Leakage-Resilient Circuits Revisited – Optimal Number of Computing Components without Leak-free Hardware

Dana Dachman-Soled* Feng-Hao Liu† Hong-Sheng Zhou‡

Abstract

Side channel attacks – attacks that exploit implementation-dependent information of a cryptosystem – have been shown to be highly detrimental, and the cryptographic community has recently focused on developing techniques for securing implementations against such attacks. An important model called *Only Computation Leaks* (OCL) [Micali and Reyzin, TCC '04] and its stronger variants were proposed to model a broad class of leakage attacks (a type of side-channel attack). These models allow for unbounded, arbitrary leakage as long as (1) information in each leakage observation is bounded, and (2) different parts of the computation leak independently. Various results and techniques have been developed for these models and we continue this line of research in the current work.

We address the problem of compiling any circuit into a circuit secure against OCL attacks. In order to leverage the OCL assumption, the resulting circuit will be split into components, where at any point in time only a single component is active. Optimally, we would like to output a circuit that has only one component, and no part of the computation needs to be leak-free. However, this task is impossible due to the result of Barak et al. [JACM '12]. The current state-of-the-art constructions achieve either two components with additional leak-free hardware, or many components without leak-free hardware.

In this work, we show how to achieve the best of both worlds: We construct two-component OCL schemes without relying on leak-free components. Our approach is general and modular – we develop generic techniques to remove the hardware component from hardware-based constructions, when the functionality provided by the hardware satisfies some properties. Our techniques use universal deniable encryption (recently constructed by Sahai and Water [STOC '14] using indistinguishable obfuscation) and non-committing encryption in a novel way. Then, we observe that the functionalities of the hardware used in previous two-component constructions of Juma and Vahlis [Crypto '10], and Dziembowski and Faust [TCC '12] satisfy the required properties.

The techniques developed in this paper have deep connections with adaptively secure and leakage tolerant multi-party computation (MPC). Our constructions immediately yield adaptively secure and leakage tolerant MPC protocols for any no-input randomized functionality in the semi-honest model. The result holds in the CRS model, without pre-processing. Our results also have implications to two-party leakage tolerant computation for arbitrary functionalities, which we obtain by combining our constructions with a recent result of Bitansky, Dachman-Soled, and Lin [Crypto '14].

1 Introduction

Side-channel attacks are attacks that exploit implementation-dependent information of a cryptosystem. Passive side-channel attacks, or *leakage attacks*, such as timing attacks, power analysis attacks, acoustic attacks, and more [1,6,24,30,36,37], have proven highly detrimental. Indeed, it has been shown that leakage attacks can be used to recover the entire secret key in common implementations of the RSA [25], AES [45] and DES [35] cryptosystems. These attacks are not just theoretical, and can be launched in complex, real-life settings, e.g. Boneh and Brumley [12] launched a practical network-based timing attack on SSL-enabled web servers.

*Dept. of Electrical and Computer Engineering , University of Maryland. Email: `danadach@ece.umd.edu`
†Dept. of Computer Science, University of Maryland. Email: `fenghao@cs.umd.edu`
‡Dept. of Computer Science, Virginia Commonwealth University. Email: `hszhou@vcu.edu`

In recent years, the cryptographic community has been devoted to developing adversarial models for side-channel attacks and constructing provably secure cryptosystems in these models. An important framework of this approach is to construct efficient *compilers* which take any circuit C (which may have a secret s hardcoded) and convert it to a new circuit $\tilde{C}$ that is secure against *leakage* attacks, where the adversarial model allows the attacker to adaptively choose inputs x and observe outputs $y = \tilde{C}(x)$ as well as adaptively obtaining leakage functions $\ell(\tilde{C})$ on the state of the circuit. Unfortunately, achieving security against adversaries who obtain even a *single bit* of arbitrary leakage is impossible since it implies *virtual black box* obfuscation, which was ruled out by the work of Barak et al. [4, 5][1].

Thus, the community turned to study reasonable ways to restrict the leakage function ℓ the adversary may leak on the internal state of the circuit. An important restricted class of leakage attacks was suggested by Micali and Reyzin [39], called the only computation leaks (OCL) model. In this model, throughout the computation, devices can have active states and inactive states, and at any point in time, information can only be leaked on the currently active state. This assumption is meant to capture a large class of attacks such as timing attacks, power analysis attacks and acoustic attacks, which only leak information on data this is currently being computed on.

Subsequently, various works have constructed so-called OCL compilers, which take any circuit C as input and convert it onto a new circuit $\tilde{C}$ such that $\tilde{C}$ is not only functionally equivalent, but also secure against OCL attacks. The way these compilers work is by splitting the computation into *components*, where during any time period only a single component is active. Specifically, consider an n-component circuit $\tilde{C} = \{\tilde{C}_1, \ldots, \tilde{C}_n\}$. At time period i where some component $\tilde{C}_j$ is active, the adversary may obtain some leakage $\ell_i(\tilde{C}_j)$ for some *bounded* output length function ℓ_i he chose. We say the scheme secure against *continual* OCL attacks if the adversary may run the circuit many times and obtain an unbounded total amount of leakage, as long as its leakage per time period is bounded. As noted previously, due to the impossibility result of Barak et al. [4], the minimal number of components required is two.

The first OCL compiler constructions were by Juma and Vahlis [34] and Goldwasser and Rothblum [28]. Both of these constructions require a secure hardware component to achieve security against OCL attacks; the construction of [34] requires only two components, and that of [28] requires many. Subsequently, Dziembowski and Faust [21] presented an alternative two-component OCL construction which achieves information-theoretic security, but also requires secure hardware. Although these works had argued that the hardware functionalities required are simple and independent of the circuit C,[2] it is still unsatisfactory if we need additional trusted assumptions on top of the existing one (the OCL assumption), especially when they are not necessary.

To date, the only known OCL compiler which does not require secure hardware is the information-theoretic construction of Goldwasser and Rothblum [29]. However, their OCL scheme requires a large number of OCL components[3]. Thus, the state-of-the art previous to this work was to either rely on secure hardware to achieve OCL with optimal number of components, or to achieve OCL without secure hardware, but with a large number of components. A major open question left along this line is:

Can we construct two-component OCL compilers without relying on secure hardware?

Beyond the fact that the question of optimal component OCL is of theoretical interest, OCL with minimal components has implications for the strength of the adversary we can tolerate, the hardware required by the OCL scheme, and for settings such as leakage-tolerant two-party computation. We discuss some of these implications below:

[1]This argument was explicitly stated in the work [29].

[2]This is to avoid the trivial solution that the hardware does all the computation of C.

[3]The original result of [29] requires $|C|$ components, where $|C|$ is the size of the original circuit. It was shown by [10] that the "ciphertext bank" of [29] can be combined with the construction of [21] to achieve an OCL compiler without secure hardware. The number of components required by this modification is a large constant, approximated by [10] as 20.

– **Strength of adversary.** Instead of characterizing a result as an OCL result, OCL results can be alternatively described as a class of leakage functions (out of all possible functions) that we provide security for. It is not hard to see that the fewer OCL components are, the larger the class of leakage functions we can handle.
– **Hardware.** Although OCL is an attractive assumption, it is not always clear whether the assumption holds universally under all environments. For example, the cold boot attacks by Halderman et al. [30] that showed memory can be leaky even if they are not active[4]. In order to implement an OCL scheme, we need an underlying hardware design that supports the OCL feature. The more components we need, the harder the design can be. Moreover, the overhead becomes larger with the number of components, so a large number of components, even though it is a constant such as twenty in the work [10], may be prohibitive.
– **Leakage tolerant computation.** As will be discussed in the sequel, two-component OCL without hardware has implications for leakage tolerant two-party computation. Loosely speaking, this is two-party secure computation where, in addition to corrupting parties, the adversary may ask for leakage on honest parties.

Model in this paper. In the literature, there are several strengthening OCL models, such as OCL^+ or a closely related model LDS (leaky distributed system) proposed by Bitansky et al. [7][5]. Unlike the OCL assumption where the adversary can only get leakage of components that follow a *particular* order (the order of activation), OCL^+ and LDS allow the adversary to leak component of an *arbitrary* order he likes. In these models, the adversary cannot leak on joint states of components, which is similar to the concept of split-state leakage (c.f. see the work of [31, 38] for further discussions). These models capture some memory attacks (such as some cold boot attacks [30]) beyond the traditional OCL model, as long as the leakage does not apply on joint states and are bounded per time period. Since the restrictions are weaker, it is easier to design hardware that achieves the requirements. Thus security guaranteed by these models is stronger.

It is not hard to see that the scheme JV [34] also achieves the notion of security under these models. It was observed [10] that the previous schemes GR [29] and DF [21] also achieve these stronger security notions. We remark that the above motivations for minimizing the number of components also hold for the OCL^+ and LDS settings. Throughout the whole paper, we consider the stronger OCL^+ model where the adversary can leak on any arbitrary order of the component. To avoid unnecessary complications, we will still call our model OCL, but the reader should keep in mind that the leakage can be obtained on any order of the components.

1.1 Our Results

In this work, we answer the question above affirmatively. We present two constructions of two-component OCL compilers from different assumptions. We take a modular approach with the following steps:

- First we establish a technique of how to get rid of hardware used in an OCL scheme – that is, given any secure hardware-based OCL scheme, suppose there exists a two-party protocol that realizes the functionality provided by the hardware with some strong property (defined later), then we can replace the hardware with the two-party protocol, resulting in a secure OCL scheme without hardware. The result can be summarized (informally) as Theorem 1.1.

- Then we consider how to construct a protocol that meets the requirements above. We show that under the existence of universal deniable encryption schemes (which can be constructed from indis-

[4]Some cold boot attacks can be captured by some strengthened OCL models as discussed later; yet we have the same motivation for reducing the number of components – the less the components, the more plausible the assumption can be.

[5]Bitansky et al. [7] showed that an LDS scheme is also an OCL^+ scheme; on the other hand, one can construct an LDS scheme from an OCL^+ scheme using non-committing encryption. Therefore, the two models are essentially equivalent.

tinguishable obfuscation by Sahai and Waters [44]) and non-committing encryption schemes [15], for any *simple* randomized functionality that takes no inputs, we can construct a protocol that achieves the goal (the strong property). The result can be summarized (informally) as Theorem 1.2.

- Finally, we look into the two currently known two-component hardware-based schemes with hardware, i.e. the JV scheme [34] and the DF scheme [21]. We observe that in both cases, the functionalities of the hardwares in both bases are "simple" in the sense that they can be expressed as no-input two-party randomized functionalities. Therefore, we can apply the theorems above to achieve two-component schemes that do not require secure hardware, by simply replacing the hardwares of JV or DF with the corresponding two-party protocols. We summarize the results as Corollary 1.3.

Our results are general and can be viewed as a design paradigm for OCL schemes: we can first construct a scheme that uses some simple hardware, which is presumably much easier to construct and to analyze. Then we can apply the generic tool to get rid of the hardware while preserving security. We state the two informal theorems below.

Theorem 1.1 (Hardware replacement theorem (Informal)) *Let $\Lambda^{\mathcal{F}}$ be some two-component OCL scheme with secure hardware implementing some two-party functionality $\mathcal{F}$. Assume there exists a two-party protocol ρ that realizes $\mathcal{F}$ (with some strong oblivious property), then there exists a two-component OCL scheme Λ' without hardware.*

Theorem 1.2 (Two-party protocol for simple hardware (Informal)) *Assume the existence of universal deniable encryption schemes and non-committing encryption schemes, then for any* no-input *two-party randomized functionality $\mathcal{F}$, there exists a two-party protocol ρ that realizes $\mathcal{F}$ with the strong oblivious property.*

By applying the theorems above to the hardware-based constructions of JV [34] and DF [21], we achieve the following corollary.

Corollary 1.3 *Assume the existence of universal deniable encryption schemes and non-committing encryption schemes. Then we achieve:*

- **(JV + Theorems 1.1, 1.2).** *If there further exists a fully homomorphic encryption (with cipher refreshing) that is secure against $2^{O(\ell(\lambda))}$ adversaries, the there exists a two-component OCL scheme that is $O(\ell)$ continual leakage resilient, where λ is the security parameter.*
- **(DF + Theorems 1.1, 1.2).** *There exists a two-component OCL scheme that is ℓ continual leakage resilient, for $\ell(\lambda) = m(\lambda)/10, m(\lambda) = \omega(\log(\lambda))$, where λ is the security parameter.*

Furthermore, both constructions do not require secure hardware.

We remark that our results rely on the existence of universal deniable encryption. This can be constructed from indistinguishable obfuscation for general circuits by Sahai and Waters [44]. Indistinguishable obfuscation for general circuits was constructed in the breakthrough result of Garg et al. [23] and followup work [2, 3, 11, 26, 42]; please refer to [46, 47] for more applications of indistinguishable obfuscation. Our constructions use universal deniable encryption in a black-box way, so they do not depend on a particular construction of universal deniable encryption nor indistinguishable obfuscation. Our results can be understood without the context of indistinguishable obfuscation, so we do not further discuss the notion to avoid digression.

1.2 Connections with Multi-party Computation

Our results have deep connections with multi-party computation (MPC) constructions that achieve different levels of security. In particular, we consider MPC for the following two classes of functionalities.

No-input randomized functionalities. The strong oblivions simulation property in Theorem 1.2 actually implies a stronger notion of adaptive security (against semi-honest corruption), called corruption-oblivious simulation by Bitansky et al. [8]. As shown in the work [8], such notion also implies leakage tolerance (against semi-honest corruptions). We will further discuss the strong oblivious property after Definition 2.3.

Thus, as an implication of the theorem, for any two-party no-input randomized functionalities, we are able to construct a two-party protocol (in the CRS model) that is simultaneously leakage tolerant, and adaptively secure (against semi-honest corruptions). In Section 5, we show how to generalize the construction to the setting of N-party no-input randomized functionalities.

Moreover, our protocols can implement randomized functionalities beyond "adaptively well-formed" ones according to Canetti et al. [17] – the functionalities do *not* need to leak its internal randomness to the adversary when all parties are corrupted. Additionally our protocols only need *two* rounds. To our knowledge, these are the *first* constructions that achieve adaptive security beyond the well-formed constraints; they are also the first constant-round protocols that are adaptively secure and leakage tolerant (against semi-honest corruptions) for this class of functionalities. We further elaborate on this in Remark 3.5.

General two-party functionalities. We observe that both the two-component constructions (JV-based and DF-based) are in fact of so-called *strong* OCL compilers (as introduced by Bitansky et al. [10]), where a strong OCL compiler is an OCL compiler with some enhanced simulation properties. Leveraging a recent result of Bitanksy et al. [10], which shows an *equivalence* between two-component strong OCL and two-party leakage tolerant computation in the input-independent preprocessing model (when no parties are corrupted), we obtain the following corollary:

Corollary 1.4 (Informal) *Assume the existence of universal deniable encryption schemes and non-committing encryption schemes, Then for every function f, there exists a two-party leakage tolerant protocol which UC-emulates f in the input-indepdendent preprocessing model when no parties are corrupted.*

Very recently, it was shown that based on standard cryptographic assumptions, the equivalence between two-component strong OCL and two-party leakage tolerant computation in the input-independent preprocessing model can be extended to the case where one or both parties are actively corrupted [9]. Combining this result with our two-component strong OCL constructions, we then obtain, for every function f, a two-party leakage tolerant protocol which UC-emulates f in the input-independent preprocessing model under static, active corruption of parties.

1.3 Techniques

In this section, we highlight some of our techniques to achieve our two main theorems.

Hardware replacement theorem. Let $\Lambda^{\mathcal{F}}$ be some secure hardware-based two-component OCL scheme where the hardware implements some functionality $\mathcal{F}$. Similar to the spirit of the Universal Composability framework [13, 14], our goal is to replace the hardware by a two-party protocol ρ while preserving OCL security.[6] Clearly the theorem cannot work with any arbitrary two-party protocol – we argue that the protocol ρ at least needs to be somewhat leakage resilient to the OCL leakage (independent leakage on each party), since the replacement theorem should also work for the trivial case where $\mathcal{F}$ is a secure hardware that computes the circuit we want to protect.

[6]We do not use the term of "composability" to avoid confusion since OCL schemes, through related, are different from protocols in syntax and many other properties.

In this work, we identified a strong oblivious simulation property that captures the spirit of the corruption-oblivious simulation defined by Bitansky et al. [8] in a compact and simple way. Then we show suppose ρ realizes $\mathcal{F}$ with such strong oblivious simulation, then we can replace the hardware by the protocol ρ while preserving the security. We note that since the syntaxes of OCL compilers and leakage tolerant protocols are quite different, it is not clear whether the hardware replacement theorem can be implied by the composition theorem by Bitansky et al. [8].

Informally, the strong oblivious simulation requires that there exist independent simulators $(\mathcal{S}_{\text{tr}}, \mathcal{S}_1, \mathcal{S}_2)$ such that the simulator $\mathcal{S}_{\text{tr}}$ can generate an indistinguishable transcript τ, and for $b = \{1, 2\}$, $\mathcal{S}_b(x_b, y_b, \tau)$ can generate an indistinguishable view (or state) of the party P_b, where x_1, x_2, y_1, y_2 are the inputs/outputs to the ideal functionality, i.e. $(y_1, y_2) \leftarrow \mathcal{F}(x_1, x_2)$. This means, any leakage function g on P_b's state can be simulated by $g(\mathcal{S}_b(x_b, y_b, \tau))$. Therefore, any leakage attack at the state of party P_b (the real world) can be translated to a leakage attack at the input/output of the party P_b in the ideal world. Using the idea, we can further show that any OCL leakage attack at the scheme Λ^ρ (a scheme where we replace the hardware functionality by ρ) can be translated to an OCL leakage attack at the scheme $\Lambda^{\mathcal{F}}$. Thus, the security of Λ^ρ is guaranteed by the security of $\Lambda^{\mathcal{F}}$.

Constructing protocols for simple functionalities. The next part of our main contribution is to construct protocols that achieve the strong oblivious simulation property. We note that this property is very strong that we do not know how to construct protocols for general functionalities. However, for a restricted but still very useful class of functionalities – no-input randomized functionalities, we show how to construct protocols that achieve the strong obvious simulation property, using deniable encryptions (recently constructed by Sahai and Waters [44] with indistinguishable obfuscation). Then we observe that the "simple" hardwares used in the literature [21, 34] can be captured by such class.

