On Sufficient Oracles for Secure Computation with Identifiable Abort

Mark Simkin[1]*, Luisa Siniscalchi[1,2], and Sophia Yakoubov[1]**

[1] Aarhus University; {simkin, lsiniscalchi, sophia.yakoubov} @cs.au.dk
[2] Concordium Blockchain Research Center

Abstract. Identifiable abort is the strongest security guarantee that is achievable for secure multi-party computation in the dishonest majority setting. Protocols that achieve this level of security ensure that, in case of an abort, all honest parties agree on the identity of at least one corrupt party who can be held accountable for the abort. It is important to understand what computational primitives must be used to obtain secure computation with identifiable abort. This can be approached by asking which oracles can be used to build perfectly secure computation with identifiable abort. Ishai, Ostrovsky, and Zikas (Crypto 2014) show that an oracle that returns correlated randomness to all n parties is sufficient; however, they leave open the question of whether oracles that return output to fewer than n parties can be used.
In this work, we show that for $t \leq n-2$ corruptions, oracles that return output to $n-1$ parties are sufficient to obtain information-theoretically secure computation with identifiable abort. Using our construction recursively, we see that for $t \leq n-\ell-2$ and $\ell \in \mathcal{O}(1)$, oracles that return output to $n-\ell-1$ parties are sufficient.
For our construction, we introduce a new kind of secret sharing scheme which we call unanimously identifiable secret sharing with public and private shares (UISSwPPS). In a UISSwPPS scheme, each share holder is given a public and a private share. Only the public shares are necessary for reconstruction, and the knowledge of a private share additionally enables the identification of at least one party who provided an incorrect share in case reconstruction fails. The important new property of UISSwPPS is that, even given all the public shares, an adversary should not be able to come up with a different public share that causes reconstruction of an incorrect message, or that avoids the identification of a cheater if reconstruction fails.

Keywords: secure computation, identifiable abort

1 Introduction

In the setting of secure multiparty computation we have n parties, each with their own private input x_i, that would like to compute an arbitrary function $f(\mathsf{x}_1, \ldots, \mathsf{x}_n)$ of their inputs in the presence of an adversary, who may actively corrupt up to t of the parties. In particular, the parties would like to compute the function in a way that prevents the adversary from learning any unnecessary information, i.e. the corrupted parties should learn no more than what they can deduce from their own inputs and outputs. From a correctness point of view, we would ideally like to guarantee that the honest parties always obtain the output no matter what the corrupted parties do, but unfortunately, such strong guarantees are unattainable when $t \geq n/2$ parties are corrupt, as was shown by Cleve [Cle86].

For this reason, protocols tolerating this many corruptions usually aim for the weaker notion of *active security with unanimous abort (UA)*, where the honest parties either all obtain the correct output or all unanimously output abort. The drawback of such protocols, however, is that they do not provide the honest parties with a mechanism for determining *who* caused the abort in a failed execution, thus potentially

* Supported by a DFF Sapere Aude Grant 9064-00068B.
** Supported by the European Research Council (ERC) under the European Unions's Horizon 2020 research and innovation programme under grant agreement No 669255 (MPCPRO).

allowing an adversary to perform a denial-of-service attack on the whole computation by only corrupting a single party. To overcome this issue, Ishai, Ostrovsky, and Zikas [IOZ14] introduced the notion of *active security with identifiable abort (IA)*, which enables the honest parties to always unanimously agree on at least one corrupted party that will be held responsible for an abort.

To eventually construct efficient protocols for either notion, it is important to understand the minimal computationally secure building blocks necessary. Towards this goal, it is convenient to study the task of constructing information-theoretically secure protocols in a world where the parties have access to oracles that compute certain sub-functions correctly and securely on their behalf. In such a world, the question of finding the minimal building blocks reduces to finding the "simplest" oracles. The hope of this approach is that simpler oracles lead to computationally less expensive solutions in an oracle-free world, where the oracles are replaced by computationally secure protocols that often represent the main efficiency bottleneck of the overall protocol.

For secure n-party computation with UA or IA in the presence of an adversary that corrupts less than half of the parties, i.e. $t < n/2$, no oracles are needed [RB89, Bea90].[3] For UA and any $t \geq n/2$, oracles are necessary and oracles that realize two-party oblivious transfer [Rab81] are sufficient [Kil88, CvT95]. In contrast to this, an impossibility result by Ishai, Ostrovsky, and Seyalioglu [IOS12] rules out secure computation with IA from *any* two-party oracle for $t \geq 2n/3$.[4] On the positive side, the authors of [IOZ14] show that an n-party oracle for setting up correlated randomness is sufficient for secure computation with IA for any t. For $t \geq n/2$ and oracles that realize k-party functionalities for $2 < k < n$, very little is known about the feasibility of IA. The only known (upper) bounds are due to Brandt et al. [BMMMQ20], who show that IA with security against t corruptions can be realized from certain $(t+2)$-party oracles, when $n \in \mathcal{O}(\log \lambda / \log \log \lambda)$, where λ is the security parameter. The authors conjecture that analogous results for larger n are not possible unless $P = NP$.

1.1 Our Contribution

In this work, we make the first progress towards constructing n-party protocols with IA for any $n \in \mathsf{poly}(\lambda)$ from k-party oracles for $k < n$. In particular, we show the following theorem.

Theorem 1 (Informal). *Any number of parties n can securely compute any function f in the presence of t corruptions with IA and information-theoretic security, when given access to oracles that compute arbitrary k-party functions with IA for $t \leq n - \ell - 1$ and $k = n - \ell$ for any constant $\ell > 0$.*

Our result refutes the conjecture of Brandt et al. mentioned above. As a technical tool, which may be of independent interest, we introduce the notion of *unanimously identifiable secret sharing with public and private shares* (UISSwPPS), which is inspired by the notion of unanimously identifiable secret sharing (UISS) of Ishai, Ostrovsky, and Seyalioglu [IOS12].

Lastly, we remark that in our work, we only focus on oracles that provide us with IA, since oracles that realize k-party functionalities with UA are of no help. To see this, observe that in our parameter settings every call to an oracle necessarily includes a corrupted party, thus the adversary can guarantee that all those calls abort without the honest parties learning anything.

1.2 Technical Overview

The starting point of our work is a result of Ishai, Ostrovsky, and Zikas [IOZ14], which shows that an n-party oracle with IA for distributing correlated randomness is sufficient for general n-party computation

[3] We assume that parties have access to point-to-point and broadcast channels, and we do not consider those as explicit oracles in this paper.

[4] In addition to their impossibility result, the authors of [IOZ14] also show that *blackbox* access to adaptively-secure two-party oblivious transfer is sufficient for constructing protocols with IA for $t > n/2$. We note that assuming blackbox access to a primitive is a stronger assumption than assuming oracle access, which is the focus of this work.

with IA. An n-party oracle generating correlated randomness takes no private inputs from the parties, computes $(\mathsf{r}_1, \ldots, \mathsf{r}_n)$ using some setup function $\texttt{Setup}$, and returns r_i to party i. To solve the general secure computation problem with IA, we can thus focus on the problem of realizing those oracles specifically from k-party oracles for $k < n$. We will require that the number of corruptions t is at most $k-1$ to ensure that every oracle call includes at least one honest party, which we need for our construction. Let us focus on the case of $k = n-1$ for now, which can then be easily extended to any $k = n-\ell$ for any constant ℓ via recursion.

From a high-level perspective, we proceed to construct functionalities of gradually increasing security and expressiveness starting from a functionality that we have oracle access to as depicted in Figure 1.

$$\begin{tikzcd} \mathcal{F}_{\texttt{Setup}',n-1,x,O} \arrow[r, "\text{Thm. 2}"] & \mathcal{F}_{\texttt{Setup}',n,x} \arrow[r, "\text{Thm. 3}"] & \mathcal{F}_{\texttt{Setup},n} \arrow[r, "\text{Thm. 4 + [IOZ14]}"] & \mathcal{F}_n \end{tikzcd}$$

Fig. 1. High-level overview of our approach. On the very left, we have an $(n-1)$-party functionality $\mathcal{F}_{\texttt{Setup}',n-1,x,O}$, which we have oracle access to. On the very right, we have $\mathcal{F}_n$ for computing arbitrary functions among n parties with IA.

The basic idea of our approach is to pick a party $x \in [n]$ and exclude it from the computation. The remaining $n-1$ parties use their oracle access to compute a function $\mathcal{F}_{\texttt{Setup}',n-1,x,O}$, which uses $\texttt{Setup}$ to generate correlated randomness, provides every party with its output and additionally secret shares the output r_x belonging to party x among the $n-1$ parties. After calling the oracle, all parties send their share of r_x to party x, who reconstructs its correlated randomness. If all parties behave honestly, then everybody receives the correct output. Privacy of the value r_x is guaranteed, since at least one honest party participated in the oracle call.

To make this approach work in the presence of an active adversary, we need to deal with malicious parties sending incorrect shares to party x or that party itself being malicious and falsely claiming that some received share was bad or not received at all. Through the use of an appropriate secret sharing scheme, we ensure that any tampering of the shares is detectable during reconstruction by party x. If tampering is detected, the excluded party x proceeds to a complain phase, which *does not* unanimously identify a malicious party, but establishes conflicts between the n parties participating in the computation. After establishing those conflicts, the parties again try to use oracle $\mathcal{F}_{\texttt{Setup}',n-1,x,O}$ to generate correlated randomness. The new oracle invocation will also get a set O as input, which contains the (publicly known) indices of parties that party x has a conflict with. Parties in the set O will not receive a share of the output of party x.

To ensure that our protocol can establish "good" conflicts during the complain phase, we rely on our new secret sharing notion of UISSwPPS. In a nutshell, this secret sharing scheme provides every participant with a public and a private share. The public shares are used for reconstructing the secret and allow the excluded party to detect if some share is malformed. The private shares allow honest share holders to agree on a set of public shares they believe to be malformed; even if the adversary outputs its public shares after seeing *all* other public shares.

Now if $\mathcal{F}_{\texttt{Setup}',n-1,x,O}$ aborts too many times, then the parties decide to switch to a different excluded party and start over. All those executions corresponding to one excluded party x realize a functionality $\mathcal{F}_{\texttt{Setup}',n,x}$, which does not achieve IA, but a much more relaxed version thereof. Using a combinatorial argument, we show that the honest parties can agree on at least one malicious party, if too many invocations of $\mathcal{F}_{\texttt{Setup}',n,x}$ (for different x) have not produced the output.

The approach outlined above realizes our desired functionality $\mathcal{F}_{\texttt{Setup},n}$ with IA for generating correlated randomness, albeit with a still slightly weaker security notion, where the adversary can chose one of several possible outputs or abort[5]. We prove that such a functionality is secure enough to be used in combination with the approach of Ishai, Ostrovsky, and Zikas [IOZ14] for realizing secure n-party computation with IA of arbitrary functions, i.e. functionality $\mathcal{F}_n$.

[5] Note that in regular security with IA, the adversary gets to see *one* output and then has to decide, whether to accept it or to abort.

1.3 Notation

We write $[n]$ to denote the set $\{1, \ldots, n\}$.

2 Secure Multiparty Computation (MPC) Definitions

We follow the real/ideal world simulation paradigm.

An n-party protocol $\Pi = (P_1, \ldots, P_n)$ is an n-tuple of probabilistic polynomial-time (PPT) interactive Turing machines (ITMs), where each party P_i is initialized with input $\mathsf{x}_i \in \{0,1\}^*$ and random coins $\mathsf{r}_i \in \{0,1\}^*$. We let $\mathcal{A}$ denote a special ITM that represents the adversary and that is initialized with input that contains the identities of the corrupt parties, their respective private inputs, and an auxiliary input. The protocol is executed in rounds (i.e., the protocol is synchronous), where each round consists of the send phase and the receive phase, where parties can respectively send the messages from this round to other parties and receive messages from other parties. In every round parties can communicate either over a broadcast channel or a fully connected point-to-point (P2P) network, where we additionally assume all communication to be private and ideally authenticated.

During the execution of the protocol, the corrupt parties receive arbitrary instructions from the adversary $\mathcal{A}$, while the honest parties faithfully follow the instructions of the protocol. We consider the adversary $\mathcal{A}$ to be rushing, i.e., during every round the adversary can see the messages the honest parties sent before producing messages from corrupt parties.

At the end of the protocol execution, the honest parties produce output, the corrupt parties produce no output, and the adversary outputs an arbitrary function of its view. The view of a party during the execution consists of its input, random coins and the messages it sees during the execution.

