these ciphertexts will also have different labels.)

Note that game G_3 proceeds like the simulator $\mathcal{A}^*$ interacting with $\mathsf{F}_{\mathsf{C}(f,g)}$ and ideal-world honest A who receives $(\mathsf{InputA}, B, x^*, \mathsf{sid})$ from $\mathcal{Z}$ for $x^* \in \{0,1\}^{n_x}$. The difference between G_3 and $\mathcal{A}^*$ are only syntactic: G_3 evaluates $g(x^*, y)$ and $f(x^*, y)$ locally (setting y to $\perp$ if $u = 0$ beforehand), while in the simulation $\mathcal{A}^*$ performs the same $y := 0$ overwrite if $u = 0$, sends y to $\mathsf{F}_{\mathsf{C}(f,g)}$, and $\mathsf{F}_{\mathsf{C}(f,g)}$ evaluates both functions and sends $v = f_v(x^*, y)$ to $\mathcal{A}^*$, but in the end both processes set pair z, v on inputs $(x^*, \{c_i\}_, y, \{d_i\}_, t, u)$ in the same way. Secondly, G_3, case $v \neq \perp$, sets all w_i's for $i \notin S$ as $(z|t_1|...|t_{n_v})$ for t_i's which encode v, but this means that message m^2_A in G_3 is set to a random string if $v = \perp$ and to encryption under K_A of $\max|t_1|...|t_{n_v}$ for $\tau = \mathsf{F}(\mathsf{K}'_B, 1)$ if $v \neq \perp$. Note that $\mathcal{A}^*$ sets m^2_A in exactly the same way. Finally, game G_3 outputs z determined by $f|_g$ or $\perp$ based on the same condition, i.e. whether $\mathsf{m}^3_B = \mathsf{F}(\mathsf{K}'_B, 2) \oplus \mathsf{F}(\mathsf{K}_A, 1)$, which decides whether $\mathsf{F}_{\mathsf{C}(f,g)}$ outputs z or $\perp$ to the dummy ideal-world player A in interaction with $\mathcal{A}^*$.

Case 2B, honest A on input $x^* = \perp$: Let G_0 be the interaction of $\mathcal{Z}$ and Adv in the real world where player A gets input $(\mathsf{InputA}, B, \perp, \mathsf{sid})$ from $\mathcal{Z}$. Note that in this case A is a random beacon, i.e. its messages $\mathsf{m}^1_A, \mathsf{m}^2_A$ are random strings of the appropriate length, and we will assume for notational convenience that A follows $\mathsf{TRec}^{\$(\tau)}$ in $\mathsf{CKEM}_{\mathsf{LA(pk)}}$ and $\mathsf{Snd}^{\$(\tau)}$ in $\mathsf{CKEM}_{\mathsf{LB(pk)}}$. Also A always sends $\perp$ back to $\mathcal{Z}$ as its computation "output".

Let G_1 be a modification of G_0 in which A sets $x{:=}0^{n_x}$ and $c_i \leftarrow \{0,1\}^{n_z + n_v \tau}$ for all i and then forms ciphertexts $\{\mathsf{ct}^w\}_{w\in X}$ and $\{\mathsf{ct}^w_i\}_{i,w\in C}$ in m^1_A by encrypting the bits of x and c_i's as in step (1) of $\mathcal{A}^*$. We have that $G_1 \approx G_0$ by the reduction to CCA security of encryption cPKE. (Note that this reduction will need to decrypt ciphertexts on sessions with other sid's, but these ciphertexts will pertain to different labels.)

Let G_2 be like G_1 but in $\mathsf{CKEM}_{\mathsf{LA(pk)}}$ it runs TRec (but G_2 ignores TRec's local output K'_B). $G_2 \approx G_1$ by covert zero-knowledge of $\mathsf{CKEM}_{\mathsf{LA(pk)}}$.

Let G_3 be a modification of G_2 which in step A2 uses K'_B output by TRec in $\mathsf{CKEM}_{\mathsf{LA(pk)}}$ to decrypt ct^{B}, $\{\mathsf{r}^{\mathsf{gc}}_i\}_{i\in S}$, $\{\mathsf{gc}_i, \mathsf{ks}^B_i, \mathsf{otr}_i\}_{i\notin S}$ from m^2_B, and completes x_B by computing $(\mathsf{gc}_i, \{k^{w,b}_i\}) \leftarrow \mathsf{GCgen}()f, g(\mathsf{r}^{\mathsf{gc}}_i)$ for $i \in S$. This creates no difference in Adv's view.

Let G_4 replace TRec with Rec in $TCKEMCorA$, and uses K'_B it outputs in step A2. $G_4 \approx G_3$ by covert zero-knowledge of $\mathsf{CKEM}_{\mathsf{LA(pk)}}$.

Let G_5 be like G_4 but in $\mathsf{CKEM}_{\mathsf{LB(pk)}}$ it runs $\mathsf{Snd}(\pi, (\mathsf{x}_B, \ell_B))$ on statement x_B computed in step A2. $G_5 \approx G_4$ by covert zero-knowledge of $\mathsf{CKEM}_{\mathsf{LB(pk)}}$.

Note that G_5 proceeds like the simulator $\mathcal{A}^*$ interacting with $\mathsf{F}_{\mathsf{C}(f,g)}$ and ideal-world honest A who receives $(\mathsf{InputA}, B, \perp, \mathsf{sid})$ from $\mathcal{Z}$: Game G_5 is like

$\mathcal{A}^*$ simplified by the assumption that $\mathsf{F}_{\mathsf{C}(f,g)}$ returns $(\mathsf{Output}, v, \mathsf{sid})$ to $\mathcal{A}^*$ for $v = \perp$, in which case m_A^3 is set as random. However, this is indeed the case for A's session which was started by $\mathcal{Z}$ on input $x^* = \perp$. Also, note that in G_5 player A always outputs $\perp$, but this is also the output of the ideal-world player A in interaction with $\mathsf{F}_{\mathsf{C}(f,g)}$ if $x^* = \perp$.

7 Covert CKEM's for *Linear Map Image* Languages

We show two types of covert CKEM's for Linear Map Image languages: In Section 7.1 we show that covert CKEM proposed proposed in [13] for any language with a Σ-protocol, i.e. a 3-round public-coin honest-verifier ZK proof systems satisfying special soundness and zero-knowledge properties, is also covert zero-knowledge and simulation-sound in the Random Oracle Model (ROM). Since all Linear Map Image language have such Σ-protocols, this implies a covert zero-knowledge and simulation-sound CKEM for all such languages in ROM. Then in Section 7.2 we show that the standard-model (assuming CRS) zero-knowledge and simulation-sound CKEM proposed by Benhamouda et al. [2] for any language L in LMI becomes *covert* zero-knowledge (with sender simulation-covertness) and simulation-sound after a simple modification, for any LMI language defined by matrix M with a full row rank. We note that all languages used in Section 6 have full row rank matrices, and their matrix specifications are shown in Appendix B. Note also that if rows of matrix M satisfy a *public* linear constraint then the verifier could reduce the matrix by eliminating linearly dependent rows, but this is the case only if the linear constraint is public.

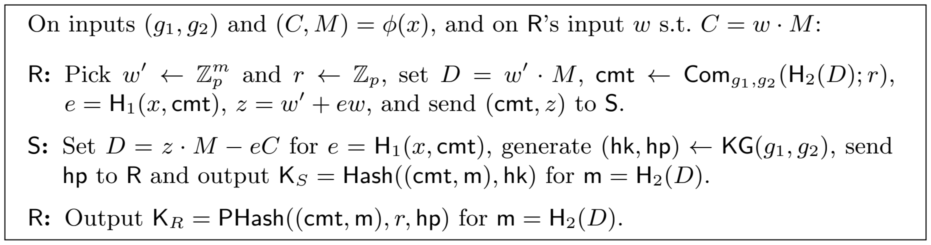
Efficiency Comparison. Both CKEM systems, the ROM-based one in Section 7.1 and the standard-model one in Section 7.2, are two-round protocols, but the standard- model CKEM uses at least twice more computation and bandwidth than the ROM-based one. Specifically, for language $\mathsf{LMI}_{n,m}$, the ROM-based CKEM takes n multi-exp's in (up to) $m+1$ bases for both S and R, and it sends (a covert encoding of) m elements in $\mathbb{Z}_p$ and 3 group elements, while the standard-model CKEM takes $2m+16$ and $2n+14$ multi-exponentiations with (up to) $2n+12$ and $2m+14$ bases respectively for S and R, and it sends (a covert encoding of) $2n+2m+26$ group elements.

7.1 Covert CKEM's in ROM for Languages with Σ-Protocols

A paper on covert mutual authentication [13] used a compiler which converts a Σ-protocol (a three-round public-coin HVZK system) with some additional properties for a given language L into a CKEM for L which satisfied the notion of CKEM covertness defined in [13]. The CKEM covertness of [13] are incomparable to the ones we define here: On one hand, the covert CKEM notion of [13] does not include covert simulation soundness (or even the existence of a trapdoor receiver), which we require for composition of CKEM's within a concurrent secure computation protocol. On the other hand, the covert CKEM notion of [13] includes receiver covertness and *strong* sender covertness, which is a covert counterpart of strong soundness of ZK proofs, i.e. it implies that that an efficient extractor can extract witness w from a receiver who distinguishes interacting with the sender (including the resulting key) from interacting with a random beacon. This stronger covertness property of [2] implies covert soundness of CKEM defined here, but not covert simulation-soundness. However, we show below that the RO-model variant of the CKEM construction of [13] does satisfy covert simulation-soundness as well. Note also that every linear map image language in a prime-order group has the Σ-protocols with the additional properties required in this construction.

