# Secure Multiparty Computation with Identifiable Abort from Vindicating Release

Ran Cohen
cohenran@runi.ac.il
Reichman University

Jack Doerner
j@ckdoerner.net
Technion

Yashvanth Kondi
yash@ykondi.net
Aarhus University

abhi shelat
abhi@neu.edu
Northeastern University

July 22, 2023

**Abstract**

In the dishonest-majority setting, generic secure multiparty computation (MPC) protocols are fundamentally vulnerable to attacks in which malicious participants learn their outputs and then force the protocol to abort before outputs are delivered to the honest participants. In other words, generic MPC protocols typically guarantee *security with abort*.

This flavor of security permits denial-of-service attacks in many applications, unless the cheating participants who cause aborts are identified. At present, there is a substantial performance gap between the best known protocols that are secure with non-identifiable abort, and the best known protocols that achieve *security with identifiable abort* (IA). Known constructions with IA rely on generic zero-knowledge proofs, adaptively secure oblivious transfer (OT) protocols, or homomorphic primitives.

We present a novel approach for realizing functionalities with a weak form of *input-revealing* IA, which is based on delicate and selective revealing of committed input values. We refer to this new approach as *vindicating release*. When our approach is applied to several well-known protocols—including a variant of PVW OT, Softspoken OT extension, DKLs multiplication, and MASCOT generic MPC—the resulting protocols can be combined to realize any *sampling* functionality with (standard) IA. Such a realization is statistically secure given a variant of statically-corruptable ideal OT, and it differs minimally in terms of cost, techniques, and analysis from the equivalent realization (using the same well-known protocols, unmodified) that lacks identifiability.

Using our protocol to sample the correlated randomness of the IOZ compiler reduces the compiler's requirements from an adaptively secure OT protocol to a variant of statically-corruptable ideal OT.

# Contents

# 1 Introduction

When a majority of participants in a secure multiparty computation protocol behave dishonestly, it is not generally possible to guarantee the delivery of output to the honest parties [Cle86]; consequently, the standard definition of security permits the adversary to halt the protocol after first learning the outputs of any dishonest parties. Correctness and privacy are still guaranteed for honest paries, and in some cases it is also guaranteed that the honest parties agree on whether or not an abort has occurred. If they are guaranteed to agree, then the protocol is said to achieve *security with unanimous abort* and otherwise it achieves *security with selective abort*. This security regime has been well-studied beginning with classic feasibility results [Yao82, GMW87, BMR90] and recently many asymptotic improvements [IPS08, DPSZ12, BGIN22] culminating in several concretely efficient protocols [HSS17, WRK17, YWZ20].

Unfortunately, protocols that are secure with abort are exposed to *denial-of-service* (DoS) attacks. Even a single corrupted party can cause the protocol to abort repeatedly, and the honest parties have no recourse, since the corrupted party cannot be identified. Such attacks are tolerable in some specific use-cases [FLNW17], but in many they are not. As a key example, while DoS attacks can be tolerated in small, closed systems where investigation can be performed out-of-band, they *cannot* be tolerated in large-scale and/or permissionless MPC applications, such as the distributed sampling of *structured reference strings* (SRSes) for public consumption, or threshold signing for blockchains. Among the most commonly used SRSes are RSA moduli [BF01, HMRT12, CCD+20, CHI+21, dMRT21, BDF+23] and "the powers of $\tau$" [BCG+15, BGM17, GKM+18, KMSV21, CDKs22], and indeed, works on the distributed sampling of these objects and on threshold signatures [CGG+20, CCL+23] aim to prevent DoS attacks by achieving the stronger notion of *identifiable abort* (IA), which guarantees that in the case of an abort, all honest parties agree upon the identity of at least one corrupted party.

The identifiable abort (IA) notion of security is less explored than the standard security with abort, although it is just as old: feasibility for both notions was shown by Goldreich, Micali, and Widgerson [GMW87]. The GMW compiler transforms any protocol that is secure against semi-honest adversaries into a protocol that is secure with IA against malicious adversaries. At a high level, the parties commit to their inputs and then run an augmented coin-tossing protocol to sample a committed random tape for each party, after which they run the original protocol over a broadcast channel, but include with each message a zero-knowledge proof that the message has been computed correctly. This approach is simple, but inefficient due to its non-black-box usage of cryptographic primitives.

A line of follow-up work demonstrated that in the dishonest-majority setting, MPC with *unanimous* abort can withstand computationally unbounded adversaries (i.e., it can make no cryptographic assumptions) if given access to an ideal *oblivious transfer* (OT) functionality [Kil88, IPS08]. Furthermore the OT oracle need not be accessed *online*; instead it is sufficient to access it *of-*

*fline*, before the parties know their inputs [Bea95]. On the other hand, Ishai, Ostrovsky, and Seyalioglu [IOS12] ruled out information-theoretic MPC with *identifiable* abort given *any* pairwise correlation. This implies that IA is separated from any ideal oracle that only interfaces with two parties, even if that oracle has guaranteed output.

Ishai, Ostrovsky, and Zikas [IOZ14] overcame this barrier by constructing a complex $n$-wise correlation that enables information-theoretic MPC with IA. Specifically, they presented a compiler like the one of GMW, which uses information-theoretic signatures for commitments, along with a distributed version of the IKOS zero-knowledge proof [IKOS07] that is information-theoretically secure given their correlation. They also present a protocol for the distributed sampling of their correlation that makes black-box use of an adaptively secure OT *protocol*,[1] and ideal commitments.

Successive works have abandoned the compiler-oriented approach and combined various $n$-wise correlations with specific online protocols to achieve better efficiency. Baum et al. [BOS16] presented a variant of the BDOZ protocol [BDOZ11] that uses homomorphic information-theoretic signatures, and a correlation-sampling protocol based upon somewhat homomorphic encryption; this approach avoids zero-knowledge proofs in the online phase. Spini and Fehr [SF16] and Cunningham et al. [CFY17] adjusted the SPDZ protocol [DPSZ12] to achieving IA. Baum et al. [BOSS20] focused on boolean (rather than arithmetic) circuits and relaxed the problem to allow computational assumptions in the online phase. Their approach is based upon multi-party garbling [BMR90, HSS17], and makes black-box use of a *statically* secure OT protocol, a pseudorandom function (PRF), and additively homomorphic commitments. Their approach also opens up a number of well-known concrete optimizations such at OT extension and free-XOR.

## 1.1 Identifying Cheaters by Revealing One's Inputs

One can view any information-theoretic online phase that requires input-independent correlated randomness (e.g. those of Ishai et al. [IOZ14] or Baum et al. [BOS16]) as reducing the problem of achieving IA for general tasks to the problem of achieving IA for *distributed sampling*. As we have argued above, distributed sampling with identifiable abort is also important for sampling SRSes. Our work focuses on this problem.

In distributed sampling protocols, the only private inputs of the participants are their random coins; in the case of an abort, these coins can safely be revealed, since neither they nor any output derived from them are used by the honest parties. This fact leads to a natural and seemingly simple approach: the parties simply commit to their randomness, and, in the case of an abort, reveal it. By using the opened commitments to emulate the protocol they could identify (and agree upon) the first player to cheat. This approach would

[1] When we say black-box use of a protocol, we mean access to an oracle computing the protocol's next-message function. This usage was introduced by Ishai et al. [IKOS07]

eliminate the need for zero-knowledge statements concerning the output of the next-message functionality, and could thus be substantially more efficient and simpler to implement.

This approach, though intuitive, is difficult to simulate: if the adversary causes an abort *after* learning the outputs, then the simulator must produce a view for each of the honest parties that is consistent with both the outputs that the adversary has learned and the messages that have been sent on the honest parties' behalves, which were simulated without knowledge of the output.

Ishai et al. [IOZ14] defined a *setup compiler* which takes any generic MPC protocol in the OT-hybrid model that has security with abort against adaptive adversaries, and produces a sampling protocol for an arbitrary efficiently-sampleable distribution, with identifiable abort. They addressed the challenge of output consistency by introducing a technical trick to ensure the random coins are never revealed once the output has been learned: instead of computing the output directly, they first compute a secret sharing of the output that is authenticated with information-theoretic signatures, then they reconstruct by broadcasting the shares and verifying the signatures. If an abort occurs before the reconstruction, then no output has yet been defined and the simulator can choose any set of random coins: by the correctness of the protocol, they correlate to an output from the appropriate distribution. An abort occurs during reconstruction (i.e. after the adversary learns the output) only if the adversary broadcasts a share/signature pair that does not verify, and this condition is identifiable without opening any random coins. They address the challenge of equivocating protocol messages in the case of a pre-reconstruction abort by requiring both the input protocol of the setup compiler and the OT protocol that realizes the input protocol's OT oracle to be secure against adaptive corruptions.

Baum et al. [BOSS20] devised a different simulation strategy that only requires *statically* secure OT. They claim that it is tailored to the particular sampling protocol that realizes the particular sampling functionality they require for their main protocol. Much like Ishai et al., they address the challenge of output consistency by indirectly revealing the output: their protocol uses homomorphic commitments instead of information-theoretic signatures, the entire computation is performed in committed and secret-shared form, in parallel, and they use statistical checks to ensure the equivalence of the committed outputs and the secret-shared outputs before opening the commitments. Instead of equivocating protocol messages, their simulator produces messages on behalf of the honest parties by running the actual honest parties' code (for the secret-shared components). This prevents direct extraction of the corrupt parties' coins, but the parallel committed computation can be used to extract instead.

## 1.2 Our Contributions

In this work we present a new and efficient approach for constructing MPC with identifiable abort that follows the spirit of modern MPC protocols with unanimous abort. We begin by defining a weakened form of identifiable abort that is sufficient for sampling functionalities. We dub this property *input-revealing*

identifiable abort (IRIA), and we present a novel approach for realizing sampling functionalities with IRIA that is based upon delicately, selectively, and *asymmetrically* revealing committed inputs in a specific order to vindicate honest parties when an abort occurs. Unlike prior works, we do not rely upon generic zero-knowledge proofs, adaptively secure primitives, homomorphic primitives, or protocol compilers. Instead, we show that under our approach, messages and randomness can be equivocated when (and sometimes *only* when) they must be.

We revisit several well-known protocols from the MPC literature, and show that they can be modified in straightforward ways to achieve IRIA. Specifically:

- We modify the PVW [PVW08] OT protocol to give the OT sender the ability to decommit both of its inputs to the public. Our protocol achieves an intermediate form of security, between IA and IRIA, that involves revealing only the *sender's* inputs to the adversary when an abort occurs. Thus our protocol realizes Sender-Committed OT with Sender-Input-Revealing IA (SCOT-SIRIA) and Public Verifiability (PV). Relative to the unmodified PVW protocol, our modification is essentially additive, with the exception that all messages are sent via broadcast. We also give a simple $n$-wise correlation with a multiplicative depth of 1 from which SCOT (with PV and any flavor of IA) can be trivially realized. Details are given in section 3.

- We demonstrate that the Softspoken OT extension protocol of Roy [Roy22] can be enhanced to make it sender committing and to achieve SIRIA, using only an ideal SCOT-SIRIA oracle, a local random oracle, and a broadcast channel. In other words, SCOT-SIRIA (with PV) can be extended in much the same way as standard OT, and with essentially identical concrete cost.[2] Our enhancement is general enough to be applicable to other OT-extension protocols. Details are given in section 4.

- We revisit the vectorized multiplication (otherwise known as Vector Oblivious Linear Evaluation, or VOLE) protocol of Doerner et al. [DKLs18, DKLs19]. We show that if the protocol's OT oracle is replaced by a SCOT-SIRIA oracle, the protocol becomes committing for both participants. Only a few extra instructions are required to decommit, and the modified protocol *statistically* realizes Committed VOLE with IRIA (CVOLE-IRIA) and PV in any finite field. This protocol provides an intuitive argument that IRIA is in some sense easier than full adaptive security, because achieving adaptive security for the same protocol would require the simulator to solve an adversarially-influenced instance of an NP-hard problem. Details are given in section 5.

- We rewrite the preprocessing protocol of MASCOT [KOS16] in terms of an ideal CVOLE-IRIA oracle. This modularizes the MASCOT preprocessing protocol, which simplifies it considerably relative to its original presentation and reveals the essence of the mechanism by which authenticated Beaver

[2] As in standard OT extension, the number of invocations of the SCOT-SIRIA oracle depends only on the security parameter, and the number of invocations of the random oracle depends upon the size of the extension.

triples are generated with an unauthenticated multiplication primitive. This change *alone* is sufficient to statistically realize the SPDZ preprocessing functionality with IRIA, and with the property that all parties are committed to their outputs. Details are given in section 6.

- Finally, we argue that when the SPDZ preprocessing is generated by a functionality with output-committingness and IRIA (as previously realized), then the SPDZ online protocol can realize *any* sampling functionality with (standard) IA. We find this surprising, since the chain of realizations we have proposed involves few changes, no new assumptions, and little concrete overhead relative to plain MASCOT.

Since we can realize any sampling functionality with IA, we can realize the sampling functionalities that generate the correlated randomness required by the security-enhancing compiler of Ishai et al. [IOZ14] (hereafter referred to as the IOZ compiler). This leads to two theory-oriented interpretations of our results:

- Whereas the IOZ compiler as originally presented required an adaptively-secure OT protocol to sample its correlated randomness, we show the sufficiency of a statically-corrupteable *ideal* OT variant involving $n$ parties, of which which $n-2$ are passive "listeners." This implies that our $n$-party ideal functionality is complete for identifiable abort.[3] Plain OT has often been considered the fundamental primitive of multiparty computation with abort [Kil88, IPS08]. We offer SCOT-IRIA[4] as a candidate for the fundamental primitive of MPC with IA.

- Our result can be viewed as factoring the complex $n$-wise correlated randomness required by the IOZ compiler[5] into many copies of a very simple $n$-wise correlation with a multiplicative depth of 1. In section 3, we give an informal argument that no correlation complete for MPC with IA can be too much simpler. This serves as evidence that SCOT-IRIA is a reasonable fundamental primitive.

## 1.3 Related Work

Security with identifiable abort was first explicitly recognized by Aumann and Lindell [AL10] in the context of covert security. It been used implicitly in a large number of works both before and after it was named, particularly in the domain of fairness [GK09, BLOO11, GK12, BOO15, AO16, BHLT17, CL17, HT17, Dac20, CHOR22], and as a stepping stone to stronger security-notions [HMZ08, ZHM09, IKK+11].

[3] That is, ideal SCOT-SIRIA yields a statistical realization for any functionality with (plain) IA, in constant rounds.

[4] That is, a variant of SCOT with (double-sided) IRIA, which is strictly weaker than SCOT-SIRIA.

[5] Or, for that matter, any other MPC that achieves IA, such as the one proposed by Baum, Orsini, and Scholl [BOS16].

Identifiable abort has been well-studied in the literature. One line of work has focused on round-efficiency, yielding two-round protocols in the CRS model, with a precise understanding of when and how many times broadcast must occur to achieve specific security guarantees [CGZ20, DMR+21, DRSY22], and yielding (optimal) four-round protocols in the plain model [CRSW22]. Round-efficient protocols are also known for quantum computations [ACC+21]. These works, however, focused on theoretical feasibility within those round and setup constraints, and are not suitable for practical implementation. While our protocols yield generic MPC with IA and constant round complexity, we do not focus on optimizing the precise number of rounds, and instead prioritize the simplicity, modularity, and (where concrete performance is concerned) computational efficiency of our construction.

Another line of work has focused on the complexity of the correlated randomness required to achieve generic MPC with identifiable abort in the information-theoretic setting. Ishai and Ostrovsky proved with Zikas [IOZ14] that $n$-wise correlations suffice, and with Seyalioglu [IOS12] that pairwise correlations do not. Simkin et al. [SSY22] proved that for any number of corruptions up to $n-2$, correlations with cardinality $n-1$ are required, and Brandt et al. [BMMM20] showed that for $n-1$ corruptions, a correlation with cardinality $n-1$ is (surprisingly) sufficient. In our work we achieve security in the presence of $n-1$ corruptions via a correlation of cardinality $n$; we make no attempt to reduce the cardinality to $n-1$.

# 2 Preliminaries

## 2.1 Notation

We use $=$ for equality, $:=$ for right-to-left assignment, $=:$ for left-to-right assignment, and $\leftarrow$ for sampling from a distribution. In general, single-letter variables are set in *italic* font, function names are set in sans-serif font, and string literals are set in `slab-serif` font. We use $\mathbb{X}$ for an unspecified domain, $\mathbb{G}$ for a group, $\mathbb{F}$ for a field, $\mathbb{Z}$ for the integers, and $\mathbb{N}$ for the natural numbers. We use $\lambda_{\mathsf{c}}$ and $\lambda_{\mathsf{s}}$ to denote the computational and statistical security parameters, respectively.

Vectors and arrays are given in bold and indexed by subscripts; thus $\mathbf{a}_i$ is the $i^{\text{th}}$ element of the vector $\mathbf{a}$, which is distinct from the scalar variable $a$. When we wish to select a row or column from a multi-dimensional array, we place a $*$ in the dimension along which we are not selecting. Thus $\mathbf{b}_{*,j}$ is the $j^{\text{th}}$ column of matrix $\mathbf{b}$, $\mathbf{b}_{j,*}$ is the $j^{\text{th}}$ row, and $\mathbf{b}_{*,*} = \mathbf{b}$ refers to the entire matrix. We use bracket notation to generate inclusive ranges, so $[n]$ denotes the integers from 1 to $n$ and $[5,7] = \{5,6,7\}$. On rare occasions, we may use one vector to index another: if $\mathbf{a} := [2,7]$ and $\mathbf{b} := \{1,3,4\}$, then $\mathbf{a}_{\mathbf{b}} = \{2,4,5\}$. We use $|x|$ to denote the bit-length of $x$, and $|\mathbf{y}|$ to denote the number of elements in the vector $\mathbf{y}$. By convention, elliptic curve operations are expressed additively, and elliptic curve points are typically given capitalized variables. We use $\mathcal{P}_i$ to indicate an actively participating party with index $i$; in a typical context,

there will be a fixed set of active participants denoted $\mathcal{P}_1, \ldots, \mathcal{P}_n$. A party that observes passively but remains silent is denoted $\mathcal{V}$.

## 2.2 Security and Communication Model

We consider a malicious PPT adversary who can statically corrupt a dishonest majority of parties. All of our proofs are expressed in the Universal Composition framework of Canetti [Can01]. We note that our techniques do not rely on any specific properties of the framework, and be applied to any security framework that supports synchronous communication and composability.

We consider all messages to be sent over an authenticated broadcast channel, often denoted by $\mathcal{F}_{\mathsf{BC}}$ but here left implicit, and we do not consider any point-to-point communication. Since MPC with identifiable abort implies such a channel [CL17], we believe this to be reasonable.

By convention, we assume that obvious protocol blunders are handled appropriately. For example, if a particular party is expected to send a message in a protocol with identifiable abort, but does not do so (or sends a message that fails to type-check), then we assume the other parties abort and identify the silent party as a cheater.

## 2.3 Notions of Security

In the introduction, we introduced a number of security notions that an ideal functionality can express through the interface that it presents to the ideal adversary $\mathcal{S}$. Here we collect and explain them in one place, for the sake of clarity.

**Security with Abort.** The adversary may cause the protocol to terminate without producing output for the honest parties, and may condition termination on the corrupt parties' outputs value. Such failures cannot be attributed to any particular party. This is the most basic notion of security against malicious adversaries. If the parties are guaranteed to agree on whether an abort has occurred, then the abort is *unanimous*; otherwise it is *selective*. Any protocol with selective abort can be modified to achieve unanimous abort via a single broadcast round at the end.

**Security with Identifiable Abort (IA).** The adversary may cause the protocol to terminate without producing output for the honest parties, and may condition termination on the corrupt parties' outputs. When causing an abort, the adversary must identify one of its corrupt parties consistently to the honest parties.

**Security with Input-Revealing Identifiable Abort (IRIA).** We introduce this notion; it is similar to but strictly weaker than plain identifiable abort. In the case of an abort, the adversary learns the inputs of all honest parties and any random coins sampled by the functionality.

**Security with Sender-Input-Revealing Identifiable Abort (SIRIA).** This notion is relevant only to functionalities with asymmetric roles for the invoking parties, and lies between plain IA and IRIA. A functionality that has a sender role and a receiver role has SIRIA if the adversary learns only the input of the sender when an abort occurs.

**Guaranteed Output Delivery (GOD).** This is the strongest notion of security; it is unrealizable for many tasks [Cle86]. Guaranteed output delivery is traditionally defined in the *stand-alone model* (e.g., [CL17]) and cannot be captured in the inherently asynchronous UC framework. Katz et al. [KMTZ13] proposed a framework for modeling synchronous communication within UC which captures guaranteed termination. When discussing guaranteed output, we implicitly use this model.

**Public Verifiability (PV).** We model public verifiability as an abstract modifier for other functionalities. The parties interacting with any particular session of an unmodified functionality become the *active participants* in the modified functionality, but there may be additional *observing verifiers* who can register to receive outputs, but cannot otherwise influence the functionality.

**Functionality 2.1. $[\![\mathcal{F}]\!]_{\mathsf{PV}}$. Public Verifiability for $\mathcal{F}$ [CDKs22]**

The functionality $[\![\mathcal{F}]\!]_{\mathsf{PV}}$ is identical to the functionality $\mathcal{F}$, except that it interacts with an arbitrary number of additional *observing verification parties* (all of them denoted by $\mathcal{V}$, as distinct from the *actively participating parties* $\mathcal{P}_1$, $\mathcal{P}_2$, etc.). Furthermore, if *all* actively participating parties are corrupt, then $[\![\mathcal{F}]\!]_{\mathsf{PV}}$ receives its random coins from the adversary $\mathcal{S}$.

**Coin Retrieval:** Whenever the code of $\mathcal{F}$ requires a random value to be sampled from the domain $\mathbb{X}$, then sample as $\mathcal{F}$ would if at least one of the active participants is honest. If all active participants are corrupt, then send $(\texttt{need-coin}, \mathsf{sid}, \mathbb{X})$ to $\mathcal{S}$, and upon receiving $(\texttt{coin}, \mathsf{sid}, x)$ such that $x \in \mathbb{X}$ in response, continue behaving as $\mathcal{F}$, using $x$ as the required random value.

**Observer Registration:** Upon receiving $(\texttt{observe}, \mathsf{sid})$ from $\mathcal{V}$, remember the identity of $\mathcal{V}$, and if any message with the same $\mathsf{sid}$ is sent to *all* active participants in the future, then send it to $\mathcal{V}$ as well.

## 2.4 Building-Block Functionalities

The constructions in this paper make use of a number of functionalities that are well known from prior works. For completeness, enumerate them in this section. We use the same one-to-many commitment functionality $\mathcal{F}_{\mathsf{Com}}$ as Cohen et al. [CDKs22], which is similar to the one-to-many commitment functionalities given by Canetti and Fischlin [CF01] and Canetti et al. [CLOS02], except that it omits explicit destination parties in favor of allowing passive verifiers to receive

commitments when wrapped with $[\![\cdot]\!]_{\mathsf{PV}}$. We make use of the CRS-sampling functionality $\mathcal{F}_{\mathsf{CRS}}$ introduced by Canetti et al. [CLOS02], which simply samples a random value from the appropriate distribution when invoked, and outputs it to all parties. We also make use of the correlation-sampling functionality $\mathcal{F}_{\mathsf{Corr}}$ of Ishai et al. [IOZ14], which samples a set of correlated random values when invoked, and outputs each of them to a different party. In both cases, we assume the functionalities to have identifiable abort (though their original descriptions do not); i.e. if the adversary wishes to prevent any honest party from receiving output, it must identify one of its corrupt parties to the honest parties. Finally, we use a standard $n$-party coin-tossing functionality $\mathcal{F}_{\mathsf{CT}}$ with identifiable abort, which is similar to $\mathcal{F}_{\mathsf{CRS}}$ except that by convention it samples uniformly from some domain, instead of from an arbitrary distribution.

# 3 Sender-Committed OT

Basic 1-out-of-2 oblivious transfer (OT) [EGL85] is a two-party functionality to which a *sender* submits two messages and a *receiver* submits a single choice bit. The receiver learns only the message corresponding to the single bit it provided, and the sender learns nothing. Beaver [Bea95] realized that this simple relationship can also be viewed as a two-party correlation: a trusted dealer provides the sender with two random strings, and gives one of them at random to the receiver as well. To realize OT information-theoretically by consuming this correlation, the sender masks (or, in the computational setting, encrypts) its two messages with the random strings, and the receiver can decrypt exactly one. The receiver need only indicate to the sender via a single transmitted bit whether the order of messages should be swapped, and since the choice bit assigned to the receiver in the correlation is uniform, this gives the sender no information about which message is actually recoverable.

OT is a complete primitive for multiparty computation with (non-identifiable) abort [Kil88, IPS08], but Ishai et al. [IOS12] have shown a separation between any two-party correlation (including OT) and identifiable abort in the information-theoretic setting. We aim to make a minimal enhancement to OT, such that the result is complete for IA, and the aforementioned separation tells us that it is *necessary* that this enhancement involves additional parties. Intuitively, the minimal role in which additional parties can participate is as passive observers, and they must at the very least observe whether the enhanced functionality aborts, and who is responsible. This is not enough on its own, however. Because neither of the active participants is bound to its input in any way, if the enhanced OT functionality itself does not abort, but incorrect inputs are provided[6], then observers cannot determine which party provided an incorrect input. The simplest way to bind a party to its input is to send that input to the observers, but this too is not enough: if the honest party's inputs are sent to even one observer, then all-but-one security becomes

[6] *Correctness* here is defined with respect to some larger protocol that uses our enhanced OT functionality.

impossible, because the honest parties' secrets are immediately leaked to the adversary. We can mitigate this by *committing* the inputs to observers, so that it can be released only when an abort occurs and the parties must vindicate themselves. Specifically, it is enough to commit the *sender*'s input only, because this also makes the functionality weakly committing for the receiver: it receives the message corresponding to its input bit, and is bound to that input bit by its inability to guess the other message, so long as the messages have high enough entropy. We propose that this $n$-party primitive is a minimal enhancement of OT that is complete for MPC with IA.

### 3.1 Functionality with Sender-Input-Revealing IA

Our minimal enhancement blends standard OT with one-to-many commitment as expressed by Canetti et al. [CLOS02]: by submitting its inputs to the functionality, the sender also *commits* to them in such a way that they can be decommitted to both the receiver and to external observers at a later time. We refer to this enhancement as Sender-Committed OT or SCOT. It differs from the previously studied notion of double-sided committing OT [CvdGT95, Gar04] in a subtle way: double-sided committing OT allows the receiver to decommit to its choice bit *without* the sender decommitting its inputs, whereas our functionality does not.

The weakest form of identifiable abort is *input-revealing* for both active participants, and we will prove that this weak form is complete for IA via Theorems 5.2 and 6.3. However, the VOLE construction for arbitrary fields that we introduce in section 5 requires that the receiver's inputs remain hidden, and so the variant we give formally has only sender-input-revealing IA.

**Functionality 3.1.** $\mathcal{F}_{\mathsf{SCOT\text{-}SIRIA}}(\mathbb{X})$**: Sender-Committed OT**

This functionality interacts with two active participants, $\mathcal{P}_\mathsf{A}$ and $\mathcal{P}_\mathsf{B}$, who we refer to as Alice and Bob, and with the ideal adversary $\mathcal{S}$. It is parameterized by a domain $\mathbb{X}$.

**OT:** On receiving $(\texttt{choose}, \mathsf{sid}, \beta)$ from Bob such that $\beta \in \{0,1\}$ and $\mathcal{P}_\mathsf{B} \| \mathcal{P}_\mathsf{A} \| \mathsf{sid}' = \mathsf{sid}$ for some fresh $\mathsf{sid}'$, send $(\texttt{choice-made}, \mathsf{sid})$ to all parties. On receiving $(\texttt{messages}, \mathsf{sid}, \alpha^0, \alpha^1)$ from Alice such that $\alpha^0, \alpha^1 \in \mathbb{X}$, store Alice's message in memory and send $(\texttt{messages-loaded}, \mathsf{sid})$ to all parties. When both Alice and Bob's messages have been received, send $(\texttt{chosen-message}, \mathsf{sid}, \alpha^\beta)$ to Bob and send $(\texttt{ot-done}, \mathsf{sid})$ to all parties.

**Opening:** On receiving $(\texttt{open}, \mathsf{sid})$ from Alice, if the record $(\texttt{messages}, \mathsf{sid}, \alpha^0, \alpha^1)$ exists in memory, then send it to all parties and ignore all future instructions with the same $\mathsf{sid}$ value.

**Abort:** On receiving $(\texttt{abort}, \mathsf{sid})$ from $\mathcal{S}$ at any point, such that $\mathcal{P}_\mathsf{B} \| \mathcal{P}_\mathsf{A} \| \mathsf{sid}' = \mathsf{sid}$, if $\mathcal{P}_c$ is corrupt for some $c \in \{\mathsf{A}, \mathsf{B}\}$, then send $(\texttt{messages}, \mathsf{sid}, \alpha^0, \alpha^1)$ to $\mathcal{S}$ if such a record exists in memory,[a] and regardless send $(\texttt{abort}, \mathsf{sid}, \mathcal{P}_c)$ to all parties and ignore all future instructions with the same $\mathsf{sid}$ value.

[a] This functionality can be transformed into $\mathcal{F}_{\mathsf{SCOT\text{-}IRIA}}$, with double-sided input-revealing IA, simply by releasing $\beta$ along with $\alpha^0$ and $\alpha^1$ in the case of abort, and it can be transformed into $\mathcal{F}_{\mathsf{SCOT\text{-}IA}}$, with *standard* IA, by releasing none of these values.

In Section 3.2 we explore a simple $n$-wise correlation, from which our functionality can be realized information-theoretically, and in Section 3.3 we give a direct *computational* realization.

## 3.2 Information-Theoretic Realization from Correlated Randomness

Our enhanced OT functionality (with *standard* identifiable abort) can be realized information-theoretically assuming an $n$-wise correlation where $n$ is the total number of participants (including passive verifiers), much as traditional OT can be realized information-theoretically assuming a pairwise correlation. Specifically, for some field $\mathbb{F}$ consider the distribution

$$\mathcal{D}_{\mathsf{SCOT\text{-}(\mathbb{F},\lambda_s)}} = \left\{ \begin{array}{r} (\dot{\alpha}^0, \dot{\alpha}^1, \dot{\beta}, \dot{\gamma}, \boldsymbol{\delta}^0, \boldsymbol{\delta}^1, \boldsymbol{\eta}^0, \boldsymbol{\eta}^1, \mathring{\boldsymbol{\alpha}}^0, \mathring{\boldsymbol{\alpha}}^1) \quad : \\ \{\dot{\alpha}^0, \dot{\alpha}^1\} \leftarrow \mathbb{F}^2, \\ \boldsymbol{\delta}^0 \| \boldsymbol{\delta}^1 \| \boldsymbol{\eta}^0 \| \boldsymbol{\eta}^1 \leftarrow \mathbb{F}^{4n-4 \times \lceil \log_2(|\mathbb{F}|)/\lambda_s \rceil}, \\ \dot{\beta} \leftarrow \{0_\mathbb{F}, 1_\mathbb{F}\}, \\ \dot{\gamma} = \dot{\beta} \cdot \dot{\alpha}^1 + (1_\mathbb{F} - \dot{\beta}) \cdot \dot{\alpha}^0, \\ \mathring{\boldsymbol{\alpha}}^0 = \{\{\boldsymbol{\delta}^0_{i,j} \cdot \dot{\alpha}^0 + \boldsymbol{\eta}^0_{i,j}\}_{j \in [\lceil \log_2 |\mathbb{F}|/\lambda_s \rceil]}\}_{i \in [n-1]}, \\ \mathring{\boldsymbol{\alpha}}^1 = \{\{\boldsymbol{\delta}^1_{i,j} \cdot \dot{\alpha}^1 + \boldsymbol{\eta}^1_{i,j}\}_{\lceil \log_2 |\mathbb{F}|/\lambda_s \rceil}\}_{i \in [n-1]} \end{array} \right\}$$

where $(\dot{\alpha}^0, \dot{\alpha}^1, \mathring{\boldsymbol{\alpha}}^0, \mathring{\boldsymbol{\alpha}}^1)$ is known to the sender, $(\dot{\beta}, \dot{\gamma}, \boldsymbol{\delta}^0_{1,*}, \boldsymbol{\delta}^1_{1,*}, \boldsymbol{\eta}^0_{1,*}, \boldsymbol{\eta}^1_{1,*})$ is known to the receiver, and each $i^{\text{th}}$ passive verifier knows $(\boldsymbol{\delta}^0_{i+1,*}, \boldsymbol{\delta}^1_{i+1,*}, \boldsymbol{\eta}^0_{i+1,*}, \boldsymbol{\eta}^1_{i+1,*})$. This is an $n$-wise correlation of degree 2 which could hypothetically be generated by a pseudorandom correlation generator [BCG+19] or pseudorandom correlation function [BCG+20], or by a more traditional interactive protocol. It comprises an ordinary random OT correlation $(\dot{\alpha}^0, \dot{\alpha}^1, \dot{\beta}, \dot{\gamma})$ and, for every party except the sender, a statistical MAC $\mathring{\boldsymbol{\alpha}}^b_{i,*}$ with $\lceil \log_2(|\mathbb{F}|)/\lambda_s \rceil$ repetitions on $\dot{\alpha}^b$ for $b \in \{0,1\}$ under the key $(\boldsymbol{\delta}^b_{i,*}, \boldsymbol{\eta}^b_{i,*})$.

Our simple information-theoretic protocol realizing $\mathcal{F}_{\mathsf{SCOT\text{-}IA}}(\mathbb{F})$ via the above correlation combines the classic protocol for realizing OT from an OT correlation with simple MAC checks during the opening phase. It cannot abort, except if a party fails to send a message or if the MAC checks fail. Because we

model the correlation-sampling functionality as requiring the active participation of all parties who receive outputs, all parties in this protocol (even verifiers) are active.

**Protocol 3.2.** $\pi_{\text{SCOT-SIRIA-PV-IT}}(n, \mathbb{F})$**: Statistical SCOT**

This protocol runs with $n$ active participants, denoted $\mathcal{P}_1, \ldots, \mathcal{P}_n$. The parties have access to an ideal correlation-sampling functionality $\mathcal{F}_{\mathsf{Corr}}^{\mathcal{D}_{\text{SCOT-}(\mathbb{F},\lambda_s)}}(n)$ that has identifiable abort. It is parameterized by a field $\mathbb{F}$.