We use a (universal) deniable encryption and a receiver non-committing encryption as our building blocks. Informally, a deniable encryption allows a sender to come up with a message and randomness that explain a ciphertext. That is, given any ciphertext c^* and a message m, the sender can come up with (indistinguishable) randomness r such that $\mathsf{Enc}(m; r) = c^*$; a receiver non-committing encryption allows a simulator to first generate a pair of simulated public-key and ciphertext (without knowing what the underlying message was), and later to come up with consistent random coins that explain the key generation, and decrypt the ciphertext to an arbitrary message m. By combining the two in a novel way, we show how to design protocols that achieve the strong oblivious simulation for no-input randomized functionalities. We give further overviews in Section 3.

1.4 Related Work

In this section, we compare our two-component OCL compilers with previous results from the literature. OCL compilers which require secure hardware were constructed by [21, 28, 34]. These OCL compilers all require two components; the compiler of [21] is information theoretic; the compiler of [28] relies on the DDH assumption and the compiler of [34] requires fully homomorphic encryption with a ciphertext refreshing property. An OCL compiler which does not require secure hardware was first constructed by [29]; moreover, theire construction is information theoretic. The compiler of [29] is described as requiring $O(|C|)$ components, where $|C|$ is the size of the underlying circuit. However, it was shown in [10] how to combine techniques from [29] and [21] to achieve an OCL compiler without secure hardware and a large constant number of components, approximated by [10] as 20.

In the following table, we present a comparison of the assumptions, number of components and leakage rates achieved by best known previous work [10, 21, 29, 34], as well our JV-based and DF-based schemes. Let ℓ be some parameter. The following table presents parameters for different schemes in order to construct an OCL compiler that tolerates ℓ-bit leakage per time.

We remark that even though the leakage rate of the previous constant-component constructions depends on the circuit size $|C|$, there is a generic way to get rid of the dependency by using FHE. We can first encrypt

Scheme	Hardware	Assumption	Components	Leakage rate (Asymptotically)
JV	Yes	2^ℓ-secure FHE	2	$\ell/w\|C\|$
DF	Yes	None	2	$1/\ell\|C\|$
GR12	No	None	$O(\|C\|)$	$1/\ell^{O(1)}$
BDL14	No	None	20	$1/\ell^{O(1)} \cdot \|C\|$
Ours (JV-based)	No	2^ℓ-secure FHE, Deniable Enc, NCE	2	$\ell/w\|C\|$
Ours (DF-based)	No	Deniable Enc, NCE	2	$1/\ell\|C\|$
Ours (DF + FHE based)	No	Deniable Enc, NCE + FHE	2	$1/\ell^{1+o(1)}$

Table 1: Comparison of various OCL schemes in the literature.

w corresponds to the length of the FHE ciphertext, $|C|$ corresponds to the size of the underlying circuit. NCE stands for non-committing encryption. We note that the ciphertext length w must be at least as large as ℓ; otherwise it is easy to break the FHE scheme in time 2^ℓ. The constant of $O(1)$ depends on the best algorithm of matrix multiplication. Both constants are greater than 1.37 under the best known algorithm by Williams [48].

the circuit C (keep it public) by the FHE, and then apply the OCL compiler only on the decryption circuit, which has size λ (can be re-parameterized as the security parameter) and is independent of the circuit C. This idea is generic and can be applied to all OCL constructions. We only state one example in the last row of the table, but note that the rates for other constructions can be also improved in this way.

Alternative *leakage on computation* models and compilers in these models. Ishai et al. [32] suggested a leakage model which captures wire probing attacks where the adversary may leak the value of individual wires during the computation. Note that the OCL model subsumes this model. Additional models for leakage on computation were introduced by [18], and [22] . These models allow unbounded-length "noisy" leakage (leakage that does not reduce the entropy of the circuit's secret state by too much) and leakage under restricted classes of leakage functions (such as $\mathcal{AC}^0$ leakage), respectively. Compilers for the wire probing model were constructed by [32]; compilers for the noisy leakage model were constructed by [20, 22]; compilers for restricted classes of leakage functions were constructed by [22,40,41,43].

2 Two-Component OCL Schemes and Hardware Replacement Theorem

A two-component OCL scheme for a (private) circuit $C(\cdot)$, consists of an efficient compiler Comp and a two-party protocol $\Pi = (P_1, P_2)$. To compute $C(\cdot)$ in a leakage-resilient way, the circuit is *compiled* ahead of time by $\mathsf{Comp}(C(\cdot))$ that produces a public parameter pp, and initial states $(\mathsf{intl}_1, \mathsf{intl}_2)$ for each party. This compilation is done "in the dark" without any leakage. Afterwards, the public parameter pp will be given to the two parties and the adversary (at all time), and then, the parties can compute together $y = C(x)$ for any input x by running the protocol Π for an arbitrary polynomial number of inputs.

Below we provide the formal definition and security requirements of OCL schemes. Here the adversaries are allowed to continually leak on the internal state during each iteration. As discussed in the introduction, here we consider a stronger adversary that he can leak on *any* arbitrary order of the components. Additionally, we consider a further stronger security notion where we require the simulator to be oblivious of the leakage queries from the adversaries.

Definition 2.1 (Two-component OCL schemes) *We say that $\Lambda = (\mathsf{Comp}, \Pi = \langle P_1, P_2\rangle)$ is a continual, two-component OCL scheme if it satisfies the following properties.*

Initialization: *For every security parameter $\lambda \in \mathbb{N}$, polynomial-sized circuit family $\mathcal{C} = \{C_\lambda\}_{\lambda\in\mathbb{N}}$, the compiler $\mathsf{Comp}(1^\lambda, C_\lambda)$ runs in time $\mathrm{poly}(\lambda)$ and outputs a public parameter pp and 2 initial states $\mathsf{intl}_1, \mathsf{intl}_2$. Note that pp will be kept the same during all evaluations, and given to all parties.*

Unbounded-time evaluation: *The evaluation procedure invokes the protocol Π between the components $P_1(\mathsf{pp}, \mathsf{intl}_1)$, $P_2(\mathsf{pp}, \mathsf{intl}_2)$, which interact in an* arbitrary *polynomial number of iterations: In the i^{th} iteration, P_1 receives an input $x_i \in \{0,1\}^{|C_\lambda|}$ and P_2 produces an output y_i[7]. At the end of the evaluation, an update procedure is carried out, producing the new initial states for the next iteration; then all information other than the new initial states are erased. Note that pp will not be erased and will be reused in the next iteration.*

For each component $b \in \{1,2\}$, denote by $\mathsf{intl}_{i,b}$ the initial states of component b at the onset of the i^{th} iteration (in the first iteration, $\mathsf{intl}_{1,b} = \mathsf{intl}_b$), and $\mathsf{evl}_{i,b}$ the random coins tossed and messages exchanged by each P_b during the i^{th} iteration, including its state during the update phase.

Correctness with adaptive input selection: *For every $\lambda \in \mathbb{N}$, polynomial-sized circuit family $\mathcal{C} = \{C_\lambda\}_{\lambda\in\mathbb{N}}$, auxiliary input $z \in \{0,1\}^{\mathrm{poly}(\lambda)}$, and PPT adversary $\mathcal{A}$, in the following real experiment $\mathbf{Real}_{\mathcal{A}}(1^\lambda, C_\lambda, z)$ where $\mathcal{A}$ initiates an arbitrary number of evaluations with adaptively chosen inputs, it holds that with all but negligible probability, the outputs of all evaluations are correct.*

We say that an OCL scheme has perfect correctness if the above holds with probability 1.

2.1 Security Model

We now describe the security experiments of OCL schemes. A scheme Λ is said to be ℓ-leakage-resilient with oblivious simulation if there is a simulator $\mathcal{S}$, such that, for every $\lambda \in \mathbb{N}$, polynomial-sized family $\mathcal{C}$, and auxiliary input $z \in \{0,1\}^{\mathrm{poly}(\lambda)}$, the views of the adversary in the following real and ideal experiments are indistinguishable. In the real world, the adversary has the power of obtaining leakage independently from each component in honest OCL evaluations over inputs chosen adaptively by the adversary, whereas in the ideal world, it obtains leakage from states of the components simulated by an oblivious simulator, given oracle access to the circuit $C_\lambda(\cdot)$. More formally,

Experiment $\mathbf{Real}_{\mathcal{A}}(1^\lambda, C_\lambda, z)$: The adversary $\mathcal{A}(1^\lambda, |C_\lambda|, z)$ proceeds as follows:

1. The initial states $(\mathsf{pp}, \mathsf{intl}_1, \mathsf{intl}_2) \leftarrow \mathsf{Comp}(1^\lambda, C_\lambda)$ are sampled.
2. $\mathcal{A}$ gets the public parameter pp and launches ℓ-bounded leakage attacks on an unbounded polynomial number of evaluations of its choice. In the i^{th} iteration, $\mathcal{A}$ works as follows:
   (a) $\mathcal{A}$ submits an input $x_i \in \{0,1\}^{|C_\lambda|}$, which is evaluated on C_λ by resuming the protocol execution of Π between the components $P_1(\mathsf{pp}, \mathsf{intl}_{i,1}), P_2(\mathsf{pp}, \mathsf{intl}_{i,2})$ with input x_i to the first component P_1.
   (b) $\mathcal{A}$ launches an ℓ-bounded leakage attack on the i^{th} evaluation. It issues leakage queries $(G_{1,b_1}, G_{2,b_2}, \ldots,)$, where each $b_k \in \{1,2\}$ to the two components (adaptively), and obtain leakage answers of all queries, i.e., $G_{k,b_k}(\mathsf{intl}_{i,b_k}, \mathsf{evl}_{i,b_k})$, as long as the total amount of leakage on each component in this iteration is smaller than $\ell(\lambda)$ bits. Denote $L_i \in \{0,1\}^{\le \ell}$ be the leakage observed in the i^{th} round.
   (c) $\mathcal{A}$ obtains the output of the evaluation, which is the output of P_2.

[7]It is without loss of generality that P_1 receives inputs and P_2 produces an output. We can always achieve this by sending one more round of message.

Denote $\text{VIEW}^{\ell}_{\mathcal{A}}(1^{\lambda}, C_{\lambda}, z) = (\mathsf{pp}, x_1, y_1, L_1, x_2, y_2, L_2, \dots,)$ as the view of $\mathcal{A}$ in the above experiment.

Experiment $\mathbf{Ideal}_{\mathcal{S},\mathcal{A}}(1^{\lambda}, C_{\lambda}, z)$: In the ideal experiment, the simulator $\mathcal{S}^{\mathcal{A}(1^{\lambda},|C_{\lambda}|,z),C_{\lambda}(\cdot)}$ gets oracle access to the adversary $\mathcal{A}$ and oracle access to the circuit $C_{\lambda}(\cdot)$. His task is to produce an indistinguishable view of the adversary.

Furthermore, we say the simulator is oblivious, if it uses the following strategy to interact with the adversary: let the adversary $\mathcal{A}(1^{\lambda}, |C_{\lambda}|, z)$ participate in the same experiment as above. The simulator at the beginning generates a public parameter $\widetilde{\mathsf{pp}}$ and gives it to the adversary $\mathcal{A}$. Then at each round i, the simulator works as follows.

(a) Let x_i be the input $\mathcal{A}$ submits in this iteration, and $y_i = C_{\lambda}(x_i)$ be the answer obtained by the oracle query. $\mathcal{S}(1^{\lambda}, i, x_i, y_i;\ \mathbf{w}_i)$ is invoked, producing simulated states $(\widetilde{\mathsf{intl}}_{i,1}, \widetilde{\mathsf{intl}}_{i,2}, \widetilde{\mathsf{evl}}_{i,1}, \widetilde{\mathsf{evl}}_{i,2})$, where w_i is the fresh random coins tossed for the simulation in iteration i and $\mathbf{w}_i = w_1, \cdots, w_i$ is all the random coins that have been tossed for simulation in the first i iterations.

(b) Let $(G_{1,b_1}, G_{2,b_2}, \dots,)$ where $b_k \in \{1, 2\}$, be the leakage queries $\mathcal{A}$ makes (perhaps in an adaptive way) in this round. Then $\mathcal{S}$ returns $G_{k,b_k}(\widetilde{\mathsf{intl}}_{i,b_k}, \widetilde{\mathsf{evl}}_{i,b_k})$ for all these queries, as long as the total amount of leakage on each component in this iteration is smaller than $\ell(\lambda)$ bits.

(c) $\mathcal{S}$ sends y_i to the adversary.

Denote $\widetilde{\text{VIEW}}^{\ell}_{\mathcal{S},\mathcal{A}}(1^{\lambda}, C_{\lambda}, z)$ as the (simulated) view of $\mathcal{A}$ in the above experiment.

Definition 2.2 (Continual ℓ-leakage-resilience with oblivious simulation) *We say that a continual OCL scheme* OCL *is continually ℓ-leakage-resilient with oblivious simulation if there is a* PPT *simulator $\mathcal{S}$, such that, for every* PPT *adversary $\mathcal{A}$, every polynomial-sized circuit family $\mathcal{C}$, the following two ensembles are indistinguishable.*

– $\{\text{VIEW}^{\ell}_{\mathcal{A}}(1^{\lambda}, C_{\lambda}, z)\}_{\lambda\in\mathbb{N}, C_{\lambda}\in\mathcal{C}, z\in\{0,1\}^{\text{poly}(\lambda)}}$

– $\{\widetilde{\text{VIEW}}^{\ell}_{\mathcal{S},\mathcal{A}}(1^{\lambda}, C_{\lambda}, z)\}_{\lambda\in\mathbb{N}, C_{\lambda}\in\mathcal{C}, z\in\{0,1\}^{\text{poly}(\lambda)}}$

$\mathcal{F}$-hybrid OCL schemes. A two-component OCL scheme may use subroutines during its execution. Let $\mathcal{F}$ denote a two-party functionality. We say a two-component OCL scheme $\Lambda = (\mathsf{Comp}, \Pi = \langle P_1, P_2 \rangle)$ is an $\mathcal{F}$-hybrid OCL scheme if Λ completes its execution by calling $\mathcal{F}$ (probably multiple times). Often we write it as $\Lambda^{\mathcal{F}} = (\mathsf{Comp}, \Pi^{\mathcal{F}})$. If the OCL scheme Λ calls $\mathcal{F}$ at a round i, and let (x_1, x_2) be the values provided by P_1, P_2 and (y_1, y_2) be the values $\mathcal{F}$ returns to P_1, P_2 respectively, then the states $\mathsf{evl}_{i,1}, \mathsf{evl}_{i,2}$ will include (x_1, y_1) and (x_2, y_2), respectively. The adversary can obtain leakage of (x_1, y_1), (x_2, y_2) (perhaps adaptively but not jointly) via (adaptive) leakage queries.

Usually, we can think of $\mathcal{F}$ as some hardware that generates messages securely, i.e. there is no leakage on the internal states. We next consider how to *replace* such $\mathcal{F}$ with a two-party protocol. Intuitively, suppose there is a two-party protocol ρ that "realizes" $\mathcal{F}$, then we have a two-component OCL scheme where we can replace the calls to $\mathcal{F}$ by running the protocol ρ. We denote the scheme as $\Lambda' = (\mathsf{Comp}, \Pi^{\rho})$. We also consider the case where ρ realizes $\mathcal{F}$ in a setting that a common reference string (CRS) crs is always available. In this case, we can combine the CRS generation into the compilation: Comp may generate a certain pubic parameter pp, and we can simply augment Comp into Comp' that generates $\mathsf{pp}' = \mathsf{pp}||\mathsf{crs}$. This is denoted as $\Lambda' = (\mathsf{Comp}', \Pi^{\rho})$.

However, standard simulation based security is not sufficient for the argument of the hardware replacement as above because the simulation (of ρ) could be a *joint* simulation for both participants. This is inconsistent with the security requirement of OCL scheme where the emulation for one component is oblivious to the emulation of the other component. In the following, we define a stronger version of realization, and prove that if ρ realizes $\mathcal{F}$ in this sense, then we can replace $\mathcal{F}$ with ρ and the OCL scheme remains secure. The definition we present is a compact and simplified version that captures the notion of "security under adaptive corruptions with a corruption-oblivious simulator" defined by [8]. We consider the semi-honest case only.

