Secure Multi-Party Computation with Identifiable Abort

Yuval Ishai*
Computer Science Department
Technion
yuvali@cs.technion.ac.il

Rafail Ostrovsky†
Computer Science Department
UCLA
rafail@cs.ucla.edu

Vassilis Zikas‡
Computer Science Department
ETH Zurich
vzikas@inf.ethz.ch

Abstract

Protocols for secure multi-party computation (MPC) that resist a dishonest majority are susceptible to "denial of service" attacks, allowing even a single malicious party to force the protocol to abort. In this work, we initiate a systematic study of the more robust notion of *security with identifiable abort*, which leverages the effect of an abort by forcing, upon abort, at least one malicious party to reveal its identity.

We present the first *information-theoretic* MPC protocol which is secure with identifiable abort (in short ID-MPC) using a correlated randomness setup. This complements a negative result of Ishai et al. (TCC 2012) which rules out information-theoretic ID-MPC in the OT-hybrid model, thereby showing that *pairwise* correlated randomness is insufficient for information-theoretic ID-MPC.

In the standard model (i.e., without a correlated randomness setup), we present the first computationally secure ID-MPC protocol making *black-box* use of a standard cryptographic primitive, namely an (adaptively secure) oblivious transfer (OT) protocol. This provides a more efficient alternative to existing ID-MPC protocols, such as the GMW protocol, that make a non-black-box use of the underlying primitives.

As a theoretically interesting side note, our black-box ID-MPC provides an example for a natural cryptographic task that can be realized using a *black-box access* to an OT protocol but cannot be realized unconditionally using an ideal OT oracle.

Keywords: Multi-Party Computation, Feasibility, Efficiency

*Supported by the European Union's Tenth Framework Programme (FP10/2010-2016) under grant agreement no. 259426 ERC-CaC, ISF grant 1361/10, and BSF grant 2012378.

†Work supported in part by NSF grants 09165174, 1065276, 1118126 and 1136174, US-Israel BSF grant 2008411, OKAWA Foundation Research Award, IBM Faculty Research Award, Xerox Faculty Research Award, B. John Garrick Foundation Award, Teradata Research Award, and Lockheed-Martin Corporation Research Award. This material is based upon work supported by the Defense Advanced Research Projects Agency through the U.S. Office of Naval Research under Contract N00014 -11 -1-0392. The views expressed are those of the author and do not reflect the official policy or position of the Department of Defense or the U.S. Government.

‡Portions of this work were done while the author was at UCLA. Work supported in part by the Swiss National Science Foundation (SNF) Ambizione grant PZ00P2_142549.

1 Introduction

Recent advances in secure multiparty computation have led to protocols that compute large circuits in a matter of seconds. Most of these protocols, however, are restricted to provide security against semi-honest adversaries, or alternatively assume an honest majority. A notable exception is the SPDZ line of work [3, 16, 14, 15, 34] which tolerates a majority of malicious parties. SPDZ is optimized for the pre-processing model and demonstrates a remarkably fast on-line phase, largely due to the fact that it uses information-theoretic techniques and, thus, avoids costly cryptographic operations. Unfortunately, all these efficient MPC protocols for the case of a dishonest majority are susceptible to the following *denial-of-service (DoS)* attack: even a single malicious party can force an abort without any consequences (i.e., without even being accused of cheating). Although classical impossibility results for MPC prove that abort-free computation is impossible against dishonest majorities, vulnerability to DoS attacks is an issue that should be accounted for in any practical application.

Summary of known results. The seminal works on MPC [47, 21, 2, 9, 42] establish tight feasibility bounds on the tolerable number of corruptions for perfect, statistical (aka information-theoretic or unconditional), and computational (aka cryptographic) security. For semi-honest adversaries, unconditionally secure protocols exist if there is an honest majority, or if the parties have access to a complete functionality oracle or other types of setup. An arguably minimal setup is giving the parties (appropriately) correlated random strings before the inputs are known. We refer to this as the *correlated randomness model*.

When there is no honest majority and the adversary is malicious, full security that includes fairness cannot be achieved [12]. Instead, one usually settles for the relaxed notion of *security with abort*: Either the protocol succeeds, in which case every party receives its output, or the protocol aborts, in which case all honest parties learn that the protocol aborted. (Because of the lack of fairness, the adversary can learn its outputs even when the protocol aborts.) The GMW protocol [21, 19] realizes this notion of security under standard cryptographic assumptions. Interestingly, this protocol also satisfies the following useful *identifiability* property: upon abort every party learns the identity of some corrupted party. This property is in the focus of our work.

To the best of our knowledge, all protocols achieving this notion of security (e.g., [21, 7]) are based on the same paradigm of using public zero-knowledge proofs to detect deviation from the protocol. While elegant and conceptually simple, this approach leads to inefficient protocols that make a non-black-box use of the underlying cryptographic primitives.[1] The situation is even worse in the information-theoretic setting, where an impossibility result from [31] (see also [44, Section 3.7]) proves that information-theoretic MPC with identifiable abort is impossible even in the OT-hybrid model, i.e., where parties can make ideal calls to an oblivious transfer (OT) functionality [41].

Our Contributions. We initiate a systematic study of this more robust and desirable notion of *secure MPC with identifiable abort* (ID-MPC). An ID-MPC protocol leverages the effect of an abort by forcing, upon abort, at least one malicious party to reveal its identity. This feature discourages cheaters from aborting, and in many applications allows for full recovery by excluding

[1] Alternatively, protocols such as the CDN protocol [13] make a use of ad-hoc zero-knowledge proofs based on specific number theoretic intractability assumptions. The disadvantage of these protocols is that they require public-key operations for each gate of the circuit being evaluated, and cannot get around this by using optimization techniques such as efficient OT extension [28].

the identified cheater and restarting the protocol. We provide formal security definitions both in the setting of Universal Composition (UC) [5] and in the stand-alone setting [21, 19, 4]. Furthermore, we study feasibility and efficiency of ID-MPC in both the information-theoretic and the computational security models.

For the information-theoretic model, we present a general compiler that transforms any MPC protocol which uses correlated randomness to achieve security against semi-honest adversaries into a similar protocol which is secure with identifiable abort against malicious adversaries. As a corollary, we get the first information-theoretic ID-MPC protocol in the correlated randomness model. This protocol complements an impossibility result from [31], which rules out information-theoretic ID-MPC in the OT-hybrid model. Indeed, the insufficiency of OT implies that *pairwise* correlated randomness is not sufficient for information-theoretic ID-MPC, but leaves open the question of whether or not n-wise correlations are, which is answered affirmatively here.

In the computational security model, we present an ID-MPC protocol for realizing sampling functionalities, namely ones that sample and distribute correlated random strings, which only makes a *black-box* use of an (adaptively secure) *OT protocol* and ideal calls to a commitment functionality.[2] Using this protocol for realizing the setup required by the information-theoretic protocol yields the first ID-MPC protocol which makes a black-box use of standard cryptographic primitives. This holds both in the UC framework [5], assuming standard UC-setups, and in the plain stand-alone model [21, 19, 4]. Combined with the above-mentioned impossibility result from [31], this provides an interesting example for a natural cryptographic task that can be realized using a *black-box* access to an OT protocol but cannot be unconditionally realized using an ideal OT oracle.

Our results demonstrate that ID-MPC is not only the most desirable notion from a practical point of view, but it also has the potential to be efficiently implemented. To this end, one can instantiate our construction with efficient OT protocols from the literature [39, 11, 36, 17].[3] Furthermore, pre-computing the randomness in an off-line phase yields a protocol in the pre-processing model which, similarly to SPDZ-style protocols, has an information-theoretic online phase. Investigating how our methodology can be fine-tuned towards practice remains an interesting direction for future work. Finally, our protocols can be used to improve the efficiency of a number of protocols in the fairness-related literature, e.g., [29, 18, 26, 37, 48, 22, 1], as these works implicitly use ID-MPC (typically instantiated by GMW) to realize a sampling functionality.

Comparison to Existing Work. Our information-theoretic protocol can be seen as a new *feasibility* result, since the current literature contains no (efficient or inefficient) information-theoretic ID-MPC protocol from correlated randomness. Similarly, our computational protocol can also be seen as a "second-order" feasibility result, since this is the first ID-MPC protocol making *black-box* use of a standard cryptographic primitive. Notwithstanding, much of the motivation for considering black-box constructions in cryptography is derived from the goal of practical efficiency, and indeed the most practical protocols today (whether Yao-based or GMW-based) are black-box protocols that do not need to know the "code" of the underlying cryptographic primitives.

[2] The ideal commitments can be replaced by a black-box use of a commitment protocol, or alternatively realized by making a black-box use of OT [27, 38]. The OT protocol can be secure against either semi-honest or malicious adversaries, as these two flavors are equivalent under black-box reductions [24, 10].

[3] Our analysis requires the underlying OT to be adaptively secure. Proving the same statement for a static OT protocol is a theoretically interesting open problem. From a practical point of view, however, many instances of adaptively secure OT can be efficiently implemented from few such instances in the (programmable) random oracle model [28, 36].

2 The Model

We prove our security statements in the universal composition (UC) framework of Canetti [5]. In a nutshell, a protocol π (securely) UC realizes a functionality $\mathcal{F}$ if for any adversary $\mathcal{A}$ attacking π there exists an ideal adversary, the simulator $\mathcal{S}$, that makes an ideal evaluation of $\mathcal{F}$ indistinguishable from a protocol execution with $\mathcal{A}$ in the eyes any environment $\mathcal{Z}$. When $\mathcal{Z}$, $\mathcal{A}$, and $\mathcal{S}$ are polynomially bounded we say that the protocol realizes $\mathcal{F}$ (with computational security); otherwise, when $\mathcal{Z}$, $\mathcal{A}$, and $\mathcal{S}$ are unbounded, we say that the protocol *unconditionally* realizes $\mathcal{F}$ (with information-theoretic security). For self containment we have included the basics of the UC model in Appendix A.1.

For simplicity we restrict our description to computation of non-reactive functionalities, also known as *secure function evaluation (SFE).* (The general case can be reduced to this case by using a suitable form of secret sharing [31] for maintaining the secret state of the reactive functionality.) Moreover, we describe our protocols as *synchronous protocols*, i.e., round-based protocols where messages sent in some round are delivered by the beginning of the next round; such protocols can be executed in UC as demonstrated in [33, 35]. The advantage of such a "synchronous" description is dual: first, it yields simpler descriptions of functionalities and protocols; indeed, because the parties are aware of the round in which each message should be sent/received, we can avoid always explicitly writing all the message/protocol IDs in the descriptions. Second, it is compatible with the protocol description in the stand-alone model of computation [20, 4], which allows us to directly translate our results into that model.

Our protocols assume n parties from the set $\mathcal{P} = \{p_1, \ldots, p_n\}$. We prove our results for a non-adaptive adversary who actively corrupts parties *at the beginning* of the protocol execution, but our results can be extended to the adaptive case.[4] Our results are with respect to an (often implicit) security parameter k, where we use the standard definition of negligible and overwhelming from [19].

Correlated Randomness as a Sampling Functionality. Our protocols are in the *correlated randomness* model, i.e., they assume that the parties initially, before receiving their inputs, receive appropriately correlated random strings. In particular, the parties jointly hold a vector $\vec{R} = (R_1, \ldots, R_n) \in (\{0,1\}^*)^n$, where p_i holds R_i, drawn from a given efficiently samplable distribution $\mathcal{D}$. This is, as usual, captured by giving the parties initial access to an ideal functionality $\mathcal{F}_{\mathsf{Corr}}^{\mathcal{D}}$, known as a *sampling functionality*, which, upon receiving a default input from any party, samples $\vec{R}$ from $\mathcal{D}$ and distributes it to the parties. Hence, a protocol in the correlated randomness model is formally an $\mathcal{F}_{\mathsf{Corr}}^{\mathcal{D}}$-hybrid protocol. Formally, a sampling functionality $\mathcal{F}_{\mathsf{Corr}}^{\mathcal{D}}$ is parameterized by an efficiently computable sampling distribution $\mathcal{D}$ and the (ID's of the parties in) the player set $\mathcal{P}$.

$\mathcal{F}_{\mathsf{Corr}}^{\mathcal{D}}(\mathcal{P})$

Upon receiving message ($\mathsf{CorrRand}$) from any party or the adversary, set $\vec{R} = (R_1, \ldots, R_n) \leftarrow \mathcal{D}$ and for each $p_i \in \mathcal{P}$ send R_i to p_i (or to the adversary if p_i is corrupted).

[4] In fact, some of our protocols use optimizations tailored to proving adaptive security.

Information-Theoretic Signatures Our protocols use information-theoretic (i.t.) signatures [45, 43, 46] to commit a party to messages it sends. Roughly speaking, these are information-theoretic analogues to standard digital signatures, i.e., they allow some party p_i, the *signer*, to send a message m to a party p_j, the *receiver*, along with a string σ that we refer to as the *signature*, such that the receiver can at a later point publicly open σ and prove to every party that the message m was indeed sent from p_i. Note that in order to achieve i.t. security the verification key cannot be publicly known. Rather, in i.t. signatures, the signer has a signing key $\mathtt{sk}$ and every party $p_i \in \mathcal{P}$ holds a different private verification key $\mathtt{vk}_i$ corresponding to $\mathtt{sk}$.

In our protocols different (independent) signing keys are used for each signature. In this case, i.t. signatures provide the following guarantees with overwhelming probability (against an unbounded adversary): *(completeness)* A signature with the correct singing key will be accepted by any honest verifier in $\mathcal{P}$; *(unforgeability)* the adversary cannot come up with a signature that will be accepted by some (honest) verifier without knowing the signing key; *(consistency)* an adversarial signer cannot come up with a signature that will be accepted by some honest verifier and rejected by another.

For self-containment, we recall the formal security definition and construction of i.t. signatures in Appendix A.1.

3 Security with Identifiable Abort

We put forward the notion of *secure multi-party computation with identifiable abort*, also referred to as *Identifiable MPC (ID-MPC)* which allows the computation to fail (abort), but ensures that when this happens every party is informed about it, and they also agree on the index i of some corrupted party $p_i \in \mathcal{P}$ (we say then that *the parties abort with* p_i). More concretely, for an arbitrary functionality $\mathcal{F}$, we define $[\mathcal{F}]_\perp^{\mathtt{ID}}$ to be the corresponding functionality with identifiable abort, which behaves as $\mathcal{F}$ with the following modification: upon receiving from the simulator a special command (abort, p_i), where $p_i \in \mathcal{P}$ is a corrupted party (if p_i is not corrupted then $[\mathcal{F}]_\perp^{\mathtt{ID}}$ ignores the message), $[\mathcal{F}]_\perp^{\mathtt{ID}}$ sets the output of all (honest) parties to (abort, p_i).

Definition 1. Let $\mathcal{F}$ be a functionality and $[\mathcal{F}]_\perp^{\mathtt{ID}}$ be the corresponding functionality with identifiable abort. We say that *a protocol* π *securely realizes* $\mathcal{F}$ *with identifiable abort* if π securely realizes the functionality $[\mathcal{F}]_\perp^{\mathtt{ID}}$.

The UC composition theorem extends in a straightforward manner to security with identifiable abort. To formally state such a theorem we first specify the class of protocols for which it is natural to replace hybrid functionalities by protocols (subroutines) that realize them with identifiable abort. Informally, these protocols have the property that as soon as one of their hybrids aborts with the identity of some (corrupted) party p_i, the calling protocol also aborts with p_i. Formally, let $\mathcal{G}$ be a functionality and π be a $\mathcal{G}$-hybrid protocol. We say that π is *abort respecting* if upon receiving (abort, p_i) from $\mathcal{G}$ for some $i \in [n]$, every honest party in π outputs (abort, p_i) and halts.

Theorem 2. *Let $\mathcal{F}$ and $\mathcal{G}$ be ideal functionalities and let π be an $\mathcal{G}$-hybrid abort respecting protocol which securely realizes $\mathcal{F}$ with identifiable abort.*[5] *Let also ρ be protocol which securely realizes $\mathcal{G}$ with identifiable abort, and denote by π^ρ the protocol derived from π by replacing ideal calls to $\mathcal{G}$ by invocations of protocols ρ. Then π^ρ securely realizes $\mathcal{F}$ with identifiable abort.*

[5] As in [5], $\mathcal{G}$ might be a collection of ideal functionalities.

The proof follows along the lines of the UC composition theorem [5].

4 Unconditional ID-MPC from Correlated Randomness

In this section we describe our unconditionally secure identifiable MPC protocol in the correlated randomness model. In fact, our result is more general, as we provide a compiler that transforms any given unconditionally secure protocol in the semi-honest correlated randomness model into an unconditionally secure ID-MPC protocol in the (malicious) correlated randomness model. Although the correlated randomness provided by the setup in the malicious protocol is different than the semi-honest, the latter can be obtained from the former by an efficient transformation. Informally, our statement can be phrased as follows:

> Let π_{sh} be an $\mathcal{F}_{\mathsf{Corr}}^{\mathcal{D}}$-hybrid protocol (for an efficiently computable distribution $\mathcal{D}$), which *unconditionally* UC realizes a functionality $\mathcal{F}$ in the presence of a *semi-honest* adversary. Then there exists a compiler turning π_{sh} into an $\mathcal{F}_{\mathsf{Corr}}^{\mathcal{D}'}$-hybrid protocol (for an appropriate efficiently computable distribution $\mathcal{D}'$), which *unconditionally* UC realizes $\mathcal{F}$ *with identifiable abort* (in the malicious model).

Overview of the Compiler. We start by providing a high-level overview of our compiler. As is typical, the semi-honest protocol π_{sh} which we compile works over standard point-to-point (insecure) channels. Furthermore, without loss of generality (see Section 4.3) we assume that π_{sh} is *deterministic.*

Let $\vec{R}^{\mathsf{sh}} = (R_1^{\mathsf{sh}}, \ldots, R_n^{\mathsf{sh}})$ denote the setup used by the semi-honest protocol π_{sh} (i.e., each p_i holds string R_i^{sh}). The setup for the compiled protocol distributes $\vec{R}^{\mathsf{sh}}$ to the parties, and commits every party to its received string. Subsequently, the parties proceed by, first, committing to their inputs and, then, executing their π_{sh}-instructions in a publicly verifiable manner: whenever, p_i would send a message m in π_{sh}, in the compiled protocol p_i broadcasts m and publicly proves, in zero-knowledge, that the broadcasted message is consistent with his committed input and setup string R_i^{sh}. For the above approach to work for unbounded adversaries and allow for identifiability, we need the commitment scheme and the associated zero-knowledge proofs to be unconditionally secure and failures to be publicly detectable. We construct such primitives relying on appropriately correlated randomness in Sections 4.1 and 4.2, respectively.

4.1 Commitments with Identifiable Abort

In this section we provide a protocol which unconditionally UC realizes the standard (one-to-many) multi-party commitment functionality $\mathcal{F}_{\text{COM}}$ with identifiable abort. $\mathcal{F}_{\text{COM}}$ allows party $p_i \in \mathcal{P}$, the *committer*, to commit to a message m and later on publicly open m while guaranteeing the following properties: (hiding) no party in $\mathcal{P} \setminus \{p_i\}$ receives any information on m during the commit phase; (binding) at the end of the commit phase a message m' is fixed (where $m' = m$ if the committer is honest), such that only m' might be accepted in the reveal phase (and m' is always accepted when the committer is honest). The one-to-many commitment functionality is described in the following:

$\mathcal{F}_{\text{COM}}(\mathcal{P})$

Commit Phase: Upon receiving message $(\text{msg\_id}, \mathsf{commit}, i, m)$ from party $p_i \in \mathcal{P}$ (or the adversary if p_i is corrupted) where $m \in \{0,1\}^*$ and msg_id is a valid message ID, record the tuple $(\text{msg\_id}, p_i, m)$ and send the message $(\text{msg\_id}, \mathsf{receipt}, p_i)$ to every party in $\mathcal{P}$ (and to the adversary). Every future commit message with the same ID msg_id is ignored.

Reveal Phase: Upon receiving a message $(\text{msg\_id}, \mathsf{reveal})$ from party $p_i \in \mathcal{P}$, if a message $(\text{msg\_id}, \mathsf{commit}, i, m)$ was previously recorded, then send the message $(\text{msg\_id}, \mathsf{reveal}, m)$ to all parties in $\mathcal{P}$ (and to the adversary); otherwise ignore the message.