**Initialization:** On receiving $(\texttt{init}, \mathsf{sid})$ from the environment such that $\mathcal{P}_1 \| \ldots \| \mathcal{P}_n \| \mathsf{sid}' = \mathsf{sid}$ for some fresh $\mathsf{sid}'$,

- $\mathcal{P}_1$ sends $(\texttt{sample}, \mathsf{sid}, \texttt{sender})$ to $\mathcal{F}_{\mathsf{Corr}}^{\mathcal{D}_{\text{SCOT-}(\mathbb{F},\lambda_s)}}(n)$ and receives $(\texttt{correlation}, \mathsf{sid}, (\dot{\alpha}^0, \dot{\alpha}^1, \mathring{\boldsymbol{\alpha}}^0, \mathring{\boldsymbol{\alpha}}^1))$ in response.
- $\mathcal{P}_2$ sends $(\texttt{sample}, \mathsf{sid}, \texttt{receiver})$ to $\mathcal{F}_{\mathsf{Corr}}^{\mathcal{D}_{\text{SCOT-}(\mathbb{F},\lambda_s)}}(n)$ and receives $(\texttt{correlation}, \mathsf{sid}, (\dot{\beta}, \dot{\gamma}, \boldsymbol{\delta}_{1,*}^0, \boldsymbol{\delta}_{1,*}^1, \boldsymbol{\eta}_{1,*}^0, \boldsymbol{\eta}_{1,*}^1))$ in response.
- $\mathcal{P}_i$ for $i \in [3, n]$ sends $(\texttt{sample}, \mathsf{sid}, \texttt{verifier})$ to $\mathcal{F}_{\mathsf{Corr}}^{\mathcal{D}_{\text{SCOT-}(\mathbb{F},\lambda_s)}}(n)$ and receives $(\boldsymbol{\delta}_{i-1,*}^0, \boldsymbol{\delta}_{i-1,*}^1, \boldsymbol{\eta}_{i-1,*}^0, \boldsymbol{\eta}_{i-1,*}^1)$ in response.

On receiving these responses, they all output $(\texttt{setup-done}, \mathsf{sid})$ to the environment.

**OT:**

1. On receiving $(\texttt{choose}, \mathsf{sid}, \beta)$ from the environment, $\mathcal{P}_2$ broadcasts $(\texttt{swap}, \mathsf{sid}, \dot{\beta} \oplus \beta)$ to all parties.
2. On receiving $(\texttt{messages}, \mathsf{sid}, \alpha^0, \alpha^1)$ from the environment and $(\texttt{swap}, \mathsf{sid}, s)$ from $\mathcal{P}_2$, party $\mathcal{P}_1$ computes $\tilde{\alpha}^0 := \alpha^s + \dot{\alpha}^0$ and $\tilde{\alpha}^1 := \alpha^{1-s} + \dot{\alpha}^1$ and broadcasts $(\texttt{transfer}, \mathsf{sid}, \tilde{\alpha}^0, \tilde{\alpha}^1)$.
3. On receiving $(\texttt{transfer}, \mathsf{sid}, \tilde{\alpha}^0, \tilde{\alpha}^1)$ from $\mathcal{P}_1$, party $\mathcal{P}_2$ computes $\gamma := \dot{\beta} \cdot \tilde{\alpha}^1 + (1 - \dot{\beta}) \cdot \tilde{\alpha}^0 - \dot{\gamma}$ and outputs $(\texttt{chosen-message}, \mathsf{sid}, \gamma)$ to the environment.
4. All parties, upon witnessing this sequence of broadcasts, output $(\texttt{ot-done}, \mathsf{sid})$ to the environment.

**Opening:** On receiving $(\texttt{open}, \mathsf{sid})$ from the environment, $\mathcal{P}_1$ broadcasts $(\texttt{opening}, \mathsf{sid}, (\dot{\alpha}^0, \dot{\alpha}^1, \mathring{\boldsymbol{\alpha}}^0, \mathring{\boldsymbol{\alpha}}^1))$ to all parties. Every other party $\mathcal{P}_i$ for $i \in [2, n]$ receives this message and verifies that $\mathring{\boldsymbol{\alpha}}_{i-1,j}^b = \boldsymbol{\delta}_{i-1,j}^b \cdot \dot{\alpha}^b + \boldsymbol{\eta}_{i-1,j}^b$ for $b \in \{0, 1\}$ and $j \in [\lceil \log_2(|\mathbb{F}|)/\lambda_s \rceil]$. If any of these equations do not hold, then $\mathcal{P}_i$ outputs $(\texttt{abort}, \mathsf{sid}, \mathcal{P}_1)$ to the environment. If they all hold, then $\mathcal{P}_i$ outputs $(\texttt{messages}, \mathsf{sid}, \tilde{\alpha}^s - \dot{\alpha}^0, \tilde{\alpha}^{1-s} - \dot{\alpha}^1)$.

**Theorem 3.3.** *For any $n \in \mathbb{N}$ such that $n \geq 2$ and any field $\mathbb{F}$,*

$\pi_{\mathsf{SCOT\text{-}SIRIA\text{-}PV\text{-}IT}}(n, \mathbb{F})$ *statistically UC-realizes* $[\![\mathcal{F}_{\mathsf{SCOT\text{-}IA}}(\mathbb{F})]\!]_{\mathsf{PV}}$ *with exactly* $n-2$ *observing verifiers in the* $\mathcal{F}_{\mathsf{Corr}}^{\mathcal{D}_{\mathsf{SCOT}\text{-}(\mathbb{F},\lambda_{\mathsf{S}})}}(n)$*-hybrid model, in the presence of an unbounded malicious adversary statically corrupting any number of verifiers (denoted* $\mathcal{P}_3, \ldots, \mathcal{P}_n$ *in the protocol) and at most one of the active participants* $\mathcal{P}_1 = \mathcal{P}_{\mathsf{A}}$ *and* $\mathcal{P}_2 = \mathcal{P}_{\mathsf{B}}$*.*

*Proof Sketch.* This theorem follows directly from the statistical security of the linear MAC scheme and the perfect security of the OT protocol on which $\pi_{\mathsf{SCOT\text{-}SIRIA\text{-}PV\text{-}IT}}$ is based. □

### 3.3 Direct Computational Realization via PVW

Peikert et al. [PVW08] proposed a composable OT framework (the *PVW framework*) instantiable from the Decisional Diffie-Hellman (DDH), Quadratic Residuosity (QR), or Learning with Errors (LWE) [Qua20] assumptions. We observe that their protocol is *already* committing for the sender, and we need only add an additional phase to the protocol by which the sender can decommit. Prior works [JKO13, GKPS18] have added a similar decommitment capability simply by revealing the sender's randomness. However, this approach does not seem to be simulation-secure: for example, in the DDH-based instantiation, simulation of a decommitment for an arbitrary pair of messages seems to require breaking the Discrete-Logarithm (DL) assumption, which invalidates the scheme.

We take a different approach: the sender reveals its inputs, and then proves that they are correct decommitments with respect to the messages already transmitted. Although our protocol achieves UC security, we do *not* require a straight-line-extractable proof of knowledge to do this, because our simulator does not require anything additional to be extracted from the sender. Instead, we combine a simple sigma protocol with a commitment scheme that is binding and equivocable, but *not* extractable, and we show that if a corrupt sender cheats, then there exists a rewinding reduction that breaks the binding property of the commitment scheme. This allows our protocol to achieve composable security while avoiding the compromises typically associated with composable zero-knowledge proofs: specifically, the computation and communication overheads and the programming of the random oracle associated with straight-line extractors [Pas03, Fis05, Ks22].[7] Our approach is general: it can be applied to any instantiation of the PVW framework, provided that the underlying *dual-mode encryption* scheme supports a sigma protocol for *proof of correct encryption*, which we define below. In addition to a general modification of the general PVW framework, we give an explicit construction of our approach based upon the DDH instantiation of PVW, Pedersen commitments, and a variant of the Schnorr protocol. We leave explicit constructions from other assumptions to future work.

[7] We note that in the UC model, the simulator is forbidden to rewind the environment, which precludes rewinding-based simulation techniques, but reductions run the whole experiment, including the environment, as a subroutine, and thus they can make use of rewinding in the usual way. See, for example, Dodis, Shoup, and Walfish [DSW08].

In the original PVW scheme, all messages correspond to valid inputs, and aborts only occur if parties fail to speak. In our new decommitment phase, a corrupt receiver can cause an abort after the sender's inputs have been revealed. Thus, our approach yields security with sender-input-revealing IA.

### 3.4 Building Blocks for PVW

**Dual-Mode Encryption.** Peikert et al. introduced a primitive that they refer to as *dual-mode* encryption, a form of public-key encryption in the CRS model with an added notion of *branches* and two *modes*. Given some public key, messages can be encrypted to any branch. In *messy* mode, some branches are decryptable in the usual sense with a corresponding secret key, whereas some branches are *messy*, meaning that the resulting encryption contains no information about the message. In messy mode, all public keys (including ill-formed ones) contain some messy branches, and messy branches can be efficiently distinguished from decryptable ones if and only if the distinguisher has knowledge of a trapdoor that is embedded in the CRS. In *decryptable* mode, on the other hand, the trapdoor can be used to sample public keys that have *only* decryptable branches. The two modes are efficiently distinguishable if and only if the distinguisher has knowledge of the trapdoor.

More formally, a two-branch dual-mode encryption scheme is a tuple of algorithms $(\mathsf{DMSetup}, \mathsf{DMKeyGen}, \mathsf{DMEnc}, \mathsf{DMDec}, \mathsf{DMTrapKeyGen}, \mathsf{DMFindMessy})$ such that:

1. $(\mathsf{crs}, \mathsf{td}) \leftarrow \mathsf{DMSetup}(1^\lambda, \mu)$ samples a CRS and a trapdoor, given the mode $\mu \in \{0,1\}$, where $\mu = 0$ indicates *messy* mode, and $\mu = 1$ indicates *decryptable* mode. We require that CRSes produced in the two modes are computationally indistinguishable unless the trapdoor is known. That is,

$$\left\{\mathsf{crs} : (\mathsf{crs}, \mathsf{td}) \leftarrow \mathsf{DMSetup}(1^\lambda, 0)\right\} \approx_c \left\{\mathsf{crs} : (\mathsf{crs}, \mathsf{td}) \leftarrow \mathsf{DMSetup}(1^\lambda, 1)\right\}$$

2. $(\mathsf{pk}, \mathsf{sk}) \leftarrow \mathsf{DMKeyGen}(\mathsf{crs}, \beta)$ samples a public key $\mathsf{pk}$, along with the secret key $\mathsf{sk}$ for branch $\beta \in \{0,1\}$.

3. $\tilde{m} \leftarrow \mathsf{DMEnc}(\mathsf{crs}, \mathsf{pk}, b, m)$ emits an encryption $\tilde{m}$ of message $m \in \mathbb{M}$ under branch $b \in \{0,1\}$ of $\mathsf{pk}$. If $b$ is a messy branch of $\mathsf{pk}$, then $\tilde{m}$ contains no information about $m$. If $b$ is a decryptable branch, then $\tilde{m}$ is semantically secure in the usual way.

4. $m := \mathsf{DMDec}(\mathsf{crs}, \mathsf{sk}, \tilde{m})$ emits a message $m \in \mathbb{M}$ if $\tilde{m}$ is the encryption of $m$ under a decryptable branch of the public key $\mathsf{pk}$ corresponding to $\mathsf{sk}$. In other words, we require that for every $m$ in the message space and $(\beta, \mu) \in \{0,1\}^2$

$$m = \mathsf{DMDec}(\mathsf{crs}, \mathsf{sk}, \mathsf{DMEnc}(\mathsf{crs}, \mathsf{pk}, \beta, m)) : \\ (\mathsf{crs}, \mathsf{td}) \leftarrow \mathsf{DMSetup}(1^\lambda, \mu), (\mathsf{pk}, \mathsf{sk}) \leftarrow \mathsf{DMKeyGen}(\mathsf{crs}, \beta)$$

with probability 1 over the coins of the algorithms. This is *perfect correctness*.

5. $(\mathsf{pk}, \mathsf{sk}_0, \mathsf{sk}_1) \leftarrow \mathsf{DMTrapKeyGen}(\mathsf{crs}, \mathsf{td})$ samples a public key $\mathsf{pk}$, along with *two* secret keys, one for each branch. We require that keys generated via the trapdoor be statistically indistinguishable from ordinary ones under the same CRS mode. That is, for $\beta \in \{0,1\}$, if $(\mathsf{crs}, \mathsf{td}) \leftarrow \mathsf{DMSetup}(1^\lambda, 1)$, then

$$\{(\mathsf{pk}, \mathsf{sk}_\beta) : (\mathsf{pk}, \mathsf{sk}) \leftarrow \mathsf{DMKeyGen}(\mathsf{crs}, \beta)\}$$
$$\approx_s \{(\mathsf{pk}, \mathsf{sk}_\beta) : (\mathsf{pk}, \mathsf{sk}_0, \mathsf{sk}_1) \leftarrow \mathsf{DMTrapKeyGen}(\mathsf{crs}, \mathsf{td})\}$$

6. $b \leftarrow \mathsf{DMFindMessy}(\mathsf{crs}, \mathsf{td}, \mathsf{pk})$ outputs the index of a messy branch of $\mathsf{pk}$. We require that for every $\beta \in \{0,1\}$,

$$1 - \beta = \mathsf{DMFindMessy}(\mathsf{crs}, \mathsf{td}, \mathsf{pk}) :$$
$$(\mathsf{crs}, \mathsf{td}) \leftarrow \mathsf{DMSetup}(1^\lambda, 0), (\mathsf{pk}, \mathsf{sk}) \leftarrow \mathsf{DMKeyGen}(\mathsf{crs}, \beta)$$

with overwhelming probability over the coins of the algorithms.

In addition to the above mentioned properties, we note that ciphertext validity is a public property; i.e., it must be possible to determine from only the CRS, public key, and ciphertext value whether decryption of a ciphertext is possible. This is easily satisfied if all ciphertexts decrypt under all keys.

Finally, we will recall the DDH-based instantiation of Peikert et al. for messages in $\mathbb{G}$. For a complete formal description of dual-mode encryption, including its security properties, for proofs of the security of the DDH-based instantiation we have recalled here, and for other instantiations, we refer the reader to Peikert et al. [PVW08] and Quach [Qua20].

**Algorithm 3.4. Dual-Mode Encryption from DDH [PVW08]**

These algorithms are parameterized by $(\mathbb{G}, G^0, q) = \mathcal{G} \leftarrow \mathsf{GroupGen}(1^\lambda)$. The message domain of this cryptosystem is $\mathbb{G}$.

$\mathsf{DMSetup}_{\mathcal{G}}(\mu)$ :
If $\mu = 0$, then sample $(\tau^0, \tau^1) \leftarrow \mathbb{Z}_q^2$ and $G^1 \leftarrow \mathbb{G}$, compute $H^b := \tau^b \cdot G^b$ for $b \in \{0,1\}$, and output $((G^0, H^0, G^1, H^1), (\tau^0, \tau^1))$.
If $\mu = 1$, then sample $\tau \leftarrow \mathbb{Z}_q$ and $H^0 \leftarrow \mathbb{G}$, compute $G^1 := \tau \cdot G^0$ and $H^1 := \tau \cdot H^0$, and output $((G^0, H^0, G^1, H^1), \tau)$.

$\mathsf{DMKeyGen}_{\mathcal{G}}((G^0, H^0, G^1, H^1), \beta)$ :
Sample $\rho \leftarrow \mathbb{Z}_q$, compute $G^2 := \rho \cdot G^\beta$ and $H^2 := \rho \cdot H^\beta$, and output $((G^2, H^2), \rho)$.

$\mathsf{DMEnc}_{\mathcal{G}}((G^0, H^0, G^1, H^1), (G^2, H^2), b, M)$ :
Sample $(r, s) \leftarrow \mathbb{Z}_q^2$, compute $U := r \cdot G^b + s \cdot H^b$ and $V := r \cdot G^2 + s \cdot H^2$, and output $(U, M + V)$.

$\mathsf{DMDec}_{\mathcal{G}}((G^0, H^0, G^1, H^1), \rho, (U, \tilde{M}))$ :
Output $\tilde{M} - \rho \cdot U$.

$\mathsf{DMTrapKeyGen}_{\mathcal{G}}((G^0, H^0, G^1, H^1), \tau)$ :
Sample $\rho \leftarrow \mathbb{Z}_q$, compute $G^2 := \rho \cdot G^0$ and $H^2 := \rho \cdot H^0$, and output $((G^2, H^2), \rho, \rho/\tau)$.

$\mathsf{DMFindMessy}_{\mathcal{G}}((G^0, H^0, G^1, H^1), (\tau^0, \tau^1), (G^2, H^2))$ :
Output 1 if $H^2 = \tau^0 \cdot G^2$. Otherwise output 0.

**Claim 3.5.** *If* $\mathsf{GroupGen}(1^\lambda)$ *outputs a distribution of cyclic groups relative to which the decisional Diffie-Hellman problem is hard, then the algorithms* $(\mathsf{DMSetup}_{\mathcal{G}}, \mathsf{DMKeyGen}_{\mathcal{G}}, \mathsf{DMEnc}_{\mathcal{G}}, \mathsf{DMDec}_{\mathcal{G}}, \mathsf{DMTrapKeyGen}_{\mathcal{G}}, \mathsf{DMFindMessy}_{\mathcal{G}})$ *constitute a two-branch dual-mode encryption scheme for messages from* $\mathbb{G}$*, given* $(\mathbb{G}, G^0, q) = \mathcal{G} \leftarrow \mathsf{GroupGen}(1^\lambda)$.

We direct the reader to Peikert et al. [PVW08] for proof of this claim.

**Proof of Correct Encryption.** Under our modification, the sender must be able to prove that a ciphertext is a valid encryption of a particular input, even when the ciphertext is messy. It does this by proving knowledge of the coins used by the encryption algorithm. That is, if we let $\mathsf{DMEnc}(\mathsf{crs}, \mathsf{pk}, b, m; r)$ be a deterministic algorithm that takes the randomness $r$ as input rather than sampling it internally, then the relation

$$\mathcal{R}_{\mathsf{DMEnc}}((\mathsf{crs}, \mathsf{pk}, b, m, \tilde{m}), r) \mapsto \tilde{m} = \mathsf{DMEnc}(\mathsf{crs}, \mathsf{pk}, b, m; r)$$

defines membership in a language for which we must construct a zero-knowledge proof of knowledge of a witness:

$$\mathcal{L}_{\mathsf{DMEnc}} = \{(\mathsf{crs}, \mathsf{pk}, b, m, \tilde{m}) : \exists r \text{ s.t. } \mathcal{R}_{\mathsf{DMEnc}}((\mathsf{crs}, \mathsf{pk}, b, m, \tilde{m}), r) = 1\}$$

For the Peikert et al.'s DDH-based instantiation of messy encryption, this language is simply the language of double discrete logarithm equality with a public offset, i.e.

$$\mathcal{L}_{\mathsf{DMEnc}_{\mathcal{G}}} = \left\{ \begin{array}{l} ((G^0, H^0, G^1, H^1), (G^2, H^2), b, M, (U, \tilde{M})) : \\ \quad \exists (r, s) \in \mathbb{Z}_q^2 \text{ s.t. } (U, \tilde{M}) = (r \cdot G^b + s \cdot H^b, r \cdot G^2 + s \cdot H^2 + M) \end{array} \right\}$$

**Sigma Protocols.** A sigma protocol is a three-message protocol for proving knowledge of the witness $w$ for some element $x$ of a language $\mathcal{L}$. Formally, it comprises a quartet of algorithms $(\Sigma^1_{\mathcal{L}}, \Sigma^2_{\mathcal{L}}, \Sigma^3_{\mathcal{L}}, \Sigma^{\mathsf{V}}_{\mathcal{L}})$ such that:

1. The prover computes $(a, s) \leftarrow \Sigma^1_{\mathcal{L}}(x, w)$ and sends $a$ to the verifier.
2. The verifier computes $e \leftarrow \Sigma^2_{\mathcal{L}}(x)$ and sends $e$ to the verifier.
3. The prover computes $z \leftarrow \Sigma^3_{\mathcal{L}}(x, w, s, e)$ and sends $z$ to the verifier.

4. The verifier accepts if and only if $\Sigma^{\mathsf{V}}_{\mathcal{L}}(x, a, e, z) = 1$

and such that the following properties hold:

- *Completeness:* If the prover runs honestly, and $x \in \mathcal{L}$ with witness $w$, then the verifier always accepts.
- *Special Soundness:* There exists an efficient *extractor* algorithm $\Sigma^{\mathsf{E}}_{\mathcal{L}}$ such that for any two transcripts $(x, a, e, z)$ and $(x, a, e', z')$, if $\Sigma^{\mathsf{V}}_{\mathcal{L}}(x, a, e, z) = 1$ and $\Sigma^{\mathsf{V}}_{\mathcal{L}}(x, a, e', z') = 1$ and $e \neq e'$, then $w := \Sigma^{\mathsf{E}}_{\mathcal{L}}(x, a, e, e', z, z')$ such that $w$ is a witness that $x \in \mathcal{L}$.
- *Special Honest Verifier Zero Knowledge:* There exists an efficient *simulator* algorithm $(a, z) \leftarrow \Sigma^{\mathsf{S}}_{\mathcal{L}}(x, e)$ that produces simulated transcripts given an instance $x$, with the same distribution as the transcripts produced by an interaction between an honest prover and an honest verifier.

Our proof in section 3.5 will require the following generic lemma about sigma protocols:

**Lemma 3.6.** *For any $(x^0, a^0, e^0, z^0)$ and $(x^1, a^1, e^1, z^1)$, if $e^0 \neq e^1$ and if for $b \in \{0, 1\}$ it holds that $\Sigma^{\mathsf{V}}_{\mathcal{L}}(x^b, a^b, e^b, z^b) = 1$ and $x^b \notin \mathcal{L}$, then either $x^0 \neq x^1$ or $a^0 \neq a^1$.*

*Proof.* Suppose the lemma were false; i.e. $e^0 \neq e^1$ and $x^0 = x^1$ and $a^0 = a^1$ and for $b \in \{0, 1\}$, $\Sigma^{\mathsf{V}}_{\mathcal{L}}(x^b, a^b, e^b, z^b) = 1$ and $x^b \notin \mathcal{L}$. If this were the case, then we could apply the extractor $w^0 := \Sigma^{\mathsf{E}}_{\mathcal{L}}(x^0, a^0, e^0, e^1, z^0, z^1)$, and by the special soundness of the sigma protocol, $w^0$ must be a witness that $x^0 \in \mathcal{L}$. This is a contradiction. □

Finally, we note that the sigma protocol for $\mathcal{L}_{\mathsf{DMEnc}_{\mathcal{G}}}$—i.e. for the language of correct encryptions under Peikert et al.'s DDH-based instantiation of dual-mode encryption—is a simple extension of Schnorr's famous identification protocol [Sch89]. In particular, omitting unused inputs we have:

**Algorithm 3.7. Sigma Protocol for Membership in $\mathcal{L}_{\mathsf{DMEnc}_{\mathcal{G}}}$**

These algorithms are parameterized by $(\mathbb{G}, G, q) = \mathcal{G} \leftarrow \mathsf{GroupGen}(1^\lambda)$.

$\Sigma^1_{\mathsf{DMEnc}_{\mathcal{G}}}(G^0, H^0, G^1, H^1, G^2, H^2, b)$ :
Sample $(\bar{r}, \bar{s}) \leftarrow \mathbb{Z}_q^2$, compute $\bar{U} := \bar{r} \cdot G^b + \bar{s} \cdot H^b$ and $\bar{V} := \bar{r} \cdot G^2 + \bar{s} \cdot H^2$, and output $((\bar{U}, \bar{V}), (\bar{r}, \bar{s}))$.

$\Sigma^2_{\mathsf{DMEnc}_{\mathcal{G}}}()$ :
Sample a uniform $e \leftarrow \mathbb{Z}_q$ and output it.

$\Sigma^3_{\mathsf{DMEnc}_{\mathcal{G}}}(G^0, H^0, G^1, H^1, G^2, H^2, b, r, s, \bar{r}, \bar{s}, e)$ :
Compute $y := e \cdot r + \bar{r}$ and $z := e \cdot s + \bar{s}$ and output $(y, z)$.

$\Sigma^{\mathsf{V}}_{\mathsf{DMEnc}_{\mathcal{G}}}(G^0, H^0, G^1, H^1, G^2, H^2, b, M, U, \tilde{M}, \bar{U}, \bar{V}, e, y, z)$ :

Output 1 if $e \cdot U + \bar{U} = y \cdot G^b + z \cdot H^b$ and $e \cdot (\tilde{M} - M) + \bar{V} = y \cdot G^2 + z \cdot H^2$. Otherwise output 0.

$\Sigma^{\mathsf{E}}_{\mathsf{DMEnc}_{\mathcal{G}}}(e, e', y, z, y', z')$ :
Compute $r := y - y'/(e - e')$ and $s := z - z'/(e - e')$ and output $(r, s)$.

$\Sigma^{\mathsf{S}}_{\mathsf{DMEnc}_{\mathcal{G}}}(G^0, H^0, G^1, H^1, G^2, H^2, b, M, U, \tilde{M}, e)$ :
Sample $(y, z) \leftarrow \mathbb{Z}_q^2$, compute $\bar{U} := y \cdot G^b + z \cdot H^b - e \cdot U$ and $\bar{V} := y \cdot G^2 + z \cdot H^2 - e \cdot (\tilde{M} - M)$ and output $(\bar{U}, \bar{V}, e, y, z)$.

**Claim 3.8.** *The algorithms* $(\Sigma^1_{\mathsf{DMEnc}_{\mathcal{G}}}, \Sigma^2_{\mathsf{DMEnc}_{\mathcal{G}}}, \Sigma^3_{\mathsf{DMEnc}_{\mathcal{G}}}, \Sigma^{\mathsf{V}}_{\mathsf{DMEnc}_{\mathcal{G}}})$ *constitute a sigma protocol for the language* $\mathcal{L}_{\mathsf{DMEnc}}$ *with knowledge error* $2^{-|q|}$.

We do not include a proof of this claim, since this protocol is a simple adaptation of Schnorr's protocol.

**Equivocable Commitments in the CRS Model.** Finally, we require a commitment scheme that is much too weak to realize the ideal commitment functionality $\mathcal{F}_{\mathsf{Com}}$: one that has hiding, binding, and equivocation, but not necessarily extractability. Informally, an equivocable commitment scheme in the CRS model is a tuple of algorithms $(\mathsf{CSetup}, \mathsf{CCom}, \mathsf{COpen}, \mathsf{CTrapSetup}, \mathsf{CTrapCom}, \mathsf{CEquiv})$ such that:

1. $\mathsf{crs} \leftarrow \mathsf{CSetup}(1^\lambda)$ samples a CRS.
2. $(c, d) \leftarrow \mathsf{CCom}(\mathsf{crs}, m)$ samples a commitment $c$ and an opening $d$ for a message $m$. We require that the commitment be *hiding*; i.e. that the adversary have negligible advantage in distinguishing a commitment $c$ to $m$ from a commitment $c'$ to $m'$, even given knowledge of $\mathsf{crs}$ and free choice of $m$ and $m'$.
3. $\mathsf{COpen}(\mathsf{crs}, c, d)$ is a deterministic algorithm that outputs $\perp$ if $d$ not is a valid opening for $c$, or $m$ if $d$ is a valid opening for $c$ and $c$ is a commitment to $m$. We require that the commitment be *binding*; i.e. that it be infeasible for the adversary to find two different valid openings $d$ and $d'$ that cause one commitment $c$ to open to two different messages $m$ and $m'$.
4. $(\mathsf{crs}, \mathsf{td}) \leftarrow \mathsf{CTrapSetup}(1^\lambda)$ samples a CRS with a trapdoor.
5. $(c, t) \leftarrow \mathsf{CTrapCom}(\mathsf{crs}, \mathsf{td})$ computes a fake commitment $c$ with an equivocation trapdoor $t$.
6. $d \leftarrow \mathsf{CEquiv}(\mathsf{crs}, \mathsf{td}, c, t, m)$ computes an opening to $m$ for a trapdoored commitment $c$. We require that for any $m$, the tuple $(\mathsf{crs}, m, c, d)$ produced in this way be indistinguishable from the same tuple produced via $\mathsf{CSetup}$ and $\mathsf{CCom}$. This is *equivocation*.

The classic commitment scheme of Pedersen [Ped91] meets the above criteria with security under the DDH assumption. Specifically, suppose $\mathsf{GroupGen}(1^\lambda)$ is a PPT algorithm that outputs a distribution of cyclic groups relative to which the decisional Diffie-Hellman problem is hard, and suppose that $\mathsf{CRHF} : \{0,1\}^{|q|} \times \{0,1\}^* \to \mathbb{Z}_q$ is a collision resistant hash function family, which is straightforward to construct under the discrete logarithm assumption.

**Algorithm 3.9. Pedersen's Commitment Scheme [Ped91]**

These algorithms are parameterized by $(\mathbb{G}, G, q) = \mathcal{G} \leftarrow \mathsf{GroupGen}(1^\lambda)$.

$\mathsf{CSetup}_\mathcal{G}()$ :
Sample $H \leftarrow \mathbb{G}$ and $k \leftarrow \{0,1\}^{|q|}$ and output $(G, H, k)$.

$\mathsf{CCom}_\mathcal{G}((G, H, k), m)$ :
Sample $d \leftarrow \mathbb{Z}_q$, compute $C := \mathsf{CRHF}_k(m) \cdot G + d \cdot H$, and output $(C, (m, d))$.

$\mathsf{COpen}_\mathcal{G}((G, H, k), C, (m, d))$ :
Output $m$ if $C = \mathsf{CRHF}_k(m) \cdot G + d \cdot H$, or output $\perp$ otherwise.

$\mathsf{CTrapSetup}_\mathcal{G}()$ :
Sample $\tau \leftarrow \mathbb{Z}_q$ and $k \leftarrow \{0,1\}^{|q|}$, compute $H := \tau \cdot G$, and output $(G, H, k)$.

$\mathsf{CTrapCom}_\mathcal{G}((G, H, k), \tau)$ :
Sample $t \leftarrow \mathbb{Z}_q$, compute $C := t \cdot G$, and output $(C, t)$.

$\mathsf{CEquiv}_\mathcal{G}((G, H, k), \tau, C, t, m)$ :
Output $(m, (t - \mathsf{CRHF}_k(m))/\tau)$.

**Claim 3.10.** *If* $\mathsf{GroupGen}(1^\lambda)$ *outputs a distribution of cyclic groups relative to which the decisional Diffie-Hellman problem is hard and* $\mathsf{CRHF} : \{0,1\}^{|q|} \times \{0,1\}^* \to \mathbb{Z}_q$ *is a collision resistant hash function family, then the algorithms* $(\mathsf{CSetup}_\mathcal{G}, \mathsf{CCom}_\mathcal{G}, \mathsf{COpen}_\mathcal{G}, \mathsf{CTrapSetup}_\mathcal{G}, \mathsf{CTrapCom}_\mathcal{G}, \mathsf{CEquiv}_\mathcal{G})$ *constitute an equivocable commitment scheme with perfect hiding and computational binding, given* $(\mathbb{G}, G, q) = \mathcal{G} \leftarrow \mathsf{GroupGen}(1^\lambda)$.

We direct the reader to Pedersen [Ped91] for proof of this claim.

## 3.5 The Modified PVW Scheme, with a Proof of Security

Now we give our SCOT-SIRIA protocol based on PVW OT. We describe it and prove it in terms of general primitives, much as Peikert et al. did, and present a proof of security afterward. We define a simple functionality $[\![\mathcal{F}_{\mathsf{CRS}}^{\mathsf{DME}}]\!]_{\mathsf{PV}}$ that samples common reference strings of the type required by the messy mode of our dual-mode cryptosystem. We also define $[\![\mathcal{F}_{\mathsf{CRS}}^{\mathsf{Com}}]\!]_{\mathsf{PV}}$ that samples common

reference strings of the type required by our equivocable commitment scheme. For the DDH-based instantiations of these primitives, both CRSes comprise sets of uniformly chosen group elements, and the functionalities are easy to realize with identifiable abort, given only one-to-many commitments [CLOS02].

We require one additional primitive, which is an efficiently invertible injective map $\mathsf{Map} : \mathbb{X} \to \mathbb{M}$ from the message domain $\mathbb{X}$ of the protocol to the message domain $\mathbb{M}$ of the dual-mode encryption scheme.

**Protocol 3.11.** $\pi_{\text{SCOT-SIRIA-PV-PVW}}(\mathbb{X}, \lambda_c)$**. Computational SCOT**

This protocol involves two active participants, $\mathcal{P}_\mathsf{A}$ and $\mathcal{P}_\mathsf{B}$, who we refer to as Alice and Bob, an a-priori-unknown number of passive verifiers (collectively denoted $\mathcal{V}$), and the ideal functionalities $[\![\mathcal{F}_{\mathsf{CRS}}^{\mathsf{DME}}]\!]_{\mathsf{PV}}$ and $[\![\mathcal{F}_{\mathsf{CRS}}^{\mathsf{Com}}]\!]_{\mathsf{PV}}$. This protocol is parameterized by the sender's message domain $\mathbb{X}$. It makes use of a dual-mode encryption scheme with a sigma protocol for the language of correct encryptions and an equivocable commitment scheme as defined in section 3.4.

**Initialization:** On receiving $(\texttt{init}, \mathsf{sid})$ from the environment, $\mathcal{P}_i$ for $i \in \{\mathsf{A}, \mathsf{B}\}$ sends $(\texttt{sample}, \mathsf{sid}\|1)$ to both $[\![\mathcal{F}_{\mathsf{CRS}}^{\mathsf{DME}}]\!]_{\mathsf{PV}}$ and $(\texttt{sample}, \mathsf{sid}\|2)$ to $[\![\mathcal{F}_{\mathsf{CRS}}^{\mathsf{Com}}]\!]_{\mathsf{PV}}$ and receives $(\texttt{crs}, \mathsf{sid}\|1, \mathsf{dmecrs})$ and $(\texttt{crs}, \mathsf{sid}\|2, \mathsf{comcrs})$ in response.

**OT:**

1. On receiving $(\texttt{choose}, \mathsf{sid}, \beta)$ from the environment, $\mathcal{P}_\mathsf{B}$ samples $(\mathsf{pk}, \mathsf{sk}) \leftarrow \mathsf{DMKeyGen}(\mathsf{dmecrs}, \beta)$ and broadcasts $(\texttt{dmepk}, \mathsf{sid}, \mathsf{pk})$ to all parties.