Definition 2.3 (Strong oblivious simulation for protocols) *Let* $\mathsf{crs} \leftarrow \mathsf{CRS.Gen}(1^\lambda)$*, and let* $\pi = (P_1, P_2)$ *be a two-party protocol using such common reference string* crs*. Let* $\mathcal{F}$ *be a two-input (perhaps randomized) ideal functionality. We say* π *realizes the functionality* $\mathcal{F}$ *with strong oblivious simulation, if there exists a* PPT *simulator* $\mathcal{S} = (\mathcal{S}_1, \mathcal{S}_2, \mathcal{S}_{\mathrm{tr}})$ *for all (non-uniform)* PPT *adversary* $\mathcal{A}$ *such that the following distributions are computationally indistinguishable:*

$$\left\{ \begin{array}{r} \mathsf{crs} \leftarrow \mathsf{CRS.Gen}(1^\lambda); (x_1, x_2) \leftarrow \mathcal{A}(\mathsf{crs}); \\ (r_1, r_2) \leftarrow U_\lambda \times U_\lambda; \\ \tau = \langle P_1(\mathsf{crs}, x_1; r_1), P_2(\mathsf{crs}, x_2; r_2)\rangle \\ : (\mathsf{crs}, x_1, r_1, \tau, x_2, r_2) \end{array} \right\} \approx \left\{ \begin{array}{r} \mathsf{crs} \leftarrow \mathsf{CRS.Gen}(1^\lambda); (x_1, x_2) \leftarrow \mathcal{A}(\mathsf{crs}); \\ (y_1, y_2) \leftarrow \mathcal{F}(x_1, x_2); \tilde{\tau} \leftarrow \mathcal{S}_{\mathrm{tr}}(\mathsf{crs}); \\ \tilde{r}_1 \leftarrow \mathcal{S}_1(\mathsf{crs}, x_1, y_1, \tilde{\tau}); \tilde{r}_2 \leftarrow \mathcal{S}_2(\mathsf{crs}, x_2, y_2, \tilde{\tau}) \\ : (\mathsf{crs}, x_1, \tilde{r}_1, \tilde{\tau}, x_2, \tilde{r}_2) \end{array} \right\}.$$

Note that r_1*,* r_2 *are the random coins of the parties, and* τ *is the transcript (i.e. message exchanges) by running the protocol* $\langle P_1(\mathsf{crs}, x_1; r_1), P_2(\mathsf{crs}, x_2; r_2)\rangle$*.*

Remark 2.4 *The notion above is related to the notion of security under adaptive corruptions with a corruption-oblivious simulator defined by Bitansky et al. [8]. We elaborate further below.*

- *Our notion implies adaptive security where the simulator uses a universal strategy that is independent of the order of corruption, i.e.* $\mathcal{S}_1$ *and* $\mathcal{S}_2$ *simulate independently (with a joint transcript* $\tilde{\tau}$ *that is independent of the inputs/outputs). In contrast, in the adaptive security case, the simulator is allowed to see the already corrupted party's input/output, i.e. if the adversary first corrupts* P_1 *and then* P_2*, then the simulator can see both* (x_1, y_1) *and* (x_2, y_2) *when simulating the view of* P_2*. Known constructions of adaptively secure two party computation for all functionalities, such as [17], do not admit such simulators.*
- *Following the approach of [8], our notion implies a strong notion of leakage tolerant two-party computation in the semi-honest setting where any* t*-bit leakage function in the real world can be translated to a* t*-bit leakage function in the ideal world. There is no prior bound on* t*. Moreover, if the leakage function in the real world does not leak on the joint states, then the ideal leakage function does not, either.*

Theorem 2.5 (OCL hardware replacement theorem) *Let* $\mathcal{F}$ *be some two-party functionality, and* $\Lambda^{\mathcal{F}} = (\mathsf{Comp}, \Pi^{\mathcal{F}})$ *be an* $\mathcal{F}$*-hybrid two-component OCL scheme that is* ℓ*-leakage-resilient with oblivious simulation. Suppose there exists a two-party protocol* ρ *using a common reference string* crs*, that realizes* $\mathcal{F}$ *with strong oblivious simulation as Definition 2.3. Then there exists a two-component OCL scheme* $\Lambda' = (\mathsf{Comp}', \Pi^\rho)$ *that is* ℓ*-leakage-resilient with oblivious simulation.*

The intuition of the proof of Theorem 2.5 is the following: The two-component OCL scheme Λ' will simply replace each call to the ideal functionality $\mathcal{F}$ with an execution of the two-party protocol ρ, where each component plays the part of one of the parties in ρ. To obtain an oblivious simulator for the composed execution Λ', we must reconstruct the entire state of each component. Note that the state of each component in Λ' simply consists of its state in Λ, concatenated with its state in ρ. Thus, a natural approach is to reconstruct each party's state by concatenating the output of the simulator $\widehat{\mathcal{S}}$ for Λ and the output of the simulator $\widetilde{\mathcal{S}}$ for ρ. Indeed, this approach does in fact work since the only shared information between the component's simulated state in Λ and its simulated state in ρ is the input/output of the ideal functionality $\mathcal{F}$. Therefore, conditioned on the input/output of $\mathcal{F}$, each component's state in ρ can be reconstructed entirely independently of its state in Λ. We note that the bound in Λ' inherits from the underlying scheme Λ. By the security of the protocol ρ as discussed above, any leakage to the evaluation states of ρ can be translated to leakage of the input/output of $\mathcal{F}$. For this part we do not need a prior bound. We defer the formal proof to Section A.2.

Remark 2.6 *Very recently, Bitansky et al. [10] defined a notion of* strong *two-component OCL schemes. They showed this notion has implications to two-party leakage tolerant protocols as discussed in the introduction. We note that the previous schemes [21, 34] satisfy this stronger notion, and a similar hardware replacement theorem can be achieved. Please refer to Appendix A.1 for more details.*

3 How to Implement Simple Functionalities

In this section, we construct two-party protocols to realize functionalities that provided by hardware components in previous schemes (with strong oblivious simulation). However, the requirement is very strong as we discussed in the previous section, and it is not clear how to construct protocols for general functionalities. Fortunately, the functionalities used in the previous constructions by Juma and Vahlis, and by Dziembowski and Faust [21, 34] (and in all the other known constructions) are "simple". We show how to construct protocols for certain simple functionalities.

3.1 Ideal Functionality

Here we define a functionality $\mathcal{F}^{\Delta}_{\text{sampling}}$ which samples correlated randomness (according to some distributions) for both parties *without* taking any inputs. This functionality captures the hardwares used in the previous schemes of Juma-Vahlis and Dziembowski-Faust.

The ideal functionality $\mathcal{F}^{\Delta}_{\text{sampling}}$ is parametrized by an efficiently samplable distribution Δ that outputs correlated random coins $(\gamma_1, \gamma_2) \leftarrow \Delta(1^\lambda)$.

Functionality $\mathcal{F}^{\Delta}_{\text{sampling}}$

$\mathcal{F}^{\Delta}_{\text{sampling}}$, parameterized with a distribution Δ, a variable done with initial value 0, running with parties (P_1, P_2) and an adversary $\mathcal{A}$, operates as follows:

- Upon receiving request from P_i, if $\mathsf{done} = 0$, then sample $(\gamma_1, \gamma_2) \leftarrow \Delta(1^\lambda)$, and set $\mathsf{done} := 1$. Return γ_i to party P_i, and ignore future request from P_i.

Figure 1: The ideal functionality for sampling correlated randomness

3.2 Building Blocks

In this section, we present two building blocks for our construction – receiver non-committing encryption, and universal deniable encryption transformation. Basically, a receiver non-committing public key encryption allows a simulator to first generate a pair of simulated public-key and ciphertext (without knowing what the underlying message was), and later to come up with consistent random coins that explain the key generation, and decrypt the ciphertext to an arbitrary message m. This notion is weaker than standard non-committing public key encryption [15], which can be constructed from *trapdoor simulatable public key encryption* [19], which in turn can be instantiated under standard assumptions such as CDH, RSA, DDH, LWE and factoring Blum integers.

Universal Deniable Encryption is a new notion proposed by Sahai and Waters [44]. Here we paraphrase the ideas as Definition 3.2: given any encryption scheme $\mathsf{E} = \{\mathsf{Gen}, \mathsf{Enc}, \mathsf{Dec}\}$, there is a one-time setup UniGen that takes input the encryption algorithm $\mathsf{E.Enc}$ and generates two programs $C_{\mathsf{encrypt}}, C_{\mathsf{explain}}$, an encryption program and an explanation program. The one-time setup is generated by some trusted party.

Basically, the encryption program C_{encrypt} takes inputs a public key pk and a message m, outputs a ciphertext. The explanation program C_{explain} takes inputs a public key pk, a ciphertext c, and a message m outputs random coins r to "explain" that c is an encryption of m. That is, running the encryption program with input pk, m and randomness r, it will output c, i.e. $c = C_{\mathsf{encrypt}}(\mathsf{pk}, m; r)$. The security requires that (1) the distribution of $\{C_{\mathsf{encrypt}}(\mathsf{pk}, m)\}$ is statistically close to that of $\{\mathsf{E.Enc}(\mathsf{pk}, m)\}$. In other words, using C_{encrypt} is essentially the same as using the encryption algorithm. (2) It is computationally hard to distinguish the real random coins from the explained random coins (by C_{explain}). Note that since $C_{\mathsf{encrypt}}, C_{\mathsf{explain}}$ can be generated without knowing any secret information, the semantic security of E preserves even given these two programs.

Actually, the universal deniable encryption in the work of Sahai and Waters [44] is more general: it allows $C_{\mathsf{encrypt}}, C_{\mathsf{explain}}$ to take public keys from different encryption schemes. In our application, this slightly restricted version already suffices. So for clarity of exposition, we present this simpler version.

Definition 3.1 (Receiver non-committing encryption [16,33]) *A one-sided non-committing encryption scheme (for the receiver) consists of a tuple* $(\mathsf{NCGen}, \mathsf{NCEnc}, \mathsf{NCDec}, \mathsf{NCSim})$ *such that* $(\mathsf{NCGen}, \mathsf{NCEnc}, \mathsf{NCDec})$ *is an encryption scheme and* $\mathsf{NCSim} = (\mathsf{NCSim}_1, \mathsf{NCSim}_2)$ *is a tuple of two simulation algorithms. On input* 1^λ*,* $\mathsf{NCSim}_1(1^\lambda)$ *outputs a simulated public key* $\widetilde{\mathsf{pk}}$ *and a simulated ciphertext* $\widetilde{c}$*; on inputs a simulated public key,* $\widetilde{\mathsf{pk}}$*, a simulated ciphertext* $\widetilde{c}$*, and a message* m*,* $\mathsf{NCSim}_2(\widetilde{\mathsf{pk}}, \widetilde{c}, m)$ *outputs random coins* $\widetilde{\sigma}$ *(for the key generation,* NCGen*). We say the scheme is secure if for all messages* m*, the following two distributions are indistinguishable:*

– *the view of honest decryptor in a normal encryption of* m*:*

$$\{(\mathsf{pk}, c, \sigma) : (\mathsf{pk}, \mathsf{sk}) \leftarrow \mathsf{NCGen}(\sigma), c \leftarrow \mathsf{NCEnc}(\mathsf{pk}, m)\},$$

– *simulated view of an encryption of* m*:*

$$\left\{(\widetilde{\mathsf{pk}}, \widetilde{c}, \widetilde{\sigma}) : (\widetilde{\mathsf{pk}}, \widetilde{c}) \leftarrow \mathsf{NCSim}_1(1^\lambda), \widetilde{\sigma} \leftarrow \mathsf{NCSim}_2(\widetilde{\mathsf{pk}}, \widetilde{c}, m)\right\}.$$

Definition 3.2 (Universal deniable encryption transformation for an encryption scheme) *Let* $\mathsf{E} = \{\mathsf{Gen}, \mathsf{Enc}, \mathsf{Dec}\}$ *be a (bit) encryption scheme. A universal deniable encryption transformation for* E *is a* PPT *algorithm* UniGen *that takes input security parameter* 1^λ*, an encryption circuit that implements the encryption algorithm* $\mathsf{E.Enc}(1^\lambda, \cdot; \cdot)$ *and outputs two programs* $C_{\mathsf{encrypt}}, C_{\mathsf{explain}}$ *with the following syntax: let* pk *be a public key,* m *be a message,* c *be a ciphertext.*

- C_{encrypt} *takes inputs* pk, m*, random coins* r*, and* $C_{\mathsf{encrypt}}(\mathsf{pk}, m; r)$ *outputs a ciphertext* c*;*
- C_{explain} *takes inputs* pk, c, m*, random coins* $\bar{v}$*, and* $C_{\mathsf{explain}}(\mathsf{pk}, c, m; \bar{v})$ *outputs a string* r*.*

We say the transformation is secure if:

(a) For all $\mathsf{pk} \in \mathsf{E.Gen}(1^\lambda)$*, messages* $m \in \{0,1\}$*, and any* $C_{\mathsf{encrypt}} \in \mathsf{UniGen}(1^\lambda)$*, the following two distributions are statistically close:* $\{C_{\mathsf{encrypt}}(\mathsf{pk}, m)\} \approx \{\mathsf{E.Enc}(\mathsf{pk}, m)\}$*. Note that the circuit* C_{encrypt} *and the encryption algorithm* $\mathsf{E.Enc}$ *might have different spaces for random coins, but the distributions can still be statistically close.*

(b) For any message $m \in \{0,1\}$*, the following two distributions are computationally indistinguishable:*

$$\{(C_{\mathsf{encrypt}}, C_{\mathsf{explain}}, \mathsf{pk}, c, r)\} \approx \{(C_{\mathsf{encrypt}}, C_{\mathsf{explain}}, \mathsf{pk}, c, r')\},$$

where $(C_{\mathsf{encrypt}}, C_{\mathsf{explain}}) \leftarrow \mathsf{UniGen}(1^\lambda)$*,* $\mathsf{pk} \leftarrow \mathsf{E.Gen}(1^\lambda)$*,* $r \leftarrow U_{\mathrm{poly}(\lambda)}$*,* $c = C_{\mathsf{encrypt}}(\mathsf{pk}, m; r)$*,* $r' \leftarrow C_{\mathsf{explain}}(\mathsf{pk}, c, m)$*, and* $U_{\mathrm{poly}(\lambda)}$ *denotes the uniform distribution over a polynomial number of bits.*

Theorem 3.3 ([44]) *Assume there exist indistinguishable obfuscation for general circuits and one way functions. Then there exists a secure universal deniable encryption transformation for any encryption scheme.*

As pointed out in the introduction, our constructions only use universal deniable encryption and non-committing encryption in a black-box way. We do not explicitly use indistinguishable obfuscation so we do not present the syntax here.

3.3 Our Construction

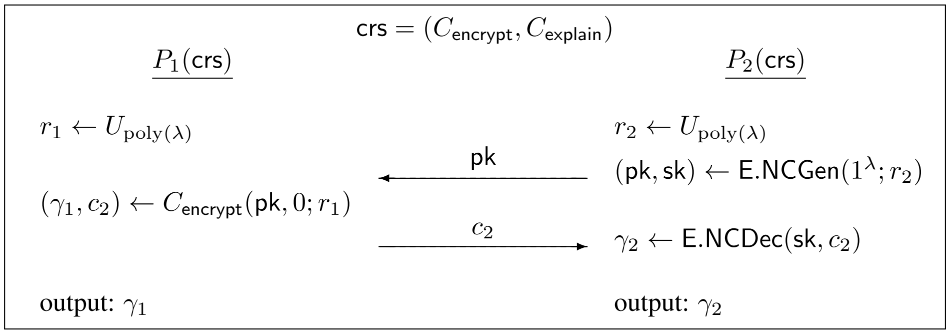
$\mathsf{crs} = (C_{\mathsf{encrypt}}, C_{\mathsf{explain}})$
$\underline{P_1(\mathsf{crs})}$
$\underline{P_2(\mathsf{crs})}$
$r_1 \leftarrow U_{\text{poly}(\lambda)}$
$r_2 \leftarrow U_{\text{poly}(\lambda)}$
pk
$(\mathsf{pk}, \mathsf{sk}) \leftarrow \mathsf{E.NCGen}(1^\lambda; r_2)$
$(\gamma_1, c_2) \leftarrow C_{\mathsf{encrypt}}(\mathsf{pk}, 0; r_1)$
c_2
$\gamma_2 \leftarrow \mathsf{E.NCDec}(\mathsf{sk}, c_2)$
output: γ_1
output: γ_2

Figure 2: A protocol for $\mathcal{F}_{\text{sampling}}$.

Now we are ready to describe our protocol. Let $\mathsf{E} = \{\mathsf{NCGen}, \mathsf{NCEnc}, \mathsf{NCDec}, \mathsf{NCSim}\}$ be a receiver non-committing encryption, Δ be the (efficiently samplable) distribution that the ideal functionality wants to sample, and UniGen is a secure universal deniable encryption transformation. First we consider a bit encryption $\mathsf{E}' = \{\mathsf{Gen}, \mathsf{Enc}, \mathsf{Dec}\}$ that works as follows:

- $\mathsf{E}'.\mathsf{Gen}(1^\lambda)$: run $(\mathsf{pk}, \mathsf{sk}) \leftarrow \mathsf{E.NCGen}(1^\lambda)$. Output $(\mathsf{pk}, \mathsf{sk})$ as the public and secret keys.
- $\mathsf{E}'.\mathsf{Enc}(1^\lambda, \mathsf{pk}, b)$: sample $(\gamma_1, \gamma_2) \leftarrow \Delta(1^\lambda)$. Then output $c \leftarrow (\gamma_1, \mathsf{E.NCEnc}(\mathsf{pk}, \gamma_2 || b))$ as the ciphertext. The random coins of this process consist of the randomness used for sampling Δ, and that for encryption algorithm $\mathsf{E.NCEnc}$.
- $\mathsf{E}'.\mathsf{Dec}(1^\lambda, \mathsf{sk}, c)$: parse $c = (c_1, c_2)$. Run $\gamma_2 || b := \mathsf{E.NCDec}(\mathsf{sk}, c_2)$, and output b.

The CRS sampling. Let C be a circuit that implements $\mathsf{E}'.\mathsf{Enc}(1^\lambda)$. The sampling algorithm runs $(C_{\mathsf{encrypt}}, C_{\mathsf{explain}}) \leftarrow \mathsf{UniGen}(C)$, and outputs $\mathsf{crs} = (C_{\mathsf{encrypt}}, C_{\mathsf{explain}})$ as the CRS.

The protocol. The parties upon receiving $\mathsf{crs} = (C_{\mathsf{encrypt}}, C_{\mathsf{explain}})$ do the following:

- P_2 first samples a random string r_2 and runs $(\mathsf{pk}, \mathsf{sk}) \leftarrow \mathsf{E.NCGen}(1^\lambda; r_2)$ and sends pk to P_1.
- P_1 then samples a random string r_1, and runs $(\gamma_1, c_2) \leftarrow C_{\mathsf{encrypt}}(\mathsf{pk}, 0; r_1)$. Then P_1 locally outputs γ_1 and then sends c_2 to P_2.
- P_2 runs $\gamma_2 || 0 := \mathsf{E.NCDec}(\mathsf{sk}, c_2)$ and then outputs γ_2.

The transcript of the protocol is (pk, c_2).

Here it is important that P_1 does not directly use $\mathsf{E}'.\mathsf{Enc}$ to generate the ciphertext. Suppose he used $\mathsf{E}'.\mathsf{Enc}$ directly, then his random coins r_1 must contain information about the underlying message of $c_2 = \mathsf{E.Enc}(\mathsf{pk}, \gamma_2 || 0)$. We argue that it is impossible to satisfy our security requirement as follows.