In Figure 5 we show the ROM version of the CKEM of [13] for an LMI language. Assume that functions ϕ, γ map instance,witness pairs of language L into instances $x = (C, M) \in G^n \times G^{n \times m}$ and witnesses $w \in \mathbb{Z}_p^m$ of $\mathsf{LMI}_{n,m}$. Recall a Σ-protocol for $\mathsf{LMI}_{n,m}$: The prover picks random $w' \leftarrow \mathbb{Z}_p^m$, sends $D = w' \cdot M$ to the verifier, and on verifier's challenge e chosen uniformly in $\mathbb{Z}_p$, it outputs $z = w' + ew$ (multiplication by a scalar a vector addition in $\mathbb{Z}_p^m$). The verifier accepts if $D = z \cdot M - eC$. Note that if $C = w \cdot M$ then $D = (w' + ew) \cdot M - ew \cdot M = w' \cdot M$. Special soundness follows from the fact that two accepting transcipts (z, e) and (z', e') for $e' \neq e$ imply that $D = z \cdot M - eC = z' \cdot M - e'C$, which means that $C = w \cdot M$ for $w = (e - e')^{-1} \cdot (z - z')$. Special honest-verifier zero-knowledge simulator picks random $z \leftarrow \mathbb{Z}_p^m$ and random $e \leftarrow \mathbb{Z}_p$ and computes $D = z \cdot M - eC$, which perfectly matches the honest verifier's view of the real prover.

Consider a simplified covert commitment of Section 4, namely $\mathsf{Com}_{g_1,g_2}(\mathsf{m})$ picks $r \leftarrow \mathbb{Z}_p$ and outputs $\mathsf{cmt} = (\mathsf{cmt}_1, \mathsf{cmt}_2) = ((g_1)^r, (g_2)^r (g_1)^{\mathsf{m}})$. This commitment is perfectly binding for $\mathsf{m} \in \mathbb{Z}_p$, and it is hiding and covert under the DDH assumption assuming random g_1, g_2. Note that an SPHF for language $\mathsf{Lc} = \{(\mathsf{cmt}, \mathsf{m}) \mid \mathsf{cmt} = \mathsf{Com}_{g_1,g_2}(\mathsf{m})\}$ is a well-known SPHF for the language of Diffie-Hellman tuples, i.e. $\mathsf{KG}(g_1, g_2)$ picks $\mathsf{hk} = (\mathsf{hk}_1, \mathsf{hk}_2) \leftarrow \mathbb{Z}_p^2$ and sets $\mathsf{hp} = (g_1)^{\mathsf{hk}_1}(g_2)^{\mathsf{hk}_2}$, $\mathsf{Hash}((\mathsf{cmt}, \mathsf{m}), \mathsf{hk})$ outputs $(\mathsf{cmt}_1)^{\mathsf{hk}_1}(\mathsf{cmt}_2/(g_1)^{\mathsf{m}})^{\mathsf{hk}_2}$, and $\mathsf{PHash}((\mathsf{cmt}, \mathsf{m}), r, \mathsf{hp})$ outputs $(\mathsf{hp})^r$. Let H be a hash function (modeled as a random oracle in the proof) with range $\mathbb{Z}_p$. Let $\mathsf{H}_i(\cdot)$ stand for $\mathsf{H}(i, \cdot)$.


On inputs (g_1, g_2) and $(C, M) = \phi(x)$, and on R's input w s.t. $C = w \cdot M$:

R: Pick $w' \leftarrow \mathbb{Z}_p^m$ and $r \leftarrow \mathbb{Z}_p$, set $D = w' \cdot M$, $\mathsf{cmt} \leftarrow \mathsf{Com}_{g_1,g_2}(\mathsf{H}_2(D); r)$, $e = \mathsf{H}_1(x, \mathsf{cmt})$, $z = w' + ew$, and send (cmt, z) to S.

S: Set $D = z \cdot M - eC$ for $e = \mathsf{H}_1(x, \mathsf{cmt})$, generate $(\mathsf{hk}, \mathsf{hp}) \leftarrow \mathsf{KG}(g_1, g_2)$, send hp to R and output $\mathsf{K}_S = \mathsf{Hash}((\mathsf{cmt}, \mathsf{m}), \mathsf{hk})$ for $\mathsf{m} = \mathsf{H}_2(D)$.

R: Output $\mathsf{K}_R = \mathsf{PHash}((\mathsf{cmt}, \mathsf{m}), r, \mathsf{hp})$ for $\mathsf{m} = \mathsf{H}_2(D)$.

Fig. 5. ROM version of CKEM of [13] adopted to LMI Languages

Generality and Costs. The cost of the above CKEM is the cost of the underlying Σ-protocol plus 2 exponentiations for R and 3 for S. Note that messages cmt, z

and hp exchanged in this protocol can be covertly encoded at low additional cost (see Section 2) because they are random in domains respectively G^2, $\mathbb{Z}_p^m$, and G. Indeed, the ROM version of the CKEM of [13] is covert zero-knowledge and simulation-sound when applied to any Σ-protocol, and the efficiency of the resulting CKEM is that of the underlying Σ-protocol (plus 2+3 exponentiations) as long as the prover's reply z is random in a domain which is easy to covertly encode. To the best of our knowledge this is the case in all Σ-protocols for various arithmetic relations between group elements, where the prover's response z is either uniform or statistically close to uniform in some integer range.

We argue the security of the above CKEM construction specifically in the case of LMI languages:

Theorem 2. *For any* LMI *language* L*, the CKEM scheme for* L *shown in Figure 5 is covert zero-knowledge and simulation-sound in ROM, assuming DDH.*

Proof. The algorithm TRec uses control of the random oracle H as its simulation *trapdoor* as follows: On input $(C, M) = \phi(x)$ it picks $z \leftarrow \mathbb{Z}_p^m$, $e \leftarrow \mathbb{Z}_p$, and $r \leftarrow \mathbb{Z}_p$, computes $D = z \cdot M - eC$ and $\mathsf{cmt} \leftarrow \mathsf{Com}_{g_1,g_2}(\mathsf{H}_2(D); r)$, sets $\mathsf{H}_1(x, \mathsf{cmt})$ to e, aborting if $\mathsf{H}_1(x, \mathsf{cmt})$ was already set, and sends (cmt, z) to S. When TRec gets hp from S, it can compute $\mathsf{K}_R = \mathsf{PHash}((\mathsf{cmt}, \mathsf{H}_2(D)), r, \mathsf{hp})$ using receiver's algorithm R, because TRec holds the same witness r for the statement that $\mathsf{m} = \mathsf{H}_2(D)$ is committed in cmt for $D = z \cdot M - eC$ and $e = \mathsf{H}_1(x, \mathsf{cmt})$.

Zero Knowledge: The probability that TRec aborts because $\mathsf{H}_1(x, \mathsf{cmt})$ was already queried is negligible because for each m commitment $\mathsf{Com}(\mathsf{m})$ samples a random element from the domain of size $|G|$. Hence, by the argument used for Fiat-Shamir NIZK, the distribution of tuples (cmt, e, z) produced by the simulator TRec is statistically close to that produced by the prover Rec.

Trapdoor-Receiver Covertness: Message (cmt, z) sent by TRec is indistinguishable from a random string because z is uniform in $\mathbb{Z}_p^m$ and Com is a covert commitment under the DDH assumption.

Sender Covertness: S's message $\mathsf{hp} = (g_1)^{\mathsf{hk}_1}(g_2)^{\mathsf{hk}_2}$ is uniform in G.

Covert Simulation-Soundness: The strong (i.e. "proof-of-knowledge") covert soundness shown for this CKEM in [13] implies the standard covert soundness of this CKEM in ROM, i.e. that no efficient adversary interacting with S (and H) on $x \notin \mathsf{L}$ can distinguish S's messages *and* its output K_S from random, i.e. specifically that $(\mathsf{hp}, \mathsf{K}_S)$ pair is statistically close to uniform in G^2. It is easy to see that this remains true given access to a TRec simulator described above, which simulates the $((\mathsf{cmt}, z), \mathsf{K}_R)$ interactions with Rec for any $x' \neq x$: The NIZK challenge is computed using $\mathsf{H}_1(x, \cdot)$ on the soundness challenge x and using $\mathsf{H}_1(x', \cdot)$ on all instances simulated by TRec, and in ROM these are all independent instances of random functions, hence simulation $\mathsf{TRec}(x')$ on any $x' \neq x$ does not affect the view of the interaction with $\mathsf{S}(x)$.