Definition 1 (Real-world execution). *Let $\Pi = (P_1, \ldots, P_n)$ be an n-party protocol and let $C \subseteq [n]$, of size at most t, denote the set of indices of the parties corrupted by $\mathcal{A}$. The joint execution of Π under $(\mathcal{A}, C)$ in the real world, on input vector $\mathsf{x} = (\mathsf{x}_1, \ldots, \mathsf{x}_n)$, auxiliary input aux to $\mathcal{A}$ and security parameter λ, denoted $\mathsf{REAL}_{\Pi,C,\mathcal{A}(\mathsf{aux})}(\mathsf{x}, \lambda)$, is defined as the output vector of $P_1, \ldots, P_n$ and $\mathcal{A}(\mathsf{aux})$ resulting from the protocol interaction.*

Definition 2 (Ideal Computation). *Let $f : (\{0,1\}^*)^n \to (\{0,1\}^*)^n$ be an n-party function and let $C \subseteq [n]$, of size at most t, be the set of indices of the corrupt parties. Then, the joint ideal execution of f under $(\mathcal{S}, C)$ on input vector $\mathsf{x} = (\mathsf{x}_1, \ldots, \mathsf{x}_n)$, auxiliary input aux to $\mathcal{S}$ and security parameter λ, denoted $\mathsf{IDEAL}_{f,C,\mathcal{S}(\mathsf{aux})}(\mathsf{x}, \lambda)$, is defined as the output vector of $P_1, \ldots, P_n$ and $\mathcal{S}(\mathsf{aux})$ resulting from the interaction to the ideal functionality $\mathcal{F}$ (Figure 2) with the simulator $\mathcal{S}$ and the honest parties. After interacting with $\mathcal{F}$, the hones parties output the message received from $\mathcal{F}$. The corrupt parties output nothing. The simulator $\mathcal{S}$ outputs an arbitrary function of the initial inputs $\{\mathsf{x}_i\}_{i \in C}$, the messages received by the corrupt parties from the trusted party and its auxiliary input.*

Definition 3. *Let $f : (\{0,1\}^*)^n \to (\{0,1\}^*)^n$ be an n-party function. A protocol Π t-securely computes the function f if for every real-world adversary $\mathcal{A}$ there exists a simulator $\mathcal{S}$ whose running time is polynomial in the running time of $\mathcal{A}$ such that for every $C \subseteq [n]$ of size at most t, it holds that*

$$\left\{\mathsf{REAL}_{\Pi,C,\mathcal{A}(\mathsf{aux})}(\mathsf{x}, \lambda)\right\}_{\mathsf{x} \in (\{0,1\}^*)^n, \lambda \in \mathbb{N}} \equiv \left\{\mathsf{IDEAL}_{f,C,\mathcal{S}(\mathsf{aux})}(\mathsf{x}, \lambda)\right\}_{\mathsf{x} \in (\{0,1\}^*)^n, \lambda \in \mathbb{N}}.$$

3 Unanimously Identifiable Secret Sharing with Public and Private Shares

A secret sharing schemes allows a dealer to split a message into shares such that certain authorized subsets of those shares can be used to reconstruct the message, whereas unauthorized subsets reveal no information about the message whatsoever.

Definition 4 (Secret Sharing Scheme). *A* secret sharing scheme *for message space $\{0,1\}^*$ consists of a probabilistic polynomial-time algorithm* `Share` *and a deterministic polynomial-time algorithm* `LRec` *with the following syntax:*

Functionality $\mathcal{F}_{f,n}$

1. For $i \in [n] \backslash C$ receive x_i from party P_i;
2. For $i \in C$ receive x_i from $\mathcal{S}$;
3. Compute $y = f(\mathsf{x}_1, \ldots, \mathsf{x}_n)$;
4. Send y to $\mathcal{S}$;
5. Receive either $\mathsf{continue}$ or $(\mathsf{abort}, \bar{i})$ (for some $\bar{i} \in C$) from $\mathcal{S}$;
6. If $\mathcal{S}$ sent $\mathsf{continue}$: send y to each party $i \in [n] \backslash C$;
7. If $\mathcal{S}$ sent $(\mathsf{abort}, \bar{i})$: send $(\mathsf{abort}, \bar{i})$ to each party $i \in [n] \backslash C$.

Fig. 2. Functionality $\mathcal{F}_{f,n}$ for secure computation of function f among n parties with identifiable abort.

$\mathtt{Share}(\mathsf{msg}) \rightarrow (\mathsf{s}_1, \ldots, \mathsf{s}_n)$**:** *takes as input a message* $\mathsf{msg} \in \{0,1\}^*$ *and outputs shares* $\mathsf{s}_1, \ldots, \mathsf{s}_n$.

$\mathtt{LRec}(\mathsf{s}_i, \{\mathsf{s}_j\}_{j \in S}) \rightarrow (\mathsf{msg}, \mathsf{L})$**:** *takes as input a share* s_i *and a subset of shares* $\{\mathsf{s}_j\}_{j \in S}$, *where* $i \in S \subset [n]$, *and outputs a reconstructed message in* $\{0,1\}^* \cup \{\bot\}$ *and a set of accusations* $\mathsf{L} \subset [n]$.

Furthermore, $(\mathtt{Share}, \mathtt{LRec})$ *should satisfy* correctness *(Definition 8, with appropriate syntactic modifications and ignoring the requirements on* $\mathtt{Rec}$*, which we do not have in a regular secret sharing scheme) and* privacy *(Definition 9, with appropriate syntactic modifications).*

We introduce the notion of unanimously identifiable secret sharing with public and private shares (UISSwPPS). In such a scheme, each share holder will receive one private and one public share. On an intuitive level, the public shares will correspond to a secret sharing of the message shared by the dealer. The private shares, on the other hand, will be used by the share holders to detect any tampering with public shares. In particular, having additional private shares for each share holder allows us to satisfy a stronger notion of local identifiability, which we define below. We show a construction of UISSwPPS in Section 5.

Definition 5 (Secret Sharing Scheme with Public and Private Shares). *A secret sharing scheme with public and private shares for message space* $\{0,1\}^*$ *consists of a probabilistic polynomial-time algorithm* $\mathtt{Share}$ *and deterministic polynomial-time algorithms* $\mathtt{Rec}$ *and* $\mathtt{LRec}$ *with the following syntax:*

$\mathtt{Share}(\mathsf{msg}) \rightarrow (\mathsf{s}_1^{\mathsf{pub}}, \ldots, \mathsf{s}_n^{\mathsf{pub}}, \mathsf{s}_1^{\mathsf{priv}}, \ldots, \mathsf{s}_n^{\mathsf{priv}})$**:** *takes as input a message* $\mathsf{msg} \in \{0,1\}^*$ *and outputs public shares* $\mathsf{s}_1^{\mathsf{pub}}, \ldots, \mathsf{s}_n^{\mathsf{pub}}$ *and private shares* $\mathsf{s}_1^{\mathsf{priv}}, \ldots, \mathsf{s}_n^{\mathsf{priv}}$.

$\mathtt{Rec}(\{\mathsf{s}_i^{\mathsf{pub}}\}_{i \in S}) \rightarrow \mathsf{msg}/\bot$**:** *takes as input a subset of public shares* $\{\mathsf{s}_i^{\mathsf{pub}}\}_{i \in S}$ *(where* $S \subset [n]$*) and outputs a value in* $\{0,1\}^* \cup \{\bot\}$.

$\mathtt{LRec}(\mathsf{s}_i^{\mathsf{priv}}, \{\mathsf{s}_j^{\mathsf{pub}}\}_{j \in S}) \rightarrow (\mathsf{msg}, \mathsf{L})$**:** *takes as input a private share* $\mathsf{s}_i^{\mathsf{priv}}$ *and a subset of public shares* $\{\mathsf{s}_j^{\mathsf{pub}}\}_{j \in S}$ *(where* $S \subset [n]$*) and outputs a reconstructed message in* $\{0,1\}^* \cup \{\bot\}$ *and a list of accusations* $\mathsf{L} \subset [n]$.

Furthermore, $(\mathtt{Share}, \mathtt{Rec}, \mathtt{LRec})$ *should satisfy* correctness *(Definition 8),* privacy *(Definition 9),* adaptive local identifiability *(Definition 10),* publicly detectable failures *(Definition 11),* consistent failures *(Definition 12) and* predictable failures *(definitions 13 and 14).*

We will use our new secret sharing scheme in combination with a new access structure that effectively corresponds to a threshold access structure with additional observers that hold no information about the dealer's message. Even though these observers are not helpful for reconstructing the message, they will still be able to verify whether other published shares are valid or not.

Definition 6 (Threshold Access Structure). *For an arbitrary but fixed threshold* $t \in [n]$, *the* t*-threshold access structure is defined as* $\mathbb{A}_{n,t} = \{S \subset [n] \mid |S| \geq t\}$.

Definition 7 (Threshold Access Structure with Observers). *For an arbitrary but fixed threshold $t \in [n]$ and set $O \subset \{1,\ldots,n\}$, the t-threshold access structure with observers O is defined as $\mathbb{A}^{O}_{n,t} = \{S \subset \{1,\ldots,n\} \mid |S \setminus O| \geq t\}$.*

Definition 8 (Correctness). *A secret sharing scheme with public and private shares* $(\mathtt{Share}, \mathtt{Rec}, \mathtt{LRec})$ *for access structure $\mathbb{A}$ is* correct *if for any $S \in \mathbb{A}$, for any $i \in S$, for any message* $\mathsf{msg} \in \{0,1\}^*$*, there exists a negligible function* $\mathsf{negl}(\cdot)$ *such that*

$$\Pr\left[\begin{bmatrix}\mathsf{s}^{\mathsf{pub}}_1,\ldots,\mathsf{s}^{\mathsf{pub}}_n\\ \mathsf{s}^{\mathsf{priv}}_1,\ldots,\mathsf{s}^{\mathsf{priv}}_n\end{bmatrix} \leftarrow \mathtt{Share}(\mathsf{msg}), (\overline{\mathsf{msg}}, \bot) \leftarrow \mathtt{LRec}\left(\mathsf{s}^{\mathsf{priv}}_i, \{\mathsf{s}^{\mathsf{pub}}_j\}_{j\in S}\right) : \overline{\mathsf{msg}} = \mathsf{msg}\right] = 1 - \mathsf{negl}(\lambda)$$

and

$$\Pr\left[\begin{bmatrix}\mathsf{s}^{\mathsf{pub}}_1,\ldots,\mathsf{s}^{\mathsf{pub}}_n\\ \mathsf{s}^{\mathsf{priv}}_1,\ldots,\mathsf{s}^{\mathsf{priv}}_n\end{bmatrix} \leftarrow \mathtt{Share}(\mathsf{msg}), \overline{\mathsf{msg}} \leftarrow \mathtt{Rec}\left(\{\mathsf{s}^{\mathsf{pub}}_j\}_{j\in S}\right) : \overline{\mathsf{msg}} = \mathsf{msg}\right] = 1 - \mathsf{negl}(\lambda)$$

where the probability is taken over the random coins of the $\mathtt{Share}$ *algorithm.*

Definition 9 (Privacy). *A secret sharing scheme* $(\mathtt{Share}, \mathtt{LRec})$ *for access structure $\mathbb{A}$ is* private *if for any $S \notin \mathbb{A}$, any two messages* $\mathsf{msg}, \mathsf{msg}' \in \{0,1\}^*$ *with* $|\mathsf{msg}| = |\mathsf{msg}'|$*, any possible vector of shares* $\{(\tilde{\mathsf{s}}^{\mathsf{pub}}_i, \tilde{\mathsf{s}}^{\mathsf{priv}}_i)\}_{i\in S}$*, it holds that*

$$\begin{aligned}&\Pr\left[\begin{bmatrix}\mathsf{s}^{\mathsf{pub}}_1,\ldots,\mathsf{s}^{\mathsf{pub}}_n\\ \mathsf{s}^{\mathsf{priv}}_1,\ldots,\mathsf{s}^{\mathsf{priv}}_n\end{bmatrix} \leftarrow \mathtt{Share}(\mathsf{msg}) : \{(\tilde{\mathsf{s}}^{\mathsf{pub}}_i, \tilde{\mathsf{s}}^{\mathsf{priv}}_i)\}_{i\in S} = \{(\mathsf{s}^{\mathsf{pub}}_i, \mathsf{s}^{\mathsf{priv}}_i)\}_{i\in S}\right]\\ -&\Pr\left[\begin{bmatrix}\mathsf{s}^{\mathsf{pub}}_1,\ldots,\mathsf{s}^{\mathsf{pub}}_n\\ \mathsf{s}^{\mathsf{priv}}_1,\ldots,\mathsf{s}^{\mathsf{priv}}_n\end{bmatrix} \leftarrow \mathtt{Share}(\mathsf{msg}') : \{(\tilde{\mathsf{s}}^{\mathsf{pub}}_i, \tilde{\mathsf{s}}^{\mathsf{priv}}_i)\}_{i\in S} = \{(\mathsf{s}^{\mathsf{pub}}_i, \mathsf{s}^{\mathsf{priv}}_i)\}_{i\in S}\right] \leq \mathsf{negl}(\lambda),\end{aligned}$$

where the probability is taken over the randomness of the secret sharing algorithm.

For our new notion of (adaptive) local identifiability, we consider an adversary that can see *all* public shares before outputting any tampered shares.