Our protocol Π_{COM} which i.t. securely realizes $\mathcal{F}_{\text{COM}}$ with identifiable abort assumes the following correlated-randomness setup: for p_i to commit to a value $m \in \{0,1\}^*$, p_i needs to hold a uniformly random string $r \in \{0,1\}^{|m|}$ along with an information-theoretic signature σ on r, where every party in $\mathcal{P}$ holds his corresponding verification key (but no party, not even p_i, gets to learn the signing key). Formally, the sampling functionality $\mathcal{F}^{\text{COM}}_{\mathsf{Corr}}$ used for our commitment scheme is as follows:

$\mathcal{F}^{\text{COM}}_{\mathsf{Corr}}(\mathcal{P})$

Upon receiving message $(\mathsf{CorrRand}, p_i, \ell)$, where $\ell = \text{poly}(k)$, from party p_i (or the adversary if p_i is corrupted), do the following

1. Choose $r \in_R \{0,1\}^\ell$ uniformly at random.
2. Set $(\mathtt{sk}, \vec{\mathtt{vk}}) := \mathsf{Gen}(1^\ell, n, 1)$ and compute $\sigma := \mathsf{Sign}(r, \mathtt{sk})$.
3. Send $R_i = (r, \sigma, \mathtt{vk}_i)$ to p_i and for each $p_j \in [n] \setminus \{i\}$ send $R_j = \mathtt{vk}_j$ to p_j.

Given the above setup, p_i can commit to m by broadcasting $y = m \oplus r$. To, later on, open the commitment y, p_i broadcasts r along with the signature σ, where every party verifies the signature and outputs $m = y \oplus r$ if it is valid, otherwise aborts with p_i (i.e., outputs (abort, p_i)).

Protocol Π_{COM} $(\mathcal{P})$

Setup: The protocol works in the $\mathcal{F}^{\text{COM}}_{\mathsf{Corr}}(\mathcal{P})$-hybrid world, i.e., in order for $p_i \in \mathcal{P}$ to commit to an ℓ-bit string, he sends $(\mathsf{CorrRand}, p_i, \ell)$ to $\mathcal{F}^{\text{COM}}_{\mathsf{Corr}}(\mathcal{P})$; every party $p_j \in \mathcal{P}$ denotes the message received from $\mathcal{F}^{\text{COM}}_{\mathsf{Corr}}(\mathcal{P})$ by R_j, where $R_i = (r, \sigma, \mathtt{vk}_i)$ and for each $p_j \in [n] \setminus \{i\}$, $R_j = \mathtt{vk}_j$ to p_j.[a]

Commit Phase Upon receiving input $(\text{msg\_id}, \mathsf{commit}, i, m)$ from $\mathcal{Z}$, p_i computes $y = m \oplus r$ and broadcasts y. If p_i broadcasts an invalid message then every party aborts with p_i; otherwise, every party $p_j \in \mathcal{P}$ adopts y as the commitment (and outputs $(\text{msg\_id}, \mathsf{receipt}, p_i)$).

Reveal Phase To open the commitment y on m, p_i broadcasts $(\text{msg\_id}, \mathsf{reveal}, m, y, \sigma)$, where σ denotes the signature on r which p_i received from the setup. Every party $p_j \in \mathcal{P}$ verifies (using the verification key $\mathtt{vk}_j \in R_j$) that $\mathsf{Ver}(m \oplus y, \sigma, \mathtt{vk}_j) = 1$; if this is not the case then p_j aborts with p_i, otherwise p_j outputs $(\text{msg\_id}, \mathsf{reveal}, m)$.

[a] Recall that messages sent to/received from the setup functionality have unique message IDs; hence, even when used to commit to multiple messages, the parties can tell which setup-string corresponds to which commitment.

The hiding property of Π_{COM} follows from the fact that r is uniformly random. Moreover, the unforgeability of the signature scheme ensures that the commitment is binding and publicly verifiable.

Finally, the completeness of the scheme ensures that the protocol aborts only when the committer p_i is corrupted. Additionally, same as all UC commitments, the above scheme is *extractable*, i.e., the simulator of a corrupted committer can learn, already in the commit phase, which message will be opened so that he can input it to the functionality, and *equivocal*, i.e., the simulator of a corrupted receiver can open a commitment to any message of his choice.[6] Taking a glimpse at the proof, both properties follow from the fact that the simulator controls the setup. Indeed, knowing r allows the simulator to extract m from the broadcasted message, whereas knowing the signing key $\mathtt{sk}$ allows him to generate a valid signature/opening to any message.

Theorem 3. *The protocol Π_{COM} unconditionally UC realizes the functionality $\mathcal{F}_{\text{COM}}$ with identifiable abort.*

Proof. We need to show that Π_{COM} securely realizes the functionality $[\mathcal{F}_{\text{COM}}]_{\perp}^{\mathtt{ID}}$. We consider two cases: (1) The committer p_i is corrupted, and (2) the committer p_i is honest. In both cases the simulator uses the adversary (in a black-box straight-line manner) and gets to emulate towards him the setup $\mathcal{F}_{\mathsf{Corr}}^{\text{COM}}$. In particular, $\mathcal{S}$ starts off by computing $(\overline{R_1}, \ldots, \overline{R_n})$ as $\mathcal{F}_{\mathsf{Corr}}^{\text{COM}}$ would, i.e., $\overline{R_i} = (\overline{r}, \overline{\sigma}, \overline{\mathtt{vk}}_i)$ and for $j \in [n] \setminus \{i\}$: $\overline{R_j} = \overline{\mathtt{vk}_j}$. $\mathcal{S}$ hands $\mathcal{A}$ the values $\overline{R_j}$ corresponding to corrupted parties p_j. Subsequently,

In **Case 1** (i.e., if p_i is corrupted), $\mathcal{S}$ waits to receive form $\mathcal{A}$ the broadcasted message y, extracts $\overline{m} := y - \overline{r}$ and hands $(\text{msg\_id}, \mathsf{commit}, i, \overline{m})$ to the functionality $[\mathcal{F}_{\text{COM}}]_{\perp}^{\mathtt{ID}}$. To emulate the opening of y, the simulator waits to receive from $\mathcal{A}$ the opening message $(\text{msg\_id}, \mathsf{reveal}, m, y', \sigma)$. If $y' \neq y$ or σ does not verify with all the (simulated) keys $\mathtt{vk}_j$ of honest parties p_j, then the simulator sends (abort, p_i) to $[\mathcal{F}_{\text{COM}}]_{\perp}^{\mathtt{ID}}$; otherwise the simulator sends $(\text{msg\_id}, \mathsf{reveal})$ to $[\mathcal{F}_{\text{COM}}]_{\perp}^{\mathtt{ID}}$. It is straightforward to verify that unless the adversary forges a signature (which, by the unforgeability property of the signature scheme, happens with negligible probability) the simulated transcript (and the honest parties' output) is distributed identically to the real transcript. Indeed, $\mathcal{S}$ chooses the setup from the same distribution as $\mathcal{F}_{\mathsf{Corr}}^{\text{COM}}$ hence the message y is distributed identically as $y = m + r$ in both cases. Furthermore, in the opening phase only m might be opened, since if the adversary attempts to open a fake value he will be caught with overwhelming probability unless he succeeds in forging a corresponding signature.

In **Case 2** (i.e., p_i is honest). In the commit-phase, i.e., as soon as $\mathcal{S}$ receives $(\text{msg\_id}, \mathsf{receipt}, p_i)$ from $[\mathcal{F}_{\text{COM}}]_{\perp}^{\mathtt{ID}}$, $\mathcal{S}$ emulates towards $\mathcal{A}$ a broadcast of a uniformly random string $\overline{y} \in \{0,1\}^{\ell}$. In the opening phase, $\mathcal{S}$ receives $(\text{msg\_id}, \mathsf{reveal}, m)$ from $[\mathcal{F}_{\text{COM}}]_{\perp}^{\mathtt{ID}}$, computes $\overline{r}' := \overline{y} \oplus m$ along with a signature $\overline{\sigma}' := \mathsf{Sign}(\overline{r}', \overline{\mathtt{sk}})$ and emulates towards the adversary p_i broadcasting the message $(\text{msg\_id}, \mathsf{reveal}, m, \overline{y}, \overline{\sigma}')$. Clearly, as $\overline{y}$ is chosen uniformly at random and $\overline{\sigma}'$ is generated given the actual (simulated) signature key the simulated view is distributed identically to the view of a protocol execution. □

4.2 Setup-Commit-Then-Proof

Next we present a protocol which allows the parties receiving random strings (drawn from some joint distribution $\mathcal{D}$) to publicly prove, in zero-knowledge, that they use these strings in a protocol. Our protocol implements the *Setup-Commit-then-Prove* functionality $\mathcal{F}_{\text{SCP}}$ which can be viewed as a modification of the Commit-then-Prove functionality from [7] restricting the committed witnesses

[6] In [31] a primitive called *unanimously identifiable commitments* (UIC) was introduced for this purpose, but the definition of UIC does not guarantee all the properties we need for UC secure commitments.

to be distributed by the setup instead of being chosen by the provers. More concretely $\mathcal{F}_{\text{SCP}}$ (see below) works in two phases: in a first phase, it provides a string/witness R_i to each $p_i \in \mathcal{P}$, where $\vec{R} = (R_1, \ldots, R_n)$ is drawn from $\mathcal{D}$; in a second phase, $\mathcal{F}_{\text{SCP}}$ allows every party p_i to prove q-many NP statements of the type $\mathcal{R}(x, R_i) = 1$ for the same publicly known NP relation $\mathcal{R}_i$ and the witness R_i received from the setup, but for potentially different (public) strings x. A detailed description of $\mathcal{F}_{\text{SCP}}$ follows.

$\mathcal{F}_{\text{SCP}}(\mathcal{P}, \mathcal{D}, \vec{\mathcal{R}} = (\mathcal{R}_1, \ldots, \mathcal{R}_n), q)$

The functionality is parametrized by $\mathcal{P}$, the distribution $\mathcal{D}$, a vector $\vec{\mathcal{R}}$ of NP relations, and a bound $q = \text{poly}(k)$ on the number of proofs allowed per party.

Setup-Commit Phase: Upon receiving message $(\mathsf{reqWitness})$ from any party $p_i \in \mathcal{P}$ (or the adversary if p_i is corrupted) sample $(R_1, \ldots, R_n) \leftarrow \mathcal{D}$ and for each $i \in [n]$ send message $(\mathsf{witness}, R_i)$ to p_i (or to the adversary if p_i is corrupted).

Prove Phase: Upon receiving a message $(\mathsf{ZK\text{-}prover}, x)$ where $x \in \{0,1\}^{\text{poly}(k)}$ from any party $p_i \in \mathcal{P}$, if $\mathcal{R}_i(x, R_i) = 1$, and p_i did not already send q-many $(\mathsf{ZK\text{-}prover}, \cdot)$-messages, then send $(\mathsf{verified}, x, p_i)$ to all parties in $\mathcal{P}$ and to the adversary; otherwise send them $(\mathsf{not\text{-}verified}, p_i)$.

In the remainder of this section we describe a protocol which unconditionally securely realizes the setup-commit-then-proof functionality $\mathcal{F}_{\text{SCP}}$ in the correlated randomness model. To this direction, we first show how to realize the sigle-use version of $\mathcal{F}_{\text{SCP}}$, denoted as $\mathcal{F}_{1\text{SCP}}$, and then use the UC composition with joint state theorem (JUC) [8] to derive a protocol for $\mathcal{F}_{\text{SCP}}$. The functionality $\mathcal{F}_{1\text{SCP}}$ works exactly as $\mathcal{F}_{\text{SCP}}$ with the restriction that it allows a specific prover $p \in \mathcal{P}$ to do *a single* (instead of q-many) proofs for a witness w of a given NP relation $\mathcal{R}$.

$\mathcal{F}_{1\text{SCP}}(\mathcal{P}, \mathcal{D}, \mathcal{R}, p)$

The functionality is parametrized by $\mathcal{P}$, the distribution $\mathcal{D}$, an NP relation $\mathcal{R}$, and (the id of) the prover $p \in \mathcal{P}$.

Setup-Commit Phase: Upon receiving message $(\mathsf{reqWitness})$ from party $p \in \mathcal{P}$ (or the simulator if p is corrupted), sample $w \leftarrow \mathcal{D}$, record (w, p) and send message $(\mathsf{witness}, w)$ to p (or the adversary if p is corrupted).

Prove Phase: Upon receiving a message $(\mathsf{ZK\text{-}prover}, x)$ where $x \in \{0,1\}^{\text{poly}(k)}$ from prover $p \in \mathcal{P}$ if $\mathcal{R}(x, w) = 1$ and p_i did not already send a $(\mathsf{ZK\text{-}prover}, \cdot)$-message then send $(\mathsf{verified}, x, p)$ to all parties in $\mathcal{P}$ and to the adversary; otherwise send them $(\mathsf{not\text{-}verified}, p)$.

Our protocol for realizing the functionality $\mathcal{F}_{1\text{SCP}}$ with identifiable abort uses the idea of "MPC in the head" [25, 30, 32]. In particular, let $\mathcal{F}_{\mathsf{D}}$ denote the $(n+1)$-party (reactive) functionality among the players in $\mathcal{P}$ and a special player p_D, the *dealer*, which works as follows: In a first phase, $\mathcal{F}_{\mathsf{D}}$ receives a message $w \in \{0,1\}^{\text{poly}(k)}$ from p_D and forwards w to $p \in \mathcal{P}$. In a second phase, p sends x to $\mathcal{F}_{\mathsf{D}}$, which computes $b := \mathcal{R}(x, w)$ and outputs (b, x) to every $p_j \in \mathcal{P} \setminus \{p_D\}$. Clearly, any protocol in the plain model which unconditionally realizes $\mathcal{F}_{\mathsf{D}}$ with an honest dealer p_D, where p_D does not participate in the second phase, can be turned into a protocol which securely realizes $\mathcal{F}_{1\text{SCP}}(\mathcal{P}, \mathcal{D}, \mathcal{R}, p)$ in the correlated randomness model. Indeed, one needs to simply have the corresponding sampling functionality play the role of p_D (where w is drawn from $\mathcal{D}$). In the following we show how to design such a protocol using the idea of player-simulation [25].

Let $\Pi_{(n+1,m),t}$ be a protocol which perfectly securely (and robustly) realizes $\mathcal{F}_{\mathsf{D}}$ in the client-server model [25, 30, 32], among the clients $\mathcal{P} \cup \{p_D\}$ and an additional m servers. Such a protocol exists assuming $t < m/3$ servers are corrupted [2]. For simplicity, assume that $\Pi_{(n+1,m),t}$ has the following properties, which are consistent to how protocols from the literature, e.g., [2], would realize functionality $\mathcal{F}_{\mathsf{D}}$ in the client-server setting: (i) for computing the first phase of $\mathcal{F}_{\mathsf{D}}$, $\Pi_{(n+1,m),t}$ has p_D share his input w among the m servers with a secret sharing scheme that is perfectly *t-private* (the shares of any t servers leak no information on w) and perfectly *t-robust* (the sharing can be reconstructed even when up to t cheaters modify their shares), and, also p_D hands *all* the shares to p (ii) p_D does not participate in the second phase of $\Pi_{(n+1,m),t}$ (this is wlog as p_D is a client with no input or output in this second phase), and (iii) the output $(\mathcal{R}(x,w),x)$ is publicly announced (i.e., is in the view of every server at the end of the protocol).

Assuming p_D is honest, a protocol Π_{n+1} for unconditionally realizing $\mathcal{F}_{\mathsf{D}}$ with identifiable abort (among only the players in $\mathcal{P}\cup\{p_D\}$) can be built based on the above protocol $\Pi_{(n+1,m),t}$ as follows: for the first phase, p_D generates shares of a t-robust and t-private sharing of w as he would do in $\Pi_{(n+1,m),t}$ and sends them to p. In addition to sending the shares, p_D commits p to each share by sending him an i.t. signature on it and distributing the corresponding verification keys to the players in $\mathcal{P}$. For the second phase, p emulates in his head the second phase of the execution of $\Pi_{(n+1,m),t}$ among m virtual servers $\hat{p}_1,\ldots,\hat{p}_m$ where each server has private input his share, as received from p_D in the first phase, and a public input x (the same for all clients); p publicly commits to the view of each server. Finally, the parties in $\mathcal{P}\setminus\{p\}$ challenge p to open a random subset $\mathcal{J}\subseteq[m]$ of size t of the committed views and announce the corresponding input-signatures which p received from p_D. If the opened views are inconsistent with an accepting execution of $\Pi_{(n+1,m),t}$ on input x and the committed shares—i.e., some output is 0, or some opening fails, or some signature does not verify for the corresponding (opened) private input, or for some pair of views the incoming messages do not match the outgoing messages—then the parties abort with p.

The security of the protocol Π_{n+1} is argued similarly to [30, Theorem 4.1]: on the one hand, when p is honest then we can use the simulator for $\Pi_{(n+1,m),t}$ to simulate the views of the parties in $\mathcal{J}$. The perfect t-security of $\Pi_{(n+1,m),t}$ and the t-privacy of the sharing ensures that this simulation is indistinguishable from the real execution. On the other hand, when p is corrupted, then we only need to worry about correctness. Roughly, correctness is argued as follows: if there are at most $t < m/3$ incorrect views, then the t-robustness of $\Pi_{(n+1,m),t}$ and of the sharing ensures that the output in any of the other views will be correct; by a standard counting argument we can show that the probability that some of these views is opened is overwhelming when $m = O(k)$. Otherwise, (i.e., if there are more than t-incorrect views) then with high probability a pair of such views will be opened and the inconsistency will be exposed.

To derive, from Π_{n+1}, a protocol for $\mathcal{F}_{1\text{SCP}}(\mathcal{P},\mathcal{D},\mathcal{R},p)$ in the correlated randomness model, we have the sampling functionality, $\mathcal{F}_{\mathsf{Corr}}^{1\text{SCP}}$ play the role of the dealer p_D. In addition to the committed shares, $\mathcal{F}_{\mathsf{Corr}}^{1\text{SCP}}$ generates the necessary setup enabling the prover $p\in\mathcal{P}$ to commit to the m (virtual) servers' views in the second phase of the protocol Π_{n+1}. Furthermore, to simplify the description, we also have $\mathcal{F}_{\mathsf{Corr}}^{1\text{SCP}}$ create a “coin-tossing setup” which players in $\mathcal{P}$ can use to sample the random subset $\mathcal{J}\in[m]$ of views to be opened: $\mathcal{F}_{\mathsf{Corr}}^{1\text{SCP}}$ hands to each $p_j\in\mathcal{P}$ a random string c_j and commits p_j to it; the coin sequence c for choosing $\mathcal{J}$ is then computed by every p_j opening c_j and taking $c=\oplus_{j=1}^{n}c_j$. The corresponding sampling functionality, denoted as $\mathcal{F}_{\mathsf{Corr}}^{1\text{SCP}}$, is described in the following. For sake of modularity we describe the functionality $\mathcal{F}_{\mathsf{Corr}}^{1\text{SCP}}$ in two pieces: First, we describe a sampling functionality $\mathcal{F}_{\mathsf{Corr}}^{\text{ZK}}$ which, for a witness w (given as a parameter), generates the necessary setup for the proof (i.e., the second) phase. The functionality $\mathcal{F}_{\mathsf{Corr}}^{1\text{SCP}}$, then, simply

$\mathcal{F}^{\text{ZK}}_{\mathsf{Corr}}(\mathcal{P}, w, m, t, \mathcal{R})$

The functionality is parameterized by $\mathcal{P}$, a string $w \in \{0,1\}^{\text{poly}(k)}$, an NP relation $\mathcal{R}$, and the numbers $m = O(k)$ and t with $t < m/3$ as in protocol $\Pi_{(n+1,m),t}$.