7.2 Covert CKEM for a LMI Languages in CRS

Benhamouda et al. [2] showed a simulation-sound and zero-knowledge CKEM for any LMI language. We argue that a simple modification of their protocol is covert for all the linear map image languages used in the Covert 2PC protocol of Section 6. Specifically, the resulting CKEM is (1) covert zero-knowledge with sender simulation-covertness, and (2) covert simulation-sound, for any LMI language defined by matrix M with a full row rank. Note that matrix M must be full row rank not only on statements in the language but on *any statement* in the implicit universe which includes the statements of this language.

Covert SPHF for LMI Languages. Every LMI language has an efficient SPHF. Let $(C, M) \in G^n \times G^{n\times m}$, let $M = [g_{ij}]_{(i,j)\in[m]\times[n]}$, and let $w \in \mathbb{Z}_p^m$.

> KG picks $\mathsf{hk} \leftarrow \mathbb{Z}_p^n$, and sets $\mathsf{hp} = M \cdot \mathsf{hk}$, i.e. $\mathsf{hp}_i = \prod_{j=1}^n (g_{ij})^{\mathsf{hk}_j}$ for $i \in [m]$;
> $\mathsf{Hash}((C, M), \mathsf{hk})$ outputs $\mathsf{H} = C \cdot \mathsf{hk} = \prod_{j=1}^n (C_j)^{\mathsf{hk}_j}$;
> $\mathsf{PHash}((C, M), w, \mathsf{hp})$ outputs $\mathsf{projH} = w \cdot \mathsf{hp} = \prod_{i=1}^m (\mathsf{hp}_i)^{w_i}$

Note that if $C = w{\cdot}M$ then $\mathsf{H} = C{\cdot}\mathsf{hk} = (w{\cdot}M){\cdot}\mathsf{hk} = w{\cdot}(M{\cdot}\mathsf{hk}) = w{\cdot}\mathsf{hp} = \mathsf{projH}$. Smoothness of this SPHF follows because if $C \notin \mathsf{span}(M)$ then $\mathsf{H} = C \cdot \mathsf{hk}$ is independent from $\mathsf{hp} = M \cdot \mathsf{hk}$. Moreover, if matrix M is *full row rank* then the SPHF is *covert smoothness* because then $\mathsf{hp} = M \cdot \mathsf{hk}$ is random in G^m.

Modified Zero-Knowledge and Simulation-Sound CKEM of [2]. Benhamouda et al. [2] construct a zero-knowledge and simulation-sound CKEM for the same language class. We show our modification of the CKEM [2] starting with a slightly simplified version, where points u'', e'' are globally set in π rather than determined per each (x, ℓ) pair. Let DH be the set of Diffie-Hellman tuples in G and let $\mathsf{DH}(g', h')$ be the set of (u', e') pairs s.t. $(g', h', u', e') \in \mathsf{DH}$. The (simplified) parameter generation $\mathsf{PG}(1^\tau)$ of [2] for language $\mathsf{LMI}_{n,m}$ defines a prime-order group (g, G, p), samples random (g', h', u', e') in G^4, and random (u'', e'') in $\mathsf{DH}(g', h')$, and defines π as $((g, G, p), g', h', u', e', u'', e'')$. The trapdoor generation $\mathsf{TPG}(1^\tau)$ of [2] samples (g', h') in G^2 and both (u', e') and (u'', e'') in $\mathsf{DH}(g', h')$, and sets $\mathsf{td} = \mathsf{DL}(g', u') = \mathsf{DL}(h', e')$. In our modification, PG and TPG are as above except both extend π by additional $n + 6$ elements $h_1, .., h_n, h'_1, ..., h'_6$ chosen at random in G. We *assume that all group elements in* π *are different from* 1. Given statement $(C, M) \in G^n \times G^{n\times m}$ and parameters π, consider an "expansion" of matrix M in $G^{n\times m}$ into matrix M' in $G^{(n+6)\times(m+7)}$ and a "doubling" of M' into matrix $\overline{M}$ in $G^{(2n+12)\times(2m+14)}$ as shown in equation (3). This is exactly as in the original CKEM of [2], except matrix M' of [2] does not have the last row e_7 and the last column of M' in equation (3).

$$\mathbf{M}' = \left[\begin{array}{ccc|c|cc|c}
\multicolumn{3}{c|}{\mathbf{1}} & \mathbf{M} & \multicolumn{2}{c|}{\mathbf{1}} & \mathbf{1} \\ \hline
g' & 1 & 1 & \mathbf{C} & 1 & 1 & 1 \\ \hline\hline
1 & g' & h' & 1\cdots1 & 1 & 1 & 1 \\
g' & u' & e' & 1\cdots1 & 1 & 1 & 1 \\ \hline\hline
1 & 1 & 1 & 1\cdots1 & g' & h' & 1 \\
1 & 1 & 1 & 1\cdots1 & u'' & e'' & 1 \\
g' & 1 & 1 & 1\cdots1 & g' & 1 & 1 \\ \hline\hline
h'_1 & h'_2 & h'_3 & h_1\cdots h_n & h'_4 & h'_5 & h'_6
\end{array}\right]
\begin{array}{l}
(r_1, ..., r_m) \\
(e_1) \\
(e_2) \\
(e_3) \\
(e_4) \\
(e_5) \\
(e_6) \\
(e_7)
\end{array}
\qquad
\overline{\mathbf{M}} = \left[\begin{array}{c|c} \mathbf{M}' & \mathbf{1} \\ \hline \mathbf{1} & \mathbf{M}' \end{array}\right] \qquad (3)$$

The point of this expansion is to modify language L of pairs (C, M) s.t. $C \in \mathsf{span}(M)$ into language L' of triples (C, M, π) s.t. *either* $M \in \mathsf{span}(C)$ *or* elements of π have some trapdoor property. Moreover, language L' is also an LMI language: Let $C' = (g'^{-1}, 1, ..., 1) \in G^{n+6}$. Observe that $C' \in \mathsf{span}(M')$ if and only if

$$\text{(c1)}\ \ C \in \mathsf{span}(M) \quad \text{or} \quad \text{(c2)}\ \ (u', e') \in \mathsf{DH}(g', h') \quad \text{or} \quad \text{(c3)}\ \ (u'', e'') \notin \mathsf{DH}(g', h')$$

Note that row e_7 cannot be used in the linear combination λ' s.t. $C' = \lambda' \cdot M'$ because $h'_6 \neq 1$. Therefore if condition (c1) fails then rows $r_1, ..., r_m, e_1$ cannot be used in this linear combination either. Further, if condition (c2) fails then rows e_2, e_3 must drop out too. Finally, if condition (c3) fails then rows e_4, e_5, e_6 must drop out as well. Hence, the additional conditions (c2) and (c3) allow PG to set π which guarantees that $C' \in \mathsf{span}(M')$ if and only if $C \in \mathsf{span}(M)$, by setting $(u', e') \in \mathsf{DH}(g', h')$ and $(u'', e'') \notin \mathsf{DH}(g', h')$. However, TPG can set π differently to inject the trapdoors which allow for arguing zero-knowledge and simulation-soundness of this CKEM, as we explain below.

Note that if $C = w \cdot M$ then $C' = \lambda_w \cdot M'$ for $\lambda_w = (w, -1, 0, 0, 0, 0, 0, 0)$ and if $(u', e') \in \mathsf{DH}(g', h')$ and $\mathsf{td} = \mathsf{DL}(g', u')$ then $C' = \lambda_{\mathsf{td}} \cdot M'$ for $\lambda_{\mathsf{td}} = (0, ..., 0, \mathsf{td}, -1, 0, 0, 0, 0)$. Figure 6 shows the CKEM of [2] which specifies protocol Rec for R and protocol Snd for S, on respective inputs $\overline{M}$, C', λ_w derived as above. The TRec protocol is exactly like Rec except it uses λ_{td} instead of λ_w.

On π and (C, M) derived from x, and on R's input $\lambda = \lambda_w$ derived from w s.t. $C = w \cdot M$:

R: Pick $\mathsf{tk} \leftarrow \mathbb{Z}_p^{2m+14}$ and send $\mathsf{tp} = (\overline{M}^T) \cdot \mathsf{tk}$ to S;

S: Pick $\mathsf{hk} \leftarrow \mathbb{Z}_p^{2n+12}$ and $\psi \leftarrow \mathbb{Z}_p$, set $\overline{C} = (C', C' \cdot \psi)$, $\mathsf{hp} = \overline{M} \cdot \mathsf{hk}$, $\mathsf{H} = \overline{C} \cdot \mathsf{hk}$, and $\mathsf{tprojH} = \mathsf{hk}^T \cdot \mathsf{tp}$. Send (ψ, hp) to R and output $\mathsf{K}_S = \mathsf{H} \cdot \mathsf{tprojH}$;

R: Set $\bar{\lambda} = (\lambda, \lambda \cdot \psi)$, $\mathsf{projH} = \bar{\lambda}_w \cdot \mathsf{hp}$, and $\mathsf{tH} = \mathsf{hp}^T \cdot \mathsf{tk}$. Output $\mathsf{K}_R = \mathsf{projH} \cdot \mathsf{tH}$.

