Cross-Domain Secure Computation*

Chongwon Cho[1], Sanjam Garg[2], and Rafail Ostrovsky[3]

[1] Information and Systems Science Laboratory, HRL Laboratories
[2] IBM Research T. J. Watson
[3] Computer Science Department and Mathematics Department, UCLA

Abstract. Consider the setting of two mutually distrustful parties Alice and Bob communicating over the Internet, who want to securely evaluate desired functions on their private inputs. In this setting all known protocols for securely evaluating general functions either require honest parties to trust an external party or provide only weaker notions of security. Thus, the question of minimizing or removing trusted set-up assumptions remains open. In this work, we introduce the cross-domain model (CD) for secure computation as a means to reducing the level of required trust. In this model, each domain consists of a set of mutually trusting parties along with a key-registration authority, where we would like parties from distinct domains to be able to perform multiple secure computation tasks concurrently. In this setting, we show the followings:

- **Positive Construction for 2 domains:** We give a multiparty-party protocol that concurrently and securely evaluates any function in the CD model with two domains, using only a *constant* number of rounds and relying only on *standard assumptions*.
- **Impossibility Results for 3 or more domains:** Consider a deterministic function (e.g., 1-out-of-2 bit OT) that Alice and Bob in the standalone setting cannot evaluate trivially and which allows only Bob to receive the output. In this setting if besides Alice and Bob there is a third party (such that all three are from distinct domains) then they cannot securely compute any such function in the CD model in *concurrent* setting even when their inputs are *pre-specified*.

These results extend to the setting of multiple parties as well. In particular, there exists an n-party concurrently secure protocol in the CD model of n domains if and only if there are exactly n domains in the system.

Keywords: Multi-party computation, Concurrent security

* The significant part of this work was done while the first and second authors were Ph.D. students at UCLA. Research supported in part by NSF grants CNS-0830803; CCF-0916574; IIS-1065276; CCF-1016540; CNS-1118126; CNS-1136174; US-Israel BSF grant 2008411, OKAWA Foundation Research Award, IBM Faculty Research Award, Xerox Faculty Research Award, B. John Garrick Foundation Award, Teradata Research Award, and Lockheed-Martin Corporation Research Award. This material is also based upon work supported by the Defense Advanced Research Projects Agency through the U.S. Office of Naval Research under Contract N00014-11-1-0392. The views expressed are those of the authors and do not reflect the official policy or position of the Department of Defense or the U.S. Government.

1 Introduction

Consider the following scenario: Amazon and Walmart are two giant wholesale stores. Each store has a distributed set of servers to handle client requests. In order to establish best prices, Amazon and Walmart often need to collaborate on a real-time basis. In other words they need to compute functions of their confidential data which itself is distributed across the different servers. Neither do they trust each other nor are they willing to trust a third-party setup.

The well-studied notion of *secure computation* [40, 18] allows them to do so, however only in the *stand-alone* setting where security holds only if a single protocol session is executed in isolation. However, the requirement of free collaborations between Amazon and Walmart in the above requires security to hold even when multiple sessions are executed concurrently as in the Internet. What if Amazon in parallel wants to collaborate with another wholesale store Costco while protecting its confidential data even if Walmart and Costco collude with each other?

Background: Concurrent Security. In the past few years a lot of effort has been made in obtaining secure computation protocols in the demanding network setting where there might be multiple concurrent protocol executions. A large number of secure protocols (in fact under an even stronger notion of security called Universal Composability (UC)) based on various trusted third-party setup assumptions [8, 7, 2, 11, 24, 12, 30, 22, 21, 15] have been proposed. One of main aims to this line of work has been to reduce the level of trust that honest parties need to place in the trusted third-party setup. For example, Katz [24] considered the hardware token model. In his model, honest parties program tokens and send them to other parties. Since honest parties can program their own tokens, they only need to trust their hardware token manufacturer. Groth and Ostrovsky [22] initiated the study of constructing UC secure protocols without relying on a single trusted external entity. In other words, one of the main goals in this line of works is to achieve those notions of security in a setting which is as close to the "plain model" as possible (also see [21, 15] for subsequent works).

The Dark Side of Concurrency. Unfortunately, very strong impossibility results have been proved ruling out the existence of secure protocols in the concurrent setting. UC secure protocols for most functionalities of interest have been ruled out by [8, 6]. Concurrent self-composability[4] for a large class of interesting functionalities (i.e., bit transmitting functionalities) was first ruled out [31] only in a setting in which the honest parties choose their inputs *adaptively* (i.e., "on the fly"). Subsequently, a series of works [3, 19, 1, 17] show that it is impossible to achieve concurrent self-composition even in the very natural setting of *static* (pre-specified) inputs. In summary, these results have firmly established that for obtaining the most general result some setup is needed unless we are willing to consider more constrained models. Finally even in a setting with bounded number of players [23], an impossibility result has been established. However, this

[4] Concurrent self-composition requires that a protocol remain secure even when multiple copies of the same protocol are executed concurrently.

is for the more demanding setting in which honest parties choose their inputs adaptively.

1.1 Overview of our setting and results

We introduce a new set-up model, called the Cross-Domain(CD) model. A domain consists of a set of mutually trusting parties along with a key-registration authority. We prove the following for the setting of n-domain multi-party protocols:

Positive result if $n = 2$. We give a multi-party protocol that concurrently and securely evaluates any function in the CD model of two fixed domains where each domain may contain arbitrarily many parties. Our protocol has a *constant round* complexity, a *black-box* proof of security and relies only on *standard assumptions.*

Impossibility results when $n \geq 3$. We show that there does not exist a two-party protocol such that parties from three distinct domain can concurrently and securely realizes any *complete* asymmetric (only one party gets the output) deterministic functionality[5] in the stand-alone setting [25, 26, 29, 5, 27]. Our impossibility results hold even in the *very restricted* setting of *static* inputs (inputs of honest parties are pre-specified) and *fixed roles* (i.e, the adversary can corrupt only two parties who play the same role across all executions).

This answers the motivating question we started with. We can equip Amazon and Walmart to collaborate freely but this can not be done if collaborations with Costco are also desired.

Our results directly extend to the setting of n-domain protocols. In particular an n-party protocol for concurrently and securely computing any function on the joint inputs of n parties form distinct domains exists, if and only if there are exactly n domains in the system.

Relation with Bare-Public Key (BPK) model. The CD model is a generalization of the BPK introduced by Canetti et al. [9] model that has been studied extensively in the literature. Recall that in the BPK model each party sets up its own public-key and private-key pair. On the other hand in our model each domain has a key-registration authority that roughly generates a public-key and a private-key pair which is then used by all parties of the domain. We stress that even in the BPK model prior to our work no results for the setting of secure computation were known and our results fully characterize what is possible in the BPK setting. We elaborate on the details of this relation in Section 6.

1.2 Previous results with weaker notions of security

To address the problem of concurrent security for general secure computation in the *plain model*, a few candidate definitions have been proposed, including input-indistinguishable security [33, 16] and super-polynomial simulation [34, 38, 4, 30,

[5] A functionality is *complete* if it can be used to securely realize any other functionality.

10]. Both of these notions, although very useful in specialized settings, *do not* suffice in general. Additionally, other models that limit the level of concurrency have also been considered [35, 19] or allow simulation using additional outputs from the ideal functionality [20]. Among these models the model of m-bounded concurrency [36, 35] which allows for m different protocol executions to be interleaved has received a lot of attention in the literature [36, 35, 31, 32]. Unbounded concurrent oblivious transfer in the restricted model where all the inputs in all the executions are assumed to be independent has been constructed in [14]. Finally the only known positive results for concurrently secure composition in the plain model are for the zero-knowledge functionality [13, 39, 28, 37, 3].