Let us recall the security definition (Definition 2.3): to prove security, we need to construct a simulator $\mathcal{S} = (\mathcal{S}_{\text{tr}}, \mathcal{S}_1, \mathcal{S}_2)$ such that we require $\mathcal{S}_1$ to simulate the view of P_1 without knowing γ_2, and similarly $\mathcal{S}_2$ to simulate the view of P_2 without knowing γ_1. Therefore, a secure protocol cannot allow one to derive γ_2 from P_1's random coins r_1; otherwise, it is impossible for $\mathcal{S}_1$ (who does not know γ_2) to simulate such view of P_1.

To tackle such challenge, we use the universal deniable encryption transformation as Definition 3.2: to generate ciphertext of E', we use the program C_{encrypt}. Note that even if the randomness spaces for C_{encrypt} and $\mathsf{E}'.\mathsf{Enc}$ are different, the output distributions are statistically close, so using C_{encrypt} is essentially the same as using $\mathsf{E}'.\mathsf{Enc}$. More importantly, by the property of randomness explainability and the security of E, we can argue that the random coins r_1 (of the program C_{encrypt}) is only linked to the ciphertext (γ_1, c_2), but not the message γ_2 under c_2. More formally, we can argue that (r_1, γ_1, c_2) is indistinguishable from $(\tilde{r}_1, \gamma_1, \tilde{c}_2)$, where $\tilde{c}_2$ is a simulated ciphertext that does not contain information about γ_2, and $\tilde{r}_1$ is explained randomness by C_{explain}.

Using the ideas above, we are able to establish the following theorem:

Theorem 3.4 *Assume that* E *is a secure receiver non-committing encryption,* Δ *is an efficiently samplable distribution, and* UniGen *is a secure universal deniable transformation for the encryption scheme* E' *defined as above. Then the protocol described above realizes* $\mathcal{F}^{\Delta}_{\mathrm{sampling}}$ *with strong oblivious simulation, using the common reference string* crs.

Before proving the theorem, we give an *interesting* remarks about implications of our protocols to adaptive security in the MPC setting.

Remark 3.5 *The protocol above allows us to realize randomized functionalities beyond "adaptively well-formed" ones as discussed in the introduction. Recall that for an adaptively well-formed randomized functionality, the adversary gets the random coins of the functionality when all parties are corrupted. We go beyond this restriction. In our protocol above, the sampling is done in the* C_{encrypt}*, and we can simply use the* C_{explain} *to reconstruct the randomness. Essentially this gives the ideal functionality a way to erase the internal randomness after generating the outputs!*

For further exposition, we take the example from the work [17, Section 3.3]. Consider the randomized functionality that outputs a value N *to both* P_1 *and* P_2 *where* $N = p \cdot q$ *and* p, q *are randomly chosen (large) primes. To handle the case that all parties are corrupted without revealing the random coins of the functionality (i.e.* p, q*) to the adversary, essentially we need to be able to sample the domain* $\{N | N = pq\}$ *(or a domain that is computationally indistinguishable from it) without knowing* p *or* q*. The work [17] explicitly pointed out that this task may be possible, though the paper did not know how to do it.*

In this paper, we show that this task is exactly what universal deniable encryption can achieve! In our protocol above, by using C_{encrypt} *and some random coins* r*,* P_1 *is able to sample* N *without knowing the* (p, q)*. Then the simulator in the ideal world, via* C_{explain} *can come up with consistent and indistinguishable random coins* r' *that explains that* N is *computed based on* C_{encrypt} *and* r'*, even though the simulator is not able to learn such* p, q*.*

How is this possible? For readers who are familiar with the Sahai-Waters instantiation [44], we further elaborate on how things work with their concrete scheme: recall that C_{encrypt} *is an obfuscated circuit that contains some keys of (three) puncturable pseudo-random functions, say one of them is* $F_1(K_1, \cdot)$ *(consistent with the notation in [44]). When a user inputs some random coins* r*, if* r *does not hit some hidden trigger (the hitting probability is negligible), then the program will use* $u = F_1(K_1, r)$ *as the random coins to sample* N*. Since the whole process (i.e. the key* K_1 *and the computation of* u*) is inside the obfuscation (i.e.* C_{encrypt}*), thus the user can only obtain the output* N *without learning the underlying coins* u *(that may contain information about* p, q*).*

Proof: To prove the theorem, we need to construct a simulator $\mathcal{S} = (\mathcal{S}_{\mathrm{tr}}, \mathcal{S}_1, \mathcal{S}_2)$ such that the distribution of the real experiment **Real** $= (\mathsf{crs}, r_1, \tau, r_2)$ is indistinguishable from that of the simulation experiment **Ideal** $= (\mathsf{crs}, \tilde{r}_1, \tilde{\tau}, \tilde{r}_2)$ according to Definition 2.3.

Now, we describe the simulators. Let (γ_1, γ_2) be the output of the functionality $\mathcal{F}^{\Delta}_{\mathrm{sampling}}$, $\mathsf{NCSim} = (\mathsf{NCSim}_1, \mathsf{NCSim}_2)$ be the simulator(s) of the non-committing encryption scheme E, $\mathsf{crs} = (C_{\mathsf{encrypt}}, C_{\mathsf{explain}})$ be the CRS sampled as described above.

- $\mathcal{S}_{\mathrm{tr}}(\mathsf{crs})$ samples $(\tilde{\mathsf{pk}}, \tilde{c}_2) \leftarrow \mathsf{NCSim}_1(1^{\lambda})$ and then outputs $\tilde{\tau} = (\tilde{\mathsf{pk}}, \tilde{c}_2)$.
- $\mathcal{S}_1(\mathsf{crs}, \tilde{\tau}, \gamma_1)$ runs $\tilde{r}_1 \leftarrow C_{\mathsf{explain}}(\tilde{\mathsf{pk}}, (\gamma_1, \tilde{c}_2), 0)$. That is, $\mathcal{S}_1$ interprets $(\gamma_1, \tilde{c}_2)$ as a ciphertext of the scheme E', and uses C_{explain} to explain the randomness (for a ciphertext $\mathsf{E}'.\mathsf{Enc}(\mathsf{pk}, 0)$).

– $\mathcal{S}_2(\mathsf{crs}, \tilde{\tau}, \gamma_2)$ runs $\tilde{r}_2 \leftarrow \mathsf{NCSim}_2(\tilde{\mathsf{pk}}, \tilde{c}_2, \gamma_2||0)$. That is, $\mathcal{S}_2$ uses the simulator of the non-committing encryption to generate random coins that decrypt $\tilde{c}_2$ to $\gamma_2||0$.

Then we will establish the following claim, and the proof of the theorem follows directly from the claim.

Claim 3.6 *The following two distributions are computationally indistinguishable:* ***Real*** $= (\mathsf{crs}, r_1, \tau, r_2) \approx$ ***Ideal*** $= (\mathsf{crs}, \tilde{r}_1, \tilde{\tau}, \tilde{r}_2)$*, where the experiments are sampled as the protocol and the simulation described above. Recall that* r_2 *is* P_2*'s randomness that generates* pk*, and* r_1 *is* P_1*'s randomness that is used for* $C_{\mathsf{encrypt}}(\mathsf{pk}, 0)$.

Proof: To prove the claim, we consider the following hybrids:

The Real experiment. Real $= (\mathsf{crs}, r_1, \tau = (\mathsf{pk}, c_2), r_2)$: recall that in the real experiment, the transcript $\tau = (\mathsf{pk}, c_2)$ is generated as follows. pk is generated by P_2, and c_2 is one part of a ciphertext of E' generated by P_1, i.e. $(\gamma_1, c_2) = C_{\mathsf{encrypt}}(\mathsf{pk}, 0; r_1)$.

Hybrid 1. $H_1 = \big(\mathsf{crs}, \tilde{r}_1, \tau = \big(\mathsf{pk}, c_2\big), r_2\big)$: this experiment is the same as the real experiment except instead of outputting r_1 as the randomness of P_1, we use $\tilde{r}_1 \leftarrow C_{\mathsf{explain}}(\mathsf{pk}, (\gamma_1, c_2), 0)$. More precisely, H_1 first samples $(\mathsf{crs}, r_1, \tau, r_2)$ as the experiment **Real** (then γ_1, c_2 are defined), and replaces the r_1 with $\tilde{r}_1$ as described.

Hybrid 2. $H_2 = \big(\mathsf{crs}, \tilde{r}_1, \tau' = \big(\mathsf{pk}, c_2\big), r_2\big)$: this experiment is the same as H_1 except it does not use $C_{\mathsf{encrypt}}(\mathsf{pk}, 0)$ to generate the transcript. Instead, it samples $\mathsf{E}'.\mathsf{Enc}(\mathsf{pk}, 0)$ as follows: first it samples $(\gamma_1, \gamma_2) \leftarrow \Delta(1^\lambda)$, and then generates $c_2 \leftarrow \mathsf{E}.\mathsf{Enc}(\mathsf{pk}, \gamma_2)$. Then the experiment generates $\tilde{r}_1 \leftarrow C_{\mathsf{explain}}(\mathsf{pk}, (\gamma_1, c_2), 0)$ as H_1. Basically, this experiment runs $\mathsf{E}'.\mathsf{Enc}$ on its own to replace $C_{\mathsf{encrypt}}(\mathsf{pk}, 0)$.

Hybrid 3. $H_3 = \Big(\mathsf{crs}, \tilde{r}_1, \tilde{\tau} = \big(\tilde{\mathsf{pk}}, \tilde{c}_2\big), \tilde{r}_2\Big)$: this experiment is the same as H_2 except it runs $(\tilde{\mathsf{pk}}, \tilde{c}_2) \leftarrow \mathsf{NCSim}_1(1^\lambda)$ to generate the transcript. Finally, it runs $\tilde{r}_2 \leftarrow \mathsf{NCSim}_2(\tilde{\mathsf{pk}}, \tilde{c}_2, \gamma_2)$ to explain the randomness of P_2. Note that this experiment is identical to the simulation experiment **Ideal**.

Then we prove the adjacent hybrids are computationally indistinguishable by the following claims:

Claim 3.7 ***Real*** $\approx H_1$.

This is by the security of the universal deniable encryption transformation (property (b) of Definition 3.2). Suppose there exists a PPT distinguisher $\mathcal{D}$ that can distinguish **Real** from H_1 (with non-negligible probability), then we can construct a PPT distinguisher $\mathcal{D}'$ that breaks the property (b) as follows: $\mathcal{D}'$ takes input $(C_{\mathsf{encrypt}}, C_{\mathsf{explain}}, \mathsf{pk}, (\gamma^*, c^*), r^*)$ where $C_{\mathsf{encrypt}}, C_{\mathsf{explain}}$ are generated as the universal deniable encryption transformation setup, (i.e. $\mathsf{UniGen}(\mathsf{E}'.\mathsf{Enc}(1^\lambda, \cdot, \cdot))$), $(\gamma^*, c^*) \leftarrow C_{\mathsf{encrypt}}(\mathsf{pk}, 0)$, and r^* is either the one that generated (γ^*, c^*) or sampled by $C_{\mathsf{explain}}(\mathsf{pk}, (\gamma^*, c^*), 0)$.

Then $\mathcal{D}'$ interprets $\mathsf{crs} = (C_{\mathsf{encrypt}}, C_{\mathsf{explain}})$, samples a random string r_2, and runs $\mathcal{D}(\mathsf{crs}, r^*, (\mathsf{pk}, c^*), r_2)$ and outputs whatever $\mathcal{D}$ outputs. Suppose r^* is distributed according to the former, then the input to $\mathcal{D}$ is distributed identical to **Real**. On the other hand, suppose r^* is distributed as the latter, then the input to $\mathcal{D}$ is distributed identical to H_1. Thus suppose $\mathcal{D}$ can distinguish **Real** from H_1, $\mathcal{D}'$ break security of the property (b).

Then we are going to show:

Claim 3.8 $H_1 \approx H_2$.

This is by the security property (a) of Definition 3.2), which says that $\{C_{\mathsf{encrypt}}(\mathsf{pk}, 0)\}$ is statistically close to $\{\mathsf{E}'.\mathsf{Enc}(\mathsf{pk}, 0)\}$. The only difference between H_1 and H_2 is the generation of the (γ_1, c_2). In H_1 it was generated by $C_{\mathsf{encrypt}}(\mathsf{pk}, 0)$, and in H_2 it was generated by $\mathsf{E}'.\mathsf{Enc}(\mathsf{pk}, 0)$. By the property (a), we know that the distributions of generating (γ_1, c_2) in both ways are statistically close. Thus H_1 is statistically close to H_2.

Then we are going to show:

Claim 3.9 $H_2 \approx H_3$.

This is by the security of the non-committing encryption E (as Definition 3.1). That is, suppose there exists a PPT distinguisher that can distinguish H_2 from H_3 (with non-negligible probability), then there exists a PPT distinguisher $\mathcal{D}'$ that breaks the non-committing encryption E as follows:

- $\mathcal{D}'$ first samples $(\gamma_1, \gamma_2) \leftarrow \Delta(1^\lambda)$ and sets $m = \gamma_2 || 0$.
- $\mathcal{D}'$ takes input $(\mathsf{pk}^*, c^*, \sigma^*)$ (from the challenger), which is distributed according to either the honest view of encryption of m or the simulated view as Definition 3.1.
- It samples $\mathsf{crs} = (C_{\mathsf{encrypt}}, C_{\mathsf{explain}}) \leftarrow \mathsf{UniGen}(C)$ where C is an encryption circuit of E'. This step is independent of the input.
- It generates $\tilde{r}_1 \leftarrow C_{\mathsf{explain}}(\mathsf{pk}^*, (\gamma_1, c^*), 0)$.
- It runs $\mathcal{D}(\mathsf{crs}, \tilde{r}_1, (\mathsf{pk}^*, c^*), \sigma^*)$, and outputs whatever $\mathcal{D}$ outputs.

It is clear that if the input $(\mathsf{pk}^*, c^*, \sigma^*)$ is distributed as the honest view of encryption of m, then $(\mathsf{crs}, \tilde{r}_1, (\mathsf{pk}^*, c^*), \sigma^*)$ is distributed identicalyl to H_2. On the other hand, if that is distributed as the simulated view, then $(\mathsf{crs}, \tilde{r}_1, (\mathsf{pk}^*, c^*), \sigma^*)$ is distributed identically to H_3. Thus, suppose $\mathcal{D}$ distinguishes H_2 from H_3 with a non-negligible probability, $\mathcal{D}'$ breaks the receiver non-committing encryption scheme E. This completes the proof of the claim.

Finally, we observe that the experiment H_3 is identical to the experiment **Ideal** output by the simulator. Thus by Claims 3.7, 3.8, 3.9, we prove Claim 3.6, i.e. **Real** $\approx$ **Ideal**. This completes the proof of Theorem 3.4. ■

■

4 Hardwares in JV and DF Schemes

In this section, we present concretely how to express the hardwares in the previous hardware-based schemes of Juma and Vahlis [34] and Dziembowski and Faust [21] as the ideal functionality $\mathcal{F}^{\Delta}_{\text{sampling}}$. Thus, we can instantiate the hardware of the JV (resp. DF) two-component OCL scheme with the two-party protocol in Theorem 3.4 and apply Theorem 2.5 to obtain the first two-component OCL schemes without secure hardware.

4.1 Sampling Distribution for the Juma-Vahlis Compiler

We define the sampling distribution Δ_{JV} for functionality $\mathcal{F}^{\Delta_{\mathsf{JV}}}_{\text{sampling}}$ that provided by the trusted hardware of the Juma-Vahlis compiler (a description of the compiler can be found in Appendix B.1). Let $\mathsf{FHE} = \mathsf{FHE}.\{\mathsf{Gen}, \mathsf{Enc}, \mathsf{Dec}, \mathsf{Eval}\}$ be a fully homomorphic encryption scheme with the additional cipher refreshing properties required by the JV construction. The distribution $\Delta_{\mathsf{JV}}(1^\lambda)$ is defined as follows:

- Sample $(\mathsf{pk}, \mathsf{sk}) \leftarrow \mathsf{Gen}(1^\lambda)$; and then sample $\mathsf{ct}_0 \leftarrow \mathsf{Enc}_{\mathsf{pk}}(0)$ and $\mathsf{ct}'_0 \leftarrow \mathsf{Enc}_{\mathsf{pk}}(0)$.
- Output (γ_1, γ_2), where $\gamma_1 = (\mathsf{pk}, \mathsf{ct}_0, \mathsf{ct}'_0)$, adn $\gamma_2 = (\mathsf{pk}, \mathsf{sk})$.

