Efficient Covert Two-Party Computation

Stanislaw Jarecki*

University of California, Irvine

Abstract. Covert computation strengthens secure computation by hiding not only participants' inputs (up to what the protocol outputs reveal), but also the fact of computation taking place (up to the same constraint). Existing maliciously-secure covert computation protocols are orders of magnitude more costly than non-covert secure computation, and they are either non-constant round [5] or they use non-black-box simulation [10]. Moreover, constant-round covert computation with black-box simulation is impossible in the plain model [10].
We show that constant-round *Covert Two-Party Computation* (2PC) of general functions secure against malicious adversaries is possible with black-box simulation under DDH in the Common Reference String (CRS) model, where the impossibility result of [10] does not apply. Moreover, our protocol, a covert variant of a "cut-and-choose over garbled circuits" approach to constant-round 2PC, is in the same efficiency ballpark as standard, i.e. non-covert, 2PC protocols of this type. In addition, the proposed protocol is covert under concurrent self-composition.
An essential tool we use is a *covert* simulation-sound Conditional KEM (CKEM) for arithmetic languages in prime-order groups, which we realize in CRS or ROM at costs which are either the same (in ROM) or very close (in CRS) to known HVZK's for such languages.

1 Introduction

Covert computation addresses a security concern which is unusual for cryptography, namely how to hide the very fact of (secure) protocol execution. Such hiding of a protocol instance is possible if the communicating parties are connected by *steganographic* channels, which are implied by any channels with sufficient entropy [11]. Consider the simplest example of a steganographic channel, the *random channel*, a.k.a. a *random beacon*. In practice such channel can be implemented e.g. using protocol nonces, padding bits, time stamps, and various other communication (and cryptographic!) mechanisms which exhibit inherent (pseudo)entropy. (Works on steganographic communication, e.g. [11], show that random messages can be embedded into any channel with sufficient entropy.) Two parties sharing a random channel can use it to send protocol messages, and their presence cannot be distinguished from an a priori channel behavior if protocol messages are indistinguishable from random bitstrings. The participants must agree on which bits to interpret as protocol messages, but this can be public information since the decoding is indistinguishable from random.

* Contact: sjarecki@uci.edu

Covert computation was formalized for the two-party honest-but-curious setting by Von Ahn et al. [28], and then generalized (and re-formulated) to the multi-party and malicious adversary setting by Chandran et al. [5], 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 implies participants' presence. Technically, in covert computation the computed function outputs an additional *reveal* bit: If this bit is 0 then each participant remains indistinguishable from a random beacon to the others, and if the reveal bit is 1 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.

Q & A on Covert Computation. *Motivation:* Who wants to compute a function while hiding this very fact from (some) potential protocol participants? A generic example is authentication whose participants want to remain undetectable except to counter-parties whose inputs (certificates, secrets, passwords, gathered observations) match their authentication policy: If two spies search for one another in a foreign country, they want to do so while preventing anyone from detecting their authentication attempts. If the spies authenticated each other using covert computation, the only way their presence can be detected is by an active attacker whose inputs are those which the spies search for.

Random Channels: 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. This is why covert protocol participants must have 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/video signals) then this entropy can be used to create a steganographic channel, and such channel can carry covert MPC protocol messages.

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 based on their on-line inputs.

Covert MPC vs. Secure MPC: Secure computation is believed to conceptualize every security task: Whatever security property we want 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 extra bit of information, because it does not hide whether some entity engages in the protocol, and in some applications, this is essential information. *Covert computation* strengthens secure computation to hide this remaining bit, and ensures undetectability of protocol participation even to active participants except (and this "escape clause" seems unavoidable) if the computation determines that its outputs, and hence also the fact of protocol participation, should be revealed. We show that, assuming CRS, this strengthening of secure computation to covertness can be achieved in the two-party case in constant rounds with black-box simulation under standard assumptions. Moreover, we achieve it at the cost which is in the same ballpark as the cost of known standard, i.e. non-covert, constant-round 2PC based on Yao's garbled circuits [29], so covertness in a sense comes "for free". As a side-benefit, the tools we use for covert enforcement of honest protocol behavior can be re-used in other covert protocols, e.g. for specific functions of interest.

Covert Computation as a Tool: Covert computation can also be a protocol tool with surprising applications, although we are aware of only one such case so far. Cho et al. [6] generalized a construction of Manulis et al. [20] to compile secure computation protocol for so-called "single instance" functionality, e.g. a pair-wise authentication by parties holding certificates issued by the same authority, into its "multiple instance" version, e.g. where each party holds a set of certificates from multiple authorities. The compiler has only linear cost, and it works by encoding messages of n instances of the single-instance protocol as points on an n-degree polynomial. Crucially, no one should distinguish which points encode valid messages until the single-instance functionality reveals its output, and [6] show that the compiler works for general functionalities *if* the single-instance protocol is *covert*.