1.3 Technical Overview

Impossibility Result. We start by giving the intuition behind the impossibility result for constructing a protocol that concurrently securely realizes the Oblivious Transfer(OT) functionality in the setting of three parties. The extension to general asymmetric two party functionalities follows using a Theorem from [1]. In the following, we consider the simplest setting where three domains exist and where each domain contains a single party.

Our impossibility result builds on the top of ideas developed by [1, 17] for the setting of plain model. Even though their result holds for the two party setting, we recall their technique for the setting of three parties. Consider a scenario with three parties Alice, Bob and Charlie. Now, consider an adversary that corrupts Bob and Charlie who (as receivers of the OT protocol) are allowed to participate in an arbitrary polynomial number of executions of the protocol with honest Alice (who plays as the sender). In this setting, we can construct a real-world adversary acting as Bob that interacts with Alice in an execution of the protocol, referred to as the *main* execution, that cannot be simulated in the ideal world.

The key idea is that the adversary has secure computation at its disposal and it can use it to its advantage. The adversary on behalf of Charlie may interact with Alice in multiple *additional* executions of the secure computation protocol and use these executions to generate messages that it needs to send in the main execution on behalf of Bob. More specifically, the adversary securely realizes Bob by using garbled circuits such that the adversary needs to evaluate the garble circuit in order to generate the messages it sends on behalf of Bob. However, the adversary does not have the OT keys necessary for evaluation of the garbled circuit. Instead, the OT keys are given to the honest Alice from which the adversary obtains the desired OT keys by the (additional) concurrent executions of the OT protocol as Charlie. Finally, the existence of a simulator simulating such an adversary that is securely implementing Bob contradicts the stand-alone security of the OT protocol. The pictorial description of our real-world adversary is provided in Figure 1.

In the CD model, each domain containing Alice, Bob and Charlie generates a certificate associated with their public-keys. The key insight in our impossibility result is to use the setting described above and to enable the garbled circuit

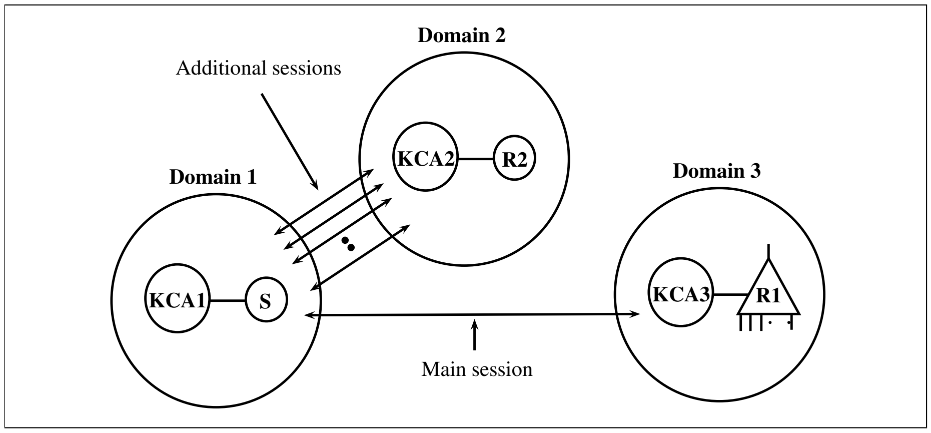

Fig. 1. Our real-world adversary $\mathcal{A}$ corrupting two receivers where R1 is Bob (replaced with the garbled circuit of its next message generator) and R2 is Charlie.

securely evaluating Bob to generate Bob's public key as well. The adversary however will generate Charlie's public key and secret key by himself, which enables the adversary to interact freely on Charlie's behalf. In particular, this allows the adversary to still obtain the OT keys for the garbled circuit from Alice as in the plain model. Finally, the existence of a simulator simulating such an adversary that is securely implementing Bob (along with its key registration) contradict the stand-alone security of the OT protocol in the CD model.

Positive result for two domains. The intuition behind the impossibility result above makes it abundantly clear that the adversary must be able to do secure computation with honest Alice if it wants to securely simulate Bob. However, if we restrict ourselves to the setting of two domains then the adversary essentially loses this ability, which eventually allows us to give a positive result.

Our protocol can roughly be partitioned into two phases– the preamble phase and the post-preamble phase. In the preamble phase, a party needs to demonstrate the knowledge of the secret key corresponding to its public and the certificate issued by its KCA. Subsequently in the post-preamble phase the actual secure computation happens. In the simulation for the proof of security, obtaining the knowledge of the adversary's secret key suffices for straight-line simulation.

Our protocol proceeds to the post-preamble phase only after the adversary has demonstrated knowledge of its secret key in the preamble phase. The adversary can interleave sessions arbitrarily and among these interleaved sessions consider the first session in which the protocol reaches the post-preamble phase. Let's call this session as the target session. Now note that since the target session was the first session in which the the post-preamble phase was reached, we can expect the same thing to happen with some probability on appropriate re-windings as well. We formalize this appropriately using *swapping* argument introduced in [37]. Now note that throughout this process of re-windings we never execute the post-preamble phase for any session. This allows us to avoid the

technical difficulties that generally arise when constructing concurrently secure two-party computation protocols. Our protocol with this limited re-windings is able to extract the secret key of the adversary and this allows our simulator to subsequently simulate all the sessions in straight-line. For our construction and the proof we build on the techniques developed in [3, 20, 16].
Organization: In Section 2, we first introduce the CD model. In Section 3, we present our impossibility result for static input concurrently secure two-domain two-party computation in the CD model in a setting with at least three domains and three parties. In Section 4, we provide the formal construction of concurrently secure two-domain two-party computation protocol in the CD model (in the setting of 2 parties) and for the proof of its security, we give the construction of a black-box simulator for the protocol in Section 5. We elaborate on the details of the relation with the bare-public key model in Section 6.

2 The Cross-Domain (CD) model

In this section we sketch the details of the CD model. In the CD model, we have multiple domains each consisting of a set of mutually trusting parties and a Key Certification Authority (KCA). Each party in a domain trusts its KCA. Intuitively, whenever a party in one domain wants to jointly compute a function with a party in another domain, each party registers its public key to its own KCA and obtains a certificate on the public key. No party communicates with the KCAs of other domains. Instead, only KCAs communicate with each other to obtain the verification information for the certificates of other parties within other KCAs. Then, every KCA delivers the obtained verification information to the parties in the own domain. The parties use the verification information of other parties received from the trusted KCA throughout the subsequent interaction. We formalize this as an interaction between multiple parties and KCA functionalities as follows. We denote a set of KCA functionalities by $\mathbf{F}_{\mathsf{KCA}} = \{\mathbf{F}^1_{\mathsf{KCA}}, \mathbf{F}^2_{\mathsf{KCA}}, \ldots, \mathbf{F}^N_{\mathsf{KCA}}\}$ where N is the number of domains.

- A party in the i-th domain registers their public key with $\mathbf{F}^i_{\mathsf{KCA}}$. Then, functionality $\mathbf{F}^i_{\mathsf{KCA}}$ generates a pair of signing key and verification key, signs the public key, and returns the verification key and the signature to the party. If $\mathbf{F}^i_{\mathsf{KCA}}$ has already generated a pair of signing key and verification key for the other parties in the domain, then it will use the same signing key and verification key to certify the public key of the current requesting party.
- If an adversary corrupts a party in a domain, then we assume that all other parties in the domain are corrupted as well.[6]

We emphasize that our main aim of the above definition is to protect the privacy of inputs of parties in domains in which no corrupted party exists from

[6] This is because all the parties in a domain trust each other. This captures the toy scenario for Amazon and Walmart described in the introduction. Servers of Amazon and Walmart are seen as the parties in our system.

interacting with corrupted parties in the other domains. The formal definition of the CD model appears in the full version.