2. On receiving $(\texttt{dmepk}, \mathsf{sid}, \mathsf{pk})$, $\mathcal{P}_\mathsf{A}$ outputs $(\texttt{choice-made}, \mathsf{sid})$ to the environment. On also receiving $(\texttt{messages}, \mathsf{sid}, \alpha^0, \alpha^1)$ from the environment, $\mathcal{P}_\mathsf{A}$ samples $r^b \leftarrow \{0,1\}^{\lambda_c}$ and computes[a]

$$\tilde{\alpha}^b \leftarrow \mathsf{DMEnc}(\mathsf{dmecrs}, \mathsf{pk}, b, \mathsf{Map}(\alpha^b); r^b)$$

   for $b \in \{0,1\}$, and broadcasts $(\texttt{transfer}, \mathsf{sid}, \tilde{\alpha}^0, \tilde{\alpha}^1)$ to all parties.

3. On receiving the `transfer` message from $\mathcal{P}_\mathsf{A}$, all parties output $(\texttt{ot-done}, \mathsf{sid})$ to the environment.

4. On receiving the `transfer` message from $\mathcal{P}_\mathsf{A}$, party $\mathcal{P}_\mathsf{B}$ computes

$$\gamma := \mathsf{Map}^{-1}(\mathsf{DMDec}(\mathsf{dmecrs}, \mathsf{sk}, \tilde{\alpha}^\beta))$$

   and outputs $(\texttt{chosen-message}, \mathsf{sid}, \gamma)$ to the environment.

**Opening:**

5. On receiving $(\mathtt{open}, \mathsf{sid})$ from the environment, $\mathcal{P}_\mathsf{A}$ samples

$$(a^b, s^b) \leftarrow \Sigma^1_{\mathsf{DMEnc}}((\mathsf{dmecrs}, \mathsf{pk}, b, \mathsf{Map}(\alpha^b), \tilde{\alpha}^b), r^b) \qquad \text{for } b \in \{0,1\}$$
$$(c, d) \leftarrow \mathsf{CCom}(\mathsf{comcrs}, (a^0, a^1))$$

   and broadcasts $(\mathtt{open\text{-}com}, \mathsf{sid}, \alpha^0, \alpha^1, c)$ to all parties.

6. On receiving the $\mathtt{open\text{-}com}$ message from $\mathcal{P}_\mathsf{A}$, party $\mathcal{P}_\mathsf{B}$ samples

$$e^b \leftarrow \Sigma^2_{\mathsf{DMEnc}}((\mathsf{dmecrs}, \mathsf{pk}, b, \alpha^b, \tilde{\alpha}^b))$$

   for $b \in \{0,1\}$ and broadcasts $(\mathtt{open\text{-}chal}, \mathsf{sid}, e^0, e^1)$ to all parties.[b]

7. On receiving the $\mathtt{open\text{-}chal}$ message from $\mathcal{P}_\mathsf{B}$, party $\mathcal{P}_\mathsf{A}$ computes

$$z^b \leftarrow \Sigma^3_{\mathsf{DMEnc}}((\mathsf{dmecrs}, \mathsf{pk}, b, \mathsf{Map}(\alpha^b), \tilde{\alpha}^b), r^b, s^b, e^b)$$

   for $b \in \{0,1\}$ and broadcasts $(\mathtt{open\text{-}resp}, \mathsf{sid}, z^0, z^1, d)$ to all parties.

8. Upon receiving $(\mathtt{open\text{-}resp}, \mathsf{sid}, z^0, z^1, d)$ from $\mathcal{P}_\mathsf{A}$, all other parties compute

$$(a^0, a^1) := \mathsf{COpen}(\mathsf{crscom}, c, d)$$

   and output $(\mathtt{messages}, \mathsf{sid}, \alpha^0, \alpha^1)$ to the environment if

$$\Sigma^\mathsf{V}_{\mathsf{DMEnc}}((\mathsf{dmecrs}, \mathsf{pk}, b, \mathsf{Map}(\alpha^b), \tilde{\alpha}^b), a^b, e^b, z^b) = 1$$

   for $b \in \{0,1\}$. On the other hand, if either of the aforementioned verification equations does not hold, then the parties output $(\mathtt{abort}, \mathsf{sid}, \mathcal{P}_\mathsf{A})$ to the environment.

**Verification:**

9. If there is an observing verifier $\mathcal{V}$, then upon receiving $(\mathtt{observe}, \mathsf{sid})$ from the environment, $\mathcal{V}$ sends $(\mathtt{observe}, \mathsf{sid})$ to $[\![\mathcal{F}^{\mathsf{DME}}_{\mathsf{CRS}}]\!]_{\mathsf{PV}}$ and $[\![\mathcal{F}^{\mathsf{Com}}_{\mathsf{CRS}}]\!]_{\mathsf{PV}}$. It then receives the broadcast messages and produces outputs to the environment as described in the forgoing protocol.

[a] We assume without loss of generality that the encryption routine requires only security-parameter many random bits.

[b] In the non-programmable global random-oracle model the parties can instead apply the Fiat-Shamir transform [FS86]: they can calculate $(e^0, e^1) \leftarrow \mathsf{RO}(\mathsf{sid}, c)$ instead of receiving $(e^0, e^1)$ from $\mathcal{P}_\mathsf{B}$. This reduces the **Open** phase of the protocol to a single message.

**Theorem 3.12.** *If* $(\mathsf{DMKeyGen}, \mathsf{DMEnc}, \mathsf{DMDec})$ *belong to a two-branch dual-mode encryption scheme the domain* $\mathbb{M}$*, and* $(\mathsf{CCom}, \mathsf{COpen})$ *belong to an equivocable commitment scheme, and* $(\Sigma^1_{\mathsf{DMEnc}}, \Sigma^2_{\mathsf{DMEnc}}, \Sigma^3_{\mathsf{DMEnc}}, \Sigma^\mathsf{V}_{\mathsf{DMEnc}})$ *is a sigma*

*protocol for* $\mathcal{L}_{\mathsf{DMEnc}}$*, and* $\mathsf{Map} : \mathbb{X} \to \mathbb{M}$ *is an efficiently invertible injective map, then* $\pi_{\text{SCOT-SIRIA-PV-PVW}}(\mathbb{X}, \lambda_c)$ *UC-realizes* $[\![\mathcal{F}_{\text{SCOT-SIRIA}}(\mathbb{X})]\!]_{\mathsf{PV}}$ *in the* $([\![\mathcal{F}_{\mathsf{CRS}}^{\mathsf{DME}}]\!]_{\mathsf{PV}}, [\![\mathcal{F}_{\mathsf{CRS}}^{\mathsf{Com}}]\!]_{\mathsf{PV}})$*-hybrid model, in the presence of a malicious adversary that statically corrupts any number of passive verifiers and at most one of the active participants* $\mathcal{P}_A$ *and* $\mathcal{P}_B$*.*

*Proof.* Formally, we will prove that for every malicious adversary $\mathcal{A}$ that statically corrupts $\mathcal{P}_A$, $\mathcal{P}_B$, or neither active participant, there exists a PPT simulator $\mathcal{S}^{\mathcal{A}}_{\text{SCOT-SIRIA-PV-PVW}}$ that uses $\mathcal{A}$ as a black box, such that for every environment $\mathcal{Z}$,

$$\left\{\text{REAL}_{\pi_{\text{SCOT-SIRIA-PV-PVW}}(\mathbb{X},\lambda),\mathcal{A},\mathcal{Z}}\,(\lambda, z)\right\}_{\lambda\in\mathbb{N},z\in\{0,1\}^*}$$
$$\approx_c \left\{\text{IDEAL}_{[\![\mathcal{F}_{\text{SCOT-SIRIA}}(\mathbb{X})]\!]_{\mathsf{PV}},\mathcal{S}^{\mathcal{A}}_{\text{SCOT-SIRIA-PV-PVW}}(\mathbb{X},\lambda),\mathcal{Z}}\,(\lambda, z)\right\}_{\lambda\in\mathbb{N},z\in\{0,1\}^*}$$

There are two main cases that we must address: the case that $\mathcal{P}_A$ is corrupt, the case that $\mathcal{P}_B$ is corrupt. The case that only passive verifiers are corrupt can be treated as a variation of the case that $\mathcal{P}_B$ is corrupt. We will specify two separate simulators, and argue for the indistinguishability of each. The main simulator $\mathcal{S}^{\mathcal{A}}_{\text{SCOT-SIRIA-PV-PVW}}(\mathbb{X}, \lambda)$ simply selects among the two whichever is appropriate, when $\mathcal{A}$ announces its (static) corruptions at the beginning of the experiment. We begin with a simulator against a corrupt $\mathcal{P}_A$. For the moment, assume that $\mathcal{A}$ corrupts only $\mathcal{P}_A$.

**Simulator 3.13.** $\mathcal{S}^{\mathcal{A}}_{\text{SCOT-SIRIA-PV-PVW-Alice}}(\mathbb{X}, \lambda_c)$**. Against Alice**

This simulator has oracle access to the adversary $\mathcal{A}$ that corrupts $\mathcal{P}_A$ and possibly $\mathcal{V}$, and emulates for it an instance of the protocol $\pi_{\text{SCOT-SIRIA-PV-PVW}}(\mathbb{X}, \lambda_c)$ involving the parties $\mathcal{P}_A$, $\mathcal{P}_B$, and $\mathcal{V}$. The simulator forwards all messages from its own environment $\mathcal{Z}$ to $\mathcal{A}$, and vice versa. $\mathcal{S}^{\mathcal{A}}_{\text{SCOT-SIRIA-PV-PVW-Alice}}(\mathbb{X}, \lambda_c)$ interacts with the ideal functionality $[\![\mathcal{F}_{\text{SCOT-SIRIA}}(\mathbb{X})]\!]_{\mathsf{PV}}$ on behalf of $\mathcal{P}_A$, and in the experiment that it emulates for $\mathcal{A}$, it interacts with $\mathcal{A}$ and the corrupt parties on behalf of $\mathcal{P}_B$ and the ideal oracles $[\![\mathcal{F}_{\mathsf{CRS}}^{\mathsf{DME}}]\!]_{\mathsf{PV}}$ and $[\![\mathcal{F}_{\mathsf{CRS}}^{\mathsf{Com}}]\!]_{\mathsf{PV}}$.

**Initialization:** On receiving $(\texttt{sample}, \mathsf{sid}\|1)$ from $\mathcal{P}_A$ on behalf of $[\![\mathcal{F}_{\mathsf{CRS}}^{\mathsf{DME}}]\!]_{\mathsf{PV}}$ and $(\texttt{init-req}, \mathsf{sid}, \mathcal{P}_B)$ directly from $[\![\mathcal{F}_{\text{SCOT-SIRIA}}(\mathbb{X})]\!]_{\mathsf{PV}}$, sample $(\mathsf{dmecrs}, \mathsf{dmetd}) \leftarrow \mathsf{DMSetup}(1^\lambda, 1)$ and send $(\texttt{crs}, \mathsf{sid}\|1, \mathsf{dmecrs})$ to $\mathcal{P}_A$ on behalf of $[\![\mathcal{F}_{\mathsf{CRS}}^{\mathsf{DME}}]\!]_{\mathsf{PV}}$.

On receiving $(\texttt{sample}, \mathsf{sid}\|2)$ from $\mathcal{P}_A$ on behalf of $[\![\mathcal{F}_{\mathsf{CRS}}^{\mathsf{Com}}]\!]_{\mathsf{PV}}$ and $(\texttt{init-req}, \mathsf{sid}, \mathcal{P}_B)$ from $[\![\mathcal{F}_{\text{SCOT-SIRIA}}(\mathbb{X})]\!]_{\mathsf{PV}}$, sample $\mathsf{comcrs} \leftarrow \mathsf{CSetup}(1^\lambda)$ and send $(\texttt{crs}, \mathsf{sid}\|2, \mathsf{comcrs})$ to $\mathcal{P}_A$ on behalf of $[\![\mathcal{F}_{\mathsf{CRS}}^{\mathsf{Com}}]\!]_{\mathsf{PV}}$.

**OT:**

1. On receiving $(\texttt{choice-made}, \mathsf{sid})$ from $[\![\mathcal{F}_{\text{SCOT-SIRIA}}(\mathbb{X})]\!]_{\mathsf{PV}}$ on behalf of $\mathcal{P}_A$, sample

$$(\mathsf{pk}, \mathsf{sk}_0, \mathsf{sk}_1) \leftarrow \mathsf{DMTrapKeyGen}(\mathsf{dmecrs}, \mathsf{dmetd})$$

and broadcast $(\texttt{dmepk}, \mathsf{sid}, \mathsf{pk})$ to all corrupt parties on behalf of $\mathcal{P}_\mathsf{B}$.

2. On receiving $(\texttt{transfer}, \mathsf{sid}, \tilde{\alpha}^0, \tilde{\alpha}^1)$ message from $\mathcal{P}_\mathsf{A}$ via the broadcast channel, compute

$$\alpha^b := \mathsf{Map}^{-1}(\mathsf{DMDec}(\mathsf{dmecrs}, \mathsf{sk}^b, \tilde{\alpha}^b))$$

for $b \in \{0,1\}$ and send $(\texttt{messages}, \mathsf{sid}, \alpha^0, \alpha^1)$ to $[\![\mathcal{F}_{\mathsf{SCOT\text{-}SIRIA}}(\mathbb{X})]\!]_{\mathsf{PV}}$ on behalf of $\mathcal{P}_\mathsf{A}$.

**Opening:**

3. On receiving $(\texttt{open-com}, \mathsf{sid}, \alpha^{0\prime}, \alpha^{1\prime}, c)$ from $\mathcal{P}_\mathsf{A}$ via the broadcast channel, sample

$$e^b \leftarrow \Sigma^2_{\mathsf{DMEnc}}((\mathsf{dmecrs}, \mathsf{pk}, b, \alpha^b, \tilde{\alpha}^b))$$

for $b \in \{0,1\}$ and broadcast $(\texttt{open-chal}, \mathsf{sid}, e^0, e^1)$ to all corrupt parties on behalf of $\mathcal{P}_\mathsf{B}$.

4. On receiving $(\texttt{open-resp}, \mathsf{sid}, z^0, z^1, d)$ from $\mathcal{P}_\mathsf{A}$ via the broadcast channel, compute

$$(a^0, a^1) := \mathsf{COpen}(\mathsf{crscom}, c, d)$$

and then verify whether

$$\Sigma^\mathsf{V}_{\mathsf{DMEnc}}((\mathsf{dmecrs}, \mathsf{pk}, b, \mathsf{Map}(\alpha^b), \tilde{\alpha}^b), a^b, e^b, z^b) = 1$$

for $b \in \{0,1\}$. If these equalities hold, send $(\texttt{open}, \mathsf{sid})$ to $[\![\mathcal{F}_{\mathsf{SCOT\text{-}SIRIA}}(\mathbb{X})]\!]_{\mathsf{PV}}$ on behalf of $\mathcal{P}_\mathsf{A}$. Otherwise, send $(\texttt{abort}, \mathsf{sid})$ directly to $[\![\mathcal{F}_{\mathsf{SCOT\text{-}SIRIA}}(\mathbb{X})]\!]_{\mathsf{PV}}$.

**Verification:**

5. If there is a corrupt observing verifier $\mathcal{V}$, then upon receiving either $(\texttt{observe}, \mathsf{sid}\|1)$ from $\mathcal{V}$ on behalf of $[\![\mathcal{F}^{\mathsf{DME}}_{\mathsf{CRS}}]\!]_{\mathsf{PV}}$ or $(\texttt{observe}, \mathsf{sid}\|2)$ from $\mathcal{V}$ on behalf of $[\![\mathcal{F}^{\mathsf{Com}}_{\mathsf{CRS}}]\!]_{\mathsf{PV}}$, send $(\texttt{observe}, \mathsf{sid})$ to $[\![\mathcal{F}_{\mathsf{SCOT\text{-}SIRIA}}(\mathbb{X})]\!]_{\mathsf{PV}}$ on behalf of $\mathcal{V}$.

Our argument will proceed via a sequence of hybrid experiments, beginning with the real world. We have:

$$\mathcal{H}_0 = \left\{\mathrm{REAL}_{\pi_{\mathsf{SCOT\text{-}SIRIA\text{-}PV\text{-}PVW}}(\mathbb{X},\lambda),\mathcal{A},\mathcal{Z}}(\lambda, z)\right\}_{\lambda\in\mathbb{N}, z\in\{0,1\}^*}$$

**Hybrid $\mathcal{H}_1$.** This hybrid experiment replaces all of the individual honest parties and ideal functionalities in $\mathcal{H}_0$ with a single simulator machine $\mathcal{S}$ that runs their code and interacts with the adversary, environment, and corrupt parties on their behalf. Since $\mathcal{S}$ interacts with the adversarial entities on behalf of the ideal functionalities, it learns any values they receive or that are defined by their

internal state. This is a purely syntactical change, and so it must be the case that $\mathcal{H}_1 = \mathcal{H}_0$.

**Hybrid $\mathcal{H}_2$.** In this hybrid, $\mathcal{S}$ samples $(\mathsf{dmecrs}, \mathsf{dmetd}) \leftarrow \mathsf{DMSetup}(1^\lambda, 1)$ and outputs $\mathsf{dmecrs}$ on behalf of $[\![\mathcal{F}_{\mathsf{CRS}}^{\mathsf{DME}}]\!]_{\mathsf{PV}}$ when queried. That is, it samples $\mathsf{dmecrs}$ from the *decryptable* CRS distribution, rather than the *messy* CRS distribution (as it did in $\mathcal{H}_1$). Under the computational indistinguishability of modes for dual-mode encryption schemes, $\mathcal{H}_2 \approx_{\mathrm{c}} \mathcal{H}_1$

**Hybrid $\mathcal{H}_3$.** $\mathcal{S}$ samples $(\mathsf{pk}, \mathsf{sk}_0, \mathsf{sk}_1) \leftarrow \mathsf{DMTrapKeyGen}(\mathsf{dmecrs}, \mathsf{dmetd})$ and transmits $\mathsf{pk}$ on behalf of $\mathcal{P}_{\mathsf{B}}$ instead of generating $\mathsf{pk}$ via $\mathsf{DMKeyGen}$ as in $\mathcal{H}_2$. Under the statistical indistinguishability of keys generated with and without the trapdoor (but using the same CRS distribution), $\mathcal{H}_3 \approx_{\mathrm{s}} \mathcal{H}_2$

**Hybrid $\mathcal{H}_4$.** This final hybrid differs from $\mathcal{H}_3$ in the following way: $\mathcal{S}$ no longer acts on behalf of any honest parties. Instead, $\mathcal{S}^{\mathcal{A}}_{\mathsf{SCOT\text{-}SIRIA\text{-}PV\text{-}PVW\text{-}Alice}}(\mathbb{X}, \lambda)$ is fully implemented in $\mathcal{H}_4$ (that is, $\mathcal{S} = \mathcal{S}^{\mathcal{A}}_{\mathsf{SCOT\text{-}SIRIA\text{-}PV\text{-}PVW\text{-}Alice}}(\mathbb{X}, \lambda)$), and the experiment now incorporates $[\![\mathcal{F}_{\mathsf{SCOT\text{-}SIRIA}}(\mathbb{X})]\!]_{\mathsf{PV}}$. The honest parties run dummy-party code as is standard for ideal-world experiments in the UC model, and $\mathcal{S}$ speaks to $[\![\mathcal{F}_{\mathsf{SCOT\text{-}SIRIA}}(\mathbb{X})]\!]_{\mathsf{PV}}$ on behalf of corrupt parties.

We argue that $\mathcal{H}_4 \approx_{\mathrm{c}} \mathcal{H}_3$. Observe first that in both $\mathcal{H}_4$ and $\mathcal{H}_3$, $\mathcal{P}_{\mathsf{B}}$ outputs the decryption of $\tilde{\alpha}^\beta$ in the **OT** phase, and by the correctness of the dual-mode cryptosystem this is the value that $\mathcal{P}_{\mathsf{A}}$ encrypted. The distributions of this value are identical in the two hybrids. Observe second that the simulator in $\mathcal{H}_4$ performs exactly the same verification steps in the **Opening** phase as $\mathcal{P}_{\mathsf{B}}$ and the passive verifiers $\mathcal{V}$ perform in $\mathcal{H}_3$, and it instructs $[\![\mathcal{F}_{\mathsf{SCOT\text{-}SIRIA}}(\mathbb{X})]\!]_{\mathsf{PV}}$ to abort if and only if they fail. Thus the distributions of aborts are the same in both hybrids. However, in $\mathcal{H}_4$, $\mathcal{P}_{\mathsf{B}}$ and the passive verifiers output the decryptions of $\tilde{\alpha}^0$ and $\tilde{\alpha}^1$ in the **Opening** phase, because these are the values delivered to them by $[\![\mathcal{F}_{\mathsf{SCOT\text{-}SIRIA}}(\mathbb{X})]\!]_{\mathsf{PV}}$, whereas in $\mathcal{H}_3$ they output the values $\alpha^0$ and $\alpha^1$ transmitted by $\mathcal{P}_{\mathsf{A}}$ in step 5 of $\pi_{\mathsf{SCOT\text{-}SIRIA\text{-}PV\text{-}PVW}}(\mathbb{X}, \lambda_{\mathsf{c}})$. Thus the adversary can distinguish the two hybrids by transmitting a value $\alpha^b$ that is not the decryption of $\tilde{\alpha}^b$ for some $b \in \{0, 1\}$, and avoiding an abort.

We argue that the case cannot occur with noticeable probability, by constructing a rewinding reduction that can use any distinguishing adversary to break the binding property of the equivocable commitment scheme. Our reduction constructs a variant of $\mathcal{H}_4$ for $(\mathcal{A}, \mathcal{Z})$ in which the reduction itself plays the role of $\mathcal{S}$, and follows the code of $\mathcal{S}$ exactly until the protocol is complete. The reduction then rewinds[8] the experiment to step 6 of the protocol, samples a fresh pair of challenges $e^{0\prime}$ and $e^{1\prime}$, and completes the protocol to receive $z^{0\prime}$, $z^{1\prime}$, and the decommitment $d'$ which allows it to compute $a^{0\prime}$ and $a^{1\prime}$. The remainder of our argument follows from two claims. First:

[8] Again, the reduction can rewind the environment because it runs the environment as a subroutine. This distinguishes it from the simulator, which has no such power.

**Claim 3.14.** *Suppose that in* $\mathcal{H}_4$*,*

$$\Sigma^{\mathsf{V}}_{\mathsf{DMEnc}}((\mathsf{dmecrs}, \mathsf{pk}, b, \mathsf{Map}(\alpha^b), \tilde{\alpha}^b), a^b, e^b, z^b) = 1 \\ \wedge \quad (\mathsf{dmecrs}, \mathsf{pk}, b, \mathsf{Map}(\alpha^b), \tilde{\alpha}^b) \notin \mathcal{L}_{\mathsf{DMEnc}}$$

*with probability* $p$ *for some* $b \in \{0,1\}$*. When the reduction runs the experiment, then rewinds it to the point that* $e^b$ *is sampled and runs it to completion a second time, with probability* $p^2 - p/q$ *the former event occurs and in addition*

$$\Sigma^{\mathsf{V}}_{\mathsf{DMEnc}}((\mathsf{dmecrs}, \mathsf{pk}, b, \mathsf{Map}(\alpha^b), \tilde{\alpha}^b), a^{b\prime}, e^{b\prime}, z^{b\prime}) = 1 \quad \wedge \quad e^{b\prime} \neq e^b$$

This follows directly from the generalized forking lemma of Bellare and Neven [BN06]. Combining this claim with lemma 3.6 and the fact that the statement $(\mathsf{dmecrs}, \mathsf{pk}, b, \mathsf{Map}(\alpha^b), \tilde{\alpha}^b)$ is fixed *before* the rewinding point and cannot be changed by the adversary yields our second claim:

**Claim 3.15.** *Suppose that in* $\mathcal{H}_4$*,*

$$\Sigma^{\mathsf{V}}_{\mathsf{DMEnc}}((\mathsf{dmecrs}, \mathsf{pk}, b, \mathsf{Map}(\alpha^b), \tilde{\alpha}^b), a^b, e^b, z^b) = 1 \\ \wedge \quad (\mathsf{dmecrs}, \mathsf{pk}, b, \mathsf{Map}(\alpha^b), \tilde{\alpha}^b) \notin \mathcal{L}_{\mathsf{DMEnc}}$$

*with probability* $p$ *for some* $b \in \{0,1\}$*. When the reduction runs the experiment, then rewinds it to the point that* $e^b$ *is sampled and runs it to completion a second time, the adversary produces* $d$ *and* $d'$ *such that*

$$(a^0, a^1) := \mathsf{COpen}(\mathsf{crscom}, c, d) \quad \wedge \quad (a^{0\prime}, a^{1\prime}) := \mathsf{COpen}(\mathsf{crscom}, c, d') \quad \wedge \quad a^{b\prime} \neq a^b$$

*with probability* $p^2 - p/q$*.*

Because we have assumed that $(\mathsf{CCom}, \mathsf{COpen})$ belong to a commitment scheme that is binding against computationally-bounded adversaries, claim 3.15 implies that $p^2 - p/q \in \mathrm{negl}(\lambda)$, which in turn implies that $p$ must be negligible, and that $\mathcal{H}_4 \approx_{\mathrm{c}} \mathcal{H}_3$.

We now have

$$\mathcal{H}_4 = \left\{ \textsc{Ideal}_{[\![\mathcal{F}_{\mathsf{SCOT\text{-}SIRIA}}(\mathbb{X})]\!]_{\mathsf{PV}}, \mathcal{S}^{\mathcal{A}}_{\mathsf{SCOT\text{-}SIRIA\text{-}PV\text{-}PVW\text{-}Alice}}(\mathbb{X},\lambda), \mathcal{Z}}(\lambda, z) \right\}_{\lambda \in \mathbb{N}, z \in \{0,1\}^*}$$

and by transitivity we also have

$$\mathcal{H}_4 \approx_{\mathrm{c}} \mathcal{H}_0 = \left\{ \textsc{Real}_{\pi_{\mathsf{SCOT\text{-}SIRIA\text{-}PV\text{-}PVW}}(\mathbb{X},\lambda), \mathcal{A}, \mathcal{Z}}(\lambda, z) \right\}_{\lambda \in \mathbb{N}, z \in \{0,1\}^*}$$

which completes the first case.

Next we give a simulator against a corrupt $\mathcal{P}_{\mathsf{B}}$. Assume henceforth that $\mathcal{A}$ corrupts only $\mathcal{P}_{\mathsf{B}}$.

**Simulator 3.16.** $\mathcal{S}^{\mathcal{A}}_{\mathsf{SCOT\text{-}SIRIA\text{-}PV\text{-}PVW\text{-}Bob}}(\mathbb{X}, \lambda_c)$**. Against Bob**

This simulator has oracle access to the adversary $\mathcal{A}$ that corrupts $\mathcal{P}_B$ and possibly $\mathcal{V}$, and emulates for it an instance of the protocol $\pi_{\mathsf{SCOT\text{-}SIRIA\text{-}PV\text{-}PVW}}(\mathbb{X}, \lambda_c)$ involving the parties $\mathcal{P}_A, \mathcal{P}_B, \mathcal{V}$. The simulator forwards all messages from its own environment $\mathcal{Z}$ to $\mathcal{A}$, and vice versa. $\mathcal{S}^{\mathcal{A}}_{\mathsf{SCOT\text{-}SIRIA\text{-}PV\text{-}PVW\text{-}Bob}}(\mathbb{X}, \lambda_c)$ interacts with the ideal functionality $[\![\mathcal{F}_{\mathsf{SCOT\text{-}SIRIA}}(\mathbb{X})]\!]_{\mathsf{PV}}$ on behalf of $\mathcal{P}_B$, and in the experiment that it emulates for $\mathcal{A}$, it interacts with $\mathcal{A}$ and the corrupt parties on behalf of $\mathcal{P}_A$ and the ideal oracles $[\![\mathcal{F}^{\mathsf{DME}}_{\mathsf{CRS}}]\!]_{\mathsf{PV}}$ and $[\![\mathcal{F}^{\mathsf{Com}}_{\mathsf{CRS}}]\!]_{\mathsf{PV}}$.

**Initialization:** On receiving $(\mathtt{sample}, \mathsf{sid}\|1)$ from $\mathcal{P}_B$ on behalf of $[\![\mathcal{F}^{\mathsf{DME}}_{\mathsf{CRS}}]\!]_{\mathsf{PV}}$ and $(\mathtt{init\text{-}req}, \mathsf{sid}, \mathcal{P}_A)$ directly from $[\![\mathcal{F}_{\mathsf{SCOT\text{-}SIRIA}}(\mathbb{X})]\!]_{\mathsf{PV}}$, sample $(\mathsf{dmecrs}, \mathsf{dmetd}) \leftarrow \mathsf{DMSetup}(1^\lambda, 0)$ and send $(\mathtt{crs}, \mathsf{sid}\|1, \mathsf{dmecrs})$ to $\mathcal{P}_B$ on behalf of $[\![\mathcal{F}^{\mathsf{DME}}_{\mathsf{CRS}}]\!]_{\mathsf{PV}}$.

On receiving $(\mathtt{sample}, \mathsf{sid}\|2)$ from $\mathcal{P}_B$ on behalf of $[\![\mathcal{F}^{\mathsf{Com}}_{\mathsf{CRS}}]\!]_{\mathsf{PV}}$ and $(\mathtt{init\text{-}req}, \mathsf{sid}, \mathcal{P}_A)$ from $[\![\mathcal{F}_{\mathsf{SCOT\text{-}SIRIA}}(\mathbb{X})]\!]_{\mathsf{PV}}$, sample $(\mathsf{comcrs}, \mathsf{comtd}) \leftarrow \mathsf{CTrapSetup}(1^\lambda)$ and send $(\mathtt{crs}, \mathsf{sid}\|2, \mathsf{comcrs})$ to $\mathcal{P}_B$ on behalf of $[\![\mathcal{F}^{\mathsf{Com}}_{\mathsf{CRS}}]\!]_{\mathsf{PV}}$.

**OT:**

1. On receiving $(\mathtt{dmepk}, \mathsf{sid}, \mathsf{pk})$ from $\mathcal{P}_B$, compute

$$\beta := 1 - \mathsf{DMFindMessy}(\mathsf{dmecrs}, \mathsf{dmetd}, \mathsf{pk})$$

   and send $(\mathtt{choose}, \mathsf{sid}, \beta)$ to $[\![\mathcal{F}_{\mathsf{SCOT\text{-}SIRIA}}(\mathbb{X})]\!]_{\mathsf{PV}}$ on behalf of $\mathcal{P}_B$.

2. On receiving $(\mathtt{chosen\text{-}message}, \mathsf{sid}, \gamma)$ from $[\![\mathcal{F}_{\mathsf{SCOT\text{-}SIRIA}}(\mathbb{X})]\!]_{\mathsf{PV}}$ on behalf of $\mathcal{P}_B$, compute

$$\tilde{\alpha}^{\beta} \leftarrow \mathsf{DMEnc}(\mathsf{dmecrs}, \mathsf{pk}, \beta, \mathsf{Map}(\alpha^{\beta}))$$
$$\tilde{\alpha}^{1-\beta} \leftarrow \mathsf{DMEnc}(\mathsf{dmecrs}, \mathsf{pk}, 1-\beta, \mu)$$

   where $\mu \in \mathbb{M}$ is an arbitrary element of the message space of $\mathsf{DMEnc}$, and broadcast $(\mathtt{transfer}, \mathsf{sid}, \tilde{\alpha}^0, \tilde{\alpha}^1)$ to all parties on behalf of $\mathcal{P}_A$.

**Opening:**

3. On receiving $(\mathtt{messages}, \mathsf{sid}, \alpha^0, \alpha^1)$ from $[\![\mathcal{F}_{\mathsf{SCOT\text{-}SIRIA}}(\mathbb{X})]\!]_{\mathsf{PV}}$ on behalf of $\mathcal{P}_B$, compute

$$(c, t) \leftarrow \mathsf{CTrapCom}(\mathsf{comcrs}, \mathsf{comtd})$$

   broadcast $(\mathtt{open\text{-}com}, \mathsf{sid}, \alpha^0, \alpha^1, c)$ to all parties on behalf of $\mathcal{P}_A$.

4. On receiving $(\texttt{open-chal}, \mathsf{sid}, e^0, e^1)$ from $\mathcal{P}_\mathsf{B}$ via the broadcast channel, compute

$$(a^b, z^b) \leftarrow \Sigma^{\mathsf{S}}_{\mathsf{DMEnc}}((\mathsf{dmecrs}, \mathsf{pk}, b, \mathsf{Map}(\alpha^b), \tilde{\alpha}^b), e^b)$$

for $b \in \{0,1\}$ and then compute

$$d \leftarrow \mathsf{CEquiv}(\mathsf{comcrs}, \mathsf{comtd}, c, t, (a^0, a^1))$$

and broadcast $(\texttt{open-resp}, \mathsf{sid}, z^0, z^1, d)$ to all parties on behalf of $\mathcal{P}_\mathsf{A}$

**Verification:**

5. If there is a corrupt observing verifier $\mathcal{V}$, then upon receiving either $(\texttt{observe}, \mathsf{sid}\|1)$ from $\mathcal{V}$ on behalf of $[\![\mathcal{F}^{\mathsf{DME}}_{\mathsf{CRS}}]\!]_{\mathsf{PV}}$ or $(\texttt{observe}, \mathsf{sid}\|2)$ from $\mathcal{V}$ on behalf of $[\![\mathcal{F}^{\mathsf{Com}}_{\mathsf{CRS}}]\!]_{\mathsf{PV}}$, send $(\texttt{observe}, \mathsf{sid})$ to $[\![\mathcal{F}_{\mathsf{SCOT\text{-}SIRIA}}(\mathbb{X})]\!]_{\mathsf{PV}}$ on behalf of $\mathcal{V}$.

Again, our argument will proceed via a sequence of hybrid experiments, beginning with the real world. We have:

$$\mathcal{H}_0 = \left\{\text{REAL}_{\pi_{\mathsf{SCOT\text{-}SIRIA\text{-}PV\text{-}PVW}}(\mathbb{X},\lambda),\mathcal{A},\mathcal{Z}}(\lambda, z)\right\}_{\lambda\in\mathbb{N}, z\in\{0,1\}^*}$$

**Hybrid $\mathcal{H}_1$.** This hybrid experiment replaces all of the individual honest parties and ideal functionalities in $\mathcal{H}_0$ with a single simulator machine $\mathcal{S}$ that runs their code and interacts with the adversary, environment, and corrupt parties on their behalf. Since $\mathcal{S}$ interacts with the adversarial entities on behalf of the ideal functionalities, it learns any values they receive or that are defined by their internal state. This is a purely syntactical change, and so it must be the case that $\mathcal{H}_1 = \mathcal{H}_0$.