Prior Work on Covert Computation. Von Ahn et al. [28] 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. Chandran et al. [5] reformulated covert computation to guarantee output correctness and generalized it to multi-party computation, but their protocol was also non-constant-round, and its efficiency was *several orders of magnitude over known non-covert MPC protocols*: Each party was covertly proving that it followed a GMW MPC 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. Goyal and Jain [10] showed that non-constant-round protocols are necessary to achieve covert computation with black-box simulation against malicious adversaries, at least in the plain MPC model, i.e., without access to some trusted parameters. [10] also showed a constant-round covert MPC with non-black-box simulation (and bounded concurrency), but their protocol re-uses the above-mentioned components of [5], and is therefore just as costly.

Whereas the covert MPC protocols of [5, 10] assumed the plain computation model, recently Cho et al. [6] exhibited practical constant-round covert computation protocols secure against active adversaries, for two specific two-party functionalities, namely string equality and set intersection, in the Random Oracle Model (ROM). (Moreover, [6] strengthened the definition of covert computation of [5] to unbounded concurrent self-composition, which we adopt here.) However, their constructions are custom-made and it is not clear how they can be extended to computation of general functions. In other related work, Jarecki [13] showed a constant-round covert Authenticated Key Exchange (AKE) with $O(1)$ public key operations, but this protocol satisfied a game-based AKE definition, and it was not a covert secure computation of any function.

Main Contribution: Efficient Constant-Round Covert 2PC in CRS. This leaves a natural open question whether assuming some relaxation in the trust model (necessary in view of the negative result of [10]) general two-party functions can be computed covertly by a constant-round protocol with black-box simulation, 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 these questions affirmatively assuming the Common Reference String (CRS) model and the Decisional Diffie-Hellman (DDH) assumption.[1] In this setting we show a covert 2PC protocol which follows the well-known paradigm for standard, i.e. non-covert, constant-round secure 2PC, initiated by [21, 18] and followed in numerous works. Namely, we use the cut-and-choose technique over $O(\tau)$ copies of Yao's garbled circuit, but we do so using *efficient* covert equivalents of standard protocol tools for enforcing honest protocol behavior, like extractable commitments and simulation-sound arguments. Moreover, the protocol is secure under concurrent composition. Remarkably, our covert 2PC protocol is roughly in the same efficiency ballpark as the non-covert secure 2PC protocols of this type. For example, for a *one-sided output* function with n-bit inputs with a Boolean circuit with c gates, our protocol requires 5 rounds, $O(n\tau)$ exponentiations, and a transfer of $O(n\tau)$ group elements and $O(c\tau)$ symmetric ciphertexts.

Challenge of Malicious Security for Covert Computation. Assuming random channels, covert *communication* is essentially as easy as secure communication: Under standard assumptions symmetric encryption modes have ciphertexts which are indistinguishable from random bitstrings. Several known public-key encryption schemes, e.g. Cramer-Shoup encryption [8], also have ciphertexts that are indistinguishable from a tuple of random group elements (under DDH), and random group elements in a prime-order subgroup of modular residues are easy to encode as random bitstrings. Von Ahn et al. [28] show that General covert *computation in the honest-but-curious setting* is also not more difficult than standard secure computation, because (1) 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 (2) because a DDH-based OT of Naor-Pinkas Oblivious Transfer (OT) [22] is covert under the same DDH assumption.

[1] We note that a work in progress by Couteau [7] aims to show a corresponding result for covert MPC protocols built using the non-constant-round GMW paradigm.

However, it is harder to achieve covert 2PC/MPC protocols secure against *malicious* adversaries because of the lack of efficient covert counterparts to standard mechanisms for enforcing honest protocol behavior. For example, the tools often used to enforce honest behavior in Yao's garbled circuit protocol are (1) Zero-Knowledge (ZK) proofs, e.g. to show that consistent inputs are input into multiple garbled circuit instances and into the OT, and (2) opening a commitment to show that the committed value is correctly formed. Either tool is publicly verifiable and thus violates covertness.