Juma and Vahlis [34] showed that assuming FHE is a fully homomorphic encryption (with cipher refreshing) that is secure against $2^{O(\ell(\lambda))}$ adversaries, then there exists a two-component OCL scheme in the $\mathcal{F}^{\Delta_{\mathsf{JV}}}_{\text{sampling}}$ hybrid world that is $O(\ell)$-leakage resilient. We denote the scheme as $\Lambda^{\mathcal{F}}_{\mathsf{JV}} = (\mathsf{Comp}_{\mathsf{JV}}, \Pi^{\mathcal{F}}_{\mathsf{JV}})$ where $\mathcal{F} = \mathcal{F}^{\Delta_{\mathsf{JV}}}_{\text{sampling}}$. By our Theorem 3.4, we can realize the functionality that provided by the trusted hardware with a protocol φ_{JV} with strong oblivious simulation. Leveraging our OCL Hardware Replacement Theorem (Theorem 2.5), we can obtain an OCL scheme $\Lambda'_{\mathsf{JV}} = (\mathsf{Comp}'_{\mathsf{JV}}, \Pi^{\varphi_{\mathsf{JV}}}_{\mathsf{JV}})$ that does not require any secure hardware. Formally, we obtain the following theorem:

Theorem 4.1 *Assume there exist a secure receiver non-committing encryption scheme and a secure universal deniable encryption transformation for any encryption scheme,*[8] *and* FHE *is a fully homomorphic encryption (with cipher refreshing) that is secure against* $2^{O(\ell(\lambda))}$ *adversaries. Then* $\Lambda'_{\mathtt{JV}}$ *is* $O(\ell)$*-leakage resilient, where* λ *is security parameter.*

4.2 Sampling Distribution for the Dziembowski-Faust Compiler

Here we define the distribution $\Delta_{\mathtt{DF}}$ for functionality $\mathcal{F}^{\Delta_{\mathtt{DF}}}_{\text{sampling}}$ that provided by the trusted hardware of the Dziembowski-Faust compiler. In the initialization stage of the DF compiler, a private circuit C is compiled. Afterwards, in each evaluation when P_1 obtains an input x, the parties then jointly compute the universal boolean circuit $U(\cdot,\cdot)$ on the underlying input (C, x), and eventually P_2 returns output $y = C(x)$. Please refer to Appendix B.2 for a description of the DF compiler[9]. Let $\mathbb{F}_2$ be binary field, and each share used in DF compiler be of length m. Note that the length m is related to the amount of leakage that can be tolerated as described in the following theorem statement. Let $\mathcal{O}^n_b$ be the uniform distribution on $(L, R) \in \mathbb{F}_2^{n\times n}$ conditioned on $\langle L, R\rangle = b$. Without loss of generality, assume the universal boolean circuit $U(\cdot,\cdot)$ consists of T number of NAND gates, labeled with a set $G = \{1, \ldots, T\}$. The distribution $\Delta_{\mathtt{DF}}(1^\lambda)$ is defined as follows:

- For $g \in G$, sample vectors $(L'_g||L''_g, R'_g||R''_g) \leftarrow \mathcal{O}^{2m^2}_0$;
  for $j \le |C|$, sample vectors $(A'_j||A''_j, B'_j||B''_j) \leftarrow \mathcal{O}^{2m}_0$,
- Output (γ_1, γ_2), where $\gamma_1 = \big(\{L'_g||L''_g\}_{g\in G}, \{A'_j||A''_j\}_{j\le|C|}\big)$ and $\gamma_2 = \big(\{R'_g||R''_g\}_{g\in G}, \{B'_j||B''_j\}_{j\le|C|}\big)$.

Dziembowski and Faust [21] showed that (without any cryptographic assumption) there exists a two-component OCL scheme in the $\mathcal{F}^{\Delta_{\mathtt{DF}}}_{\text{sampling}}$ hybrid world, denoted as $\Lambda^{\mathcal{F}}_{\mathtt{DF}} = (\mathsf{Comp}_{\mathtt{DF}}, \Pi^{\mathcal{F}}_{\mathtt{DF}})$ where $\mathcal{F} = \mathcal{F}^{\Delta_{\mathtt{DF}}}_{\text{sampling}}$. By our Theorem 3.4, we can realize the functionality that provided by the hardware with a protocol $\varphi_{\mathtt{DF}}$ with strong oblivious simulation. Leveraging our OCL Hardware Replacement Theorem (Theorem 2.5), we can obtain an OCL scheme $\Lambda'_{\mathtt{DF}} = (\mathsf{Comp}'_{\mathtt{DF}}, \Pi^{\varphi_{\mathtt{DF}}}_{\mathtt{DF}})$ that does not require any secure hardware. Formally, we obtain the following theorem:

Theorem 4.2 *Assume there exist a secure receiver non-committing encryption scheme and a secure universal deniable encryption transformation for any encryption scheme. Then* $\Lambda'_{\mathtt{DF}}$ *is* ℓ*-leakage resilient for* $\ell(\lambda) = m(\lambda)/10, m(\lambda) = \omega(\log(\lambda))$*, where* λ *is the security parameter.*

5 Extension: Multi-Component OCL Schemes

In this section, we discuss some extensions of our main results. First we note that the hardware replacement theorem (Theorem 2.5) also holds for any N-component OCL schemes. Even if we have two-component constructions (in this paper), still potentially, there can be other more-component constructions that are more efficient or achieve better leakage rate. As we emphasize before, this can be viewed as a general design paradigm of OCL constructions. The definition of N-component OCL can be found in the work of Bitansky et al. [10], and the corresponding hybrid schemes can be defined analogously. A natural extension of strong oblivious simulation (Definition 2.3) for N-party protocols can be defined as follow:

Definition 5.1 (Strong oblivious simulation for N-party protocols) *Let* $\mathsf{crs} \leftarrow \mathsf{CRS.Gen}(1^\lambda)$*, and let* $\pi = (P_1, \ldots, P_N)$ *be an* N*-party protocol using such common reference string* crs*. Let* $\mathcal{F}$ *be an* N*-input (perhaps randomized) ideal functionality. We say* π *realizes the functionality* $\mathcal{F}$ *with strong oblivious*

[8]This can be implied by the existence of indistinguishable obfuscation and one-way functions (Theorem 3.3).

[9]Our presentation of the DF scheme adapts from the simplified version of the DF scheme presented in the work [9].

simulation, if there exists a PPT *simulator* $\mathcal{S} = (\mathcal{S}_1, \ldots, \mathcal{S}_N, \mathcal{S}_{\text{tr}})$ *for all (non-uniform)* PPT *adversary* $\mathcal{A}$ *such that the following distributions are computationally indistinguishable:*

$$\left\{ \begin{array}{r} \mathsf{crs} \leftarrow \mathsf{CRS.Gen}(1^\lambda); (x_1, \ldots, x_N) \leftarrow \mathcal{A}(\mathsf{crs}); \\ (r_1, \ldots, r_N) \leftarrow (U_\lambda)^N; \\ \tau = \langle P_1(\mathsf{crs}, x_1; r_1), \ldots, P_N(\mathsf{crs}, x_N; r_N) \rangle \\ : (\mathsf{crs}, \{x_i, r_i\}_{i \in [N]}, \tau) \end{array} \right\} \approx \left\{ \begin{array}{r} \mathsf{crs} \leftarrow \mathsf{CRS.Gen}(1^\lambda); (x_1, x_2) \leftarrow \mathcal{A}(\mathsf{crs}); \\ (y_1, \ldots, y_N) \leftarrow \mathcal{F}(x_1, \ldots, x_N); \tilde{\tau} \leftarrow \mathcal{S}_{\text{tr}}(\mathsf{crs}); \\ \tilde{r}_i \leftarrow \mathcal{S}_i(\mathsf{crs}, x_i, y_i, \tilde{\tau}); \\ : (\mathsf{crs}, \{x_i, \tilde{r}_i\}_{i \in [N]}, \tilde{\tau}) \end{array} \right\}.$$

Note that r_i*'s are the random coins of the parties, and* τ *is the transcript (i.e. message exchanges) by running the* N*-party protocol .*

A similar OCL hardware replacement theorem can be obtained for N-component OCL scheme. Since the proof is essentially the same as that of Theorem 2.5, we only state the theorem but omit the proof.

Theorem 5.2 (N-component OCL hardware replacement theorem) *Let* $\mathcal{F}$ *be some* N*-party functionality, and* $\Lambda^{\mathcal{F}} = (\mathsf{Comp}, \Pi^{\mathcal{F}})$ *be a* $\mathcal{F}$*-hybrid* N*-component OCL scheme that is* ℓ*-leakage-resilient with oblivious simulation. Suppose there exists an* N*-party protocol* ρ *using a common reference string* crs*, that realizes* $\mathcal{F}$ *with strong oblivious simulation as above. Then there exists an* N*-component OCL scheme* $\Lambda' = (\mathsf{Comp}', \Pi^{\rho})$ *that is* ℓ*-leakage-resilient with oblivious simulation.*

Our construction in Section 3.3 can be extended to any N-output $\mathcal{F}^{\Delta}_{\text{sampling}}$ functionality for any N-output distribution Δ. Let $\mathsf{E} = \{\mathsf{NCGen}, \mathsf{NCEnc}, \mathsf{NCDec}, \mathsf{NCSim}\}$ be a receiver non-committing encryption, Δ be an N-output distribution that the ideal functionality wants to sample, and UniGen is a secure universal deniable encryption transformation. Similarly we consider a bit encryption $\mathsf{E}' = \{\mathsf{Gen}, \mathsf{Enc}, \mathsf{Dec}\}$ that works as follows:

- $\mathsf{E}'.\mathsf{Gen}(1^\lambda)$: run $(\mathsf{pk}_2, \mathsf{sk}_2, \ldots, \mathsf{pk}_N, \mathsf{sk}_N) \leftarrow \mathsf{E.NCGen}(1^\lambda)$ (running the generation $N-1$ times. Here we deliberately start the index with 2.). Set the public key to be $\mathsf{pk} = (\mathsf{pk}_2, \ldots, \mathsf{pk}_N)$, and $\mathsf{sk} = (\mathsf{sk}_2, \ldots, \mathsf{sk}_N)$.
- $\mathsf{E}'.\mathsf{Enc}(1^\lambda, \mathsf{pk}, b)$: sample $(\gamma_1, \ldots, \gamma_N) \leftarrow \Delta(1^\lambda)$. Then output

  $$c \leftarrow (\gamma_1, \mathsf{E.NCEnc}(\mathsf{pk}_2, \gamma_2 || b), \ldots, \mathsf{E.NCEnc}(\mathsf{pk}_N, \gamma_N || b))$$

  as the ciphertext. The random coins of this process consist of the randomness used for sampling Δ, and that for encryption algorithm $\mathsf{E.NCEnc}$.
- $\mathsf{E}'.\mathsf{Dec}(1^\lambda, \mathsf{sk}, c)$: parse $c = (\gamma_1, c_2, \ldots, c_N)$. Run $\gamma_2 || b := \mathsf{E.NCDec}(\mathsf{sk}, c_2)$, and output b.

The CRS sampling. Let C be a circuit that implements $\mathsf{E}'.\mathsf{Enc}(1^\lambda)$. The sampling algorithm runs $(C_{\mathsf{encrypt}}, C_{\mathsf{explain}}) \leftarrow \mathsf{UniGen}(C)$, and outputs $\mathsf{crs} = (C_{\mathsf{encrypt}}, C_{\mathsf{explain}})$ as the CRS.

The protocol. The parties upon receiving $\mathsf{crs} = (C_{\mathsf{encrypt}}, C_{\mathsf{explain}})$ do the following:

- For $i \in [N] \setminus \{1\}$, P_i first samples a random string r_i and runs $(\mathsf{pk}_i, \mathsf{sk}_i) \leftarrow \mathsf{E.NCGen}(1^\lambda; r_i)$ and sends pk_i to P_1.
- P_1 then samples a random string r_1, and runs $(\gamma_1, c_2, \ldots, c_N) \leftarrow C_{\mathsf{encrypt}}(\mathsf{pk}, 0; r_1)$. Then P_1 locally outputs γ_1 and then sends c_i to P_i for all $i \in [N] \setminus \{1\}$.
- For $i \in [N] \setminus \{1\}$, P_i runs $\gamma_i || 0 := \mathsf{E.NCDec}(\mathsf{sk}_i, c_i)$ and then outputs γ_i.

The analysis of the protocol is essentially the same the previous one. For succinctness of presentation, we only state the theorem below, but omit the details to avoid repetition.

Theorem 5.3 *Assume that* E *is a secure receiver non-committing encryption,* Δ *is an efficiently samplable* N*-output distribution, and* UniGen *is a secure universal deniable transformation for the encryption scheme* E' *defined as above. Then the* N*-party protocol described above realizes* $\mathcal{F}^{\Delta}_{\mathrm{sampling}}$ *with strong oblivious simulation, using the common reference string* crs.

Similar to the two-party case, the connection between the above protocol and MPC was already discussed in the introduction and Definition 2.3. We restate the implication: for any N-party randomized functionality (even beyond the adaptively well-formed ones [17]; the discussions in Remark 3.5 also apply to the N-party setting), we are able to construct a protocol that is adaptively secure and leakage tolerance, using the above construction.

References

[1] Dakshi Agrawal, Bruce Archambeault, Josyula R. Rao, and Pankaj Rohatgi. The EM side-channel(s). In Burton S. Kaliski Jr., Çetin Kaya Koç, and Christof Paar, editors, *CHES 2002*, volume 2523 of *LNCS*, pages 29–45. Springer, August 2002.

[2] Prabhanjan Ananth, Divya Gupta, Yuval Ishai, and Amit Sahai. Optimizing obfuscation: Avoiding barrington's theorem. Cryptology ePrint Archive, Report 2014/222, 2014. `http://eprint.iacr.org/2014/222`.

[3] Boaz Barak, Sanjam Garg, Yael Tauman Kalai, Omer Paneth, and Amit Sahai. Protecting obfuscation against algebraic attacks. In Phong Q. Nguyen and Elisabeth Oswald, editors, *EUROCRYPT 2014*, volume 8441 of *LNCS*, pages 221–238. Springer, May 2014.

[4] Boaz Barak, Oded Goldreich, Russell Impagliazzo, Steven Rudich, Amit Sahai, Salil P. Vadhan, and Ke Yang. On the (im)possibility of obfuscating programs. In Joe Kilian, editor, *CRYPTO 2001*, volume 2139 of *LNCS*, pages 1–18. Springer, August 2001.

[5] Boaz Barak, Oded Goldreich, Russell Impagliazzo, Steven Rudich, Amit Sahai, Salil P. Vadhan, and Ke Yang. On the (im)possibility of obfuscating programs. *Journal of the ACM*, 59(2):6, 2012.

[6] Eli Biham and Adi Shamir. Differential fault analysis of secret key cryptosystems. In Burton S. Kaliski Jr., editor, *CRYPTO'97*, volume 1294 of *LNCS*, pages 513–525. Springer, August 1997.

[7] Nir Bitansky, Ran Canetti, Shafi Goldwasser, Shai Halevi, Yael Tauman Kalai, and Guy N. Rothblum. Program obfuscation with leaky hardware. In Dong Hoon Lee and Xiaoyun Wang, editors, *ASIACRYPT 2011*, volume 7073 of *LNCS*, pages 722–739. Springer, December 2011.

[8] Nir Bitansky, Ran Canetti, and Shai Halevi. Leakage-tolerant interactive protocols. In Ronald Cramer, editor, *TCC 2012*, volume 7194 of *LNCS*, pages 266–284. Springer, March 2012.

[9] Nir Bitansky, Dana Dachman-Soled, and Huijia Lin. Personal communication, 2014.

[10] Nir Bitansky, Dana Dachman-Soled, and Huijia Lin. Leakage-tolerant computation with input-independent preprocessing. In Juan A. Garay and Rosario Gennaro, editors, *CRYPTO 2014, Part II*, volume 8617 of *LNCS*, pages 146–163. Springer, August 2014.

[11] Zvika Brakerski and Guy N. Rothblum. Virtual black-box obfuscation for all circuits via generic graded encoding. In Yehuda Lindell, editor, *TCC 2014*, volume 8349 of *LNCS*, pages 1–25. Springer, February 2014.

[12] David Brumley and Dan Boneh. Remote timing attacks are practical. *Computer Networks*, 48(5):701–716, 2005.

[13] Ran Canetti. Universally composable security: A new paradigm for cryptographic protocols. Cryptology ePrint Archive, Report 2000/067, 2000. `http://eprint.iacr.org/2000/067`.

[14] Ran Canetti. Universally composable security: A new paradigm for cryptographic protocols. In *42nd FOCS*, pages 136–145. IEEE Computer Society Press, October 2001.

[15] Ran Canetti, Uriel Feige, Oded Goldreich, and Moni Naor. Adaptively secure multi-party computation. In *28th ACM STOC*, pages 639–648. ACM Press, May 1996.

[16] Ran Canetti, Shai Halevi, and Jonathan Katz. Adaptively-secure, non-interactive public-key encryption. In Joe Kilian, editor, *TCC 2005*, volume 3378 of *LNCS*, pages 150–168. Springer, February 2005.

[17] Ran Canetti, Yehuda Lindell, Rafail Ostrovsky, and Amit Sahai. Universally composable two-party and multi-party secure computation. In *34th ACM STOC*, pages 494–503. ACM Press, May 2002. `http://eprint.iacr.org/2002/140`.

[18] Suresh Chari, Charanjit S. Jutla, Josyula R. Rao, and Pankaj Rohatgi. Towards sound approaches to counteract power-analysis attacks. In Michael J. Wiener, editor, *CRYPTO'99*, volume 1666 of *LNCS*, pages 398–412. Springer, August 1999.

[19] Seung Geol Choi, Dana Dachman-Soled, Tal Malkin, and Hoeteck Wee. Improved non-committing encryption with applications to adaptively secure protocols. In Mitsuru Matsui, editor, *ASIACRYPT 2009*, volume 5912 of *LNCS*, pages 287–302. Springer, December 2009.

[20] Alexandre Duc, Stefan Dziembowski, and Sebastian Faust. Unifying leakage models: From probing attacks to noisy leakage. In Phong Q. Nguyen and Elisabeth Oswald, editors, *EUROCRYPT 2014*, volume 8441 of *LNCS*, pages 423–440. Springer, May 2014.

[21] Stefan Dziembowski and Sebastian Faust. Leakage-resilient circuits without computational assumptions. In Ronald Cramer, editor, *TCC 2012*, volume 7194 of *LNCS*, pages 230–247. Springer, March 2012.

[22] Sebastian Faust, Tal Rabin, Leonid Reyzin, Eran Tromer, and Vinod Vaikuntanathan. Protecting circuits from leakage: the computationally-bounded and noisy cases. In Henri Gilbert, editor, *EUROCRYPT 2010*, volume 6110 of *LNCS*, pages 135–156. Springer, May 2010.

[23] Sanjam Garg, Craig Gentry, Shai Halevi, Mariana Raykova, Amit Sahai, and Brent Waters. Candidate indistinguishability obfuscation and functional encryption for all circuits. In *54th FOCS*, pages 40–49. IEEE Computer Society Press, October 2013.

[24] Daniel Genkin, Itamar Pipman, and Eran Tromer. Get your hands off my laptop: Physical side-channel key-extraction attacks on PCs. In *CHES 2014*, pages 242–260, 2014.

[25] Daniel Genkin, Adi Shamir, and Eran Tromer. RSA key extraction via low-bandwidth acoustic cryptanalysis. In Juan A. Garay and Rosario Gennaro, editors, *CRYPTO 2014, Part I*, volume 8616 of *LNCS*, pages 444–461. Springer, August 2014.