**Hybrid $\mathcal{H}_2$.** In this hybrid, $\mathcal{S}$ samples $(\mathsf{dmecrs}, \mathsf{dmetd}) \leftarrow \mathsf{DMSetup}(1^\lambda, 0)$ and outputs $\mathsf{dmecrs}$ on behalf of $[\![\mathcal{F}^{\mathsf{DME}}_{\mathsf{CRS}}]\!]_{\mathsf{PV}}$ when queried. This does not change the distribution of $\mathsf{dmecrs}$. $\mathcal{S}$ also samples $(\mathsf{comcrs}, \mathsf{comtd}) \leftarrow \mathsf{CTrapSetup}(1^\lambda)$ and outputs $\mathsf{comcrs}$ on behalf of $[\![\mathcal{F}^{\mathsf{Com}}_{\mathsf{CRS}}]\!]_{\mathsf{PV}}$ when queried. In the **Opening** phase, $\mathcal{S}$ samples $(c,t) \leftarrow \mathsf{CTrapCom}(\mathsf{comcrs}, \mathsf{comtd})$ and then samples $d \leftarrow \mathsf{CEquiv}(\mathsf{comcrs}, \mathsf{comtd}, c, t, (a^0, a^1))$ rather than generating $c$ and $d$ as in step 5 of $\pi_{\mathsf{SCOT\text{-}SIRIA\text{-}PV\text{-}PVW}}$. Notice that the values $a^0$ and $a^1$ to which the commitment opens remain unchanged. By the equivocation property of the commitment scheme, $\mathcal{H}_2 \approx_\text{c} \mathcal{H}_1$.

**Hybrid $\mathcal{H}_3$.** In this hybrid, $\mathcal{S}$ does not calculate $a^0$ and $a^1$ using $\Sigma^1_{\mathsf{DMEnc}}$ or $z^0$ and $z^1$ using $\Sigma^3_{\mathsf{DMEnc}}$, but instead waits until *after* receiving $e^0$ and $e^1$ from $\mathcal{P}_\mathsf{B}$, and then calculates $(a^b, z^b) \leftarrow \Sigma^{\mathsf{S}}_{\mathsf{DMEnc}}((\mathsf{dmecrs}, \mathsf{pk}, b, \mathsf{Map}(\alpha^b), \tilde{\alpha}^b), e^b)$ for $b \in \{0,1\}$. By the special honest verifier zero knowledge of the sigma protocol, $\mathcal{H}_3 = \mathcal{H}_2$. Notice that the simulator no longer needs to use the randomness of the dual-mode cryptosystem to generate the sigma protocol.

**Hybrid $\mathcal{H}_4$.** In $\mathcal{H}_4$, $\mathcal{S}$ computes $\beta := 1 - \mathsf{DMFindMessy}(\mathsf{dmecrs}, \mathsf{dmetd}, \mathsf{pk})$ after it receives $\mathsf{pk}$ from $\mathcal{P}_\mathsf{B}$, and then computes $\tilde{\alpha}^{1-\beta} \leftarrow \mathsf{DMEnc}(\mathsf{dmecrs}, \mathsf{pk}, 1-\beta, \mu)$ where $\mu \in \mathbb{M}$ is an arbitrary element of the message space, rather than computing $\tilde{\alpha}^{1-\beta}$ using $\alpha^{1-\beta}$ as in step 2 of the protocol. By the fact that $\mathsf{DMFindMessy}$ identifies the messy branch with overwhelming probability, and the fact that encryptions under messy branches contain no information about encrypted values, $\mathcal{H}_4 \approx_\mathrm{s} \mathcal{H}_3$.

**Hybrid $\mathcal{H}_5$.** This final hybrid differs from $\mathcal{H}_4$ in the following way: $\mathcal{S}$ no longer acts on behalf of any honest parties. Instead, $\mathcal{S}^{\mathcal{A}}_{\mathsf{SCOT\text{-}SIRIA\text{-}PV\text{-}PVW\text{-}Bob}}(\mathbb{X}, \lambda)$ is fully implemented in $\mathcal{H}_5$ (that is, $\mathcal{S} = \mathcal{S}^{\mathcal{A}}_{\mathsf{SCOT\text{-}SIRIA\text{-}PV\text{-}PVW\text{-}Bob}}(\mathbb{X}, \lambda)$), and the experiment now incorporates $[\![\mathcal{F}_{\mathsf{SCOT\text{-}SIRIA}}(\mathbb{X})]\!]_{\mathsf{PV}}$. The honest parties run dummy-party code as is standard for ideal-world experiments in the UC model, and $\mathcal{S}$ speaks to $[\![\mathcal{F}_{\mathsf{SCOT\text{-}SIRIA}}(\mathbb{X})]\!]_{\mathsf{PV}}$ on behalf of corrupt parties. This is purely a syntactical change, and so $\mathcal{H}_5 = \mathcal{H}_4$.

We now have

$$\mathcal{H}_5 = \left\{ \textsc{Ideal}_{[\![\mathcal{F}_{\mathsf{SCOT\text{-}SIRIA}}(\mathbb{X})]\!]_{\mathsf{PV}}, \mathcal{S}^{\mathcal{A}}_{\mathsf{SCOT\text{-}SIRIA\text{-}PV\text{-}PVW\text{-}Bob}}(\mathbb{X}, \lambda), \mathcal{Z}}(\lambda, z) \right\}_{\lambda \in \mathbb{N}, z \in \{0,1\}^*}$$

and by transitivity we also have

$$\mathcal{H}_5 \approx_\mathrm{c} \mathcal{H}_0 = \left\{ \textsc{Real}_{\pi_{\mathsf{SCOT\text{-}SIRIA\text{-}PV\text{-}PVW}}(\mathbb{X}, \lambda), \mathcal{A}, \mathcal{Z}}(\lambda, z) \right\}_{\lambda \in \mathbb{N}, z \in \{0,1\}^*}$$

which completes the second case.

The remaining case is the one in which only observing verifiers are corrupt. In this case, the simulator works in a similar manner to $\mathcal{S}_{\mathsf{SCOT\text{-}SIRIA\text{-}PV\text{-}PVW\text{-}Bob}}$, with the messages of $\mathcal{P}_\mathsf{B}$ being generated using the honest protocol code of $\mathcal{P}_\mathsf{B}$ and a random $\beta$, and both ciphertexts $\tilde{\alpha}^0$ and $\tilde{\alpha}^1$ being generated as encryptions of an arbitrary $\mu \in \mathbb{M}$. The argument for this final simulator is similar to our argument for $\mathcal{S}_{\mathsf{SCOT\text{-}SIRIA\text{-}PV\text{-}PVW\text{-}Bob}}$, and by conjunction of the cases, the theorem holds. □

**Corollary 3.17.** *If* $\mathsf{GroupGen}(1^\lambda)$ *outputs a distribution of cyclic groups relative to which the decisional Diffie-Hellman problem is hard, and* $\mathsf{Map} : \mathbb{X} \to \mathbb{G}$ *is an invertible injective map, then there exists a protocol that UC-realizes* $[\![\mathcal{F}_{\mathsf{SCOT\text{-}SIRIA}}(\mathbb{X})]\!]_{\mathsf{PV}}$ *in the* $([\![\mathcal{F}^{\mathsf{DME}}_{\mathsf{CRS}}]\!]_{\mathsf{PV}}, [\![\mathcal{F}^{\mathsf{Com}}_{\mathsf{CRS}}]\!]_{\mathsf{PV}})$*-hybrid model given* $(\mathbb{G}, G, q) = \mathcal{G} \leftarrow \mathsf{GroupGen}(1^\lambda)$*, in the presence of a malicious adversary that statically corrupts any number of passive verifiers and at most one of the active participants* $\mathcal{P}_\mathsf{A}$ *and* $\mathcal{P}_\mathsf{B}$*.*

*Proof.* This corollary arises by conjunction of theorem 3.12 and claims 3.5, 3.8, and 3.10; that is, it arises by instantiating the dual-mode cryptosystem with Peikert et al.'s DDH-based scheme (algorithm 3.4), instantiating the commitment scheme with Pedersen's commitment scheme (algorithm 3.9), and using the appropriate variant of Schnorr's sigma protocol (algorithm 3.7). □

**On CRS Reuse.** We note that as written, our protocol samples individual CRSes for each session, which is wasteful in practice. This is a notational convenience. The CRSes can be sampled once and safely reused, but the simulator requires the CRS to be programmed differently when Alice is corrupt and when Bob is corrupt, which means that each pair of parties will require two CRSes, one for each direction in which an OT can be performed.

# 4 Extending SCOT-SIRIA

In this section, we describe how to construct OT extension with sender openability, starting from OT with the same property. Since OT extension is only a meaningful improvement over plain OT where concrete efficiency is concerned, we allow ourselves the use of a programmable random oracle. We begin with a functionality $\mathcal{F}_{\mathsf{SCOTE\text{-}SIRIA}}$, which is in essence a vectorized version of $\mathcal{F}_{\mathsf{SCOT\text{-}SIRIA}}$. In the rest of this paper, we assume that any time a pair of parties instantiates $\mathcal{F}_{\mathsf{SCOT\text{-}SIRIA}}$ multiple times (concurrently or sequentially), a single instance of $\mathcal{F}_{\mathsf{SCOTE\text{-}SIRIA}}$ can be substituted.

**Functionality 4.1.** $\mathcal{F}_{\mathsf{SCOTE\text{-}SIRIA}}$**: SCOT Extension**

This functionality interacts with two active participants, $\mathcal{P}_{\mathsf{A}}$ and $\mathcal{P}_{\mathsf{B}}$, who we refer to as Alice and Bob, and with the ideal adversary $\mathcal{S}$.

**Initialization:** On receiving $(\texttt{init}, \mathsf{sid})$ from $\mathcal{P}_i$ for $i \in \{\mathsf{A}, \mathsf{B}\}$ such that $\mathcal{P}_{\mathsf{B}} \| \mathcal{P}_{\mathsf{A}} \| \mathsf{sid}' = \mathsf{sid}$ for some fresh $\mathsf{sid}'$, send $(\texttt{init-req}, \mathsf{sid}, \mathcal{P}_i)$ to all parties. On receiving $(\texttt{init}, \mathsf{sid})$ from both Alice and Bob, send $(\texttt{initialized}, \mathsf{sid})$ to all parties and store $(\texttt{initialized}, \mathsf{sid})$ in memory.

**SCOT Extension:** On receiving $(\texttt{choose}, \mathsf{sid}, \mathsf{eid}, \{\mathbb{X}_1, \ldots, \mathbb{X}_{\ell_{\mathsf{OTE}}}\}, \boldsymbol{\beta})$ from Bob, if $\mathsf{eid}$ is fresh, and $\boldsymbol{\beta} \in \{0,1\}^{\ell_{\mathsf{OTE}}}$, and $(\texttt{initialized}, \mathsf{sid})$ exists in memory, but no record of the form $(\texttt{choice}, \mathsf{sid}, \mathsf{eid}, *)$ exists in memory, then store $(\texttt{choice}, \mathsf{sid}, \mathsf{eid}, \{\mathbb{X}_1, \ldots, \mathbb{X}_{\ell_{\mathsf{OTE}}}\}, \boldsymbol{\beta})$ and send $(\texttt{choice-made}, \mathsf{sid}, \mathsf{eid}, \{\mathbb{X}_1, \ldots, \mathbb{X}_{\ell_{\mathsf{OTE}}}\})$ to all parties. On subsequently receiving $(\texttt{messages}, \mathsf{sid}, \mathsf{eid}, \boldsymbol{\alpha}^0, \boldsymbol{\alpha}^1)$ from Alice such that $\boldsymbol{\alpha}^b \in \mathbb{X}_1 \times \ldots \times \mathbb{X}_{\ell_{\mathsf{OTE}}}$ for $b \in \{0,1\}$, store her message in memory and:

1. Let $\boldsymbol{\gamma}_i := \boldsymbol{\alpha}_i^{\boldsymbol{\beta}_i}$ for every $i \in [\ell_{\mathsf{OTE}}]$.
2. Send $(\texttt{chosen-messages}, \mathsf{sid}, \mathsf{eid}, \boldsymbol{\gamma})$ to Bob.
3. Send $(\texttt{ote-req}, \mathsf{sid}, \mathsf{eid}, \{\mathbb{X}_1, \ldots, \mathbb{X}_{\ell_{\mathsf{OTE}}}\})$ to $\mathcal{S}$.
4. If no active participants are corrupt or $\mathcal{S}$ sends $(\texttt{proceed}, \mathsf{sid}, \mathsf{eid})$ in response, then send $(\texttt{ote-done}, \mathsf{sid}, \mathsf{eid}, \{\mathbb{X}_1, \ldots, \mathbb{X}_{\ell_{\mathsf{OTE}}}\})$ to all parties.
5. If $\mathcal{P}_c$ for some $c \in \{\mathsf{A}, \mathsf{B}\}$ is corrupt and $\mathcal{S}$ sends $(\texttt{abort}, \mathsf{sid}, \mathsf{eid})$ in response, then send $(\texttt{abort}, \mathsf{sid}, \mathcal{P}_c)$ to all parties, find all records of the

form $(\texttt{messages}, \mathsf{sid}, *, *, *)$ in memory and send them to $\mathcal{S}$, and ignore all future messages with the same session ID.

**Opening:** On receiving $(\texttt{open}, \mathsf{sid})$ from Alice send $(\texttt{open-req}, \mathsf{sid})$ to all parties and to $\mathcal{S}$. If Alice is corrupt and $\mathcal{S}$ responds with $(\texttt{abort}, \mathsf{sid})$, then send $(\texttt{abort}, \mathsf{sid}, \mathcal{P}_\mathsf{A})$ to all parties. If Alice is honest, or $\mathcal{S}$ responds with $(\texttt{proceed}, \mathsf{sid})$, then find all records of the form $(\texttt{messages}, \mathsf{sid}, *, *, *)$, and send them to all parties. Regardless, ignore all future messages with the same session ID.

Now we give a protocol to realize the previous functionality, which is derived from the SoftspokenOT protocol of Roy [Roy22]. Our specific treatment of the SoftspokenOT protocol (i.e. the presentational aspects) is primarily derived from the one of Keller et al. [KOS15]. Unlike some of the other protocols introduced in this paper, our protocol does not merely append to SoftspokenOT, but also makes other minor changes to the original. We annotate these where we think them significant. Perhaps the most noticeable change (outside of the **Opening** and **Verification** phases) is the transmission by $\mathcal{P}_\mathsf{B}$ of images of his seed values under the random oracle in step 1. This change enables a technique that we call *OT extension extension*, which we discussion section 4.1.

**Protocol 4.2.** $\pi_{\mathsf{SCOTE\text{-}SIRIA\text{-}PV}}(\lambda_\mathsf{c}, \lambda_\mathsf{s})$**. SCOT-SIRIA Extension**

This protocol involves two active participants, $\mathcal{P}_\mathsf{A}$ and $\mathcal{P}_\mathsf{B}$, who we refer to as Alice and Bob, an a-priori-unknown number of passive verifiers (collectively denoted $\mathcal{V}$), and the ideal functionalities $[\![\mathcal{F}_{\mathsf{SCOT\text{-}SIRIA}}(\{0,1\}^{\lambda_\mathsf{c}})]\!]_{\mathsf{PV}}$ and $[\![\mathcal{F}_{\mathsf{Com}}]\!]_{\mathsf{PV}}$. In addition, the parties have access to a (local) random oracle with a parametric range, which is specified by its subscript: $\mathsf{RO}_\mathbb{X} : \{0,1\}^* \to \mathbb{X}$. Note that in this protocol, operations are performed with respect to multiple fields, and so we specify in which field operations are performed, where relevant.

**Initialization:**

1. When $\mathcal{P}_\mathsf{B}$ receives $(\texttt{init}, \mathsf{sid})$ from the environment, he samples $\lambda_\mathsf{c}$ pairs of random $\lambda_\mathsf{c}$-bit seeds and computes their images under the random oracle

   $$\boldsymbol{\kappa}^0 \leftarrow \{0,1\}^{\lambda_\mathsf{c} \times \lambda_\mathsf{c}} \qquad \boldsymbol{\kappa}^1 \leftarrow \{0,1\}^{\lambda_\mathsf{c} \times \lambda_\mathsf{c}}$$
   $$\boldsymbol{\sigma}^0 := \{\mathsf{RO}_{\{0,1\}^{\lambda_\mathsf{c}}}(\mathsf{sid}, \boldsymbol{\kappa}_i^0)\}_{i \in [\lambda_\mathsf{c}]} \qquad \boldsymbol{\sigma}^1 := \{\mathsf{RO}_{\{0,1\}^{\lambda_\mathsf{c}}}(\mathsf{sid}, \boldsymbol{\kappa}_i^1)\}_{i \in [\lambda_\mathsf{c}]}$$

   and broadcasts $(\texttt{seed-images}, \mathsf{sid}, \boldsymbol{\sigma}^0, \boldsymbol{\sigma}^1)$ to all other parties.[a]

2. When $\mathcal{P}_\mathsf{A}$ receives $(\texttt{init}, \mathsf{sid})$ from the environment and $(\texttt{seed-images}, \mathsf{sid}, \boldsymbol{\sigma}^0, \boldsymbol{\sigma}^1)$ from $\mathcal{P}_\mathsf{B}$, she samples $\Delta \leftarrow \mathbb{F}_{2^{\lambda_\mathsf{c}}}$ uniformly and calculates $\boldsymbol{\Delta} := \mathsf{Bits}(\Delta)$. For every $i \in [\lambda_\mathsf{c}]$, $\mathcal{P}_\mathsf{A}$ sends $(\texttt{choose}, \mathcal{P}_\mathsf{A} \| \mathcal{P}_\mathsf{B} \| \mathsf{sid} \| i, \boldsymbol{\Delta}_i)$ to $[\![\mathcal{F}_{\mathsf{SCOT\text{-}SIRIA}}(\{0,1\}^{\lambda_\mathsf{c}})]\!]_{\mathsf{PV}}$.

3. When $\mathcal{P}_\mathsf{B}$ receives $(\texttt{choice-made}, \mathcal{P}_\mathsf{A}\|\mathcal{P}_\mathsf{B}\|\mathsf{sid}\|i)$ from $[\![\mathcal{F}_{\text{SCOT-SIRIA}}(\{0,1\}^{\lambda_\mathsf{c}})]\!]_{\mathsf{PV}}$ for some $i \in [\lambda_\mathsf{c}]$, he sends $(\texttt{messages}, \mathcal{P}_\mathsf{A}\|\mathcal{P}_\mathsf{B}\|\mathsf{sid}\|i, \boldsymbol{\kappa}_i^0, \boldsymbol{\kappa}_i^1)$ to $[\![\mathcal{F}_{\text{SCOT-SIRIA}}(\{0,1\}^{\lambda_\mathsf{c}})]\!]_{\mathsf{PV}}$.

4. On receiving $(\texttt{chosen-message}, \mathcal{P}_\mathsf{A}\|\mathcal{P}_\mathsf{B}\|\mathsf{sid}\|i, \boldsymbol{\kappa}_i^2)$ from $[\![\mathcal{F}_{\text{SCOT-SIRIA}}(\{0,1\}^{\lambda_\mathsf{c}})]\!]_{\mathsf{PV}}$ for every $i \in [\lambda_\mathsf{c}]$, $\mathcal{P}_\mathsf{A}$ verifies that

$$\boldsymbol{\sigma}_i^{\boldsymbol{\Delta}_i} = \mathsf{RO}_{\{0,1\}^{\lambda_\mathsf{c}}}(\mathsf{sid}, \boldsymbol{\kappa}_i^2)$$

for every $i \in [\lambda_\mathsf{c}]$. If all of these equations hold, then she broadcasts $(\texttt{ok}, \mathsf{sid})$ to all parties and outputs $(\texttt{initialized}, \mathsf{sid})$ to the environment. Otherwise, she broadcasts $(\texttt{jaccuse}, \mathsf{sid})$ to all parties and outputs $(\texttt{abort}, \mathsf{sid}, \mathcal{P}_\mathsf{B})$ to the environment.

5. On receiving $(\texttt{ok}, \mathsf{sid})$ from $\mathcal{P}_\mathsf{A}$, party $\mathcal{P}_\mathsf{B}$ outputs $(\texttt{initialized}, \mathsf{sid})$ to the environment.

6. On receiving $(\texttt{jaccuse}, \mathsf{sid})$ from $\mathcal{P}_\mathsf{A}$, $\mathcal{P}_\mathsf{B}$ sends $(\texttt{open}, \mathcal{P}_\mathsf{A}\|\mathcal{P}_\mathsf{B}\|\mathsf{sid}\|i)$ to $[\![\mathcal{F}_{\text{SCOT-SIRIA}}(\{0,1\}^{\lambda_\mathsf{c}})]\!]_{\mathsf{PV}}$ for every $i \in [\lambda_\mathsf{c}]$. He outputs $(\texttt{abort}, \mathsf{sid}, \mathcal{P}_\mathsf{A})$ to the environment and performs no future instructions related to this $\mathsf{sid}$.

**Extend:**

7. When $\mathcal{P}_\mathsf{B}$ receives $(\texttt{choose}, \mathsf{sid}, \mathsf{eid}, \{\mathbb{X}_1, \ldots, \mathbb{X}_{\ell_{\mathsf{OTE}}}\}, \boldsymbol{\beta})$ from the environment, where $\mathsf{eid}$ is fresh, he computes

$$\begin{aligned}
\ell_{\mathsf{OTE}}{}' &\coloneqq \lceil \ell_{\mathsf{OTE}}/\lambda_\mathsf{s} \rceil \cdot \lambda_\mathsf{s} + \lambda_\mathsf{s} \\
\dot{\boldsymbol{\beta}} &\leftarrow \{0,1\}^{\ell_{\mathsf{OTE}}{}'} \\
\mathbf{t}^0 &\coloneqq \{\mathsf{RO}_{\{0,1\}^{\ell_{\mathsf{OTE}}{}'}}(\mathsf{sid}, \mathsf{eid}\|i\|1\|\boldsymbol{\kappa}_{i,*}^0)\}_{i\in[\lambda_\mathsf{c}]} \\
\mathbf{t}^1 &\coloneqq \{\mathsf{RO}_{\{0,1\}^{\ell_{\mathsf{OTE}}{}'}}(\mathsf{sid}, \mathsf{eid}\|i\|1\|\boldsymbol{\kappa}_{i,*}^1)\}_{i\in[\lambda_\mathsf{c}]} \\
\mathbf{u} &\coloneqq \left\{ \left\{ \mathbf{t}_{i,j}^0 \oplus \mathbf{t}_{i,j}^1 \oplus \dot{\boldsymbol{\beta}}_j \right\}_{j\in[\ell_{\mathsf{OTE}}{}']} \right\}_{i\in[\lambda_\mathsf{c}]}
\end{aligned}$$

and sends $(\texttt{commit}, \mathcal{P}_\mathsf{B}\|\mathsf{sid}\|\mathsf{eid}, \dot{\boldsymbol{\beta}})$ to $[\![\mathcal{F}_{\mathsf{Com}}]\!]_{\mathsf{PV}}$, and broadcasts $(\texttt{coded-choices}, \mathsf{sid}, \mathsf{eid}, \{\mathbb{X}_1, \ldots, \mathbb{X}_{\ell_{\mathsf{OTE}}}\}, \mathbf{u})$ to all parties.[b]

8. When $\mathcal{P}_\mathsf{A}$ receives $(\texttt{messages}, \mathsf{sid}, \mathsf{eid}, \boldsymbol{\alpha}^0, \boldsymbol{\alpha}^1)$ from the environment and $(\texttt{coded-choices}, \mathsf{sid}, \mathsf{eid}, \{\mathbb{X}_1, \ldots, \mathbb{X}_{\ell_{\mathsf{OTE}}}\}, \mathbf{u})$ from $\mathcal{P}_\mathsf{B}$ and $(\texttt{committed}, \mathcal{P}_\mathsf{B}\|\mathsf{sid}\|\mathsf{eid})$ from $[\![\mathcal{F}_{\mathsf{Com}}]\!]_{\mathsf{PV}}$, she samples $\boldsymbol{\chi} \leftarrow \mathbb{F}_{2^{\lambda_\mathsf{s}}}^{\lceil \ell_{\mathsf{OTE}}/\lambda_\mathsf{s} \rceil}$ uniformly, broadcasts $(\texttt{challenge}, \mathsf{sid}, \mathsf{eid}, \boldsymbol{\chi})$ to all parties, and outputs $(\texttt{choice-made}, \mathsf{sid}, \mathsf{eid}, \{\mathbb{X}_1, \ldots, \mathbb{X}_{\ell_{\mathsf{OTE}}}\})$ to the environment.[c]

9. Upon receiving $(\texttt{challenge}, \mathsf{sid}, \mathsf{eid}, \boldsymbol{\chi})$ from $\mathcal{P}_\mathsf{A}$, $\mathcal{P}_\mathsf{B}$ computes

$$\hat{\beta} := \sum_{k\in[\lambda_\mathsf{s}]} 2^{k-1} \left( \dot{\boldsymbol{\beta}}_{\lceil \ell_{\mathsf{OTE}}/\lambda_\mathsf{s}\rceil\cdot\lambda_\mathsf{s}+k} + \sum_{j\in[\lceil \ell_{\mathsf{OTE}}/\lambda_\mathsf{s}\rceil]} \boldsymbol{\chi}_j \cdot \dot{\boldsymbol{\beta}}_{j\cdot\lambda_\mathsf{s}+k} \right)$$

$$\hat{\mathbf{t}} := \left\{ \sum_{k\in[\lambda_\mathsf{s}]} 2^{k-1} \left( \mathbf{t}^0_{i,\lceil \ell_{\mathsf{OTE}}/\lambda_\mathsf{s}\rceil\cdot\lambda_\mathsf{s}+k} + \sum_{j\in[\lceil \ell_{\mathsf{OTE}}/\lambda_\mathsf{s}\rceil]} \boldsymbol{\chi}_j \cdot \mathbf{t}^0_{i,j\cdot\lambda_\mathsf{s}+k} \right) \right\}_{i\in[\lambda_\mathsf{c}]}$$

with all operations being performed over $\mathbb{F}_{2^{\lambda_\mathsf{s}}}$, and broadcasts $(\texttt{response}, \mathsf{sid}, \mathsf{eid}, \hat{\beta}, \hat{\mathbf{t}})$ to all parties.

10. When $\mathcal{P}_\mathsf{A}$ receives $(\texttt{response}, \mathsf{sid}, \mathsf{eid}, \hat{\beta}, \hat{\mathbf{t}})$ from $\mathcal{P}_\mathsf{B}$, she computes

$$\mathbf{t}^2 := \{\mathsf{RO}_{\{0,1\}^{\ell_{\mathsf{OTE}'}}}(\mathsf{sid}, \mathsf{eid} \| i \| 1 \| \boldsymbol{\kappa}^2_{i,*})\}_{i\in[\lambda_\mathsf{c}]}$$

$$\mathbf{q} := \left\{ \left\{ \boldsymbol{\Delta}_i \cdot \mathbf{u}_{i,j} \oplus \mathbf{t}^2_{i,j} \right\}_{j\in[\ell_{\mathsf{OTE}'}]} \right\}_{i\in[\lambda_\mathsf{c}]}$$

$$\hat{\mathbf{q}} := \left\{ \sum_{k\in[\lambda_\mathsf{s}]} 2^{k-1} \left( \mathbf{q}_{i,\lceil \ell_{\mathsf{OTE}}/\lambda_\mathsf{s}\rceil\cdot\lambda_\mathsf{s}+k} + \sum_{j\in[\lceil \ell_{\mathsf{OTE}}/\lambda_\mathsf{s}\rceil]} \boldsymbol{\chi}_j \cdot \mathbf{q}_{i,j\cdot\lambda_\mathsf{s}+k} \right) \right\}_{i\in[\lambda_\mathsf{c}]}$$

with operations in the last of these computations being performed over $\mathbb{F}_{2^{\lambda_\mathsf{s}}}$. Using these values, $\mathcal{P}_\mathsf{A}$ verifies that

$$\hat{\mathbf{q}} = \left\{ \hat{\mathbf{t}}_i + \boldsymbol{\Delta}_i \cdot \hat{\beta} \right\}_{i\in[\lambda_\mathsf{c}]}$$

with this check also being performed over $\mathbb{F}_{2^{\lambda_\mathsf{s}}}$. If the equation holds, then $\mathcal{P}_\mathsf{A}$ broadcasts $(\texttt{ok}, \mathsf{sid}, \mathsf{eid})$, and if not, then she broadcasts $(\texttt{jaccuse}, \mathsf{sid}, \mathsf{eid})$, outputs $(\texttt{abort}, \mathsf{sid}, \mathcal{P}_\mathsf{B})$ to the environment and performs no future instructions related to this $\mathsf{sid}$ (halting other **Extension** and **Opening** phases in the middle, if necessary).

11. When $\mathcal{P}_\mathsf{B}$ receives $(\texttt{ok}, \mathsf{sid}, \mathsf{eid})$ from $\mathcal{P}_\mathsf{A}$, he computes $\tilde{\boldsymbol{\beta}} := \{\boldsymbol{\beta}_j \oplus \dot{\boldsymbol{\beta}}_j\}_{j\in[\ell_{\mathsf{OTE}}]}$ and broadcasts $(\texttt{adjust}, \mathsf{sid}, \mathsf{eid}, \tilde{\boldsymbol{\beta}})$.[d]

12. When $\mathcal{P}_\mathsf{A}$ receives $(\texttt{adjust}, \mathsf{sid}, \mathsf{eid}, \tilde{\boldsymbol{\beta}})$ from $\mathcal{P}_\mathsf{B}$, she computes

$$\tilde{\boldsymbol{\alpha}}^0 := \left\{ \mathsf{RO}_{\mathbb{X}_j}(\mathsf{sid}, \mathsf{eid} \| j \| 2 \| (\mathbf{q}_{*,j} + \tilde{\boldsymbol{\beta}}_j \cdot \Delta)) \oplus \boldsymbol{\alpha}^0_j \right\}_{j\in[\ell_{\mathsf{OTE}}]}$$

$$\tilde{\boldsymbol{\alpha}}^1 := \left\{ \mathsf{RO}_{\mathbb{X}_j}(\mathsf{sid}, \mathsf{eid} \| j \| 2 \| (\mathbf{q}_{*,j} + (1 - \tilde{\boldsymbol{\beta}}_j) \cdot \Delta)) \oplus \boldsymbol{\alpha}^1_j \right\}_{j\in[\ell_{\mathsf{OTE}}]}$$

and broadcasts $(\texttt{ciphertexts}, \mathsf{sid}, \mathsf{eid}, \tilde{\boldsymbol{\alpha}}^0, \tilde{\boldsymbol{\alpha}}^1)$ to all parties and outputs $(\texttt{ote-done}, \mathsf{sid}, \mathsf{eid}, \{\mathbb{X}_1, \ldots, \mathbb{X}_{\ell_{\mathsf{OTE}}}\})$ to the environment.

13. On receiving $(\texttt{ciphertexts}, \mathsf{sid}, \mathsf{eid}, \tilde{\boldsymbol{\alpha}}^0, \tilde{\boldsymbol{\alpha}}^1)$ from $\mathcal{P}_\mathsf{A}$, $\mathcal{P}_\mathsf{B}$ computes

$$\boldsymbol{\gamma} := \left\{ \mathsf{RO}_{\mathbb{X}_j}(\mathsf{sid}, \mathsf{eid} \| j \| 2 \| \mathbf{t}^0_{*,j}) \oplus \tilde{\boldsymbol{\alpha}}_j^{\dot{\boldsymbol{\beta}}_j} \right\}_{j \in [\ell_{\mathsf{OTE}}]}$$

and outputs to the environment both $(\texttt{chosen-messages}, \mathsf{sid}, \mathsf{eid}, \boldsymbol{\gamma})$ and $(\texttt{ote-done}, \mathsf{sid}, \mathsf{eid}, \{\mathbb{X}_1, \ldots, \mathbb{X}_{\ell_{\mathsf{OTE}}}\})$.

14. If $\mathcal{P}_\mathsf{B}$ receives $(\texttt{jaccuse}, \mathsf{sid}, \mathsf{eid})$ from $\mathcal{P}_\mathsf{A}$, then he sends $(\texttt{decommit}, \mathcal{P}_\mathsf{B} \| \mathsf{sid} \| \mathsf{eid})$ to $[\![\mathcal{F}_{\mathsf{Com}}]\!]_{\mathsf{PV}}$ and $(\texttt{open}, \mathcal{P}_\mathsf{A} \| \mathcal{P}_\mathsf{B} \| \mathsf{sid} \| i)$ to $[\![\mathcal{F}_{\text{SCOT-SIRIA}}(\{0,1\}^{\lambda_\mathsf{c}})]\!]_{\mathsf{PV}}$ for every $i \in [\lambda_\mathsf{c}]$. He outputs $(\texttt{abort}, \mathsf{sid}, \mathcal{P}_\mathsf{A})$ to the environment and performs no future instructions related to this $\mathsf{sid}$ (halting other **Extension** and **Opening** phases in the middle, if necessary).

**Opening:**

15. Upon receiving $(\texttt{open}, \mathsf{sid})$ from the environment, $\mathcal{P}_\mathsf{A}$ broadcasts $(\texttt{open-keys}, \mathsf{sid}, \boldsymbol{\Delta}, \boldsymbol{\kappa}^2)$.

16. The other parties check that $\boldsymbol{\sigma}_i^{\boldsymbol{\Delta}_i} = \mathsf{RO}_{\{0,1\}^{\lambda_\mathsf{c}}}(\mathsf{sid}, \boldsymbol{\kappa}_i^2)$ for every $i \in [\lambda_\mathsf{c}]$. If these equations do not hold, then they output $(\texttt{abort}, \mathsf{sid}, \mathcal{P}_\mathsf{A})$ to the environment and perform no future instructions related to this $\mathsf{sid}$ (halting other **Extension** and **Opening** phases in the middle, if necessary).