3 Impossibility of Concurrent Security in the CD Model

In this section, we provide strong impossibility results ruling out constructions for secure MPC protocols in the CD model. We heavily rely on the recent works of [1, 17] in proving these results. In fact, we show the impossibility result in the simplest case of the CD model: We show that there does not exist a concurrently secure protocol in the CD model of two domains when three domains and three parties exist in the system. Since each party belongs to the distinct domains in the following discussion, we discuss the impossibility result simply focusing on the parties without considering the KCA functionalities.

We start by showing that string OT functionality can not be concurrently and securely realized even in the setting of *static* inputs in the bare-public key model in the setting of *three* parties even against adversaries that corrupt two parties playing the *same role*, i.e. of the sender or the receiver. Next we generalize this impossibility to essentially all functionalities of interest. Finally we extend our impossibility result to the setting of larger number of parties. In particular we show that no n-party protocol in the CD model (for a large class of functionalities, discussed later) can be concurrently secure in the setting of $n+1$ parties. We use the notation used by [17] and some of the texts here have been taken verbatim from [17].

3.1 The Case of String OT

String OT is a two-party functionality between a sender S, with input (m_0, m_1) and a receiver R with input b which allows R to learn m_b without learning anything about m_{1-b}. At the same time the sender S learns nothing about b. More formally string OT functionality $\mathcal{F}_{OT} : (\{0,1\}^{p(k)} \times \{0,1\}^{p(k)}) \times \{0,1\} \rightarrow \{0,1\}^{p(k)}$ is defined as, $\mathcal{F}_{OT}((m_0, m_1), b) = m_b$, where $p(\cdot)$ is any polynomial and only R gets the output.

Note that string OT is a two-party functionality, however, the protocol realizing the string OT functionality can be executed among multiple parties. We consider the setting of three parties and each of the parties registers exactly one key. We show that for some polynomial $p(\cdot)$ (to be fixed later), there does not exist a protocol π that concurrently securely realizes the $\mathcal{F}_{OT}$ functionality among these three parties. More specifically we show that there exists an adversary $\mathcal{A}$ who corrupts 2 parties, registers keys on their behalf, starts a polynomial number of sessions (say $\ell(k)$) of the protocol π with the honest (with pre-specified inputs drawn from a particular distribution $\mathcal{D}$) such that no ideal-world adversary whose output is computationally indistinguishable from the output of real-world adversary $\mathcal{A}$ exists. We stress that the parties corrupted by the adversary (we construct) corrupts two parties playing the same role – either the sender S or the receiver R in all the $\ell(k)$ sessions.

Theorem 1 *(impossibility of static input concurrent-secure string OT in CD model) Let π be any protocol which implements*[7] *the $\mathcal{F}_{OT}$ functionality for a particular (to be determined later) polynomial p in the CD model. Then, in the setting of 3 parties (assuming one-way functions exist) there exists a polynomial ℓ and a distribution $\mathcal{D}$ over ℓ-tuple vectors of inputs and an adversarial strategy $\mathcal{A}$, that corrupts 2 parties, such that for every probabilistic polynomial-time simulation strategy $\mathcal{S}$, (see full-version for formal definition)* cannot *be satisfied when the inputs of the parties are drawn from $\mathcal{D}$.*

Implications for bounded concurrency. Observe that the attack described in the above proof (in the unboundend concurrent setting) has natural implications in the bounded setting as well. In particular, the number of sessions that our adversary executes, or the "extent" of concurrency used by the adversary in the proof above in order to arrive at a contradiction is bounded by the communication complexity of the protocol. More specifically the adversary needs to make one additional OT call for every bit that the Sender sends in the protocol.

3.2 Extending to all asymmetric functionalities

The goal of this section is to generalize the impossibility result for string OT provided in the previous section to all finite deterministic "non-trivial" asymmetric functionalities $\mathcal{F}$. Consider a two-party functionality $\mathcal{F}_{asym}$ between a sender S, with input x and a receiver R with input y which allows R to learn $f(x, y)$ and at the same time S should not learn anything. More formally, let $f : X \times Y \to Z$ be any finite function[8] then an asymmetric functionality $\mathcal{F}_{asym}$ is defined as, $\mathcal{F}_{asym}(x, y) = (\perp, f(x, y))$ where S gets no output and R gets $f(x, y)$. We show that there does not exist a protocol π that concurrently securely realizes any *complete* $\mathcal{F}_{asym}$ functionality as defined below.

$\mathcal{F}_{asym}$ is said to be *complete* [26][9] in the setting of stand-alone two-party computation in the presence of *malicious* adversaries iff $\forall b_0, \exists b_1, a_0, a_1$ such that

$$f(a_0, b_0) = f(a_1, b_0) \wedge f(a_0, b_1) \neq f(a_1, b_1).$$

Lemma 1 (Theorems 1 and 3, [1]). *Given any protocol ρ that concurrently securely realizes a non-trivial asymmetric functionality $\mathcal{F}$ secure under concurrent self-composition in the static-input, fixed-role setting we have that there exists a protocol Π that securely realizes the $\mathcal{F}_{OT}$ functionality secure under concurrent self-composition in the static-input, fixed-role setting.*

The proof of [1] is for the setting of plain model but extends to the setting of the CD model in a direct manner.

[7] We say that a protocol implements a functionality if the protocol allows two parties to evaluate the desired function. This protocol however may not be secure.

[8] Recall that a function is said to be finite if both the domain and the range are of finite size.

[9] Recall that a functionality is said to be complete if it can be used to securely realize any other functionality.

Now any hypothetical protocol for any non-trivial asymmetric functionality $\mathcal{F}$ (using Lemma 1), we will obtain a protocol for $\mathcal{F}_{OT}$, contradicting Theorem 1. This gives our impossibility result for the setting of two parties:

Theorem 2 *(impossibility of static input concurrent security for asymmetric complete functionalities) Let π be any protocol which implements any $\mathcal{F}_{asym}$ functionality that is complete in the stand-alone setting in the CD model. Then, in the setting of 3 parties (assuming one-way functions exist) there exists a polynomial ℓ and a distribution $\mathcal{D}$ over ℓ-tuple vectors of inputs and an adversarial strategy $\mathcal{A}$, that corrupts two parties, such that for every probabilistic polynomial-time simulation strategy $\mathcal{S}$, (see full-version for formal definition)* cannot *be satisfied when the inputs of the parties are drawn from $\mathcal{D}$.*

Extending to n-party protocols So far we have only considered the setting of 3-parties only. We now explain how these results can be extended to the setting of $n+1$ parties executing an n party protocol. Consider an n-party functionality $f(x_1, x_2 \dots x_n)$ with $x_1, x_2 \dots x_n$ as input. Let S and $\overline{S}$ be disjoint partitions of the n parties such that only a subset of the parties in $\overline{S}$ get the outputs. Let g be a two-argument function obtained by viewing f as a function of $\{x_i\}_{i \in S}$ and $\{x_i\}_{i \in \overline{S}}$. For any f, if there exist such partitions S and $\overline{S}$ such that g is a complete two-party asymmetric functionality,[10] then we can use our impossibility result for concurrently securely realizing g in the CD model in the setting of 3 parties to argue that f can not be concurrently securely realized in CD model in the setting of $n+1$ parties. The proof follows in a very similar manner and we omit the details.

4 Possibility of Concurrent Security in the CD model

In this section, we present the positive side of our result by constructing a *constant-round* concurrently secure MPC protocol in the CD model with *black-box simulation.* Our protocol, the ingredients needed and the proof build upon the construction of [16] (and its full version) and parts of the texts have been taken verbatim from there without explicitly mentioning again and again. Let $\mathcal{F}$ be a well-formed functionality where such a functionality admits a constant-round two-party computation protocol in the semi-honest setting.[11] In fact, for simplicity, we present a constant-round concurrently secure two-party computation protocol in the CD model, denoted by Π, where a party belongs to either of the two domains.