Fig. 6. CKEM for Linear Map Image Language due to Benhamouda et al. [2]

The CKEM of Figure 6 is correct because it composes two SPHF's for two LMI languages: In the first SPHF, S generates $(\mathsf{hk}, \mathsf{hp})$ to verify that R holds λ s.t. $\overline{C} = \bar{\lambda} \cdot \overline{M}$. In the second SPHF, R generates $(\mathsf{tk}, \mathsf{tp})$ to verify whether S forms hp correctly, i.e. if $\mathsf{hp} = \overline{M} \cdot \mathsf{hk} = \mathsf{hk}^T \cdot \overline{M}^T$ for some hk.

Zero-Knowledge and Simulation-Soundness. We recall the Zero-Knowledge and Simulation-Soundness arguments for this CKEM given by [2], since they apply to the matrix M' as modified here without any changes, and the covertness arguments we supply next will rely on them.

Zero-Knowledge: Zero-Knowledge follows from perfect witness-hiding of an interaction with $\mathsf{Rec}(\lambda, \cdot)$: For any $\bar{\lambda}, \bar{\lambda}'$ s.t. $\overline{C} = \bar{\lambda} \cdot \overline{M} = \bar{\lambda}' \cdot \overline{M}$ it holds that (1) if $\mathsf{hp} = \overline{M} \cdot \mathsf{hk}$ for some hk then $\bar{\lambda} \cdot \mathsf{hp} = \bar{\lambda}' \cdot \mathsf{hp} = \overline{C} \cdot \mathsf{hk}$; and (2) if $\mathsf{hp}^T \notin \mathsf{span}(\overline{M})^T$

then $\mathsf{tH} = \mathsf{hp}^T \cdot \mathsf{tk}$, and hence also K_R, is independent of $\mathsf{hp} = (\overline{M})^T \cdot \mathsf{tk}$ in S's view. Zero-knowledge follows because if $(u', e') \in \mathsf{DH}(g', h')$, as set by TPG, then Rec and TRec run the same algorithm on two valid witnesses λ_w and λ_{td}.

Soundness: If $C \notin \mathsf{span}(M)$ and $(u', e') \notin \mathsf{DH}(g', h')$ and $(u'', e'') \in \mathsf{DH}(g', h')$ (as is set by PG) then conditions (c1)-(c3) all fail and $C' \notin \mathsf{span}(M')$. The point of the "matrix doubling" technique is that $C' = \lambda \cdot M'$ if and only if $(C, C' \cdot \psi) = (\lambda, \lambda \cdot \psi) \cdot \overline{M}$ for every $\psi \neq 0$, and the key lemma of [2] shows that if $C' \notin \mathsf{span}(M')$ then for any tp there exists at most one ψ value s.t. $\overline{C} = (C', C' \cdot \psi) \in \mathsf{span}(\overline{M}, \mathsf{tp}^T)$, where $\mathsf{span}(\overline{M}, \mathsf{tp}^T)$ is the subspace spanned by the rows of $\overline{M}$ *and* vector tp^T. Therefore, if $C' \notin \mathsf{span}(M')$ then except for probability $1/p$ we have that $\overline{C} \notin \mathsf{span}(\overline{M}, \mathsf{tp}^T)$, in which case $\overline{C} + \mathsf{tp}^T$ (a component-wise product of $\overline{C}$ and tp^T) is not in $\mathsf{span}(\overline{M})$, and consequently, using the additive notation for operations in group G, we have that $\mathsf{K}_S = \mathsf{H} + \mathsf{tprojH} = (\overline{C} \cdot \mathsf{hk}) + (\mathsf{tp}^T \cdot \mathsf{hk}) = (\overline{C} + \mathsf{tp}^T) \cdot \mathsf{hk}$ is independent from $\mathsf{hp} = \overline{M} \cdot \mathsf{hk}$.

Simulation-Soundness: Note that soundness and zero-knowledge require opposite property of CRS π: Soundess needs $(u', e') \notin \mathsf{DH}(g', h')$ because otherwise $C' \in \mathsf{span}(M')$ even if $C \notin \mathsf{span}(M)$, while zero-knowledge needs $(u', e') \in \mathsf{DH}(g', h')$ to let TRec compute projH even if $C \notin \mathsf{span}(M)$. This is a problem for simulation-soundness because the set-up needed for the soundness challenge (x, ℓ) is different than the set-up for TRec queries (x', ℓ'). The CKEM of Benhamouda et al. [2] handles this by defining pair (u'', e'') used in matrix M' *separately per each* (x, ℓ) *instance* using the Waters [20] function $W : \{0,1\}^{2\tau} \rightarrow G^2$ and a collision–resistant hash H onto $\{0,1\}^{2\tau}$. Namely, parameter π output by PG includes $2\tau+1$ pairs $(u_i, e_i)_{i=0}^{2\tau}$, all random in $\mathsf{DH}(g', h')$, and pair (u'', e'') for instance (x, ℓ) is defined as $W(\mathsf{H}(x, \ell))$ where $W(m) = (u_0, e_0) \cdot \Pi_{i=1}^{2\tau}(u_i, e_i)^{m_i}$. Simulation-soundness is shown under the DDH assumption by a simulator which picks (u', e') in G^2 and samples (u_i, e_i)'s with knowledge of $\mathsf{DL}(g', u_i)$ and $\mathsf{DL}(h', e_i)$ s.t. with high-enough probability two facts hold: (1) If $(u'', e'') = W(\mathsf{H}(x, \ell))$ for the soundness challenge (x, ℓ) then $(u'', e'') \in \mathsf{DH}(g', h')$, which means that if $x \notin \mathsf{L}$ then none of the conditions (c1)-(c3) above are met, hence the same soundness argument as above implies that Snd's output K_S is independent of the adversary's view; (2) For each (x', ℓ') query to TRec, we have $(u'', e'') = W(\mathsf{H}(x', \ell')) \notin \mathsf{DH}(g', h')$, and the simulator uses the knowledge of $\mathsf{DL}(g', u'')$ and $\mathsf{DL}(h', e'')$ to compute (α, β) s.t. $(g')^\alpha (u'')^\beta = g'$ and $(h')^\alpha (e'')^\beta = 1$ and run Rec on $\lambda'_{\mathsf{td}} = (0, ..., 0, \alpha, \beta, -1, 0)$ s.t. $C' = \lambda'_{\mathsf{td}} \cdot M'$ instead of λ_{td} used by TRec.

Covert Zero-Knowledge and Covert Simulation-Soundness. We argue that under DDH assumption the above CKEM also satisfies (I) *covert zero-knowledge with sender simulation-covertness* and (II) *covert simulation-soundness*. Recall the covertness properties of CKEM as defined in Section 5), and note that given the (standard) zero-knowledge of this CKEM, shown in [2] and recalled above, in order to argue part (I) we need to argue *trapdoor-receiver covertness* and *sender simulation-covertness* of this CKEM. Moreover, given the (standard) simulation-soundness of this CKEM, also shown in [2] and recalled above, by Lemma 1 part (II) will follow from the same sender simulation-soundness property. Below we argue both properties under the DDH assumption. The first property is immediate, the second is more involved.

Theorem 3. *CKEM of Figure 6 is* trapdoor-receiver covert *under the DDH assumption.*

Proof. The only message TRec sends is $\mathsf{tp} = (\overline{M})^T \cdot \mathsf{tk}$ for $\mathsf{tk} \leftarrow \mathbb{Z}_p^{2m+14}$. By the fact that $\overline{M}$ has two copies of M' on the diagonal we can split tk into two parts, $\mathsf{tk}_L, \mathsf{tk}_R$ both random in $\mathbb{Z}_p^{m+7}$, and parse tp as $(\mathsf{tp}_L, \mathsf{tp}_R)$ where $\mathsf{tp}_L = (M')^T \cdot \mathsf{tk}_L$ and $\mathsf{tp}_R = (M')^T \cdot \mathsf{tk}_R$. For notational convenience think of tp and tk as horizontal vectors, i.e. $\mathsf{tp}_L = \mathsf{tk}_L \cdot M'$ and $\mathsf{tp}_R = \mathsf{tk}_R \cdot M'$. Let $M'_{r_1 \ldots e_6}$ be matrix M' with row e_7 removed. Let $h = (h'_1, h'_2, h'_3, h_1, ..., h_m, h'_4, h'_5, h'_6)$ be row e_7 of M'. Let $\mathsf{tk}_L^{[1,\ldots,m+6]}$ be the first $m+6$ elements of tp_L and let α be the last element in tk_L. Note that $\mathsf{tp}_L = \mathsf{tk}_L \cdot M' = A + B$ for $A = \mathsf{tk}^{[1,\ldots,m+6]} \cdot M'_{r_1 \ldots e_6}$, $B = \alpha \cdot h$, where $+$ stands for component-wise group operation in G. Since TPG picks elements in h at random in G, just like PG does, then the DDH assumption implies that vector $B = (h'_1, h'_2, h'_3, h_1, ..., h_m, h'_4, h'_5, h'_6)^\alpha$ is indistinguishable from a random vector in G^{m+6}. (Note that TRec trapdoor td does not need the discrete logarithms of elements in row e_7.) It follows that tp_L is also indistinguishable from a random vector in G^{m+6}, and since the same argument applies to tp_R, the theorem follows.