Definition 10 (Adaptive Local Identifiability). *Consider the game described in Figure 3. A secret sharing scheme with public and private shares* $(\mathtt{Share}, \mathtt{Rec}, \mathtt{LRec})$ *for access structure $\mathbb{A}$ has* adaptive local identifiability *if for any message* $\mathsf{msg} \in \{0,1\}^*$ *and adversary $\mathcal{A}$, there exists a negligible function* $\mathsf{negl}(\cdot)$ *such that*

$$\Pr[\mathcal{A} \text{ wins } \mathsf{game}_{\mathsf{ali}}(\mathcal{A})] \leq \mathsf{negl}(\lambda)$$

where the probability is taken over the random coins of the $\mathcal{C}$ and $\mathcal{A}$.

Remark 1. We will assume that local reconstruction outputs message $\bot$ whenever the list of accusations is not empty.

We require a UISSwPSS to satisfy a mild notion of error detection for outside parties that receive a set of potentially tampered shares and attempt to reconstruct the secret.

Definition 11 (Publicly Detectable Failures). *Consider the game described in Figure 4. A secret sharing scheme with public and private shares* $(\mathtt{Share}, \mathtt{Rec}, \mathtt{LRec})$ *has* publicly detectable failures *if for any message* $\mathsf{msg} \in \{0,1\}^*$ *and adversary $\mathcal{A}$, there exists a negligible function* $\mathsf{negl}(\cdot)$ *such that*

$$\Pr[\mathcal{A} \text{ wins } \mathsf{game}_{\mathsf{pdf}}(\mathcal{A})] \leq \mathsf{negl}(\lambda)$$

where the probability is taken over the random coins of $\mathtt{Share}$ *and $\mathcal{A}$.*

Finally, we require that $\mathtt{Rec}$ fails whenever $\mathtt{LRec}$ fails.

Game $\mathsf{game}_{\mathsf{ali}}(\mathcal{A})$

$\mathcal{A}$ | $\mathcal{C}$

$$\begin{bmatrix} \mathsf{s}_1^{\mathsf{pub}}, \dots, \mathsf{s}_n^{\mathsf{pub}} & \\ & \mathsf{s}_1^{\mathsf{priv}}, \dots, \mathsf{s}_n^{\mathsf{priv}} \end{bmatrix} \leftarrow \mathtt{Share}(\mathsf{msg})$$

$$\xleftarrow{\{\mathsf{s}_i^{\mathsf{pub}}\}_{i \in [n]}}$$

$$\xrightarrow{C \subset [n]}$$

$$\xleftarrow{\{\mathsf{s}_i^{\mathsf{priv}}\}_{i \in C}}$$

$$\xrightarrow{\{\tilde{\mathsf{s}}_i^{\mathsf{pub}}\}_{i \in C}}$$

Define $\tilde{\mathsf{s}}_i^{\mathsf{pub}} := \mathsf{s}_i^{\mathsf{pub}}$ for all $i \in [n] \backslash C$.

$\mathcal{A}$ wins unless for each $S \in \mathbb{A}$ with $S \not\subset C$, one of the following happens:

(1) $\forall i \in [n] : (\mathsf{msg}, \bot) = \mathtt{LRec}\left(\mathsf{s}_i^{\mathsf{priv}}, \{\tilde{\mathsf{s}}_j^{\mathsf{pub}}\}_{j \in S}\right)$

(2) $\forall i, j \in S \setminus C \ : \emptyset \neq \mathsf{L}_i = \mathsf{L}_j \subset [n]$, where

$(\mathsf{msg}_i, \mathsf{L}_i) \leftarrow \mathtt{LRec}\left(\mathsf{s}_i^{\mathsf{priv}}, \{\tilde{\mathsf{s}}_k^{\mathsf{pub}}\}_{k \in S}\right)$ and

$(\mathsf{msg}_j, \mathsf{L}_j) \leftarrow \mathtt{LRec}\left(\mathsf{s}_j^{\mathsf{priv}}, \{\tilde{\mathsf{s}}_k^{\mathsf{pub}}\}_{k \in S}\right)$

Fig. 3. Security game for adaptive local identifiability.

Game $\mathsf{game}_{\mathsf{pdf}}(\mathcal{A})$

$\mathcal{A}$ | $\mathcal{C}$

$$\xrightarrow{C \subset [n]}$$

$$\begin{bmatrix} \mathsf{s}_1^{\mathsf{pub}}, \dots, \mathsf{s}_n^{\mathsf{pub}} & \\ & \mathsf{s}_1^{\mathsf{priv}}, \dots, \mathsf{s}_n^{\mathsf{priv}} \end{bmatrix} \leftarrow \mathtt{Share}(\mathsf{msg})$$

$$\xleftarrow{\{\mathsf{s}_i^{\mathsf{pub}}\}_{i \in C}}$$

$$\xrightarrow{\{\tilde{\mathsf{s}}_i^{\mathsf{pub}}\}_{i \in C}}$$

Define $\tilde{\mathsf{s}}_i^{\mathsf{pub}} = \mathsf{s}_i^{\mathsf{pub}}$ for all $i \in [n] \backslash C$ and set $\{\tilde{\mathsf{s}}_i^{\mathsf{pub}}\}_{i \in [n]}$.

$\mathcal{A}$ wins if $\exists\ S \in \mathbb{A} : S \cap C \neq \emptyset$ and $\mathtt{Rec}(\{\tilde{\mathsf{s}}_i^{\mathsf{pub}}\}_{i \in S}) \notin \{\mathsf{msg}, \bot\}$

Fig. 4. Security game for publicly detectable failures.

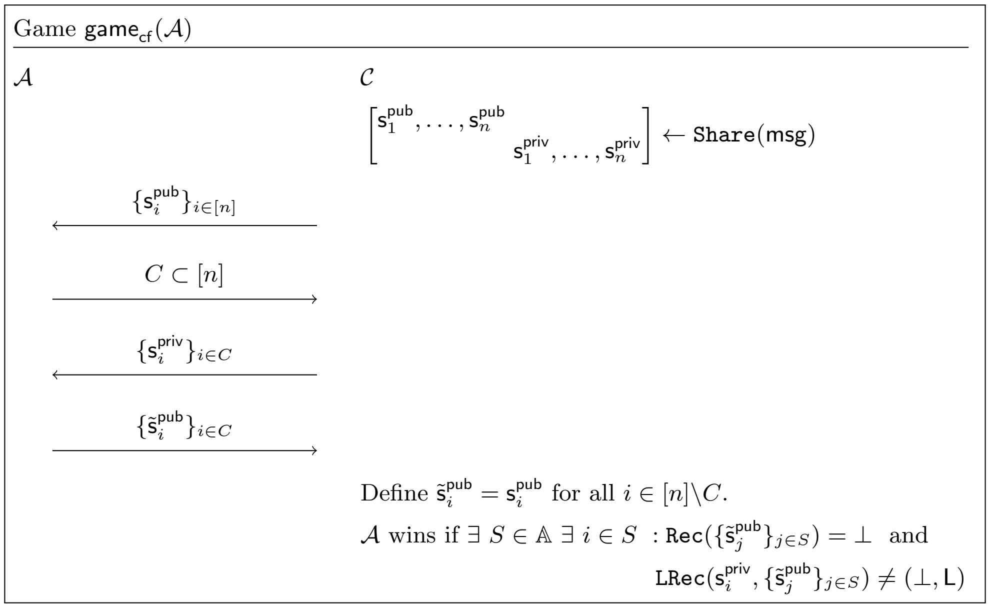
Game $\mathsf{game}_{\mathsf{cf}}(\mathcal{A})$
$\mathcal{A}$
$\mathcal{C}$
$\begin{bmatrix} \mathsf{s}_1^{\mathsf{pub}}, \ldots, \mathsf{s}_n^{\mathsf{pub}} & \\ & \mathsf{s}_1^{\mathsf{priv}}, \ldots, \mathsf{s}_n^{\mathsf{priv}} \end{bmatrix} \leftarrow \mathtt{Share}(\mathsf{msg})$
$\{\mathsf{s}_i^{\mathsf{pub}}\}_{i \in [n]}$
$C \subset [n]$
$\{\mathsf{s}_i^{\mathsf{priv}}\}_{i \in C}$
$\{\tilde{\mathsf{s}}_i^{\mathsf{pub}}\}_{i \in C}$
Define $\tilde{\mathsf{s}}_i^{\mathsf{pub}} = \mathsf{s}_i^{\mathsf{pub}}$ for all $i \in [n] \backslash C$.
$\mathcal{A}$ wins if $\exists\ S \in \mathbb{A}\ \exists\ i \in S\ : \mathtt{Rec}(\{\tilde{\mathsf{s}}_j^{\mathsf{pub}}\}_{j \in S}) = \bot$ and
$\mathtt{LRec}(\mathsf{s}_i^{\mathsf{priv}}, \{\tilde{\mathsf{s}}_j^{\mathsf{pub}}\}_{j \in S}) \neq (\bot, \mathsf{L})$

Fig. 5. Security game for consistent failures.

Definition 12 (Consistent Failures). *Consider the game described in Figure 5. A secret sharing scheme with public and private shares* ($\mathtt{Share}, \mathtt{Rec}, \mathtt{LRec}$) *for access structure* $\mathbb{A}$ *has* consistent failures *if for any message* $\mathsf{msg} \in \{0,1\}^*$ *and adversary* $\mathcal{A}$*, there exists a negligible function* $\mathsf{negl}(\cdot)$ *such that*

$$\Pr[\mathcal{A} \textit{ wins } \mathsf{game}_{\mathsf{cf}}(\mathcal{A})] \leq \mathsf{negl}(\lambda)$$

where the probability is taken over the random coins of $\mathcal{C}$ *and* $\mathcal{A}$*.*

Definition 13 (Predictable Failures with respect to LRec). *Consider the game described in Figure 6. A secret sharing scheme* ($\mathtt{Share}, \mathtt{LRec}$) *for access structure* $\mathbb{A}$ *has* predictable failures *if there exists a probabilistic polynomial-time algorithm* $\mathtt{SLRec}$ *such that for any message* $\mathsf{msg} \in \{0,1\}^*$ *and adversary* $\mathcal{A}$*, there exists a negligible function* $\mathsf{negl}(\cdot)$ *such that*

$$\Pr[\mathcal{A} \textit{ wins } \mathsf{game}_{\mathsf{pflrec}}(\mathcal{A})] \leq \mathsf{negl}(\lambda)$$

where the probability is taken over the random coins of $\mathcal{C}$ *and* $\mathcal{A}$*.*

Definition 14 (Predictable Failures with respect to Rec). *Consider the game described in Figure 7. A secret sharing scheme with public and private shares* ($\mathtt{Share}, \mathtt{Rec}, \mathtt{LRec}$) *for access structure* $\mathbb{A}$ *has* predictable failures with respect to $\mathtt{Rec}$ *if there exists a probabilistic polynomial-time algorithm* $\mathtt{SRec}$ *such that for any message* $\mathsf{msg} \in \{0,1\}^*$ *and adversary* $\mathcal{A}$*, there exists a negligible function* $\mathsf{negl}(\cdot)$ *such that*

$$\Pr[\mathcal{A} \textit{ wins } \mathsf{game}_{\mathsf{pfrec}}(\mathcal{A})] \leq \mathsf{negl}(\lambda)$$

where the probability is taken over the random coins of $\mathcal{C}$ *and* $\mathcal{A}$*.*

4 Bootstrapping MPC With Identifiable Abort

In this section, we describe how to instantiate MPC with identifiable abort for n parties and $t \leq n-2$ given MPC with identifiable abort for $n-1$ parties and $t \leq n-2$. In Section 4.1, we describe the protocol. In Section 4.2, we prove its security.

Game $\mathsf{game}_{\mathsf{pflrec}}(\mathcal{A})$

$\mathcal{A}$ $\qquad\qquad\qquad\qquad$ $\mathcal{C}$

$$\xrightarrow{C \subset [n], C \notin \mathbb{A}}$$

$$\begin{bmatrix} \mathsf{s}_1^{\mathsf{pub}}, \ldots, \mathsf{s}_n^{\mathsf{pub}} & \\ & \mathsf{s}_1^{\mathsf{priv}}, \ldots, \mathsf{s}_n^{\mathsf{priv}} \end{bmatrix} \leftarrow \mathtt{Share}(\mathsf{msg})$$

$$\xleftarrow{\{\mathsf{s}_i^{\mathsf{pub}}\}_{i\in[n]}, \{\mathsf{s}_i^{\mathsf{priv}}\}_{i\in C}}$$

$$\xrightarrow{\{\tilde{\mathsf{s}}_i^{\mathsf{pub}}\}_{i\in C}, S \in \mathbb{A}, S \not\subset C \text{ and } i^* \in S}$$

Define $(b, \mathsf{L}') := \mathtt{SLRec}(C, S, i^*, \{\mathsf{s}_i^{\mathsf{pub}}\}_{i\in[n]}, \{\mathsf{s}_i^{\mathsf{priv}}\}_{i\in C}, \{\tilde{\mathsf{s}}_i^{\mathsf{pub}}\}_{i\in S\cap C})$

Define $\tilde{\mathsf{s}}_i^{\mathsf{pub}} := \mathsf{s}_i^{\mathsf{pub}}$ for all $i \in [n]\backslash C$ and $(\overline{\mathsf{msg}}, \mathsf{L}) := \mathtt{LRec}(\mathsf{s}_{i^*}^{\mathsf{priv}}, \{\tilde{\mathsf{s}}_j^{\mathsf{pub}}\}_{j\in S})$

$\mathcal{A}$ wins unless any one of the following happens:

- $b = 1$ and $\overline{\mathsf{msg}} = \mathsf{msg}$
- $b = 0$ and $\mathsf{L} = \mathsf{L}' \neq \perp$

Fig. 6. Security game for predictable failures with respect to `LRec`.