Second Contribution: Efficient Covert Simulation-Sound CKEM's. Our tool for enforcing honest protocol behavior is an efficient covert *Conditional Key Encapsulation Mechanism* (CKEM) for a wide class of discrete-log-based languages, including statements that a Cramer-Shoup ciphertext [8] is computed correctly, that an encrypted value is a bit, or that a commitment decommits to a given plaintext. A CKEM is a variant of Conditional OT [9], and 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 [5, 10, 13] assures that an interaction with either S or R is indistinguishable from a random beacon. In particular, even given the witness w for x the receiver cannot distinguish $\mathsf{S}(x)$ from random: It can compute key K of this CKEM instance, but this key is random and should not the sender distinguishable from a randomness source. Hence, covert CKEMs can provide a covert counterpart to Zero-Knowledge Proofs: Instead of A proving to B that its protocol messages are correct, B and A run a covert CKEM for the same language as resp. S and R, and use key K to (covertly) encrypt subsequent protocol messages: If A's messages were malformed, covert CKEM assures that key K is pseudorandom to A and all subsequent messages of B are pseudorandom.

To be most useful as a protocol tool, a covert CKEM should be *concurrent*, *simulatable*, and *simulation sound* (a *proof of knowledge* property can help too, but our covert 2PC protocol does not utilize it). We exhibit constructions of such covert CKEM's for two classes of languages. The first class are so-called *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 $f_M(w) = w \cdot M$ defined by M (where scalar multiplication stands for a homomorphic one-way function, e.g. exponentiation). The second class are languages with so-called Σ-protocols, i.e. three-round public-coin HVZK proofs with special soundness and zero-knowledge properties which are typically satisfied by e.g. proofs of arithmetic statements in prime-order groups. We overview our covert CKEM constructions below, but in the nutshell we show (1) a 2-round CKEM in CRS for LMI languages defined by a full row rank matrix M, whose cost is about 2-4 times that of the underlying HVZK, (2) a 2-round CKEM in ROM for Σ-protocol languages whose cost almost matches the underlying HVZK, and (3) a 4-round CKEM in CRS for Σ-protocol languages with a larger but still additive overhead over the underlying HVZK.

Prior Work on Covert CKEM's. Covert CKEM was introduced as *Zero-Knowledge Send* (ZKSend) by Chandran et al. [5], and strengthened to proof of knowledge, simulation-soundness, and bounded concurrency in [10], but it is not clear how these constructions can yield *practical* covert CKEM's for Σ-protocol or LMI languages. The ZKSend of [5] reduces L to a Hamiltonian Cycle (HC) instance, replaces the verification step in Blum's binary-challenge ZK proof for HC with covert garbled circuit evaluation of this step, and repeats this $O(\tau)$ times for negligible soundness error. The ZKSend of [10] follows this paradigm using the ZK argument for NP by Pass [23]. Both constructions aimed at feasibility of covert CKEM for NP, while we want covert CKEM's at costs comparable to the underying HVZK's, for LMI or Σ-protocol languages. Moreover, much of the complexity in the constant-round covert CKEM of [10] was to assure covertness (and bounded concurrency) with careful usage of rewinding in the simulation. Indeed, the negative result in [10] for constant-round covertness with black-box simulation was due to rewinding necessary in simulation in the plain model. By contrast, if we assume CRS, an assumption without which we do not know how to achieve even *secure* constant-round protocols with unbounded concurrency and/or practical efficiency, we can get concurrency with straight-line simulation using a CRS trapdoor, and the negative result of [10] no longer applies.

Efficient Covert Simulation-Sound CKEM's: A Closer Look. One starting point for a practical straight-line simulatable covert CKEM for an LMI language can be an efficient SPHF, because an SPHF for an LMI language defined by a full row rank matrix is covert. However, SPHF by itself is not *simulatable*: Note that a simulator who plays the role of an honest party typically does not form its messages as the honest party would, which would make the statement in the SPHF instance (that the honest party's protocol messages are well-formed) incorrect in the simulation. Hence, by SPHF security (a.k.a. smoothness), the simulator could not recover key K, and would fail in simulation of subsequent protocol rounds. To amend precisely this shortcoming of SPHF's, Benhamouda et al. [3] upgraded SPHF's for LMI languages to CKEM's in CRS with (1) *(concurrent) simulatability*, i.e. the ability for the simulator in possession of the CRS trapdoor to derive key K even on the wrong statement $x \notin \mathsf{L}$; and (2) *simulation-soundness*, i.e. that a cheating receiver cannot recover S's key K for an 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 $x' \notin \mathsf{L}$. Both are needed of ZK proofs in a compiler from (concurrent) honest-but-curious MPC to (concurrent) maliciously-secure MPC, and [3] showed that this compiler works if ZKP's are replaced by CKEM's. However, their goal was to reduce rounds in MPC by replacing ZKP's with CKEM's, and *not*, as in our case, to assure MPC covertness. In particular, their CKEM's are not covert: To assure straight-line extraction [3] modify LMI statement M, C s.t. $C = w \cdot M$ for both the original witness w_{R} and the simulator's CRS trapdoor w_{td}. This way the receiver and the simulator can both compute hash value $C \cdot \mathsf{hk}$ from projection $\mathsf{hp} = M \cdot \mathsf{hk}$, respectively as $w_{\mathsf{R}} \cdot \mathsf{hp}$ and $w_{\mathsf{td}} \cdot \mathsf{hp}$. However, this holds only if hp is formed correctly, i.e. if