17. For every $\mathsf{eid}$ value that has been associated with this session ID, the parties use the revealed values of $\boldsymbol{\Delta}$ and $\boldsymbol{\kappa}^2$ to compute the value $\mathbf{q}$ corresponding to $\mathsf{eid}$ using the same equations as $\mathcal{P}_\mathsf{A}$ used in step 10. Then, using the values of $\tilde{\boldsymbol{\alpha}}^0$, $\tilde{\boldsymbol{\alpha}}^1$, and $\tilde{\boldsymbol{\beta}}$ that correspond to $\mathsf{eid}$, they compute

$$\boldsymbol{\alpha}^0 := \left\{ \mathsf{RO}_{\mathbb{X}_j}(\mathsf{sid}, \mathsf{eid} \| j \| 2 \| (\mathbf{q}_{*,j} + \tilde{\boldsymbol{\beta}}_j \cdot \Delta)) \oplus \tilde{\boldsymbol{\alpha}}_j^0 \right\}_{j \in [\ell_{\mathsf{OTE}}]}$$
$$\boldsymbol{\alpha}^1 := \left\{ \mathsf{RO}_{\mathbb{X}_j}(\mathsf{sid}, \mathsf{eid} \| j \| 2 \| (\mathbf{q}_{*,j} + (1 - \tilde{\boldsymbol{\beta}}_j) \cdot \Delta)) \oplus \tilde{\boldsymbol{\alpha}}_j^1 \right\}_{j \in [\ell_{\mathsf{OTE}}]}$$

and output $(\texttt{messages}, \mathsf{sid}, \mathsf{eid}, \boldsymbol{\alpha}^0, \boldsymbol{\alpha}^1)$ to the environment.[e] After this, they perform no future instructions related to this $\mathsf{sid}$.

**Verification:**

18. If there is an observing verifier $\mathcal{V}$, then upon receiving $(\texttt{observe}, \mathsf{sid})$ from the environment, $\mathcal{V}$ sends $(\texttt{observe}, \mathcal{P}_\mathsf{A} \| \mathcal{P}_\mathsf{B} \| \mathsf{sid} \| i)$ to $[\![\mathcal{F}_{\text{SCOT-SIRIA}}(\{0,1\}^{\lambda_\mathsf{c}})]\!]_{\mathsf{PV}}$ for every $i \in [\lambda_\mathsf{c}]$.

19. If $\mathcal{V}$ observes $\mathcal{P}_\mathsf{A}$ to broadcast `jaccuse` in step 4, then $\mathcal{V}$ waits to receive $(\texttt{messages}, \mathcal{P}_\mathsf{A}\|\mathcal{P}_\mathsf{B}\|\mathsf{sid}\|i, \boldsymbol{\kappa}_i^0, \boldsymbol{\kappa}_i^1)$ from $[\![\mathcal{F}_{\text{SCOT-SIRIA}}(\{0,1\}^{\lambda_\mathsf{c}})]\!]_{\text{PV}}$ for every $i \in [\lambda_\mathsf{c}]$. If it holds that

$$\boldsymbol{\sigma}^0 = \{\mathsf{RO}_{\{0,1\}^{\lambda_\mathsf{c}}}(\mathsf{sid}, \boldsymbol{\kappa}_i^0)\}_{i\in[\lambda_\mathsf{c}]} \quad \text{and} \quad \boldsymbol{\sigma}^1 = \{\mathsf{RO}_{\{0,1\}^{\lambda_\mathsf{c}}}(\mathsf{sid}, \boldsymbol{\kappa}_i^1)\}_{i\in[\lambda_\mathsf{c}]}$$

then $\mathcal{V}$ outputs $(\texttt{abort}, \mathsf{sid}, \mathcal{P}_\mathsf{A})$ to the environment. If either of these equations does not hold, then $\mathcal{V}$ outputs $(\texttt{abort}, \mathsf{sid}, \mathcal{P}_\mathsf{B})$. Regardless, $\mathcal{V}$ performs no further instructions associated with this $\mathsf{sid}$.

20. If $\mathcal{V}$ observes $\mathcal{P}_\mathsf{B}$ to broadcast $(\texttt{coded-choices}, \mathsf{sid}, \mathsf{eid}, \{\mathbb{X}_1, \ldots, \mathbb{X}_{\ell_{\mathsf{OTE}}}\}, \mathbf{u})$, then $\mathcal{V}$ sends $(\texttt{observe}, \mathcal{P}_\mathsf{B}\|\mathsf{sid}\|\mathsf{eid})$ to $[\![\mathcal{F}_{\mathsf{Com}}]\!]_{\text{PV}}$.

21. If $\mathcal{V}$ observes $\mathcal{P}_\mathsf{A}$ to broadcast `jaccuse` in step 10, then $\mathcal{V}$ waits to receive $(\texttt{decommitted}, \mathcal{P}_\mathsf{B}\|\mathsf{sid}\|\mathsf{eid}, \dot{\boldsymbol{\beta}})$ from $[\![\mathcal{F}_{\mathsf{Com}}]\!]_{\text{PV}}$ and $(\texttt{messages}, \mathcal{P}_\mathsf{A}\|\mathcal{P}_\mathsf{B}\|\mathsf{sid}\|i, \boldsymbol{\kappa}_i^0, \boldsymbol{\kappa}_i^1)$ from $[\![\mathcal{F}_{\text{SCOT-SIRIA}}(\{0,1\}^{\lambda_\mathsf{c}})]\!]_{\text{PV}}$ for every $i \in [\lambda_\mathsf{c}]$. If

$$\boldsymbol{\sigma}^0 \neq \{\mathsf{RO}_{\{0,1\}^{\lambda_\mathsf{c}}}(\mathsf{sid}, \boldsymbol{\kappa}_i^0)\}_{i\in[\lambda_\mathsf{c}]} \quad \text{and} \quad \boldsymbol{\sigma}^1 \neq \{\mathsf{RO}_{\{0,1\}^{\lambda_\mathsf{c}}}(\mathsf{sid}, \boldsymbol{\kappa}_i^1)\}_{i\in[\lambda_\mathsf{c}]}$$

then $\mathcal{V}$ outputs $(\texttt{abort}, \mathsf{sid}, \mathcal{P}_\mathsf{B})$ to the environment. Next, $\mathcal{V}$ computes $\mathbf{u}$, $\hat{\beta}$, and $\hat{\mathbf{t}}$ using the (honest) code of $\mathcal{P}_\mathsf{B}$ from steps 7 and 9. If these values do not match the ones actually transmitted by $\mathcal{P}_\mathsf{B}$, then $\mathcal{V}$ outputs $(\texttt{abort}, \mathsf{sid}, \mathcal{P}_\mathsf{B})$ to the environment. If the recomputed messages of $\mathcal{P}_\mathsf{B}$ match their expected values and

$$\boldsymbol{\sigma}^0 = \{\mathsf{RO}_{\{0,1\}^{\lambda_\mathsf{c}}}(\mathsf{sid}, \boldsymbol{\kappa}_i^0)\}_{i\in[\lambda_\mathsf{c}]} \quad \text{and} \quad \boldsymbol{\sigma}^1 = \{\mathsf{RO}_{\{0,1\}^{\lambda_\mathsf{c}}}(\mathsf{sid}, \boldsymbol{\kappa}_i^1)\}_{i\in[\lambda_\mathsf{c}]}$$

then $\mathcal{V}$ outputs $(\texttt{abort}, \mathsf{sid}, \mathcal{P}_\mathsf{A})$ to the environment. Regardless, $\mathcal{V}$ performs no further instructions associated with this $\mathsf{sid}$.

22. If $\mathcal{V}$ observes $\mathcal{P}_\mathsf{A}$ to broadcast `open-keys` in step 15, then it follows along with the instructions in steps 16 and 17. Afterward, $\mathcal{V}$ performs no further instructions associated with this $\mathsf{sid}$.

---

[a] The values $\boldsymbol{\sigma}^0$ and $\boldsymbol{\sigma}^1$ are new to our protocol and do not exist in the original. The checks involving these values are similarly new.

[b] This commitment was not made in the original SoftspokenOT protocol. Furthermore, the expansions of $\boldsymbol{\kappa}^0$ and $\boldsymbol{\kappa}^1$ to produce $\mathbf{t}^0$ and $\mathbf{t}^1$ were performed via a pseudorandom generator in the original protocol, whereas we use the random oracle. The original protocol used the input choice bits $\boldsymbol{\beta}$ *extended* with random bits, rather than sampling a complete set of random bits $\dot{\boldsymbol{\beta}}$ and adjusting them later, as we do.

[c] The Fiat-Shamir transform [FS86] can be applied here to reduce the number of rounds: the parties compute

$$\boldsymbol{\chi} := \mathsf{RO}_{\mathbb{F}_{2^{\lambda_\mathsf{s}}}^{\lceil \ell_{\mathsf{OTE}}/\lambda_\mathsf{s} \rceil}}(\mathsf{sid}, \mathbf{u}\|\mathsf{eid})$$

rather than receiving $\boldsymbol{\chi}$ from $\mathcal{P}_\mathsf{A}$. This reduces the number of rounds in the **Extend** phase to two. Though we already require a random oracle for other reasons, we omit this optimization from our formal description for simplicity of analysis.

[d] In the original SoftspokenOT protocol, this step does not exist: $\mathcal{P}_\mathsf{B}$ uses his input bits directly rather than sampling uniform ones and adjusting them later, meaning that no adjustment is necessary, and $\mathcal{P}_\mathsf{A}$ simply aborts instead of producing output if the check fails.

[e] Here we assume *all* extensions are opened simultaneously, but of course a version of this protocol can be imagined that opens only some subset.

**Theorem 4.3.** $\pi_{\mathsf{SCOTE\text{-}SIRIA\text{-}PV}}(\lambda_\mathsf{c}, \lambda_\mathsf{s})$ *UC-realizes* $[\![\mathcal{F}_{\mathsf{SCOTE\text{-}SIRIA}}]\!]_{\mathsf{PV}}$ *in the* $([\![\mathcal{F}_{\mathsf{SCOT\text{-}SIRIA}}]\!]_{\mathsf{PV}}, [\![\mathcal{F}_{\mathsf{Com}}]\!]_{\mathsf{PV}})$*-hybrid local random oracle model, in the presence of a malicious adversary that statically corrupts any number of passive verifiers and at most one of the active participants* $\mathcal{P}_\mathsf{A}$ *and* $\mathcal{P}_\mathsf{B}$*.*

*Proof Sketch.* The proof of the original SoftspokenOT protocol is long and elaborate, and we do not feel that reproducing it here would add significant value to this work. Therefore, we assume that the original protocol is secure, and argue informally that our augmentations allow the protocol to realize our enhanced functionality without compromising the existing proof.

- *Steps 1 through 6.* In the **Initialization** phase, the main distinction between our protocol and the original SoftspokenOT protocol is that $\mathcal{P}_\mathsf{B}$ transmits $\{\boldsymbol{\sigma}_i^{1-\boldsymbol{\Delta}_i}\}_{i\in[\lambda_\mathsf{c}]}$ to $\mathcal{P}_\mathsf{A}$. Since $\mathcal{P}_\mathsf{A}$ receives $\{\boldsymbol{\kappa}_i^{\boldsymbol{\Delta}_i}\}_{i\in[\lambda_\mathsf{c}]}$ via the base OT instances, it is already possible for $\mathcal{P}_\mathsf{A}$ to compute $\{\boldsymbol{\sigma}_i^{\boldsymbol{\Delta}_i}\}_{i\in[\lambda_\mathsf{c}]}$ locally.

  If $\mathcal{P}_\mathsf{A}$ is corrupt, the simulator can behave as an honest $\mathcal{P}_\mathsf{B}$ would, sampling uniform values for $\{\boldsymbol{\kappa}_i^{1-\boldsymbol{\Delta}_i}\}_{i\in[\lambda_\mathsf{c}]}$, transmitting their images under the random oracle, and releasing them if $\mathcal{P}_\mathsf{A}$ demands it. The only means by which $\mathcal{P}_\mathsf{A}$ can cheat is the inappropriate transmission of a `jaccuse` message, and in this case it is easy to see that all verifiers will identify her as a cheater if $\mathcal{P}_\mathsf{B}$ has behaved honestly. Under this simulation $\mathcal{P}_\mathsf{A}$'s view is identical to her view in the real world.

  If $\mathcal{P}_\mathsf{B}$ is corrupt, then he might send a value of $\boldsymbol{\sigma}_i^b$ for some $i \in [\lambda_\mathsf{c}]$ and $b \in \{0,1\}$ that is not calculated as the image of $\boldsymbol{\kappa}_i^b$ under the random oracle. If $\boldsymbol{\Delta}_i = b$, this cheat is detected (and an abort occurs) in the real world with overwhelming probability, but if $\boldsymbol{\Delta}_i \neq b$, then an honest $\mathcal{P}_\mathsf{A}$ will not complain, and he learns the value of $\boldsymbol{\Delta}_i$. By observing the random oracle queries and extracting the values of $\boldsymbol{\kappa}_i^b$ for $b \in \{0,1\}$ and $i \in [\lambda_\mathsf{c}]$, the simulator can detect in how many locations $\mathcal{P}_\mathsf{B}$ has cheated, and flip a matching number of uniform coins to determine whether to send `jaccuse` on behalf of $\mathcal{P}_\mathsf{A}$ (after which he will be identifies by the verifiers as a cheater with certainty). The distribution of aborts implied by this simulation strategy is identical to the distribution of aborts in the real world; the outstanding discrepancy between the two is thus the leakage implied by a successful selective failure attack on the part of $\mathcal{P}_\mathsf{B}$. This leakage is of exactly the same flavor as the leakage implied by the SoftspokenOT consistency check, and Roy proved that the protocol remains simulation secure under such leakage [Roy22, Theorem 5.1].

- *Steps 7 and 14.* In this step, we add a commitment to $\mathcal{P}_\mathsf{B}$'s random choice bits $\dot{\boldsymbol{\beta}}$, which $\mathcal{P}_\mathsf{A}$ can compel $\mathcal{P}_\mathsf{B}$ to open. We must make two arguments:

first, that if a corrupt $\mathcal{P}_\mathsf{A}$ compels an honest $\mathcal{P}_\mathsf{B}$ to open the commitment, this action is simulatable (and observing verifiers identify $\mathcal{P}_\mathsf{A}$ as a cheater), and second, that an honest $\mathcal{P}_\mathsf{A}$ detects a failure in the consistency check in step 10 and compels $\mathcal{P}_\mathsf{B}$ to open the commitment, then any observers identify $\mathcal{P}_\mathsf{B}$ as a cheater with overwhelming probability.

In the case that a corrupt $\mathcal{P}_\mathsf{A}$ compels an honest $\mathcal{P}_\mathsf{B}$ to open his commitment, we must give a simulation strategy. In particular, the simulator must emit a uniform set of choice bits $\dot{\boldsymbol{\beta}}$ on behalf of $[\![\mathcal{F}_{\mathsf{Com}}]\!]_{\mathsf{PV}}$ and two sets of keys $\boldsymbol{\kappa}^0$ and $\boldsymbol{\kappa}^1$ on behalf of $[\![\mathcal{F}_{\mathsf{SCOT\text{-}SIRIA}}(\{0,1\}^{\lambda_\mathsf{c}})]\!]_{\mathsf{PV}}$ that are consistent with the values of $\mathbf{u}$, $\hat{\beta}$, and $\hat{\mathbf{t}}$ that it has already transmitted on behalf $\mathcal{P}_\mathsf{B}$. Because the simulator can program the random oracle, this is easy: it samples $\dot{\boldsymbol{\beta}}$, and $\mathbf{t}^0$ uniformly subject to

$$\hat{\beta} = \sum_{k\in[\lambda_\mathsf{s}]} 2^{k-1} \left( \dot{\boldsymbol{\beta}}_{\lceil \ell_{\mathsf{OTE}}/\lambda_\mathsf{s}\rceil\cdot\lambda_\mathsf{s}+k} + \sum_{j\in[\lceil \ell_{\mathsf{OTE}}/\lambda_\mathsf{s}\rceil]} \boldsymbol{\chi}_j \cdot \dot{\boldsymbol{\beta}}_{j\cdot\lambda_\mathsf{s}+k} \right)$$

$$\hat{\mathbf{t}} = \left\{ \sum_{k\in[\lambda_\mathsf{s}]} 2^{k-1} \left( \mathbf{t}^0_{i,\lceil \ell_{\mathsf{OTE}}/\lambda_\mathsf{s}\rceil\cdot\lambda_\mathsf{s}+k} + \sum_{j\in[\lceil \ell_{\mathsf{OTE}}/\lambda_\mathsf{s}\rceil]} \boldsymbol{\chi}_j \cdot \mathbf{t}^0_{i,j\cdot\lambda_\mathsf{s}+k} \right) \right\}_{i\in[\lambda_\mathsf{c}]}$$

and sample $\boldsymbol{\kappa}^0$ and $\boldsymbol{\kappa}^1$ uniformly, and then programs $\mathsf{RO}_{\{0,1\}^{\ell_{\mathsf{OTE}}'}}(\mathsf{sid},\mathsf{eid}\|i\|1\|\boldsymbol{\kappa}^0_{i,*}) \mapsto \mathbf{t^0}_{i,*}$ for every $i \in [\lambda_\mathsf{c}]$. The distributions of these values are all identical to their distributions in the real world: the adversary can distinguish the two only if it queries the random oracle on the value $\mathsf{eid}\|i\|1\|\boldsymbol{\kappa}^0_{i,*}$ for some $i \in [\lambda_\mathsf{c}]$ *before* the oracle is programmed, and thus notices the reprogramming. The adversary can make at most polynomially many calls to the random oracle, and its chance of successfully guessing the (as-yet-undefined) value $\boldsymbol{\kappa}^0_{i,*}$ for any $i \in [\lambda_\mathsf{c}]$ is $\lambda_\mathsf{c} \cdot 2^{-\lambda_\mathsf{c}}$ per query. Thus the adversary's probability of distinguishing based upon the values released in this event is bounded overall by $O(\mathrm{poly}(\lambda_\mathsf{c})/2^{\lambda_\mathsf{c}}) \in \mathrm{negl}(\lambda_\mathsf{c})$.

Toward our argument that a corrupt $\mathcal{P}_\mathsf{B}$ will be identified as a cheater with overwhelming probability in the event that an honest $\mathcal{P}_\mathsf{A}$ compels him to open his commitment to $\dot{\boldsymbol{\beta}}$, we partition the outcomes of the protocol into three mutually exclusive cases:

1. $\mathcal{P}_\mathsf{B}$ acts honestly. In this case, by the correctness of the original SoftspokenOT protocol, $\mathcal{P}_\mathsf{A}$ will not compel him to open his commitment.
2. $\mathcal{P}_\mathsf{B}$ acts honestly, except that in step 7 he commits to a set of choice bits that is inconsistent with the rest of his view. As in the previous case, he will never be compelled to open his commitment, by the correctness of the original SoftspokenOT protocol. It follows that his cheat will go undetected, and his protocol execution is consistent with respect to *some* set of choice bits so far as any other party is concerned. If the simulator detects that this case has occurred, it behaves as though $\mathcal{P}_\mathsf{B}$ has acted honestly.

3. $\mathcal{P}_\mathsf{B}$ deviates from the instructions in steps 7 and 9, in some way other than simply committing to an inconsistent value of $\dot{\boldsymbol{\beta}}$. In this case, notice that $\mathcal{P}_\mathsf{B}$'s instructions in steps 7 and 9 are deterministic once $\dot{\boldsymbol{\beta}}$, $\boldsymbol{\kappa}^0$, and $\boldsymbol{\kappa}^1$ are fixed. Because we have excluded case 2, we know that $\mathcal{P}_\mathsf{B}$'s protocol execution is not consistent with *any* fixed set of choice bits. Therefore, if $\mathcal{P}_\mathsf{A}$ compels $\mathcal{P}_\mathsf{B}$ to open his commitment, then regardless of which set of bits he has committed to, the observing verifiers will notice an inconsistency upon tracing his instructions forward, and he will be identified as a cheater.

- *Step 11.* The *adjustment* of $\mathcal{P}_\mathsf{B}$'s choice bits performed in this step is a standard technique with perfect security. We also use it in section 3.2.

- *Steps 15 through 17.* This phase is completely independent of $\mathcal{P}_\mathsf{B}$'s private inputs. It remains only to show that a corrupt $\mathcal{P}_\mathsf{A}$ cannot open *incorrect* values without detection, and that the view of an honest $\mathcal{P}_\mathsf{A}$ can be simulated efficiently.

  Our simulation argument is simple: the simulator, in its role as $[\![\mathcal{F}_{\mathsf{SCOT\text{-}SIRIA}}(\{0,1\}^{\lambda_\mathsf{c}})]\!]_{\mathsf{PV}}$, has already extracted $\boldsymbol{\kappa}^0$, $\boldsymbol{\kappa}^1$ in their entirety from $\mathcal{P}_\mathsf{B}$, and the original simulation strategy of SoftspokenOT implies that it has also extracted $\dot{\boldsymbol{\beta}}$, and therefore $\boldsymbol{\beta}$. Due to the selective failure attacks in the **Initialization** and **Extension** phases, some simulated choice bits may already have been fixed. The simulator now flips coins to determine the remaining unfixed choice bits. Before releasing $\boldsymbol{\kappa}_i^{\boldsymbol{\Delta}_i}$ for $i \in [\lambda_\mathsf{c}]$, the simulator calculates the implied values of $\mathbf{q}$, and programs the random oracle such that

$$\mathsf{RO}_{\mathbb{X}_j}(\mathsf{sid}, \mathsf{eid} \| j \| 2 \| (\mathbf{q}_{*,j} + (1 - \boldsymbol{\beta}_j) \cdot \Delta)) \mapsto \boldsymbol{\alpha}_j^{1-\dot{\boldsymbol{\beta}}_i} \oplus \tilde{\boldsymbol{\alpha}}_j^{1-\dot{\boldsymbol{\beta}}_i}$$

  for every $j \in [\ell_{\mathsf{OTE}}]$. The adversary can distinguish this simulation from the real world if it queries the random oracle on the value $\mathsf{eid} \| j \| 2 \| (\mathbf{q}_{*,j} + (1 - \boldsymbol{\beta}_j) \cdot \Delta)$ for some $j \in [\ell_{\mathsf{OTE}}]$ *before* the oracle is programmed, and thus notices the reprogramming. The adversary can make at most polynomially many calls to the random oracle, and its chance of successfully guessing the (as-yet-undefined) value $\mathbf{q}_{*,j}$ for any $j \in [\ell_{\mathsf{OTE}}]$ is $\ell_{\mathsf{OTE}} \cdot 2^{-\lambda_\mathsf{c}}$ per query. Thus the adversary's probability of distinguishing based upon the values released in this event is bounded overall by $O(\mathrm{poly}(\lambda_\mathsf{c})/2^{\lambda_\mathsf{c}}) \in \mathrm{negl}(\lambda_\mathsf{c})$.

  Finally, we argue that a corrupt $\mathcal{P}_\mathsf{A}$ must open the correct values, or else be identified as a cheater. We observe first of all that fixing $\boldsymbol{\Delta}$, $\boldsymbol{\kappa}^2$, and $\mathcal{P}_\mathsf{B}$'s message $\mathbf{u}$ also fixes $\mathbf{q}$, and thus implies a unique decryption $\boldsymbol{\alpha}^b$ of the ciphertext $\tilde{\boldsymbol{\alpha}}^b$ for $b \in \{0,1\}$. In order to trick the verifiers into decrypting any other value, $\mathcal{P}_\mathsf{A}$ must convince them to accept incorrect values of $\boldsymbol{\Delta}$ and $\boldsymbol{\kappa}^2$. Let the values actually transmitted by a corrupt $\mathcal{P}_\mathsf{A}$ be denoted $\boldsymbol{\Delta}^*$ and $\boldsymbol{\kappa}^{2*}$; in order to be accepted by a verifier, they must satisfy the relation $\boldsymbol{\sigma}_i^{\boldsymbol{\Delta}_i^*} = \mathsf{RO}_{\{0,1\}^{\lambda_\mathsf{c}}}(\mathsf{sid}, \boldsymbol{\kappa}_i^{2*})$ for every $i \in [\lambda_\mathsf{c}]$. This implies that for every index $i$ at which $\mathcal{P}_\mathsf{A}$ cheats, she must find a first or second $\lambda_\mathsf{c}$-bit preimage of $\boldsymbol{\sigma}_i^{\boldsymbol{\Delta}_i^*}$ under the random oracle. She has no better strategy than brute force search

for doing so, and she can make at most polynomially many attempts, which implies that the probability that she convinces passive verifiers to accept any incorrect opening is in $O(\text{poly}(\lambda_c)/2^{\lambda_c}) \in \text{negl}(\lambda_c)$.

We have now covered each change or addition that we have made in our protocol, relative to the original SoftspokenOT protocol, and in each case we have given a simulation strategy. This concludes our proof sketch. □

## 4.1 SCOT Extension Extension

As we discussed in section 1.1, the intuitive approach of *revealing one's randomness* is difficult to simulate. It has another disadvantage: once the secret randomness of a protocol is revealed, it cannot be reused, nor can any protocol state derived from it. This is especially irksome if corrupt parties can cause honest parties to reveal their randomness, because it means that expensive setup procedures must be performed again in order for the protocol to continue after the cheaters are eliminated. Our strategy mitigates this problem. We observe that if an abort occurs within the context of $\pi_{\mathsf{SCOTE\text{-}SIRIA\text{-}PV}}$ itself, then the underlying instances of $[\![\mathcal{F}_{\mathsf{SCOT\text{-}SIRIA}}]\!]_{\mathsf{PV}}$ are opened; in this case, either $\mathcal{P}_\mathsf{A}$ or $\mathcal{P}_\mathsf{B}$ is certainly corrupt, which means that they will not need to interact again. On the other hand, they may be compelled to use the `open` interface in some larger protocol, even though they are both honest. In this case, the randomness of $\pi_{\mathsf{SCOTE\text{-}SIRIA\text{-}PV}}$ is revealed *without* opening the underlying instances of $[\![\mathcal{F}_{\mathsf{SCOT\text{-}SIRIA}}]\!]_{\mathsf{PV}}$. This distinction becomes meaningful if the underlying instances of $[\![\mathcal{F}_{\mathsf{SCOT\text{-}SIRIA}}]\!]_{\mathsf{PV}}$ are replaced by a single call to the extension interface of $[\![\mathcal{F}_{\mathsf{SCOTE\text{-}SIRIA}}]\!]_{\mathsf{PV}}$, and a single instance of $[\![\mathcal{F}_{\mathsf{SCOTE\text{-}SIRIA}}]\!]_{\mathsf{PV}}$ is shared among multiple instances of $\pi_{\mathsf{SCOTE\text{-}SIRIA\text{-}PV}}$. We refer to this technique as *OT Extension Extension*. In other words, if we use one *root* OT extension protocol to supply the base OTs for many more *leaf* instances of the same OT extension protocol, then `open`ing a single one of the leaf instances does not imply `open`ing the root instance, and does not affect the other leaf instances. On the other hand, if one of the leaf instances experiences an abort, then the root instance is opened, which implies that *all* of the leaf instances are opened. When the protocol is instantiated in this way, two honest parties need *never* re-run their shared OT extension initialization, regardless of the adversary's behavior. We note that, under this instantiation, our protocol realizes a more complex functionality in which blocks of messages can be opened selectively, and in which unopened blocks can be extended but opened blocks cannot, and new blocks can be created so long as no abort has occurred. We elide a formal specification of this functionality from this work.

**Corollary 4.4.** *There is a protocol that UC-realizes* $[\![\mathcal{F}_{\mathsf{SCOTE\text{-}SIRIA}}]\!]_{\mathsf{PV}}$ *where the number of public key operations is independent of the number of* `choose` *and* `open` *queries.*

*Proof Sketch.* The above corollary is achieved simply by instantiating the $\lambda_c$ independent instances $[\![\mathcal{F}_{\mathsf{SCOT\text{-}SIRIA}}]\!]_{\mathsf{PV}}$ used by $\pi_{\mathsf{SCOTE\text{-}SIRIA\text{-}PV}}$ with a single call to

the `choose` interface of an instance of $[\![\mathcal{F}_{\text{SCOTE-SIRIA}}]\!]_{\text{PV}}$ that is shared with other similarly-modified instances of $\pi_{\text{SCOTE-SIRIA-PV}}$. In other words, it is achieved by using extended OTs to instantiate the base OTs for multiple instances of $\pi_{\text{SCOTE-SIRIA-PV}}$. □

# 5 From SCOT-(S)IRIA to CVOLE-IRIA

In this section, we use the SCOT-SIRIA functionality introduced in section 3 to statistically UC-realize Committed Vector Oblivious Linear Evaluation with (double-sided) Input-Revealing Identifiable Abort (CVOLE-IRIA) over any finite field. In section 5.3 we also discuss simple modifications to our functionality and protocol that allow only one of the two inputs to be revealed (either the vector, or the single value by which the vector is multiplied). First we introduce the functionality.

**Functionality 5.1.** $\mathcal{F}_{\text{CVOLE-IRIA}}(\mathbb{F}, \ell)$**: Committed Vector OLE**

This functionality interacts with two active participants, $\mathcal{P}_\mathsf{A}$ and $\mathcal{P}_\mathsf{B}$, who we refer to as Alice and Bob, and with the ideal adversary $\mathcal{S}$. It is parameterized by a field $\mathbb{F}$ and an integer $\ell$.

**VOLE:** On receiving $(\texttt{correlation}, \mathsf{sid}, b)$ from Bob such that $b \in \mathbb{F}$ and $\mathcal{P}_\mathsf{B} \| \mathcal{P}_\mathsf{A} \| \mathsf{sid}' = \mathsf{sid}$ for some fresh $\mathsf{sid}'$, send $(\texttt{correlation-loaded}, \mathsf{sid})$ to all parties. On receiving $(\texttt{vector}, \mathsf{sid}, \mathbf{a})$ from Alice such that $\mathbf{a} \in \mathbb{F}^\ell$, send $(\texttt{vector-loaded}, \mathsf{sid})$ to all parties. When both Alice and Bob's messages have been received:

- If Alice is corrupt, then receive $(\texttt{adv-share}, \mathsf{sid}, \mathbf{c})$ from the adversary and compute $\mathbf{d} := \{b \cdot \mathbf{a}_i - \mathbf{c}_i\}_{i \in [\ell]}$.
- If Bob is corrupt, then receive $(\texttt{adv-share}, \mathsf{sid}, \mathbf{d})$ from the adversary and compute $\mathbf{c} := \{b \cdot \mathbf{a}_i - \mathbf{d}_i\}_{i \in [\ell]}$.
- If neither active participant is corrupt, then sample $\mathbf{c}, \mathbf{d} \leftarrow \mathbb{F}^\ell$ uniformly subject for every $i \in [\ell]$ to $b \cdot \mathbf{a}_i = \mathbf{c}_i + \mathbf{d}_i$.

Finally, store $(\texttt{vole}, \mathsf{sid}, \mathbf{a}, b, \mathbf{c}, \mathbf{d})$ in memory and send $(\texttt{share}, \mathsf{sid}, \mathbf{c})$ to Alice and $(\texttt{share}, \mathsf{sid}, \mathbf{d})$ to Bob and $(\texttt{vole-done}, \mathsf{sid})$ to all parties.

**Opening:** On receiving $(\texttt{open}, \mathsf{sid})$ from $\mathcal{P}_i$ for some $i \in \{\mathsf{A}, \mathsf{B}\}$, send $(\texttt{open-req}, \mathsf{sid}, \mathcal{P}_i)$ to all parties. On receiving such a message from both Alice and Bob, if a record of the form $(\texttt{vole}, \mathsf{sid}, *, *, *, *)$ exists in memory, then send it to all parties and ignore all future instructions with the same $\mathsf{sid}$ value.

**Abort:** On receiving $(\mathtt{abort}, \mathsf{sid})$ from $\mathcal{S}$ at any point, such that $\mathcal{P}_\mathsf{B} \| \mathcal{P}_\mathsf{A} \| \mathsf{sid}' = \mathsf{sid}$, if $\mathcal{P}_c$ is corrupt for some $c \in \{\mathsf{A}, \mathsf{B}\}$, then send $(\mathtt{vole}, \mathsf{sid}, *, *, *, *)$ to $\mathcal{S}$ if such a record exists in memory, and regardless send $(\mathtt{abort}, \mathsf{sid}, \mathcal{P}_c)$ to all parties and ignore all future instructions with the same $\mathsf{sid}$ value.

## 5.1 Special Case for Booleans

Before we move on to our main construction, we observe that if we were to modify $[\![\mathcal{F}_{\mathsf{SCOT\text{-}SIRIA}}(\mathbb{Z}_2^\ell)]\!]_{\mathsf{PV}}$ to reveal Bob's inputs along with Alice's, then it would be identical to $[\![\mathcal{F}_{\mathsf{CVOLE\text{-}IRIA}}(\mathbb{Z}_2, \ell)]\!]_{\mathsf{PV}}$ up to renaming of the interfaces. We first show that the latter functionality can be realized very simply using two instances of the latter.

**Theorem 5.2.** *For any $\ell \in \mathbb{N}^+$, there is a protocol that statistically UC-realizes $[\![\mathcal{F}_{\mathsf{CVOLE\text{-}IRIA}}(\mathbb{Z}_2, \ell)]\!]_{\mathsf{PV}}$ against a malicious adversary statically corrupting at most one active participant and any number of passive verifiers in the $[\![\mathcal{F}_{\mathsf{SCOT\text{-}SIRIA}}(\mathbb{Z}_2^{\ell+\lambda_\mathsf{s}})]\!]_{\mathsf{PV}}$-hybrid model. This protocol requires at most 2 invocations of $[\![\mathcal{F}_{\mathsf{SCOT\text{-}SIRIA}}]\!]_{\mathsf{PV}}$ in total.*

*Proof Sketch.* Let $\ell' := \ell + \lambda_\mathsf{s}$. We can realize $[\![\mathcal{F}_{\mathsf{CVOLE\text{-}IRIA}}(\mathbb{Z}_2, \ell)]\!]_{\mathsf{PV}}$ using two instance of $[\![\mathcal{F}_{\mathsf{SCOT\text{-}SIRIA}}(\mathbb{Z}_2^{\ell'})]\!]_{\mathsf{PV}}$ in the following way: When Bob receives the `correlation` instruction, he makes a corresponding `choose` instruction to the first $[\![\mathcal{F}_{\mathsf{SCOT\text{-}SIRIA}}(\mathbb{Z}_2^{\ell'})]\!]_{\mathsf{PV}}$ instance. When Alice receives the `vector` instruction, she makes a corresponding `mesages` instruction to the first $[\![\mathcal{F}_{\mathsf{SCOT\text{-}SIRIA}}(\mathbb{Z}_2^{\ell'})]\!]_{\mathsf{PV}}$ instance, and concatenates each of her two inputs with a uniform $\lambda_\mathsf{s}$-bit tag (which Bob omits when computing his output).