Game $\mathsf{game}_{\mathsf{pfrec}}(\mathcal{A})$

$\mathcal{A}$ $\qquad\qquad\qquad\qquad$ $\mathcal{C}$

$$\xrightarrow{C \subset [n], C \notin \mathbb{A}}$$

$$\begin{bmatrix} \mathsf{s}_1^{\mathsf{pub}}, \ldots, \mathsf{s}_n^{\mathsf{pub}} & \\ & \mathsf{s}_1^{\mathsf{priv}}, \ldots, \mathsf{s}_n^{\mathsf{priv}} \end{bmatrix} \leftarrow \mathtt{Share}(\mathsf{msg})$$

$$\xleftarrow{\{\mathsf{s}_i^{\mathsf{pub}}\}_{i\in[n]}, \{\mathsf{s}_i^{\mathsf{priv}}\}_{i\in C}}$$

$$\xrightarrow{\{\tilde{\mathsf{s}}_i^{\mathsf{pub}}\}_{i\in C}, S \in \mathbb{A} \text{ and } S \not\subset C}$$

Define $b := \mathtt{SRec}(C, S, \{\mathsf{s}_i^{\mathsf{pub}}\}_{i\in[n]}, \{\mathsf{s}_i^{\mathsf{priv}}\}_{i\in C}, \{\tilde{\mathsf{s}}_i^{\mathsf{pub}}\}_{i\in S\cap C})$

Define $\tilde{\mathsf{s}}_i^{\mathsf{pub}} := \mathsf{s}_i^{\mathsf{pub}}$ for all $i \in [n]\backslash C$ and $\overline{\mathsf{msg}} := \mathtt{Rec}(\{\tilde{\mathsf{s}}_j^{\mathsf{pub}}\}_{j\in S})$

$\mathcal{A}$ wins unless any one of the following happens:

- $b = 1$ and $\overline{\mathsf{msg}} = \mathsf{msg}$
- $b = 0$ and $\overline{\mathsf{msg}} = \perp$

Fig. 7. Security game for predictable failures with respect to `Rec`.

4.1 Protocol

Ishai et al. [IOZ14] show that given correlated randomness, it is possible to securely compute any function with any threshold t, with identifiable abort and with information-theoretic security. Let $\texttt{Setup}() \to (\mathsf{r}_1, \ldots, \mathsf{r}_n)$ be the randomized function that produces the appropriate correlated randomness. $\texttt{Setup}$ takes no inputs (since correlated randomness is independent of the parties' inputs), and outputs n correlated objects, one for each party.

We would like to make use of the availability of MPC with identifiable abort for $n-1$ parties to run $\texttt{Setup}$ (for n parties). In order to do this, we define $\texttt{Setup}'_x$ to be $\texttt{Setup}$ augmented to return shares of r_x to parties $i \in [n] \backslash \{x\}$, and nothing to party x. We then expect parties $i \in [n] \backslash \{x\}$ to send those shares to party x. Of course, we need to make sure that party x won't accept incorrect shares; so, we use UISSwPPS to authenticate the shares.

If party x is dissatisfied with the shares she receives, she broadcasts all the shares. Then, one of two things happens. Either (1) all parties acknowledge that they sent the broadcast shares to party x, in which case, because of the adaptive local identifiability (Definition 10) of the secret sharing, we obtain identifiable abort among the parties who participated in the MPC; or (2) one of the parties (say, party i) claims that party x misrepresented the share she sent, in which case we have established a conflict between parties i and x, and can repeat the MPC excluding party i from the set of parties who hold shares of r_x.

We define $\texttt{Setup}'_{x,O}$ to be the augmented correlated randomness setup function that distributes shares of r_x to parties $i \in [n]$ with observers $O \subseteq [n]$ (where $x \in O$). (We only create an observer share for party x for ease of notation; this share is never used.) $\texttt{Setup}'_{x,O}$ is described in Figure 8.

Algorithm $\texttt{Setup}'_{x,O}$

```
$(\mathsf{r}_1, \ldots, \mathsf{r}_n) \leftarrow \texttt{Setup}()$
$\{\mathsf{s}_i^{\mathsf{pub}}, \mathsf{s}_i^{\mathsf{priv}}\}_{i\in[n]} \leftarrow \texttt{Share}(\mathbb{A}_t^O, \mathsf{r}_x)$
**return** $\{(\mathsf{r}_i, \mathsf{s}_i^{\mathsf{pub}}, \mathsf{s}_i^{\mathsf{priv}})\}_{i\in[n]\backslash\{x\}}$
```

Fig. 8. Algorithm $\texttt{Setup}'_{x,O}$

Figure 9 describes the functionality $\mathcal{F}_{RS(k),\texttt{Setup},n}$ that computes $\texttt{Setup}$ with identifiable abort; the subscript $RS(k)$ signifies that we allow *rejection sampling* by the adversary, who is able to request fresh outputs of $\texttt{Setup}$ up to k times. Figure 13 describes the protocol $\Pi_{RS(k=n^2),\texttt{Setup},n}$ that realizes $\mathcal{F}_{RS(k),\texttt{Setup},n}$ for $k = n^2$. This protocol calls upon a weaker ideal functionality $\mathcal{F}_{RS(k=n),\texttt{Setup},n,x}$, which is described in Figure 10; this ideal functionality only has identifiable abort among $n-1$ of the parties (party x cannot necessarily identify a cheater). $\mathcal{F}_{RS(k=n),\texttt{Setup},n,x}$ either (1) distributes the correlated randomness successfully, (2) identifiably aborts, or (3) identifiably aborts only among $[n]\backslash\{x\}$, in which case $\Pi_{RS(k=n^2),\texttt{Setup},n}$ calls $\mathcal{F}_{RS(k=n),\texttt{Setup},n,x}$ again with a different x. Figure 12 describes the protocol $\Pi_{RS(k=n),\texttt{Setup},n,x}$ that realizes $\mathcal{F}_{RS(k=n),\texttt{Setup},n,x}$. $\Pi_{RS(k=n),\texttt{Setup},n,x}$ in turn calls upon an ideal functionality $\mathcal{F}_{\texttt{Setup}',n,x,O}$; this ideal functionality computes $\texttt{Setup}'$ among $n-1$ parties with identifiable abort (without rejection sampling). We do not give a protocol realizing $\mathcal{F}_{\texttt{Setup}',n,x,O}$, as we assume that secure protocols with identifiable abort exist for any $(n-1)$-party function.

Theorem 2. *Protocol* $\Pi_{RS(k=n),\texttt{Setup},n,x}^{\mathcal{F}_{\texttt{Setup}',n-1,x,O}}$ *(Figure 12) securely realizes the functionality* $\mathcal{F}_{(k=n),\texttt{Setup},n,x}$ *(Figure 10) against* $t \leq n-2$ *corruptions, assuming the availability of a broadcast channel and oracle access to* $\mathcal{F}_{\texttt{Setup}',n-1,x,O}$*.*

```
Functionality $\mathcal{F}_{RS(k),\mathtt{Setup},n}$

1. Let $\ell := 1$;
2. Run $(\mathsf{r}_1^\ell, \dots, \mathsf{r}_n^\ell) \leftarrow \mathtt{Setup}()$;
3. Send $\{\mathsf{r}_i^\ell\}_{i \in C}$ to $\mathcal{S}$;
4. Receive either $\mathsf{retry}$, $\mathsf{continue}$ or $(\mathsf{abort}, \bar{i})$ (for some $\bar{i} \in C$) from $\mathcal{S}$;
5. If $\mathcal{S}$ sent $\mathsf{retry}$: increment $\ell$ by one ($\ell := \ell + 1$). We only allow up to $k$ retries, so we
   never get $\ell > k$. Restart from step 2.
6. If $\mathcal{S}$ sent $\mathsf{continue}$: send $\mathsf{r}_i$ to each party $i \in [n] \backslash C$;
7. If $\mathcal{S}$ sent $(\mathsf{abort}, \bar{i})$: send $(\mathsf{abort}, \bar{i})$ to each party $i \in [n] \backslash C$.
```

Fig. 9. Functionality $\mathcal{F}_{RS(k),\mathtt{Setup},n}$ for secure computation of the correlated randomness setup function $\mathtt{Setup}$ among n parties with identifiable abort, which allows the simulator to reject the output (requesting a redo) at most k times.

```
Functionality $\mathcal{F}_{RS(k),\mathtt{Setup},n,x}$

1. Let $\ell := 1$;
2. Run $(\mathsf{r}_1^\ell, \dots, \mathsf{r}_n^\ell) \leftarrow \mathtt{Setup}()$;
3. Send $\{\mathsf{r}_i^\ell\}_{i \in C}$ to $\mathcal{S}$;
4. Receive either $\mathsf{retry}$, $\mathsf{continue}$ or $(\mathsf{abort}, \bar{i})$ (for some $\bar{i} \in C$) from $\mathcal{S}$;
5. If $\mathcal{S}$ sent $\mathsf{retry}$: increment $\ell$ by one ($\ell := \ell + 1$). We only allow up to $k$ retries, so we
   never get $\ell > k$. Restart from step 2.
6. If $\mathcal{S}$ sent $\mathsf{continue}$: send $\mathsf{r}_i$ to each party $i \in [n] \backslash C$;
7. If $\mathcal{S}$ sent $(\mathsf{abort}, \bar{i})$:
   (a) If $\bar{i} = x$: send $(\mathsf{abort}, x)$ to each party $i \in [n] \backslash C$;
   (b) If $\bar{i} \neq x$:
         i. Send $(\mathsf{gadgabort}, \bar{i})$ to each party $i \in [n] \backslash (\{x\} \cup C)$
        ii. Send $\mathsf{gadgabort}$ to party $x$ if $x \notin C$.
```

Fig. 10. Functionality $\mathcal{F}_{RS(k),\mathtt{Setup},n,x}$ for secure computation of the correlated randomness setup function $\mathtt{Setup}$ among n parties with identifiable abort among all the parties except for x which allows the simulator to reject the output (requesting a redo) at most k times.

```
Functionality $\mathcal{F}_{\mathtt{Setup}',n-1,x,O}$

1. Compute $\{(\mathsf{r}_i, \mathsf{s}_i^{\mathsf{pub}}, \mathsf{s}_i^{\mathsf{priv}})\}_{i \in [n]} \leftarrow \mathtt{Setup}'_{x,O}()$;
2. Send $\{(\mathsf{r}_i, \mathsf{s}_i^{\mathsf{pub}}, \mathsf{s}_i^{\mathsf{priv}})\}_{i \in C}$ to $\mathcal{S}$;
3. Receive either $\mathsf{continue}$ or $(\mathsf{abort}, \bar{i})$ (for some $\bar{i} \in C$) from $\mathcal{S}$;
4. If $\mathcal{S}$ sent $\mathsf{continue}$: send $(\mathsf{r}_i, \mathsf{s}_i^{\mathsf{pub}}, \mathsf{s}_i^{\mathsf{priv}})$ to each party $i \in [n] \backslash (\{x\} \cup C)$;
5. Otherwise: send $(\mathsf{abort}, \bar{i})$ to each party $i \in [n] \backslash (\{x\} \cup C)$.
```

Fig. 11. Functionality $\mathcal{F}_{\mathtt{Setup}',n-1,x,O}$ for secure computation of function $\mathtt{Setup}'_{x,O}$ with identifiable abort among $n-1$ parties (parties $i \in [n] \backslash \{x\}$).

Theorem 3. *Protocol $\Pi^{\mathcal{F}_{(k=n),\mathtt{Setup},n,x}}_{(k=n^2),\mathtt{Setup},n}$ (Figure 13) securely realizes the functionality $\mathcal{F}_{(k=n^2),\mathtt{Setup},n}$ (Figure 9) against $t \leq n-2$ corruptions, assuming the availability of a broadcast channel and oracle access to $\mathcal{F}_{(k=n),\mathtt{Setup},n,x}$.*

Theorem 4. *If a protocol Π securely realizes $\mathcal{F}_{f,n}$ given a single oracle access to $\mathcal{F}_{\mathtt{Setup},n}$ as a first action, then Π also securely realizes $\mathcal{F}_{f,n}$ given a single oracle access to $\mathcal{F}_{RS(k=n^2),\mathtt{Setup},n}$ as a first action (where $\mathcal{F}_{RS(k=n^2),\mathtt{Setup},n}$ replaces $\mathcal{F}_{\mathtt{Setup},n}$).*

Given these three theorems, if we have a protocol $\Pi_{RS(k=n^2),\mathtt{Setup},n}$ realizing $\mathcal{F}_{RS(k=n^2),\mathtt{Setup},n}$, we can use that setup to achieve secure computation with identifiable abort of any function f using the approach of Ishai et al. Since $\Pi_{\mathtt{Setup},n}$ only requires oracle access to $\mathcal{F}_{RS(k=n),\mathtt{Setup},n,x}$ — which in turn can be realized given only oracle access to $\mathcal{F}_{\mathtt{Setup}',n-1,x,O}$ — we can claim the following corollary.