We emphasize that this two-party protocol easily extends to a concurrently secure protocol for any polynomially many parties in CD model of two fixed domains where a party is under either of two domains. Subsequently, our protocol easily extends to a concurrently secure protocol for any polynomially many

[10] Note here this implies that at least one party in S and at least one party in $\overline{S}$ has an input.

[11] See [7] for the notion of well-formed functionality.

parties in CD model of N fixed domains where each party belongs to one of the N domains.

4.1 Building Blocks and Notations

Due to the space restrictions, see the full version for the details of the building blocks. Let $g : \{0,1\}^n \rightarrow \{0,1\}^{3n}$ be a length tripling pseudo-random generator. Let $\textsc{PBcom}(\cdot)$ denote a non-interactive perfectly binding commitment scheme, and let $\langle C, R \rangle$ denote an one-slot extractable commitment scheme . Furthermore, we will denote a constant round strong WI proof system by $\langle P, V \rangle$ and a special constant-round NMWI argument of knowledge protocol by $\langle P', V' \rangle$. Finally we denote a constant-round SWI argument by $\langle P_{\mathsf{swi}}, V_{\mathsf{swi}} \rangle$, and a constant-round *semi-honest* two-party computation protocol by $\langle P_1^{\mathsf{sh}}, P_2^{\mathsf{sh}} \rangle$ which securely computes $\mathcal{F}$ as per the standard simulation-based definition of secure computation.

4.2 Construction of our protocol

We now provide the formal construction of concurrently secure two-party computation protocol in the CD model. Some notations and the protocol description closely resemble those of [16]. Let $\mathbf{F}_{\mathsf{KCA}} = \{\mathbf{F}^1_{\mathsf{KCA}}, \mathbf{F}^2_{\mathsf{KCA}}\}$ be the key certification authority(KCA) functionality with two domains in the CD model, which is a special case of $\mathbf{F}_{\mathsf{KCA}}$ where $N = 2$. See the formal description in full-version.

Let n be the security parameter. Let P_1 and P_2 be two parties with private inputs x_1 and x_2 respectively. Without loss of generality, let P_1 and P_2 be in the domains $\mathbf{F}^1_{\mathsf{KCA}}$ and $\mathbf{F}^2_{\mathsf{KCA}}$ respectively. Also, P_1 and P_2 have unique identifiers id_1 and id_2 respectively. Protocol $\Pi = \langle P_1, P_2 \rangle$ proceeds as follows. We omit session identifiers for the succinct specification.
I. Key Registration Phase.

1. P_1 samples random strings sk_1^0 and sk_1^1 and sets $\mathsf{pk}_1^0 := g(\mathsf{sk}_1^0)$ and $\mathsf{pk}_1^1 := g(\mathsf{sk}_1^1)$.
2. P_1 registers both public keys pk_1^0 and pk_1^1 by sending $(register, \mathsf{id}_1, \mathsf{pk}_1^0)$ and $(register, \mathsf{id}_1, \mathsf{pk}_1^1)$ to functionality $\mathbf{F}^1_{\mathsf{KCA}}$.[12]
3. P_1 obtains $(\sigma_{\mathsf{pk}_1^0}, mvk_1)$ and $(\sigma_{\mathsf{pk}_1^1}, mvk_1)$ from $\mathbf{F}^1_{\mathsf{KCA}}$ where $\sigma_{\mathsf{pk}_1^0}$ and $\sigma_{\mathsf{pk}_1^1}$ are signatures on pk_1^0 and pk_1^1 respectively where mvk_1 is the respective verification key.
4. P_1 chooses a random bit $b_1 \in \{0,1\}$ and sets $\mathsf{pk}_1 = \mathsf{pk}_1^{b_1}$ and $\mathsf{sk}_1 = \mathsf{sk}_1^{b_1}$. We now denote the corresponding signature by σ_{pkpo}.
5. P_2 acts analogously, registers pk_2^0 and pk_2^1 with $\mathbf{F}^2_{\mathsf{KCA}}$, and obtains $(\sigma_{\mathsf{pk}_2^0}, mvk_2)$ and $(\sigma_{\mathsf{pk}_1}, mvk_2)$. It sets $\mathsf{pk}_2 = \mathsf{pk}_2^{b_2}$ and $\mathsf{sk}_2 = \mathsf{sk}_2^{b_2}$ where b_2 is a random bit. Finally, σ_{pkpt} is the corresponding signature.

[12] The registration request is not required to be two distinct requests to the functionality. Registering pk_1^0 and pk_1^1 can be viewed as a registering one public key which is a concatenation of two public keys and the functionality simply decomposes it into two strings, signs both and returns them to the party.

II. Trapdoor Creation Phase. Let $\mathcal{R}_1$ be a NP-relation where the NP theorem is a string mvk and the witness is a tuple $(\tilde{pk}, \tilde{sk}, \tilde{\sigma}, \tilde{c})$ such that $(mvk, \tilde{pk}, \tilde{sk}, \tilde{\sigma}, \tilde{c}) \in \mathcal{R}_1$ if and only if $\tilde{c}$ is the commitment to $\tilde{pk}||\tilde{sk}||\tilde{\sigma}$ with respect to protocol $\langle C, R\rangle$, $\tilde{pk} = g(\tilde{sk})$, and $\mathsf{Ver}(\tilde{pk}, \tilde{\sigma}, mvk) = 1$. For convenience, we let $(mvk, t, \tilde{c}) \in \mathcal{R}_1$ if $t = \tilde{pk}||\tilde{sk}||\tilde{\sigma}$ and $(mvk, \tilde{pk}, \tilde{sk}, \tilde{\sigma}, \tilde{c}) \in \mathcal{R}_1$. In addition, let $\mathcal{R}_2$ be a NP-relation where the NP theorem is a string mvk and the witness is a tuple $(\tilde{pk}, \tilde{sk}, \tilde{\sigma})$ such that $(mvk, \tilde{pk}, \tilde{sk}, \tilde{\sigma}) \in \mathcal{R}_2$ if and only if $\tilde{pk} = g(\tilde{sk})$ and $\mathsf{Ver}(\tilde{pk}, \tilde{\sigma}, mvk) = 1$. Similarly, we denote we let $(mvk, t) \in \mathcal{R}_2$ if $t = \tilde{pk}||\tilde{sk}||\tilde{\sigma}$ and $(mvk, \tilde{pk}, \tilde{sk}, \tilde{\sigma}) \in \mathcal{R}_2$. The trapdoor creation phase proceeds as follows.