Upon receiving message $(\mathsf{CorrRand}, p_i)$ from any $p_i \in \mathcal{P}$ (or the adversary if p is corrupted) do the following:

- *Commit p_i to a sharing of w:* Compute a perfectly t-robust and t-private sharing $\langle w \rangle = (\langle w \rangle^1, \ldots, \langle w \rangle^m)$ of w, and for each $\ell \in [m]$ do the following: Set $(\mathtt{sk}^\ell, \vec{\mathtt{vk}}^\ell) := \mathsf{Gen}(1^{|\langle w \rangle^\ell|}, n, 1)$ and compute $\sigma(\langle w \rangle^\ell) = \mathsf{Sign}(\langle w \rangle^\ell, \mathtt{sk}^\ell)$; send $(\langle w \rangle^\ell, \sigma(\langle w \rangle^\ell), \mathtt{vk}^\ell_i)$ to p_i, and for each $j \in [n] \setminus \{i\}$ send $\mathtt{vk}^\ell_j$ to p_j
- *"Coin-tossing setup"* For every party $p_j \in \mathcal{P}$:
  1. Chose $c_j \in_R \{0,1\}^{t \log(m)}$.
  2. Set $(\mathtt{sk}, \vec{\mathtt{vk}}) := \mathsf{Gen}(1^{|c_j|}, n, 1)$ and compute $\sigma(c_j) = \mathsf{Sign}(c_j, \mathtt{sk})$; send $(c_j, \sigma(c_j), \mathtt{vk}_j)$ to p_j, and for each $\ell \in [n] \setminus \{j\}$ send $\mathtt{vk}_\ell$ to p_ℓ
- *Setup for committing to a $\Pi_{(n+1,m),t}$ execution:* For every $\ell \in [m]$ emulate a call to $\mathcal{F}^{\text{COM}}_{\mathsf{Corr}}(\mathcal{P})$ with input $(\mathsf{CorrRand}, p, \mathbb{V}_\ell)$, where $\mathbb{V}_\ell$ is the size of the ℓth server's view in an execution of $\Pi_{(n+1,m),t}$ for computing $\mathcal{F}_{\mathsf{D}}$.

samples the witness w from $\mathcal{D}$ and (internally) calls $\mathcal{F}^{\text{ZK}}_{\mathsf{Corr}}$ with parameter w.

$\mathcal{F}^{\text{1SCP}}_{\mathsf{Corr}}(\mathcal{P}, \mathcal{D}, m, t, \mathcal{R})$

The functionality is parameterized by $\mathcal{P}$, an efficiently sampleable distribution $\mathcal{D}$, an NP relation $\mathcal{R}$, and the numbers $m = O(k)$ and t with $t < m/3$ as in protocol $\Pi_{(n+1,m),t}$.
Upon receiving message $(\mathsf{CorrRand}, p)$ from party $p \in \mathcal{P}$ (or the adversary if p is corrupted) do the following:

1. Sample w from distribution $\mathcal{D}$.
2. Emulate an invocation of $\mathcal{F}^{\text{ZK}}_{\mathsf{Corr}}(\mathcal{P}, w, m, t, \mathcal{R})$ on input $(\mathsf{CorrRand}, p)$ and distribute all the generated outputs.

In the following we give a detailed description of the protocol Π_{1SCP} for implementing $\mathcal{F}_{\text{1SCP}}$, where we denote by $\langle w \rangle = (\langle w \rangle^1, \ldots, \langle w \rangle^m)$ a perfectly t-private and t-robust secret sharing of a given value w among players in some $\hat{\mathcal{P}} = (\hat{p}_1, \ldots, \hat{p}_m)$ (e.g., the sharing from [2] which is based on bivariate polynomials), where $\langle w \rangle^i$ denotes the ith share of $\langle w \rangle$, i.e., the state of the (virtual) server $\hat{p}_i$ after the sharing is done.

Theorem 4. *Let $\Pi_{(n+1,m),t}$ be a protocol as described above among $n+1$ clients and $m = O(k)$ servers which perfectly securely (and robustly) realizes the functionality $\mathcal{F}_D$ in the presence of $t < m/3$ corrupted servers. The ($\mathcal{F}^{\text{1SCP}}_{\mathsf{Corr}}(\mathcal{P}, \mathcal{D}, m, t, \mathcal{R})$-hybrid) protocol $\Pi_{\text{1SCP}}(\mathcal{P}, \mathcal{D}, \mathcal{R}, m, t, p)$ unconditionally securely realizes the functionality $\mathcal{F}_{\text{1SCP}}(\mathcal{P}, \mathcal{D}, \mathcal{R}, p)$ with identifiable abort.*

Proof. To prove the statement we need to show that protocol $\Pi_{\text{1SCP}}(\mathcal{P}, \mathcal{D}, \mathcal{R}, m, t, p)$ information-theoretically securely realizes the functionality $[\mathcal{F}_{\text{1SCP}}(\mathcal{P}, \mathcal{D}, \mathcal{R}, p)]^{\mathtt{ID}}_{\perp}$. We prove the statement in the $\mathcal{F}_{\text{COM}}$-hybrid world, i.e., where all commitment are done by calls to the functionality $\mathcal{F}_{\text{COM}}$. Because our commitments are unconditionally secure and all the signatures used for the commitments are

Protocol $\Pi_{\text{1SCP}}(\mathcal{P}, \mathcal{D}, m, t, \mathcal{R}, p)$

Setup-Commit Phase: To obtain the appropriate setup, i.e., upon receiving input $(\mathsf{reqWitness})$, prover p sends $(\mathsf{CorrRand}, p)$ to the sampling functionality $\mathcal{F}^{\text{1SCP}}_{\mathsf{Corr}}(\mathcal{P}, \mathcal{D}, m, t, \mathcal{R})$, which distributes the following random strings and signatures (where every $p_j \in \mathcal{P}$ receives the corresponding verification keys):

- The prover p receives a sharing $\langle w \rangle = (\langle w \rangle^1, \ldots, \langle w \rangle^m)$ of w along with corresponding signatures $\sigma(\langle w \rangle^1), \ldots, \sigma(\langle w \rangle^m)$ and (privately) outputs $(\mathsf{witness}, w)$.
- Every $p_i \in \mathcal{P}$ receives the challenge-string c_i along with a corresponding signature $\sigma(c_i)$.
- The prover also receives random strings $v_1, \ldots, v_m$ along with corresponding signatures $\sigma(v_1), \ldots, \sigma(v_m)$ to use for committing to the server's views in $\Pi_{(n+1,m),t}$.

Prove Phase: Upon p receiving input $(\mathsf{ZK\text{-}prover}, x)$ the following steps are executed:

1. If $\mathcal{R}(x, w) = 0$ then p broadcasts $(\mathsf{not\text{-}verified}, p)$ and every party halts with output $(\mathsf{not\text{-}verified}, p)$. Otherwise, p broadcasts $(\mathcal{R}, x)$.
2. p emulates in its head the second phase of protocol $\Pi_{(n+1,m),t}$ where each server $\hat{p}_j \in \hat{\mathcal{P}} = \{\hat{p}_1, \ldots, \hat{p}_m\}$ has private input $\langle w \rangle^j$ and public input x.
3. For each $\hat{p}_j \in \hat{\mathcal{P}}$, p commits, by invocation of protocol $\Pi_{\text{COM}}(\mathcal{P})$, to the view $\textsc{View}_j \in \{0,1\}^{\mathbb{V}_j}$ of $\hat{p}_j$ in the above emulated execution using v_j from his setup.
4. For each $p_i \in \mathcal{P}$: p_i announces the random string c_i and the corresponding signature $\sigma(c_i)$ and every $p_j \in \mathcal{P}$ verifies, using his corresponding verification keys, validity of the signatures and aborts with p_i in case the check fails.
5. The parties compute $c = \sum_{i=1}^{n} c_i$ and use it as random coins to sample a random t-size set $\mathcal{J} \subseteq [m]$.
6. For each $j \in \mathcal{J}$: p opens the commitment to $\textsc{View}_j$ and announces the signature $\sigma(\langle w \rangle^j)$. If any of the openings fails or any of the announced signatures is not valid for the input-share appearing in the corresponding view, then the protocol aborts with p_i.
7. Otherwise, the parties check that the announced views are consistent with an execution of protocol $\Pi_{(n+1,m),t}$ with the announced inputs in which the (global) output is 1, i.e., they check that in all the announced views the output equals 1 and all signatures are valid, and that for all pairs $(j, k) \in \mathcal{J}^2$: the incoming messages in $\hat{p}_j$'s view match the outgoing messages in $\hat{p}_k$'s view. If any of these checks fails then the protocol aborts with p_i, otherwise, every party outputs $(\mathsf{verified}, x, p)$.

generated by use of fresh independent keys (i.e., different invocations of Π_{COM} do not share a state), the security of the protocol Π_{1SCP} follows then directly by applying the UC composition theorem.

We consider two cases: (1) Prover p is honest, and (2) Prover p is corrupted. In both cases, the simulator $\mathcal{S}$ invokes the adversary $\mathcal{A}$ and relays communication between $\mathcal{A}$ and $\mathcal{Z}$. We point out that in both cases, by inspection of the protocol one can verify that the protocol might only abort with the identity of a corrupted party p_i; this follows from the unforgeability of digital signatures[7] and the commitment scheme, as the protocol aborts only when some party p_i attempts to cheat by opening an inconsistent commitment, opening an inconsistent signature, or if p_i is the prover, by trying to prove a false statement.

In **Case 1,** the simulator works as follows:

In the setup-commit phase, $\mathcal{S}$ samples a witness w' from the distribution $\mathcal{D}$ and emulates an invocation of $\mathcal{F}^{\text{ZK}}_{\mathsf{Corr}}(\mathcal{P}, w', m, t, \mathcal{R})$ for prover p, while storing the corresponding values. Observe, that by emulating the setup, the simulator already computes all the selection strings $\overline{c_1}, \ldots, \overline{c_m}$

[7] We do not need the consistency here as the parties do not get to see the signing keys.

which will be used in the simulation before any of the proof starts; thereby, $\mathcal{S}$ knows, before even starting to simulate the prove phase, the set $\mathcal{J} \subseteq [m]$ of (virtual) clients whose views will to be opened during the simulation of the proof-phase and can prepare for them. Completing the setup-commit phase, $\mathcal{S}$ hands to $\mathcal{A}$ his setup-values. Clearly, the view of the adversary in this simulation (i.e., his setup messages) is distributed identically to the corresponding view in a real-protocol execution.

In the beginning of the prove phase, $\mathcal{S}$ receives his output from the functionality $[\mathcal{F}_{\text{1SCP}}]_{\perp}^{\texttt{ID}}$. If the output is $(\mathsf{not\text{-}verified}, p)$ then $\mathcal{S}$ emulates p broadcasting $(\mathsf{not\text{-}verified}, p)$ towards $\mathcal{A}$, and instructs $[\mathcal{F}_{\text{1SCP}}]_{\perp}^{\texttt{ID}}$ to deliver the output to the honest parties. Otherwise, i.e., if the output is $(\mathsf{verified}, x, p)$ then $\mathcal{S}$ emulates towards $\mathcal{A}$ the protocol execution as follows: $\mathcal{S}$ uses the simulator $\mathcal{S}_{\Pi_{(n+1,m),t}}$ for $\Pi_{(n+1,m),t}$ corrupting the players with indexes in $\mathcal{J}$ (which is guaranteed to exist by the security of $\Pi_{(n+1,m),t}$) on inputs the shares generated in the emulation of the setup $\mathcal{F}_{\mathsf{Corr}}^{\text{ZK}}$. For each $j \in \mathcal{J}$, $\mathcal{S}$ emulates a commitment to the view $\overline{\text{VIEW}_j}$ of $\hat{p}_j$ as generated by $\mathcal{S}_{\Pi_{(n+1,m),t}}$; for all $j \in [m] \setminus \mathcal{J}$, $\mathcal{S}$ emulates commitments to random views $\overline{\text{VIEW}_j}$ (of appropriate size) towards $\mathcal{A}$ (note that as we are in the $\mathcal{F}_{\text{COM}}$-hybrid world, $\mathcal{A}$ only expects a $(\mathsf{receipt})$-message in this step of the simulation). Subsequently, for each honest $p_j \in \mathcal{P}$, $\mathcal{S}$ emulates towards $\mathcal{A}$ announcement of the choice-strings $\overline{c_j}$ and the corresponding signatures; symmetrically, $\mathcal{S}$ receives from $\mathcal{A}$ the selection strings c_j for corrupted p_j's along with the corresponding openings. If for some of the choice-strings c_j which $\mathcal{A}$ announced the signature verification fails (i.e., it does not verify for the key of some $p_i \in \mathcal{P}$), then $\mathcal{S}$ sends (abort, p_i) to $[\mathcal{F}_{\text{1SCP}}]_{\perp}^{\texttt{ID}}$. Otherwise, $\mathcal{S}$ emulates towards $\mathcal{A}$ opening of the views $\overline{\text{VIEW}_j}$ for $j \in \mathcal{J}$ and instructs the functionality $[\mathcal{F}_{\text{1SCP}}]_{\perp}^{\texttt{ID}}$ to deliver its output (i.e., $(\mathsf{verified}, x, p)$) to all parties.

The soundness of the simulation is argued similarly to the proof of the zero-knowledge property from [30, Theorem 3.1]. In particular, it is easy to verify that the simulation might only abort with p_j when the adversary tries to announce a choice-string $c_j \neq \overline{c_j}$ in which case, the unforgeability of the signature scheme ensures that (with overwhelming probability) the real protocol would also have aborted with p_j. When all announcements succeed, the announced set $\mathcal{J}$ will be the one the simulator has prepared the views for. The fact that these views are statistically indistinguishable from the protocol execution follows, as in [30, Theorem 4.1], from the fact that $\mathcal{S}_{\Pi_{(n+1,m),t}}$ is a perfect simulator for $\Pi_{(n+1,m),t}$ and the fact that the outputs of the (virtual) parties whose views are opened are t shares of a t-private secret sharing and, therefore, are independent of the shared value.

In **Case 2** (i.e, the case of a corrupted prover) the argument is similar to the soundness argument from [30, Theorem 4.1]:

In the setup-commit phase, $\mathcal{S}$ sends $(\mathsf{reqWitness})$ to $[\mathcal{F}_{\text{1SCP}}]_{\perp}^{\texttt{ID}}$ and receives the witness w. Subsequently, $\mathcal{S}$ emulates an invocations of $\mathcal{F}_{\mathsf{Corr}}^{\text{ZK}}(\mathcal{P}, w, m, t, \mathcal{R})$ with prover p and stores the corresponding values. Recall, that, as in Case 1, by emulating the setup, the simulator already computes all the selection strings $c_1, \ldots, c_n$ before the proof phase starts and therefore knows the set $\mathcal{J}$ of virtual players whose views are to be opened. Finally, $\mathcal{S}$ hands to $\mathcal{A}$ the sharing $\langle w \rangle$, as computed by the emulation of $\mathcal{F}_{\mathsf{Corr}}^{\text{ZK}}$, along with all his ($\mathcal{A}$'s) other messages from the emulation. Clearly, the above simulation of this phase is perfect.

For the prove phase $\mathcal{S}$ emulates towards $\mathcal{A}$ the honest parties/verifiers in the protocol execution as follows: In Step 3, if the corrupted prover p broadcasts $(\mathsf{not\text{-}verified}, p)$ then $\mathcal{S}$ inputs $(\mathsf{ZK\text{-}prover}, x')$ for some x' with $\mathcal{R}(x', w) = 0$ which result in every party in the ideal setting outputting $(\mathsf{not\text{-}verified}, p)$ as they would do in the protocol. Otherwise, as in Case 1, $\mathcal{S}$ emulates the opening of the challenge commitments for honest parties and receives from $\mathcal{A}$ his openings; if for any corrupted $p_i \in \mathcal{P}$ the opening aborts then the simulator sends to $[\mathcal{F}_{\text{1SCP}}]_{\perp}^{\mathtt{ID}}$ the message (abort, p_i). Otherwise, $\mathcal{S}$ receives from $\mathcal{A}$ the openings of the committed views for the virtual parties in $\mathcal{J}$ (again, if some p_i fails to open $\mathcal{S}$ sends (abort, p_i) to $[\mathcal{F}_{\text{1SCP}}]_{\perp}^{\mathtt{ID}}$ as the honest parties would in the protocol). Subsequently, $\mathcal{S}$ checks, as the honest parties would that the announced views are consistent (with each-other and with the sharing of w which $\mathcal{S}$ gave $\mathcal{A}$ in the setup-commit phase). If the check fails $\mathcal{S}$ sends (abort, p) to $[\mathcal{F}_{\text{1SCP}}]_{\perp}^{\mathtt{ID}}$, otherwise, $\mathcal{S}$ sends $(\mathsf{ZK\text{-}prover}, , x)$ to $[\mathcal{F}_{\text{1SCP}}]_{\perp}^{\mathtt{ID}}$ and allows it to deliver the outputs to honest parties. Since the simulator follows exactly the protocol of honest parties (who, recall, have no input), $\mathcal{S}$ is a good simulation as long as correctness of the outputs is guaranteed. Because the relation $\mathcal{R}$ and the public input x is necessarily part of all virtual parties in $\Pi_{(n+1,m),t}$, the only way that the adversary can cheat is by committing to views which are inconsistent with an honest execution of $\Pi_{(n+1,m),t}$ for the witness w. In the remainder of the proof we argue that such an adversary will be caught with overwhelming probability.

The argument is similar to the proof of soundness from [30, Theorem 3.1]; the only difference is that we need to ensure that the prover uses the the witness w which he was committed to in the setup phase;[8] however, for self-containment we include here the complete argument. In particular, we show that if the (adversarial) prover tries to cheat, i.e., uses $w' \neq w$ or uses x such $\mathcal{R}(x, w) = 0$, then he is caught with overwhelming probability. To this direction, consider the following inconsistency graph G defined on the m committed views $\textsc{View}_1, \ldots, \textsc{View}_m$: The graph G has m vertices corresponding to the m views and there is an edge (i, j) in G if any of the following conditions is satisfied:

- The (private) input (i.e., the witness share) in any of the views $\textsc{View}_i$ or $\textsc{View}_j$ is not the share $\langle w \rangle_i$ or $\langle w \rangle_j$ of w which the prover was committed to in the setup-distribution phase.
- The views $\textsc{View}_i$ and $\textsc{View}_j$ are inconsistent with respect to $\Pi_{(n+1,m),t}, \mathcal{R}, x$ *and* $(\langle w \rangle_i, \langle w \rangle_j)$, that is incoming messages from $\hat{p}_j$ in the view $\textsc{View}_i$ are different from outgoing messages to $\hat{p}_i$ (implicit) in the view $\textsc{View}_j$.

We consider the same cases for the graph G as in [30, Theorem 4.1]:

Case 2A: G has a vertex cover set B of size at most t. We argue that in this case the only way the adversary can cheat is by choosing x such that $\mathcal{R}(x, w) = 0$ in which case the output in all views $\textsc{View}_j$ with $j \notin B$ must be 0. To this direction, consider an execution of $\Pi_{(n+1,m),t}$ where the adversary corrupts the players in B and makes them misbehave so that the view of players p_j with $j \notin B$ is $\textsc{View}_j$. Since B is a vertex cover, every pair of views $(\textsc{View}_i, \textsc{View}_j)$ with $i, j \in [m] \setminus B$ are not connected in the graph G and therefore , by definition, they are consistent and they both include the right shares of $\langle w \rangle$. The perfect t-robustness of $\langle w \rangle$ ensures that in this case the actual witness w is used in the evaluation of $\mathcal{R}(x, w)$. Hence, the only way to cheat is for the prover to have given x such that $\mathcal{R}(x, w) = 0$. But, in this case the perfect t-robustness of $\Pi_{(n+1,m),t}$ ensures that the corruption of parties in B cannot influence the correctness of the output of the honest players (i.e., players with indexes in $[m] \setminus B$) which must be 0. Hence, to catch the adversary $\mathcal{A}$ cheating in this case it suffices to open one player $\hat{p}_j$'s view with $j \in [m] \setminus B$; by the choice of the parameters, the probability that this does not happen is at most $(t/m)^t = 2^{-\Omega(t)} = 2^{-\Omega(k)}$, i.e., negligible.

[8] Note that in [30] there is no such requirement as the prover is free to chose the witness as long as it satisfies the relation $\mathcal{R}$.