$\mathsf{hp}^T \in \mathsf{span}(M^T)$, hence [3] run add a secondary SPHF where R sends $\mathsf{tp} = M^T \cdot \mathsf{tk}$ for random tk to S, who can compute hash value $\mathsf{hp}^T \cdot \mathsf{tk}$ as $\mathsf{hk}^T \cdot \mathsf{tp}$ if and only if $\mathsf{hp}^T = \mathsf{hk}^T \cdot M^T$. Consequently, the CKEM of [3] publicly sends $\mathsf{hp} = M \cdot \mathsf{hk}$ and $\mathsf{tp} = M^T \cdot \mathsf{tk}$. Matrix M is typically full row-rank, assuring that $M \cdot \mathsf{hk}$ is a random vector in the column space, but it is not full column-rank, which makes $M^T \cdot \mathsf{tk}$ not uniformly random. However, we show how to modify matrix M s.t. the secondary projection $M^T \cdot \mathsf{tk}$ is *pseudo*-random during simulation.

A different route towards Covert CKEM's is to take as a starting point an efficient compiler from Σ-protocol to covert CKEM of [13]. The CKEM notion of [13] is covert and proof-of-knowledge, but not simulatable and simulation-sound. The covert CKEM of [13] bears some similarity to the ZKSend of [5]: Both start from an HVZK proof (resp. Σ-protocol and Blum's HVZK for HC), identify some proof messages which are already pseudorandom, replace the offending non-covert message with its commitment, and replace the verification equation with some covert gadget: [5] commit to the final response in Blum's proof and use garbled circuit for the verification step on the committed plaintext, and [13] commit to the prover's first message, and uses the "special simulation" property of a Σ-protocol, i.e. that the first message can be computed from the other two, to replace the verification step with a covert SPHF that checks that the committed first message equals to this single verification-passing value. The benefit of the compiler of [13], in contrast to the above approach, is that it adds only a (small) additive overhead to the underlying Σ-protocol. We show that the 2-round instantiation of this compiler is simulatable and simulation-sound assuming ROM, and that the 4-round version of this compiler can be modified so that the result is covert simulatable and simulation-sound in CRS.

Organization. In Section 2 we introduce covertness-related notation. In Section 3 we define concurrent covert 2PC for arbitrary functions. In Section 4 we define covert counterparts to standard protocol building blocks, including covert CCA PKE, commitment, OT, HbC-secure circuit garbling, and SPHF's. In Section 5 we define covert CKEM's. In Section 6 we discuss our covert CKEM's constructions. Finally, in Section 7 we present our concurrent covert 2PC protocol.

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 $y \leftarrow S$ 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 $\mathsf{ks} = \{(k^{i,0}, k^{i,1})\}_{i\in[n]}$, 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 $p^b_{\mathcal{A}} = \Pr[1 \leftarrow \mathcal{A}^{P_b(x_b)}(z) \,|\, (x_0, x_1, z) \leftarrow \mathsf{Setup}(1^\tau)]$, where the probability goes over the coins of Setup, $\mathcal{A}$, and P_b.

Covert Encodings. In our protocols all communicated values are either random fixed-size bitstrings, or random integers from some range $\mathbb{Z}_n$, or random elements of a prime-order group G. In the latter two cases what is sent on the wire are not the values themselves but their covert encodings. A covert encoding of domain D is a randomized function $\mathsf{EC} : D \to \{0,1\}^{p(\tau)}$ defined for some polynomial p, s.t. a random variable $\{\mathsf{EC}(a; \mathsf{r})\}$, induced by random a in D and random r, is statistically close to a random bitstring of length $p(\tau)$. Moreover, there must exist a decoding procedure DC s.t. $\mathsf{DC}(\mathsf{EC}(a; r)) = a$ for all $a \in D$ and all r. For example, if domain D is an integer range $\mathbb{Z}_n$ then $\mathsf{EC}(a)$ 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 multiplicative group $\mathbb{Z}^*_q$ of residues modulo q for $q = p \cdot t + 1$ s.t. $gcd(p, t) = 1$, 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 Defining 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 [6]. 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 [6], which we recall (and refine) below, follows the definition of stand-alone (i.e. "single-shot") covert computation given by Chandran et al. [5], here restricted to the two-party case. The definition casts this notion in the framework of universal composability (UC) by Canetti [4], 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 [4]. 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 [5], 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.

Parameters: Admission function g and the main function f.