1. $P_1 \Rightarrow P_2$: P_1 sends a request $(\mathit{retrieval}, \mathsf{id}_2)$ to $\mathbf{F}^1_{\mathsf{KCA}}$ and obtains mvk_2, a verification key from $\mathbf{F}^1_{\mathsf{KCA}}$. Recall that $\mathbf{F}^1_{\mathsf{KCA}}$ obtains mvk_2 by interacting with $\mathbf{F}^2_{\mathsf{KCA}}$. P_2 analogously obtains mvk_1 from $\mathbf{F}^2_{\mathsf{KCA}}$.
2. $P_1 \Rightarrow P_2$: P_1 executes $\langle C, R\rangle$ with P_2, where P_1 commits to $\mathsf{trap}_1 = \mathsf{pk}_1||\mathsf{sk}_1||\sigma_{\mathsf{pk}_1}$. We denote this execution by $\langle C, R\rangle^{\mathsf{trap}_1}_{1\to 2}$ and the commitment by $\tilde{c_1}$. Next P_1 proves to P_2 by using strong WI proof system $\langle P, V\rangle$ with common input mvk_1, the following NP-statement: there exists $(\mathsf{pk}_1, \mathsf{sk}_1, \sigma_{\mathsf{pk}_1})$ where $(mvk_1, \mathsf{pk}_1, \mathsf{sk}_1, \sigma_{\mathsf{pk}_1}) \in \mathcal{R}_1$. If the verifier V in $\langle P, V\rangle^{\mathsf{sk}_1}_{1\to 2}$ aborts, then P_2 aborts. We denote this execution by $\langle P, V\rangle^{\mathsf{trap}_1}_{1\to 2}$.
3. $P_2 \Rightarrow P_1$: P_2 acts analogously in Step 2 by first committing to $\mathsf{trap}_2 = \mathsf{pk}_2||\mathsf{sk}_2||\sigma_{\mathsf{pk}_2}$ using $\langle C, R\rangle$ and then giving a proof using $\langle P, V\rangle$. We denote this execution by $\langle C, R\rangle^{\mathsf{trap}_2}_{2\to 1}$ and $\langle P, V\rangle^{\mathsf{trap}_2}_{2\to 1}$.
4. $P_1 \Rightarrow P_2$: P_1 commits to bit 0 as $com_1 = \textsc{PBcom}(0)$ and sends com_1 to P_2. Next P_1 and P_2 executes constant-round NMWI argument of knowledge $\langle P', V'\rangle$ in which P_1 and P_2 respectively play as P' and V'. The common inputs for this execution of $\langle P', V'\rangle$ are com_1 and mvk_2. In this execution, P_1 proves to P_2 that com_1 is a commitment to 0 or there exists a string t such that $(mvk_2, t) \in \mathcal{R}_2$. Honest party P_1's private input is the de-commitment information of com_1.[13] That is, by the execution of $\langle P', V'\rangle$, P_1 proves to P_2 that com_1 is a commitment to bit 0.
5. $P_2 \Rightarrow P_1$: P_2 proceeds symmetrically as does P_1 above. In summary, it generates a commitment com_2 to bit 0 and then proves the same using $\langle P', V'\rangle$.

III. Input Commitment Phase.

Let $\mathsf{Enc}_{\mathsf{pk}}(\cdot)$ denote the encryption algorithm of an dense encryption scheme with pseudo-random public keys with public-keys of length ℓ.

1. $P_1 \Leftrightarrow P_2$: P_1 samples a random string $\alpha_1 \in \{0,1\}^\ell$ and sends $c_1' = \textsc{PBcom}(\alpha_1)$ to P_2. Upon receiving c_1', P_2 responds with a random string $\beta_1 \in \{0,1\}^\ell$. At this point, P_1 reveals the value α_1 to P_2 and proves the following NP-statement to P_2 by executing $\langle P_{\mathsf{swi}}, V_{\mathsf{swi}}\rangle$:
   (a) *either* there exists randomness such that c_1' is a commitment to the string α_1,

[13] Looking ahead the secret key corresponding to the public key pk_2 will allow the simulator to cheat in the simulation.

   (b) *or* com_1 is a commitment to 1.

   Both parties set $\mathsf{pk}_\mathsf{c}^1 = \alpha_1 \oplus \beta_1$ (public key generated using the coin flipping).
2. $P_2 \Leftrightarrow P_1$: P_2 and P_1 proceed symmetrically as above to generate the public key $\mathsf{pk}_\mathsf{c}^2 = \alpha_2 \oplus \beta_2$.
3. $P_1 \Rightarrow P_2$: P_1 samples a random string r_1 of appropriate length which is to be used as randomness to execute semi-honest two-party computation $\langle P_1^{\mathsf{sh}}, P_2^{\mathsf{sh}} \rangle$. P_1 computes $y_1 = \mathsf{Enc}_{\mathsf{pk}_\mathsf{c}^2}(x_1 || r_1)$. Then, it sends y_1 to P_2.
4. $P_2 \Rightarrow P_1$: P_2 proceeds symmetrically as does P_1 above. Let x_2 and r_2 be the input and the random string chosen by P_2 to be used in the execution of $\langle P_1^{\mathsf{sh}}, P_2^{\mathsf{sh}} \rangle$. Let $y_2 = \mathsf{Enc}_{\mathsf{pk}_\mathsf{c}^1}(x_2 || r_2)$ be the cipher-text generated.

IV. Secure Computation Phase. In the secure computation phase, parties P_1 and P_2 jointly evaluate the desired functionality $\mathcal{F}$ based on a constant-round semi-honest two-party computation protocol $\langle P_1^{\mathsf{sh}}, P_2^{\mathsf{sh}} \rangle$. Party P_1 plays P_1^{sh} while party P_2 plays P_2^{sh}. Note that $\langle P_1^{\mathsf{sh}}, P_2^{\mathsf{sh}} \rangle$ is secure against semi-honest adversaries. Thus, we require that the coins of participating parties are indeed uniform. Moreover, we require each party to prove the validity of every message it sends to the other party. That is, whenever a party generates and sends a message to the other party, it is required to prove by using $\langle P_{\mathsf{swi}}, V_{\mathsf{swi}} \rangle$ that the message is honestly generated with respect to its input, random coins and the instructions of $\langle P_1^{\mathsf{sh}}, P_2^{\mathsf{sh}} \rangle$. In the following, let t be the round complexity of $\langle P_1^{\mathsf{sh}}, P_2^{\mathsf{sh}} \rangle$ where each round consists of two messages: w.l.o.g. a message from P_1 followed by a message from P_2. We denote the next message generators of $\langle P_1^{\mathsf{sh}}, P_2^{\mathsf{sh}} \rangle$ simply by P_1^{sh} and P_1^{sh}. We define transcript $T_{1,i}$ (resp., $T_{2,i}$) by the set (or vector) of all the messages (belonging to $\langle P_1^{\mathsf{sh}}, P_2^{\mathsf{sh}} \rangle$) which are exchanged between P_1 and P_2 before P_1 (resp., P_2) needs to send the i-th round message of $\langle P_1^{\mathsf{sh}}, P_2^{\mathsf{sh}} \rangle$ for $i \in [t]$. In particular, P_1 obtains the i-th round message, denoted by $\beta_{1,i}$, of $\langle P_1^{\mathsf{sh}}, P_2^{\mathsf{sh}} \rangle$ as it computes $\beta_{1,i} = P_1^{\mathsf{sh}}(T_{1,i}, x_1, r_1'')$. The P_2's i-th message is symmetrically defined as $\beta_{2,i} = P_1^{\mathsf{sh}}(T_{2,i}, x_2, r_2'')$. The formal definition of the secure computation phase is provided as follows.

1. $P_1 \Rightarrow P_2$: P_1 samples a random string r_2' of appropriate length and sends it to P_2.
2. $P_1 \Leftarrow P_2$: P_2 similarly samples a random string r_1' of appropriate length and sends it to P_1.
3. P_1 computes $r_1'' = r_1 \oplus r_1'$ and P_2 computes $r_2'' = r_2 \oplus r_2'$. Then, r_1'' and r_2'' are the random coins to be used respectively by P_1 and P_2 in the execution of $\langle P_1^{\mathsf{sh}}, P_2^{\mathsf{sh}} \rangle$.
4. For $i \in [t]$, parties P_1 and P_2 repeats the following procedure.
   (a) $P_1 \Rightarrow P_2$: P_1 computes $\beta_{1,i} = P_1^{\mathsf{sh}}(T_{1,i}, x_1, r_1'')$ and send it to P_2.
   (b) $P_1 \Rightarrow P_2$: P_1 proves to P_2 by using $\langle P_{\mathsf{swi}}, V_{\mathsf{swi}} \rangle$, the NP-statement which is a disjunction of the following NP-statements:
      i. There exist values $\hat{x}_1$, $\hat{r}_1$ such that
         A. there exists randomness such that $y_1 = \mathsf{Enc}_{\mathsf{pk}_\mathsf{c}^2}(\hat{x}_1 || \hat{r}_1)$
         B. and $\beta_{1,i} = P_1^{\mathsf{sh}}(T_{1,i}, \hat{x}_1, \hat{r}_1 \oplus r_1')$
      ii. com_1 is a commitment to bit 1.