Case 2B: min-VC$(G) > t$ (where, as in [30], min-VC(G) denotes the size of a minimum vertex-cover of G). We argue that in such a graph, opening a random constant fraction of the vertices, hits an edge with overwhelming probability. Indeed, as argued in [30], such a graph G must have a matching of size $> t/2$. Now it is clear that if the random challenge (note that the challenge is chosen from the setups and is therefore always uniformly random) picks to open both vertices in at least one edge of G then the protocol will abort with p. Now similar to [30], the probability that the random selection misses all edges in G is smaller than the probability that it misses all edges of the matching which is again $2^{-\Omega(t)} = 2^{-\Omega(k)}$. □

The multiple-proof extension of $\mathcal{F}_{\mathbf{1scp}}$ In order to realize functionality $\mathcal{F}_{\text{SCP}}$ we need to extend $\mathcal{F}_{\text{1SCP}}$ to distribute a vector $\vec{R} = (R_1, \dots, R_n)$ of witnesses, one for each party, (instead of only one witness) sampled from some efficient distribution $\mathcal{D}$, and allow every $p_i \in \mathcal{P}$ to prove up to q statements of the type $\mathcal{R}(R_i, x)$ for potentially different public inputs x. The corresponding sampling functionality, denoted as $\mathcal{F}^{\text{SCP}}_{\textsf{Corr}}$, (see below) is derived as follows: it first samples $\vec{R}$ and subsequently it emulates, for each $p_i \in \mathcal{P}$, q independent invocations of $\mathcal{F}^{\text{ZK}}_{\textsf{Corr}}(\mathcal{P}, R_i, m, t, \mathcal{R}_i)$ on input $(\textsf{CorrRand}, p_i)$ with $m = O(k)$ and $t = \lceil m/3 \rceil - 1$.

$\mathcal{F}^{\text{SCP}}_{\textsf{Corr}}(\mathcal{P}, \mathcal{D}, \vec{\mathcal{R}} = (\mathcal{R}_1, \dots, \mathcal{R}_n), q)$

The functionality is parameterized but an efficiently sampleable distribution $\mathcal{D}$ with range $(\{0,1\}^{\text{poly}(k)})^n$ out of which the witnesses will be drawn, a vector of NP relations $(\mathcal{R}_1, \dots, \mathcal{R}_n)$ and an upper bound $q = poly(k)$ on the number of statements that each party will be allowed to prove.

Upon receiving message $(\textsf{CorrRand})$ from any party $p \in \mathcal{P}$ (or the adversary), if such a message was already received ignore it, else do the following:

1. Sample $(R_1, \dots, R_n)$ from distribution $\mathcal{D}$
2. For each $p_i \in \mathcal{P}$ and each $\ell = 1, \dots, q$ emulate an invocations of $\mathcal{F}^{\text{ZK}}_{\textsf{Corr}}(\mathcal{P}, R_i, m, t, \mathcal{R}_i, p_i)$ on input $(\textsf{CorrRand}, p_i)$ with parameters $m = k$ and $t = \lceil m/3 \rceil - 1$ (without distributing the outputs).
3. Distribute all the outputs generated by invoking $\mathcal{F}^{\text{ZK}}_{\textsf{Corr}}$ in the previous step.

Given such a sampling functionality the protocol Π_{SCP} for unconditionally securely realizing $\mathcal{F}_{\text{SCP}}$ with identifiable abort is straight-forward: The parties receive the random strings $R_1, \dots, R_n$ along with q proof setups for each party. Then, for each invocation of the prove phase, party p_i executes the prove phase of protocol Π_{1SCP} using the corresponding proof setup.[9]

[9] Recall that we implicitly assume that all messages generated from the setup have unique identifiers so that the parties know which ones to use for which proof.

Protocol $\Pi_{\text{SCP}}(\mathcal{P}, \mathcal{D}, \vec{\mathcal{R}}, q)$

Setup-Commit Phase: To obtain the appropriate setup, upon receiving message $(\mathsf{reqWitness})$ any party $p \in \mathcal{P}$ (e.g., p_1) sends $(\mathsf{CorrRand})$ to the sampling functionality $\mathcal{F}^{\text{SCP}}_{\mathsf{Corr}}(\mathcal{P}, \mathcal{D}, \vec{\mathcal{R}} = (\mathcal{R}_1, \ldots, \mathcal{R}_n), q)$, which samples $\vec{R} = (R_1, \ldots, R_n)$ from $\mathcal{D}$ and distributes the following random strings and signatures (where every $p_j \in \mathcal{P}$ receives the corresponding verification keys):

- Every $p_i \in \mathcal{P}$ receives q-many sharing $\langle R_i \rangle_1, \ldots, \langle R_i \rangle_q$ of R_i along with corresponding signatures (and privately outputs $(\mathsf{witness}, R_i)$).
- For every $(p_i, p_j) \in \mathcal{P}^2$, p_i receives q challenge-strings $c^1_{i,j}, \ldots, c^q_{i,j}$ along with a corresponding signature $\sigma(c_i)$, to be used in the q proofs with prover p_j, along with corresponding signatures.
- Every p_i receives q vectors of random strings $\vec{v}_{i,1}, \ldots, \vec{v}_{i,q}$, where for $\ell \in [q] : \vec{v}_{i,\ell} = (v_{i,\ell,1}, \ldots, v_{i,\ell,m})$ is to be used in the ℓth proof, along with corresponding signatures.

Prove Phase: Upon p receiving input $(\mathsf{ZK\text{-}prover}, x)$, if q-many inputs of the type $(\mathsf{ZK\text{-}prover}, \cdot)$ were already received, then p_i broadcasts $(\mathsf{not\text{-}verified}, p)$ and every party halts with output $(\mathsf{not\text{-}verified}, p)$. Otherwise, i.e., if $\ell < q$ inputs $(\mathsf{ZK\text{-}prover}, \cdot)$ were received, p_i initiates the prove phase of protocol $\Pi_{1\text{SCP}}$ where the parties use the ℓth setup.[a]

[a] Observe that in each execution of $\Pi_{1\text{SCP}}$ (i.e., for every input $(\mathsf{ZK\text{-}prover}, x)$) the prover starts off by broadcasting a message, which allows the parties to keep track of the number of inputs.

Theorem 5. *Protocol $\Pi_{\text{SCP}}(\mathcal{P}, \mathcal{D}, \mathcal{R}, q)$ unconditionally securely realizes the functionality $\mathcal{F}_{\text{SCP}}(\mathcal{P}, \mathcal{D}, \mathcal{R}, q)$ with identifiable abort.*

The proof follows from the security of $\Pi_{1\text{SCP}}$ by a direct application of the universal composition with joint state (JUC) theorem [8].

4.3 The "Semi-honest to Malicious with Abort" Compiler

We are now ready to describe our main compiler, denoted as $\mathrm{C}(\cdot)$ which compiles any given protocol π_{sh} secure in the semi-honest model using (only) correlated randomness into a protocol $\mathrm{C}(\pi_{sh})$ which is secure with abort in the (malicious) correlated randomness model.[10]

We make the following simplifying assumptions on the semi-honest protocol π_{sh} which are without loss of generality, since all existing semi-honest protocols in the correlated randomness model can be trivially turned to satisfy them:

- We assume that π_{sh} has a known (polynomial) upper bound $\mathtt{Rnd}_{\pi_{sh}}$ on the number of rounds, where in each round every party sends a single message.
- We assume that π_{sh} is *deterministic*. Any π_{sh} can be turned into such by having the setup include for each $p_i \in \mathcal{P}$ a uniformly random and independent string r_i that p_i uses as his coins.
- Finally, we assume that π_{sh} starts off by having every party send to all parties a one-time pad encryption of his input x_i using as key the first $|x_i|$ bits from r_i (those bits are not reused). Clearly, this modification does not affect the security of π_{sh} as the simulator can easily simulate this step by broadcasting a random string. Looking ahead in the proof, this will allow the simulator to extract the corrupted parties' inputs.

The compiler $\mathrm{C}(\pi_{sh})$ uses the protocol Π_{SCP} as follows: Denote by $R^{\mathsf{sh}} = (R^{\mathsf{sh}}_1, \ldots, R^{\mathsf{sh}}_n)$ the setup used by π_{sh} and by $\mathcal{D}^{\mathsf{sh}}$ the corresponding distribution. Let also $\mathcal{R}_{\pi_{sh},i}$ denote the relation

[10] Note that $\mathrm{C}(\pi_{sh})$ uses broadcast which can be trivially realized by a protocol assuming appropriate correlated randomness, e.g., [40].

corresponding to p_i's next message function. More concretely, if $h_{\pi_{sh},i} \in \{0,1\}^*$ denotes the history of messages seen by p_i and m is a message, then $\mathcal{R}_{\pi_{sh},i}((h_{\pi_{sh},i}, m), R_i) = 1$ if m is the next message of p_i in an execution with history $h_{\pi_{sh},i}$ and setup R_i, otherwise $\mathcal{R}_{\pi_{sh},i}((h_{\pi_{sh},i}, m), R_i) = 0$. The compiled protocol $\mathrm{C}(\pi_{sh})$ starts by executing the setup-commit phase of protocol $\Pi_{\text{SCP}}(\mathcal{P}, \mathcal{D}^{\mathsf{sh}}, \vec{\mathcal{R}} = (\mathcal{R}_{\pi_{sh},1}, \ldots, \mathcal{R}_{\pi_{sh},n}), \mathtt{Rnd}_{\pi_{sh}})$. Subsequently, every $p_i \in \mathcal{P}$ executes his π_{sh} instructions, where in each round instead of sending its message m over the point-to-point channel, p_i broadcasts m and proves, using the proof phase of protocol Π_{SCP}, that $\mathcal{R}_{\pi_{sh},i}((h_{\pi_{sh},i}, m), R_i) = 1$. If Π_{SCP} aborts with some p_i then our compiler also aborts with p_i. Otherwise, the security of Π_{SCP} ensures that every p_i followed π_{sh} for the given setup; therefore, security of our compiler follows from the security of π_{sh}. Note that the corresponding sampling functionality for $\mathrm{C}(\pi_{sh})$ is computable in time polynomial in the running time of the sampling functionality $\mathcal{F}_{\mathsf{Corr}}^{\mathcal{D}^{\mathsf{sh}}}$ for protocol π_{sh}.

Protocol $\mathrm{C}(\pi_{sh})$

Setup: The protocol works in the $\mathcal{F}_{\text{SCP}}(\mathcal{P}, \mathcal{D}^{\mathsf{sh}}, \vec{\mathcal{R}} = (\mathcal{R}_{\pi_{sh},1}, \ldots, \mathcal{R}_{\pi_{sh},n}), \mathtt{Rnd}_{\pi_{sh}})$-hybrid world, where $\mathcal{P}$ is the player set, $\mathcal{D}^{\mathsf{sh}}$ is the distribution out of which the setup for protocol π_{sh} is drawn, $\mathcal{R}_{\pi_{sh},i}$ is the next-message relation defined above, and $\mathtt{Rnd}_{\pi_{sh}}$ is (an upper bound on) the number of rounds in π_{sh}. The parties maintain a public list $h_{\pi_{sh}}$ (initially empty) which, at any point, includes all the messages broadcasted in the computation.

1. The parties execute the setup-commit phase of protocol Π_{SCP} and receives their setup string including the witnesses $R_1^{\mathsf{sh}}, \ldots, R_n^{\mathsf{sh}}$.
2. Let $\mathtt{Rnd}_{\pi_{sh}}$ be the number of rounds of protocol π_{sh}. Upon receiving inputs $(x_1, \ldots, x_n)$ (where p_i receives input x_i) the parties execute the following steps (sequentially) for : $\rho = 1, \ldots, \mathtt{Rnd}_{\pi_{sh}}$:
    1. Every $p_i \in \mathcal{P}$ computes his ρ-round message $m_{\rho,i}$ for π_{sh} on input x_i and setup string R_i, and broadcasts it. If some party p_i broadcasts an inconsistent message abort with p_i.
    2. Every $p_i \in \mathcal{P}$ proves that m is indeed his next π_{sh}-message by invoking the second phase of protocol Π_{SCP} with public input $(h_{\pi_{sh}}, m)$ and private input R_i^{sh}. If Π_{SCP} aborts with p_j or outputs $(\mathsf{not\text{-}verified}, p_j)$ for some $p_j \in \mathcal{P}$ then $\mathrm{C}(\pi_{sh})$ aborts with p_j.
    3. Otherwise every party includes the broadcasted messages $m_{\rho,1}, \ldots, m_{\rho,n}$ to the history $h_{\pi_{sh}}$.

Output: Every party outputs his output as computed by π_{sh} and halts.

Theorem 6. *Let π_{sh} be a protocol as above which unconditionally UC realizes a functionality $\mathcal{F}$ in the presence of a semi-honest adversary in the $\mathcal{F}_{\mathsf{Corr}}^{\mathcal{D}^{\mathsf{sh}}}$-hybrid (correlated randomness) model. Then the compiled protocol $\mathrm{C}(\pi_{sh})$ unconditionally UC realizes the functionality $\mathcal{F}$ with identifiable abort in the presence of a malicious adversary in the $\mathcal{F}_{\mathsf{Corr}}^{\text{SCP}}$-hybrid (correlated randomness) model.*

Proof (sketch). We prove the $\mathrm{C}(\pi_{sh})$ statistically UC securely realizes the functionality $[\mathcal{F}]_{\perp}^{\mathtt{ID}}$. For simplicity, we do the proof assuming $\mathrm{C}(\pi_{sh})$ is a $[\mathcal{F}_{\text{SCP}}]_{\perp}^{\mathtt{ID}}$-hybrid protocol, where invocation of Π_{SCP} is replaced by calls to $[\mathcal{F}_{\text{SCP}}]_{\perp}^{\mathtt{ID}}$. The proof follows then by applying the composition theorem.

Simulation is as follows: the simulator $\mathcal{S}$ uses the adversary $\mathcal{A}$ in a black-box manner (and forwards all messages sent between $\mathcal{A}$ and $\mathcal{Z}$). $\mathcal{S}$ also uses the simulator $\mathcal{S}_{\mathsf{sh}}$ for the protocol π_{sh}(which is assumed to exists by the security of π_{sh}), where $\mathcal{S}$ plays towards $\mathcal{S}_{\mathsf{sh}}$ the role of an adversary in π_{sh}. Initially, $\mathcal{S}$ receives the setup $\overline{R} = (\overline{R_1}, \ldots, \overline{R_n})$ from $\mathcal{S}_{\mathsf{sh}}$ (i.e., the simulated setup for π_{sh}), and hands to $\mathcal{A}$ the message $(\mathsf{witness}, \overline{R_i})$ for each corrupted p_i. In the first round, the simulator simulates towards the adversary broadcasting of one-time pad encryptions of random inputs[11] for honest parties with keys taken from $\overline{R}$ and receives from $\mathcal{A}$ one-time-pad encryptions of the corrupted players inputs. Note that, as the simulator knows all the $\overline{R_i}$'s that the adversary is supposed to use (and therefore the random keys he is supposed to use for encrypting), he can extract the adversary's inputs $\hat{x}_i$'s (for corrupted p_i's) by decrypting the broadcasted message.

From that point on, the simulator uses the simulator $\mathcal{S}_{\texttt{sh}}$ for computing the messages of honest parties in each round. More precisely, $\mathcal{S}$ maintains (as the parties in $\mathrm{C}(\pi_{sh})$ would) an (initially empty) list of messages $\overline{h_{\pi_{sh}}}$. In each round ρ:

- For each honest p_i: $\mathcal{S}$ sends $\mathcal{A}$ the ρ-round message $\overline{m_{\rho,i}}$ of p_i as generated by $\mathcal{S}_{\texttt{sh}}$ along with a message $(\mathsf{verified}, (h_{\pi_{sh}}, m_{\rho,i}), p_i)$ as $[\mathcal{F}_{\text{SCP}}]_{\perp}^{\texttt{ID}}$ would in a proof with the honest p_i (note that and honest p_i would never try to prove a false message).
- For each corrupted p_i, $\mathcal{S}$ receives from $\mathcal{A}$ his ρ-round broadcasted message $m_{\rho,i}$ along with the message $(\mathsf{ZK\text{-}prover}, x)$ for the functionality $[\mathcal{F}_{\text{SCP}}]_{\perp}^{\texttt{ID}}$. If $x \neq (\overline{h_{\pi_{sh}}}, m_{\rho,i})$ then $\mathcal{S}$ sends (abort, p_i) to its functionality $[\mathcal{F}]_{\perp}^{\texttt{ID}}$ and halts. Otherwise, $\mathcal{S}$ includes the messages $m_{\rho,i}$ broadcasted by $\mathcal{A}$ in this round to $h_{\pi_{sh}}$ and hands them to $\mathcal{S}_{\texttt{sh}}$ (as the messages that $\mathcal{S}_{\texttt{sh}}$ expects to see from its adversary).

At the end of the simulation, if no abort occurred $\mathcal{S}$ hand to $[\mathcal{F}]_{\perp}^{\texttt{ID}}$ the extracted inputs $\hat{x}_i$ (and instructs $[\mathcal{F}]_{\perp}^{\texttt{ID}}$ to deliver its outputs).

It is straight-forward to verify that the above is a good simulator: indeed, the use of $[\mathcal{F}_{\text{SCP}}]_{\perp}^{\texttt{ID}}$ ensures that the adversary might either faithfully execute the protocol π_{sh} (on the inputs encrypted in the first step) or force an abort with some corrupted party. When the adversary does faithfully execute protocol π_{sh} then the simulated transcript consists of the messages as sent by $\mathcal{S}_{\texttt{sh}}$ along with confirmation of this fact (corresponding to outputs of $[\mathcal{F}_{\text{SCP}}]_{\perp}^{\texttt{ID}}$); hence, in this case the statistically closeness of the simulated and the real transcripts follows from the fact that $\mathcal{S}_{\texttt{sh}}$ is a good simulator from π_{sh} (which is statistically secure) □

Note that any (semi-honest) OT-hybrid protocol can be cast as a protocol in the correlated randomness model by precomputing the OT. Hence, by instantiating π_{sh} with any semi-honest OT hybrid protocol. e.g., [20], we obtain the following corollary.

Corollary 7. *There exists a protocol which unconditionally UC realizes any well-formed [7] multi-party functionality with identifiable abort.*

The question of feasibility of unconditional security with identifiable abort from correlated randomness has been open even in the simpler *standalone* model [21, 19, 4] (for self-containment we have included a formal description in Appendix B). As a corollary of Theorem 6 one can derive a positive statement also for that model.

Corollary 8 (Stand-alone security with identifiable abort)**.** *There exists a protocol which unconditionally securely evaluates any given function f with identifiable abort in the stand alone correlated randomness model.*

[11] Recall that we assume that π_{sh} starts by having every party broadcast its input one-time pad encrypted with randomness drawn from the setup.

5 SFE Using Black-box OT

In this section, we provide a generic MPC protocol which is (computationally) secure with identifiable abort making black-box use of an (adaptively) secure UC protocol for one-out-of two oblivious transfer $\mathcal{F}_{\mathsf{OT}}$ (see [39] for a formal description) in the Common Reference String (CRS) model.

The high-level idea of our construction is the following: as we have already provided an unconditional implementation of ID-MPC based (only) on correlated randomness, it suffices to provide a protocol $\Pi^{\mathrm{CSP}}_{\mathtt{Corr}}$ with the above properties for implementing the corresponding sampling functionality $\mathcal{F}^{\mathrm{SCP}}_{\mathsf{Corr}}$. Indeed, given such a protocol $\Pi^{\mathrm{CSP}}_{\mathtt{Corr}}$, we can first use it to compute the setup needed for $\mathrm{C}(\pi_{sh})$ (for any appropriate semi-honest protocol π_{sh}, e.g., the one from [21]) and then use π_{sh} to evaluate any given functionality; if either the setup generation or $\mathrm{C}(\pi_{sh})$ aborts with some p_i then the construction also aborts with p_i.

In the remainder of this section we describe $\Pi^{\mathrm{CSP}}_{\mathtt{Corr}}$. In fact, we provide a protocol $\Pi^{\mathcal{D}}_{\mathtt{Corr}}$ which allows to implement any sampling functionality $\mathcal{F}^{\mathcal{D}}_{\mathsf{Corr}}$ for a given efficiently computable distribution $\mathcal{D}$. The key idea behind our construction in the following: as the functionality $\mathcal{F}^{\mathcal{D}}_{\mathsf{Corr}}$ receives no (private) inputs from the parties, we can have every party commit to its random tape, and then attempt to realize $\mathcal{F}^{\mathcal{D}}_{\mathsf{Corr}}$ by a protocol which is secure with (non-identifiable) abort; if the evaluation aborts then the parties open the commitments to their random tapes and use these tapes to detect which party cheated. Note that, as the parties have no private inputs, announcing their views does not violate privacy of the computation.