When Bob receives the `open` instruction, he samples two XOR shares of his output from the first instance $[\![\mathcal{F}_{\mathsf{SCOT\text{-}SIRIA}}(\mathbb{Z}_2^{\ell'})]\!]_{\mathsf{PV}}$ *including* Alice's tag, and then invokes a second instance of $[\![\mathcal{F}_{\mathsf{SCOT\text{-}SIRIA}}(\mathbb{Z}_2^{\ell'})]\!]_{\mathsf{PV}}$ as the *sender*, using these shares as his input. This commits him publicly to the output of his first $[\![\mathcal{F}_{\mathsf{SCOT\text{-}SIRIA}}(\mathbb{Z}_2^{\ell'})]\!]_{\mathsf{PV}}$ instance. When Alice receives the `open` instruction and a notification that Bob is committed, she sends a corresponding `open` instruction to the first instance of $[\![\mathcal{F}_{\mathsf{SCOT\text{-}SIRIA}}(\mathbb{Z}_2^{\ell'})]\!]_{\mathsf{PV}}$, revealing her input. Then Bob opens the second instance of $[\![\mathcal{F}_{\mathsf{SCOT\text{-}SIRIA}}(\mathbb{Z}_2^{\ell'})]\!]_{\mathsf{PV}}$, which effectively decommits his output from the first instance and in turn implies a specific set of choice bits. Bob can cheat only by guessing the OT message that he did not receive, which is as hard a guessing Alice's tag.

Since a corrupt party can cause either the first instance of $[\![\mathcal{F}_{\mathsf{SCOT\text{-}SIRIA}}(\mathbb{Z}_2^{\ell'})]\!]_{\mathsf{PV}}$ to abort, revealing Alice's inputs, or the second instance, revealing Bob's, this protocol realizes a functionality with (double sided) IRIA. □

## 5.2 For Any Finite Field

Now we move on to our main construction of Committed VOLE for any finite field. This construction is based upon the two-round OT-multiplication protocol of Doerner et al. [DKLs18, DKLs19]. Doerner et al. derived their technique in turn from the semi-honest OT-based multiplication protocol of Gilboa [Gil99]. They observed that Gilboa's protocol had perfect security against the party we designate as Bob, in the OT-hybrid model. They combined Gilboa's protocol with a random linear check to ensure that Alice is forced to use consistent inputs, and mitigated selective failure attacks on her part by using a bit-fixing randomness extractor to encode Bob's inputs in such a way that she cannot learn enough bits to distinguish the encoding from uniform with non-negligible probability, without triggering an abort with overwhelming probability.

We make a few simple modifications to adapt the Doerner et al. protocol to our setting. First, we transform the basic two-input multiplier into a Vector OLE; this has already been explored both in the original Doerner et al. [DKLs18] incarnation and in the follow-up works of Chen et al. [CCD+20] and Doerner et al. [DKLs23]. Since we only wish Bob to reuse his input, the technique is simple: Alice uses the same batch of OT instances, corresponding to one encoded input for Bob, to evaluate the protocol for all of the inputs that she wishes to multiply by Bob's one input.

Second, we generate the randomness for the linear check interactively, rather than by using the random oracle. This increases the round count to four, but allows the protocol to achieve statistical security in the OT-hybrid model. We also repeat the random linear check enough times to achieve statistical security even if the working field is small. This was previously explored by Chen et al. [CCD+20].

Finally we add an input-decommitment phase, and specify that it is triggered involuntarily in the case of an abort, or voluntarily if no abort occurs. While Chen et al. [CCD+20] explored the addition of a similar (but purely voluntary) decommitment phase, their method is substantially different from ours, and faces an inherent barrier that prevents it from being used with fields of order $\omega(\lambda_c)$. We evade this barrier.

The most obvious way to allow Bob to reveal his input is simply to have him broadcast the messages he has chosen to receive from the relevant OT instances; these messages bind him to his choice bits, and therefore to an encoding of his input. If $[\![\mathcal{F}_{\mathsf{SCOT\text{-}SIRIA}}]\!]_{\mathsf{PV}}$ is used instead of plain OT, then the OT functionality can subsequently reveal the true OT messages, and any cheating on Bob' part becomes obvious. However, if Alice is corrupt, then this simple method is unsimulatable. If Alice behaves honestly in the multiplication protocol, then the simulator can find an encoding of Bob's input in the same way that an honest Bob would. However, Alice might cheat in the multiplication protocol by sending message pairs to some OT instances that are not correlated in the correct way: in this case, the simulator must sample and fix the OT choice bits of the corresponding instances in order to determine the impact of Alice's cheats. This effectively fixes some bits of Bob's encoding; under the condition

that some bits are fixed ahead of time, finding an encoding of an arbitrary input for Bob (in order to open his input, when it is later learned) involves solving an adversarially-influenced instance of subset sum. We avoid this difficulty by ensuring that Bob only reveals his choice bits (and thus the simulator only has to find a valid encoding) when Alice has been *honest*. Since our security notion only requires a single corrupt party to be identified in the case of an abort, Bob's behavior need not be investigated once it is determined with certainty that Alice is corrupt.

We note that the success of the above strategy clearly illustrates the intuitive notion that identifiable abort is *easier* than security against adaptive corruptions. To our knowledge, the type of multiplication protocol we have presented is *not* adaptively secure because an adversary corrupting Alice could cheat in such a way that when it later corrupts Bob, the simulator is forced to solve an instance of subset sum. The subset sum problem is NP-Complete in general, and we do not know of any strategy under which it is efficiently solveable in the specific parameter regime that an adaptive adversary induces.

Whereas a simulator against an adaptive adversary is required to produce explanations for past messages *on the spot*, protocols that achieve identifiable can execute additional instructions upon detection of inconsistencies in the protocol. In our case, these extra instructions produce *selective* explanations for past messages, which are sufficient to vindicate honest parties, yet always simulatable, because the case of explaining two corrupt parties' messages simultaneously is avoided.

**Protocol 5.3.** $\pi_{\mathsf{CVOLE\text{-}IRIA\text{-}PV}}(\mathbb{F}_q, \ell, \lambda_\mathsf{s})$**. Committed Vector OLE**

This protocol involves two active participants, $\mathcal{P}_\mathsf{A}$ and $\mathcal{P}_\mathsf{B}$, who we refer to as Alice and Bob, an a-priori-unknown number of passive verifiers (collectively denoted $\mathcal{V}$), and the ideal functionality $[\![\mathcal{F}_{\mathsf{SCOT\text{-}SIRIA}}]\!]_{\mathsf{PV}}$. It is parameterized by a finite field $\mathbb{F}_q$ of size $q$; unless otherwise stated all mathematical operations in this protocol are defined over this field. It is also parameterized by an integer $\ell$ and the statistical security parameter $\lambda_\mathsf{s}$. For convenience, we define $\xi := 2|q| + 2\lambda_\mathsf{s}$ to be the bit-length of Bob's inputs after they have been encoded, and $\rho := \lceil \lambda_\mathsf{s}/|q| \rceil$ to be the number of $\mathbb{F}_q$ elements required for a $\lambda_\mathsf{s}$-bit MAC.

**VOLE - Correlation Input:**

1. When Bob receives $(\texttt{correlation}, \mathsf{sid}, b)$ from the environment such that $\mathcal{P}_\mathsf{B} \| \mathcal{P}_\mathsf{A} \| \mathsf{sid}' = \mathsf{sid}$ and Bob has *never* interacted with Alice before, Bob samples $\mathbf{g}' \leftarrow \mathbb{F}_q^{|q|+2\lambda_\mathsf{s}}$ and broadcasts $(\texttt{coefficients}, \mathcal{P}_\mathsf{B} \| \mathcal{P}_\mathsf{A}, \mathbf{g}')$. Both parties then compute $\mathbf{g} := \{2^{i-1}\}_{i \in [|q|]} \| \mathbf{g}'$.

2. When Bob receives $(\texttt{correlation}, \mathsf{sid}, b)$ from the environment such that $\mathcal{P}_\mathsf{B} \| \mathcal{P}_\mathsf{A} \| \mathsf{sid}' = \mathsf{sid}$ where $\mathsf{sid}'$ is a fresh value, he samples an encoding of

his input

$$\dot{\boldsymbol{\beta}} \leftarrow \{0,1\}^{|q|+2\lambda_s}$$
$$\dot{b} := \left\langle \mathbf{g}_{[|q|+1:\xi]}, \dot{\boldsymbol{\beta}} \right\rangle$$
$$\boldsymbol{\beta} := \mathsf{Bits}(b - \dot{b}) \| \dot{\boldsymbol{\beta}}$$

and then Bob sends $(\texttt{choose}, \mathsf{sid} \| i, \boldsymbol{\beta}_i)$ to $[\![\mathcal{F}_{\mathsf{SCOT\text{-}SIRIA}}(\mathbb{F}_q^{(1+\rho)\cdot\ell})]\!]_{\mathsf{PV}}$ for every $i \in [\xi]$ and outputs $(\texttt{correlation-loaded}, \mathsf{sid})$. Note that the $\mathsf{Bits}$ function outputs a little-endian bit-vector representation of its argument.

3. If Alice and Bob receive $(\texttt{abort}, \mathsf{sid} \| j, \mathcal{P}_c)$ from $[\![\mathcal{F}_{\mathsf{SCOT\text{-}SIRIA}}(\mathbb{F}_q^{(1+\rho)\cdot\ell})]\!]_{\mathsf{PV}}$ for any $j \in [\xi]$, then they output $(\texttt{abort}, \mathsf{sid}, \mathcal{P}_c)$ to the environment and ignore all future instructions with the same $\mathsf{sid}$.

4. Upon receiving $(\texttt{choice-made}, \mathsf{sid} \| j)$ from $[\![\mathcal{F}_{\mathsf{SCOT\text{-}SIRIA}}(\mathbb{F}_q^{(1+\rho)\cdot\ell})]\!]_{\mathsf{PV}}$ for every $j \in [\xi]$, Alice outputs $(\texttt{correlation-loaded}, \mathsf{sid})$ to the environment.

**VOLE - Vector Input:**

5. When Alice receives $(\texttt{vector}, \mathsf{sid}, \mathbf{a})$ from the environment such that $\mathcal{P}_\mathsf{B} \| \mathcal{P}_\mathsf{A} \| \mathsf{sid}' = \mathsf{sid}$ where $\mathsf{sid}'$ is a fresh value, she computes

$$\mathbf{e} \leftarrow \mathbb{F}_q^{\ell \times \rho}$$
$$\dot{\mathbf{a}} \leftarrow \mathbb{F}_q^{\xi \times \ell}$$
$$\dot{\mathbf{e}} \leftarrow \mathbb{F}_q^{\xi \times \ell \times \rho}$$
$$\mathbf{c} := \{\langle \mathbf{g}, \dot{\mathbf{a}}_{*,i} \rangle\}_{i \in [\ell]}$$
$$\boldsymbol{\alpha}^0 := \{\dot{\mathbf{a}}_{j,*} \| \dot{\mathbf{e}}_{j,*,1} \| \ldots \| \dot{\mathbf{e}}_{j,*,\rho}\}_{j \in [\xi]}$$
$$\boldsymbol{\alpha}^1 := \left\{ \{\dot{\mathbf{a}}_{j,i} + \mathbf{a}_i\}_{i \in [\ell]} \,\|\, \{\dot{\mathbf{e}}_{j,i,1} + \mathbf{e}_{i,1}\}_{i \in [\ell]} \,\|\, \ldots \,\|\, \{\dot{\mathbf{e}}_{j,i,\rho} + \mathbf{e}_{i,\rho}\}_{i \in [\ell]} \right\}_{j \in [\xi]}$$

and sends $(\texttt{messages}, \mathsf{sid} \| j, \boldsymbol{\alpha}^0_{j,*}, \boldsymbol{\alpha}^1_{j,*})$ to $[\![\mathcal{F}_{\mathsf{SCOT\text{-}SIRIA}}(\mathbb{F}_q^{(1+\rho)\cdot\ell})]\!]_{\mathsf{PV}}$ for every $j \in [\xi]$ and outputs $(\texttt{vector-loaded}, \mathsf{sid})$ to the environment.

6. On receiving $(\texttt{chosen-message}, \mathsf{sid} \| i, \boldsymbol{\gamma}_{j,[(1+\rho)\cdot\ell]})$ from $[\![\mathcal{F}_{\mathsf{SCOT\text{-}SIRIA}}(\mathbb{F}_q^{(1+\rho)\cdot\ell})]\!]_{\mathsf{PV}}$ for every $j \in [\xi]$, Bob outputs $(\texttt{vector-loaded}, \mathsf{sid})$ to the environment. He samples $\mathbf{f} \leftarrow \mathbb{F}_q^\ell$, and broadcasts $(\texttt{challenge}, \mathsf{sid}, \mathbf{f})$. [a]

7. When Alice receives $(\texttt{challenge}, \mathsf{sid}, \mathbf{f})$, she computes

$$\mathbf{u} := \left\{ \{ \mathbf{a}_i \cdot \mathbf{f}_i + \mathbf{e}_{i,k} \}_{k \in [\rho]} \right\}_{i \in [\ell]}$$
$$\mathbf{v} := \left\{ \left\{ \{ \dot{\mathbf{a}}_{j,i} \cdot \mathbf{f}_i + \dot{\mathbf{e}}_{j,i,k} \}_{k \in [\rho]} \right\}_{j \in [\xi]} \right\}_{i \in [\ell]}$$

and then she broadcasts $(\texttt{checks}, \mathsf{sid}, \mathbf{u}, \mathbf{v})$.[b]

8. Upon receiving $(\texttt{checks}, \mathsf{sid}, \mathbf{u}, \mathbf{v})$ from Alice, Bob computes

$$\mathbf{v}' := \left\{ \left\{ \{ \mathbf{u}_{i,k} \cdot \boldsymbol{\beta}_j - \mathbf{f}_i \cdot \boldsymbol{\gamma}_{j,i} - \boldsymbol{\gamma}_{j,i+\ell \cdot k} \}_{k \in [\rho]} \right\}_{j \in [\xi]} \right\}_{i \in [\ell]}$$

and checks whether $\mathbf{v} = \mathbf{v}'$.[c] If this relation holds, then Bob computes

$$\mathbf{d} := \left\{ \langle \mathbf{g}, \boldsymbol{\gamma}_{*,i} \rangle \right\}_{i \in [\ell]}$$

and broadcasts $(\texttt{ok}, \mathsf{sid})$, and then he outputs $(\texttt{share}, \mathsf{sid}, \mathbf{d})$ and $(\texttt{vole-done}, \mathsf{sid})$ to the environment. On the other hand, if $\mathbf{v} \neq \mathbf{v}'$ then Bob broadcasts $(\texttt{jaccuse}, \mathsf{sid})$ and jumps to step 10.

9. If Alice receives $(\texttt{ok}, \mathsf{sid})$ from Bob, then she outputs $(\texttt{share}, \mathsf{sid}, \mathbf{c})$ and $(\texttt{vole-done}, \mathsf{sid})$ to the environment. If she receives $(\texttt{jaccuse}, \mathsf{sid})$ then she jumps to step 10. Regardless, she sends $(\texttt{observe}, \mathsf{sid})$ to $[\![\mathcal{F}_{\mathsf{Com}}]\!]_{\mathsf{PV}}$.

**Opening:**

10. On receiving $(\texttt{open}, \mathsf{sid})$ from the environment or jumping from step 8, Bob sends $(\texttt{commit}, \mathsf{sid}, (\boldsymbol{\beta}, \boldsymbol{\gamma}))$ to $[\![\mathcal{F}_{\mathsf{Com}}]\!]_{\mathsf{PV}}$.

11. On receiving $(\texttt{committed}, \mathsf{sid})$ from $[\![\mathcal{F}_{\mathsf{Com}}]\!]_{\mathsf{PV}}$, if Bob did not previously broadcast $(\texttt{jaccuse}, \mathsf{sid})$, then Alice outputs $(\texttt{open-req}, \mathsf{sid}, \mathcal{P}_{\mathsf{B}})$ to the environment.

12. On receiving $(\texttt{open}, \mathsf{sid})$ from the environment from the environment or jumping from step 9, Alice broadcasts $(\texttt{open}, \mathsf{sid})$.

13. When Bob receives $(\texttt{open}, \mathsf{sid})$ from Alice, if Bob did not previously broadcast $(\texttt{jaccuse}, \mathsf{sid})$, he outputs $(\texttt{open-req}, \mathsf{sid}, \mathcal{P}_{\mathsf{A}})$ to the environment.

14. When Alice completes steps 11 and 12, she sends $(\texttt{open}, \mathsf{sid} \| j)$ to $[\![\mathcal{F}_{\mathsf{SCOT\text{-}SIRIA}}(\mathbb{F}_q^{(1+\rho)\cdot\ell})]\!]_{\mathsf{PV}}$ for every $j \in [\xi]$.

15. If Bob receives $(\texttt{abort}, \mathsf{sid} \| j, \mathcal{P}_{\mathsf{A}})$ from $[\![\mathcal{F}_{\mathsf{SCOT\text{-}SIRIA}}(\mathbb{F}_q^{(1+\rho)\cdot\ell})]\!]_{\mathsf{PV}}$ for any $j \in [\xi]$, then he outputs $(\texttt{abort}, \mathsf{sid}, \mathcal{P}_{\mathsf{A}})$ to the environment

and ignores all future instructions involving $\mathsf{sid}$. If Bob receives $(\mathtt{messages}, \mathsf{sid} \| j, \boldsymbol{\alpha}^0_{j,*}, \boldsymbol{\alpha}^1_{j,*})$ for every $j \in [\xi]$, then he computes

$$\mathbf{a} := \left\{\boldsymbol{\alpha}^1_{1,i} - \boldsymbol{\alpha}^0_{1,i}\right\}_{i\in[\ell]}$$
$$\mathbf{e} := \left\{\left\{\boldsymbol{\alpha}^1_{1,i+k\cdot\ell} - \boldsymbol{\alpha}^0_{1,i+k\cdot\ell}\right\}_{k\in[\rho]}\right\}_{i\in[\ell]}$$

and then for $i \in [\ell]$ and $j \in [\xi]$ and $k \in [\rho]$ he verifies that

$$\begin{aligned}\mathbf{a}_i &= \boldsymbol{\alpha}^1_{j,i} - \boldsymbol{\alpha}^0_{j,i}\\ \mathbf{e}_{i,k} &= \boldsymbol{\alpha}^1_{j,i+k\cdot\ell} - \boldsymbol{\alpha}^0_{j,i+k\cdot\ell}\\ \mathbf{u}_{i,k} &= \mathbf{a}_i \cdot \mathbf{f}_i + \mathbf{e}_{i,k}\\ \mathbf{v}_{i,j,k} &= \boldsymbol{\alpha}^0_{j,i} \cdot \mathbf{f}_i + \boldsymbol{\alpha}^0_{j,i+k\cdot\ell}\end{aligned}$$

and if any of these conditions do not hold,[d] then he outputs $(\mathtt{abort}, \mathsf{sid}, \mathcal{P}_{\mathsf{A}})$ to the environment and ignores all future instructions involving $\mathsf{sid}$. If all of the conditions hold, then he sends $(\mathtt{decommit}, \mathsf{sid})$ to $[\![\mathcal{F}_{\mathsf{Com}}]\!]_{\mathsf{PV}}$ and computes

$$\mathbf{c} := \left\{\left\langle \mathbf{g}, \boldsymbol{\alpha}^0_{*,i}\right\rangle\right\}_{i\in[\ell]}$$

and outputs $(\mathtt{vole}, \mathsf{sid}, \mathbf{a}, b, \mathbf{c}, \mathbf{d})$ to the environment. Afterward, he ignores all instructions referencing $\mathsf{sid}$.

16. On receiving $(\mathtt{decommitted}, \mathsf{sid}, (\boldsymbol{\beta}, \boldsymbol{\gamma}))$ from $[\![\mathcal{F}_{\mathsf{Com}}]\!]_{\mathsf{PV}}$, Alice verifies for $i \in [\ell]$ and $j \in [\xi]$ that

$$(\boldsymbol{\alpha}^0_{j,i} = \boldsymbol{\gamma}_{j,i} \wedge \boldsymbol{\beta}_j = 0) \vee (\boldsymbol{\alpha}^1_{j,i} = \boldsymbol{\gamma}_{j,i} \wedge \boldsymbol{\beta}_j = 1)$$

and if these equalities do not hold, or if Bob previously broadcasted $(\mathtt{jaccuse}, \mathsf{sid})$, then Alice outputs $(\mathtt{abort}, \mathsf{sid}, \mathcal{P}_{\mathsf{B}})$ to the environment. If all of the aforementioned equalities do hold and Bob never sent $(\mathtt{jaccuse}, \mathsf{sid})$, then Alice computes

$$b := \langle \mathbf{g}, \boldsymbol{\beta}\rangle$$
$$\mathbf{d} := \left\{\left\langle \mathbf{g}, \left\{\dot{\mathbf{a}}_{j,i} + \mathbf{a}_i \cdot \boldsymbol{\beta}_j\right\}_{j\in[\xi]}\right\rangle\right\}_{i\in[\ell]}$$

and outputs $(\mathtt{vole}, \mathsf{sid}, \mathbf{a}, b, \mathbf{c}, \mathbf{d})$ to the environment. Afterward, she ignores all instructions referencing $\mathsf{sid}$.

**Verification:**

17. If there is an observing verifier $\mathcal{V}$, then upon receiving $(\texttt{observe}, \mathsf{sid})$ from the environment, $\mathcal{V}$ sends $(\texttt{observe}, \mathsf{sid}\|j)$ to $[\![\mathcal{F}_{\mathsf{SCOT\text{-}SIRIA}}(\mathbb{F}_q^{(1+\rho)\cdot\ell})]\!]_{\mathsf{PV}}$ for every $j \in [\xi]$.

18. If $\mathcal{V}$ receives $(\texttt{abort}, \mathsf{sid}\|j, \mathcal{P}_c)$ from $[\![\mathcal{F}_{\mathsf{SCOT\text{-}SIRIA}}(\mathbb{F}_q^{(1+\rho)\cdot\ell})]\!]_{\mathsf{PV}}$ for any $j \in [\xi]$, then it outputs $(\texttt{abort}, \mathsf{sid}, \mathcal{P}_c)$ to the environment.

19. If $\mathcal{V}$ receives $(\texttt{choice-made}, \mathsf{sid}\|j)$ from $[\![\mathcal{F}_{\mathsf{SCOT\text{-}SIRIA}}(\mathbb{F}_q^{(1+\rho)\cdot\ell})]\!]_{\mathsf{PV}}$ for every $j \in [\xi]$, then it outputs $(\texttt{correlation-loaded}, \mathsf{sid})$ to the environment.

20. If $\mathcal{V}$ receives $(\texttt{messages-loaded}, \mathsf{sid}\|j)$ from $[\![\mathcal{F}_{\mathsf{SCOT\text{-}SIRIA}}(\mathbb{F}_q^{(1+\rho)\cdot\ell})]\!]_{\mathsf{PV}}$ for every $j \in [\xi]$, then it outputs $(\texttt{vector-loaded}, \mathsf{sid})$ to the environment.

21. When Bob broadcasts $(\texttt{challenge}, \mathsf{sid}, \mathbf{f})$ and Alice responds with $(\texttt{checks}, \mathsf{sid}, \mathbf{u}, \mathbf{v})$, $\mathcal{V}$ outputs $(\texttt{vole-done}, \mathsf{sid})$ to the environment if Bob subsequently broadcasts $(\texttt{ok}, \mathsf{sid})$. If Bob broadcasts $(\texttt{jaccuse}, \mathsf{sid})$ instead, then $\mathcal{V}$ jumps to step 24. Regardless $\mathcal{V}$ sends $(\texttt{observe}, \mathsf{sid})$ to $[\![\mathcal{F}_{\mathsf{Com}}]\!]_{\mathsf{PV}}$.

22. On receiving $(\texttt{open}, \mathsf{sid})$ from Alice, if Bob did not previously broadcast $(\texttt{jaccuse}, \mathsf{sid})$, then $\mathcal{V}$ outputs $(\texttt{open-req}, \mathsf{sid}, \mathcal{P}_\mathsf{A})$ to the environment.

23. When $\mathcal{V}$ receives $(\texttt{committed}, \mathsf{sid})$ from $[\![\mathcal{F}_{\mathsf{Com}}]\!]_{\mathsf{PV}}$, if Bob did not previously broadcast $(\texttt{jaccuse}, \mathsf{sid})$, then $\mathcal{V}$ outputs $(\texttt{open-req}, \mathsf{sid}, \mathcal{P}_\mathsf{B})$ to the environment.

24. If $\mathcal{V}$ receives $(\texttt{abort}, \mathsf{sid}\|j, \mathcal{P}_c)$ from $[\![\mathcal{F}_{\mathsf{SCOT\text{-}SIRIA}}(\mathbb{F}_q^{(1+\rho)\cdot\ell})]\!]_{\mathsf{PV}}$ for any $j \in [\xi]$ after it has received $(\texttt{committed}, \mathsf{sid})$ from $[\![\mathcal{F}_{\mathsf{Com}}]\!]_{\mathsf{PV}}$, then it outputs $(\texttt{abort}, \mathsf{sid}, \mathcal{P}_c)$ to the environment.

25. If after $\mathcal{V}$ has received $(\texttt{committed}, \mathsf{sid})$ from $[\![\mathcal{F}_{\mathsf{Com}}]\!]_{\mathsf{PV}}$, $\mathcal{V}$ receives $(\texttt{messages}, \mathsf{sid}\|j, \boldsymbol{\alpha}^0_{j,*}, \boldsymbol{\alpha}^1_{j,*})$ from $[\![\mathcal{F}_{\mathsf{SCOT\text{-}SIRIA}}(\mathbb{F}_q^{(1+\rho)\cdot\ell})]\!]_{\mathsf{PV}}$ for every $j \in [\xi]$, then $\mathcal{V}$ checks the same conditions as Bob does in step 15, and if they do not hold, then $\mathcal{V}$ outputs $(\texttt{abort}, \mathsf{sid}, \mathcal{P}_\mathsf{A})$ to the environment.

26. When $\mathcal{V}$ receives $(\texttt{decommitted}, \mathsf{sid}, (\boldsymbol{\beta}, \boldsymbol{\gamma}))$ from $[\![\mathcal{F}_{\mathsf{Com}}]\!]_{\mathsf{PV}}$, $\mathcal{V}$ checks the same conditions as Alice does in step 16, and if they do not hold, or if Bob previously broadcasted $(\texttt{jaccuse}, \mathsf{sid})$, then $\mathcal{V}$ outputs $(\texttt{abort}, \mathsf{sid}, \mathcal{P}_\mathsf{B})$ to the environment. Otherwise, $\mathcal{V}$ computes $\mathbf{a}$ and $\mathbf{c}$ as Bob does in step 15 and $b$ and $\mathbf{d}$ as Alice does in step 16, and outputs $(\texttt{vole}, \mathsf{sid}, \mathbf{a}, b, \mathbf{c}, \mathbf{d})$ to the environment.

[a] In the Random Oracle Model, this round of interaction can be eliminated by generating the challenge using non-interactive coin-tossing: the parties simply put the contents of the broadcast channel into the random oracle to generate a challenge.

[b] Here again there is an optimization in the Random Oracle Model: Alice can send $v := \mathsf{RO}(\mathbf{v})$ instead of sending $\mathbf{v}$ itself.

[c] Or $v = \mathsf{RO}(\mathbf{v}')$.

[d] If the previously mentioned Random Oracle Model optimizations are applied, Bob must modify the final equality in the natural way and also check that Alice computed $\mathbf{f}$ correctly.

**Theorem 5.4.** *For any finite field $\mathbb{F}$ and any $\ell \in \mathbb{N}^+$, $\pi_{\mathsf{CVOLE\text{-}IRIA\text{-}PV}}(\mathbb{F}, \ell, \lambda_{\mathsf{s}})$ statistically UC-realizes $[\![\mathcal{F}_{\mathsf{CVOLE\text{-}IRIA}}(\mathbb{F}, \ell)]\!]_{\mathsf{PV}}$ against a malicious adversary statically corrupting at most one active participant and any number of passive verifiers in the $[\![\mathcal{F}_{\mathsf{SCOT\text{-}SIRIA}}]\!]_{\mathsf{PV}}$-hybrid model.*

*Proof Sketch.* The **Correlation Input** and **Vector Input** phases of $\pi_{\mathsf{CVOLE\text{-}IRIA\text{-}PV}}$ closely follow the OT-based multiplication protocols of Doerner et al. [DKLs18, DKLs19] with the input reuse and check-value repetition modifications of Chen et al. [CCD+20], differing only in that they are written in the hybrid model of many individual OT instances rather than the hybrid model of a single OT-extension instance, and in that Bob supplies the challenge $f$ interactively, whereas in the Doerner et al. protocols it is generated using a random oracle. The simulation and security argument for our protocol follows theirs exactly, until an abort occurs or the parties elect to `open`.

Since Alice and Bob's roles are asymmetric, we have two different simulation strategies in the case of an abort or opening. If Bob is corrupt, then the simulator will have extracted $\boldsymbol{\beta}$ and for every $j \in [\xi]$ and $i \in [(1+\rho)\cdot\ell]$ it will have sampled a uniform value for $\boldsymbol{\gamma}_{j,i} = \boldsymbol{\alpha}^0_{j,i}$ if $\boldsymbol{\beta}_j = 0$ or $\boldsymbol{\gamma}_{j,i} = \boldsymbol{\alpha}^1_{j,i}$ if $\boldsymbol{\beta}_j = 1$. When the simulator learns $\mathbf{a}$, it computes $\mathbf{e}_{i,k} := \mathbf{u}_{i,k} - \mathbf{a}_i \cdot f$ for every $i \in [\ell]$ and $k \in [\rho]$ and it fixes the heretofore unfixed OT message values such that they satisfy $\boldsymbol{\alpha}^1_{j,i} - \boldsymbol{\alpha}^0_{j,i} = \mathbf{a}_i$ and $\boldsymbol{\alpha}^1_{j,i+k\cdot\ell} - \boldsymbol{\alpha}^0_{j,i+k\cdot\ell} = \mathbf{e}_{i,k}$ for every $k \in [\rho]$, $j \in [\xi]$, and $i \in [\ell]$. These messages are then sent to Bob on behalf of the relevant $[\![\mathcal{F}_{\mathsf{SCOT\text{-}SIRIA}}]\!]_{\mathsf{PV}}$ instances. Apart from this, the simulator behaves exactly as Alice would: aborting and identifying Bob if he decommits a value for $\boldsymbol{\gamma}_j$ for some $j$ that does not match either $\boldsymbol{\alpha}^0$ or $\boldsymbol{\alpha}^1$, and aborting similarly if he accuses Alice of cheating.

If Alice is corrupt, the simulator will have extracted both $\boldsymbol{\alpha}^0$ and $\boldsymbol{\alpha}^1$. When the opening process begins, the simulator learns Bob's true input $b$. If after Alice instructs all instances of $[\![\mathcal{F}_{\mathsf{SCOT\text{-}SIRIA}}]\!]_{\mathsf{PV}}$ to open, it is revealed that she has cheated (in a way that was not detected earlier), then the simulator instructs the functionality to abort and identify her as corrupt. If she has not cheated, then the simulator generates an encoding $\boldsymbol{\beta}$ of $b$ in exactly the same manner as Bob does in step 2 of the protocol. The simulator then computes $\boldsymbol{\gamma}_{j,i} := \boldsymbol{\alpha}^1_{j,i} \cdot \boldsymbol{\beta}_j + \boldsymbol{\alpha}^0_{j,i} \cdot (1 - \boldsymbol{\beta}_j)$ for every $j \in [\xi]$ and decommits every $\boldsymbol{\gamma}$ on behalf of $[\![\mathcal{F}_{\mathsf{Com}}]\!]_{\mathsf{PV}}$. □

### 5.3 CVOLE with Single-Sided Input Revealing

A simple adjustment of $\mathcal{F}_{\text{CVOLE-IRIA}}$ and $\pi_{\text{CVOLE-IRIA-PV}}$ allows us to create a *one-sided* variant. This is in some sense analogous to the functionality we specify for OT and OT extension in sections 3 and 4, where Alice is the *sender*. Thus we refer to our modified functionality as SCVOLE-SIRIA.

The change to the functionality is simple: the functionality reveals only the inputs and outputs of $\mathcal{P}_\mathsf{A}$ upon receiving the `open` instruction from $\mathcal{P}_\mathsf{A}$ alone. That is, $\mathcal{P}_\mathsf{B}$ need send no message to trigger the **Opening** phase, and only $\mathbf{a}$ and $\mathbf{c}$ are released. Likewise in the case of an abort, only the inputs and outputs of $\mathcal{P}_\mathsf{A}$ are revealed. The protocol changes in the following way: steps 10, 11, and 13 are eliminated completely; in step 15, $\mathcal{P}_\mathsf{B}$ does not `decommit` anything; and in step 16, $\mathcal{P}_\mathsf{A}$ does not check the values that $\mathcal{P}_\mathsf{B}$ no longer decommits. From these modifications, the changes to the instructions of the passive verifiers $\mathcal{V}$ can easily be inferred.

Given the above modification, it is possible to modify the functionality such that the *sender's* input (i.e. Bob's input) is vectorized, rather than Alice's. Chen et al. [CCD+20] give a mechanism for forcing Alice to reuse her input with more than one of Bob's inputs. We simply apply this mechanism instead of the mechanism for forcing Bob to reuse his inputs. All else follows directly from what we have described.

# 6 From CVOLE-IRIA to Sampling Functionalities and a Security-Enhancing Protocol Compiler

In this section we show how to use the Committed Vector OLE functionality introduced in section 5 to construct generic MPC with plain identifiable abort via purely information theoretic techniques. As our first stepping stone, we construct generic MPC with input-revealing identifiable abort. We use the well-known information-theoretically secure online phase of the SPDZ [DPSZ12] protocol, which requires a certain type of correlated randomness and typically achieves only security with abort. However, a simple modification allows it to be upgraded to input-revealing identifiable abort: we use a "preprocessing" functionality to sample the correlated randomness, which has IA and also supports voluntary public opening of all parties' outputs. In the case that there is an abort in the online phase, the preprocessing functionality is instructed to open its outputs, and afterward anyone can verify whether the protocol parties sent messages that were consistent with their internal states. We begin by giving the preprocessing functionality.