Corollary 5. *For any function f, there exists a protocol that securely realizes the functionality $\mathcal{F}_{f,n}$ (with identifiable abort) against $t \leq n-2$ corruptions, given oracle access to $\mathcal{F}_{\mathtt{Setup}',n-1,x,O}$.*

By using our construction recursively to realize $\mathcal{F}_{\mathtt{Setup}',n-1,x,O}$ given oracle access to a $(n-2)$-party ideal functionality, and so on, we can claim the following corollary.

Corollary 6. *For any function f, for any constant ℓ, there exists a protocol that securely realizes the functionality $\mathcal{F}_{f,n}$ (with identifiable abort) against $t \leq n-\ell-2$ corruptions, given oracle access to $(n-\ell)$-party ideal functionalities (with identifiable abort).*

We require that the recursion depth ℓ be constant because every $(n-l)$-party instance calls at most $n-l$ $(n-l-1)$-party instances, and additionally may require $p'(n-l,\lambda) = p(\lambda)$ work for some polynomials p', p. Thus, we can only guarantee that the protocol is polynomial time if $p^{\ell}(\lambda) \in \mathsf{poly}(\lambda)$, which is only true when ℓ is constant.

Conflict Graphs Before presenting our protocol $\Pi_{\mathtt{Setup},n}$, which requires keeping tack of conflict graphs, we introduce some notation that we use for such graphs. We let S_x be the set of conflicts (denoted as tuples (i,j)) occurring among parties $[n] \setminus \{x\}$. These conflicts result from a call to a functionality with identifiable abort among these $n-1$ parties. Parties i and j are considered to be in conflict if they accuse different parties of aborting the functionality. Since we do not allow a party to accuse itself, this includes the case when one of them accuses the other. For simplicity, we let S^i_x denote the set of parties that party i is in conflict with within S_x.

4.2 Security

Proof of Theorem 2 We prove this theorem by demonstrating a sequence of hybrids. In the first hybrid, a dummy simulator interfaces with a real-world adversary $\mathcal{A}$, executing the protocol $\Pi^{\mathcal{F}_{\mathtt{Setup}',n-1,x,O}}_{RS(k=n),\mathtt{Setup},n,x}$ on behalf of the honest parties exactly as in the real world. In the last hybrid, the simulator $\mathcal{S}_{RS(k=n),\mathtt{Setup},n,x}$ interfaces with the real-world adversary $\mathcal{A}$ and the ideal functionality $\mathcal{F}_{RS(k=n),\mathtt{Setup},n,x}$. We show a number of intermediate hybrids, wherein the simulator "imagines" interfacing with a partially-functional ideal functionality.

H_0 is the real world, where $\mathcal{S}_0$ acts on behalf of the honest parties in the protocol. Note that at this point, $\mathcal{S}_0$ is already responsible for running the ideal functionality $\mathcal{F}_{\mathtt{Setup}',n-1,x,O}$; it does so honestly (by running $\{(\mathsf{r}_i, \mathsf{s}^{\mathsf{pub}}_i, \mathsf{s}^{\mathsf{priv}}_i)\}_{i\in[n]} \leftarrow \mathtt{Setup}'_{x,O}()$).

H_1 is the same as H_0, except that the simulator $\mathcal{S}_1$ starts using a partially-functional ideal functionality $\mathcal{F}'$ in its head. $\mathcal{F}'$ excepts the command retry. In case $\mathcal{F}'$ receives this command she runs $(\mathsf{r}_1,\ldots,\mathsf{r}_n) \leftarrow \mathtt{Setup}()$ and sends $\{\mathsf{r}_i\}_{i\in C}$ to $\mathcal{S}_1$. $\mathcal{S}_1$ uses $(\mathsf{r}_1,\ldots,\mathsf{r}_n)$ to respond to the calls to $\mathcal{F}_{\mathtt{Setup}',n-1,x,O}$ (note that by the construction of the protocol $\Pi^{\mathcal{F}_{\mathtt{Setup}',n-1,x,O}}_{RS(k=n),\mathtt{Setup},n,x}$ the simulator $\mathcal{S}_1$ does not receive more than n calls to $\mathcal{F}_{\mathtt{Setup}',n-1,x,O}$ and therefore she will not sends retry more than n times to $\mathcal{F}'$).
Of course, since $(\mathsf{r}_1,\ldots,\mathsf{r}_n)$ is picked from the same distribution (whether by an honest run of $\mathcal{F}_{RS(k=n),\mathtt{Setup},n,x}$ or by a partially-functional $\mathcal{F}'$), H_1 is indistinguishable from H_0.

Protocol $\Pi^{\mathcal{F}_{\texttt{Setup}',n-1,x,O}}_{RS(k=n),\texttt{Setup},n,x}$

Let $\mathsf{GS} = [n] \setminus \{x\}$. Let $O = \{x\}$. (GS denotes the fixed set of parties calling the ideal functionality; O denotes the set of parties who do not get a share of r_x, which might change over time.)
The parties repeat the three phases described below until one of the following termination conditions occurs:

1. The parties $i \in \mathsf{GS}$ receive a special broadcast message done from party x. When this happens, each party $i \in [n]$ outputs r_i.
2. All of the parties $i \in \mathsf{GS}$ identify party x as a cheater. When this happens, the parties $i \in \mathsf{GS}$ output (abort, x).
3. One of the calls to $\mathcal{F}_{\texttt{Setup}',n-1,x,O}$ results in an (identifiable) abort. When this happens, the parties $i \in \mathsf{GS}$ output $(\mathsf{abort}, \bar{i})$ (where $\bar{i} = \min(\mathsf{L}_i)$ and L_i is the list of parties identified by party i).
4. Though $\mathcal{F}_{\texttt{Setup}',n-1,x,O}$ did not abort, the honest parties among the $n-1$ parties who called the functionality unanimously identify (a) cheater(s). When this happens, the parties $i \in \mathsf{GS}$ output $(\mathsf{abort}, \bar{i})$ (where $\bar{i} = \min(\mathsf{L}_i)$ and L_i is the list of parties identified by party i).

Call the Ideal Functionality:

1. The parties $i \in \mathsf{GS}$ invoke the ideal functionality $\mathcal{F}_{\texttt{Setup}',n-1,x,O}$ to compute $\texttt{Setup}'_{x,O}$, so that each party $i \in \mathsf{GS}$ learns $(\mathsf{r}_i, \mathsf{s}_i^{\mathsf{pub}}, \mathsf{s}_i^{\mathsf{priv}})$.
2. If $\mathcal{F}_{\texttt{Setup}',n-1,x,O}$ aborts, we are in termination condition 3; otherwise the parties proceed to the reconstruct phase.

Reconstruct:

3. Each party $i \in \mathsf{GS} \setminus O$ sends $\mathsf{s}_i^{\mathsf{pub}}$ to party x.
4. Party x runs $\mathsf{r}_x \leftarrow \texttt{Rec}(\{\mathsf{s}_i^{\mathsf{pub}}\}_{i \in \mathsf{GS} \setminus O})$. If $\mathsf{r}_x \neq \perp$, party x broadcasts done. (We are now in termination condition 1; the parties all output r_i.) If $\mathsf{r}_x = \perp$, party x broadcasts $\mathsf{complain}$ and the parties proceed to the complain phase.

Complain:

1. Party x broadcasts the shares it received as $\{\tilde{\mathsf{s}}_i^{\mathsf{pub}}\}_{i \in \mathsf{GS} \setminus O}$.
2. Each party $i \in \mathsf{GS} \setminus O$ broadcasts $\mathsf{s}_i^{\mathsf{pub}}$.
3. If there is an i such that $\tilde{\mathsf{s}}_i^{\mathsf{pub}} \neq \mathsf{s}_i^{\mathsf{pub}}$:
   (a) All parties set $O = O \cup \{i\}_{i \in \mathsf{GS} \text{ s.t. } \tilde{\mathsf{s}}_i^{\mathsf{pub}} \neq \mathsf{s}_i^{\mathsf{pub}}}$.
   (b) If $O = \mathsf{GS}$ (that is, all parties have had conflicting claims with party x), all parties output $(\perp, \{x\})$.
4. Otherwise:
   (a) If $\texttt{Rec}(\{\tilde{\mathsf{s}}_i^{\mathsf{pub}}\}_{i \in [n] \setminus O}) \neq \perp$: all parties $i \in \mathsf{GS}$ identify party x as a cheater. (We are now in termination condition 2; the parties broadcast and output (abort, x).)
   (b) Otherwise: all parties $i \in \mathsf{GS}$ compute $(\perp, \mathsf{L}_i) \leftarrow \texttt{LRec}(\mathsf{s}_i^{\mathsf{priv}}, \{\mathsf{s}_j^{\mathsf{pub}}\}_{j \in \mathsf{GS}})$, and broadcast and output $(\mathsf{abort}, \bar{i})$ where $\bar{i} = \min(\mathsf{L}_i)$. (We are now in termination condition 4.)

Fig. 12. Protocol $\Pi^{\mathcal{F}_{\texttt{Setup}',n-1,x,O}}_{RS(k=n),\texttt{Setup},n,x}$ for secure computation of the correlated randomness setup function $\texttt{Setup}$ among n parties (with identifiable abort among all of the parties except for x, with threshold $t = n-2$) given access to an ideal functionality $\mathcal{F}_{\texttt{Setup}',n-1,x,O}$ that distributes the output of $\texttt{Setup}'$ to $n-1$ parties (with identifiable abort, with threshold $t = n-2$).

Protocol $\Pi_{RS(k=n^2),\mathtt{Setup},n}^{\mathcal{F}_{RS(k=n),\mathtt{Setup},n,x}}$

```
1. For $x \in [n]$: the parties $i \in [n]$ invoke the ideal functionality $\mathcal{F}_{RS(k=n),\mathtt{Setup},n,x}$ to
   compute $\mathtt{Setup}$. One of the following occurs:
   (a) Each party $i \in [n]$ receives and outputs $\mathsf{r}_i$.
   (b) Each party $i \in [n]$ receives and outputs $(\mathsf{abort}, x)$.
   (c) Each party $i \in [n] \setminus \{x\}$ receives $(\mathsf{gadgabort}, \bar{i})$ (for $\bar{i} \neq x$), and party $x$ receives
       $\mathsf{gadgabort}$. In this case, each party $i \in [n] \setminus \{x\}$ broadcasts $\bar{i}$ as $\bar{i}_i$.
       i. A. If there exists a party $i \in [n] \setminus \{x\}$ such that $\bar{i}_i = i$: all parties output
             $(\mathsf{abort}, i)$.
          B. Otherwise: the parties record the obtained conflict graph $S_x$: $(i, j)$ is added
             to $S_x$ if $\bar{i}_i \neq \bar{i}_j$.
2. For $i \in [n]$:
   (a) For $j \in [n]$:
       i. If $S_i^j \cap S_j^i \neq \emptyset$: let $\bar{i} := \min(S_i^j \cap S_j^i)$.
          A. If $i \in S_j^k$ and $j \in S_i^k$: party $k$ outputs $(\mathsf{abort}, \min(\{i, j\}))$.
          B. Otherwise: party $k$ outputs $(\mathsf{abort}, \bar{i})$.
```

Fig. 13. Protocol $\Pi_{RS(k=n^2),\mathtt{Setup},n}$ for secure computation of the correlated randomness setup function $\mathtt{Setup}$ among n parties (with identifiable abort, with threshold $t = n - 2$) given access to an ideal functionality $\mathcal{F}_{RS(k=n),\mathtt{Setup},n,x}$ that distributes the output of $\mathtt{Setup}$ to n parties (with identifiable abort amongall the parties except for x, with threshold $t = n - 2$).

H_2 is the same as H_1, except that the simulator $\mathcal{S}_2$ modifies the partially-functional ideal functionality $\mathcal{F}'$. The partially-functional $\mathcal{F}'$ expects either retry, $\mathsf{continue}$ or $(\mathsf{abort}, \bar{i})$ after $\mathcal{S}_2$ sends $\{\mathsf{r}_i\}_{i \in C}$. It responds to retry exactly as it did in H_1. If it receives $\mathsf{continue}$ or (abort, x), $\mathcal{F}'$ does nothing further; otherwise, it sends (a) $(\mathsf{gadgabort}, \bar{i})$ to each (simulated) honest party $i \in [n] \backslash (\{x\} \cup C)$ and (b) $\mathsf{gadgabort}$ to (simulated) party x if $x \notin C$. Those simulated parties use that output. If the execution reaches step 2, $\mathcal{S}_2$ sends $(\mathsf{abort}, \bar{i})$ to $\mathcal{F}'$, where $(\mathsf{abort}, \bar{i})$ is the output of $\mathcal{F}_{\mathtt{Setup}',n-1,x,O}$.
H_2 is indistinguishable from H_1 since the distributions are identical.