For the above idea to work we need to ensure that deviation from the honest protocol can be consistently detected by every party (upon opening the committed random coins). Therefore, we define the following *$\mathcal{P}$-verifiability* property. For any given execution of a protocol Π, we say that a party p_i *correctly executed* Π *with respect to* (x_i, r_i) *(up to round* ρ*) in the CRS model* if p_i sent all his messages as instructed by Π on this input x_i, random coins r_i and the common reference string C. Let Π be a protocol in the CRS model which starts by having every party commit to its random tape. Π is *$\mathcal{P}$-verifiable* if there exists a deterministic polynomial algorithm $\mathcal{D}$, called *the detector*, with the following property: given the CRS, the inputs of the parties, their committed randomness, and the view of any honest p_j, $\mathcal{D}$ outputs the identity of a party $p_i \in \mathcal{P}$ who did not correctly execute Π (if such a party exists). Formally:

Definition 9 ($\mathcal{P}$-verifiability)**.** Let Π be a multi-party protocol in the CRS model which starts by having every party publicly commit to its randomness. We say that Π is $\mathcal{P}$-verifiable if there exists a deterministic polynomial algorithm $\mathcal{D}$, called *the detector*, which on input the CRS, an n-vector $\vec{U} = ((x_1, r_1), \ldots, (x_n, r_n))$ of input/randomness-pairs, where r_i is the p_i's initially committed random string, and the view of any honest $p \in \mathcal{P}$ in any given round of an execution of Π, $\mathcal{D}$ outputs a value $i \in \{0, 1, \ldots, n\}$ (the same for all $p \in \mathcal{P}$) satisfying the following properties: if there exists at least one party in $\mathcal{P}$ that did not correctly execute Π with respect to (x_i, r_i) and the CRS up to that round, then i is the index of such a party, otherwise, i.e., if no such party exists, $i = 0$.

As our protocols makes black-box use of a UC secure one-out-of-two-OT (in short, 12OT) protocol in the CRS model, for it to be $\mathcal{P}$-verifiable the underlying 12OT protocol needs to also be $\mathcal{P}$-verifiable. Therefore, in the following, first, we show how to obtain from any given OT protocol $\Pi_{\mathtt{OT}}$ a $\mathcal{P}$-verifiable OT protocol Π_{VOT} (making black-box use of $\Pi_{\mathtt{OT}}$), and, subsequently, we show how to use Π_{VOT} to transform an OT-hybrid SFE protocol into a $\mathcal{P}$-verifiable SFE protocol in the CRS model. Finally, at the end of the current section, we show how to use our $\mathcal{P}$-verifiable SFE protocol to implement any sampling functionality $\mathcal{F}_{\mathsf{Corr}}^{\mathcal{D}}$ with identifiable abort making black-box use of $\Pi_{\mathtt{OT}}$.

$\mathcal{P}$-Verifiable OT Let $\Pi_{\mathtt{OT}}$ be a (two-party) protocol which adaptively UC securely realizes $\mathcal{F}_{\mathsf{OT}}$, among parties p_1 and p_2 in the CRS model (e.g., [11, 39]). For $i \in \{1,2\}$ denote by $f_{\Pi_{\mathtt{OT}}}^{i}$ the next message function of p_i defined as follows: let $\textsc{View}_i$ be the view of party p_i at the beginning of round ρ in an execution of $\Pi_{\mathtt{OT}}$;[12] then $f_{\Pi_{\mathtt{OT}}}^{i}(\textsc{View}_i) = m$ is the message which p_i sends in round ρ of protocol $\Pi_{\mathtt{OT}}$, given that his current view is $\textsc{View}_i$ (if ρ is the last round, then, by default, $m = (\mathsf{out}, y)$, where y is p_i's output). Observe that $f_{\Pi_{\mathtt{OT}}}$ is a deterministic function. Without loss of generality, assume that protocol $\Pi_{\mathtt{OT}}$ has a known number of rounds $\mathtt{Rnd}_{\Pi_{\mathtt{OT}}}$, where in each round only one of the parties p_1 and p_2 sends a message (from $\{0,1\}^k$). Let, also, $\mathcal{F}_{\mathsf{OT}}^{\mathcal{P}}$ denote the multi-party extension of $\mathcal{F}_{\mathsf{OT}}$, in which parties other than p_1 and p_2 provide a default input and receive a default output, i.e., $\mathcal{F}_{\mathsf{OT}}^{\mathcal{P}}$ corresponds to the function $f_{\text{OT}}^{\mathcal{P}}((x_0, x_1), b, \lambda, \ldots, \lambda) = (\bot, x_b, \bot, \bot)$. We describe a multi-party $\mathcal{P}$-verifiable protocol Π_{VOT} which securely realizes the functionality$\mathcal{F}_{\mathsf{OT}}^{\mathcal{P}}$.

The protocol Π_{VOT} works as follows: Initially, every party commits to its random tape. Subsequently, the parties execute their $\Pi_{\mathtt{OT}}$ instructions with the following modification: whenever, for $i, j \in \{1,2\}$, p_i is to send a message $m \in \{0,1\}^k$ to p_j, he chooses the first k unused bits from his random tape (denote by K the string resulting by concatenating these bits), broadcast a one-time pad encryption $c = m \oplus K$ of m with key K, and *privately* opens the corresponding commitments towards p_j. If the opening fails then p_j publicly complains and p_i replies by broadcasting K; p_j recovers m by decrypting c. Clearly, the above modification does not affect the security of $\Pi_{\mathtt{OT}}$ (as all keys are chosen using fresh and independent randomness), therefore Π_{VOT} securely realizes $\mathcal{F}_{\mathsf{OT}}^{\mathcal{P}}$. Additionally, the above protocol is $\mathcal{P}$-verifiable: indeed, because the entire transcript is broadcasted, the view of any party contains all information needed to check whether or not the transcript is consistent with any given set of inputs and committed randomness. For simplicity, in the following we state the security in the $\{CRS, \hat{\mathcal{F}}_{\text{COM}}\}$*-hybrid model* i.e., where, in addition to the CRS the protocol can make ideal calls to a (one-to-many) commitment functionality $\hat{\mathcal{F}}_{\text{COM}}$ which behaves exactly as $\mathcal{F}_{\text{COM}}$ but allows both public and private opening of the committed value (see Appendix C for a detailed description of $\hat{\mathcal{F}}_{\text{COM}}$). [13]

The formal description of protocol Π_{VOT} follows. For clarity we assume without loss of generality that each party $p_i \in \mathcal{P}$ has two distinct and independent random tapes $r_{i,1}$ and $r_{i,2}$, where $r_{i,1}$ is used as the random tape for $\Pi_{\mathtt{OT}}$ and $r_{i,2}$ is used only in Π_{VOT} for encrypting the broadcasted messages. We denote by $K \in_{r_{i,2}} \{0,1\}^{\ell}$ the operation of taking K to be the first ℓ unused bits of $r_{i,2}$.

[12] Recall that $\textsc{View}_i$ consists of the inputs and randomness of p_i along with all messages received up to the current round.

[13] We can use any of the CRS-based commitment protocols [6, 7] to instantiate $\hat{\mathcal{F}}_{\text{COM}}$.

Protocol Π_{VOT} $(\mathcal{P}, p_1, p_2, (x_{1,0}, x_{1,1}), b)$

Inputs and randomness: p_1 has input $x_1 = (x_{1,0}, x_{1,1}) \in \{0,1\}^2$ and p_2 has input $x_2 = b \in \{0,1\}$ (every $p_i \in \mathcal{P} \setminus \{1,2\}$ has no input, i.e., $x_i = \lambda$). Each party $p_i \in \mathcal{P}$ has two random coins sequences, denoted by $r_{i,1}$ and $r_{i,2}$. Initially, for each $p_i \in \mathcal{P}$: $\textsc{View}_i^{\Pi_{\mathtt{OT}}} := (x_i, r_{i,1})$.

1. Each party $p_i \in \{p_1, p_2\}$ commits to its random tapes $r_{i,1}$ and $r_{i,2}$.
2. For each round ρ of $\Pi_{\mathtt{OT}}$ the following steps are executed sequentially by each $p_i \in \{p_1, p_2\}$:
   - 2.1. p_i computes $m := f^i_{\Pi_{\mathtt{OT}}}(\textsc{View}_i^{\Pi_{\mathtt{OT}}})$.
   - 2.2. p_i picks $K \in_{r_{i,2}} \{0,1\}^k$ and opens the commitments to the bits of K *privately* towards p_{3-i} who denotes the opened values as $K^{(3-i)}$.
   - 2.3. p_{3-i} broadcast a complain bit b, where $b = 1$ if p_{3-i} received no message in the previous step (i.e., the opening failed) and $b = 1$ otherwise.
   - 2.4. If p_{3-i} broadcasted $b = 1$, then p_i broadcasts K; p_{3-i} adopts the broadcasted value as the value for $K^{(3-i)}$ ($K^{(3-i)} = 0^\ell$ if an invalid value is broadcasted).
   - 2.5. p_i broadcasts $c := m \oplus K$.
   - 2.6. p_{3-i} computes $m^{(3-i)} := c \oplus K^{(3-i)}$ and adds it to his $\Pi_{\mathtt{OT}}$ view, i.e., sets $\textsc{View}_{3-i}^{\Pi_{\mathtt{OT}}} := (\textsc{View}_{3-i}^{\Pi_{\mathtt{OT}}}, m^{(3-i)})$.
3. In the last round of $\Pi_{\mathtt{OT}}$, each $p_i \in \{p_1, p_2\}$ computes $(\mathsf{out}, y_i) := f^i_{\Pi_{\mathtt{OT}}}(\textsc{View}_i^{\Pi_{\mathtt{OT}}})$ and outputs y_i. Every $p_j \in \mathcal{P} \setminus \{p_1, p_2\}$ outputs $\perp$.

The following lemma states the security of protocol Π_{VOT}; we point out that all security statements in the lemma are with respect to an adaptive adversary.

Lemma 10. *Assuming* $\Pi_{\mathtt{OT}}$ *UC securely realizes the two-party 12OT functionality* $\mathcal{F}_{\mathsf{OT}}$ *in the CRS model, the protocol* Π_{VOT} *(defined above) satisfies the following properties: (security)* Π_{VOT} *UC securely realizes the multi-party extension* $\mathcal{F}^{\mathcal{P}}_{\mathsf{OT}}$ *of* $\mathcal{F}_{\mathsf{OT}}$ *(defined above) in the* $\{CRS, \hat{\mathcal{F}}_{\text{COM}}\}$*-hybrid model; (*$\mathcal{P}$*-verifiability)* Π_{VOT} *is* $\mathcal{P}$*-verifiable. Furthermore,* Π_{VOT} *makes black-box use of (the next-message function of)* $\Pi_{\mathtt{OT}}$.

Proof. The fact that Π_{VOT} makes black-box use of (the next-message function of) $\Pi_{\mathtt{OT}}$ follows by inspection of the protocol. We next argue the security and the $\mathcal{P}$-verifiability property, separately:

(security) The correctness of the outputs follows trivially from the security of $\Pi_{\mathtt{OT}}$. We next provide a simulator for any given adversary $\mathcal{A}$ (we denote by $\mathcal{H}$ and $\mathcal{M}$ the sets of honest and corrupted parties, respectively):

As usually, $\mathcal{S}$ gets to emulate the CRS and the functionality $\hat{\mathcal{F}}_{\text{COM}}$ towards $\mathcal{A}$. Similarly to the parties, $\mathcal{S}$ chooses for the players p_1 and p_2 random tapes $\overline{r_{1,2}}$ and $\overline{r_{2,2}}$, respectively, from which the one-time pad keys will be drawn. For any of the parties in $\mathcal{P} \setminus \{p_1, p_2\}$, the simulation of these parties is trivial as they send no message during the protocol. In particular, if $\mathcal{A}$ requests to corrupt any of those parties, $\mathcal{S}$ chooses independent random tapes r_1 and r_2 and hands them to $\mathcal{A}$. For emulating p_1 and p_2, $\mathcal{S}$ uses the simulator $\mathcal{S}_{\text{OT}}$ which is guaranteed to exists by the security of $\Pi_{\mathtt{OT}}$ as follows (any messages intended to $\mathcal{F}_{\mathsf{OT}}$ are sent by $\mathcal{S}$ to $\mathcal{F}^{\mathcal{P}}_{\mathsf{OT}}$): As long as none of the players p_1 and p_2 is corrupted, $\mathcal{S}$ chooses for each round ρ in which p_i is to send a message a new ciphertext $c_{i,\rho} \in_{\overline{r_{i,2}}} \{0,1\}^k$, sends $c_{i,\rho}$ to $\mathcal{A}$ and plays with $\mathcal{S}_{\text{OT}}$ the ρ-th round of the protocol (with both p_1 and p_2 honest). If at some round $\mathcal{A}$ requests to corrupt one of the parties in $\{p_1, p_2\}$ for the first time (wlog assume that this party is p_1), then $\mathcal{S}$ corrupts p_1 receives his input and emulates a corruption request to $\mathcal{S}_{\text{OT}}$ (for p_1) and receives from him randomness $r_{1,1}$. $\mathcal{S}$ hands $r_{1,1}$ and $\overline{r_{1,2}}$ as

p_1's randomness to $\mathcal{A}$. For the remaining rounds of the simulation, while only p_1 is corrupted, $\mathcal{S}$ uses tape $\overline{r_{2,2}}$ for p_2's one-time pad keys, and in each round ρ uses $\mathcal{S}_{\text{OT}}$ to obtain p_2's ρ-round message $m_{2,\rho}$, samples a key $K_{2,\rho} \in_{\overline{r_{2,2}}} \{0,1\}^k$ and emulates towards $\mathcal{A}$ a broadcast of $c_{2,\rho} = m_{2,\rho} \oplus K_{2,\rho}$. If the adversary requests to corrupt also p_2 (say in round ρ_2 of te simulation) then $\mathcal{S}$ corrupts p_2 and emulates a corruption request to $\mathcal{S}_{\text{OT}}$ (for p_2)[14] and receives from him randomness $r_{2,1}$ (note that the (adaptive) UC security of $\Pi_{\mathtt{OT}}$ ensures that this randomness is consistent with the messages that the corrupted p_1 has seen so far and the randomness $r_{1,1}$)). $\mathcal{S}$ hands $r_{2,1}$ and $\overline{r_{2,2}}$ to $\mathcal{A}$ as the random tapes of p_2 and from there on simply forwards $\mathcal{A}$'s messages to $\mathcal{Z}$.

The fact that $\mathcal{S}$ is a good simulator for Π_{VOT} follows from the soundness of $\mathcal{S}_{\text{OT}}$'s simulation for $\Pi_{\mathtt{OT}}$ and is argued as follows: The distribution of the messages seen by $\mathcal{A}$ is indistinguishable from one-time pad encryptions (and corresponding keys) of messages that adversary $\mathcal{A}^{\Pi_{\mathtt{OT}}}$ attacking $\Pi_{\mathtt{OT}}$ would receive in a simulation of $\Pi_{\mathtt{OT}}$ with $\mathcal{S}_{\text{OT}}$. Hence, if $\mathcal{A}$ can distinguish between the Π_{VOT} view and the view produced by $\mathcal{S}$, then he can be used by a $\Pi_{\mathtt{OT}}$ adversary to distinguish between a view of the execution of $\Pi_{\mathtt{OT}}$ and a simulated view from $\mathcal{S}_{\text{OT}}$ contradicting the security of $\Pi_{\mathtt{OT}}$.

($\mathcal{P}$-verifiability) The $\mathcal{P}$-verifiability follows by inspection of the protocol and the fact that our sequential way of sampling the keys from $r_{1,2}$ and $r_{2,2}$ ensures that the random coins of p_1 and p_2 uniquely define which messages should be input to the (deterministic) next message function $f_{\Pi_{\mathtt{OT}}}$ of $\Pi_{\mathtt{OT}}$. Indeed, let $U = ((x_1, \vec{r}_1), (x_2, \vec{r}_2), \ldots, (x_n, \vec{r}_n))$, where for each $j \in [n]$: $\vec{r}_j = (r_{j,1}, r_{j,2})$ is the vector of p_j's committed ramdon tapes. The algorithm $\mathcal{D}$ works as follows: It emulates, round by round, an execution of Π_{VOT} where p_1 and p_2 have inputs/randomness $(x_1, \vec{r}_1)$ and $(x_2, \vec{r}_2)$, respectively, and compares the result with the broadcasted values. Because the second tape of p_i uniquely defines the (plain-text) messages m (via the one-time pad decryption with keys from it) $\mathcal{D}$ can recover all underlying $\Pi_{\mathtt{OT}}$ messages and compare them to an execution of $\Pi_{\mathtt{OT}}$ with these inputs and the first random tapes. As soon as $\mathcal{D}$ finds a round in which, according to the given view from the protocol execution) some p_i did not send the same message as the simulated execution, $\mathcal{D}$ outputs i and halts. Otherwise $\mathcal{D}$ outputs 0. Clearly, as all the messages are broadcasted (either in plaintext or encrypted) and $\vec{U}$ uniquely defines all the messages that should be broadcasted and the encryption keys, if some party p_j does not correctly follow its Π_{VOT} instructions with respect to $\vec{U}$, $\mathcal{D}$ will output $i = j$. □

In the following, we refer to Π_{VOT} as the *$\mathcal{P}$-verifiable OT protocol corresponding to* $\Pi_{\mathtt{OT}}$.

$\mathcal{P}$-verifiable MPC with (non-identifiable) abort The next step is to add verifiability to a given adaptively UC secure OT-hybrid MPC protocol $\Pi^{\mathcal{F}_{\mathsf{OT}}}$. Wlog, we assume that $\Pi^{\mathcal{F}_{\mathsf{OT}}}$ only makes calls to $\mathcal{F}_{\mathsf{OT}}$ and to a broadcast channel. (Indeed, $\mathcal{F}_{\mathsf{OT}}$ can be used to also implement secure bilateral communication as follows: to send message x, the sender inputs (x, x) and the receiver input $b = 1$.)

Denote by $\Pi^{\Pi_{\text{VOT}}}$ the version of $\Pi^{\mathcal{F}_{\mathsf{OT}}}$ which starts off by having every party publicly commit to its random tape and has all calls to $\mathcal{F}_{\mathsf{OT}}$ replaced by invocations of protocol Π_{VOT} instantiated with fresh/independent randomness. More precisely, $\Pi^{\Pi_{\text{VOT}}}$ is derived from $\Pi^{\mathcal{F}_{\mathsf{OT}}}$ as follows:

- Initially every party commits to its random tape using one-to-many commitments.
- All calls to $\mathcal{F}_{\mathsf{OT}}$ (including the ones used as above to implement bilateral communication) are replaced by invocations of protocol Π_{VOT}. (The random coins do not need to be committed again; the above commitments are used in the invocations of Π_{VOT}.)

[14] Note that $\mathcal{F}_{\mathsf{OT}}$ is deterministic hence $\mathcal{S}$ can trivially emulate its internal state when corrupting p_1 and p_2.

- For each party p_i a specific part of p_i's random tape is associated with each invocation of Π_{VOT}. This part is used only in this invocation and nowhere else in the protocol.

The following lemma states the achieved security, where as in Lemma 10 all security statements are with respect to an adaptive adversary. The proof follows from the security of $\Pi^{\mathcal{F}_{\mathsf{OT}}}$ and the security/$\mathcal{P}$-verifiability of Π_{VOT}.