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 $\mathsf{F}_{\mathsf{C}(f,g)}$

Ideal and Real Models. The definition of the *ideal model* is the UC analogue of the ideal model of Chandran et al. [5], except that composability guarantees 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. As in [5] function g is an admission function, i.e. if $g(x, y) = 0$ then functionality $\mathsf{F}_{\mathsf{C}(f,g)}$ returns return $\perp$ to both parties, and f is the "real function" i.e. if $g(x, y) \neq 0$ then functionality $\mathsf{F}_{\mathsf{C}(f,g)}$ prepares A's output as z and B's output as v where $(z, v) = f(x, y)$. 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 $\mathsf{Ideal}_{\mathsf{F},\mathcal{A}^*,\mathcal{Z}}(\tau, \mathsf{aux}, r)$ 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 aux, and random input $\mathsf{r} = (\mathsf{r}_{\mathcal{Z}}, \mathsf{r}_{\mathcal{A}^*}, \mathsf{r}_{\mathsf{F}})$, as described above. Let $\mathsf{Ideal}_{\mathsf{F},\mathcal{A}^*,\mathcal{Z}}\ (\tau, \mathsf{aux})$ be the random variable $\mathsf{Ideal}_{\mathsf{F},\mathcal{A}^*,\mathcal{Z}}(\tau, \mathsf{aux}; \mathsf{r})$ when r is uniformly chosen. We denote the random variable $\mathsf{Ideal}_{\mathsf{F},\mathcal{A}^*,\mathcal{Z}}(\tau, \mathsf{aux})$ as $\{\mathsf{Ideal}_{\mathsf{F},\mathcal{A}^*,\mathcal{Z}}(\tau, \mathsf{aux})\}_{\tau\in\mathbb{N};\mathsf{aux}\in\{0,1\}^*}$. Correspondingly we let $\mathsf{Real}_{\Pi,\mathsf{Adv},\mathcal{Z}}(\tau, \mathsf{aux}; \mathsf{r})$ be the output of $\mathcal{Z}$ after interacting with a real-world adversary Adv and parties running protocol Π on security parameter τ, input aux, 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 random variable $\{\mathsf{Real}_{\Pi,\mathsf{Adv},\mathsf{F}}(\tau, \mathsf{aux})\}_{\tau\in\mathbb{N};\mathsf{aux}\in\{0,1\}^*}$.

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 $\mathcal{Z}$,*

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

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 7 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 3) 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 [8]. 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 the full version [14] we show that the proof of CCA security of Cramer-Shoup PKE under the DDH assumption [8] 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. [28] used a covert version of Naor-Pinkas OT [22] for their covert 2PC secure against honest-but-curious adversaries. Here we will use a covert version of the OT of Aiello et al. [2] instead because it is compatible with CCA-covert Cramer-Shoup encryption and covert CKEM's of Section 6. 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. [28] shows a covert version of Yao's garbling $\mathsf{GCgen}(f)$ for any $f : \{0,1\}^n \rightarrow \{0,1\}^m$. 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 [28] 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 SPHF notion of [25], for alternative formulation see e.g. [15]) 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 $\mathsf{KG}(\pi, x)$. In the context of our protocols SPHF values are elements of group G uniquely defined by security parameter τ via the Cramer-Shoup key generation procedure, hence we can define SPHF *smoothness* as 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 we need a stronger notion we call *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 (CKEM)

Conditional Key Encapsulation Mechanism (CKEM). A Conditional KEM (CKEM) [13] is a KEM version of *Conditional Oblivious Transfer* (COT) [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 $(x_R, w_R) \in \mathcal{R}[\mathsf{L}]$. CKEM 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 can use proof-system terminology to define CKEM security properties. 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. [3] 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 [3], 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 $x' \notin \mathsf{L}$.

To distinguish between different CKEM sessions the CKEM syntax must also be amended by *labels*, denoted ℓ, 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 labels ℓ:

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

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

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 CKEM. A *covert* CKEM was introduced as *Zero-Knowledge Send* (ZK-Send) by Chandran et al. [5], who strengthened simulatable CKEM by adding covertness, i.e. that an interaction with either S or R (but not the keys they compute locally) is indistinguishable from a random beacon. Goyal and Jain [10] strengthened the covert CKEM of [5] to proof-of-knowledge and simulation-soundness under (bounded) concurrent composition. Our covert CKEM notion is essentially the same as the covert ZKSend of [10] (minus the proof-of-knowledge property), but we adopt it to the CRS setting of a straight-line simulation using a global CRS trapdoor as in [3].

Covert CKEM Zero-Knowledge. We say that a CKEM for language 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. $(x, w) \in \mathcal{R}[\mathsf{L}]$.[2]
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 $(\mathsf{st}, x, \ell) \leftarrow \mathcal{A}_1(\pi, \mathsf{td})$.