(c) $P_2 \Rightarrow P_1$: P_2 acts symmetrically.

This completes the formal definition of protocol Π. We claim the following.

Theorem 3 *If there exist a constant-round semi-honest OT, an encryption system with dense(pseudo-random) public keys, and a family of collision-resistant hash functions, then there exists a constant-round concurrently secure two-party computation protocol for every well-formed functionality $\mathcal{F}$ in the CD model.*

5 Proof of Theorem 3 (Simulator $\mathcal{S}$)

In this section, we prove Theorem 3 by constructing an Expected Probabilistic Polynomial-Time (EPPT) simulator $\mathcal{S}$ for protocol Π. That is, the EPPT simulator $\mathcal{S}$ with a black-box access to the adversary $\mathcal{A}$ simulates the view of adversary which is computationally indistinguishable from the view of adversary interacting with a honest party in the real world execution of Π. Here we will only give a description of our simulator and refer the reader to the full version for a formal proof of indistinguishability.

Notice that the NP-statement for an instance of SWI (in Step 4b of Secure Computation Phase) is a disjunction of two NP-statements (Statement 4(b)i and Statement 4(b)ii). In the rest of the work, we refer to Statement 4(b)i as *real* theorem while we refer to Statement 4(b)ii as the *trapdoor* theorem. We call the witness corresponding to statement 4(b)i (resp. statement 4(b)ii) as *real* (resp. *trapdoor*) witness.

Notation. In the following, we denote the honest party and the adversary by H and $\mathcal{A}$ respectively. Also, let $\mathbf{F}^{H}_{\mathsf{KCA}}$ be the domain to which the honest party belong. Similarly, we use $\mathbf{F}^{\mathcal{A}}_{\mathsf{KCA}}$ to denote the domain where the adversary corrupts a party. Without loss of generality, we define our simulator in the case where the honest party (thus, the simulator in the following) sends the first message in the protocol. We omit the other case where the corrupted party sends the first message. Let $m = \mathsf{poly}(n)$ be the running time of the PPT adversary $\mathcal{A}$. And let l be the number of public keys registered by the corrupted party. The running time of $\mathcal{A}$ serves as an upper-bounds on the number of concurrent sessions and also on the number of registered public keys. In the course of simulation, simulator $\mathcal{S}$ maintains two sets denoted by `Database1` and `Database2`. `Database1` contains an element of the form $(\mathsf{pk}, \mathsf{sk}, \sigma_{pk})$ for $i \in [l]$. `Database2` contains elements of the form $(\mathsf{sid}, x_i^{\mathsf{sid}}, r_i^{\mathsf{sid}})$ where $\mathsf{sid} \in [m]$ and $i \in [l]$. Initially, `Database1` and `Database2` are set to be empty. We sometimes omit the session identifier sid in order to simplify notations.

We preserve the notations for the execution of building blocks as in Section 4.2. For example, we denote by $\langle P, V\rangle_{\mathcal{S}\to\mathcal{A}}$, an instance of $\langle P, V\rangle$ where simulator $\mathcal{S}$ and corrupted party $\mathcal{A}$ play as the prover P and the verifier V respectively in the execution of the protocol $\langle P, V\rangle$. We demarcate the following two *special* messages in the protocol Π:

– **Message Σ_1^{sid}**: Σ_1^{sid} denotes the second message of $\langle C, R\rangle^{\mathsf{trap}_{\mathcal{A}}}_{\mathcal{A}\to\mathcal{S}}$ in session sid. Recall that the second message of the protocol $\langle C, R\rangle$ is a random

string (challenge) from the receiver to the committer. In the execution of $\langle C, R\rangle^{\mathsf{trap}_{\mathcal{A}}}_{\mathcal{A}\to\mathcal{S}}$, this message is sent by the simulator (on behalf of H) to the adversary $\mathcal{A}$.
– **Message $\boldsymbol{\Sigma}_2^{\mathsf{sid}}$**: $\boldsymbol{\Sigma}_2^{\mathsf{sid}}$ denotes the message of session sid when the simulator (on behalf of the honest party H) sends the commitment to 0 using the commitment scheme PBcom. The simulator will behave honestly until this point and will cheat only after this point is reached.

Description of $\mathcal{S}$. We provide the simulation strategy of $\mathcal{S}$ in each phase of Π as follows.

I. Simulation of Key Registration Phase: In the key registration phase, simulator $\mathcal{S}$ follows an honest party's strategy. That is, $\mathcal{S}$ interacting with $\mathbf{F}^{H}_{\mathsf{KCA}}$ registers public keys $\mathsf{pk}^0_{\mathcal{S}}$ and $\mathsf{pk}^1_{\mathcal{S}}$ (on behalf of the honest party H) where $(\mathsf{pk}^0_{\mathcal{S}}, \mathsf{sk}^0_{\mathcal{S}})$ and $(\mathsf{pk}^1_{\mathcal{S}}, \mathsf{sk}^1_{\mathcal{S}})$ are obtained as in the honest setting. Finally, $\mathcal{S}$ completes the simulation of key registration phase by setting $\mathsf{pk}_{\mathcal{S}}$, $\mathsf{sk}_{\mathcal{S}}$, and $\sigma_{\mathsf{pk}_{\mathcal{S}}}$ following the honest strategy.

II. Simulation of Trapdoor Creation Phase: Simulator $\mathcal{S}$ behaves according to the honest party strategy until it needs to send the $\boldsymbol{\Sigma}_2^{\mathsf{sid}}$ for some session session $\mathsf{sid} \in [m]$. At this point, $\mathcal{S}$ by interacting with $\mathbf{F}^{H}_{\mathsf{KCA}}$ obtained a verification key $mvk_{\mathcal{A}}$ of $\mathbf{F}^{\mathcal{A}}_{\mathsf{KCA}}$. To successfully simulate trapdoor creation phase, $\mathcal{S}$ wants to do the following:

1. For all sessions, $\mathcal{S}$ commits to 1 (recall that this differs from the real execution in the fact that honest party commit to 0) by executing $com_{\mathcal{S}} = \text{PBcom}(1)$ and then sends $com_{\mathcal{S}}$ in to the adversary.
2. For all sessions, $\mathcal{S}$ proves to $\mathcal{A}$ by executing $\langle P', V'\rangle_{\mathcal{S}\to\mathcal{A}}$ using a trapdoor information $\mathsf{trap}_{\mathcal{A}}$ (stored in the database `Database1`) as its witness that $(mvk_{\mathcal{A}}, \mathsf{trap}_{\mathcal{A}}) \in \mathcal{R}_2$.

Thus, before sending the commitment to 1, $\mathcal{S}$ checks if `Database1` contains $\mathsf{trap}_{\mathcal{A}}$ such that $(mvk_{\mathcal{A}}, \mathsf{trap}_{\mathcal{A}}) \in \mathcal{R}_2$. If so, then $\mathcal{S}$ proceeds as above. Otherwise, $\mathcal{S}$ employs a rewinding strategy to extract the trapdoor information. Note that session sid (called *target session*) is the session in which the simulator needs to send the commitment to bit 1 using the commitment scheme PBcom without the corresponding trapdoor information in `Database1`. We will denote this session by sid^{target}. When this point is reached, our simulator $\mathcal{S}$ executes the following look-ahead thread strategy.[14]

1. $\mathcal{S}$ rewinds adversary $\mathcal{A}$ back to the point before $\mathcal{S}$ had sent $\boldsymbol{\Sigma}_1^{\mathsf{sid}^{target}}$ to $\mathcal{A}$.
2. In the look-ahead thread, the simulator $\mathcal{S}$ sends to $\mathcal{A}$ a fresh random challenge for the message $\boldsymbol{\Sigma}_1^{\mathsf{sid}^{target}}$ and behaves honestly subsequently. If in this look-ahead thread, the first session in which the simulator needs to send $\boldsymbol{\Sigma}_2^{\mathsf{sid}}$ is not the target session (in other words $\mathsf{sid} \neq \mathsf{sid}^{target}$), then $\mathcal{S}$ rewinds again and repeats this step. If the number of rewindings reaches 2^n, then $\mathcal{S}$ aborts completely and outputs `Rewind Abort`.