Theorem 4. *Let L be a language with implicit universe U_x, and let $\phi : U_x \to G^n \times G^{n \times m}$ s.t. $x \in \mathsf{L}$ if and only if $\phi(x) \in \mathsf{LMI}_{n,m}$. CKEM protocol in Figure 6 is* sender simulation-covert *under the DDH assumption for L if for each $x \in U_x$ and $(C, M) = \phi(x)$ matrix M has full row rank m.*

Proof. Our goal is to show that for (π, td) generated by TPG, value $\mathsf{hp} = \overline{M} \cdot \mathsf{hk}$ sent by Snd is indistinguishable from a random vector in $G^{2(m+7)}$ for all x (including $x \notin \mathsf{L}$) by an adversary who is given π and an access to oracle $\mathsf{TRec}_{\mathsf{Block}(x,\ell)}(\mathsf{td}, \cdot)$. (Note that value ψ sent by Snd is uniformly random.) Since $\overline{M}$ is a diagonal matrix containing two instances of M', the SPHF for $\overline{M}$ consists of two independent instances of an SPHF for M', hence we need only consider how the SPHF acts on M', i.e. consider $\mathsf{hp} = (\mathsf{hp}_1, ..., \mathsf{hp}_{m+7}) = M' \cdot \mathsf{hk}$ for $\mathsf{hk} \leftarrow \mathbb{Z}_p^{n+6}$, and we need to argue that it is indistinguishable from random in G^{m+7} given access to oracle $\mathsf{TRec}_{\mathsf{Block}(x,\ell)}(\mathsf{td}, \cdot)$. Let $(C, M) = \phi(x)$. First, note that since the last row of M' is independent from the others, element hp_{m+7} is random in G and independent from elements $\mathsf{hp}_{[1,\ldots,m+6]}$. Thus we can consider only the reduced form of matrix M', with the last row and column eliminated, i.e. the original matrix M' used by [2]. Since by assumption M has rank m, and row e_1 has $g' \neq 1$ in the first column, space $S_1 = \mathsf{span}(r_1, ..., r_m, e_1)$ has dimension $m+1$, which implies that $\mathsf{hp}_{[1,m+1]}$ is random in G^{m+1}.

$$\mathbf{F} = \left[\begin{array}{c} 1 \\ \hline g' \\ \hline\hline 1 \\ \hline 1 \\ \hline g' \end{array}\right] \qquad \mathbf{M}^* = \left[\begin{array}{cc|cc} g' & h' & 1 & 1 \\ \hline u' & e' & 1 & 1 \\ \hline\hline 1 & 1 & g' & h' \\ \hline 1 & 1 & u'' & e'' \\ \hline 1 & 1 & g' & 1 \end{array}\right] \begin{array}{c} (e_2^*) \\ (e_3^*) \\ (e_4^*) \\ (e_5^*) \\ (e_6^*) \end{array} \tag{4}$$

Consider matrix M^* made of the 2nd, 3rd, $(n+4)$th, and $(n+5)$th column of rows e_2-e_6 of M', and let F contain the entries in the first column of M' in the same rows, as shown in equation (4). Consider vector $\mathsf{hp}^* = M^* \cdot \mathsf{hk}^*$ for $\mathsf{hk}^* = (\mathsf{hk}_{2,3}, \mathsf{hk}_{n+4,n+5})$. Observe by inspection of matrix M' in equation (3) that (1) $\mathsf{hp}_{[1,m+1]}$ is a function only of elements $(\mathsf{hk}_1, \mathsf{hk}_{[4,..,n+3]})$ of hk, and (2) $\mathsf{hp}_{[m+2,m+6]} = F \cdot \mathsf{hk}_1 + M^* \cdot \mathsf{hk}^* = F \cdot \mathsf{hk}_1 + \mathsf{hp}^*$. By fact (1) it follows that $\mathsf{hp}_{[1,m+1]}$ is independent of hk^*, because hk^* is independent of $(\mathsf{hk}_1, \mathsf{hk}_{[4,..,n+3]})$. By fact (2) it follows that if we show that $\mathsf{hp}^* = M^* \cdot \mathsf{hk}^*$ is indistinguishable from a random tuple in G^5 for $\mathsf{hk}^* \leftarrow G^4$ (given access to the TRec oracle) then this will imply that $\mathsf{hp}_{[m+2,m+6]}$ is also indistinguishable from random in G^5 because hp^* acts like a one-time pad in the above equation for $\mathsf{hp}_{[m+2,m+6]}$. Together with the fact that $\mathsf{hp}_{[1,m+1]}$ is random in G^{m+1}, this will complete the argument for sender simulation covertness.

It remains to argue that for $(\pi, \mathsf{td}) \leftarrow \mathsf{TPG}(1^\tau)$ and any (x, ℓ) output by adversary $\mathcal{A}$ given π and access to oracle $\mathsf{TRec}(\mathsf{td}, \cdot)$, if M^* is defined as above by $(u'', e'') = W(\mathsf{H}(x, \ell))$, then variable $\{\mathsf{hp}^* = M^* \cdot \mathsf{hk}^*\}_{\mathsf{hk}^* \leftarrow G^4}$ is indistinguishable from a random tuple in G^5, given $\mathcal{A}$'s access to oracle $\mathsf{TRec}_{\mathsf{Block}(x,\ell)}(\mathsf{td}, \cdot)$. This argument uses similar game changes as the simulation-soundness argument of [2] for this CKEM recalled above. First, consider the modified TPG which sets all (u_i, e_i) pairs at random instead of sampling them from $\mathsf{DH}(g', h')$. By DDH this modified TPG presents an indistinguishable view from the original. Now consider a TPG which sets $(u_i, e_i) = ((g')^{\delta_i}, (h')^{\zeta_i})$ for random $(\delta_i, \zeta_i) \leftarrow \mathbb{Z}_p^2$ for each $i = 0, ..., 2\tau$. This modification does not change the adversary's view. Note that except for negligible probability it holds that for every TRec query (x', ℓ') we have $(u'', e'') \notin \mathsf{DH}(g', h')$ for $(u'', e'') = W(m) = (u_0, e_0) \cdot \Pi_{i=1}^{2\tau} (u_i, e_i)^{m_i}$ where $m = \mathsf{H}(x', \ell')$. Therefore the modified TRec can use (δ_i, ζ_i)'s to compute (α, β) s.t. $(g')^\alpha (u'')^\beta = g'$ and $(h')^\alpha (e'')^\beta = 1$, in which case $\lambda'_{\mathsf{td}} = (0, ..., 0, \alpha, \beta, -1, 0)$ satisfies that $C' = \lambda'_{\mathsf{td}} \cdot M'$, and by argument that $\mathsf{TRec}(\mathsf{td}, \cdot)$ oracle is witness-hiding (see the zero-knowledge argument above), we have that this modification also does not change the adversary's view. Since this modification of TRec does not use the original trapdoor $\mathsf{td} = \mathsf{DL}(g', u') = \mathsf{DL}(h', e')$, by reduction to DDH we can modify TPG further by sampling (u', e') at random from G^2 instead of from $\mathsf{DH}(g', h')$. After this change rows (e_1^*, e_2^*) of M^* are independent (except for negligible probability), hence $\mathsf{hp}^*_{1,2}$, which is a function of only $\mathsf{hk}^*_{1,2}$, can now be replaced by a random pair in G^2.

It remains to argue that $\mathsf{hp}^*_{[3-5]}$ is indistinguishable from random in G^3 for random $\mathsf{hk}^*_{3,4} \leftarrow \mathbb{Z}_p^2$. Denote $(r, s) = (\mathsf{hk}^*_3, \mathsf{hk}^*_4)$, and observe that $\mathsf{hp}^*_{3,4,5} = ((g')^r (h')^s, (u'')^r (e'')^s, (g')^r)$. Let us now move back from the above modifications of $\mathsf{TPG}, \mathsf{TRec}$ to the original $\mathsf{TPG}, \mathsf{TRec}$: First we change back (u', e') distribution from G^2 to $\mathsf{DH}(g', h')$, which is indistinguishable under DDH. Then we replace the modified TRec with the original, which uses $\mathsf{td} = \mathsf{DL}(g', u') = \mathsf{DL}(h', e')$ instead of $\mathsf{td} = \{\delta_i, \zeta_i\}_{i=0,..,2\tau}$ for $(\delta_i, \zeta_i) = (\mathsf{DL}(g', u_i), \mathsf{DL}(h', e_i))$. This does not change the view by the witness-hiding property of $\mathsf{TRec}(\mathsf{td}, \cdot)$ used above. At this point we can modify TPG to sample e_0 at random from G and pick each e_i for $i > 0$ as $e_i = (h')^{\zeta_i}$ for $\zeta_i \leftarrow \mathbb{Z}_p$. Note that for any $m = \mathsf{H}(x, \ell)$ it holds that $e'' =$

$e_0(h')^\zeta$ where $\zeta = \sum_{i=1}^{2\tau}(\zeta_i)^{m_i}$, and therefore $(e'')^s = ((h')^s)^\zeta(e_0)^s$. We can use this relation to replace pair $(h')^s, (e_0)^s$ used in computing $\mathsf{hp}^*_{3,4,5}$ with a random pair in G, and by reduction to DDH the resulting view is indistinguishable. Note that after this change $\mathsf{hp}^*_{3,4,5} = ((g')^r g_1\,,\,(u'')^r(g_1)^\zeta g_2\,,\,(g')^r$ for $g_1, g_2 \leftarrow G^2$ and $r \leftarrow \mathbb{Z}_p$, hence it is uniform in G^3, which completes the proof.