[26] Craig Gentry, Allison Lewko, Amit Sahai, and Brent Waters. Indistinguishability obfuscation from the multilinear subgroup elimination assumption. Cryptology ePrint Archive, Report 2014/309, 2014. `http://eprint.iacr.org/2014/309`.

[27] Oded Goldreich, Silvio Micali, and Avi Wigderson. How to play any mental game or A completeness theorem for protocols with honest majority. In Alfred Aho, editor, *19th ACM STOC*, pages 218–229. ACM Press, May 1987.

[28] Shafi Goldwasser and Guy N. Rothblum. Securing computation against continuous leakage. In Tal Rabin, editor, *CRYPTO 2010*, volume 6223 of *LNCS*, pages 59–79. Springer, August 2010.

[29] Shafi Goldwasser and Guy N. Rothblum. How to compute in the presence of leakage. In *53rd FOCS*, pages 31–40. IEEE Computer Society Press, October 2012.

[30] J. Alex Halderman, Seth D. Schoen, Nadia Heninger, William Clarkson, William Paul, Joseph A. Calandrino, Ariel J. Feldman, Jacob Appelbaum, and Edward W. Felten. Lest we remember: Cold boot attacks on encryption keys. In *USENIX Security Symposium*, pages 45–60, 2008.

[31] Shai Halevi and Huijia Lin. After-the-fact leakage in public-key encryption. In Yuval Ishai, editor, *TCC 2011*, volume 6597 of *LNCS*, pages 107–124. Springer, March 2011.

[32] Yuval Ishai, Amit Sahai, and David Wagner. Private circuits: Securing hardware against probing attacks. In Dan Boneh, editor, *CRYPTO 2003*, volume 2729 of *LNCS*, pages 463–481. Springer, August 2003.

[33] Stanislaw Jarecki and Anna Lysyanskaya. Adaptively secure threshold cryptography: Introducing concurrency, removing erasures. In Bart Preneel, editor, *EUROCRYPT 2000*, volume 1807 of *LNCS*, pages 221–242. Springer, May 2000.

[34] Ali Juma and Yevgeniy Vahlis. Protecting cryptographic keys against continual leakage. In Tal Rabin, editor, *CRYPTO 2010*, volume 6223 of *LNCS*, pages 41–58. Springer, August 2010.

[35] John Kelsey, Bruce Schneier, David Wagner, and Chris Hall. Side channel cryptanalysis of product ciphers. *Journal of Computer Security*, 8(2/3):141–158, 2000.

[36] Paul C. Kocher. Timing attacks on implementations of Diffie-Hellman, RSA, DSS, and other systems. In Neal Koblitz, editor, *CRYPTO'96*, volume 1109 of *LNCS*, pages 104–113. Springer, August 1996.

[37] Paul C. Kocher, Joshua Jaffe, and Benjamin Jun. Differential power analysis. In Michael J. Wiener, editor, *CRYPTO'99*, volume 1666 of *LNCS*, pages 388–397. Springer, August 1999.

[38] Feng-Hao Liu and Anna Lysyanskaya. Tamper and leakage resilience in the split-state model. In Reihaneh Safavi-Naini and Ran Canetti, editors, *CRYPTO 2012*, volume 7417 of *LNCS*, pages 517–532. Springer, August 2012.

[39] Silvio Micali and Leonid Reyzin. Physically observable cryptography (extended abstract). In Moni Naor, editor, *TCC 2004*, volume 2951 of *LNCS*, pages 278–296. Springer, February 2004.

[40] Eric Miles. Iterated group products and leakage resilience against NC1. In Moni Naor, editor, *ITCS 2014*, pages 261–268. ACM, January 2014.

[41] Eric Miles and Emanuele Viola. Shielding circuits with groups. In Dan Boneh, Tim Roughgarden, and Joan Feigenbaum, editors, *45th ACM STOC*, pages 251–260. ACM Press, June 2013.

[42] Rafael Pass, Karn Seth, and Sidharth Telang. Indistinguishability obfuscation from semantically-secure multilinear encodings. In Juan A. Garay and Rosario Gennaro, editors, *CRYPTO 2014, Part I*, volume 8616 of *LNCS*, pages 500–517. Springer, August 2014.

[43] Guy N. Rothblum. How to compute under AC^0 leakage without secure hardware. In Reihaneh Safavi-Naini and Ran Canetti, editors, *CRYPTO 2012*, volume 7417 of *LNCS*, pages 552–569. Springer, August 2012.

[44] Amit Sahai and Brent Waters. How to use indistinguishability obfuscation: deniable encryption, and more. In David B. Shmoys, editor, *46th ACM STOC*, pages 475–484. ACM Press, May / June 2014.

[45] Eran Tromer, Dag Arne Osvik, and Adi Shamir. Efficient cache attacks on aes, and countermeasures. *J. Cryptology*, 23(1):37–71, 2010.

[46] Brent Waters. CS 395T Special Topic: Obfuscation in Cryptography. 2014. `http://www.cs.utexas.edu/~bwaters/classes/CS395T-Fall-14/outline.html`.

[47] Brent Waters. How to use indistinguishability obfuscation. In *Visions of Cryptography*, 2014. Talk slides available at `http://www.cs.utexas.edu/~bwaters/presentations/files/how-to-use-IO.ppt`.

[48] Virginia Vassilevska Williams. Multiplying matrices faster than coppersmith-winograd. In Howard J. Karloff and Toniann Pitassi, editors, *44th ACM STOC*, pages 887–898. ACM Press, May 2012.

A Supplement for Section 2

A.1 Strong Leakage Resilience

Following Bitansky et al. [10] we define the notion of strong two-component OCL schemes (also referred to as two-component OCL with strong simulation). Recall that the oblivious simulator $\mathcal{S}$ described above reconstructs the states of the two components in each evaluation i depending only on (x_i, y_i). Strong OCL schemes are equipped with an oblivious simulator that satisfies an additional property. Specifically, a strong two-component OCL scheme has a simulator $\mathcal{S} = (\mathcal{S}_1, \mathcal{S}_2)$ where $\mathcal{S}_1$ takes (x_i, y_i) as input and reconstructs the state of the first component and $\mathcal{S}_2$ takes y_i as input reconstructs the state of the second component. Thus, in strong OCL, the simulation of the first component depends on both the input and output, (x_i, y_i), but the simulation of the second component depends solely on the output, y_i. As discussed previously, it was shown in [10] that strong two-component OCL can be used to obtain constructions of leakage-tolerant two-party computation.

Definition A.1 (Continual ℓ-leakage-resilience with strong simulation) *We say that a continual OCL scheme* OCL *is continually ℓ-leakage-resilient with strong oblivious simulation if it satisfies the following property:*

Strong ℓ-leakage resilience: OCL *admits an oblivious simulator $\mathcal{S}$ satisfying Definition 2.2 with the following structure: $\mathcal{S}$ consists of two sub-algorithms $(\mathcal{S}_1, \mathcal{S}_2)$ and on input $(1^\lambda, i, x_i, y_i\ ;\ \mathbf{w}_i)$, $\mathcal{S}$ invokes these sub-algorithms as follows:*

- $\mathcal{S}_1(1^\lambda, i, x_i, y_i;\ \mathbf{w}_i) = (\widetilde{\mathsf{intl}}_{i,1}, \widetilde{\mathsf{evl}}_{i,1})$
- $\mathcal{S}_2(1^\lambda, i, y_i;\ \mathbf{w}_i) = (\widetilde{\mathsf{intl}}_{i,2}, \widetilde{\mathsf{evl}}_{i,2})$

and outputs $(\widetilde{\mathsf{intl}}_{i,1}, \widetilde{\mathsf{intl}}_{i,2}, \widetilde{\mathsf{evl}}_{i,1}, \widetilde{\mathsf{evl}}_{i,2})$.

A similar hardware replacement theorem can be achieved: suppose we have a strong OCL in some $\mathcal{F}$-hybrid world, and a two-party protocol ρ that realizes $\mathcal{F}$ with additional properties as Definition 2.3, then we can construct a strong OCL without hardware. We present the formal statement as the following Theorem A.2. The only difference between this theorem and Theorem 2.5 is the condition of strong. The proof of Theorem A.2 is almost identical to that of Theorem 2.5 except we use the stronger underlying simulator $\widehat{\mathcal{S}} = (\widehat{\mathcal{S}}_1, \widehat{\mathcal{S}}_2)$ to generate the evaluation states.[10] We omit the almost repetitive proof, and refer curious readers to read the proof of Theorem 2.5 directly in Section A.2.

[10]Recall that we use $\widehat{S}$ to denote the underlying simulator of $\Lambda^{\mathcal{F}}$, and $\widehat{S}^{\rho}$ to denote the simulator for the protocol ρ.

Theorem A.2 (Hardware replacement theorem for strong OCL) *Let $\mathcal{F}$ be some two-party functionality, and $\Lambda^{\mathcal{F}} = (\mathsf{Comp}, \Pi^{\mathcal{F}})$ be a $\mathcal{F}$-hybrid two-component OCL scheme that is ℓ-leakage-resilient with* strong *oblivious simulation. Suppose there exists a two-party protocol ρ using a common reference string* crs*, that realizes $\mathcal{F}$ with strong oblivious simulation as Definition 2.3. Then there exists a two-component OCL scheme $\Lambda' = (\mathsf{Comp}', \Pi^{\rho})$ that is ℓ-leakage-resilient with* strong *oblivious simulation.*

A.2 Proof for Theorem 2.5

From the premise of the theorem, we have a two-component OCL scheme $\Lambda^{\mathcal{F}} = (\mathsf{Comp}, \Pi^{\mathcal{F}})$ that is ℓ-leakage-resilient with oblivious simulation, and a two-party protocol ρ that realizes $\mathcal{F}$ using a common reference string crs. This means, there exists a simulator $\widetilde{\mathcal{S}}$ such that for all adversaries $\mathcal{A}$, the view in the real ($\mathcal{F}$-hybrid) world $\mathbf{Real}^{\mathcal{F}}_{\mathcal{A}}$ is indistinguishable from the view in ideal world $\mathbf{Ideal}_{\widetilde{\mathcal{S}},\mathcal{A}}$ produced by the simulator; on the other hand, for the protocol ρ, we have a simulator $\widehat{\mathcal{S}}^{\rho} = (\widehat{\mathcal{S}}^{\rho}_1, \widehat{\mathcal{S}}^{\rho}_2, \widehat{\mathcal{S}}^{\rho}_{\text{tr}})$ such that for all (poly-sized) adversaries $\mathcal{A}$, the simulator can produce an indistinguishable view as Definition 2.3.

To prove the theorem, first we construct a scheme $\Lambda' = (\mathsf{Comp}', \Pi^{\rho})$ as stated in the paragraph $\mathcal{F}$-hybrid OCL schemes: Comp' just samples a $\mathsf{crs} \leftarrow \mathsf{CRS.Gen}(1^{\lambda})$, $(\mathsf{pp}, \mathsf{intl}_1, \mathsf{intl}_2) \leftarrow \mathsf{Comp}(1^{\lambda})$, and outputs $\mathsf{pp}' = (\mathsf{pp}||\mathsf{crs})$. Then the scheme invokes the two party protocol Π^{ρ} where the parties get the public parameter and their initial inputs, and execute the protocol. Recall that the protocol Π^{ρ} is the same as the hybrid scheme $\Pi^{\mathcal{F}}$ except whenever the parties call $\mathcal{F}$ in the hybrid scheme, in Π^{ρ} the parties run the protocol ρ instead.

Then we proceed to prove that this construction (replacement of $\mathcal{F}$) gives an OCL scheme that is ℓ-leakage-resilient with oblivious simulation. In particular, we need to construct a simulator $\mathcal{S}$ so that for all adversaries $\mathcal{A}$, its view in the real experiment $\mathbf{Real}_{\mathcal{A}}$ and the simulated view by $\mathcal{S}$ in the ideal experiment $\mathbf{Ideal}_{\mathcal{S},\mathcal{A}}$ are indistinguishable. We construct a universal simulator (for all adversaries) as follows:

The Simulator. Given any adversary $\mathcal{A}$, any circuit C_{λ}, the simulator $\mathcal{S}^{\mathcal{A}(1^{\lambda},|C_{\lambda}|,z),C_{\lambda}(\cdot)}$ gets oracle access to the adversary $\mathcal{A}$ and the circuit. Note that the simulator $\mathcal{S}$ does not have input C_{λ}. The simulator needs to produce an indistinguishable view of the adversary.

Initially, the simulator $\mathcal{S}$ runs $\mathsf{crs} \leftarrow \mathsf{CRS.Gen}(1^{\lambda})$, runs $\widetilde{\mathcal{S}}$ to generate a simulated $\widetilde{\mathsf{pp}}$, and gives $\widetilde{\mathsf{pp}}' = \widetilde{\mathsf{pp}}||\mathsf{crs}$ to the adversary $\mathcal{A}$. Later, at each round i, the simulator works as follows:

1. Let x_i be the input that $\mathcal{A}$ submits in this iteration, and $y_i = C_{\lambda}(x_i)$ be the answer obtained by the oracle query. The simulator $\mathcal{S}$ invokes $\widetilde{\mathcal{S}}(1^{\lambda}, i, x_i, y_i;\ \mathbf{w}_i)$ to obtain simulated states $(\mathsf{intl}_{i,1}, \mathsf{intl}_{i,2}, \mathsf{evl}_{i,1}, \mathsf{evl}_{i,2})$, where w_i is the fresh random coins tossed for the simulation in iteration i and $\mathbf{w}_i = w_1, \cdots, w_i$ is all the random coins that have been tossed for simulation in the first i iterations.

   Assume that n_i copies of $\mathcal{F}$ are used in this iteration. We note that the (evaluation) states provided by the simulator $\widetilde{\mathcal{S}}$ must include n_i inputs and outputs to the functionality (in the real $\mathcal{F}$-hybrid scheme, the inputs/outputs to $\mathcal{F}$ are included in the evaluation states, so $\widetilde{\mathcal{S}}$ must simulate these). Denote them as $\alpha^j_{i,1}$ (input for P_1), $\beta^j_{i,1}$ (output for P_1), $\alpha^j_{i,2}$ (input for P_2), and $\beta^j_{i,2}$ (output for P_2) for $j \in [n_i]$.

   Observe that the only missing piece here is the evaluation states of the protocols ρ's. Now the simulator $\mathcal{S}$ proceeds to emulate these states for the two parties that are consistent with the inputs and outputs. In particular, he invokes $\widehat{\mathcal{S}}^{\rho,j}_i = (\widehat{\mathcal{S}}^{\rho,j}_{i,\text{tr}}, \widehat{\mathcal{S}}^{\rho,j}_{i,1}, \widehat{\mathcal{S}}^{\rho,j}_{i,2})$ for all $j \in [n_i]$, where each $\widehat{\mathcal{S}}^{\rho,j}_i$ is an independent (and identical) copy of the simulator $\widehat{\mathcal{S}}^{\rho}$ (indexed by sup j and sub i), and he computes $\tau^j_i \leftarrow \widehat{\mathcal{S}}^{\rho,j}_{i,\text{tr}}(\mathsf{crs})$ and $r^j_{i,1} \leftarrow \widehat{\mathcal{S}}^{\rho,j}_{i,1}(\mathsf{crs}, \alpha^j_{i,1}, \beta^j_{i,1}, \tau^j_i)$, $r^j_{i,2} \leftarrow \widehat{\mathcal{S}}^{\rho,j}_{i,2}(\mathsf{crs}, \alpha^j_{i,2}, \beta^j_{i,2}, \tau^j_2)$. Note that for all $j \in [n_i]$, $(\alpha^j_{i,b}, \beta^j_{i,b})$ are included in $\mathsf{evl}_{i,b}$. For $b \in \{0, 1\}$, set $\widetilde{\mathsf{intl}}_{i,b} := \mathsf{intl}_{i,b}$ and $\widetilde{\mathsf{evl}}_{i,b} := \mathsf{evl}_{i,b}||\{r^j_{i,b}, \tau^j_i\}_{j\in[n_i]}$.

2. Let $(G_{1,b_1}, G_{2,b_2}, \ldots,)$ where $b_k \in \{1,2\}$ be the leakage queries $\mathcal{A}$ makes (perhaps in an adaptive way) in this round. Then $\mathcal{S}$ returns $G_{k,b_k}(\widetilde{\mathsf{intl}}_{i,b_k}, \widetilde{\mathsf{evl}}_{i,b_k})$ for all these queries, as long as the total amount of leakage on each component in this iteration is smaller than $\ell(\lambda)$ bits.

3. $\mathcal{S}$ sends y_i to the adversary.

Given such simulation strategy, a circuit C_λ, adversary $\mathcal{A}$, and auxiliary z, experiment $\mathbf{Ideal}_{\mathcal{S},\mathcal{A}}(1^\lambda, C_\lambda, z)$ is defined, and denote $\text{VIEW}^{\ell}_{\mathcal{S},\mathcal{A}}(1^\lambda, C_\lambda, z)$ as the (simulated) view of $\mathcal{A}$ in the experiment.

Next, we need to show the emulated view by $\mathcal{S}$ in the ideal experiment is indistinguishable from the real world view. To this goal, we start with the real experiment $\mathbf{Real}_{\mathcal{A}}$, and then we describe a sequence of hybrid experiments $\mathbf{Hyb}_{\mathcal{S}_0,\mathcal{A}}$, $\{\mathbf{Hyb}_{\mathcal{S}_{i,j},\mathcal{A}}\}_{i\in[m],j\in[n_i]}$, and show the views in all consecutive experiments are indistinguishable. We finally show the view in hybrid experiment $\mathbf{Hyb}_{\mathcal{S}_{m,n_m},\mathcal{A}}$ where m is the maximal (arbitrary polynomial) number of iterations in the execution, and n_m is the maximal number of copies of $\mathcal{F}$ are involved in the m^{th} round, and the view in the ideal experiment described above are indistinguishable. This means that the scheme $\Lambda' = (\mathsf{Comp}', \Pi^\rho)$ is ℓ leakage resilient with oblivious simulation.