H_3 is the same as H_2, except that, if the execution reaches step 4b, $\mathcal{S}_3$ sends $(\mathsf{abort}, \bar{i})$ to $\mathcal{F}'$, where $\bar{i}$ is computed as $\bar{i} := \min(\mathsf{L}_i)$ for $i = \min(\mathsf{GS} \backslash C)$ and $(\bot, \mathsf{L}_i) \leftarrow \mathsf{LRec}(\mathsf{s}_i^{\mathsf{priv}}, \{\mathsf{s}_j^{\mathsf{pub}}\}_{j \in \mathsf{GS}})$.
We observe that the only way in which H_3 can differ from H_2 is when, in the protocol, termination condition 4 occurs. In H_3, if termination condition 4 occurs, each party $i \in [n] \backslash (\{x\} \cup C)$ outputs $(\mathsf{abort}, \bar{i})$ for the *same* index $\bar{i}$. The same is true in H_2 because (1) in H_3 (in case of a termination condition 4) $\mathtt{Rec}$ outputs $\bot$ and by the consistent failure property of UISSwPPS we can be sure that LRec outputs a list of cheater(s); (2) from the locally identifiable property of UISSwPPS it follows that each party $i \in [n] \backslash (\{x\} \cup C)$ agrees on the same set of cheater(s).

H_4 is the same as H_3, except that the simulator $\mathcal{S}_4$ augments the (simulated) partially-functional $\mathcal{F}'$ so that if it receives (abort, x), it sends (abort, x) to each (simulated) honest party $i \in [n] \backslash C$. Those simulated parties use that output. If the execution reaches step 4a, $\mathcal{S}_4$ sends (abort, x) to $\mathcal{F}'$.
The output of H_4 is distributed identically to that of H_3 because when termination condition 2 occurs, $\mathcal{S}_4$ in H_4 acts in the exact same way as the honest parties in H_3.

H_5 is the same as H_4, except that the simulator $\mathcal{S}_5$ augments the (simulated) partially-functional $\mathcal{F}'$ so that if it receives $\mathsf{continue}$, it sends r_i to each (simulated) honest party $i \in [n] \backslash C$. Those simulated parties use that output. If the execution reaches step 4, $\mathcal{S}_5$ sends $\mathsf{continue}$ to $\mathcal{F}'$.
We observe that the only way in which H_5 can differ from H_4 is in protocol termination condition 1, which is when party x recovers the output r_x. Note that, in H_4, if party $x \notin C$ outputs r_x, then r_x must be the message secret shared during the execution of $\mathtt{Setup}'_{x,O}()$. The above observation follows form the

Simulator $\mathcal{S}_{RS(k=n),\mathtt{Setup},n,x}$

Let $\mathsf{GS} := [n]\backslash\{x\}$, and $O := \{x\}$. $\mathcal{S}_{RS(k=n),\mathtt{Setup},n,x}$ *repeatedly* starts with the "call the ideal functionality" phase until it terminates.

Call the Ideal Functionality: Upon receiving a call to $\mathcal{F}_{\mathtt{Setup}',n-1,x,O}$ from $\mathcal{A}$, $\mathcal{S}_{RS(k=n),\mathtt{Setup},n,x}$ does the following:

1. $\mathcal{S}_{RS(k=n),\mathtt{Setup},n,x}$ obtains $\{\mathsf{r}_i\}_{i\in C}$ from $\mathcal{F}_{RS(k=n),\mathtt{Setup},n,x}$;
   - (a) If $x \in C$: $\mathcal{S}_{RS(k=n),\mathtt{Setup},n,x}$ computes $\{(\mathsf{s}_i^{\mathsf{pub}}, \mathsf{s}_i^{\mathsf{priv}})\}_{i\in[n]} \leftarrow \mathtt{Share}(\mathbb{A}_t^O, \mathsf{r}_x)$;
   - (b) Otherwise (if $x \notin C$): $\mathcal{S}_{\mathtt{Setup},n,x}$ computes $\{(\mathsf{s}_i^{\mathsf{pub}}, \mathsf{s}_i^{\mathsf{priv}})\}_{i\in[n]} \leftarrow \mathtt{Share}(\mathbb{A}_t^O, 0)$.
2. $\mathcal{S}_{RS(k=n),\mathtt{Setup},n,x}$ sends $\{(\mathsf{r}_i, \mathsf{s}_i^{\mathsf{pub}}, \mathsf{s}_i^{\mathsf{priv}})\}_{i\in\mathsf{GS}\cap C}$ (on behalf of $\mathcal{F}_{\mathtt{Setup}',n,x,O}$) to $\mathcal{A}$.
3. If $\mathcal{A}$ sends $(\mathsf{abort}, \bar{i})$: $\mathcal{S}_{RS(k=n),\mathtt{Setup},n,x}$ sends $(\mathsf{abort}, \bar{i})$ to $\mathcal{F}_{RS(k=n),\mathtt{Setup},n,x}$ and terminates returning $\mathcal{A}$'s output.

Reconstruct:

4. If $x \in C$:
   - (a) For each $i \in \mathsf{GS}\backslash(O \cup C)$, $\mathcal{S}_{RS(k=n),\mathtt{Setup},n,x}$ sends $\mathsf{s}_i^{\mathsf{pub}}$ to $\mathcal{A}$ (on behalf of party i).
   - (b) If $\mathcal{A}$ broadcasts done (on behalf of party x): $\mathcal{S}_{RS(k=n),\mathtt{Setup},n,x}$ terminates returning $\mathcal{A}$'s output.
5. Otherwise (if $x \notin C$):
   - (a) $\mathcal{S}_{RS(k=n),\mathtt{Setup},n,x}$ receives $\tilde{\mathsf{s}}_i^{\mathsf{pub}}$ from $\mathcal{A}$ on behalf of each party $i \in \mathsf{GS} \cap C\backslash O$.
   - (b) Let $\tilde{\mathsf{s}}_i^{\mathsf{pub}} := \mathsf{s}_i^{\mathsf{pub}}$ for $i \in \mathsf{GS}\backslash(C \cup O)$. $\mathcal{S}_{RS(k=n),\mathtt{Setup},n,x}$ computes $\mathsf{r}_x \leftarrow \mathtt{Rec}(\{\tilde{\mathsf{s}}_i^{\mathsf{pub}}\}_{i\in\mathsf{GS}\backslash O})$.
     - i. If $\mathsf{r}_x \neq \perp$: $\mathcal{S}_{RS(k=n),\mathtt{Setup},n,x}$ broadcasts done (on behalf of party x) and terminates returning $\mathcal{A}$'s output.
     - ii. Otherwise (if $\mathsf{r}_x = \perp$): $\mathcal{S}_{RS(k=n),\mathtt{Setup},n,x}$ broadcasts $\mathsf{complain}$ (on behalf of party x) and proceeds to the complain phase.

Complain:

1. If $x \in C$: $\mathcal{S}_{RS(k=n),\mathtt{Setup},n,x}$ receives $\{\bar{\mathsf{s}}_i^{\mathsf{pub}}\}_{i\in C\backslash O}$ from $\mathcal{A}$ (on behalf of party x).
2. Otherwise (if $x \notin C$): $\mathcal{S}_{RS(k=n),\mathtt{Setup},n,x}$ broadcasts the shares it received as $\{\tilde{\mathsf{s}}_i^{\mathsf{pub}}\}_{i\in\mathsf{GS}\backslash O}$ (on behalf of party x).
3. For each party $i \in \mathsf{GS}\backslash(O \cup C)$, $\mathcal{S}_{RS(k=n),\mathtt{Setup},n,x}$ sets $\bar{\mathsf{s}}_i^{\mathsf{pub}} := \mathsf{s}_i^{\mathsf{pub}}$ and broadcasts $\bar{\mathsf{s}}_i^{\mathsf{pub}}$.
4. $\mathcal{S}_{RS(k=n),\mathtt{Setup},n,x}$ receives $\{\bar{\mathsf{s}}_i^{\mathsf{pub}}\}_{i\in\mathsf{GS}\cap C\backslash O}$ from $\mathcal{A}$.
5. If there is an i such that $\tilde{\mathsf{s}}_i^{\mathsf{pub}} \neq \bar{\mathsf{s}}_i^{\mathsf{pub}}$: $\mathcal{S}_{RS(k=n),\mathtt{Setup},n,x}$ sets $O = O \cup \{i\}_{i\in\mathsf{GS}\text{ s.t. }\tilde{\mathsf{s}}_i^{\mathsf{pub}}\neq\bar{\mathsf{s}}_i^{\mathsf{pub}}}$.
   - (a) If $O = \mathsf{GS}$: $\mathcal{S}_{RS(k=n),\mathtt{Setup},n,x}$ sends (abort, x) to $\mathcal{F}_{RS(k=n),\mathtt{Setup},n,x}$.
   - (b) Otherwise: $\mathcal{S}_{RS(k=n),\mathtt{Setup},n,x}$ sends retry to $\mathcal{F}_{RS(k=n),\mathtt{Setup},n,x}$, and goes back to step 1a of the "call the ideal functionality" phase.
6. Otherwise:
   - (a) If $\mathtt{Rec}(\{\tilde{\mathsf{s}}_i^{\mathsf{pub}}\}_{i\in[n]\backslash O}) \neq \perp$ (and thus $x \in C$): $\mathcal{S}_{RS(k=n),\mathtt{Setup},n,x}$ sends (abort, x) to $\mathcal{F}_{RS(k=n),\mathtt{Setup},n,x}$, and broadcasts (abort, x) (on behalf of each party $i \in \mathsf{GS}\backslash C$). $\mathcal{S}_{\mathtt{Setup},n,x}$ terminates returning $\mathcal{A}$'s output.
   - (b) Otherwise: $\mathcal{S}_{RS(k=n),\mathtt{Setup},n,x}$ chooses an (arbitrary) index $i \in \mathsf{GS}\backslash C$, and lets $(\perp, \mathsf{L}_i) \leftarrow \mathtt{LRec}(\mathsf{s}_i^{\mathsf{priv}}, \{\bar{\mathsf{s}}_j^{\mathsf{pub}}\}_{j\in\mathsf{GS}})$. $\mathcal{S}_{RS(k=n),\mathtt{Setup},n,x}$ sends $(\mathsf{abort}, \bar{i})$ (where $\bar{i} = \min(\mathsf{L}_i)$) to $\mathcal{F}_{RS(k=n),\mathtt{Setup},n,x}$, and broadcasts $(\mathsf{abort}, \bar{i})$ (on behalf of each party $i \in \mathsf{GS}\backslash C$). $\mathcal{S}_{RS(k=n),\mathtt{Setup},n,x}$ terminates returning $\mathcal{A}$'s output.

Fig. 14. The simulator $\mathcal{S}_{RS(k=n),\mathtt{Setup},n,x}$.

detectable failure property of UISSwPPS. We can conclude that, in H_5, if termination condition 1 occurs, the simulated party x outputs r_x which corresponds to the value that party x would have reconstruct via $\mathtt{Rec}$ in H_4.

H_6 is the same as H_5, except that the simulator $\mathcal{S}_6$ augments the (simulated) partially-functional $\mathcal{F}'$ to use $\mathsf{r}_x = 0$ to generate the shares $\{(\mathsf{s}_i^{\mathsf{pub}}, \mathsf{s}_i^{\mathsf{priv}})\}_{i\in[n]} \leftarrow \mathtt{Share}(\mathbb{A}_t^O, \mathsf{r}_x)$ if $x \notin C$. Note that, at this point, $\mathcal{F}'$ is the same as $\mathcal{F}_{RS(k=n),\mathtt{Setup},n,x}$.
The view of $\mathcal{A}$ in H_6 is distributed statistically close to the view of $\mathcal{A}$ in H_5. To see that this is true, observe that: (1) Due to the privacy property of UISSwPPS, the shares $\{(\mathsf{s}_i^{\mathsf{pub}}, \mathsf{s}_i^{\mathsf{priv}})\}_{i\in[C]}$ do not reveal any information about the secret shared message; (2) Due to the predictable failure property of UISSwPPS, if the reconstruction algorithms $\mathtt{Rec}$ and $\mathtt{LRec}$ fail, no information about the secret shared message is revealed.