**Functionality 6.1.** $\mathcal{F}_{\text{SPDZPrep-IA}}(n, \mathbb{F}_q, \lambda_\mathsf{s})$**: SPDZ Preprocessing**

This functionality interacts with $n$ active participants, $\mathcal{P}_1, \ldots, \mathcal{P}_n$. In addition to the party count, it is parameterized by a finite field $\mathbb{F}_q$ of size

$q$ over which the correlated randomness is generated, and a statistical parameter $\lambda_{\mathsf{s}}$. For convenience, we define $\rho := \lceil \lambda_{\mathsf{s}} / |q| \rceil$ to be the number of $\mathbb{F}_q$ elements required for a $\lambda_{\mathsf{s}}$-bit MAC.

**SPDZ Preprocessing:** On receiving $(\mathtt{prep}, \mathsf{sid}, \mathbf{u}, v)$ from some $\mathcal{P}_i$ such that $\mathcal{P}_1 \| \ldots \| \mathcal{P}_n \| \mathsf{sid}' = \mathsf{sid}$ for some fresh $\mathsf{sid}'$, send $(\mathtt{prep\text{-}req}, \mathsf{sid}, \mathbf{u}, v, \mathcal{P}_i)$ to all parties. On receiving $(\mathtt{prep}, \mathsf{sid}, \mathbf{u}, v)$ from every $\mathcal{P}_i$ for $i \in [n]$, wait for $\mathcal{S}$ to send $(\mathtt{proceed}, \mathsf{sid})$ or $(\mathtt{abort}, \mathsf{sid}, \mathcal{P}_c)$ such that $c \in [n]$ and $\mathcal{P}_c$ is corrupt. In the latter case, forward $(\mathtt{abort}, \mathsf{sid}, \mathcal{P}_c)$ to all parties and ignore all future messages with the same $\mathsf{sid}$. In the former case (i.e. when $\mathcal{S}$ sends $\mathtt{proceed}$):

1. For every $i \in [n]$ such that $\mathcal{P}_i$ is corrupt, wait for $\mathcal{S}$ to send $(\mathtt{adv\text{-}shares}, \mathsf{sid}, i, \boldsymbol{\delta}_{i,[\rho]}, \mathbf{m}_{i,[\mathbf{u}_i]}, \{\mathbf{w}_{i,j,[\mathbf{u}_j]}\}_{j\in[n]}, \mathbf{x}_{i,[v]}, \mathbf{y}_{i,[v]}, \mathbf{z}_{i,[v]})$.

2. For every $i \in [n]$ such that $\mathcal{P}_i$ is honest, sample

$$\boldsymbol{\delta}_{i,[\rho]} \leftarrow \mathbb{F}_q^{\rho} \qquad \mathbf{m}_{i,[\mathbf{u}_i]} \leftarrow \mathbb{F}_q^{\mathbf{u}_i} \qquad \mathbf{w}_{i,j,[\mathbf{u}_j]} \leftarrow \mathbb{F}_q^{\mathbf{u}_j} \quad \text{for } j \in [n]$$

$$\mathbf{x}_{i,[v]} \leftarrow \mathbb{F}_q^{v} \qquad \mathbf{y}_{i,[v]} \leftarrow \mathbb{F}_q^{v} \qquad \mathbf{z}_{i,[v]} \leftarrow \mathbb{F}_q^{v}$$

   subject for ever $k \in [v]$ to

$$\sum_{j\in[n]} \mathbf{z}_{j,k} = \sum_{j\in[n]} \mathbf{x}_{j,k} \cdot \sum_{j\in[n]} \mathbf{y}_{j,k}$$

   and subject to

$$\mathbf{m} = \left\{ \left\{ \sum_{i\in[n]} \mathbf{w}_{i,j,k} \right\}_{k\in[\mathbf{u}_j]} \right\}_{j\in[n]}$$

3. Sample

$$\mathring{\mathbf{w}}_{[n],j,[\mathbf{u}_j],[\rho]} \leftarrow \mathbb{F}_q^{n\times \mathbf{u}_j \times \rho} \quad \text{for } j \in [n] \qquad \mathring{\mathbf{x}}, \mathring{\mathbf{y}}, \mathring{\mathbf{z}} \leftarrow \mathbb{F}_q^{n\times v\times \rho}$$

   subject for every $k \in [v]$ and $l \in [\rho]$ to

$$\sum_{i\in[n]} \mathring{\mathbf{x}}_{i,k,l} = \sum_{i\in[n]} \boldsymbol{\delta}_{i,l} \cdot \sum_{i\in[n]} \mathbf{x}_{i,k}$$

$$\sum_{i\in[n]} \mathring{\mathbf{y}}_{i,k,l} = \sum_{i\in[n]} \boldsymbol{\delta}_{i,l} \cdot \sum_{i\in[n]} \mathbf{y}_{i,k}$$

$$\sum_{i\in[n]} \mathring{\mathbf{z}}_{i,k,l} = \sum_{i\in[n]} \boldsymbol{\delta}_{i,l} \cdot \sum_{i\in[n]} \mathbf{z}_{i,k}$$

and subject for every $j \in [n]$ and $k \in [\mathbf{u}_j]$ and $l \in [\rho]$ to

$$\sum_{i\in[n]} \mathring{\mathbf{w}}_{i,j,k,l} = \sum_{i\in[n]} \boldsymbol{\delta}_{i,l} \cdot \sum_{i\in[n]} \mathbf{w}_{i,j,k}$$

4. Output

$$(\mathtt{prep}, \mathsf{sid}, \boldsymbol{\delta}_{i,*}, \mathbf{m}_{i,*}, \mathbf{w}_{i,*,*}, \mathring{\mathbf{w}}_{i,*,*,*}, \mathbf{x}_{i,*}, \mathring{\mathbf{x}}_{i,*,*}, \mathbf{y}_{i,*}, \mathring{\mathbf{y}}_{i,*,*}, \mathbf{z}_{i,*}, \mathring{\mathbf{z}}_{i,*,*})$$

to $\mathcal{P}_i$ for every $i \in [n]$ and store $(\mathtt{prep}, \mathsf{sid}, \boldsymbol{\delta}, \mathbf{m}, \mathbf{w}, \mathring{\mathbf{w}}, \mathbf{x}, \mathring{\mathbf{x}}, \mathbf{y}, \mathring{\mathbf{y}}, \mathbf{z}, \mathring{\mathbf{z}})$ in memory.

**Opening:** On receiving $(\mathtt{open}, \mathsf{sid})$ from some $\mathcal{P}_i$, send $(\mathtt{open\text{-}req}, \mathsf{sid}, \mathcal{P}_i)$ to all parties. On receiving $(\mathtt{open}, \mathsf{sid})$ from some $\mathcal{P}_i$ for every $i \in [n]$, if $\mathcal{S}$ sends $(\mathtt{proceed}, \mathsf{sid})$ or all active participants are honest, then find the record $(\mathtt{prep}, \mathsf{sid}, \boldsymbol{\delta}, \mathbf{m}, \mathbf{w}, \mathring{\mathbf{w}}, \mathbf{x}, \mathring{\mathbf{x}}, \mathbf{y}, \mathring{\mathbf{y}}, \mathbf{z}, \mathring{\mathbf{z}})$ in memory and send it to all parties. If $\mathcal{S}$ sends $(\mathtt{abort}, \mathsf{sid}, \mathcal{P}_c)$ such that $\mathcal{P}_c$ is corrupt, then send $(\mathtt{abort}, \mathsf{sid}, \mathcal{P}_c)$ to all parties. Regardless, ignore all future instructions with this $\mathsf{sid}$.

The above functionality produces secret shares of a MAC key, $\delta$, secret shares of a set of Beaver triples, where each triple contains three elements $(x, y, z)$ such that $x \cdot y = z$, secret shares of a set of masks (denoted $m$), and secret shares of linear MACs on both the masks, and the Beaver triple components. To realize the above functionality in the $[\![\mathcal{F}_{\mathsf{CVOLE\text{-}IRIA}}]\!]_{\mathsf{PV}}$-hybrid model, we adapt the MASCOT protocol [KOS16]. Because we work with VOLE rather than with OT, the original protocol can be simplified substantially: pairwise VOLE instances can be used to compute shares of $z$ given shares of $x$ and $y$, and a second set of pairwise VOLE instances can be used to calculate the MACs, while enforcing that the MAC key shares are used consistently. Nothing enforces that a cheating party uses the same share of $z$ as input for the second set of VOLE instances that it received as output from the first set, and so MASCOT includes a statistical check to ensure that this is the case, which we retain.

**Protocol 6.2.** $\pi_{\mathsf{MASCOT\text{-}IA\text{-}PV}}(n, \mathbb{F}_q, \lambda_{\mathsf{s}})$**: MASCOT with IA**

This protocol involves $n$ active participants, $\mathcal{P}_1, \ldots, \mathcal{P}_n$, an a-priori-unknown number of passive verifiers (collectively denoted $\mathcal{V}$), and the ideal functionalities $[\![\mathcal{F}_{\mathsf{CVOLE\text{-}IRIA}}]\!]_{\mathsf{PV}}$, $[\![\mathcal{F}_{\mathsf{Com}}]\!]_{\mathsf{PV}}$, and $[\![\mathcal{F}_{\mathsf{CT}}]\!]_{\mathsf{PV}}$. In addition to the party count, it is parameterized by a finite field $\mathbb{F}_q$ of size $q$ over which the correlated randomness is generated, and a statistical parameter $\lambda_{\mathsf{s}}$. For convenience, we define $\rho := \lceil \lambda_{\mathsf{s}} / |q| \rceil$ to be the number of $\mathbb{F}_q$ elements required for a $\lambda_{\mathsf{s}}$-bit MAC.

**SPDZ Preprocessing:**

1. When $\mathcal{P}_i$ receives $(\mathtt{prep}, \mathsf{sid}, \mathbf{u}, v)$ from the environment such that $\mathcal{P}_1 \| \ldots \| \mathcal{P}_n \| \mathsf{sid}' = \mathsf{sid}$ for some fresh $\mathsf{sid}'$, $\mathcal{P}_i$ forwards this message on the broadcast channel, computes $\mathbf{u}' := \{\mathbf{u}_j\}_{j \in [n]}$, samples

$$\mathbf{x}_{i,[v]} \leftarrow \mathbb{F}_q^v \qquad \text{and} \qquad \mathbf{y}_{i,[v]} \leftarrow \mathbb{F}_q^v$$

and for $j \in [n] \setminus \{i\}$ and $k \in [v]$, $\mathcal{P}_i$ sends $(\mathtt{vector}, \mathcal{P}_j \| \mathcal{P}_i \| \mathsf{sid} \| \mathtt{z} \| k, \{\mathbf{y}_{i,k}\})$ and $(\mathtt{correlation}, \mathcal{P}_i \| \mathcal{P}_j \| \mathsf{sid} \| \mathtt{z} \| k, \mathbf{x}_{i,k})$ to $[\![\mathcal{F}_{\mathsf{CVOLE\text{-}IRIA}}(\mathbb{F}_q, 1)]\!]_{\mathsf{PV}}$.

2. $\mathcal{P}_i$ sends $(\mathtt{observe}, \mathcal{P}_j \| \mathcal{P}_l \| \mathsf{sid} \| \mathtt{z} \| k)$ to $[\![\mathcal{F}_{\mathsf{CVOLE\text{-}IRIA}}(\mathbb{F}_q, 1)]\!]_{\mathsf{PV}}$ for every $j \in [n] \setminus \{i\}$ and $l \in [n] \setminus \{i, j\}$ and $k \in [v]$.

3. If $\mathcal{P}_i$ receives $(\mathtt{abort}, \mathcal{P}_j \| \mathcal{P}_l \| \mathsf{sid} \| \mathtt{z} \| k, \mathcal{P}_c)$ from $[\![\mathcal{F}_{\mathsf{CVOLE\text{-}IRIA}}(\mathbb{F}_q, 1)]\!]_{\mathsf{PV}}$ for any $j \in [n]$ and $l \in [n] \setminus \{j\}$ and $k \in [v]$, then $\mathcal{P}_i$ outputs $(\mathtt{abort}, \mathsf{sid}, \mathcal{P}_c)$ to the environment and ignores all future instructions involving $\mathsf{sid}$.

4. When $\mathcal{P}_i$ receives $(\mathtt{share}, \mathcal{P}_i \| \mathcal{P}_j \| \mathsf{sid} \| \mathtt{z} \| k, \{\mathbf{d}_{i,j,k}\})$ and $(\mathtt{share}, \mathcal{P}_j \| \mathcal{P}_i \| \mathsf{sid} \| \mathtt{z} \| k, \{\mathbf{c}_{i,j,k}\})$ and $(\mathtt{vole\text{-}done}, \mathcal{P}_j \| \mathcal{P}_l \| \mathsf{sid} \| \mathtt{z} \| k)$ from $[\![\mathcal{F}_{\mathsf{CVOLE\text{-}IRIA}}(\mathbb{F}_q, 1)]\!]_{\mathsf{PV}}$ for every $j \in [n] \setminus \{i\}$ and $l \in [n] \setminus \{i, j\}$ and $k \in [v]$, $\mathcal{P}_i$ samples

$$\boldsymbol{\delta}_{i,[\rho]} \leftarrow \mathbb{F}_q^\rho \qquad \mathbf{w}_{i,j,[\mathbf{u}'_j]} \leftarrow \mathbb{F}_q^{\mathbf{u}'_j} \quad \text{for } j \in [n] \qquad \mathbf{s}_{i,[v]}, \mathbf{t}_{i,[v]} \leftarrow \mathbb{F}_q^v$$

and computes

$$w := 5v + \sum_{j \in [n]} \mathbf{u}'_j$$

$$\mathbf{z}_{i,k} := \mathbf{x}_{i,k} \cdot \mathbf{y}_{i,k} + \sum_{j \in [n] \setminus \{i\}} (\mathbf{c}_{i,j,k} + \mathbf{d}_{i,j,k}) \quad \text{for every } k \in [v]$$

$$\mathbf{r}_{i,[w]} := \mathbf{w}_{i,1,*} \| \ldots \| \mathbf{w}_{i,n,*} \| \mathbf{x}_{i,*} \| \mathbf{y}_{i,*} \| \mathbf{z}_{i,*} \| \mathbf{s}_{i,*} \| \mathbf{t}_{i,*}$$

and then $\mathcal{P}_i$ sends $(\mathtt{correlation}, \mathcal{P}_i \| \mathcal{P}_j \| \mathsf{sid} \| \mathtt{mac} \| k, \boldsymbol{\delta}_{i,k})$ and $(\mathtt{vector}, \mathcal{P}_j \| \mathcal{P}_i \| \mathsf{sid} \| \mathtt{mac} \| k, \mathbf{r}_{i,*})$ to $[\![\mathcal{F}_{\mathsf{CVOLE\text{-}IRIA}}(\mathbb{F}_q, w)]\!]_{\mathsf{PV}}$ for every $j \in [n] \setminus \{i\}$ and $k \in [\rho]$.

5. $\mathcal{P}_i$ sends $(\mathtt{observe}, \mathcal{P}_j \| \mathcal{P}_l \| \mathsf{sid} \| \mathtt{mac} \| k)$ to $[\![\mathcal{F}_{\mathsf{CVOLE\text{-}IRIA}}(\mathbb{F}_q, w)]\!]_{\mathsf{PV}}$ for every $j \in [n] \setminus \{i\}$ and $l \in [n] \setminus \{i, j\}$ and $k \in [\rho]$.

6. If $\mathcal{P}_i$ receives $(\mathtt{abort}, \mathcal{P}_j \| \mathcal{P}_l \| \mathsf{sid} \| \mathtt{mac} \| k, \mathcal{P}_c)$ from $[\![\mathcal{F}_{\mathsf{CVOLE\text{-}IRIA}}(\mathbb{F}_q, w)]\!]_{\mathsf{PV}}$ for any $j \in [n]$ and $l \in [n] \setminus \{j\}$ and $k \in [\rho]$, then $\mathcal{P}_i$ outputs $(\mathtt{abort}, \mathsf{sid}, \mathcal{P}_c)$ to the environment and ignores all future instructions involving $\mathsf{sid}$.

7. When $\mathcal{P}_i$ receives $(\mathtt{share}, \mathcal{P}_i \| \mathcal{P}_j \| \mathsf{sid} \| \mathtt{mac} \| k, \mathring{\mathbf{d}}_{i,j,[w],k})$ and $(\mathtt{share}, \mathcal{P}_j \| \mathcal{P}_i \| \mathsf{sid} \| \mathtt{mac} \| k, \mathring{\mathbf{c}}_{i,j,[w],k})$ and $(\mathtt{vole\text{-}done}, \mathcal{P}_j \| \mathcal{P}_l \| \mathsf{sid} \| \mathtt{mac} \| k)$ from $[\![\mathcal{F}_{\mathsf{CVOLE\text{-}IRIA}}(\mathbb{F}_q, w)]\!]_{\mathsf{PV}}$ for every $j \in [n] \setminus \{i\}$ and $l \in [n] \setminus \{i, j\}$ and $k \in [\rho]$, $\mathcal{P}_i$ computes

$$\mathring{\mathbf{r}}_{i,l,k} \coloneqq \mathbf{r}_{i,l} \cdot \boldsymbol{\delta}_{i,k} + \sum_{j \in [n] \setminus \{i\}} \left( \mathring{\mathbf{c}}_{i,j,l,k} + \mathring{\mathbf{d}}_{i,j,l,k} \right) \quad \text{for every } l \in [w]$$

$$\mathring{\mathbf{r}}_{i,*,k} =: \mathring{\mathbf{w}}_{i,1,*,k} \| \ldots \| \mathring{\mathbf{w}}_{i,n,*,k} \| \mathring{\mathbf{x}}_{i,*,k} \| \mathring{\mathbf{y}}_{i,*,k} \| \mathring{\mathbf{z}}_{i,*,k} \| \mathring{\mathbf{s}}_{i,*,k} \| \mathring{\mathbf{t}}_{i,*,k}$$

for $k \in [\rho]$ and sends $(\mathtt{sample}, \mathsf{sid} \| 1)$ to $[\![\mathcal{F}_{\mathsf{CT}}(n, \mathbb{F}_q^{v \times \rho})]\!]_{\mathsf{PV}}$.

8. On receiving $(\mathtt{coins}, \mathsf{sid} \| 1, \boldsymbol{\sigma})$ from $[\![\mathcal{F}_{\mathsf{CT}}(n, \mathbb{F}_q^{v \times \rho})]\!]_{\mathsf{PV}}$, $\mathcal{P}_i$ computes

$$\boldsymbol{\tau}_{i,k,o} \coloneqq \boldsymbol{\sigma}_{k,o} \cdot \mathbf{x}_{i,k} - \mathbf{s}_{i,k}$$
$$\mathring{\boldsymbol{\tau}}_{i,k,l,o} \coloneqq \boldsymbol{\sigma}_{k,o} \cdot \mathring{\mathbf{x}}_{i,k,l} - \mathring{\mathbf{s}}_{i,k,l}$$

for every $k \in [v]$ and $l, o \in [\rho]$ and broadcasts $(\mathtt{triple\text{-}check}, \mathsf{sid}, \boldsymbol{\tau}_{i,*,*})$. $\mathcal{P}_i$ also broadcasts $(\mathtt{masks}, \mathsf{sid}, j, \mathbf{w}_{i,j,*})$ for every $j \in [n] \setminus \{i\}$.

9. When $\mathcal{P}_i$ receives $(\mathtt{triple\text{-}check}, \mathsf{sid}, \boldsymbol{\tau}_{j,[v],[\rho]})$ and $(\mathtt{masks}, \mathsf{sid}, i, \mathbf{w}_{j,i,*})$ from every $\mathcal{P}_j$, $\mathcal{P}_i$ computes

$$\boldsymbol{\phi}_{i,k,o} \coloneqq \boldsymbol{\sigma}_{k,o} \cdot \mathbf{z}_{i,k} - \mathbf{t}_{i,k} - \mathbf{y}_{i,k} \cdot \sum_{j \in [n]} \boldsymbol{\tau}_{j,k,o}$$
$$\mathring{\boldsymbol{\phi}}_{i,k,l,o} \coloneqq \boldsymbol{\sigma}_{k,o} \cdot \mathring{\mathbf{z}}_{i,k,l} - \mathring{\mathbf{t}}_{i,k,l} - \mathring{\mathbf{y}}_{i,k,l} \cdot \sum_{j \in [n]} \boldsymbol{\tau}_{j,k,o}$$

for every $k \in [v]$ and $l, o \in [\rho]$ and sends $(\mathtt{observe}, \mathcal{P}_j \| \mathsf{sid})$ to $[\![\mathcal{F}_{\mathsf{Com}}]\!]_{\mathsf{PV}}$ for every $j \in [n] \setminus \{i\}$ and $(\mathtt{sample}, \mathsf{sid} \| 2)$ to $[\![\mathcal{F}_{\mathsf{CT}}(n, \mathbb{F}_q^{\mathbf{u}'_1 \times \rho} \times \ldots \times \mathbb{F}_q^{\mathbf{u}'_n \times \rho} \times \mathbb{F}_q^{v \times \rho} \times \mathbb{F}_q^{v \times \rho})]\!]_{\mathsf{PV}}$.

10. On receiving $(\mathtt{coins}, \mathsf{sid} \| 2, \boldsymbol{\chi})$ from $[\![\mathcal{F}_{\mathsf{CT}}(n, \mathbb{F}_q^{\mathbf{u}'_1 \times \rho} \times \ldots \times \mathbb{F}_q^{\mathbf{u}'_n \times \rho} \times \mathbb{F}_q^{v \times \rho} \times \mathbb{F}_q^{v \times \rho})]\!]_{\mathsf{PV}}$, $\mathcal{P}_i$ computes

$$\mathbf{m}_{i,k} \coloneqq \sum_{j \in [n]} \mathbf{w}_{j,i,k} \quad \text{for } k \in [\mathbf{u}'_i]$$
$$\boldsymbol{\psi}_{i,o} \coloneqq \sum_{k \in [\mathbf{u}'_i]} \left( \boldsymbol{\chi}_{i,k,o} \cdot \mathbf{m}_{i,k} \right) \quad \text{for } o \in [\rho]$$

and broadcasts $(\mathtt{mask\text{-}check}, \mathsf{sid}, \boldsymbol{\psi}_{i,*})$.

11. When $\mathcal{P}_i$ receives $(\texttt{mask-check}, \mathsf{sid}, \boldsymbol{\psi}_{j,[\rho]})$ from every $\mathcal{P}_j$, $\mathcal{P}_i$ computes

$$\mathring{\boldsymbol{\psi}}_{i,l,o} := \sum_{j\in[n]} \sum_{k\in[\mathbf{u}'_j]} \boldsymbol{\chi}_{j,k,o} \cdot \mathring{\mathbf{w}}_{i,j,k,l} + \sum_{k\in[v]} \left( \boldsymbol{\chi}_{n+1,k,o} \cdot \mathring{\boldsymbol{\tau}}_{i,k,l,o} + \boldsymbol{\chi}_{n+2,k,o} \cdot \mathring{\boldsymbol{\phi}}_{i,k,l,o} \right)$$

$$\boldsymbol{\omega}_{i,l,o} := \mathring{\boldsymbol{\psi}}_{i,l,o} - \boldsymbol{\delta}_{i,l} \cdot \left( \sum_{j\in[n]} \boldsymbol{\psi}_{j,o} + \sum_{k\in[v]} \boldsymbol{\chi}_{n+1,k,o} \cdot \sum_{j\in[n]} \boldsymbol{\tau}_{j,k,o} \right)$$

for every $l, o \in [\rho]$ and sends $(\texttt{commit}, \mathcal{P}_i \| \mathsf{sid}, \boldsymbol{\omega}_{i,*,*})$ to $[\![\mathcal{F}_{\mathsf{Com}}]\!]_{\mathsf{PV}}$.

12. When $\mathcal{P}_i$ receives $(\texttt{committed}, \mathcal{P}_j \| \mathsf{sid})$ to $[\![\mathcal{F}_{\mathsf{Com}}]\!]_{\mathsf{PV}}$ for every $j \in [n] \backslash \{i\}$, it sends $(\texttt{decommit}, \mathcal{P}_i \| \mathsf{sid})$ to $[\![\mathcal{F}_{\mathsf{Com}}]\!]_{\mathsf{PV}}$.

13. When $\mathcal{P}_i$ receives $(\texttt{decommitted}, \mathcal{P}_j \| \mathsf{sid}, \boldsymbol{\omega}_{j,[\rho],[\rho]})$ from $[\![\mathcal{F}_{\mathsf{Com}}]\!]_{\mathsf{PV}}$ for every $j \in [n] \setminus \{i\}$, $\mathcal{P}_i$ verifies that

$$\sum_{j\in[n]} \boldsymbol{\omega}_{j,l,o} = 0$$

for every $l, o \in [\rho]$. If this equation holds, then $\mathcal{P}_i$ outputs

$$\left( \texttt{prep}, \mathsf{sid}, \boldsymbol{\delta}_{i,*}, \mathbf{m}_{i,[\mathbf{u}_i]}, \left\{ \mathbf{w}_{i,j,[\mathbf{u}_j]} \right\}_{j\in[n]}, \left\{ \mathring{\mathbf{w}}_{i,j,[\mathbf{u}_j],*} \right\}_{j\in[n]}, \mathbf{x}_{i,*}, \mathring{\mathbf{x}}_{i,*,*}, \mathbf{y}_{i,*}, \mathring{\mathbf{y}}_{i,*,*}, \mathbf{z}_{i,*}, \mathring{\mathbf{z}}_{i,*,*} \right)$$

to the environment broadcasts $(\texttt{ok}, \mathsf{sid})$, and does not go on to step 16. If the equation does not hold, then $\mathcal{P}_i$ broadcasts $(\texttt{jaccuse}, \mathsf{sid})$ and jumps to step 14.

**Opening:**

14. On receiving $(\texttt{open}, \mathsf{sid})$ from the environment $\mathcal{P}_i$, sends

- $(\texttt{open}, \mathcal{P}_i \| \mathcal{P}_j \| \mathsf{sid} \| \mathbf{z} \| k)$ to $[\![\mathcal{F}_{\mathsf{CVOLE\text{-}IRIA}}(\mathbb{F}_q, 1)]\!]_{\mathsf{PV}}$ for $k \in [v]$
- $(\texttt{open}, \mathcal{P}_j \| \mathcal{P}_i \| \mathsf{sid} \| \mathbf{z} \| k)$ to $[\![\mathcal{F}_{\mathsf{CVOLE\text{-}IRIA}}(\mathbb{F}_q, 1)]\!]_{\mathsf{PV}}$ for $k \in [v]$
- $(\texttt{open}, \mathcal{P}_i \| \mathcal{P}_j \| \mathsf{sid} \| \texttt{mac} \| k)$ to $[\![\mathcal{F}_{\mathsf{CVOLE\text{-}IRIA}}(\mathbb{F}_q, w)]\!]_{\mathsf{PV}}$ for $k \in [\rho]$
- $(\texttt{open}, \mathcal{P}_j \| \mathcal{P}_i \| \mathsf{sid} \| \texttt{mac} \| k)$ to $[\![\mathcal{F}_{\mathsf{CVOLE\text{-}IRIA}}(\mathbb{F}_q, w)]\!]_{\mathsf{PV}}$ for $k \in [\rho]$

for $j \in [n] \setminus \{i\}$.

15. On receiving the input and output records for all of the $[\![\mathcal{F}_{\text{CVOLE-IRIA}}]\!]_{\text{PV}}$ instances associated specifically with $\mathcal{P}_j$ in the foregoing protocol, if $\mathcal{P}_i$ did not jump to this phase from step 13, then $\mathcal{P}_i$ outputs $(\texttt{open-req}, \mathsf{sid}, \mathcal{P}_j)$ to the environment.

16. On receiving the input and output records for all of the $[\![\mathcal{F}_{\text{CVOLE-IRIA}}]\!]_{\text{PV}}$ instances involved in the foregoing protocol, $\mathcal{P}_i$ checks the work of all other parties in ascending order. If it finds a party $\mathcal{P}_c$ that has sent a message inconsistent with the values revealed by $[\![\mathcal{F}_{\text{CVOLE-IRIA}}]\!]_{\text{PV}}$, it outputs $(\texttt{abort}, \mathsf{sid}, \mathcal{P}_c)$. Otherwise, having calculated all input and output shares in the way the other parties would have, it outputs $(\texttt{prep}, \mathsf{sid}, \boldsymbol{\delta}, \mathbf{m}, \mathbf{w}, \mathring{\mathbf{w}}, \mathbf{x}, \mathring{\mathbf{x}}, \mathbf{y}, \mathring{\mathbf{y}}, \mathbf{z}, \mathring{\mathbf{z}})$ to the environment. Regardless, $\mathcal{P}_i$ ignores all future instructions involving $\mathsf{sid}$.

**Verification:**

17. If there is an observing verifier $\mathcal{V}$, then upon receiving $(\texttt{observe}, \mathsf{sid})$ from the environment, $\mathcal{V}$ waits to receive $(\texttt{prep}, \mathsf{sid}, \mathbf{u}, v)$ from any active participant on the broadcast channel, and then sends $(\texttt{observe}, \mathsf{sid}\|1)$ and $(\texttt{observe}, \mathsf{sid}\|2)$ to $[\![\mathcal{F}_{\mathsf{CT}}]\!]_{\text{PV}}$ and

    - $(\texttt{observe}, \mathcal{P}_i\|\mathcal{P}_j\|\mathsf{sid}\|\mathtt{z}\|k)$ to $[\![\mathcal{F}_{\text{CVOLE-IRIA}}(\mathbb{F}_q, 1)]\!]_{\text{PV}}$ for $k \in [v]$
    - $(\texttt{observe}, \mathcal{P}_j\|\mathcal{P}_i\|\mathsf{sid}\|\mathtt{z}\|k)$ to $[\![\mathcal{F}_{\text{CVOLE-IRIA}}(\mathbb{F}_q, 1)]\!]_{\text{PV}}$ for $k \in [v]$
    - $(\texttt{observe}, \mathcal{P}_i\|\mathcal{P}_j\|\mathsf{sid}\|\mathtt{mac}\|k)$ to $[\![\mathcal{F}_{\text{CVOLE-IRIA}}(\mathbb{F}_q, w)]\!]_{\text{PV}}$ for $k \in [\rho]$
    - $(\texttt{observe}, \mathcal{P}_j\|\mathcal{P}_i\|\mathsf{sid}\|\mathtt{mac}\|k)$ to $[\![\mathcal{F}_{\text{CVOLE-IRIA}}(\mathbb{F}_q, w)]\!]_{\text{PV}}$ for $k \in [\rho]$
    - $(\texttt{observe}, \mathcal{P}_i\|\mathsf{sid})$ to $[\![\mathcal{F}_{\mathsf{Com}}]\!]_{\text{PV}}$

    for $i \in [n]$ and $j \in [n] \setminus \{i\}$.

18. If any active participant broadcasts $(\texttt{jaccuse}, \mathsf{sid})$ in step 13, then $\mathcal{V}$ jumps to step 14 and runs the same code as the active participants would.

**Theorem 6.3.** *For any finite field $\mathbb{F}$ and any $n, \lambda_{\mathsf{s}} \in \mathbb{N}^+$, $\pi_{\text{MASCOT-IA-PV}}(n, \mathbb{F}, \lambda_{\mathsf{s}})$ statistically UC-realizes $[\![\mathcal{F}_{\text{SPDZPrep-IA}}(n, \mathbb{F}, \lambda_{\mathsf{s}})]\!]_{\text{PV}}$ against a malicious adversary statically corrupting up to $n - 1$ active participants in the $([\![\mathcal{F}_{\text{CVOLE-IRIA}}]\!]_{\text{PV}}, [\![\mathcal{F}_{\mathsf{Com}}]\!]_{\text{PV}}, [\![\mathcal{F}_{\mathsf{CT}}]\!]_{\text{PV}})$-hybrid model.*

*Proof Sketch.* Our protocol is simply the original MASCOT protocol, with beaver triple and MAC generation performed in a black-box way by $[\![\mathcal{F}_{\text{CVOLE-IRIA}}]\!]_{\text{PV}}$, and the MAC repeated sufficiently many times to achieve statistical security regardless of the field size. We observe that under these changes, the protocol is information theoretic, which makes simulation far easier than it was in the case of the original MASCOT. Any time the simulator must send a value on behalf of a functionality or honest party, and that value is not fixed by the existing protocol transcript, it samples one uniformly; this fixes the outputs

of the corrupt parties in a way that the simulator can predict, and the simulator then supplies these outputs to $[\![\mathcal{F}_{\mathsf{SPDZPrep\text{-}IA}}]\!]_{\mathsf{PV}}$ using the `adv-shares` interface.

Given any well-formed protocol output and any adversarial observation of the intermediate state of the protocol (which comprises the messages of honest parties, the outputs of $[\![\mathcal{F}_{\mathsf{CT}}]\!]_{\mathsf{PV}}$, and the inputs and outputs to instances of $[\![\mathcal{F}_{\mathsf{CVOLE\text{-}IRIA}}]\!]_{\mathsf{PV}}$ in which corrupt parties were active participants), it is always possible to retroactively compute a set of protocol-consistent inputs and outputs for the honest parties in their interactions with $[\![\mathcal{F}_{\mathsf{CVOLE\text{-}IRIA}}]\!]_{\mathsf{PV}}$. When the simulator receives instructions to `open` the protocol and learns the honest parties' ideal outputs from $[\![\mathcal{F}_{\mathsf{SPDZPrep\text{-}IA}}]\!]_{\mathsf{PV}}$, it does this, and then emits the calculated intermediate values on behalf of $[\![\mathcal{F}_{\mathsf{CVOLE\text{-}IRIA}}]\!]_{\mathsf{PV}}$. □

Next, we give the functionality for general MPC with IRIA, and argue that it can be realized by using the SPDZ online phase to consume the preprocessing generated by $[\![\mathcal{F}_{\mathsf{SPDZPrep\text{-}IA}}(n, \mathbb{F}, \lambda_{\mathsf{s}})]\!]_{\mathsf{PV}}$, and opening everything in the case of an abort.

**Functionality 6.4.** $\mathcal{F}_{\mathsf{MPC\text{-}IRIA}}(n, \mathbb{F})$**. Reactive MPC with IRIA**

This functionality interacts with $n$ actively participating parties denoted by $\mathcal{P}_1 \dots \mathcal{P}_n$ and with the ideal adversary $\mathcal{S}$. It is also parameterized by the description of a field $\mathbb{F}$ over which it operates. Whenever this functionality receives a message from any party, it notifies the other parties that the message was received, including in the notification the sender's ID, the session ID, and any subsession IDs.