Experiment $\mathbf{Real}_{\mathcal{A}}(1^\lambda, C_\lambda, z)$: The adversary $\mathcal{A}(1^\lambda, |C_\lambda|, z)$ proceeds as follows:

1. The initial states $(\mathsf{pp}', \mathsf{intl}'_1, \mathsf{intl}'_2) \leftarrow \mathsf{Comp}'(1^\lambda, C_\lambda)$ are sampled, where $\mathsf{crs} \leftarrow \mathsf{CRS.Gen}(1^\lambda)$, $(\mathsf{pp}, \mathsf{intl}_1, \mathsf{intl}_2) \leftarrow \mathsf{Comp}(1^\lambda, C_\lambda)$, and $\mathsf{pp}' = \mathsf{pp}||\mathsf{crs}$, $\mathsf{intl}'_b = \mathsf{intl}_b$ for $b \in [2]$.

2. $\mathcal{A}$ launches ℓ-bounded leakage attacks on an unbounded number of evaluations of its choice: In the i^{th} iteration,

   (a) $\mathcal{A}$ submits an input $x_i \in \{0,1\}^{|C_\lambda|}$, which is evaluated on C_λ by resuming the protocol execution of Π^ρ between the components $P_1(\mathsf{pp}', \mathsf{intl}'_{i,1}), P_2(\mathsf{pp}', \mathsf{intl}'_{i,2})$ with input x_i to the first component P_1.
   Assume there are n_i copies of ρ sub-routines in the i^{th} evalution. For $b \in \{1,2\}$ and $j \in \{1, \ldots, n_i\}$, let $\alpha^j_{i,b}$ be the input that component P_b sends to the j^{th} copy of ρ, and $\beta^j_{i,b}$ be the corresponding output, $r^j_{i,b}$ be the randomness used, and τ^j_i be the generated transcript between the two components. Note that $\{\alpha^j_{i,b}, \beta^j_{i,b}, r^j_{i,b}, \tau^j_i\}_{j\in[n_i]}$, as part of evaluation state, are included in $\mathsf{evl}'_{i,b}$.

   (b) $\mathcal{A}$ launches an ℓ-bounded leakage attack on the current iteration. It issues leakage queries $(G_{1,b_1}, G_{2,b_2}, \ldots,)$, where each $b_k \in \{1,2\}$ to the two components (adaptively), and obtain leakage answers of all internal state, i.e., $G_{k,b_k}(\mathsf{intl}'_{i,b_k}, \mathsf{evl}'_{i,b_k})$, as long as the total amount of leakage on each component in this iteration is smaller than $\ell(\lambda)$ bits. Denote $L_i \in \{0,1\}^{\le \ell}$ be the leakage observed in the i^{th} round.

   (c) $\mathcal{A}$ obtains the output of the evaluation, which is the output of P_2.

Denote $\text{VIEW}^{\ell}_{\mathcal{A}}(1^\lambda, C_\lambda, z) = (\mathsf{pp}', x_1, y_1, L_1, x_2, y_2, L_2, \ldots,)$ as the view of $\mathcal{A}$ in the above experiment.

Experiment $\mathbf{Hyb}_{\mathcal{S}_0,\mathcal{A}}(1^\lambda, C_\lambda, z)$: In the hybrid experiment, the simulator $\mathcal{S}_0^{\mathcal{A}(1^\lambda,|C_\lambda|,z)}(C_\lambda)$ gets input C_λ, and oracle access to the adversary $\mathcal{A}$. Let $\mathcal{A}(1^\lambda, |C_\lambda|, z)$ be the adversary who participates in the real experiment as described above. The simulator $\mathcal{S}_0$ simulates the whole real experiment for $\mathcal{A}$. Denote $\text{VIEW}^{\ell}_{\mathcal{S}_0,\mathcal{A}}(1^\lambda, C_\lambda, z)$ as the (simulated) view of $\mathcal{A}$ in the above experiment.

Since hybrid experiment $\mathbf{Hyb}_{\mathcal{S}_0,\mathcal{A}}$ is a reformulation of the real experiment, we immediately have:

Claim A.3 $\text{VIEW}^{\ell}_{\mathcal{A}}(1^\lambda, C_\lambda, z) \equiv \text{VIEW}^{\ell}_{\mathcal{S}_0,\mathcal{A}}(1^\lambda, C_\lambda, z)$.

Experiment $\mathbf{Hyb}_{\mathcal{S}_{i,j},\mathcal{A}}(1^\lambda, C_\lambda, z)$: In the hybrid experiment, the simulator $\mathcal{S}_{i,j}^{\mathcal{A}(1^\lambda,|C_\lambda|,z)}(C_\lambda)$ gets input C_λ, and oracle access to the adversary $\mathcal{A}$. Let the adversary $\mathcal{A}(1^\lambda, |C_\lambda|, z)$ participate in the same experiment as above. The simulator $\mathcal{S}_{i,j}$ is the same as $\mathcal{S}_0$ except that it generates the evaluation states slightly differently:

For all subroutines in the first $i-1$ iterations, and the first j subroutines in the i^{th} iteration, instead of generating $\{(\alpha^{\jmath}_{\imath,b}, \beta^{\jmath}_{\imath,b}, r^{\jmath}_{\imath,b}, \tau^{\jmath}_{\imath})\}_{\imath\in[i-1],\jmath\in[n_\imath],b\in\{1,2\}}$ and $\{(\alpha^{\jmath}_{\imath,b}, \beta^{\jmath}_{\imath,b}, r^{\jmath}_{\imath,b}, \tau^{\jmath}_{\imath})\}_{\imath\in[i-1],\jmath\in[j],b\in\{1,2\}}$ by running the corresponding copies of ρ, the simulator $\mathcal{S}_{i,j}$ operates as follows: for each subroutine, the simulator now provides the input $\alpha^{\jmath}_{\imath,b}$ to the corresponding copy of functionality $\mathcal{F}$, and learns the output $\beta^{\jmath}_{\imath,b}$; then the simulator computes $\tau^{\jmath}_{\imath} \leftarrow \widehat{\mathcal{S}}^{\rho,\jmath}_{\imath,\text{tr}}(\mathsf{crs})$ and $r^{\jmath}_{\imath,b} \leftarrow \widehat{\mathcal{S}}^{\rho,\jmath}_{\imath,b}(\mathsf{crs}, \alpha^{\jmath}_{\imath,b}, \beta^{\jmath}_{\imath,b}, \tau^{\jmath}_{\imath})$.

For the remaining subroutines in the execution, all $\{(\alpha^{\jmath}_{\imath,b}, \beta^{\jmath}_{\imath,b}, r^{\jmath}_{\imath,b}, \tau^{\jmath}_{\imath})\}$ are generated by running the corresponding copies of ρ.

Note that, in each iteration i for each $b \in \{0,1\}$, $\{\alpha^{\jmath}_{\imath,b}, \beta^{\jmath}_{\imath,b}, r^{\jmath}_{\imath,b}, \tau^{\jmath}_{\imath}\}_{\jmath\in[n_\imath]}$, are included in $\mathsf{evl}'_{\imath,b}$.

Denote $\text{VIEW}^{\ell}_{\mathcal{S}_{i,j},\mathcal{A}}(1^\lambda, C_\lambda, z)$ as the (simulated) view of $\mathcal{A}$ in the experiment.

Claim A.4 *For all $i \in \{1,\ldots,m\}$ and $j \in \{1,\ldots,n_i\}$, it holds that*

$$\text{VIEW}^{\ell}_{\mathcal{S}_{i,j-1},\mathcal{A}}(1^\lambda, C_\lambda, z) \approx \text{VIEW}^{\ell}_{\mathcal{S}_{i,j},\mathcal{A}}(1^\lambda, C_\lambda, z)$$

where m is the maximal (arbitrary polynomial) number of iterations in the execution, and n_i is the maximal number of copies of subroutines are involved in the i^{th} iteration.

Proof: Assume the claim does not hold. That means, there exist PPT adversary $\mathcal{A}$ with auxiliary input z, and circuit C_λ, such that the views generated in hybrid experiments $\mathbf{Hyb}_{\mathcal{S}_{i,j-1},\mathcal{A}}$ and $\mathbf{Hyb}_{\mathcal{S}_{i,j},\mathcal{A}}$ are distinguishable with non-negligible probability. We now construct a PPT machine $\mathcal{D}$ which has $\mathcal{A}$, z, and C_λ hardwired, to break the security of the subroutine ρ.

Basically, $\mathcal{D}$ simulates the experiment $\mathbf{Hyb}_{\mathcal{S}_{i,j-1},\mathcal{A}}$ up to the i-th iteration, and the first $j-1$ calls of the subroutine ρ (right before the j-th call). Now $\mathcal{D}$ has the inputs for the two parties, and $\mathcal{D}$ can ask the challenger to obtain the states either by running ρ, or by the subroutine simulator $\mathcal{S}^\rho$. Then $\mathcal{D}$ finishes the simulation of the rest of $\mathbf{Hyb}_{\mathcal{S}_{i,j-1},\mathcal{A}}$. The challenge distributions will correspond to either $\mathbf{Hyb}_{\mathcal{S}_{i,j-1},\mathcal{A}}$ or $\mathbf{Hyb}_{\mathcal{S}_{i,j},\mathcal{A}}$. Since these two distributions are distinguishable, $\mathcal{D}$ can break the security of ρ. More formally, we describe $\mathcal{D}$'s strategy as follows:

First compute $\mathsf{crs} \leftarrow \mathsf{CRS.Gen}(1^\lambda)$. $\mathcal{D}(\mathsf{crs})$ proceeds as follows:

1. Generate the initial states $(\mathsf{pp}', \mathsf{intl}'_1, \mathsf{intl}'_2) \leftarrow \mathsf{Comp}'(1^\lambda, C_\lambda)$ where $(\mathsf{pp}, \mathsf{intl}_1, \mathsf{intl}_2) \leftarrow \mathsf{Comp}(1^\lambda, C_\lambda)$, and $\mathsf{pp}' = \mathsf{pp}||\mathsf{crs}$, $\mathsf{intl}'_b = \mathsf{intl}_b$ for $b \in \{1,2\}$.

2. $\mathcal{D}$ simulates the hybrid experiment: $\mathcal{A}$ launches ℓ-bounded leakage attacks on an unbounded number of evaluations of its choice: In the $\imath^{\text{th}}$ iteration,

   (a) $\mathcal{A}$ submits an input $x_\imath \in \{0,1\}^{|C_\lambda|}$, which is evaluated on C_λ by resuming the protocol execution of Π^ρ between the components $P_1(\mathsf{pp}', \mathsf{intl}'_{\imath,1}), P_2(\mathsf{pp}', \mathsf{intl}'_{\imath,2})$ with input $x_\imath$ to the first component P_1.

   - If $\imath < i$, generate $\{(\alpha^{\jmath}_{\imath,b}, \beta^{\jmath}_{\imath,b}, r^{\jmath}_{\imath,b}, \tau^{\jmath}_{\imath})\}_{\jmath\in[n_\imath],b\in[2]}$ as follows: for each subroutine in this iteration, provide the input $\alpha^{\jmath}_{\imath,b}$ to the corresponding copy of functionality $\mathcal{F}$, and learn the output $\beta^{\jmath}_{\imath,b}$; then compute $\tau^{\jmath}_{\imath} \leftarrow \widehat{\mathcal{S}}^{\rho,\jmath}_{\imath,\text{tr}}(\mathsf{crs})$ and $r^{\jmath}_{\imath,b} \leftarrow \widehat{\mathcal{S}}^{\rho,\jmath}_{\imath,b}(\mathsf{crs}, \alpha^{\jmath}_{\imath,b}, \beta^{\jmath}_{\imath,b}, \tau^{\jmath}_{\imath})$. Finally, $\{\alpha^{\jmath}_{\imath,b}, \beta^{\jmath}_{\imath,b}, r^{\jmath}_{\imath,b}, \tau^{\jmath}_{\imath}\}_{\jmath\in[n_\imath]}$, as part of evaluation state, are included in $\mathsf{evl}'_{\imath,b}$ for $b \in \{1,2\}$.
   - If $\imath = i$, generate $\{(\alpha^{\jmath}_{\imath,b}, \beta^{\jmath}_{\imath,b}, r^{\jmath}_{\imath,b}, \tau^{\jmath}_{\imath})\}_{\jmath\in[n_\imath],b\in\{1,2\}}$ as follows:
     - For $\jmath \in \{1,\ldots,j-1\}$:
       for $b \in \{1,2\}$, provide the input $\alpha^{\jmath}_{\imath,b}$ to the corresponding copy of functionality $\mathcal{F}$, and learn the output $\beta^{\jmath}_{\imath,b}$; then compute $\tau^{\jmath}_{\imath} \leftarrow \widehat{\mathcal{S}}^{\rho,\jmath}_{\imath,\text{tr}}(\mathsf{crs})$ and $r^{\jmath}_{\imath,b} \leftarrow \widehat{\mathcal{S}}^{\rho,\jmath}_{\imath,b}(\mathsf{crs}, \alpha^{\jmath}_{\imath,b}, \beta^{\jmath}_{\imath,b}, \tau^{\jmath}_{\imath})$.

  - For $\jmath = j$:
    send $(\alpha^{\jmath}_{\imath,1}, \alpha^{\jmath}_{\imath,2})$ to the challenger, and the challenger returns (r_1^*, τ^*, r_2^*). Then $\mathcal{D}$ sets $r^{\jmath}_{\imath,1} := r_1^*$, $r^{\jmath}_{\imath,2} := r_2^*$, and $\tau^{\jmath}_{\imath} := \tau^*$. Note that $(\beta^{\jmath}_{\imath,1}, \beta^{\jmath}_{\imath,2})$ can be easily derived.
  - For $\jmath \in \{j+1, \ldots, n_\imath\}$:
    for $b \in \{1,2\}$, provide the input $\alpha^{\jmath}_{\imath,b}$ to the corresponding copy of ρ, and learn the output $\beta^{\jmath}_{\imath,b}$; here $\tau^{\jmath}_{\imath} = \langle P_1^{\rho}(\mathsf{crs}, \alpha^{\jmath}_{\imath,1}; r^{\jmath}_{\imath,1}), P_2^{\rho}(\mathsf{crs}, \alpha^{\jmath}_{\imath,2}; r^{\jmath}_{\imath,2})\rangle$, and $r^{\jmath}_{\imath,1}, r^{\jmath}_{\imath,2}$ are the random coins used by the two players in ρ.

  Finally, $\{\alpha^{\jmath}_{\imath,b}, \beta^{\jmath}_{\imath,b}, r^{\jmath}_{\imath,b}, \tau^{\jmath}_{\imath}\}_{\jmath \in [n_\imath]}$, as part of evaluation state, are included in $\mathsf{evl}'_{\imath,b}$ for $b \in \{1,2\}$.

- If $\imath > i$, generate $\{(\alpha^{\jmath}_{\imath,b}, \beta^{\jmath}_{\imath,b}, r^{\jmath}_{\imath,b}, \tau^{\jmath}_{\imath})\}_{\jmath \in [n_\imath], b \in [2]}$ as follows: for each subroutine in this iteration, provide the input $\alpha^{\jmath}_{\imath,b}$ to the corresponding copy of ρ, and learn the output $\beta^{\jmath}_{\imath,b}$; here $\tau^{\jmath}_{\imath} = \langle P_1^{\rho}(\mathsf{crs}, \alpha^{\jmath}_{\imath,1}; r^{\jmath}_{\imath,1}), P_2^{\rho}(\mathsf{crs}, \alpha^{\jmath}_{\imath,2}; r^{\jmath}_{\imath,2})\rangle$, and $r^{\jmath}_{\imath,1}, r^{\jmath}_{\imath,2}$ are the random coins used by the two players in ρ. Finally, $\{\alpha^{\jmath}_{\imath,b}, \beta^{\jmath}_{\imath,b}, r^{\jmath}_{\imath,b}, \tau^{\jmath}_{\imath}\}_{\jmath \in [n_\imath]}$, as part of evaluation state, are included in $\mathsf{evl}'_{\imath,b}$ for $b \in \{1,2\}$.

(b) $\mathcal{A}$ launches an ℓ-bounded leakage attack on the current iteration. It issues leakage queries $(G_{1,b_1}, G_{2,b_2}, \ldots,)$, where each $b_k \in \{1,2\}$ to the two components (adaptively), and obtain leakage answers of all internal state, i.e., $G_{k,b_k}(\mathsf{intl}'_{i,b_k}, \mathsf{evl}'_{i,b_k})$, as long as the total amount of leakage on each component in this iteration is smaller than $\ell(\lambda)$ bits. Denote $L_i \in \{0,1\}^{\leq \ell}$ be the leakage observed in the $\imath$th round.

(c) $\mathcal{A}$ obtains the output of the evaluation, which is the output of P_2.

Upon receiving $(\alpha^{\jmath}_{\imath,1}, \alpha^{\jmath}_{\imath,2})$, if $\mathcal{D}$'s challenger generates (r_1^*, τ^*, r_2^*) by running a copy of ρ, then the view of $\mathcal{A}$ is identical to that in the hybrid experiment $\mathbf{Hyb}_{\mathcal{S}_{i,j-1},\mathcal{A}}(1^\lambda, C_\lambda, z)$. On the other hand, if the challenges are generated by computing $\tau^* \leftarrow \widehat{\mathcal{S}}^{\rho}_{\mathrm{tr}}(\mathsf{crs})$ and $r_b^* \leftarrow \widehat{\mathcal{S}}^{\rho}_b(\mathsf{crs}, \alpha^{\jmath}_{\imath,b}, \beta^{\jmath}_{\imath,b}, \tau^*)$ for $b \in \{1,2\}$, then the view of $\mathcal{A}$ is identical to that in the hybrid experiment $\mathbf{Hyb}_{\mathcal{S}_{i,j},\mathcal{A}}(1^\lambda, C_\lambda, z)$. Since $\mathcal{A}$ can distinguish the views generated in two hybrid experiments with non- negligible probability, $\mathcal{D}$ can distinguish the internal state from the simulated internal state of the subroutine with non negligible probability, which break the security of ρ protocol. ■

Claim A.5 *For all $i \in \{1, \ldots, m\}$, it holds that*

$$\text{VIEW}^{\ell}_{\mathcal{S}_{i+1,1},\mathcal{A}}(1^\lambda, C_\lambda, z) \approx \text{VIEW}^{\ell}_{\mathcal{S}_{i,n_i},\mathcal{A}}(1^\lambda, C_\lambda, z)$$

where m is the maximal (arbitrary polynomial) number of iterations in the execution.