H_7 is the ideal game. $\mathcal{S}_7 = \mathcal{S}_{RS(k=n),\mathtt{Setup},n,x}$ uses $\mathcal{F}_{RS(k=n),\mathtt{Setup},n,x}$ instead of $\mathcal{F}'$. At this point, $\mathcal{F}_{RS(k=n),\mathtt{Setup},n,x}$ returns output directly to the honest parties, so $\mathcal{S}_{RS(k=n),\mathtt{Setup},n,x}$ never sees the honest party values.
H_7 is indistinguishable from H_6 because the distributions are identical.

```
Simulator $\mathcal{S}_{RS(k=n^2),\mathtt{Setup},n}$
1. For each $x \in [n]$: upon receiving a call to $\mathcal{F}_{RS(k=n),\mathtt{Setup},n,x}$ from $\mathcal{A}$, $\mathcal{S}_{RS(k=n^2),\mathtt{Setup},n}$ does the following:
   (a) $\mathcal{S}_{RS(k=n^2),\mathtt{Setup},n}$ obtains $\{\mathsf{r}_i\}_{i\in C}$ from $\mathcal{F}_{RS(k=n^2),\mathtt{Setup},n}$;
   (b) $\mathcal{S}_{RS(k=n^2),\mathtt{Setup},n}$ sends $\{\mathsf{r}_i\}_{i\in C}$ (on behalf of $\mathcal{F}_{RS(k=n),\mathtt{Setup},n,x}$) to $\mathcal{A}$.
   (c) If $\mathcal{A}$ responds with retry: the simulator sends retry to $\mathcal{F}_{RS(k=n^2),\mathtt{Setup},n}$, and goes back to Step 1a.
       (Recall that the functionality $\mathcal{F}_{RS(k=n),\mathtt{Setup},n,x}$ only accepts at most $n$ retry commands.)
   (d) If $\mathcal{A}$ responds with continue: the simulator sends continue to $\mathcal{F}_{RS(k=n^2),\mathtt{Setup},n}$ and terminates.
   (e) If $\mathcal{A}$ responds with $(\mathsf{abort}, x)$: the simulator sends $(\mathsf{abort}, x)$ to $\mathcal{F}_{RS(k=n^2),\mathtt{Setup},n}$ and terminates.
   (f) If $\mathcal{A}$ responds with $(\mathsf{gadgabort}, \bar{i})$:
       i. $\mathcal{S}_{RS(k=n^2),\mathtt{Setup},n}$ broadcasts $\bar{i}_i := \bar{i}$ on behalf of each party $i \in [n]\backslash(\{x\} \cup C)$.
       ii. Let $\bar{i}_i$ be the accusation broadcast by $\mathcal{A}$ on behalf of each party $i \in C\backslash\{x\}$.
       iii. If there exists an $i \in C\backslash\{x\}$ such that $\bar{i}_i = i$: $\mathcal{S}_{RS(k=n^2),\mathtt{Setup},n}$ sends $(\mathsf{abort}, i)$ to $\mathcal{F}_{RS(k=n^2),\mathtt{Setup},n}$
            and terminates.
       iv. $\mathcal{S}_{RS(k=n^2),\mathtt{Setup},n}$ records the obtained conflict graph as $S_x$.
       v. If $x = n$: $\mathcal{S}_{RS(k=n^2),\mathtt{Setup},n}$ finds the first $(i, j)$ such that $S_i^j \cap S_j^i \neq \emptyset$.
          A. If $i \in S_j^k$ and $j \in S_i^k$ for every $k \in [n] \setminus C$: $\mathcal{S}_{RS(k=n^2),\mathtt{Setup},n}$ sets $\bar{i}^* := \min(i, j)$.
          B. Otherwise: $\mathcal{S}_{RS(k=n^2),\mathtt{Setup},n}$ sets $\bar{i}^* := \min(S_i^j \cap S_j^i)$.
          $\mathcal{S}_{RS(k=n^2),\mathtt{Setup},n}$ sends $(\mathsf{abort}, \bar{i}^*)$ to $\mathcal{F}_{RS(k=n^2),\mathtt{Setup},n}$ and terminates.
       vi. Otherwise: $\mathcal{S}_{RS(k=n^2),\mathtt{Setup},n}$ sends retry to $\mathcal{F}_{RS(k=n^2),\mathtt{Setup},n}$. (We now proceed to the next $x$.)
```

Fig. 15. The simulator $\mathcal{S}_{RS(k=n^2),\mathtt{Setup},n}$.

Proof of Theorem 3 We start by making several simple observations about conflict graphs.

Observation 1 *Within a given conflict graph S_x, all honest parties are in conflict with the same other parties. This is apparent because all honest parties accuse the same party — the one who really aborted the functionality; thus, any party who disagrees with one honest party disagrees with them all.*

Observation 2 *If two parties i and j are in conflict within a given conflict graph S_x, then every other participating party k is in conflict with at least one of these two parties:*

$$j \in S_x^i \Rightarrow \forall k \in [n] \setminus \{i, j, x\}, k \in S_x^i \vee k \in S_x^j.$$

This is true because, assuming no party accuses themselves, two parties who are in conflict always make different accusations; every other party must thus disagree with at least one of them.

As before, we prove Theorem 3 by demonstrating a sequence of hybrids. In the first hybrid, a dummy simulator interfaces with a real-world adversary $\mathcal{A}$, executing the protocol $\Pi^{\mathcal{F}_{RS(k=n),\mathtt{Setup},n,x}}_{RS(k=n^2),\mathtt{Setup},n}$ on behalf of the honest parties exactly as in the real world. In the last hybrid, the simulator $\mathcal{S}_{RS(k=n^2),\mathtt{Setup},n}$ (described in Figure 15) interfaces with the real-world adversary $\mathcal{A}$ and the ideal functionality $\mathcal{F}_{RS(k=n^2),\mathtt{Setup},n}$. We show a number of intermediate hybrids, wherein the simulator "imagines" interfacing with a partially-functional ideal functionality.

H_0 is the real world, where $\mathcal{S}_0$ acts on behalf of the honest parties in the protocol. Note that at this point, $\mathcal{S}_0$ is already responsible for running the ideal functionality $\mathcal{F}_{RS(k=n),\mathtt{Setup},n,x}$; it does so honestly (by running $(\mathsf{r}_1,\ldots,\mathsf{r}_n) \leftarrow \mathtt{Setup}()$).

H_1 is the same as H_0, except that the simulator $\mathcal{S}_1$ starts using a partially-functional ideal functionality $\mathcal{F}'$ in its head. $\mathcal{F}'$ excepts the command retry. In case $\mathcal{F}'$ receives this command she runs $(\mathsf{r}_1,\ldots,\mathsf{r}_n) \leftarrow \mathtt{Setup}()$ and sends $\{\mathsf{r}_i\}_{i\in C}$ to $\mathcal{S}_1$. $\mathcal{S}_1$ uses $(\mathsf{r}_1,\ldots,\mathsf{r}_n)$ to respond to the calls to $\mathcal{F}_{RS(k=n),\mathtt{Setup},n,x}$ (note that by the construction of the protocol $\Pi^{\mathcal{F}_{RS(k=n),\mathtt{Setup},n,x}}_{RS(k=n^2),\mathtt{Setup},n}$ the simulator $\mathcal{S}_1$ does not receive more than n^2 calls to $\mathcal{F}_{RS(k=n),\mathtt{Setup},n,x}$ and therefore she will not send retry more than n^2 times to $\mathcal{F}'$).
Of course, since $(\mathsf{r}_1,\ldots,\mathsf{r}_n)$ is picked from the same distribution, H_1 is indistinguishable from H_0.

H_2 is the same as H_1, except that the simulator $\mathcal{S}_2$ augments the (simulated) partially-functional $\mathcal{F}'$ to expect either retry, $\mathsf{continue}$ or $(\mathsf{abort},\bar{i})$ after sending $\{\mathsf{r}_i\}_{i\in C}$. It responds to retry exactly as it did in H_1. If it receives $(\mathsf{abort},\bar{i})$, $\mathcal{F}'$ does nothing further; otherwise (if it receives $\mathsf{continue}$), it sends r_i to each (simulated) honest party $i \in [n] \setminus C$. Those simulated parties use that output. If the execution reaches step 1b, $\mathcal{S}_2$ sends $\mathsf{continue}$ to $\mathcal{F}'$.
H_2 is indistinguishable from H_1 because the distributions are identical.

H_3 is the same as H_2, except that the simulator $\mathcal{S}_3$ augments the (simulated) partially-functional $\mathcal{F}'$ to send $(\mathsf{abort},\bar{i})$ to each (simulated) honest party $i \in [n] \setminus C$ if it receives $(\mathsf{abort},\bar{i})$ from $\mathcal{S}_3$. Those simulated parties terminate, returning $(\mathsf{abort},\bar{i})$. If the execution reaches step 1c, $\mathcal{S}_3$ sends $(\mathsf{abort},\bar{i})$ to $\mathcal{F}'$.
H_3 is indistinguishable from H_2 because the distributions are identical.

H_4 is the same as H_3, except that the simulator $\mathcal{S}_4$ sends $(\mathsf{abort},\bar{i})$ to $\mathcal{F}'$ if the execution reaches step 1(c)iA.
H_4 is indistinguishable from H_3 because the distributions are identical.

H_5 is the same as H_4, except that the simulator $\mathcal{S}_5$ sends $(\mathsf{abort},\min(\{i,j\}))$ to $\mathcal{F}'$ if the execution reaches step 2(a)iA.
H_5 is indistinguishable from H_4 because the only thing that might change is that if, in the previous hybrid, parties disagreed about whom to blame in this step, now $\mathcal{F}'$ forces unanimity. However, thanks to the broadcast accusations, all parties have the same view of the conflict graphs. By Observation 1, if it holds for one, it holds for *all* honest parties k that $i \in S^k_j$ and $j \in S^k_i$; so, the honest parties unanimously accuse $\min(\{i,j\})$.

H_6 is the same as H_5, except that the simulator $\mathcal{S}_6$ sends $(\mathsf{abort},\bar{i})$ to $\mathcal{F}'$ if the execution reaches step 2(a)iB.
H_6 is indistinguishable from H_5 because the only thing that might change is that if, in the previous hybrid, parties disagreed about whom to blame in this step, now $\mathcal{F}'$ forces unanimity. However, due to the same logic as in the previous hybrid, unanimity is guaranteed in this hybrid as well.
Informally, in H_5 the honest parties are guaranteed to unanimously accuse a corrupt party because an honest party j knows that all honest parties but party i are aleady in conflict with $\bar{i}$ (by Observation 1), and i is now also in conflict with $\bar{i}$. Similarly, an honest party i knows that all honest parties but party j are aleady in conflict with $\bar{i}$ (by Observation 1), and j is now also in conflict with $\bar{i}$.

H_7 is the ideal game. $\mathcal{S}_7 = \mathcal{S}_{RS(k=n^2),\mathtt{Setup},n}$ uses $\mathcal{F}_{RS(k=n^2),\mathtt{Setup},n}$ instead of $\mathcal{F}'$. At this point, $\mathcal{F}_{RS(k=n^2),\mathtt{Setup},n}$ returns output directly to the honest parties, so $\mathcal{S}_{RS(k=n^2),\mathtt{Setup},n}$ never sees the honest party values.
To argue that H_7 is indistinguishable from H_6, we need to show that we always end up in one of the termination conditions already covered in previous hybrids; in other words, that, if we reach step 2, we are guaranteed to have parties i, j such that $S^j_i \cap S^i_j \neq \emptyset$.

We prove this by contradiction. Assume that all of the calls to $\mathcal{F}_{RS(k=n),\texttt{Setup},n,x}$ result in an abort, and that $S_i^j \cap S_j^i = \emptyset$ for all $i, j \in [n], i \neq j$. That is, $i \in S_j^{\bar{i}} \Leftrightarrow j \notin S_i^{\bar{i}}$. We make Observation 3 under this assumption, which is a clear contradiction.
First, we define a *clique*. Let a *clique* C in a conflict graph S_x be a set of parties all of whom share a conflict with the same party.

Observation 3 *If there exists a size-c clique in some conflict graph S_i, then there also exists a size-$(c+1)$ clique in some (different) conflict graph.*

Proof. Let $C_i = \{j_1, \dots, j_c\}$ be the size-c clique in S_i.
By definition, there exists l such that $l \in S_i^{j_k}$ for $k \in [c]$. (Otherwise, the relevant call to the ideal functionality cannot have resulted in an abort.)
Now, consider the conflict graph S_l. By assumption,

$$l \in S_i^{j_k} \Rightarrow i \notin S_l^{j_k}.$$

We know that $S_l^i \neq \emptyset$; without loss of generality, say $m \in S_l^i, m \notin C_i$.
By Observation 2, for $k \in [c]$, $m \in S_l^i \Rightarrow j_k \in S_l^m \vee j_k \in S_l^i$. Since $j_k \notin S_l^i$, it follows that $j_k \in S_l^m$.
Therefore, $C_i \cup \{i\}$ form a size-$(c+1)$ clique all of are in conflict with m in S_l.

Note that Observation 3 is a contradiction, since the base case of a size-1 clique trivially exists, and since Observation 3 will lead to cliques larger than $n-1$, which is the number of parties participating in each call to the ideal functionality.