[2] 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 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).

Note that *Zero-Knowledge* and *Trapdoor-Receiver Covertness* imply a *Receiver Covertness* property, which asks that $\mathsf{Rec}(\pi, x, \ell, w)$ instances are indistinguishable from $\mathsf{Rec}^{\$(\tau)}$ for any $(x, w) \in \mathcal{R}[\mathsf{L}]$. This holds because an interaction with $\mathsf{Rec}(\pi, x, \ell, w)$ for $(x, w) \in \mathcal{R}[\mathsf{L}]$ 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 [3] 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 in the zero-knowledge and trapdoor-receiver covertness properties includes the simulator's trapdoor td, which implies that both properties will hold in the presence of multiple CKEM instances simulated by TRec using td. This is not the case for in sender simulation-covertness, but there the adversary has *oracle access* to simulator TRec who uses td on other CKEM instances, which suffices for the same goal of preserving the CKEM covertness property under concurrent composition.

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 [3] 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 [3] 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 [3] 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 indistinguishable 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.

6 Covert CKEM's for LMI and Σ-Protocol Languages

Linear Map Image (LMI) Languages. The Covert 2PC protocol of Section 7 relies on covert zero-knowledge and simulation-sound (covert-zk-and-ss) 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.

Covert CKEM's for LMI Languages. We show three types of covert CKEM's for LMI languages: (1) a 2-round CKEM in CRS for LMI languages defined by a full row rank matrix M, whose cost is about 2-4 times that of the underlying HVZK, (2) a 2-round CKEM in ROM for Σ-protocol languages whose cost matches the underlying HVZK, and (3) a 4-round CKEM in CRS for Σ-protocol languages with a small additive overhead over the underlying HVZK. Note that every LMI languages in prime-order groups has a Σ-protocol, so constructions (2) and (3) apply to LMI languages. For lack of space we include below only the last construction and we refer to the full version [14] for the other two.

6.1 2-Round Covert CKEM for Σ-Protocol Languages in ROM

The covert mutual authentication scheme of Jarecki [13] uses a compiler which converts a Σ-protocol into a 2-round CKEM for the same language, assuming ROM. The resulting CKEM was shown to satisfy the CKEM covertness notion of [13], which included receiver covertness and *strong* sender covertness, a covert counterpart of strong soundness (a.k.a. proof of knowledge), but did not include simulatability or simulation soundness. However, it is not hard to see that this 2-round CKEM does achieve both zero-knowledge and simulation soundness.

We recall this construction in Figure 2, and we briefly explain how it works. Assume that the instance,witness pairs of language L are mapped 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 $a = w' \cdot M$ 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 $a = z \cdot M - eC$, which holds if $C = w \cdot M$. As is well known, this Σ-protocol becomes a NIZK in ROM if the verifier's e is computed via a random oracle.

Consider an ElGmal-based covert commitment: Let g_1, g_2 be two random group elements in the CRS. Let $\mathsf{Com}_{g_1,g_2}(\mathsf{m})$ for $\mathsf{m} \in \mathbb{Z}_p$ pick $r \leftarrow \mathbb{Z}_p$ and output $\mathsf{cmt} = (\mathsf{cmt}_1, \mathsf{cmt}_2) = ((g_1)^r, (g_2)^r(g_1)^{\mathsf{m}})$. This is perfectly binding and covert under the DDH assumption. Define language $\mathsf{Lc} = \{(\mathsf{cmt}, \mathsf{m}) \mid \mathsf{cmt} = \mathsf{Com}_{g_1,g_2}(\mathsf{m})\}$, and note that Lc has a well-known covert SPHF: $\mathsf{KG}(g_1, g_2)$ generates $\mathsf{hk} \leftarrow \mathbb{Z}_p^2$ and $\mathsf{hp} = \mathsf{hk} \cdot (g_1, g_2) = (g_1)^{\mathsf{hk}_1}(g_2)^{\mathsf{hk}_2}$, $\mathsf{Hash}((\mathsf{cmt}, \mathsf{m}), \mathsf{hk}) = \mathsf{hk} \cdot (\mathsf{cmt}/(1, g_1^{\mathsf{m}})) = (\mathsf{cmt}_1)^{\mathsf{hk}_1}(\mathsf{cmt}_2/(g_1)^{\mathsf{m}})^{\mathsf{hk}_2}$, and $\mathsf{PHash}((\mathsf{cmt}, \mathsf{m}), r, \mathsf{hp}) = r \cdot \mathsf{hp} = (\mathsf{hp})^r$.