Corollary 1. *Let* L *be a language with implicit universe* U_x*, and let* $\phi : U_x \rightarrow G^n \times G^{n\times m}$ *s.t.* $x \in \mathsf{L}$ *if and only if* $\phi(x) \in \mathsf{LMI}_{n,m}$*. CKEM protocol in Figure 6 is* covert zero-knowledge with sender simulation-covertness *and* covert simulation-sound *under the DDH assumption for* L *if for each* $x \in U_x$ *and* $(C, M) = \phi(x)$ *matrix* M *has full row rank* m*.*

Proof. The corollary follows from theorems 3 and 4, lemma 1 of Section 5, and standard, i.e. non-covert, zero-knowledge and simulation-soundness of this CKEM shown in [2].

References

1. W. Aiello, Y. Ishai, and O. Reingold. Priced oblivious transfer: How to sell digital goods. In *Advances in Cryptology - EUROCRYPT 2001, International Conference on the Theory and Application of Cryptographic Techniques, Innsbruck, Austria, May 6-10, 2001, Proceeding*, pages 119–135, 2001.
2. F. Benhamouda, G. Couteau, D. Pointcheval, and H. Wee. Implicit zero-knowledge arguments and applications to the malicious setting. In *Advances in Cryptology - CRYPTO 2015 - 35th Annual Cryptology Conference, Santa Barbara, CA, USA, August 16-20, 2015, Proceedings, Part II*, pages 107–129, 2015.
3. R. Canetti. Universally composable security: A new paradigm for cryptographic protocols. In *Proceedings of the 42Nd IEEE Symposium on Foundations of Computer Science*, FOCS '01, pages 136–, Washington, DC, USA, 2001. IEEE Computer Society.
4. N. Chandran, V. Goyal, R. Ostrovsky, and A. Sahai. Covert multi-party computation. In *FOCS*, pages 238–248, 2007.
5. C. Cho, D. Dachman-Soled, and S. Jarecki. Efficient concurrent covert computation of string equality and set intersection. In *Topics in Cryptology - CT-RSA 2016 - The Cryptographers' Track at the RSA Conference 2016, San Francisco, CA, USA, February 29 - March 4, 2016, Proceedings*, pages 164–179, 2016.
6. G. Couteau. Revisiting covert multiparty computation. Cryptology ePrint Archive, Report 2016/951, 2016. http://eprint.iacr.org/2016/951.
7. R. Cramer and V. Shoup. Universal hash proofs and and a paradigm for adaptive chosen ciphertext secure public-key encryption. *Electronic Colloquium on Computational Complexity (ECCC)*, 8(072), 2001.
8. G. D. Crescenzo, R. Ostrovsky, and S. Rajagopalan. Conditional oblivious transfer and timed-release encryption. In *Advances in Cryptology - EUROCRYPT '99, International Conference on the Theory and Application of Cryptographic Techniques, Prague, Czech Republic, May 2-6, 1999, Proceeding*, pages 74–89, 1999.
9. G. D. Crescenzo, R. Ostrovsky, and S. Rajagopalan. Conditional oblivious transfer and timed-release encryption. In *EUROCRYPT*, pages 74–89, 1999.

10. V. Goyal and A. Jain. On the round complexity of covert computation. In *Proceedings of the Forty-second ACM Symposium on Theory of Computing*, STOC '10, pages 191–200, New York, NY, USA, 2010. ACM.
11. N. J. Hopper, L. von Ahn, and J. Langford. Provably secure steganography. *IEEE Trans. Computers*, 58(5):662–676, 2009.
12. Y. Huang, J. Katz, and D. Evans. Efficient secure two-party computation using symmetric cut-and-choose. In *Advances in Cryptology - CRYPTO 2013 - 33rd Annual Cryptology Conference, Santa Barbara, CA, USA, August 18-22, 2013. Proceedings, Part II*, pages 18–35, 2013.
13. S. Jarecki. Practical covert authentication. In H. Krawczyk, editor, *Public-Key Cryptography PKC 2014*, volume 8383 of *Lecture Notes in Computer Science*, pages 611–629. Springer Berlin Heidelberg, 2014.
14. J. Katz and V. Vaikuntanathan. Round-optimal password-based authenticated key exchange. In Y. Ishai, editor, *Theory of Cryptography*, volume 6597 of *Lecture Notes in Computer Science*, pages 293–310. Springer Berlin Heidelberg, 2011.
15. Y. Lindell. Fast cut-and-choose-based protocols for malicious and covert adversaries. *J. Cryptology*, 29(2):456–490, 2016.
16. Y. Lindell and B. Pinkas. An efficient protocol for secure two-party computation in the presence of malicious adversaries. *J. Cryptology*, 28(2):312–350, 2015.
17. M. Naor and B. Pinkas. Computationally secure oblivious transfer. *J. Cryptology*, 18(1):1–35, 2005.
18. M. D. Raimondo and R. Gennaro. Provably secure threshold password-authenticated key exchange. *J. Comput. Syst. Sci.*, 72(6):978–1001, 2006.
19. L. von Ahn, N. Hopper, and J. Langford. Covert two-party computation. In *Proceedings of the Thirty-seventh Annual ACM Symposium on Theory of Computing*, STOC '05, pages 513–522, New York, NY, USA, 2005. ACM.
20. B. Waters. Efficient identity-based encryption without random oracles. In *Advances in Cryptology - EUROCRYPT 2005, 24th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Aarhus, Denmark, May 22-26, 2005, Proceedings*, pages 114–127, 2005.

A Covertness of Cramer-Shoup Encryption

Proof of CCA-Covertness of Cramer-Shoup PKE. Recall the Cramer-Shoup encryption in Section 4, and recall its proof of CCA security given in [7]. Consider a (purely syntactic) restriction on an attacker $\mathcal{A}$ in the standard CCA-security game which specifies only one challenge message m^* and the encryption challenge pair is formed as $m_1 \leftarrow m^*$ and $m_0 \leftarrow G$. Recall the CCA-security proof of [7]: It shows a simulator $\mathcal{S}$ which on input a four-tuple $T = (g_1, g_2, u_1, u_2)$ of G elements interacts with attacker $\mathcal{A}$ in such a way that if $T \leftarrow \mathsf{DH}$, i.e. if (g_1, g_2, u_1, u_2) is a random Diffie-Hellman tuple, then for any choice of challengers bit b, $\mathcal{A}$'s view of real execution (denoted $\mathsf{R}(b)$) is statistically indistinguishable from $\mathcal{A}$'s view of an interaction with $\mathcal{S}$ (denoted $\mathcal{S}(b, T)$). Simulator $\mathcal{S}$ on input (b, T) for $T = (g_1, g_2, u_1, u_2)$, uses a different representation of the private key by picking $\mathsf{sk}' = (x_1, x_2, y_1, y_2, z_1, z_2) \leftarrow \mathbb{Z}_p^6$ and setting $(c, d, h) \leftarrow (g_1^{x_1} g_2^{x_2}, g_1^{y_1} g_2^{y_2}, g_1^{z_1} g_2^{z_2})$, (note that h is set differently from the real execution). Furthermore, $\mathcal{S}$ forms the challenge ciphertext $\mathsf{ct}^* = (u_1, u_2, e, v)$ using (u_1, u_2) from its input 4-tuple, and setting $e = u_1^{z_1} u_2^{z_2} m_b$ and $v = u_1^{x_1 + y_1 \xi} u_2^{x_2 + y_2 \xi}$

for $\xi = \mathsf{H}(\ell, u_1, u_2, e)$. Furthermore, $\mathcal{S}(b, T)$ decrypts $\mathcal{A}$'s ciphertext queries $(\bar{\mathsf{ct}}, \bar{\ell})$ for $\bar{\mathsf{ct}} = (\bar{u}_1, \bar{u}_2, \bar{e}, \bar{v})$ slightly differently than in $\mathsf{R}(b)$: It decrypts only if $\bar{v} = \bar{u}_1^{x_1+y_1\bar{\xi}} \bar{u}_2^{x_2+y_2\bar{\xi}}$ for $\bar{\xi} = \mathsf{H}(\bar{\ell}, \bar{u}_1, \bar{u}_2, \bar{e})$, as in $\mathsf{R}b$, but it forms the decrypted plaintext as $\bar{e} \cdot \bar{u}_1^{z_1} \bar{u}_2^{z_2}$. The proof in [7] shows that $\mathcal{A}$'s view of $\mathcal{S}(1, T)$ and $\mathcal{S}(0, T)$ are statistically close if $T \leftarrow G^4$. Moreover DDH implies that $\mathcal{A}$'s view of $\mathcal{S}(b, T)$ for $T \leftarrow \mathsf{DH}$ is indistinguishable from $\mathcal{A}$'s view of $\mathcal{S}(b, T)$ for $T \leftarrow G^4$.