The proof is exactly the same as that for Claim A.4.

Claim A.6 $\text{VIEW}^{\ell}_{\mathcal{S}_{m,n_m},\mathcal{A}}(1^\lambda, C_\lambda, z) \approx \text{VIEW}^{\ell}_{\mathcal{S},\mathcal{A}}(1^\lambda, C_\lambda, z)$*, where m is the maximal number of iterations in the execution, and n_m is the number of subroutines in the mth iteration.*

Proof: Given that $\Lambda^{\mathcal{F}} = (\mathsf{Comp}, \Pi^{\mathcal{F}})$ is ℓ-leakage-resilient with oblivious simulation, we have a simulator $\widetilde{\mathcal{S}}$, for all PPT adversary $\widetilde{\mathcal{A}}$ so that for auxiliary z and circuit C_λ, the simulated view by $\widetilde{\mathcal{S}}$ in the ideal experiment $\mathbf{Ideal}_{\widetilde{\mathcal{S}},\widetilde{\mathcal{A}}}(1^\lambda, C_\lambda, z)$ is indistinguishable from the view of adversary $\widetilde{\mathcal{A}}$ in the real experiment $\mathbf{Real}_{\widetilde{\mathcal{A}}}(1^\lambda, C_\lambda, z)$.

Recall that $\mathcal{S}_{m,n_m}$ basically runs the protocol $\Pi^{\mathcal{F}}$ and uses the protocol simulator $\widehat{\mathcal{S}}_\rho$ to simulate the internal states for the $\mathcal{F}$ subroutine calls; the simulator $\mathcal{S}$ in the ideal world uses $\widetilde{\mathcal{S}}$ to simulate the protocol $\Pi^{\mathcal{F}}$ and the protocol simulator $\widehat{\mathcal{S}}_\rho$ to simulate the internal states for the $\mathcal{F}$ subroutine calls. By the security of $\Lambda^{\mathcal{F}}$, these two experiments are indistinguishable.

Formally, we argue: suppose there exists an adversary $\mathcal{A}$, circuit C_λ, and z such that the views $\text{VIEW}^{\ell}_{\mathcal{S}_{m,nm},\mathcal{A}}(1^\lambda, C_\lambda, z)$ and $\text{VIEW}^{\ell}_{\mathcal{S},\mathcal{A}}(1^\lambda, C_\lambda, z)$ are distinguishable with non-negligible probability. Then we can construct an adversary $\widetilde{\mathcal{A}}$ such that $\mathbf{Ideal}_{\widetilde{\mathcal{S}},\widetilde{\mathcal{A}}}(1^\lambda, C_\lambda, z)$ is distinguishable from $\mathbf{Real}_{\widetilde{\mathcal{A}}}(1^\lambda, C_\lambda, z)$. This contradicts the fact that $\Lambda^{\mathcal{F}}$ is a secure OCL scheme.

Given an adversary $\mathcal{A}$, we define $\widetilde{\mathcal{A}}$ as follows:

- $\widetilde{\mathcal{A}}$ samples $\mathsf{crs} \leftarrow \mathsf{CRS.Gen}(1^\lambda)$. Then it receives input a public parameter pp from the experiment.
- In each iteration i, $\widetilde{\mathcal{A}}$ runs $\mathcal{A}$ $\big($with the public parameter $\mathsf{pp}' = (\mathsf{crs}||\mathsf{pp})\big)$ and forwards the input x_i from the $\mathcal{A}$.
- Then $\widetilde{\mathcal{A}}$ runs $\mathcal{A}$ and obtains leakage queries $(G_{1,b_1}, G_{2,b_2}, \ldots,)$, where each $b_k \in \{1,2\}$. $\widetilde{\mathcal{A}}$ answers these queries by querying the experiments with $(\widetilde{G}_{1,b_1}, \widetilde{G}_{2,b_2}, \ldots,)$, where $\widetilde{G}_{k,b_k}(\mathsf{intl}_{i,b_k}, \mathsf{evl}_{i,b_k})$ does the following:
    1. first computes $\mathsf{evl}'_{i,b_k} = \mathsf{evl}_{i,b_k}||\{r^j_{i,b_k}, \tau^j_i\}_{j\in[n_i]}$ by using $\widehat{\mathcal{S}}^{\rho}_{\widetilde{r}}, \widehat{\mathcal{S}}^{\rho}_{b_k}$ as the simulator $\mathcal{S}$. Note that the evaluation state of each component $\mathsf{evl}_{i,b}$ includes $\{\alpha^j_{i,b}, \beta^j_{i,b}\}_{j\in[n_i]}$; here n_i is the number of copies of $\mathcal{F}$ are used in the i^{th} iteration, and $\alpha^j_{i,b}$ is the input provided by P_b to the j^{th} copy of $\mathcal{F}$ and $\beta^j_{i,b}$ is the corresponding output.
    2. Then the function outputs $G_{k,b_k}(\mathsf{intl}_{i,b_k}, \mathsf{evl}'_{i,b_k})$.
- $\widetilde{A}$ on input y_i sends this input to $\mathcal{A}$.

From the definition of $\widetilde{\mathcal{A}}$, it is obvious that the view $\mathbf{Real}_{\widetilde{\mathcal{A}}}(1^\lambda, C_\lambda, z)$ is identical to that of $\text{VIEW}^{\ell}_{\mathcal{S}_{m,nm},\mathcal{A}}(1^\lambda, C_\lambda, z)$ and the view $\mathbf{Ideal}_{\widetilde{\mathcal{S}},\widetilde{\mathcal{A}}}(1^\lambda, C_\lambda, z)$ is identical to that of $\text{VIEW}^{\ell}_{\mathcal{S},\mathcal{A}}(1^\lambda, C_\lambda, z)$. Thus, suppose one can distinguish $\text{VIEW}^{\ell}_{\mathcal{S}_{m,nm},\mathcal{A}}(1^\lambda, C_\lambda, z)$ from $\mathbf{Ideal}_{\widetilde{\mathcal{S}},\widetilde{\mathcal{A}}}(1^\lambda, C_\lambda, z)$ with non-negligible probability, then $\mathbf{Real}_{\widetilde{\mathcal{A}}}(1^\lambda, C_\lambda, z)$ and $\mathbf{Ideal}_{\widetilde{\mathcal{S}},\widetilde{\mathcal{A}}}(1^\lambda, C_\lambda, z)$ are distinguishable. This completes the proof of the claim.

The proof of the theorem follows directly from Claims A.3, A.4, A.5, A.6.

■

B Supplement for Section 4: the JV and DF Schemes

In this section, we describe two different implementations of OCL schemes, based on the Juma-Vahlis [34] and Dziembowski-Faust [21] OCL schemes.

B.1 The JV OCL scheme.

The idea behind the basic JV scheme is simple. The scheme is based on a fully homomorphic encryption scheme $\mathsf{FHE} = \{\mathsf{Gen}, \mathsf{Enc}, \mathsf{Dec}, \mathsf{Eval}\}$. At the initial compilation phase, the state of the left component P_1 is set to be an encryption $\mathsf{Enc}_{\mathsf{pk}}(C)$ of the input circuit C. The state of the right component P_2 is set to be the corresponding decryption key sk. In evaluation, when P_1 obtains the input x, it homomorphically evaluates C on x, and sends the result $\mathsf{Enc}_{\mathsf{pk}}(C(x))$ to P_2 over the secure channel. P_2 then decrypts and obtains the output. To ensure that the scheme yields an OCL, we add an additional cipher refresh step; concretely, we homomorphically add to $\mathsf{Enc}(C(x))$ an encryption $\mathsf{Enc}_{\mathsf{pk}}(0)$, and add to $\mathsf{Enc}_{\mathsf{pk}}(C)$ another encryption $\mathsf{Enc}_{\mathsf{pk}}(0)$.

The full scheme is given in Figure 3 below. $\mathcal{F}_{\text{sampling}}$ has been described in Section 4.1.

OCL scheme $\mathsf{OCL_{JV}} = (\mathsf{Comp}, P_1, P_2)$

Initial Compilation by Comp**:**

- Given input C, compute the following:
  - $(\mathsf{sk}_0, \mathsf{pk}_0) \leftarrow \mathsf{Gen}(1^\lambda)$, $\mathsf{ct}_{0,C} \leftarrow \mathsf{Enc}_{\mathsf{pk}_0}(C)$,
- Return the following as output:
  - $\mathsf{intl}_1 = \mathsf{intl}_{0,1} = (\mathsf{pk}_0, \mathsf{ct}_{0,C})$,
  - $\mathsf{intl}_2 = \mathsf{intl}_{0,2} = \mathsf{sk}_0$,
  - $\mathsf{pp} = \emptyset$.

i**-th Evaluation by** (P_1, P_2)**:**

- On input x (for clarity of prevention, we omit the subscript i of the input),
  - P_1 obtains $\gamma_{i,1} = (\mathsf{pk}_i, \mathsf{ct}_{i,0}, \mathsf{ct}'_{i,0})$ from $\mathcal{F}_{\text{sampling}}$.
  - P_2 obtains $\gamma_{i,2} = (\mathsf{sk}_i, \mathsf{pk}_i)$ from $\mathcal{F}_{\text{sampling}}$,
- First, P_2 does the following:
  - compute $\mathsf{ct}_{i,\mathsf{sk}_{i-1}} \leftarrow \mathsf{Enc}_{\mathsf{pk}_i}(\mathsf{sk}_{i-1}; r_i)$, where r_i is randomly chosen,
  - send $\mathsf{ct}_{i,\mathsf{sk}_{i-1}}$ to P_1.
- Then P_1, based on his input x, does the following:
  - compute $\mathsf{ct}_{i,C(x)} \leftarrow \mathsf{Eval}_{\mathsf{pk}_i}(\mathsf{ct}_{i,\mathsf{sk}_{i-1}}, x, \mathsf{ct}_{i-1,C})$,
  - compute $\tilde{\mathsf{ct}}_{i,C(x)} \leftarrow \mathsf{ct}_{i,C(x)} + \mathsf{ct}_{i,0}$, and $\mathsf{ct}_{i,C} \leftarrow \mathsf{ct}_{i-1,C} + \mathsf{ct}'_{i,0}$, where "+" means homomorphic addition).
  - send $\tilde{\mathsf{ct}}_{i,C(x)}$ to P_2.
  - update state to $\mathsf{intl}_{i,1} = (\mathsf{pk}_i, \mathsf{ct}_{i,C})$.
- Now P_2 does the following:
  - decrypt $\tilde{\mathsf{ct}}_{i,C(x)}$ by using sk_i and output $y = C(x)$.
  - update state to $\mathsf{intl}_{i,2} = \mathsf{sk}_i$.

Figure 3: The JV OCL scheme

B.2 The DF OCL scheme.

At high level, the DF scheme follows the classic GMW [27] paradigm for semi-honest two-party computation. At the onset of the computation, the parties share each bit C_i of the private circuit C; Then, when P_1 obtains the input x, it shares each bit of x with P_2. The parties then homomorphically compute the boolean circuit $U(\cdot,\cdot)$ on the underlying input (C, x). To guarantee leakage-resilience, the DF scheme relies on a leakage-resilient secret sharing scheme—*the inner product two-source extractor*.

We adapt the simplified version of the DF scheme presented in the work [9]. The full scheme is given in Figure 4 below, where $U(\cdot,\cdot)$ denotes the universal circuit, and is assumed to have T NAND gates labeled $G = \{1, \ldots, T\}$, among them n input gates labeled $I = \{1, \ldots, n\}$, and t output gates labeled $O = \{T - t + 1, \ldots, T\}$. We denote by $\mathcal{O}_b^m$ the distribution on vector pairs in $\mathbb{F}_2^m \times \mathbb{F}_2^m$ whose inner product is b. $\mathcal{F}_{\text{sampling}}$ has been described in Section 4.2.

OCL scheme $\mathsf{OCL}_{\mathtt{DF}} = (\mathsf{Comp}, P_1, P_2)$

Initial Compilation by Comp**:**

- For each bit C_j of the circuit C, sample $(L_{C_j}, R_{C_j}) \leftarrow \mathcal{O}^m_{C_j}$,

The initial state:

- $\mathsf{intl}_1 = \mathsf{intl}_{0,1} = \{L_{0,C_j}\}_{j\in[|C|]}$
- $\mathsf{intl}_2 = \mathsf{intl}_{0,2} = \{R_{0,C_j}\}_{j\in[|C|]}$
- $\mathsf{pp} = \emptyset$

***i*-th Evaluation by** (P_1, P_2)**:**

- On input x (for clarity of prevention, we omit the subscript i of the input), P_1 and P_2 obtain $\gamma_{i,1}$ and $\gamma_{i,2}$ from $\mathcal{F}_{\text{sampling}}$ respectively, where $\gamma_{i,1} = \big(\{L'_{i,g}||L''_{i,g}\}_{g\in G}, \{A'_{i,j}||A''_{i,j}\}_{j\in|C|}\big)$ and $\gamma_{i,2} = \big(\{R'_{i,g}||R''_{i,g}\}_{g\in G}, \{B'_{i,j}||B''_{i,j}\}_{j\in|C|}\big)$.
- P_1, based on his input x, does the following:
  for $j \in [|x|]$, computes $(L_{x_j}, R_{x_j}) \leftarrow \mathcal{O}^m_{x_j}$, and sends $\{R_{x_j}\}_{j\in[|x|]}$ to P_2.
- For every gate $g \in G$, with input wires $u = u(g), v = v(g)$ and output wire $w = w(g)$, and input shares (L_u, L_v, R_u, R_v), each in $\mathbb{F}_2^m$, the parties run a homomorphic NAND procedure (see Figure 6):

$$(L_w, R_w) \leftarrow \mathsf{NAND}(L_u, L_v, L'_{i,g}, L''_{i,g}; R_u, R_v, R'_{i,g}, R''_{i,g}) \ .$$

- For each output wire $g \in O$, the players P_1 and P_2 receive the shares $L_{w(g)}$ and $R_{w(g)}$ respectively.
- P_1 sends $\{L_{w(g)}\}_{g\in O}$ to P_2, and then P_2 outputs $\{\langle L_{w(g)}, R_{w(g)}\rangle\}_{g\in O}$.
- For $j \in |C|$, P_1 and P_2 run a Refresh procedure (see Figure 5):

$$(L_{i,C_j}, R_{i,C_j}) \leftarrow \mathsf{Refresh}(L_{i-1,C_j}, A'_{i,j}, A''_{i,j}; R_{i-1,C_j}, B'_{i,j}, B''_{i,j})$$

  P_1 updates $\mathsf{intl}_{i,1} := \{L_{i,C_j}\}_{j\in[|C|]}$.
  P_2 updates $\mathsf{intl}_{i,2} := \{R_{i,C_j}\}_{j\in[|C|]}$.

Figure 4: The DF OCL scheme

Procedure Refresh

Procedure $(L^\star, R^\star) \leftarrow \mathsf{Refresh}(L, A', A''; R, B', B'')$**:**

- P_1 samples a random matrix $M' \in \mathbb{F}_2^{m\times m}$ such that $M'L = A'$, and sends it to P_2
- P_2 sets $R^\star := R + (M')^{\mathrm{T}}B'$.
- P_2 samples a random matrix $M'' \in \mathbb{F}_2^{m\times m}$ such that $M''R^\star = B''$, and sends it to P_1
- P_1 sets $L^\star := L + (M'')^{\mathrm{T}}A''$.
- If any of the above is impossible due to zero shares, the parties abort.

Figure 5: Procedure Refresh that used in the DF scheme in Figure 4.

Procedure NAND

Procedure $(L_w, R_w) \leftarrow \mathsf{NAND}(L_u, L_v, L', L''; R_u, R_v, R', R'')$**:**

- Each party locally computes $L_w^{\otimes} := L_u \otimes L_v \in \mathbb{F}_2^{m^2}$, $R_w^{\otimes} := R_u \otimes R_v \in \mathbb{F}_2^{m^2}$.
- The parties then run a $\mathsf{Refresh}^{\otimes}$ procedure (see Figure 7):
$$(L_w^{\otimes,\star}, R_w^{\otimes,\star}) \leftarrow \mathsf{Refresh}^{\otimes}(L_w^{\otimes}, L', L''; R_w^{\otimes}, R', R'') \ .$$
- P_2 sends $R_w^{\otimes,\star}[m,\ldots,m^2]$ to P_1, who then computes $b := \langle L_w^{\otimes,\star}[m,\ldots,m^2], R_w^{\otimes,\star}[m,\ldots,m^2]\rangle$, and sets
$$L_w := (1 + b | L_w^{\otimes,\star}[1,\ldots,m-1]) \ .$$
- P_2 sets his own share to be
$$R_w := (1 | R_w^{\otimes,\star}[1,\ldots,m-1]) \ .$$

Figure 6: Procedure NAND that used in the DF scheme in Figure 4. Here $X[m,\ldots,m^2]$ denotes the last $m^2 - m + 1$ entries of X, and $X[1,\ldots,m-1]$ the first $m-1$ entries of X.

Procedure $\mathsf{Refresh}^{\otimes}$

Procedure $(L^{\otimes,\star}, R^{\otimes,\star}) \leftarrow \mathsf{Refresh}^{\otimes}(L^{\otimes}, L', L''; R^{\otimes}, R', R'')$**:**

- P_1 samples a random matrix $M' \in \mathbb{F}_2^{m^2 \times m^2}$ such that $M'L^{\otimes} = L'$, and sends it to P_2
- P_2 sets $R^{\otimes,\star} := R^{\otimes} + (M')^{\mathrm{T}} R'$.
- P_2 samples a random matrix $M'' \in \mathbb{F}_2^{m^2 \times m^2}$ such that $M''R^{\otimes,\star} = R''$, and sends it to P_1
- P_1 sets $L^{\otimes,\star} := L^{\otimes} + (M'')^{\mathrm{T}} L''$.
- If any of the above is impossible due to zero shares, the parties abort.

Figure 7: Procedure $\mathsf{Refresh}^{\otimes}$ that used in Figure 6.