Let H be a hash onto $\mathbb{Z}_p$. The 2-round ROM-based covert CKEM of [13] works just like a ROM-based NIZK except that the prover replaces message

a with its commitment $\mathsf{cmt} = \mathsf{Com}(\mathsf{H}(a))$, and the non-interactive verification check whether $a = z \cdot M - eC$, the verifier computes $a = z \cdot M - eC$ locally and uses the covert SPHF for Lc to verify if cmt is a commitment to $\mathsf{H}(a)$. This protocol is shown in Figure 2, where $\mathsf{H}_i(x)$ stands for $\mathsf{H}(i, x)$.

On inputs (g_1, g_2), $(C, M) = \phi(x)$, ℓ, 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 $a = w' \cdot M$, $\mathsf{cmt} \leftarrow \mathsf{Com}_{g_1,g_2}(\mathsf{H}_2(a); r)$, $e = \mathsf{H}_1(x, \ell, \mathsf{cmt})$, $z = w' + ew$, and send (cmt, z) to S.

S: Set $a = z \cdot M - eC$ for $e = \mathsf{H}_1(x, \ell, \mathsf{cmt})$, 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 $\mathsf{m} = \mathsf{H}_2(a)$.

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

Fig. 2. 2-round covert-zk-and-ss CKEM in ROM for LMI (adopted from [13])

Figure 2 is written specifically for LMI languages but it is easy to see that the same works for any Σ-protocol language. Note that its cost is that of the Σ-protocol for language L plus 2 exponentiations for S and 1 exponentiation for R. We refer to the full version of the paper [14] for the proof of theorem 1:

Theorem 1. *For any* LMI *language* L*, the CKEM scheme for* L *shown in Figure 2 is covert zero-knowledge and covert simulation-sound in ROM, assuming DDH.*

7 Covert Computation of General 2-Party Functions

We describe protocol Π_{COMP}, Figure 3, which realizes the concurrent 2-party covert computation functionality $\mathsf{F}_{\mathsf{C}(f,g)}$ in the CRS model. Protocol Π_{COMP} is a *covert* variant of the cut-and-choose method over $O(\tau)$ copies of Yao's garbled circuit [29], which has been the common paradigm for standard 2PC protocols, initiated by [21, 18], followed by many subsequent works, e.g. [19, 26, 12], including several implementation efforts, e.g. [24, 26, 16, 27, 1].

A standard way of implementing a cut-and-choose involves tools which are inherently non-covert: First, the 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, this 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 can also use secondary 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.

Here we show that (concurrent and simulation-sound) covert CKEM's can be effectively used in both of the above 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). 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-Shoup 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 when the admission function evaluates to true and the functionality reveals computation outputs, requires several adjustments, which we attempt to explain in the technical protocol overview below.

Defining the Garbled Circuit. We first explain how we use the covert garbling procedure GCgen of [28], 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 evaluate 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 [28] 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 $c \oplus d$:

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

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 A 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 [28] 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_A^2, an encryption key for B's final message m_B^3, 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 key pk and label ℓ are 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 full version [14] 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. [2], formed using procedure OTrsp (see Section 4), which the sender uses in Yao's protocol to send keys corresponding to *receiver's wires*:

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

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

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

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

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

Notation in Figure 3. 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 of Section 6 have the same asymptotic computation and bandwidth costs as the HVZK 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, 17], it does not use an OT extension, and it does not use many other

$\mathsf{PG}(1^\tau)$ 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 $\mathsf{m}_B^2 = \mathsf{SE}_{\mathsf{K}_B}^0[\mathsf{ct}^{\mathsf{B}}, \{\mathsf{r}_i^{\mathsf{gc}}\}_{i\in S}, \{\mathsf{gc}_i, \mathsf{ks}_i^B, \{\mathsf{otr}_i^w\}_{w\in\overline{X}}\}_{i\notin S}]$ to A.

A2: on K'_B and m_B^2:
set $(\mathsf{ct}^{\mathsf{B}}, \{\mathsf{r}_i^{\mathsf{gc}}\}_{i\in S}, \{\mathsf{gc}_i, \mathsf{ks}_i^B, \{\mathsf{otr}_i^w\}_{w\in\overline{X}}\}_{i\notin S}) \leftarrow \mathsf{SD}_{\mathsf{K}'_B}^0(\mathsf{m}_B^2)$;
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 $\mathsf{m}_A^2 \leftarrow \mathsf{SE}_{\mathsf{K}_A}^0(\mathsf{F}(\mathsf{K}'_B, 1), t_1, ..., t_{n_v})$; Otherwise set $z{:=}\perp$ and $\mathsf{m}_A^2 \leftarrow \{0,1\}^{\tau(n_v+1)}$. Send m_A^2 to B.