In particular it follows that $\mathcal{A}$'s view of $\mathsf{R}(1)$, where $\mathsf{ct}^* = \mathsf{E}(\mathsf{pk}, m^*, \ell^*)$, is indistinguishable from $\mathcal{A}$'s view of $\mathcal{S}(0, T)$ for $T \leftarrow G^4$. Note that if $b = 0$ then $\mathcal{S}$ can pick $e \leftarrow G$ and the view does not change because $m_0 \leftarrow G$ in the original game. Consider Sim_1 which picks $g_1, u_1, u_2 \leftarrow G^3$ and $w \leftarrow \mathbb{Z}_p$, sets $g_2 \leftarrow (g_1)^w$, and then runs $\mathcal{S}(g_1, g_2, u_1, u_2)$. Clearly, the view of $\mathcal{S}_1$ is identical to the view of $\mathcal{S}(b = 0, T)$ for $T \leftarrow G^4$. Consider $\mathcal{S}_2$ which proceeds as $\mathcal{S}_1$ except the decryption oracle is modified to decrypt $(\bar{\mathsf{ct}}, \bar{\ell})$ query, for $\bar{c} = (\bar{u}_1, \bar{u}_2, \bar{e}, \bar{v})$, only if (1) $\bar{u}_2 = (\bar{u}_1)^w$ and (2) $\bar{v} = \bar{u}_1^{x_1+y_1\bar{\xi}} \bar{u}_2^{x_2+y_2\bar{\xi}}$ for $\bar{\xi} = \mathsf{H}(\bar{\ell}, \bar{u}_1, \bar{u}_2, \bar{e})$. The CCA-security proof in [7] uses a crucial technical lemma that, except for a negligible probability, the simulator $\mathcal{S}$ (as well as the real decryptor) rejects a decryption query unless $(g_1, g_2, \bar{u}_1, \bar{u}_2) \in \mathsf{DH}$. It follows that view presented by $\mathcal{S}_2$ is statistically close to that of $\mathcal{S}_1$, and hence also indistinguishable from the view of the real interaction on $\mathcal{A}$'s challenge plaintext m^*.

Note that values (u_1, u_2, e) in the challenge ciphertext ct^* in produced by $\mathcal{S}_2$ are three random elements of G. To see that the value v in ct^* is also random in G, examine the information which an all-powerful observer (who can compute discrete logs) sees about values (x_1, x_2, y_1, y_2) used by $\mathcal{S}_2$. Let r_1, r_2 be such that that $(u_1, u_2) = (g_1^{r_1}, g_2^{r_2})$. Note that values (g_1, g_2, u_1, u_2) determine (w, r_1, r_2), values c, d determine $\bar{x} = x_1 + wx_2$ and $\bar{y} = y_1 + wy_2$, value v determines $\alpha = r_1(x_1 + \xi y_1) + wr_2(x_2 + \xi y_2)$, and the values which the decryption oracle uses are w and (since $\bar{u}_2 = (\bar{u}_2)^w$) $\beta = (x_1 + \bar{\xi} y_1) + w(x_2 + \bar{\xi} y_2)$. However, note that $\beta = (x_1 + wx_2) + \bar{\xi}(y_1 + wy_2) = \bar{x} + \bar{\xi}\bar{y}$, which means that β does not reveal any more information than $\bar{x}, \bar{y}$. Note that $(\bar{x}, \bar{y}, \alpha)$ are computed as follows:

$$\begin{bmatrix} \bar{x} \\ \bar{y} \\ \alpha \end{bmatrix} = \begin{bmatrix} 1 & w & 0 & 0 \\ 0 & 0 & 1 & w \\ r_1 & wr_2 & r_1\xi & wr_2\xi \end{bmatrix} \cdot \begin{bmatrix} x_1 \\ x_2 \\ y_1 \\ y_2 \end{bmatrix}$$

If $r_1 \neq r_2$ then the rows in the above matrix are linearly independent, and in that case $(\bar{x}, \bar{y}, \alpha)$ are uniform in $\mathbb{Z}_p^3$ for $(x_1, x_2, y_1, y_2) \leftarrow \mathbb{Z}_p^4$. It follows that ct^* produced by $\mathcal{S}_2$ is statistically close to uniform in $\mathbb{Z}_p^4$ (over the choice of r_1, r_2 determined by u_1, u_2).

Now we can "move backwards" and modify simulation $\mathcal{S}_2$ so that it looks like the real execution except $\mathsf{ct}^* \leftarrow \mathbb{Z}_p^4$. Consider $\mathcal{S}_3$ which chooses $z \leftarrow \mathbb{Z}_p$ instead of (z_1, z_2) sets $h = g_1^z$ and decrypts as $\bar{e}/(\bar{u}_1)^z$. By equation (1) in the decryption test, and since $z = z_1 + wz_2$, we can see that $\mathcal{S}_3$ presents an identical view to $\mathcal{S}_2$. Finally, consider $\mathcal{S}_4$ which picks $(g_1, g_2, u_1, u_2, e, v)$ at random in G^6, and therefore does not know w, and uses only condition (2) above in the decryption test, as in the real Cramer-Shoup decryption. By the same technical lemma above the view produced by $\mathcal{S}_4$ is statistically close to that of $\mathcal{S}_3$. Since $\mathcal{S}_4$ now acts like the CCA challenger except the ciphertext challenge ct^* is sampled uniformly in G^4, this concludes the proof.

B Linear Map Image Languages for Covert 2PC

We list the LMI languages used in the covert 2PC protocol of Section 6, and we explain why these languages are in the LMI class by specifying the mapping between the language instance x and the (C, M) pair which defines the instance of LMI. Let $(\mathsf{Kg}, \mathsf{E}, \mathsf{D})$ be the CCA-covert Cramer-Shoup PKE. All languages below are implicitly parametrized by the public key $\mathsf{pk}((g, G, p, \mathsf{H}), g_1, g_2, c, d, h)$ output by $\mathsf{Kg}(1^\tau)$. Formally, the public key pk and any other parameter like a label ℓ will be part of the language statement in each language below.

Encryption Correctness. Language Le contains correct (ciphertext, plaintext) tuples, i.e.

$$\mathsf{Le}^\ell(\mathsf{pk}) = \{(\mathsf{ct}, \mathsf{m}) \text{ s.t. } \mathsf{ct} \in \mathsf{E}^\ell_{\mathsf{pk}}(\mathsf{m})\}$$

$\mathsf{Le}^\ell(\mathsf{pk})$ statements can be mapped onto statements in $\mathsf{LMI}_{4,1}$, because $\mathsf{ct} = \mathsf{E}^\ell_{\mathsf{pk}}(\mathsf{m}; r)$ for $\mathsf{ct} = (u_1, u_2, e, v)$ and some $r \in \mathbb{Z}_p$ holds if and only if $C = r \cdot M$ for C, M as shown in equation (5), for $\xi = \mathsf{H}(\ell, u_1, u_2, e)$.

$$\mathbf{C} = [u_1, u_2, e/\mathsf{m}, v] \quad \mathbf{M} = [g_1, g_2, h, cd^\xi] \tag{5}$$

Encryption of a Bit. Another example is language $\mathsf{Lbit}^\ell(\mathsf{pk})$ of (shifted) encryptions of a bit, i.e.

$$\mathsf{Lbit}^\ell(\mathsf{pk}) = \{(\mathsf{ct}, \ell) \text{ s.t. } \exists b \ (\mathsf{ct}, g^b) \in \mathsf{Le}^\ell(\mathsf{pk}) \wedge b \in \{0, 1\}\}$$

This language can be expressed using only arithmetic constraints, i.e. without resorting to disjunctions. The above constraints can be restated as $(u_1, u_2, e, v) = ((g_1)^r, (g_2)^r, h^r g^b, (cd^\xi)^r)$ and $b(b-1) = 0$ for some $r, b \in \mathbb{Z}_p$. However, the second constraint can be expressed as the constraint that $1 = (u_1)^b (g_1)^\lambda$ and $1 = (e/g)^b h^\lambda$ for some $\lambda \in \mathbb{Z}_p$. This is because if $u_1 = (g_1)^r$ then $(u_1)^b (g_1)^\lambda = g_1^{rb+\lambda}$ so the first constraint is equivalent to $rb + \lambda = 0$, and if $e = h^r g^b$ then $(e/g)^b h^\lambda = h^{rb+\lambda} g^{(b-1)b}$, hence the two constraints imply that $(b-1)b = 0$. Therefore $\mathsf{Lbit}^\ell(\mathsf{pk})$ statements can be mapped onto statements (C, M) of $\mathsf{LMI}_{6,3}$ as shown in equation (6), for $\xi = \mathsf{H}(\ell, u_1, u_2, e)$, with witness (r, b) mapped onto vector $w = (r, b, -rb)$.

$$\mathbf{C} = \begin{bmatrix} u_1 & u_2 & e & v & 1 & 1 \end{bmatrix} \quad \mathbf{M} = \begin{bmatrix} g_1 & g_2 & h & (cd^\xi) & 1 & 1 \\ 1 & 1 & g & 1 & u_1 & e/g \\ 1 & 1 & 1 & 1 & g_1 & h \end{bmatrix} \tag{6}$$