Lemma 11. *Let $\mathcal{F}$ be a UC functionality and $\Pi^{\mathcal{F}_{\mathsf{OT}}}$ be a protocol which unconditionally UC securely realizes $\mathcal{F}$ in the $\mathcal{F}_{\mathsf{OT}}$-hybrid model with (non-identifiable) abort, and for a protocol Π_{OT} which UC securely realizes $\mathcal{F}_{\mathsf{OT}}$ in the CRS model, let Π_{VOT} be the corresponding $\mathcal{P}$-verifiable protocol (as in Lemma 10). Then protocol $\Pi^{\Pi_{\mathrm{VOT}}}$, defined above, satisfies the following properties: (security) $\Pi^{\Pi_{\mathrm{VOT}}}$ UC securely realizes $\mathcal{F}$ with (non-identifiable) abort in the $\{CRS, \hat{\mathcal{F}}_{\mathrm{COM}}\}$-hybrid model; ($\mathcal{P}$-verifiability) Protocol $\Pi^{\Pi_{\mathrm{VOT}}}$ is $\mathcal{P}$-verifiable. Furthermore, $\Pi^{\Pi_{\mathrm{VOT}}}$ makes black-box use of (the next-message function of) Π_{OT}.*

Proof (sketch). The fact that Π_{VOT} makes black-box use of Π_{OT} follows from the fact that $\Pi^{\mathcal{F}_{\mathsf{OT}}}$ is information-theoretically secure in the $\mathcal{F}_{\mathsf{OT}}$-hybrid world and the fact that Π_{VOT} makes black-box use of Π_{OT}. We next argue the security and the $\mathcal{P}$-verifiability property, separately:

(security) The security of $\Pi^{\Pi_{\mathrm{VOT}}}$ is argued as follows: Let $\Pi^{\mathcal{F}^{\mathcal{P}}_{\mathsf{OT}}}$ be the protocol which results by replacing, in $\Pi^{\mathcal{F}_{\mathsf{OT}}}$, calls to $\mathcal{F}_{\mathsf{OT}}$ by calls to $\mathcal{F}^{\mathcal{P}}_{\mathsf{OT}}$ (where in each such call every party other than the OT sender and receiver hands $\mathcal{F}^{\mathcal{P}}_{\mathsf{OT}}$ a default input λ). Then it is straight-forward to verify that $\Pi^{\mathcal{F}^{\mathcal{P}}_{\mathsf{OT}}}$ unconditionally UC securely emulates $\Pi^{\mathcal{F}_{\mathsf{OT}}}$. Given this fact, the security of $\Pi^{\Pi_{\mathrm{VOT}}}$ in the $\{CRS, \hat{\mathcal{F}}_{\mathrm{COM}}\}$-hybrid model follows by the security of Π_{VOT} for $\mathcal{F}^{\mathcal{P}}_{\mathsf{OT}}$ in the $\{CRS, \hat{\mathcal{F}}_{\mathrm{COM}}\}$-hybrid model by applying the JUC theorem [8].

($\mathcal{P}$-Verifiability) Let $\vec{U}$ denote a given vector of inputs and randomness and View_i denote the view of any party p_i. $\mathcal{D}$ emulates in a step-by-step manner an execution of $\Pi^{\mathcal{F}_{\mathsf{OT}}}$ using the inputs and randomness implied by $\vec{U}$ (and also the CRS) and compares all the messages against View_i. For each invocation of Π_{VOT} with inputs x_1, x_2 from p_i and b from receiver p_j, $\mathcal{D}$ runs the corresponding detector $\mathcal{D}_{\Pi_{\mathrm{VOT}}}$ on these inputs (and input λ for all parties in $\mathcal{P} \setminus \{p_i, p_j\}$) and randomness as implied by $\vec{U}$.[15] If in some of those invocations $\mathcal{D}_{\Pi_{\mathrm{VOT}}}$ outputs some $i \neq 0$ then $\mathcal{D}$ also output i and halts. If $\mathcal{D}$ completes his step-by-step emulation of $\Pi^{\Pi_{\mathrm{VOT}}}$ without finding any inconsistency it outputs $i = 0$. Clearly, as all the messages are broadcasted, either in plaintext or encrypted, and $\vec{U}$ uniquely defines all the messages that should be broadcasted and the corresponding keys, if some party p_j does not correctly follow its $\Pi^{\Pi_{\mathrm{VOT}}}$ instructions with respect to $\vec{U}$, $\mathcal{D}$ will output $i = j$ (if there is more than one such party, $\mathcal{D}$ will output the one that deviates first). □

The Setup Compiler We next describe the protocol $\Pi^{\mathcal{D}}_{\mathsf{Corr}}$ which securely realizes any given sampling functionality $\mathcal{F}^{\mathcal{D}}_{\mathsf{Corr}}$ (for an efficiently computable distribution $\mathcal{D}$), while making black-box use of a UC secure OT-protocol in the CRS model and ideal calls to $\hat{\mathcal{F}}_{\mathrm{COM}}$. The idea is to, first, have every party commit to its random coins and then invoke $\Pi^{\Pi_{\mathrm{VOT}}}$ to securely realize functionality $\mathcal{F}^{\mathcal{D}}_{\mathsf{Corr}}$ using these coins; if the evaluation aborts, then the parties open their committed randomness and use the detector $\mathcal{D}$ to figure out which party cheated. Because the parties have no inputs, opening their randomness does not violate privacy.

[15] Recall that by construction of the protocol, each invocation of Π_{VOT} uses a unique part of the randomness of each party associated with it.

Unfortunately, the above over-simplistic protocol is not simulatable. Intuitively, the reason is that $\Pi^{\Pi_{\mathrm{VOT}}}$ might abort after the adversary has seen his outputs of $\mathcal{F}^{\mathcal{D}}_{\mathsf{Corr}}$, in which case the simulator needs to come up with random coins for the simulated honest parties which are consistent with the adversary's view. We resolve this by the following technical trick, which ensures that $\mathcal{S}$ needs to invoke $\mathcal{F}^{\mathcal{D}}_{\mathsf{Corr}}$ only if the computation of $\Pi^{\Pi_{\mathrm{VOT}}}$ was successful: instead of directly computing $\mathcal{F}^{\mathcal{D}}_{\mathsf{Corr}}$, we use $\Pi^{\Pi_{\mathrm{VOT}}}$ to realize the functionality $\langle\mathcal{F}^{\mathcal{D}}_{\mathsf{Corr}}\rangle$ which computes an authenticated (by means of i.t. signatures) n-out-of-n secret sharing of the output of $\mathcal{F}^{\mathcal{D}}_{\mathsf{Corr}}$. This sharing is then reconstructed by having every party announce its share. The authenticity of the output sharing ensures that either the reconstruction will succeed or a party that did not announce a properly signed share will be caught, in which case the protocol identifies this party. The detailed description of protocol $\Pi^{\mathcal{D}}_{\mathtt{Corr}}$ is given in the following.

Let $\mathcal{F}^{\mathcal{D}}_{\mathsf{Corr}}$ be a sampling functionality for the distribution $\mathcal{D}$, i.e., $\mathcal{F}^{\mathcal{D}}_{\mathsf{Corr}}$ securely evaluates $f_{\mathcal{D}}(\lambda,\ldots,\lambda) = R := (R_1,\ldots,R_n)$, where $R \in (\{0,1\}^*)^n$ is drawn from the efficiently sampleable distribution $\mathcal{D}$. We first describe the functionality $\langle\mathcal{F}^{\mathcal{D}}_{\mathsf{Corr}}(\mathcal{P})\rangle$ which computes an authenticated sharing of the output of $\mathcal{F}^{\mathcal{D}}_{\mathsf{Corr}}$ as a public vector.

More concretely, $\langle\mathcal{F}^{\mathcal{D}}_{\mathsf{Corr}}\rangle$ computes $(R_1,\ldots,R_n) = f_{\mathcal{D}}(\lambda,\ldots,\lambda)$ as $\mathcal{F}^{\mathcal{D}}_{\mathsf{Corr}}$ would, but instead of handing a (private) output y_i to each p_i, the functionality does the following: it computes a vector $\vec{y} = (y_1,\ldots,y_n)$, where each $y_i = R_i + K_i$ for a uniformly chosen one-time pad key $K_i \in \{0,1\}^{|R_i|}$. Subsequently, $\langle\mathcal{F}^{\mathcal{D}}_{\mathsf{Corr}}\rangle$ computes an authenticated n-out-of-n sharing (e.g., a sum sharing) $\langle\vec{y}\rangle$ of $\vec{y}$, by choosing $\langle\vec{y}\rangle_1,\ldots,\langle\vec{y}\rangle_n$ uniformly at random such that $\sum_{i=1}^{n}\langle\vec{y}\rangle_i = \vec{y}$; each ith share $\langle\vec{y}'\rangle_i$ is authenticated as follows: $\langle\mathcal{F}^{\mathcal{D}}_{\mathsf{Corr}}\rangle$ generates a fresh key-pair $(\mathtt{sk}_i, (\mathtt{vk}_{i,1},\ldots,\mathtt{vk}_{i,n})$ and computes a corresponding signature $\sigma_i := \mathsf{Sign}(\langle\vec{y}\rangle_i, \mathtt{vk}_i)$. The actual output of each $p_i \in \mathcal{P}$ is then his authenticated share $(\langle\vec{y}\rangle_i, \sigma_i)$ along with his corresponding verification keys $(\mathtt{vk}_{1,i},\ldots,\mathtt{vk}_{n,i})$ and the one-time pad key K_i. Observe that the signing keys are not given to any party.

$\langle\mathcal{F}^{\mathcal{D}}_{\mathsf{Corr}}(\mathcal{P})\rangle$

Inputs and randomness: Every p_i has input $x_i = \lambda$.

1. Sample $(R_1,\ldots,R_n)$ from distribution $\mathcal{D}$.
2. For each $i \in [n]$ choose $K_i \in_R \{0,1\}^{|R_i|}$ uniformly at random and compute $\vec{y} = (y_1,\ldots,y_n) = (R_1+K_1,\ldots,R_n+K_n)$.
3. Choose $\langle\vec{y}\rangle_1,\ldots,\langle\vec{y}\rangle_n \in \{0,1\}^{|R_1|}\times\cdots\times\{0,1\}^{|R_n|}$ uniformly at random such that $\sum_{i=1}^{n}\langle\vec{y}\rangle_i = \vec{y}$
4. For each $i \in [n]$, generate a fresh key-pair $(\mathtt{sk}_i, (\mathtt{vk}_{i,1},\ldots,\mathtt{vk}_{i,n}))$ for signing $\langle\vec{y}\rangle_i$, and compute a signature $\sigma_i := \mathsf{Sign}(\langle\vec{y}\rangle_i, \mathtt{sk}_i)$.
5. Output to each p_i the tuple $((\langle\vec{y}\rangle_i, \sigma_i), ((\mathtt{vk}_{1,i},\ldots,\mathtt{vk}_{n,i}), K_i)$.

We are now ready to describe the protocol $\Pi^{\mathcal{D}}_{\mathtt{Corr}}(\mathcal{P})$.

Protocol $\Pi^{\mathcal{D}}_{\texttt{Corr}}$ $(\mathcal{P})$

Inputs: Every p_i has input $x_i = \lambda$ and random coins denoted by r_i.

1. Every $p_i \in \mathcal{P}$ commits to his entire random-tape r_i (using one-to-many commitment).
2. The parties invoke protocol $\Pi^{\Pi_{\mathrm{VOT}}}$ to compute the functionality $\langle \mathcal{F}^{\mathcal{D}}_{\mathsf{Corr}}(\mathcal{P}) \rangle$ using their random tapes $(r_1, \ldots, r_n)$.
3. If $\Pi^{\Pi_{\mathrm{VOT}}}$ aborts, then the parties execute the following steps to identify a corrupted party:
   - 3.1. Every $p_i \in \mathcal{P}$ publicly opens his commitment to r_i; if the opening fails, then the protocol aborts with p_i (if there is more than one such p_i's take that one with the smallest index); otherwise denote by $(r'_1, \ldots, r'_n)$ the announces strings.
   - 3.2. The parties apply the detector $\mathcal{D}$ that is guaranteed to exists from the Π-identifiability of $\Pi^{\Pi_{\mathrm{VOT}}}$ on input the vector $\vec{U} = ((\lambda, r'_1), \ldots, (\lambda, r'_n))$, and abort with p_i, where i denotes the output of $\mathcal{D}$.
4. If $\Pi^{\Pi_{\mathrm{VOT}}}$ did not abort, denote by $((\langle \vec{y} \rangle_i, \sigma_i), (\mathtt{vk}_{1,i}, \ldots, \mathtt{vk}_{n,i}), K_i)$ the output of p_i.
5. The parties execute the following steps to reconstruct the sharing:
   - 5.1. Every $p_i \in \mathcal{P}$ broadcast $(\langle \vec{y} \rangle_i, \sigma_i)$; If the signature is not valid then abort with p_i (if there is more than one such p_i take the one with the smallest index).
   - 5.2. Every party reconstructs $\vec{y} = (y_1, \ldots, y_n)$ by adding the announced shares and outputs $R_i := y_i + K_i$.

Theorem 12. *Assuming $\Pi_{\texttt{OT}}$, Π_{VOT}, and $\Pi^{\Pi_{\mathrm{VOT}}}$ as in Lemma 11, the protocol $\Pi^{\mathcal{D}}_{\mathsf{Corr}}$ securely realizes $\mathcal{F}^{\mathcal{D}}_{\mathsf{Corr}}$ with identifiable abort in the CRS model while making black-box use of $\Pi_{\texttt{OT}}$ and ideal calls to the commitment functionality $\hat{\mathcal{F}}_{\mathrm{COM}}$.*

Proof (sketch). We start by showing correctness: On the one hand, when $\Pi^{\Pi_{\mathrm{VOT}}}$ does not abort, then it follows from the security of $\Pi^{\Pi_{\mathrm{VOT}}}$ that it outputs an authenticated sharing of the output of $\mathcal{F}^{\mathcal{D}}_{\mathsf{Corr}}$. As long as the adversary announces the shares of corrupted parties with their actual signatures, the completeness of our signature scheme ensures that the sharing will be correctly reconstructed. Otherwise, i.e., if some corrupted p_i tries to announce a signature other than σ_i, the unforgeability property of the i.t. signatures ensure that with overwhelming probability all parties will reject it and abort with p_i. One the other hand, when $\Pi^{\Pi_{\mathrm{VOT}}}$ aborts, we argue that the output is the index of a corrupted party: If the opening of some commitment fails then this is trivial. Otherwise, the binding property of the commitment scheme ensures that with overwhelming probability for all $r'_i \in [n] : r'_i = r_i$. Because honest parties follow their protocol correctly with respect to $\vec{U}$, it must be the case that $\mathcal{D}$ outputs either an index of a corrupted party or 0. However, outputting 0 would imply that every $p_i \in \mathcal{P}$ follows his protocol instructions correctly with respect to $\vec{U}$, in which case the security of $\Pi^{\Pi_{\mathrm{VOT}}}$ ensures that it does not abort in Step 3. In the remainder of the proof we describe the simulator and show the soundness of the simulation.

A simulator $\mathcal{S}$ for any adversary $\mathcal{A}$ works as follows (wlog assume that $\mathcal{H} = (p_1, \ldots, p_m)$ ia the set of honest parties): If every party is corrupted then the simulation is trivial. For the remainder of the proof, assume that $|\mathcal{H}| > 0$: For simulating the first step, $\mathcal{S}$ commits to random tapes and emulates broadcasting these commitments to $\mathcal{A}$; $\mathcal{S}$ also receives from $\mathcal{A}$ the commitments of corrupted parties.

For the simulation of Step 2 (i.e., the execution of $\Pi^{\Pi_{\mathrm{VOT}}}$) $\mathcal{S}$ uses the simulator $\mathcal{S}_{\Pi^{\Pi_{\mathrm{VOT}}}}$ to emulate towards $\mathcal{A}$ an execution of the first two steps of the protocol with honest parties $p_1, \ldots, p_m$. Moreover, $\mathcal{S}$ simulates $\langle \mathcal{F}^{\mathcal{D}}_{\mathsf{Corr}} \rangle$ towards $\mathcal{S}_{\Pi^{\Pi_{\mathrm{VOT}}}}$.

We next describe how $\mathcal{S}$ proceeds in each of the following two cases: (1) $\mathcal{S}_{\Pi^{\Pi_{\mathrm{VOT}}}}$ aborts, and (2) $\mathcal{S}_{\Pi^{\Pi_{\mathrm{VOT}}}}$ does not abort.

In **Case (1)** $\mathcal{S}$ does the following: At the point of abort, $\mathcal{S}$ emulates a corruption request to $\mathcal{S}_{\Pi^{\Pi_{\mathrm{VOT}}}}$ for all remaining parties in $\mathcal{P}$ (i.e., $p_1 \ldots, p_m$) with input λ for all of them; as a reply, $\mathcal{S}$ receives from $\mathcal{S}_{\Pi^{\Pi_{\mathrm{VOT}}}}$ random tapes $r_1, \ldots, r_m$ for all honest parties. $\mathcal{S}$ uses the equivocality of the commitments generated in Step 1 and opens them towards the adversary as $r_1, \ldots, r_m$; moreover, $\mathcal{S}$ receives from $\mathcal{A}$ the openings to his own commitments, denote by $r_{m+1}, \ldots, r_n$ (if any of the openings fails then $\mathcal{S}$ aborts with the corresponding party). Finally, $\mathcal{S}$ uses the detector $\mathcal{D}$ to find the index i of a party which did not correctly execute the protocol with respect to the vector $\vec{U} = ((\lambda, r_1,), \ldots, (\lambda, r_n))$ and sends $\mathcal{F}^{\mathcal{D}}_{\mathsf{Corr}}$ the message (abort, p_i). Note that all the messages in $\Pi^{\Pi_{\mathrm{VOT}}}$ are sent over the broadcast channel, and therefore the entire transcript of the protocol appears in the interface between $\mathcal{S}$ and $\mathcal{A}$.

In the following we argue that the distribution of the index i output by $\mathcal{S}$ is computationally indistinguishable from the distribution of the index with which the real protocol aborts. For proving this, we need to prove that the view of $\mathcal{A}$ in the protocol execution is indistinguishable from his view in the simulated execution. To this direction we consider the following hybrid experiments:

- H_1: is the above simulation.
- H_2: is the same as H_1 with the difference that the messages from $\mathcal{S}$'s internal emulation of $\langle\mathcal{F}^{\mathcal{D}}_{\mathsf{Corr}}\rangle$, are replaced by the actual messages in the state of $\langle\mathcal{F}^{\mathcal{D}}_{\mathsf{Corr}}\rangle$ in the protocol execution.
- H_3: the real protocol execution.

We show that $\mathcal{A}$ (seeing his view up to the point of abort) cannot distinguish between (A) H_1 and H_2, and in (B) H_2 and H_3. The indistinguishability of H_1 and H_3 follows then from the standard (triangular inequality) property of indistinguishability. To prove (A), we observe that, as $\langle\mathcal{F}^{\mathcal{D}}_{\mathsf{Corr}}\rangle$ has no input, the internal emulation of $\mathcal{S}$ is distributed identically to an actual call to $\langle\mathcal{F}^{\mathcal{D}}_{\mathsf{Corr}}\rangle$; hence H_1 and H_2 are identically distributed. Similarly, (B) follows from the fact that $\mathcal{S}_{\Pi^{\Pi_{\mathrm{VOT}}}}$ is a good simulator for $\Pi^{\Pi_{\mathrm{VOT}}}$. Indeed, if $\mathcal{A}$ can distinguish between H_2 and H_3 then he can be used by an adversary $\mathcal{A}'$ to attack $\Pi^{\Pi_{\mathrm{VOT}}}$ as follows: $\mathcal{A}'$ runs $\mathcal{A}$ up to the point of abort, then corrupts all parties, receives the random values from the simulator $\mathcal{S}_{\Pi^{\Pi_{\mathrm{VOT}}}}$, hands them to $\mathcal{A}$, and output his output. Clearly, if $\mathcal{A}$ has a distinguishing advantage in distinguishing between H_2 and H_3 then so does the above attacker, contradicting the security of $\Pi^{\Pi_{\mathrm{VOT}}}$.