B3: Set $(\tau, t_1, ..., t_{n_v}) \leftarrow \mathsf{SD}_{\mathsf{K}'_A}^0(\mathsf{m}_A^2)$. 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. 3. Protocol Π_{COMP} for Concurrent 2-Party Covert Function Computation $\mathsf{F}_{\mathsf{C}(f,g)}$

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, 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 $O(n_O\tau)$ where n_O 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), we hope that it might be avoidable with a more careful analysis. Moreover, since our covert 2PC is self-composable and both sides commit to their input bits in the same way, two instances of this protocol can be run in parallel for *one-sided output* versions of f, one for A's output and one for B's output, on same committed inputs. This multiplies all the other costs by 2 but drops the $O(n_O\tau)$ growth in the circuit size.

Theorem 2. *Protocol* Π_{COMP} *in Figure 3 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.

For lack of space we only show the algorithm of an ideal adversary $\mathcal{A}^*$, i.e. the simulator, divided into two parts depending on whether the ideal adversary simulates an instance of an honest party B, shown in Figure 4, or an honest party A, shown in Figure 5. The proof, included in the full version of the paper [14], shows that no efficient environment can distinguish an interaction with $\mathcal{A}^*$ and ideal honest players interacting via functionality $\mathsf{F}_{\mathsf{C}(f,g)}$ (where $\mathcal{A}^*$ additionally interacts with a local copy of the real-world adversary Adv), from an interaction with Adv and real-world honest players who interact via our protocol Π_{COMP}.

References

1. A. Afshar, P. Mohassel, B. Pinkas, and B. Riva. Non-interactive secure computation based on cut-and-choose. In *Advances in Cryptology - EUROCRYPT*, pages 387–404, 2014.
2. 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.

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

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

3. 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.
4. 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.
5. N. Chandran, V. Goyal, R. Ostrovsky, and A. Sahai. Covert multi-party computation. In *FOCS*, pages 238–248, 2007.
6. 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.
7. G. Couteau. Revisiting covert multiparty computation. Cryptology ePrint Archive, Report 2016/951, 2016. http://eprint.iacr.org/2016/951.
8. 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.
9. 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.
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. S. Jarecki. Efficient covert two-party computation. Cryptology ePrint Archive, Report 2016/1032, 2016. http://eprint.iacr.org/2016/1032.
15. 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.
16. B. Kreuter, A. Shelat, and C.-H. Shen. Billion-gate secure computation with malicious adversaries. In *USENIX Security*, pages 14–25, 2012.
17. Y. Lindell. Fast cut-and-choose-based protocols for malicious and covert adversaries. *J. Cryptology*, 29(2):456–490, 2016.
18. Y. Lindell and B. Pinkas. An efficient protocol for secure two-party computation in the presence of malicious adversaries. In *Advances in Cryptology - EUROCRYPT'07*, pages 52–78, 2007.
19. Y. Lindell and B. Pinkas. Secure two-party computation via cut-and-choose oblivious transfer. *J. Cryptology*, 25(4):680–722, Oct. 2012.

20. M. Manulis, B. Pinkas, and B. Poettering. Privacy-preserving group discovery with linear complexity. In *Proceedings of the 8th International Conference on Applied Cryptography and Network Security*, ACNS'10, pages 420–437, Berlin, Heidelberg, 2010. Springer-Verlag.
21. P. Mohassel and M. Franklin. Efficiency tradeoffs for malicious two-party computation. In *Public-Key Cryptography PKC 2006*, pages 458–473, 2006.
22. M. Naor and B. Pinkas. Computationally secure oblivious transfer. *J. Cryptology*, 18(1):1–35, 2005.
23. R. Pass. Bounded-concurrent secure multi-party computation with a dishonest majority. In *Proceedings of the 36th Annual ACM Symposium on Theory of Computing, Chicago, IL, USA, June 13-16, 2004*, pages 232–241, 2004.
24. B. Pinkas, T. Schneider, N. P. Smart, and S. C. Williams. Secure two-party computation is practical. In *Advances in Cryptology - ASIACRYPT*, pages 250–267, 2009.
25. M. D. Raimondo and R. Gennaro. Provably secure threshold password-authenticated key exchange. *J. Comput. Syst. Sci.*, 72(6):978–1001, 2006.
26. A. Shelat and C.-H. Shen. Two-output secure computation with malicious adversaries. In *Advances in Cryptology - Eurocrypt*, pages 386–405, 2011.
27. A. Shelat and C.-H. Shen. Fast two-party secure computation with minimal assumptions. In *ACM Conference on Computer and Communications Security - CCS*, pages 523–534, 2013.
28. 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.
29. A. Yao. How to generate and exchange secrets. In *27th FOCS*, pages 162–167, 1986.