[14] Note that the transcript generated by the execution of look-ahead threads will not be included in the view of the main thread simulation.

3. Since $\mathcal{S}$ need to send $\Sigma_2^{\mathsf{sid}^{target}}$ in both the main thread and the rewound thread, it must have obtained two distinct *valid* de-commitments of $\langle C, R\rangle_{\mathcal{A}\to\mathcal{S}}^{\mathsf{trap}_{\mathcal{A}}}$ in the target session sid^{target} in both the main thread and the look-ahead threads. At this point, using two distinct valid de-commitments, $\mathcal{S}$ obtains $\mathsf{trap}_{\mathcal{A}}$. $\mathcal{S}$ executes the rest of the main concurrent execution with the updated $\texttt{Database1}$. Notice that a single successful extraction of $\mathsf{trap}_{\mathcal{A}}$ in one session suffices to simulate all other sessions.

III. Simulation of Input Commitment Phase.

1. The simulator behaves honestly in the generation of the public key $\mathsf{pk}_c^{\mathcal{A}}$.
2. Now, we describe simulation strategy in generation of the public key $\mathsf{pk}_c^{\mathcal{S}}$. $\mathcal{S}$ starts by generating a a fresh public key $\mathsf{pk}_c^{\mathcal{S}}$ along with the secret key $\mathsf{sk}_c^{\mathcal{S}}$. It generates the commitment $c'_{\mathcal{S}}$ as the commitment to the zero string. Then, $\mathcal{S}$ receives $\beta_{\mathcal{S}}$ from $\mathcal{A}$. Finally $\mathcal{S}$ opens $\alpha_{\mathcal{S}}$ as $\mathsf{pk}_c^{\mathcal{S}} \oplus \beta_{\mathcal{S}}$. $\mathcal{S}$ executes $\langle P_{\mathsf{swi}}, V_{\mathsf{swi}}\rangle_{\mathcal{S}\to\mathcal{A}}$ where $\mathcal{S}$ uses the trapdoor witness. $\mathcal{S}$ possesses the trapdoor witness since it committed to bit 1 instead of 0 during the simulation of the trapdoor creation phase.
3. $\mathcal{S}$ generates $y_{\mathcal{S}}$ as encryption of the zero string using the public key $\mathsf{pk}_{\mathsf{c}}^{\mathcal{A}}$ and sends it to the adversary. (instead of using its actual input and random coins needed for the semi-honest two-party computation)
4. Upon the receiving $y_{\mathcal{A}}$, the simulator $\mathcal{S}$ extracts the input and randomness $x_{\mathcal{A}}^{\mathsf{sid}}$ and $r_{\mathcal{A}}^{\mathsf{sid}}$ of $\mathcal{A}$ using the secret key $\mathsf{sk}_c^{\mathcal{S}}$. Now, $\mathcal{S}$ adds $(\mathsf{sid}, x_{\mathcal{A}}^{\mathsf{sid}}, r_{\mathcal{A}}^{\mathsf{sid}})$ to $\texttt{Database2}$.

IV. Simulation of Secure Computation Phase. Let S_{sh} denote the simulator for the semi-honest two-party protocol $\langle P_1^{\mathsf{sh}}, P_2^{\mathsf{sh}}\rangle$ used in our construction. $\mathcal{S}$ internally runs simulator S_{sh} on adversary $\mathcal{A}_{\mathsf{sh}}$'s input $x_{\mathcal{A}} \in \texttt{Database2}$. S_{sh} at some point makes a call to ideal functionality $\mathcal{F}$ in the ideal world with an input string $x_{\mathcal{A}}$. Then, $\mathcal{S}$ makes a query $(\mathsf{sid}, x_{\mathcal{A}})$ to $\mathcal{F}$. Then, $\mathcal{S}$ forwards the output returned by $\mathcal{F}$ to S_{sh}. At some point of internal simulation of $\langle P_1^{\mathsf{sh}}, P_2^{\mathsf{sh}}\rangle$, S_{sh} finally halts and outputs a transcript $\beta_{S_{\mathsf{sh}},1}, \beta_{\mathcal{A}_{\mathsf{sh}},1}, \ldots, \beta_{S_{\mathsf{sh}},t}, \beta_{\mathcal{A}_{\mathsf{sh}},t}$ and associated random coin $\hat{r}_{\mathcal{A}}$. $\mathcal{S}$ proceeds with the following instructions.

1. $\mathcal{S}$ computes a random string $\tilde{r}_{\mathcal{A}}$ such that $\tilde{r}_{\mathcal{A}} = r_{\mathcal{A}} \oplus \hat{r}_{\mathcal{A}}$. Then, $\mathcal{S}$ sends $\tilde{r}_{\mathcal{A}}$ to $\mathcal{A}$.
2. For each round $j \in [t]$, $\mathcal{S}$ sends $\beta_{S_{\mathsf{sh}},j}$ to $\mathcal{A}$. Then, $\mathcal{S}$ executes $\langle P_{\mathsf{swi}}, V_{\mathsf{swi}}\rangle_{\mathcal{S}\to\mathcal{A}}$ with $\mathcal{A}$ where $\mathcal{S}$ uses the trapdoor witness, decommitment information of $com_{\mathcal{S}}$ (commitment to 1 instead of 0). If $\mathcal{A}$ aborts upon $\beta_{S_{\mathsf{sh}},j}$ for some $j \in [t]$, $\mathcal{S}$ outputs a special abort message $\texttt{ABORT}_1$.
3. Upon receiving $\mathcal{A}$'s next message $\beta_{\mathcal{A},j}$ in the protocol $\langle P_1^{\mathsf{sh}}, P_2^{\mathsf{sh}}\rangle$, $\mathcal{S}$ plays the honest verifier in an execution of $\langle P_{\mathsf{swi}}, V_{\mathsf{swi}}\rangle_{\mathcal{A}\to\mathcal{S}}$. For any $j \in [t]$, if the j^{th} message $\beta_{\mathcal{A},j}$ sent by adversary $\mathcal{A}$ is not identical to $\beta_{\mathcal{A}_{\mathsf{sh}},j}$ (obtained from the internal execution of S_{sh}) and if $\langle P_{\mathsf{swi}}, V_{\mathsf{swi}}\rangle_{\mathcal{A}\to\mathcal{S}}$ on $\beta_{\mathcal{A},j}$ is accepting, then $\mathcal{S}$ aborts and outputs a special abort message $\texttt{ABORT}_2$.

Finally, the output of simulator $\mathcal{S}$ contains all messages exchanged between the simulator and the adversary including the output of the adversary in the communication of all sessions.

6 Relation with the Bare-Public Key (BPK) model

The CD model defined in this paper is a generalization of the BPK model introduced by Canetti et al. [9]. In the BPK model each party sets up its own public-key and private-key pair. It publishes its public-key in a public file while keeping the private-key secret. This phase of publishing the public-keys happens prior to any protocol executions, implicitly also placing a bound on the number of parties in the system.

The CD model is a generalization of the BPK model, where each party corresponding to the BPK model is now associated with a domain of mutually trusting entities, equipped with a key registration authority. A key registration authority in the CD model generates a common public-key for all entities in its domain and issues a private-key for each of these entities. Just as in the BPK model in which the number of parties are bounded, the CD model bounds the number of domains while putting no bound on the number of parties.

As a real-world example, consider a setting of the BPK model where one of the parties owns multiple (physically distinct) devices and would like to use each of these devices for various secure computation tasks. In the CD model, each one of these devices is seen as a separate entity and the owner who generates and distributes the keys across these devices is seen as the key registration authority.