Case (2) ($\Pi^{\Pi_{\mathrm{VOT}}}$ does not abort): In this case the committed randomness of the parties is no longer revealed. Furthermore, the outputs of the adversary are shares of the public output and therefore provide no information on the actual output. As soon as (in the simulation) all outputs have been distributed, $\mathcal{S}$ queries the functionality $\mathcal{F}^{\mathcal{D}}_{\mathsf{Corr}}$ and receives the actual outputs $R_{m+1}, \ldots, R_n$ of the adversary. $\mathcal{S}$ changes the last $n - m$ components of the share of $p_1 \in \mathcal{H}$ (i.e., the ones corresponding to the adversary's outputs) so that so that together with the remaining shares (i.e., shares of parties in $\mathcal{P} \setminus \{p_1\}$) they reconstruct to $R_{m+1} + \hat{K}_{m+1}, \ldots, R_n + \hat{K}_n$, where $\hat{K}_{m+1}, \ldots, \hat{K}_n$ are the simulated one-time pad keys that $\mathcal{S}$ handed $\mathcal{A}$ as part of his $\Pi^{\Pi_{\mathrm{VOT}}}$ output. Furthermore, the simulator generates a valid signature on p_1's modified shares with the key he used during his emulation of $\mathcal{F}^{\mathcal{D}}_{\mathsf{Corr}}$ during the simulation of $\Pi^{\Pi_{\mathrm{VOT}}}$ in the previous step. Finally, $\mathcal{S}$ emulates towards $\mathcal{A}$ broadcast of the shares of honest parties and the corresponding signatures and receives from $\mathcal{A}$ the ones corresponding to the corrupted parties. If $\mathcal{A}$ broadcasts, on behalf of some p_i, an inconsistent share or an invalid signature, then $\mathcal{S}$ sends $\mathcal{F}^{\mathcal{D}}_{\mathsf{Corr}}$ the message (abort, p_i). Otherwise, $\mathcal{S}$ halts with $\mathcal{A}$'s output.

The indistinguishability of the above simulation follows from the fact that the outputs of $\mathcal{A}$ from $\Pi^{\Pi_{\text{VOT}}}$ are shares from an n-out-of-n sharing of the public output and therefore independent of the actual outputs. Indeed, exactly as in the real protocol, in the above simulation the adversary sees shares of a sharing of a vector where the last $n-m$ components are encryptions of his output with the keys he received from the simulation of $\Pi^{\Pi_{\text{VOT}}}$, whereas the first m components (corresponding to honest parties) are encryptions of values for which $\mathcal{A}$ does not get any information on the keys. □

By combining Theorems 6 and 12 with the universal composition theorem, and instantiating $\Pi^{\Pi_{\text{VOT}}}$ with, e.g., the IPS protocol [32] we obtain the following corollary.

Corollary 13. *There exists a protocol which UC realizes any given functionality with identifiable abort, while making black-box use of a protocol for UC realizing $\mathcal{F}_{\mathsf{OT}}$ and a protocol for UC realizing $\hat{\mathcal{F}}_{\text{COM}}$ in the CRS model.*

The Stand-alone model. The proof of Theorem 12 does not use the equivocality of the commitments. Therefore, assuming an adaptive 12OT protocol and extractable commitments, it can be carried over to the stand-alone setting. Such extractable commitments can be constructed by making a black-box use of a one-way function [38], which in turns can be obtained via a black-box use of OT [27]. Thus, we get the following result for the stand-alone model.

Lemma 14 (Stand-alone)**.** *There exists a protocol which securely realizes any given functionality with identifiable abort in the plain model making black-box use of an adaptively secure OT protocol in the plain model.*

Proof (sketch). First, it is straight-forward to verify that replacing in Π_{VOT} the calls to the UC secure OT protocol by an adaptively secure OT protocol in the plain model, we get an adaptively secure protocol for realizing the stand alone version of $\mathcal{F}_{\mathsf{OT}}^{\mathcal{P}}$, i.e., evaluating the function $f_{OT}^{\mathcal{P}}((x_0,x_1),b,\lambda,\ldots,\lambda) = (\perp,x_b,\perp,\ldots,\perp)$, which is also $\mathcal{P}$-verifiable.[16] In fact both the simulator and the detector $\mathcal{D}$ follow the same strategy as in Lemma 10.

Similarly, by replacing uses to Π_{VOT} with the above stand-alone protocol in $\Pi^{\Pi_{\text{VOT}}}$ (and ensuring that no two instances are run at the same time) we obtain a stand-alone counterpart of $\Pi^{\Pi_{\text{VOT}}}$, which securely implements any given functionality in the stand-alone model making black-box use of the underlying OT protocol. Again, both the simulator and the detector $\mathcal{D}$ follow the same strategy as in Lemma 11, where the fact that no two OT's are run at the same time ensures that we can use the modular composition theorem instead of the UC one.

For the remainder of the proof, we show that the protocol $\Pi_{\texttt{SA-Corr}}^{\mathcal{D}}$ that results by replacing in $\Pi_{\texttt{Corr}}^{\mathcal{D}}$ the protocol $\Pi^{\Pi_{\text{VOT}}}$ and the commitments with with the above stand-alone SFE protocol and stand-alone extractable commitments securely implements the sampling functionality $\mathcal{F}_{\mathsf{Corr}}^{\mathcal{D}}$. Indeed, we observe that the simulator of Theorem 12 does not use the adversary's committed randomness in his simulation unless it has been openly revealed (i.e., properly decommitment). Hence, extractable commitments suffice for the simulation. However, to ensure unanimous abort we need to make sure that the used commitments have public opening (i.e., are one-to-many commitments). Such commitments making black-box use of the underlying one-way function can be easily derived by modifying the construction of Pass and Wee [38] as follows: all the messages

[16] The definition of $\mathcal{P}$-verifiability in the standalone plain setting is analogous with the one in the CRS model with the difference that $\mathcal{D}$ does not take the CRS as input.

are broadcasted and the challenges are computed via a multi-party coin-tossing protocol, e.g. [23]. The proof that the above is a good one-to-many commitment scheme follows easily from the proof from [38].

Finally, we point out that the one-way function in the above commitments can also be constructed using black-box OT [27]. Such instantiation makes $\Pi^{\mathcal{D}}_{\texttt{SA-Corr}}$ a protocol which only makes black-box use to an OT protocol in the plain (standalone) model. Hence, per the modular composition theorem, we can use $\Pi_{\texttt{SA-Corr}}$ to instantiate the sampling functionality in Corollary 8. Since, Corollary 8 is unconditional, the resulting protocol makes black-box use of the underlying OT. $\square$

References

[1] Amos Beimel, Yehuda Lindell, Eran Omri, and Ilan Orlov. $1/p$-Secure multiparty computation without honest majority and the best of both worlds. In Phillip Rogaway, editor, *CRYPTO 2011*, volume 6841 of *LNCS*, pages 277–296. Springer, August 2011.

[2] Michael Ben-Or, Shafi Goldwasser, and Avi Wigderson. Completeness theorems for noncryptographic fault-tolerant distributed computations. In *20th ACM STOC*, pages 1–10. ACM Press, May 1988.

[3] Rikke Bendlin, Ivan Damgård, Claudio Orlandi, and Sarah Zakarias. Semi-homomorphic encryption and multiparty computation. In Kenneth G. Paterson, editor, *EUROCRYPT 2011*, volume 6632 of *LNCS*, pages 169–188. Springer, May 2011.

[4] Ran Canetti. Security and composition of multiparty cryptographic protocols. *Journal of Cryptology*, 13(1):143–202, 2000.

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

[6] Ran Canetti and Marc Fischlin. Universally composable commitments. In Joe Kilian, editor, *CRYPTO 2001*, volume 2139 of *LNCS*, pages 19–40. Springer, August 2001.

[7] Ran Canetti, Yehuda Lindell, Rafail Ostrovsky, and Amit Sahai. Universally composable two-party and multi-party secure computation. In *34th ACM STOC*, pages 494–503. ACM Press, May 2002.

[8] Ran Canetti and Tal Rabin. Universal composition with joint state. In Dan Boneh, editor, *CRYPTO 2003*, volume 2729 of *LNCS*, pages 265–281. Springer, August 2003.

[9] David Chaum, Claude Crépeau, and Ivan Damgård. Multiparty unconditionally secure protocols. In *20th ACM STOC*, pages 11–19. ACM Press, May 1988.

[10] Seung Geol Choi, Dana Dachman-Soled, Tal Malkin, and Hoeteck Wee. Simple, black-box constructions of adaptively secure protocols. In Omer Reingold, editor, *TCC 2009*, volume 5444 of *LNCS*, pages 387–402. Springer, March 2009.

[11] Seung-Geol Choi, Jonathan Katz, Hoeteck Wee, and Hong-Sheng Zhou. Efficient, adaptively secure, and composable oblivious transfer with a single, global crs. In Kaoru Kurosawa and Goichiro Hanaoka, editors, *PKC 2013*, volume 7778 of *LNCS*, pages 73–88. Springer Berlin Heidelberg, 2013.

[12] R Cleve. Limits on the security of coin flips when half the processors are faulty. In *Proceedings of the Eighteenth Annual ACM Symposium on Theory of Computing*, STOC '86, pages 364–369, New York, NY, USA, 1986. ACM.

[13] Ronald Cramer, Ivan Damgård, and Jesper Buus Nielsen. Multiparty computation from threshold homomorphic encryption. In *EUROCRYPT*, pages 280–299, 2001.

[14] Ivan Damgård, Marcel Keller, Enrique Larraia, Christian Miles, and Nigel P. Smart. Implementing AES via an actively/covertly secure dishonest-majority MPC protocol. In Ivan Visconti and Roberto De Prisco, editors, *SCN 12*, volume 7485 of *LNCS*, pages 241–263. Springer, September 2012.

[15] Ivan Damgård, Marcel Keller, Enrique Larraia, Valerio Pastro, Peter Scholl, and Nigel P. Smart. Practical covertly secure MPC for dishonest majority - or: Breaking the SPDZ limits. In Jason Crampton, Sushil Jajodia, and Keith Mayes, editors, *ESORICS 2013: 18th European Symposium on Research in Computer Security*, volume 8134 of *Lecture Notes in Computer Science*, pages 1–18, Egham, UK, Sep 9–13, 2003. Springer, Berlin, Germany.

[16] Ivan Damgård, Valerio Pastro, Nigel P. Smart, and Sarah Zakarias. Multiparty computation from somewhat homomorphic encryption. In Reihaneh Safavi-Naini and Ran Canetti, editors, *CRYPTO 2012*, volume 7417 of *LNCS*, pages 643–662. Springer, August 2012.

[17] Tore Kasper Frederiksen, Thomas Pelle Jakobsen, Jesper Buus Nielsen, Peter Sebastian Nordholt, and Claudio Orlandi. Minilego: Efficient secure two-party computation from general assumptions. In *EUROCRYPT*, pages 537–556, 2013.

[18] Juan A. Garay, Philip D. MacKenzie, Manoj Prabhakaran, and Ke Yang. Resource fairness and composability of cryptographic protocols. In Shai Halevi and Tal Rabin, editors, *TCC 2006*, volume 3876 of *LNCS*, pages 404–428. Springer, March 2006.

[19] Oded Goldreich. *Foundations of Cryptography: Basic Tools*, volume 1. Cambridge University Press, Cambridge, UK, 2001.

[20] Oded Goldreich. *Foundations of Cryptography: Basic Applications*, volume 2. Cambridge University Press, Cambridge, UK, 2004.

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

[22] S. Dov Gordon and Jonathan Katz. Partial fairness in secure two-party computation. In Henri Gilbert, editor, *EUROCRYPT 2010*, volume 6110 of *LNCS*, pages 157–176. Springer, May 2010.

[23] Vipul Goyal, Chen-Kuei Lee, Rafail Ostrovsky, and Ivan Visconti. Constructing non-malleable commitments: A black-box approach. In *FOCS*, pages 51–60. IEEE Computer Society, 2012.

[24] Iftach Haitner, Yuval Ishai, Eyal Kushilevitz, Yehuda Lindell, and Erez Petrank. Black-box constructions of protocols for secure computation. *SIAM J. Comput.*, 40(2):225–266, 2011.

[25] Martin Hirt and Ueli M. Maurer. Player simulation and general adversary structures in perfect multiparty computation. *Journal of Cryptology*, 13(1):31–60, 2000.

[26] Martin Hirt, Ueli M. Maurer, and Vassilis Zikas. MPC vs. SFE: Unconditional and computational security. In Josef Pieprzyk, editor, *ASIACRYPT 2008*, volume 5350 of *LNCS*, pages 1–18. Springer, December 2008.

[27] Russell Impagliazzo and Michael Luby. One-way functions are essential for complexity-based cryptography. In *30th FOCS*, pages 230–235. IEEE Computer Society Press, October / November 1989.

[28] Yuval Ishai, Joe Kilian, Kobbi Nissim, and Erez Petrank. Extending oblivious transfers efficiently. In Dan Boneh, editor, *CRYPTO 2003*, volume 2729 of *LNCS*, pages 145–161. Springer, August 2003.

[29] Yuval Ishai, Eyal Kushilevitz, Yehuda Lindell, and Erez Petrank. On combining privacy with guaranteed output delivery in secure multiparty computation. In Cynthia Dwork, editor, *CRYPTO 2006*, volume 4117 of *LNCS*, pages 483–500. Springer, August 2006.

[30] Yuval Ishai, Eyal Kushilevitz, Rafail Ostrovsky, and Amit Sahai. Zero-knowledge from secure multiparty computation. In David S. Johnson and Uriel Feige, editors, *39th ACM STOC*, pages 21–30. ACM Press, June 2007.

[31] Yuval Ishai, Rafail Ostrovsky, and Hakan Seyalioglu. Identifying cheaters without an honest majority. In Ronald Cramer, editor, *TCC 2012*, volume 7194 of *LNCS*, pages 21–38. Springer, March 2012.

[32] Yuval Ishai, Manoj Prabhakaran, and Amit Sahai. Founding cryptography on oblivious transfer - efficiently. In David Wagner, editor, *CRYPTO 2008*, volume 5157 of *LNCS*, pages 572–591. Springer, August 2008.

[33] Jonathan Katz, Ueli Maurer, Björn Tackmann, and Vassilis Zikas. Universally composable synchronous computation. In *TCC 2013*, volume 7785 of *LNCS*, pages 477–498. Springer Berlin Heidelberg, 2013. Full version available at Cryptology ePrint Archive, Report 2011/310.

[34] Marcel Keller, Peter Scholl, and Nigel P. Smart. An architecture for practical actively secure MPC with dishonest majority. In Ahmad-Reza Sadeghi, Virgil D. Gligor, and Moti Yung, editors, *ACM CCS 13: 20th Conference on Computer and Communications Security*, pages 549–560, Berlin, Germany, Nov 4–8, 2013. ACM Press.

[35] Eyal Kushilevitz, Yehuda Lindell, and Tal Rabin. Information-theoretically secure protocols and security under composition. In Jon M. Kleinberg, editor, *38th ACM STOC*, pages 109–118. ACM Press, May 2006.

[36] Jesper Buus Nielsen, Peter Sebastian Nordholt, Claudio Orlandi, and Sai Sheshank Burra. A new approach to practical active-secure two-party computation. In *CRYPTO*, pages 681–700, 2012.

[37] Shien Jin Ong, David C. Parkes, Alon Rosen, and Salil P. Vadhan. Fairness with an honest minority and a rational majority. In Omer Reingold, editor, *TCC 2009*, volume 5444 of *LNCS*, pages 36–53. Springer, March 2009.

[38] Rafael Pass and Hoeteck Wee. Black-box constructions of two-party protocols from one-way functions. In Omer Reingold, editor, *TCC 2009*, volume 5444 of *LNCS*, pages 403–418. Springer, March 2009.

[39] Chris Peikert, Vinod Vaikuntanathan, and Brent Waters. A framework for efficient and composable oblivious transfer. In David Wagner, editor, *CRYPTO 2008*, volume 5157 of *LNCS*, pages 554–571. Springer, August 2008.

[40] Birgit Pfitzmann and Michael Waidner. Unconditional byzantine agreement for any number of faulty processors. In Alain Finkel and Matthias Jantzen, editors, *STACS 92*, volume 577 of *LNCS*, pages 337–350. Springer Berlin Heidelberg, 1992.

[41] Michael O. Rabin. How to exchange secrets with oblivious transfer. Technical Report TR-81, Aiken Computation Lab, Harvard University, 1981. Online version available at `http://eprint.iacr.org/2005/187`.

[42] Tal Rabin and Michael Ben-Or. Verifiable secret sharing and multiparty protocols with honest majority. In *21st ACM STOC*, pages 73–85. ACM Press, May 1989.

[43] Takenobu Seito, Tadashi Aikawa, Junji Shikata, and Tsutomu Matsumoto. Information-theoretically secure key-insulated multireceiver authentication codes. In Daniel J. Bernstein and Tanja Lange, editors, *AFRICACRYPT 10*, volume 6055 of *LNCS*, pages 148–165. Springer, May 2010.

[44] Hakan Seyalioglu. *Reducing Trust When Trust is Essential.* PhD thesis, UCLA, 2012.

[45] Junji Shikata, Goichiro Hanaoka, Yuliang Zheng, and Hideki Imai. Security notions for unconditionally secure signature schemes. In Lars R. Knudsen, editor, *EUROCRYPT 2002*, volume 2332 of *LNCS*, pages 434–449. Springer, April / May 2002.

[46] Colleen Swanson and Douglas R. Stinson. Unconditionally secure signature schemes revisited. In Serge Fehr, editor, *ICITS*, volume 6673 of *Lecture Notes in Computer Science*, pages 100–116. Springer, 2011.

[47] Andrew C. Yao. Protocols for secure computations. In *23rd FOCS*, pages 160–164. IEEE Computer Society Press, November 1982.

[48] Vassilis Zikas, Sarah Hauser, and Ueli M. Maurer. Realistic failures in secure multi-party computation. In Omer Reingold, editor, *TCC 2009*, volume 5444 of *LNCS*, pages 274–293. Springer, March 2009.

A The model (Cont'd)

In this section we give complementary material to Section 2.

A.1 Overview of the UC Security Definition

We provide a high-level description of the UC security definition which should be sufficient for understanding our results, and we refer interested readers to [5] for a thorough and detailed definition.

In a nutshell, security of protocols is argued via the ideal-world/real-world paradigm. In the real-world the players execute the protocol over some communication resource, usually authenticated channels or a broadcast channel. In the ideal-world a specification of the task we want the protocol to implement is described in terms of code of a trusted functionality $\mathcal{F}$, where: the player sends their input(s) to $\mathcal{F}$; $\mathcal{F}$ runs its program on the received inputs (while running the program, $\mathcal{F}$ might receive additional inputs from the players or the adversary or send values to the adversary), and returns to the players their specified outputs. The specification of $\mathcal{F}$ is such that this ideal-evaluation captures, as tightly as possible, the goals of the designed protocol.

Intuitively, a protocol π securely realizes a functionality $\mathcal{F}$, when the adversary cannot achieve more in the protocol than what she could achieve in an ideal-evaluation of $\mathcal{F}$. This is formalized by requiring that for every real-world adversary $\mathcal{A}$, there exists an ideal-world adversary $\mathcal{S}$, aka the simulator, such that no environment $\mathcal{Z}$ can distinguish between interacting with players running the protocol π in the presence of $\mathcal{A}$ and players simply acting as *dummy* forwarders of inputs/outputs between $\mathcal{Z}$ and $\mathcal{F}$ (in an ideal evaluation of $\mathcal{F}$) in the presence of the simulator $\mathcal{S}$. We refer to [5] for a formal description of the real-world/ideal-world experiment.

The $\mathcal{F}$-hybrid model The power of the simulation-based definition is that it allows arguing about security of protocols in a composable manner. In particular, let π_1 be a protocol which securely realizes a functionality $\mathcal{F}_1$. If we can prove that π_2 securely realizes a functionality $\mathcal{F}_2$ using *ideal-calls* to $\mathcal{F}_1$ (we say that π_1 is an $\mathcal{F}_1$-hybrid protocol) then it follows automatically that the protocol $\pi_2^{\pi_1}$ which results by replacing, in π_2, the calls to $\mathcal{F}_1$ by invocations of π_1 also securely realizes $\mathcal{F}_2$. Therefore we only need to prove the security of π_2 in the so-called *$\mathcal{F}_1$-hybrid* model, where the players run π_2 and are allowed to make ideal-calls to $\mathcal{F}_1$. We point out that, in UC protocols come with their hybrids, e.g., in the above example one does not have to write $\pi_2^{\mathcal{F}_1}$ and can simply write π_2; nevertheless, at times we might want to make the actual hybrid used by a protocol explicit, in which case we write it as in $\pi_2^{\mathcal{F}_1}$.