**Initialization:** On receiving $(\texttt{init}, \mathsf{sid}, \mathbf{u}, v)$ from each $\mathcal{P}_i$ for $i \in [n]$ such that $\mathcal{P}_1 \| \dots \| \mathcal{P}_n \| \mathsf{sid}' = \mathsf{sid}$ for some fresh $\mathsf{sid}'$, $\mathbf{u} \in \mathbb{N}^n$, and $v \in \mathbb{N}$, send $(\texttt{init-req}, \mathsf{sid}, \mathbf{u}, v)$ to $\mathcal{S}$. If $\mathcal{S}$ responds with $(\texttt{abort}, \mathsf{sid}, \mathcal{P}_j)$ and $\mathcal{P}_j$ is corrupt, then send $(\texttt{abort}, \mathsf{sid}, \mathcal{P}_j)$ to all parties and ignore all future instructions with the same session ID. If $\mathcal{S}$ respond with $(\texttt{proceed}, \mathsf{sid})$ or $\mathcal{P}_j$ is honest, then send $(\texttt{initialized}, \mathsf{sid}, \mathbf{u}, v)$ to all parties and store $(\texttt{initialized}, \mathsf{sid}, \mathbf{u}, v)$ in memory.

**Input:** On receiving $(\texttt{input}, \mathsf{sid}, \mathsf{ssid}, \mathcal{P}_i, x)$ for $x \in \mathbb{F}$ from $\mathcal{P}_i$ for some $i \in [n]$ and $(\texttt{input}, \mathsf{sid}, \mathsf{ssid}, \mathcal{P}_i)$ from $\mathcal{P}_j$ for every $j \in [n] \setminus \{i\}$, if $\mathsf{ssid}$ is fresh for this $\mathsf{sid}$ and the record $(\texttt{initialized}, \mathsf{sid}, \mathbf{u}, v)$ exists in memory, and fewer than $\mathbf{u}_i$ records of the form $(\texttt{wire}, \mathsf{sid}, *, i, *)$ exist in memory, then store $(\texttt{wire}, \mathsf{sid}, \mathsf{ssid}, i, x)$ in memory.

**Addition:** On receiving $(\mathsf{add}, \mathsf{sid}, \mathsf{ssid}_1, \mathsf{ssid}_2, \mathsf{ssid}_3)$ from $\mathcal{P}_i$ for every $i \in [n]$, if $\mathsf{ssid}_3$ is fresh for this $\mathsf{sid}$ and records of the form $(\texttt{wire}, \mathsf{sid}, \mathsf{ssid}_1, *, x)$ and $(\texttt{wire}, \mathsf{sid}, \mathsf{ssid}_2, *, y)$ are stored in memory, then store $(\texttt{wire}, \mathsf{sid}, \mathsf{ssid}_3, \mathsf{add}, x + y)$ in memory.

**Multiplication:** On receiving $(\texttt{mult}, \mathsf{sid}, \mathsf{ssid}_1, \mathsf{ssid}_2, \mathsf{ssid}_3)$ from $\mathcal{P}_i$ for every $i \in [n]$, if $\mathsf{ssid}_3$ is fresh for this $\mathsf{sid}$ and records of the form $(\texttt{wire}, \mathsf{sid}, \mathsf{ssid}_1, *, x)$ and $(\texttt{wire}, \mathsf{sid}, \mathsf{ssid}_2, *, y)$ and $(\texttt{initialized}, \mathsf{sid}, \mathbf{u}, v)$ exist in memory, and fewer than $v$ records of the form $(\texttt{wire}, \mathsf{sid}, *, \texttt{mul}, *)$ exist in memory, then store $(\texttt{wire}, \mathsf{sid}, \mathsf{ssid}_3, \texttt{mul}, x \cdot y)$ in memory.

**Output:** On receiving $(\texttt{output}, \mathsf{sid}, \mathsf{ssid})$ from $\mathcal{P}_i$ for every $i \in [n]$, if at least one actively participating party is honest and a record of the form $(\texttt{wire}, \mathsf{sid}, \mathsf{ssid}, *, x)$ exists in memory, then send $(\texttt{candidate-output}, \mathsf{sid}, \mathsf{ssid}, x)$ to $\mathcal{S}$, and receive either $(\texttt{proceed}, \mathsf{sid}, \mathsf{ssid})$ or $(\texttt{abort}, \mathsf{sid}, \mathcal{P}_j)$ in response. If $\texttt{abort}$ is received and $\mathcal{P}_j$ is corrupt, then send all records of the form $(\texttt{wire}, \mathsf{sid}, *, *, *)$ to $\mathcal{S}$ and send $(\texttt{abort}, \mathsf{sid}, \mathcal{P}_j)$ to all parties; if $\texttt{proceed}$ is received or $\mathcal{P}_j$ is not corrupt, then send $(\texttt{output}, \mathsf{sid}, \mathsf{ssid}, x)$ to all parties.

**Corollary 6.5.** *For any finite field $\mathbb{F}$ and any $n \in \mathbb{N}^+$, there exists a protocol that statistically UC-realizes $[\![\mathcal{F}_{\mathsf{MPC\text{-}IRIA}}(n, \mathbb{F})]\!]_{\mathsf{PV}}$ against a malicious adversary statically corrupting up to $n-1$ active participants in the $[\![\mathcal{F}_{\mathsf{SCOT\text{-}SIRIA}}]\!]_{\mathsf{PV}}$-hybrid model.*

*Proof Sketch.* To realize $[\![\mathcal{F}_{\mathsf{MPC\text{-}IRIA}}(n, \mathbb{F})]\!]_{\mathsf{PV}}$, we use the SPDZ [DPSZ12] online protocol (adjusted such that the MAC is repeated $\lceil \lambda_{\mathsf{s}} / \log_2 |\mathbb{F}| \rceil$ times, where $\lambda_{\mathsf{s}}$ is the statistical security parameter), and generate the required correlated randomness using $[\![\mathcal{F}_{\mathsf{SPDZPrep\text{-}IA}}(n, \mathbb{F}, \lambda_{\mathsf{s}})]\!]_{\mathsf{PV}}$. Simulation is straightforward. Any time the simulator must send a value on behalf of a functionality or honest party that is not fixed by the existing protocol transcript, it samples one uniformly; this fixes the outputs of the corrupt parties in a way that the simulator can predict. If an abort occurs, the simulator learns the honest parties' inputs and outputs, and there are enough degrees of freedom that it can compute a set of MACs that are consistent, which it outputs to the adversary on behalf of $[\![\mathcal{F}_{\mathsf{SPDZPrep\text{-}IA}}(n, \mathbb{F}, \lambda_{\mathsf{s}})]\!]_{\mathsf{PV}}$.

By sequentially applying theorems 6.3 and 5.4, we can UC-realize this functionality in the $([\![\mathcal{F}_{\mathsf{SCOT\text{-}SIRIA}}]\!]_{\mathsf{PV}}, [\![\mathcal{F}_{\mathsf{Com}}]\!]_{\mathsf{PV}}, [\![\mathcal{F}_{\mathsf{CT}}]\!]_{\mathsf{PV}})$-hybrid model.

In the $[\![\mathcal{F}_{\mathsf{Com}}]\!]_{\mathsf{PV}}$-hybrid model, we can realize $[\![\mathcal{F}_{\mathsf{CT}}]\!]_{\mathsf{PV}}$ with identifiable abort in the following simple way: the parties locally sample XOR shares of the coins to be tossed, and then commit and release them. $[\![\mathcal{F}_{\mathsf{Com}}]\!]_{\mathsf{PV}}$ is trivially realizable in the $[\![\mathcal{F}_{\mathsf{SCOT\text{-}SIRIA}}]\!]_{\mathsf{PV}}$-hybrid model: the committer simply sends XOR shares of the value to be committed to the OT functionality (playing the role of Alice). To open the commitment, the committer instructs the functionality to open the OT messages; otherwise, the receiver learns only one share. Using these two realizations yields the corollary. □

MPC with input-revealing identifiable abort cannot be used in any context where the honest parties have sensitive inputs. In the context of inputless sampling functionalities, however, it has no downside: the inputs and outputs of a sampling functionality are *discarded* (and thus of no value to the adversary) in the case of an abort. In other words, for inputless sampling functionalities,

IA and IRIA are *equivalent*. This fact will help us to realize MPC with IA for *all* functions: we can use MPC with IRIA to sample the correlated randomness required by an information-theoretic protocol for MPC with IA. We demonstrate two pathways. The first involves a specific functionality for generic gate-by-gate computation of any circuit.

**Functionality 6.6.** $\mathcal{F}_{\mathsf{MPC\text{-}IA}}(n, \mathbb{F})$**. Reactive MPC with IA**

This functionality is the same as $\mathcal{F}_{\mathsf{MPC\text{-}IRIA}}(n, \mathbb{F})$, except that $\mathcal{S}$ does not receive any records of the form $(\mathtt{wire}, \mathsf{sid}, *, *, *)$ in the case of an abort.

**Corollary 6.7.** *For any finite field* $\mathbb{F}$ *and any* $n \in \mathbb{N}^+$*, there exists a protocol that statistically UC-realizes* $[\![\mathcal{F}_{\mathsf{MPC\text{-}IA}}(n, \mathbb{F})]\!]_{\mathsf{PV}}$ *against a malicious adversary statically corrupting up to* $n-1$ *active participants in the* $[\![\mathcal{F}_{\mathsf{SCOT\text{-}SIRIA}}]\!]_{\mathsf{PV}}$*-hybrid model.*

*Proof Sketch.* We can construct a protocol satisfying this corollary by using $[\![\mathcal{F}_{\mathsf{MPC\text{-}IRIA}}]\!]_{\mathsf{PV}}$ to compute the correlated randomness for the information-theoretic online phase of the protocol of Baum et al. [BOS16]. If the resulting protocol aborts while the correlated randomness is being generated, then the correlated randomness is published, but never used. If there is no abort during the generation of the correlated randomness, then it is never revealed, and the online phase identifies cheaters while hiding the honest inputs and outputs. □

Our second pathway involves a protocol compiler, which transforms any protocol with information-theoretic security in the semi-honest setting into a protocol that achieves identifiable abort.

**Corollary 6.8.** *Let* $\pi_{\mathsf{SH}}$ *be a* $\mathcal{F}_{\mathsf{Corr}}^{\mathcal{D}}$*-hybrid protocol (for an efficiently computable distribution* $\mathcal{D}$*) that information-theoretically UC-realizes a functionality* $\mathcal{F}$ *in the presence of a semi-honest adversary statically corrupting* $n-1$ *participants. There exists a compiler to turn* $\pi_{\mathsf{SH}}$ *into an* $[\![\mathcal{F}_{\mathsf{SCOT\text{-}SIRIA}}]\!]_{\mathsf{PV}}$*-hybrid protocol that information-theoretically UC-realizes* $\mathcal{F}$ *with publicly verifiable identifiable abort in the presence of a malicious adversary statically corrupting* $n-1$ *participants. Moreover, this compiler is asymptotically round-preserving.*

*Proof Sketch.* We can construct a compiler satisfying this corollary by using $[\![\mathcal{F}_{\mathsf{MPC\text{-}IRIA}}]\!]_{\mathsf{PV}}$ to compute the correlated randomness for the information-theoretic online phase of the compiler of Ishai et al. [IOZ14]. If this compiler is applied to an appropriate input protocol, and the compiled protocol aborts while the correlated randomness is being generated, then the correlated randomness is published, but never used. If there is no abort during the generation of the correlated randomness, then it is never revealed, and the online phase identifies cheaters while hiding the honest inputs and outputs. □

Finally, we give an efficiency result for biased sampling functionalities with guaranteed output. Our efficiency result is phrased in terms of the number of

instances of the $[\![\mathcal{F}_{\text{SCOT-SIRIA}}]\!]_{\text{PV}}$ functionality that are required. This functionality is the only functionality whose realization requires public key operations: specifically, each instance requires a constant number of public key operations when our protocol from section 3 is used. Thus we effectively upper-bound the number of public-key operations required by the same asymptotic figure.

**Corollary 6.9.** *For any $n \in \mathbb{N}^+$ and any PPT sampling function, there exists a protocol that performs rejection-biased distributed sampling using that sampling function, with publicly verifiable and universally composable guaranteed output delivery against a malicious adversary statically corrupting up to $n-1$ active participants, in the $[\![\mathcal{F}_{\text{SCOT-SIRIA}}]\!]_{\text{PV}}$-hybrid random oracle model. This protocol requires each party to invoke $[\![\mathcal{F}_{\text{SCOT-SIRIA}}]\!]_{\text{PV}}$ at most $O(n \cdot \lambda_c)$ times.*

*Proof Sketch.* We begin with by combining $[\![\mathcal{F}_{\text{MPC-IRIA}}]\!]_{\text{PV}}$ with the GMW player-elimination technique [GMW87]. The parties invoke $[\![\mathcal{F}_{\text{MPC-IRIA}}(n)]\!]_{\text{PV}}$ on secret, uniform inputs of appropriate dimension, and then feed it a circuit description of the sampling function. If an abort occurs, the party identified as having caused it is eliminated, and the remaining parties try again.

If we realize $[\![\mathcal{F}_{\text{MPC-IRIA}}]\!]_{\text{PV}}$ via corollary 6.5, the resulting protocol is information theoretically secure in the $[\![\mathcal{F}_{\text{SCOT-SIRIA}}]\!]_{\text{PV}}$-hybrid. Now the many independent instances of $[\![\mathcal{F}_{\text{SCOT-SIRIA}}]\!]_{\text{PV}}$ that are invoked by each pair of parties can be replaced by a single pairwise invocation of $[\![\mathcal{F}_{\text{SCOTE-SIRIA}}]\!]_{\text{PV}}$ that is initialized once and reused to generate additional batches of OTs when required. Because one of the two invoking paries is eliminated any time this functionality aborts, no more than $n$ instances will ever be initialized per party.

Via corollary 4.4, each instance of $[\![\mathcal{F}_{\text{SCOTE-SIRIA}}]\!]_{\text{PV}}$ can be realized via $O(\lambda_c)$ instances of $[\![\mathcal{F}_{\text{SCOT-SIRIA}}]\!]_{\text{PV}}$ in the random oracle model; this implies $O(n \cdot \lambda_c)$ instances of $[\![\mathcal{F}_{\text{SCOT-SIRIA}}]\!]_{\text{PV}}$ per party in total. □

## Acknowledgements

The authors of this work were supported variously by the NSF under grants 1646671, 1816028, and 2055568, by the ERC under projects NTSC (742754), SPEC (803096), and HSS (852952), by ISF grant 2774/2, by AFOSR award FA9550-21-1-0046, by the Algorand Centres of Excellence programme managed by the Algorand Foundation, and by the Carlsberg Foundation under the Semper Ardens Research Project CF18-112 (BCM). Any opinions, findings, and conclusions or recommendations expressed in this material are those of the authors and do not necessarily reflect the views of any funding entity.

## References

[ACC+21] Bar Alon, Hao Chung, Kai-Min Chung, Mi-Ying Huang, Yi Lee, and Yu-Ching Shen. Round efficient secure multiparty quantum computation with identifiable abort. In *Advances in Cryptology – CRYPTO 2021, part I*, pages 436–466, 2021.

[AL10] Yonatan Aumann and Yehuda Lindell. Security against covert adversaries: Efficient protocols for realistic adversaries. *Journal of Cryptology*, 23(2):281–343, 2010.

[AO16] Bar Alon and Eran Omri. Almost-optimally fair multiparty cointossing with nearly three-quarters malicious. In *Proceedings of the 14th Theory of Cryptography Conference, TCC 2016-B, part I*, pages 307–335, 2016.

[BCG+15] Eli Ben-Sasson, Alessandro Chiesa, Matthew Green, Eran Tromer, and Madars Virza. Secure sampling of public parameters for succinct zero knowledge proofs. In *Proceedings of the 36th IEEE Symposium on Security and Privacy, (S&P)*, pages 287–304, 2015.

[BCG+19] Elette Boyle, Geoffroy Couteau, Niv Gilboa, Yuval Ishai, Lisa Kohl, and Peter Scholl. Efficient pseudorandom correlation generators: Silent OT extension and more. In *Advances in Cryptology – CRYPTO 2019, part III*, pages 489–518, 2019.

[BCG+20] Elette Boyle, Geoffroy Couteau, Niv Gilboa, Yuval Ishai, Lisa Kohl, and Peter Scholl. Correlated pseudorandom functions from variable-density LPN. In *Proceedings of the 61st Annual Symposium on Foundations of Computer Science (FOCS)*, pages 1069–1080, 2020.

[BDF+23] Jakob Burkhardt, Ivan Damgård, Tore Kasper Frederiksen, Satrajit Ghosh, and Claudio Orlandi. Improved distributed RSA key generation using the miller-rabin test. *IACR Cryptol. ePrint Arch.*, page 644, 2023.

[BDOZ11] Rikke Bendlin, Ivan Damgård, Claudio Orlandi, and Sarah Zakarias. Semi-homomorphic encryption and multiparty computation. In *Advances in Cryptology – EUROCRYPT 2011*, pages 169–188, 2011.

[Bea95] Donald Beaver. Precomputing oblivious transfer. In *Advances in Cryptology – CRYPTO 1995*, pages 97–109, 1995.

[BF01] Dan Boneh and Matthew K. Franklin. Efficient generation of shared RSA keys. *Journal of the ACM*, 48(4):702–722, 2001.

[BGIN22] Elette Boyle, Niv Gilboa, Yuval Ishai, and Ariel Nof. Secure multiparty computation with sublinear preprocessing. In *Advances in Cryptology – EUROCRYPT 2022, part I*, pages 427–457, 2022.

[BGM17] Sean Bowe, Ariel Gabizon, and Ian Miers. Scalable multi-party computation for zk-snark parameters in the random beacon model. *IACR Cryptol. ePrint Arch.*, 2017:1050, 2017.

[BHLT17] Niv Buchbinder, Iftach Haitner, Nissan Levi, and Eliad Tsfadia. Fair coin flipping: Tighter analysis and the many-party case. In *Proceedings of the 28th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 2580–2600, 2017.

[BLOO11] Amos Beimel, Yehuda Lindell, Eran Omri, and Ilan Orlov. $1/p$-secure multiparty computation without honest majority and the best of both worlds. In *Advances in Cryptology – CRYPTO 2011*, pages 277–296, 2011.

[BMMM20] Nicholas-Philip Brandt, Sven Maier, Tobias Müller, and Jörn Müller-Quade. Constructing secure multi-party computation with identifiable abort. http://eprint.iacr.org/2020/153, 2020.

[BMR90] Donald Beaver, Silvio Micali, and Phillip Rogaway. The round complexity of secure protocols (extended abstract). In *Proceedings of the 22nd Annual ACM Symposium on Theory of Computing (STOC)*, pages 503–513, 1990.

[BN06] Mihir Bellare and Gregory Neven. Multi-signatures in the plain public-key model and a general forking lemma. In *Proceedings of the 13th ACM Conference on Computer and Communications Security, (CCS)*, 2006.

[BOO15] Amos Beimel, Eran Omri, and Ilan Orlov. Protocols for multiparty coin toss with a dishonest majority. *Journal of Cryptology*, 28(3):551–600, 2015.

[BOS16] Carsten Baum, Emmanuela Orsini, and Peter Scholl. Efficient secure multiparty computation with identifiable abort. In *Proceedings of the 14th Theory of Cryptography Conference, TCC 2016-B, part I*, pages 461–490, 2016.

[BOSS20] Carsten Baum, Emmanuela Orsini, Peter Scholl, and Eduardo Soria-Vazquez. Efficient constant-round MPC with identifiable abort and public verifiability. In *Advances in Cryptology – CRYPTO 2020, part II*, pages 562–592, 2020.

[Can01] Ran Canetti. Universally composable security: A new paradigm for cryptographic protocols. In *Proceedings of the 42nd Annual Symposium on Foundations of Computer Science (FOCS)*, pages 136–145, 2001.

[CCD+20] Megan Chen, Ran Cohen, Jack Doerner, Yashvanth Kondi, Eysa Lee, Schuyler Rosefield, and abhi shelat. Multiparty generation of an RSA modulus. In *Advances in Cryptology – CRYPTO 2020, part III*, pages 64–93, 2020.

[CCL+23] Guilhem Castagnos, Dario Catalano, Fabien Laguillaumie, Federico Savasta, and Ida Tucker. Bandwidth-efficient threshold EC-DSA revisited: Online/offline extensions, identifiable aborts proactive and adaptive security. *Theor. Comput. Sci.*, 939:78–104, 2023.

[CDKs22] Ran Cohen, Jack Doerner, Yashvanth Kondi, and abhi shelat. Guaranteed output in $O(\sqrt{n})$ rounds for round-robin sampling protocols. In *Advances in Cryptology – EUROCRYPT 2022, part I*, pages 241–271, 2022.

[CF01] Ran Canetti and Marc Fischlin. Universally composable commitments. In *Advances in Cryptology – CRYPTO 2001*, pages 19–40, 2001.

[CFY17] Robert K. Cunningham, Benjamin Fuller, and Sophia Yakoubov. Catching MPC cheaters: Identification and openability. In *Proceedings of the 10th International Conference on Information Theoretic Security (ICITS)*, pages 110–134, 2017.

[CGG+20] Ran Canetti, Rosario Gennaro, Steven Goldfeder, Nikolaos Makriyannis, and Udi Peled. UC non-interactive, proactive, threshold ECDSA with identifiable aborts. In *Proceedings of the 27th ACM Conference on Computer and Communications Security, (CCS)*, pages 1769–1787. ACM, 2020.

[CGZ20] Ran Cohen, Juan A. Garay, and Vassilis Zikas. Broadcast-optimal two-round MPC. In *Advances in Cryptology – EUROCRYPT 2020, part II*, pages 828–858, 2020.

[CHI+21] Megan Chen, Carmit Hazay, Yuval Ishai, Yuriy Kashnikov, Daniele Micciancio, Tarik Riviere, abhi shelat, Muthu Venkitasubramaniam, and Ruihan Wang. Diogenes: Lightweight scalable RSA modulus generation with a dishonest majority. In *Proceedings of the 42nd IEEE Symposium on Security and Privacy, (S&P)*, pages 590–607, 2021.

[CHOR22] Ran Cohen, Iftach Haitner, Eran Omri, and Lior Rotem. From fairness to full security in multiparty computation. *Journal of Cryptology*, 35(1):4, 2022.

[CL17] Ran Cohen and Yehuda Lindell. Fairness versus guaranteed output delivery in secure multiparty computation. *Journal of Cryptology*, 30(4):1157–1186, 2017.

[Cle86] Richard Cleve. Limits on the security of coin flips when half the processors are faulty (extended abstract). In *Proceedings of the 18th Annual ACM Symposium on Theory of Computing (STOC)*, pages 364–369, 1986.

[CLOS02] Ran Canetti, Yehuda Lindell, Rafail Ostrovsky, and Amit Sahai. Universally composable two-party and multi-party secure computation. In *Proceedings of the 34th Annual ACM Symposium on Theory of Computing (STOC)*, pages 494–503, 2002.

[CRSW22] Michele Ciampi, Divya Ravi, Luisa Siniscalchi, and Hendrik Waldner. Round-optimal multi-party computation with identifiable abort. In *Advances in Cryptology – EUROCRYPT 2022, part I*, pages 335–364, 2022.

[CvdGT95] Claude Crépeau, Jeroen van de Graaf, and Alain Tapp. Committed oblivious transfer and private multi-party computation. In *Advances in Cryptology – CRYPTO 1995*, pages 110–123, 1995.

[Dac20] Dana Dachman-Soled. Revisiting fairness in MPC: polynomial number of parties and general adversarial structures. In *Proceedings of the 18th Theory of Cryptography Conference, TCC 2020, part II*, pages 595–620, 2020.

[DKLs18] Jack Doerner, Yashvanth Kondi, Eysa Lee, and abhi shelat. Secure two-party threshold ECDSA from ECDSA assumptions. In *Proceedings of the 39th IEEE Symposium on Security and Privacy, (S&P)*, pages 980–997, 2018.

[DKLs19] Jack Doerner, Yashvanth Kondi, Eysa Lee, and abhi shelat. Threshold ECDSA from ECDSA assumptions: The multiparty case. In *Proceedings of the 40th IEEE Symposium on Security and Privacy, (S&P)*, 2019.

[DKLs23] Jack Doerner, Yashvanth Kondi, Eysa Lee, and abhi shelat. Threshold ecdsa in three rounds. Cryptology ePrint Archive, Paper 2023/765, 2023. https://eprint.iacr.org/2023/765.

[DMR+21] Ivan Damgård, Bernardo Magri, Divya Ravi, Luisa Siniscalchi, and Sophia Yakoubov. Broadcast-optimal two round MPC with an honest majority. In *Advances in Cryptology – CRYPTO 2021, part II*, pages 155–184, 2021.

[dMRT21] Cyprien Delpech de Saint Guilhem, Eleftheria Makri, Dragos Rotaru, and Titouan Tanguy. The return of eratosthenes: Secure generation of RSA moduli using distributed sieving. In *Proceedings of the 28th ACM Conference on Computer and Communications Security, (CCS)*, pages 594–609, 2021.

[DPSZ12] Ivan Damgård, Valerio Pastro, Nigel P. Smart, and Sarah Zakarias. Multiparty computation from somewhat homomorphic encryption. In *Advances in Cryptology – CRYPTO 2012*, pages 643–662, 2012.

[DRSY22] Ivan Damgård, Divya Ravi, Luisa Siniscalchi, and Sophia Yakoubov. Minimizing setup in broadcast-optimal two round MPC. http://eprint.iacr.org/2022/293, 2022.

[DSW08] Yevgeniy Dodis, Victor Shoup, and Shabsi Walfish. Efficient constructions of composable commitments and zero-knowledge proofs. In *Advances in Cryptology – CRYPTO 2008*, pages 515–535, 2008.

[EGL85] Shimon Even, Oded Goldreich, and Abraham Lempel. A randomized protocol for signing contracts. *Communications of the ACM*, 28(6):637–647, 1985.

[Fis05] Marc Fischlin. Communication-efficient non-interactive proofs of knowledge with online extractors. In *Advances in Cryptology – CRYPTO 2005*, pages 152–168, 2005.

[FLNW17] Jun Furukawa, Yehuda Lindell, Ariel Nof, and Or Weinstein. High-throughput secure three-party computation for malicious adversaries and an honest majority. In *Advances in Cryptology – EUROCRYPT 2017, part II*, pages 225–255, 2017.

[FS86] Amos Fiat and Adi Shamir. How to prove yourself: Practical solutions to identification and signature problems. In *Advances in Cryptology – CRYPTO 1986*, pages 186–194, 1986.

[Gar04] Juan A. Garay. Efficient and universally composable committed oblivious transfer and applications. In *Proceedings of the First Theory of Cryptography Conference, TCC 2004*, pages 297–316, 2004.

[Gil99] Niv Gilboa. Two party RSA key generation. In *Advances in Cryptology – CRYPTO 1999*, pages 116–129, 1999.

[GK09] S. Dov Gordon and Jonathan Katz. Complete fairness in multiparty computation without an honest majority. In *Proceedings of the 6th Theory of Cryptography Conference, TCC 2009*, pages 19–35, 2009.

[GK12] S. Dov Gordon and Jonathan Katz. Partial fairness in secure two-party computation. *Journal of Cryptology*, 25(1):14–40, 2012.

[GKM+18] Jens Groth, Markulf Kohlweiss, Mary Maller, Sarah Meiklejohn, and Ian Miers. Updatable and universal common reference strings with applications to zk-snarks. In *Advances in Cryptology – CRYPTO 2018, part III*, pages 698–728, 2018.

[GKPS18] Chaya Ganesh, Yashvanth Kondi, Arpita Patra, and Pratik Sarkar. Efficient adaptively secure zero-knowledge from garbled circuits. In Michel Abdalla and Ricardo Dahab, editors, *Public-Key Cryptography - PKC 2018 - 21st IACR International Conference on Practice and Theory of Public-Key Cryptography, Rio de Janeiro, Brazil,*

*March 25-29, 2018, Proceedings, Part II*, volume 10770 of *Lecture Notes in Computer Science*, pages 499–529. Springer, 2018.

[GMW87] Oded Goldreich, Silvio Micali, and Avi Wigderson. How to play any mental game or A completeness theorem for protocols with honest majority. In *Proceedings of the 19th Annual ACM Symposium on Theory of Computing (STOC)*, pages 218–229, 1987.

[HMRT12] Carmit Hazay, Gert Læssøe Mikkelsen, Tal Rabin, and Tomas Toft. Efficient RSA key generation and threshold Paillier in the two-party setting. In *Proceedings of the Cryptographers' Track at the RSA Conference (CT-RSA)*, pages 313–331, 2012.

[HMZ08] Martin Hirt, Ueli Maurer, and Vassilis Zikas. MPC vs. SFE : Unconditional and computational security. In *Advances in Cryptology – ASIACRYPT 2008*, pages 1–18, 2008.

[HSS17] Carmit Hazay, Peter Scholl, and Eduardo Soria-Vazquez. Low cost constant round MPC combining BMR and oblivious transfer. In *Advances in Cryptology – ASIACRYPT 2017, part I*, pages 598–628, 2017.

[HT17] Iftach Haitner and Eliad Tsfadia. An almost-optimally fair three-party coin-flipping protocol. *SIAM Journal on Computing*, 46(2):479–542, 2017.

[IKK+11] Yuval Ishai, Jonathan Katz, Eyal Kushilevitz, Yehuda Lindell, and Erez Petrank. On achieving the "best of both worlds" in secure multiparty computation. *SIAM Journal on Computing*, 40(1):122–141, 2011.

[IKOS07] Yuval Ishai, Eyal Kushilevitz, Rafail Ostrovsky, and Amit Sahai. Zero-knowledge from secure multiparty computation. In *Proceedings of the 39th Annual ACM Symposium on Theory of Computing (STOC)*, pages 21–30, 2007.

[IOS12] Yuval Ishai, Rafail Ostrovsky, and Hakan Seyalioglu. Identifying cheaters without an honest majority. In *Proceedings of the 9th Theory of Cryptography Conference, TCC 2012*, pages 21–38, 2012.

[IOZ14] Yuval Ishai, Rafail Ostrovsky, and Vassilis Zikas. Secure multiparty computation with identifiable abort. In *Advances in Cryptology – CRYPTO 2014, part II*, pages 369–386, 2014.

[IPS08] Yuval Ishai, Manoj Prabhakaran, and Amit Sahai. Founding cryptography on oblivious transfer - efficiently. In *Advances in Cryptology – CRYPTO 2008*, pages 572–591, 2008.

[JKO13] Marek Jawurek, Florian Kerschbaum, and Claudio Orlandi. Zero-knowledge using garbled circuits: how to prove non-algebraic statements efficiently. In *Proceedings of the 20th ACM Conference on Computer and Communications Security, (CCS)*, pages 955–966. ACM, 2013.

[Kil88] Joe Kilian. Founding cryptography on oblivious transfer. In *Proceedings of the 20th Annual ACM Symposium on Theory of Computing (STOC)*, pages 20–31, 1988.

[KMSV21] Markulf Kohlweiss, Mary Maller, Janno Siim, and Mikhail Volkhov. Snarky ceremonies. In *Advances in Cryptology – ASIACRYPT 2021, part III*, pages 98–127, 2021.

[KMTZ13] Jonathan Katz, Ueli Maurer, Björn Tackmann, and Vassilis Zikas. Universally composable synchronous computation. In *Proceedings of the 10th Theory of Cryptography Conference, TCC 2013*, pages 477–498, 2013.

[KOS15] Marcel Keller, Emmanuela Orsini, and Peter Scholl. Actively secure OT extension with optimal overhead. In *Advances in Cryptology – CRYPTO 2015, part I*, pages 724–741, 2015.

[KOS16] Marcel Keller, Emmanuela Orsini, and Peter Scholl. MASCOT: faster malicious arithmetic secure computation with oblivious transfer. In *Proceedings of the 23th ACM Conference on Computer and Communications Security, (CCS)*, pages 830–842, 2016.

[Ks22] Yashvanth Kondi and abhi shelat. Improved straight-line extraction in the random oracle model with applications to signature aggregation. In *Advances in Cryptology – ASIACRYPT 2022, part II*, pages 279–309, 2022.

[Pas03] Rafael Pass. On deniability in the common reference string and random oracle model. In *Advances in Cryptology – CRYPTO 2003*, pages 316–337, 2003.

[Ped91] Torben P. Pedersen. Non-interactive and information-theoretic secure verifiable secret sharing. In *Advances in Cryptology – CRYPTO 1991*, pages 129–140, 1991.

[PVW08] Chris Peikert, Vinod Vaikuntanathan, and Brent Waters. A framework for efficient and composable oblivious transfer. In *Advances in Cryptology – CRYPTO 2008*, pages 554–571, 2008.

[Qua20] Willy Quach. Uc-secure OT from lwe, revisited. In *Security and Cryptography for Networks - 12th International Conference, SCN 2020, Amalfi, Italy, September 14-16, 2020, Proceedings*, pages 192–211, 2020.

[Roy22] Lawrence Roy. Softspokenot: Quieter OT extension from small-field silent VOLE in the minicrypt model. In *Advances in Cryptology – CRYPTO 2022, part I*, pages 657–687, 2022.

[Sch89] Claus-Peter Schnorr. Efficient identification and signatures for smart cards. In *Advances in Cryptology – CRYPTO 1989*, pages 239–252, 1989.

[SF16] Gabriele Spini and Serge Fehr. Cheater detection in SPDZ multiparty computation. In *Proceedings of the 9th International Conference on Information Theoretic Security (ICITS)*, pages 151–176, 2016.

[SSY22] Mark Simkin, Luisa Siniscalchi, and Sophia Yakoubov. On sufficient oracles for secure computation with identifiable abort. In *Proceedings of the 13th Conference on Security and Cryptography for Networks (SCN)*, pages 494–515, 2022.

[WRK17] Xiao Wang, Samuel Ranellucci, and John Katz. Global-scale secure multiparty computation. In *Proceedings of the 24th ACM Conference on Computer and Communications Security, (CCS)*, pages 39–56, 2017.

[Yao82] Andrew Chi-Chih Yao. Protocols for secure computations (extended abstract). In *Proceedings of the 23rd Annual Symposium on Foundations of Computer Science (FOCS)*, pages 160–164, 1982.

[YWZ20] Kang Yang, Xiao Wang, and Jiang Zhang. More efficient MPC from improved triple generation and authenticated garbling. In *Proceedings of the 27th ACM Conference on Computer and Communications Security, (CCS)*, pages 1627–1646, 2020.

[ZHM09] Vassilis Zikas, Sarah Hauser, and Ueli Maurer. Realistic failures in secure multi-party computation. In *Proceedings of the 6th Theory of Cryptography Conference, TCC 2009*, pages 274–293, 2009.