In the BPK model, one party could coordinate between different concurrent executions that it takes part in. For example, a party could ensure that it takes part in all the protocol executions sequentially and hence avoid all the problems that arise because of concurrent executions. This coordination is certainly not desirable but might very well be acceptable in various real world applications. On the other hand in our CD model, different entities represent possibly separate devices, coordinating which is not possible. The key advantage of the CD model over the BPK model is that it makes this distinction in functionality clear.

Finally, we note that our results in the CD model, directly imply positive and negative results in the BPK model. We stress that even in the BPK model prior to our work no results for the setting of secure computation were known and our results fully characterize what is possible in this model. More formally, these results are directly implied by the following lemma.

Lemma 2. *There exists an n-party black-box concurrently secure protocol Π among n-domains in the CD-model where each party is associated with distinct domains if and only if there exists an n-party black-box concurrently secure protocol Π' in the BPK model with n parties.*

Proof. We give a proof sketch here. We start by giving a protocol Π' secure in the BPK model given a protocol Π secure in the CD-model. Each party in protocol Π' that we are trying to construct executes the public setup of the key-registration authority of protocol Π and generates the private-key assuming only one entity in its domain. Subsequently to this setup phase, parties in Π' execute all concurrent execution as a party of Π using the secret key that it had generated earlier as the key-registration authority. Security of the protocol Π' follows immediately. The other direction can be argued in a similar manner.

In particular, the above lemma along with our results in the CD model implies that there exists an n-party concurrently secure protocol in the BPK model if and only if there are exactly n parties in the system.

References

1. Agrawal, S., Goyal, V., Jain, A., Prabhakaran, M., Sahai, A.: New impossibility results for concurrent composition and a non-interactive completeness theorem for secure computation. In: CRYPTO. pp. 443–460 (2012)
2. Barak, B., Canetti, R., Nielsen, J., Pass, R.: Universally composable protocols with relaxed set-up assumptions. In: FOCS. pp. 186–195 (2004)
3. Barak, B., Prabhakaran, M., Sahai, A.: Concurrent non-malleable zero knowledge. In: FOCS. pp. 345–354 (2006)
4. Barak, B., Sahai, A.: How to play almost any mental game over the net - concurrent composition via super-polynomial simulation. In: FOCS. pp. 543–552 (2005)
5. Beimel, A., Malkin, T., Micali, S.: The all-or-nothing nature of two-party secure computation. In: CRYPTO. pp. 80–97 (1999)
6. Canetti, R., Kushilevitz, E., Lindell, Y.: On the limitations of universally composable two-party computation without set-up assumptions. J. Cryptology 19(2), 135–167 (2006)
7. Canetti, R., Lindell, Y., Ostrovsky, R., Sahai, A.: Universally composable two-party and multi-party secure computation. In: STOC. pp. 494–503 (2002)
8. Canetti, R., Fischlin, M.: Universally composable commitments. In: CRYPTO. pp. 19–40 (2001)
9. Canetti, R., Goldreich, O., Goldwasser, S., Micali, S.: Resettable zero-knowledge (extended abstract). In: STOC. pp. 235–244 (2000)
10. Canetti, R., Lin, H., Pass, R.: Adaptive hardness and composable security in the plain model from standard assumptions. In: FOCS. pp. 541–550 (2010)
11. Canetti, R., Pass, R., Shelat, A.: Cryptography from sunspots: How to use an imperfect reference string. In: FOCS. pp. 249–259 (2007)
12. Chandran, N., Goyal, V., Sahai, A.: New constructions for uc secure computation using tamper-proof hardware. In: EUROCRYPT. pp. 545—562 (2008)
13. Dwork, C., Naor, M., Sahai, A.: Concurrent zero-knowledge. In: STOC. pp. 409–418 (1998)
14. Garay, J.A., MacKenzie, P.D.: Concurrent oblivious transfer. In: FOCS. pp. 314–324 (2000)
15. Garg, S., Goyal, V., Jain, A., Sahai, A.: Bringing people of different beliefs together to do uc. In: TCC. pp. 311–328 (2011)
16. Garg, S., Goyal, V., Jain, A., Sahai, A.: Concurrently secure computation in constant rounds. In: EUROCRYPT. pp. 99–116 (2012)
17. Garg, S., Kumarasubramanian, A., Ostrovsky, R., Visconti, I.: Impossibility results for static input secure computation. In: CRYPTO. pp. 424–442 (2012)

18. Goldreich, O., Micali, S., Wigderson, A.: How to play any mental game or a completeness theorem for protocols with honest majority. In: STOC. pp. 218–229 (1987)
19. Goyal, V.: Positive results for concurrently secure computation in the plain model. In. FOCS. pp. 41–50 (2012)
20. Goyal, V., Jain, A., Ostrovsky, R.: Password-authenticated session-key generation on the internet in the plain model. In: CRYPTO. pp. 277–294 (2010)
21. Goyal, V., Katz, J.: Universally composable multi-party computation with an unreliable common reference string. In: TCC. pp. 142–154 (2008)
22. Groth, J., Ostrovsky, R.: Cryptography in the multi-string model. In: CRYPTO. pp. 323–341 (2007)
23. Jain, A., Ostrovsky, R., Richelson, S., Visconti, I.: Concurrent zero knowledge in the bounded player model. Cryptology ePrint Archive, Report 2012/279 (2012), `http://eprint.iacr.org/`
24. Katz, J.: Universally composable multi-party computation using tamper-proof hardware. In: Eurocrypt. pp. 115–128 (2007)
25. Kilian, J.: Founding cryptography on oblivious transfer. In: STOC. pp. 20–31 (1988)
26. Kilian, J.: More general completeness theorems for secure two-party computation. In: STOC. pp. 316–324 (2000)
27. Kilian, J., Kushilevitz, E., Micali, S., Ostrovsky, R.: Reducibility and completeness in private computations. SIAM J. Comput. 29(4), 1189–1208 (2000)
28. Kilian, J., Petrank, E.: Concurrent and resettable zero-knowledge in polyloalgorithm rounds. In: STOC. pp. 560–569 (2001)
29. Kushilevitz, E., Micali, S., Ostrovsky, R.: Reducibility and completeness in multiparty private computations. In: FOCS. pp. 478–489 (1994)
30. Lin, H., Pass, R., Venkitasubramaniam, M.: A unified framework for concurrent security: universal composability from stand-alone non-malleability. In: STOC. pp. 179–188 (2009)
31. Lindell, Y.: Lower bounds for concurrent self composition. In: TCC. pp. 203–222. (2004)
32. Lindell, Y.: Lower bounds and impossibility results for concurrent self composition. J. Cryptology 21(2), 200–249 (2008)
33. Micali, S., Pass, R., Rosen, A.: Input-indistinguishable computation. In: FOCS. pp. 367–378 (2006)
34. Pass, R.: Simulation in quasi-polynomial time, and its application to protocol composition. In: EUROCRYPT. pp. 160–176 (2003)
35. Pass, R.: Bounded-concurrent secure multi-party computation with a dishonest majority. In: STOC. pp. 232–241 (2004)
36. Pass, R., Rosen, A.: Bounded-concurrent secure two-party computation in a constant number of rounds. In: FOCS. pp. 404–413 (2003)
37. Prabhakaran, M., Rosen, A., Sahai, A.: Concurrent zero knowledge with logarithmic round-complexity. In: FOCS. pp. 366–375 (2002)
38. Prabhakaran, M., Sahai, A.: New notions of security: achieving universal composability without trusted setup. In: STOC. pp. 242–251 (2004)
39. Richardson, R., Kilian, J.: On the concurrent composition of zero-knowledge proofs. In: EUROCRYPT. pp. 415–431 (1999)
40. Yao, A.C.C.: How to generate and exchange secrets (extended abstract). In: FOCS. pp. 162–167 (1986)