Secure Implementation as Protocol Emulation In order to unify the terminology, in [5] the notion of protocol emulation was introduced which is a generalization of the realization notion discussed above. More concretely, let π be an $\mathcal{F}$-hybrid protocol. Using the UC notation, for an adversary $\mathcal{A}$ and an environment $\mathcal{Z}$ we denote by $\{\text{EXEC}_{\pi,\mathcal{A},\mathcal{Z}}(k,z)\}_{k\in\mathbb{Z},z\in\{0,1\}^*}$ the random variable ensemble describing the output of $\mathcal{Z}$ in an execution of π with adversary $\mathcal{A}$, input z (for the environment), and security parameter k. (As in [5], for sake of simplicity, whenever it is implied by the context we omit the parameter k and the environment's input z from the above notation, i.e., we write $\text{EXEC}_{\pi,\mathcal{A},\mathcal{Z}}$ instead of $\{\text{EXEC}_{\pi,\mathcal{A},\mathcal{Z}}(k,z)\}_{k\in\mathbb{Z},z\in\{0,1\}^*}$.)

For a protocol ρ (recall that ρ might have its hybrids) *we say that a protocol π (UC) securely emulates protocol ρ* if for every adversary $\mathcal{A}$ attacking π there exists an adversary $\mathcal{S}$ attacking protocol ρ and running in time polynomial in the runtime of $\mathcal{A}$ such that for any environment $\mathcal{Z}$ the ensembles $\text{EXEC}_{\pi,\mathcal{A},\mathcal{Z}}$ and $\text{EXEC}_{\rho,\mathcal{S},\mathcal{Z}}$ are indistinguishable; formally

$$\text{EXEC}_{\pi,\mathcal{A},\mathcal{Z}} \approx \text{EXEC}_{\rho,\mathcal{S},\mathcal{Z}}.$$

In this work we consider both indistinguishability with respect to computationally bounded environments and adversaries, also known as *computational or cryptographic security*, as well as for unbounded adversaries/environments which we refer to as *information-theoretic (i.t.) or statistic security.* For clarity, we shall use $\overset{c}{\approx}$ for computational and $\overset{s}{\approx}$ for statistical indistinguishability.

Having defined protocol emulation, (secure) realization of a functionality $\mathcal{F}$ is equivalent to secure emulation of the *dummy* $\mathcal{F}$-hybrid protocol, i.e., the protocol that simply forwards all its inputs (as received from $\mathcal{Z}$) to $\mathcal{F}$ and relays all messages received from $\mathcal{F}$ to $\mathcal{Z}$ (as outputs).

$\mathcal{P}$-verifiable Information-Theoretic Signatures We recall the definition and construction of information-theoretic signatures [45, 43] but slightly modify the terminology to what we consider to be more intuitive. The signature scheme (in particular the key-generation algorithm) needs to know the total number of verifiers or alternatively the list $\mathcal{P}$ of their identities. Furthermore, as usually for i.t. primitives, the key-length needs to be proportional to the number of times that the key is used. Therefore, the scheme is parameterized by two natural numbers $\ell_{\mathcal{S}}$ and ℓ_V which will be upper bounds on the number of signatures that can be generated and verified, respectively, without violating the security.

A *$\mathcal{P}$-verifiable signature scheme* consists of a triple of randomized algorithms $(\mathsf{Gen}, \mathsf{Sign}, \mathsf{Ver})$, where

1. $\mathsf{Gen}(1^k, n, \ell_{\mathcal{S}}, \ell_V)$ outputs a pair $(\mathtt{sk}, \vec{\mathtt{vk}})$, where $\mathtt{sk} \in \{0,1\}^k$ is a signing key, $\vec{\mathtt{vk}} = (\mathtt{vk}_1, \ldots, \mathtt{vk}_n) \in (\{0,1\}^k)^n$ is a verification key-vector consisting of (private) verification sub-keys and $\ell_{\mathcal{S}}, \ell_V \in \mathbb{N}$
2. Sign on input a message m and the signing-key $\mathtt{sk}$ outputs a signature $\sigma \in \{0,1\}^{poly(k)}$
3. Ver on input the message m, a signature σ and a verification sub-key $\mathtt{vk}_i$ outputs a decision-bit $d \in \{0,1\}$.

Definition 15. A $\mathcal{P}$-verifiable signature scheme $(\mathsf{Gen}, \mathsf{Sign}, \mathsf{Ver})$ is said to be *information-theoretically $(\ell_{\mathcal{S}}, \ell_V)$-secure* if it satisfies the following properties:

- (completeness) A valid signature is accepted from any honest receiver:

$$\Pr[\mathsf{Gen} \to (\mathtt{sk}, (\mathtt{vk}_1, \ldots, \mathtt{vk}_n)); \quad \text{for } i \in [n] : (\mathsf{Ver}(m, \mathsf{Sign}(m, \mathtt{sk}), \mathtt{vk}_i) = 1] = 1$$

- Let $\mathcal{O}^S_{\mathtt{sk}}$ denote a signing oracle (on input m, $\mathcal{O}^S_{\mathtt{sk}}$ outputs $\sigma = \mathsf{Sign}(m, \mathtt{sk})$) and $\mathcal{O}^V_{\vec{\mathtt{vk}}}$ denote a verification oracle (on input (m, σ, i), $\mathcal{O}^V_{\vec{\mathtt{vk}}}$ outputs $\mathsf{Ver}(m, \sigma, \mathtt{vk}_i)$). Also, let $A^{\mathcal{O}^S_{\mathtt{sk}}, \mathcal{O}^V_{\vec{\mathtt{vk}}}}$ denote an adversary[17] that makes at most $\ell_{\mathcal{S}}$ calls to $\mathcal{O}^S_{\mathtt{sk}}$ and at most ℓ_V calls to $\mathcal{O}^V_{\vec{\mathtt{vk}}}$, and gets to see the verification keys indexed by some set $\mathcal{I} \subsetneq [n]$. The following properties hold:
  - (unforgeability) A computationally unbounded adversary cannot generate a valid signature on message m' of his choice, other than the one he queries $\mathcal{O}^S_{\mathtt{sk}}$ on (except with negligible probability). Formally,

$$\Pr\left[\begin{array}{c} \mathsf{Gen} \to (\mathtt{sk}, \vec{\mathtt{vk}});\ \text{for some } \mathcal{I} \subsetneq [n] \text{ chosen by } A^{\mathcal{O}^S_{\mathtt{sk}}, \mathcal{O}^V_{\vec{\mathtt{vk}}}} : \\ \left(A^{\mathcal{O}^S_{\mathtt{sk}}, \mathcal{O}^V_{\vec{\mathtt{vk}}}}(\vec{\mathtt{vk}}|_{\mathcal{I}}) \to (m, \sigma, j)\right) \ \wedge \ (m \text{ was not queried to } \mathcal{O}^S_{\mathtt{sk}}) \ \wedge \\ (j \in [n] \setminus \mathcal{I}) \ \wedge \ \big(\mathsf{Ver}(m, \sigma, \mathtt{vk}_j) = 1\big) \end{array}\right] = \text{negl.}$$

[17] For the purpose of information-theoretic security, the assumed adversary is computationally unbounded.

- (consistency)[18] A computationally unbounded adversary cannot (except with negligible probability) create a signature that is accepted by some (honest) party and rejected by some other even after seeing $\ell_{\mathcal{S}}$ valid signatures and verifying ℓ_V signatures:

$$\Pr\left[\begin{array}{c} \mathsf{Gen} \to (\mathtt{sk}, \vec{\mathtt{vk}}); \text{for some } \mathcal{I} \subsetneq [n] \text{ chosen by} A^{\mathcal{O}^S_{\mathtt{sk}}, \mathcal{O}^V_{\vec{\mathtt{vk}}}}(\mathtt{sk}) : \\ A^{\mathcal{O}^S_{\mathtt{sk}}, \mathcal{O}^V_{\vec{\mathtt{vk}}}}(\mathtt{sk}, \vec{\mathtt{vk}}|_{\mathcal{I}}) \to (m, \sigma) \\ \exists i, j \in [n] \setminus \mathcal{I} \text{ s.t. } \mathsf{Ver}(m, \sigma, \mathtt{vk}_i) \neq \mathsf{Ver}(m, \sigma, \mathtt{vk}_j) \end{array}\right] = \text{negl.}$$

In [45, 46] a signature scheme satisfying the above notion of security was constructed. These signatures have a deterministic signature generation algorithm Sign. In the following (Figure 1) we describe the construction from [45] (as described by [46] but for a single signer). We point out that the keys and signatures in the described scheme are elements of a sufficiently large finite field $\mathbb{F}$ (i.e. $|\mathbb{F}| = O(2^{\text{poly}(k)})$; one can easily derive a scheme for strings of length $\ell = \text{poly}(k)$ by applying an appropriate encoding: e.g., map the ith element of $\mathbb{F}$ to the ith string (in the lexicographic order) and vice versa. We say that a value σ is *a valid signature* on message m (with respect to a given key setup $(sk, \vec{\mathtt{vk}})$), if for every honest p_i : $\mathsf{Ver}(m, \sigma, \mathtt{vk}_i) = 1$.

Key Generation: The algorithm for key generation $\mathsf{Gen}(1^k, n, \ell_{\mathcal{S}})$ is as follows:

1. For $(j, k) \in \{0, \ldots, n-1\} \times \{0, \ldots, \ell_{\mathcal{S}}\}$: choose $a_{ij} \in_R \mathbb{F}$ uniformly at random and set the signing key to be (the description of) the multi-variate polynomial

$$\mathtt{sk} := f(y_1, \ldots, y_{n-1}, x) = \sum_{k=0}^{\ell_{\mathcal{S}}} a_{0,k} x^k + \sum_{j=1}^{n-1} \sum_{k=0}^{\ell_{\mathcal{S}}} a_{j,k} y_j x^k$$

2. For $i \in [n]$, choose vector $\vec{v}_i = (v_{i,1}, \ldots, v_{i,n-1}) \in_R \mathbb{F}^{n-1}$ uniformly at random and set the ith verification key to be

$$\mathtt{vk}_i = (\vec{v}_i, f_{\vec{v}_i}(x)),$$

where $f_{\vec{v}_i}(x) = f(v_{i,1}, \ldots, v_{i,n-1}, x)$.

Signature Generation The algorithm for signing a message $m \in \mathbb{F}$ given the above signing key is (a description of) the following polynomial

$$\mathsf{Sign}(m, \mathtt{sk}) := g(y_1, \ldots, y_{n-1}) := f(y_1, \ldots, y_{n-1}, m)$$

Signature Verification The algorithm for verifying a signature $\sigma = g(y_1, \ldots, y_n)$ on a given message m using the i'th verification key is

$$\mathsf{Ver}(m, \sigma, \mathtt{vk}_i) = \begin{cases} 1, & \text{if } g(\vec{v}_i) = f_{\vec{v}_i}(m) \\ 0, & \text{otherwise} \end{cases}$$

Figure 1: Construction of i.t. signatures [46]

Theorem ([46]). *Assuming $|\mathbb{F}| = \Omega(2^k)$ and $\ell_{\mathcal{S}} = poly(k)$ the above signature scheme (Figure 1) is an information-theoretically $(\ell_{\mathcal{S}}, poly(k))$-secure* $\mathcal{P}$-verifiable signature scheme.

[18] This property is often referred to as transferability.

B The Stand-alone Definition

Because this work is the first to provide a systematic and comprehensive study of security with identifiable abort, in this section we also provide the definition corresponding to the simple *stand-alone* secure function evaluation (SFE) with a static adversary as discussed in [20, 4]. As in UC security, the stand-alone definition is based on an ideal-world/real-world paradigm.

The real world The real word is the same as in the stand-alone security definitions of [20, 4]. Note that there, the protocols are *by default* synchronous and are executed over ideally secure channels. (Hybrid worlds where the parties have access to a broadcast channel and or additional setup assumptions such as correlated randomness are also supported.) Initially the parties $p_1, \ldots, p_n$ are given their inputs $x_1, \ldots, x_n$, respectively (denote $\vec{x} = (x_1, \ldots, x_n)$). The adversary $\mathcal{A}$ is allowed to corrupt parties and learn their input x_i and randomness. Then, a semi-honest adversary allows the corrupted parties to faithfully execute their protocol but gets read-access to p_i's internal state; whereas a malicious adversary gets full control over the corrupted parties. As the protocol is synchronous it proceeds in rounds where every message sent in some round ρ is delivered by the beginning of round $\rho + 1$. At the last round every honest party p_i outputs some string $y_i \in \{0,1\}^*$ which we refer to as p_i's output. For any adversary $\mathcal{A}$ and an honest party p_i denote by $\textsc{Out}_{\pi,\mathcal{A},i}(\vec{x})$ the random variable describing the output of p_i in an execution of protocol π with (inputs) $x_1, \ldots, x_n$ in the presence of $\mathcal{A}$; [19] for corrupted parties we set $\textsc{Out}_{\pi,\mathcal{A},i}(\vec{x}) = \perp$. Denote also by $\textsc{View}_{\pi,\mathcal{A}}(\vec{x})$ the view of the adversary consisting of his input and randomness, along with all the message that $\mathcal{A}$ sees in the protocol execution. Finally, let

$$\textsc{Real}_{\pi,\mathcal{A}}(\vec{x}) = \left(\textsc{Out}_{\pi,\mathcal{A},1}(\vec{x}), \ldots, \textsc{Out}_{\pi,\mathcal{A},n}(\vec{x}), \textsc{View}_{\pi,\mathcal{A}}(\vec{x})\right).$$

The ideal world Next we describe the ideal world experiment corresponding to securely evaluating a multi-party function f with identifiable abort. As in [20, 4] the ideal experiment involves the parties interacting with a trusted party computing f, denoted as $\mathcal{F}^f_{\mathsf{ID\text{-}SFE}}$. In fact, the experiment is identical to the ideal experiment from [20, 4] with the only difference that the simulator $\mathcal{S}$ is allowed to make the experiment abort with the identity of a corrupted party. We point out that the simulator can do so even after seeing the outputs of corrupted parties.

[19] Formally, $\textsc{Out}_{\pi,\mathcal{A},i}(\vec{x})$ is a random variable ensemble parameterized by a security parameter $k \in \mathbb{N}$; furthermore, in order to prove/use (modular) composition one also needs to assume that in addition to its inputs, the parties also have some *auxiliary string*. To avoid overcomplicating our description we abstract away from these details and refer to [4] for a detailed and formal handling.

SFE with Identifiable abort– Ideal Model. *Each $p_i \in \mathcal{P}$ has input x_i. The function to be computed is $f(\cdot)$ and is given to the corresponding trusted party/functionality $\mathcal{F}^f_{\textsf{ID-SFE}}$ as parameter. The simulator $\mathcal{S}$ chooses the parties to corrupt, denote by $\mathcal{I}$ the set of corrupted parties, and gets to see and possibly replace the inputs they hand to the functionality. In the output phase, $\mathcal{S}$ gets to see the outputs of corrupted parties and decide whether or not to abort with the identity of some $p_i, i \in \mathcal{I}$, depending on them.*

1. Every $p_i \in \mathcal{P} \setminus \mathcal{I}$ sends his input to the trusted party. For each $p_i \in \mathcal{I}$, $\mathcal{S}$ after seeing $\vec{x}|_{\mathcal{I}}$ sends some input x'_i to $\mathcal{F}^f_{\textsf{ID-SFE}}$ (if $\mathcal{S}$ sends no value or an invalid value for some $p_i \in \mathcal{I}$ then the functionality takes a default value λ for this input). $\mathcal{F}^f_{\textsf{ID-SFE}}$ denotes the vector of received values by $(x'_1, \ldots, x'_n)$.
2. $\mathcal{F}^f_{\textsf{ID-SFE}}$ computes $f(x'_1, \ldots, x'_n) = (y_1, \ldots, y_n)$ (if f is randomized then $\mathcal{F}^f_{\textsf{ID-SFE}}$ internally generates the necessary random coins).
3. $\mathcal{F}^f_{\textsf{ID-SFE}}$ sends to $\mathcal{S}$ the outputs of corrupted parties. I.e., for each $p_i \in \mathcal{I}$, $\mathcal{F}^f_{\textsf{ID-SFE}}$ sends y_i to $\mathcal{S}$.
4. $\mathcal{S}$ sends $\mathcal{F}^f_{\textsf{ID-SFE}}$ a message $\textsf{abt} \in \{(\textsf{abort}, i), \textsf{ok}\}$. If $\mathcal{S}$ sends $\textsf{abt} = (\textsf{abort}, i)$ where $i \in \mathcal{I}$, then for every (honest) $p_j \in \mathcal{P} \setminus \mathcal{I}$, $\mathcal{F}^f_{\textsf{ID-SFE}}$ updates $y_j := (\textsf{abort}, j)$.
5. For each $p_j \in \mathcal{P} \setminus \mathcal{I} : \mathcal{F}^f_{\textsf{ID-SFE}}$ sends p_j his output y_j (corrupted parties output $\perp$).

As above, for any simulator $\mathcal{S}$ denote by $\textsc{Out}_{f,\mathcal{S},i}(\vec{x})$ the random variable describing the output of p_i in the above ideal experiment and denote also by $\textsc{Out}_{f,\mathcal{S}}(\vec{x})$ the output of $\mathcal{S}$; and, let

$$\textsc{ideal}_{f,\mathcal{S}}(\vec{x}) = (\textsc{Out}_{f,\mathcal{S},1}(\vec{x}), \ldots, \textsc{Out}_{f,\mathcal{S},n}(\vec{x}), \textsc{Out}_{f,\mathcal{S}}(\vec{x}))\,.$$

Definition 16. We say that protocol π *information-theoretically securely evaluates the function* f (in the stand-alone model) if for any input vector $\vec{x}$ and for every adversary $\mathcal{A}$ there exists a simulator $\mathcal{S}$ with running time efficient in the running time of $\mathcal{A}$ such that

$$\textsc{ideal}_{f,\mathcal{S}}(\vec{x}) \overset{s}{\approx} \textsc{Real}_{\pi,\mathcal{A}}(\vec{x}).$$

Similarly, we say that protocol π *computationally securely evaluates the function f with identifiable abort* (in the stand-alone model) if for any input vector $\vec{x}$ and for every adversary $\mathcal{A}$ there exists a simulator $\mathcal{S}$ as above such that

$$\textsc{ideal}_{f,\mathcal{S}}(\vec{x}) \overset{c}{\approx} \textsc{Real}_{\pi,\mathcal{A}}(\vec{x}).$$

As in the UC case, for abort respecting protocols, the modular composition theorem extends directly to the case of security with identifiable abort.

C SFE Using Black-box OT (Cont'd)

Here we include complementary material to Section 5 such as formal definitions, protocol descriptions and detailed security proofs. The appendix follows the structure of the main body of the paper.

For clarity we provide the formal specification of the "dual-opening-mode" one-to-many commitment functionality $\hat{\mathcal{F}}_{\text{COM}}$ which allows for both private and public opening of a committed value.

$\hat{\mathcal{F}}_{\text{COM}}(\mathcal{P})$

Commit Phase: Upon receiving message $(\text{msg\_id}, \textsf{commit}, i, m)$ from party $p_i \in \mathcal{P}$ (or the adversary if p_i is corrupted) where $m \in \{0,1\}^*$ and msg_id is a valid message ID, record the tuple $(\text{msg\_id}, p_i, m)$ and send the message $(\text{msg\_id}, \textsf{receipt}, p_i)$ to every party in $\mathcal{P}$ (and to the adversary). Every future commit message with the same ID msg_id is ignored.

Reveal Phase:

Public Reveal Upon receiving a message $(\text{msg\_id}, \textsf{reveal})$ from party $p_i \in \mathcal{P}$, if a message $(\text{msg\_id}, \textsf{commit}, i, m)$ was previously recorder, then send the message $(\text{msg\_id}, \textsf{reveal}, m)$ to all parties in $\mathcal{P}$ (and to the adversary); otherwise ignore the message.

Private Reveal Upon receiving a message $(\text{msg\_id}, \textsf{reveal}, p_j)$ from party $p_i \in \mathcal{P}$, if a message $(\text{msg\_id}, \textsf{commit}, i, m)$ was previously recorder, then send the message $(\text{msg\_id}, \textsf{reveal}, m)$ to $p_j \in \mathcal{P}$ (and to the adversary); otherwise ignore the message.