Encryption Disjunction. Language Ldis contains tuples $(\mathsf{ct}, \mathsf{m}, \mathsf{ck}_0, \mathsf{ck}_1)$ s.t. *either* $\mathsf{ck}_0 = \mathsf{E}(\mathsf{pk}, g^{\mathsf{m}})$ and b=0 *or* $\mathsf{ck}_1 = \mathsf{E}(\mathsf{pk}, g^{\mathsf{m}})$ and b=1, for $\mathsf{ct} = \mathsf{E}(\mathsf{pk}, g^b)$.

$$\mathsf{Ldis}^{\ell,\ell_0,\ell_1}(\mathsf{pk}) = \{(\mathsf{ct}, \mathsf{m}, \mathsf{ck}_0, \mathsf{ck}_1) \text{ s.t. } \exists b \in \{0,1\} \ (\mathsf{ct}, g^b) \in \mathsf{Le}^\ell(\mathsf{pk}) \wedge (\mathsf{ck}_b, g^{\mathsf{m}}) \in \mathsf{Le}^{\ell_b}(\mathsf{pk})\}$$

One can define this language using a disjunction but it can also be expressed directly using a set of linear constraints. First we have the encryption constraints, i.e. that there exists $b, \mathsf{m}_0, \mathsf{m}_1, r, r_0, r_1$ s.t.

$$\begin{aligned} \mathsf{ct} = \ (u_1, u_2, e, v) \ &= ((g_1)^r, (g_2)^r, h^r g^b, (cd^{\xi})^r) \\ \mathsf{ck}_0 = (u_1^0, u_2^0, e^0, v^0) &= ((g_1)^{r_0}, (g_2)^{r_0}, h^{r_0} g^{m_0}, (cd^{\xi_0})^{r_0}) \\ \mathsf{ck}_1 = (u_1^1, u_2^1, e^1, v^1) &= ((g_1)^{r_1}, (g_2)^{r_1}, h^{r_1} g^{m_1}, (cd^{\xi_1})^{r_1}) \end{aligned}$$

for appropriately computed hash values ξ, ξ_0, ξ_1. Secondly, we have a constraint that $0 = b(b-1)$, which can be expressed as in Lbit, i.e. with constraints that $1 = (u_1)^b (g_1)^{\lambda}$ and $1 = (e/g)^b h^{\lambda}$ for some λ. Third, we have a constraint that there exists m' s.t. $m_0 = m - b \cdot (m - m')$ and $m_1 = m' + b \cdot (m - m')$, which guarantees that $m = m_b$. This can be restated as there exist s, δ, m' s.t. $s = b \cdot \delta$, $\delta = m - m'$, $m_0 = m - s$, and $m_1 = m' + s$. The last three constraints can be expressed as $g^m = g^{\delta} g^{m'}$, $g^m = g^{m_0} g^s$, and $1 = g^{m_1} (g^{-1})^{m'} (g^{-1})^s$, while constraint $s = b \cdot \delta$ can be expressed using similar indirection as in the case of constraint $0 = b(b-1)$, namely that $1 = e^{\delta} h^{\gamma} (g^{-1})^s$ and $1 = (u_1)^{\delta} (g_1)^{\gamma}$ for $\gamma = -r\delta$. Thus we map a statement in Ldis onto statement (C, M) in $\mathsf{LMI}_{19,11}$ as shown in equation (7), where $f = cd^{\xi}$ for $\xi = \mathsf{H}(\ell, u_1, u_2, e)$ and $f_b = cd^{\xi_b}$ for $\xi_b = \mathsf{H}(\ell_b, u_1^b, u_2^b, e^b)$ for $b = 0, 1$. The witness (r, b, r_0, r_1) is mapped onto vector $w = (r, b, -rb, r_0, m_0, r_1, m_1, m', s, \delta, -r\delta)$, where (m', s, δ) is set to $(m_1, 0, m_0 - m_1)$ if $(m{=}m_0) \wedge (b{=}0)$, and to $(m_0, m_1 - m_0, m_1 - m_0)$ if $(m{=}m_1) \wedge (b{=}1)$.

$$\mathbf{M} = \left[\begin{array}{cc|ccc|cc|cccc|cccc|cccc} 1 & 1 & 1 & 1 & 1 & 1 & 1 & g_1 & g_2 & h & f & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 & u_1 & e/g & 1 & 1 & g & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 & g_1 & h & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \\ \hline 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & g_1 & g_2 & h & f_0 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & g & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & g & 1 & 1 & 1 & 1 & 1 \\ \hline 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & g_1 & g_2 & h & f_1 \\ 1 & 1 & g^{-1} & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & g & 1 \\ \hline 1 & 1 & g & 1 & g & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \\ 1 & g^{-1} & g & g & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \\ u_1 & e & 1 & 1 & g & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \\ g_1 & h & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \end{array}\right] \qquad (7)$$

$$\mathbf{C} = \left[\, 1 \ 1 \,\middle|\, 1 \ g^m \ g^m \,\middle|\, 1 \ 1 \,\middle|\, u_1 \ u_2 \ e \ v \,\middle|\, u_1^0 \ u_2^0 \ e^0 \ v^0 \,\middle|\, u_1^1 \ u_2^1 \ e^1 \ v^1 \,\right]$$

Correct OT Reponse. Another LMI language used in covert 2PC protocol is Lotr, which involves verification that the response message OTrsp in the OT of Aiello et al. [1] (see Section 4) is computed on $\mathsf{m}_0, \mathsf{m}_1$ committed in $\mathsf{ck}_0, \mathsf{ck}_1$, i.e. that (1) $\mathsf{otr} = (s_0, t_0, s_1, t_1)$ where $(s_0, t_0) = (g_1^{\alpha_0} h^{\beta_0}, u_1^{\alpha_0} e^{\beta_0} \mathsf{m}_0)$, and $(s_1, t_1) = (g_1^{\alpha_1} h^{\beta_1}, u_1^{\alpha_1} (e/g)^{\beta_1} \mathsf{m}_1)$, for some $(\alpha_0, \beta_0, \alpha_1, \beta_1) \in \mathbb{Z}_p^4$, where the receiver's OT message was $\mathsf{ct} = (u_1, u_2, e, v)$; and (2) that $\mathsf{m}_0, \mathsf{m}_1$ in the two equations above are encrypted respectively in $\mathsf{ck}_0, \mathsf{ck}_1$. In other words:

$$\begin{aligned} \mathsf{Lotr}^{\ell}(\mathsf{pk}) \ = \ \{ \ & (\mathsf{otr}, \mathsf{ct}, \mathsf{ck}_0, \mathsf{ck}_1) \ \text{ s.t. } \ \exists \mathsf{m}_0, \mathsf{m}_1, \mathsf{r} \\ & (\mathsf{ck}_0, \mathsf{m}_0) \in \mathsf{Le}^{[\ell|0]}(\mathsf{pk}) \wedge (\mathsf{ck}_1, \mathsf{m}_1) \in \mathsf{Le}^{[\ell|1]}(\mathsf{pk}) \wedge \mathsf{otr} = \mathsf{OTrsp}_{\mathsf{pk}}(\mathsf{ct}, k_0, k_1; \mathsf{r}) \ \} \end{aligned}$$

If ciphertexts $\mathsf{ck}_0, \mathsf{ck}_1$ are formed using the basic, i.e. *not* "shifted", version of the encryption then the plaintexts m_b (for $b = 0, 1$) are in the base in both t_b and in the e_b component of $\mathsf{ck}_b = (u_{1,b}, u_{2,b}, e_b, v_b)$, so we can cancel these plaintexts out by replacing (for $b = 0, 1$) constraints $e_b = h^{r_b}\mathsf{m}_b$ implied by $\mathsf{ck}_b = \mathsf{E}^{[\ell|b]}(\mathsf{pk}, \mathsf{m}_b; r_b)$ with constraints $t_b/e_b = {u_1}^{\alpha_b}(e/g^b)^{\beta_b}(h^{-1})^{r_b}$. This results in 10 linear constraints with 6 variables, and statements in Lotr can be mapped onto statement (C, M) in $\mathsf{LMI}_{10,6}$ as shown in equation (8), where $\xi_b = \mathsf{H}([\ell|b], u_{1,b}, u_{2,b}, e_b)$. The witness $(\mathsf{m}_0, r_0, \mathsf{m}_1, r_1, \alpha_0, \beta_0, \alpha_1, \beta_1)$ is mapped onto vector $w = (r_0, r_1, \alpha_0, \beta_0, \alpha_1, \beta_1)$.

$$\mathbf{M}' = \left[\begin{array}{ccc|ccc|cccc} g_1 & g_2 & (cd^{\xi_0}) & 1 & 1 & 1 & 1 & h^{-1} & 1 & 1 \\ 1 & 1 & 1 & g_1 & g_2 & (cd^{\xi_1}) & 1 & 1 & 1 & h^{-1} \\ 1 & 1 & 1 & 1 & 1 & 1 & g_1 & u_1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 & 1 & h & e & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & g_1 & u_1 \\ 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & h & e/g \end{array}\right] \tag{8}$$

$$\mathbf{C} = \left[\begin{array}{ccc|ccc|cccc} u_{1,0} & u_{2,0} & v_0 & u_{1,1} & u_{2,1} & v_1 & s_0 & t_0/e_0 & s_1 & t_1/e_1 \end{array}\right]$$