Proof of Theorem 4 We prove this by contradiction. Imagine that we have an adversary $\mathcal{A}$ that breaks the security of Π with a single oracle access to $\mathcal{F}_{RS(k=n^2),\texttt{Setup},n}$; that is, there does not exist a simulator which successfully simulates $\mathcal{A}$'s output in the ideal world. Given $\mathcal{A}$, we build another adversary $\mathcal{A}'$ that breaks the security of Π' with a single oracle access to $\mathcal{F}_{\texttt{Setup},n}$ by acting as a proxy between $\mathcal{F}_{\texttt{Setup},n}$ and $\mathcal{A}$ (which $\mathcal{A}'$ runs internally). We do this by having $\mathcal{A}'$ guess the iteration on which $\mathcal{A}$ will accept the output it is given. In more detail, $\mathcal{A}'$ guesses $k' \in [n^2]$. In its head, $\mathcal{A}'$ runs $\texttt{Setup}$ $k'-1$ times. In the beginning, and when $\mathcal{A}$ queries the $\mathcal{F}_{RS(k=n^2),\texttt{Setup},n}$ oracle with the command retry, $\mathcal{A}'$ feeds $\mathcal{A}$ the next set of simulated $\texttt{Setup}$ outputs; after these $k'-1$ outputs have been exhausted, $\mathcal{A}'$ feeds $\mathcal{A}$ the actual outputs given to it by $\mathcal{F}_{\texttt{Setup},n}$. If $\mathcal{A}$ responds with $\mathsf{continue}$ before reaching the actual outputs, or if $\mathcal{A}$ does not respond with $\mathsf{continue}$ when given the actual outputs, $\mathcal{A}'$ aborts; otherwise, it runs $\mathcal{A}$ to completion, and outputs whatever $\mathcal{A}$ outputs. If $\mathcal{A}'$ guesses k' correctly, then the view of $\mathcal{A}$ is perfectly indistinguishable from its view in an ideal world execution with $\mathcal{F}_{RS(k=n^2),\texttt{Setup},n}$, where by assumption no simulator exists.

By the security of Π', there exists a simulator $\mathcal{S}'$ for $\mathcal{A}'$. Observe that $\mathcal{S}'$ in combination with $\mathcal{A}'$ defines a simulator $\mathcal{S}$ for $\mathcal{A}$. More concretely, $\mathcal{S}$ has black-box access to $\mathcal{A}$ and internally runs $\mathcal{S}'$ and $\mathcal{A}'$, while emulating $\mathcal{F}_{RS(k=n^2),\texttt{Setup},n}$. However, $\mathcal{S}$ modifies $\mathcal{A}'$ not to guess k' and never to abort unless $\mathcal{A}$ aborts. We note that if $\mathcal{S}'$ correctly simulates the view of $\mathcal{A}'$, then $\mathcal{S}$ correctly simulates the view of $\mathcal{A}$.

The existence of $\mathcal{S}$ contradicts our initial assumption.

5 Building UISSwPPS

In this section, we build a unanimously identifiable secret sharing scheme with public and private shares. In Section 5.1, we describe two building blocks: unanimously identifiable commitments and unanimously identifiable secret sharing. In Section 5.2, we describe our construction and prove its security.

5.1 Building Blocks

Unanimously Identifiable Commitments Unanimously identifiable commitments (UIC) have been introduced by Ishai, Ostrovsky, and Seyalioglu [IOS12]. Such commitments allow a trusted dealer to commit to a message msg by distributing $\mathsf{com}_1, \ldots, \mathsf{com}_n$ among n recipients and providing a sender with decommitment information dec. From a security point of view, we require that the joint view of all recipients should contain no information about msg and that any decommitment information dec' published by the sender either causes all honest parties to reconstruct msg or all parties to unanimously abort. Ishai, Ostrovsky, and Seyalioglu have shown how to construct such commitments with information-theoretic security.

Definition 15 (Unanimously Identifiable Commitments). *A UIC scheme consists of a probabilistic polynomial-time algorithm* $\mathtt{Commit}$ *and a deterministic polynomial-time algorithm* $\mathtt{Open}$ *with the following syntax:*

$\mathtt{Commit}(s) \to (\mathsf{com}_1, \ldots, \mathsf{com}_n, \mathsf{dec})$**:** *takes as input a message* $\mathsf{msg} \in \{0,1\}^*$, *and outputs n commitments* $\mathsf{com}_1, \mathsf{com}_2, \ldots, \mathsf{com}_n$, *and decommitment information* dec.

$\mathtt{Open}(\mathsf{com}_i, \mathsf{dec}) \to \mathsf{msg}/\bot$**:** *takes as input* com_i *and the decommitment information* dec, *and outputs a value in* $\{0,1\}^* \cup \{\bot\}$.

Furthermore, $(\mathtt{Commit}, \mathtt{Open})$ *should satisfy* correctness *(Definition 16),* privacy *(Definition 17), and* binding with agreement on abort *(Definition 18).*

Definition 16 (Correctness). *A UIC* $(\mathtt{Commit}, \mathtt{Open})$ *is* correct *if for any* $\mathsf{msg} \in \{0,1\}^*$ *and any* $i \in [n]$,

$$\Pr[(\mathsf{com}_1, \mathsf{com}_2, \ldots, \mathsf{com}_n, \mathsf{dec}) \leftarrow \mathtt{Commit}(\mathsf{msg}) : \mathtt{Open}(\mathsf{com}_i, \mathsf{dec}) = \mathsf{msg}] = 1.$$

Definition 17 (Privacy). *A UIC* $(\mathtt{Commit}, \mathtt{Open})$ *is* private *if for any* $\mathsf{msg}, \mathsf{msg}' \in \{0,1\}^*$ *with* $|\mathsf{msg}| = |\mathsf{msg}'|$

$$\begin{aligned} &\{(\mathsf{com}_1, \ldots, \mathsf{com}_n) \mid (\mathsf{com}_1, \mathsf{com}_2, \ldots, \mathsf{com}_n, \mathsf{dec}) \leftarrow \mathtt{Commit}(\mathsf{msg})\} \\ \equiv &\{(\mathsf{com}_1, \ldots, \mathsf{com}_n) \mid (\mathsf{com}_1, \mathsf{com}_2, \ldots, \mathsf{com}_n, \mathsf{dec}) \leftarrow \mathtt{Commit}(\mathsf{msg}')\}. \end{aligned}$$

Definition 18 (Binding with Agreement on Abort). *Consider the security game described in Figure 16. A UIC* $(\mathtt{Commit}, \mathtt{Open})$ *is* binding with agreement on abort *if for any message* $\mathsf{msg} \in \{0,1\}^*$ *and adversary* $\mathcal{A}$, *there exists a negligible function* $\mathsf{negl}(\cdot)$ *such that*

$$\Pr[\mathcal{A} \text{ wins } \mathsf{game}_{\mathsf{baa}}(\mathcal{A})] \leq \mathsf{negl}(\lambda)$$

where the probability is taken over the random coins of $\mathcal{C}$ *and* $\mathcal{A}$.

Remark 2. For technical convenience, we slightly modified the security game $\mathsf{game}_{\mathsf{baa}}(\mathcal{A})$ by allowing the adversary to first obtain dec and then query the set C. The original UIC construction of Ishai, Ostrovsky, and Seyalioglu [IOS12] directly satisfies our new notion and if necessary all of our proofs can also be done with the original security definition; albeit with a slightly larger security loss.

Unanimously Identifiable Secret Sharing Unanimously identifiable secret sharing (UISS) is another primitive that has been introduced and constructed with information-theoretic security by Ishai, Ostrovsky, and Seyalioglu [IOS12].

Definition 19 (Unanimously Identifiable Secret Sharing Scheme). *A* unanimously identifiable secret sharing scheme *for message space* $\{0,1\}^*$ *is a secret sharing scheme (Definition 4) that additionally satisfies* local identifiability *(Definition 20) and* predictable failures *(Definition 14 with the appropriate syntactic modifications).*

A secret sharing scheme is said to be unanimously identifiable if all share holders either reconstruct the correct message, or unanimously agree on some subset of shares which they consider to be invalid.

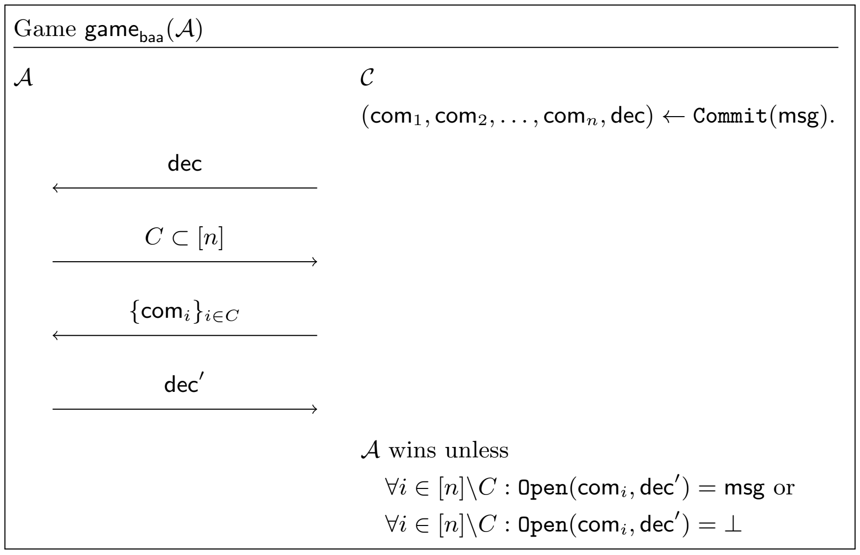
Game $\mathsf{game}_{\mathsf{baa}}(\mathcal{A})$
$\mathcal{A}$
$\mathcal{C}$
$(\mathsf{com}_1, \mathsf{com}_2, \ldots, \mathsf{com}_n, \mathsf{dec}) \leftarrow \mathtt{Commit}(\mathsf{msg})$.
dec
$C \subset [n]$
$\{\mathsf{com}_i\}_{i \in C}$
dec'
$\mathcal{A}$ wins unless
$\forall i \in [n] \backslash C : \mathtt{Open}(\mathsf{com}_i, \mathsf{dec}') = \mathsf{msg}$ or
$\forall i \in [n] \backslash C : \mathtt{Open}(\mathsf{com}_i, \mathsf{dec}') = \perp$

Fig. 16. Security game for binding with agreement on abort.

Definition 20 (Local Identifiability). *Consider the game described in Figure 17. A secret sharing scheme* $(\mathtt{Share}, \mathtt{LRec})$ *for access structure* $\mathbb{A}$ *is* locally identifiable *if for any message* $\mathsf{msg} \in \{0,1\}^*$ *and adversary* $\mathcal{A}$*, there exists negligible function* $\mathsf{negl}(\cdot)$ *such that*

$$\Pr[\mathcal{A} \textit{ wins } \mathsf{game}_{\mathsf{li}}(\mathcal{A})] \leq \mathsf{negl}(\lambda)$$

where the probability is taken over the random coin of $\mathcal{C}$ *and* $\mathcal{A}$*.*

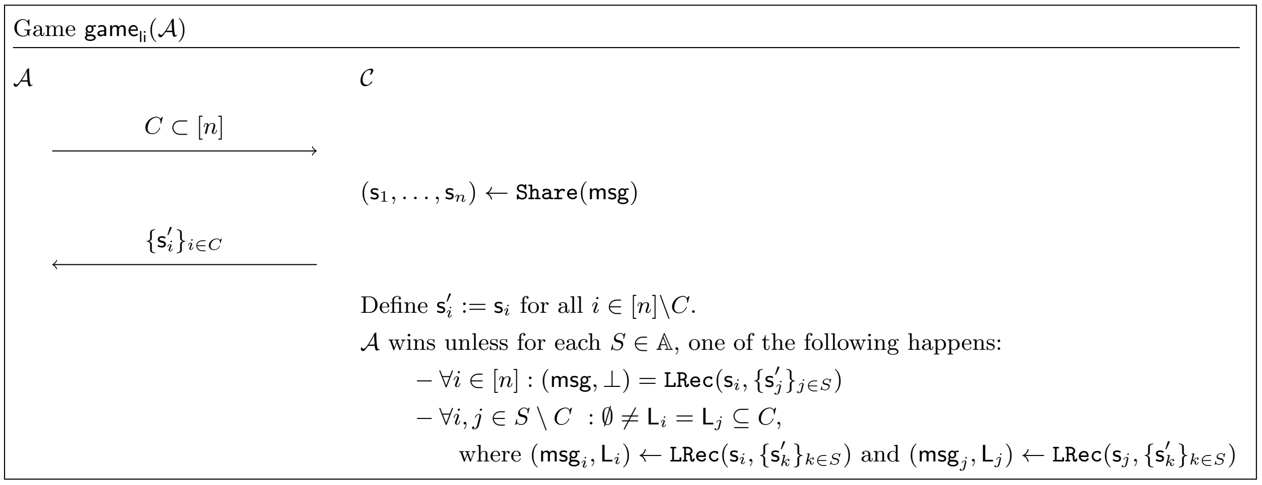
Game $\mathsf{game}_{\mathsf{li}}(\mathcal{A})$
$\mathcal{A}$
$\mathcal{C}$
$C \subset [n]$
$(\mathsf{s}_1, \ldots, \mathsf{s}_n) \leftarrow \mathtt{Share}(\mathsf{msg})$
$\{\mathsf{s}'_i\}_{i \in C}$
Define $\mathsf{s}'_i := \mathsf{s}_i$ for all $i \in [n] \backslash C$.
$\mathcal{A}$ wins unless for each $S \in \mathbb{A}$, one of the following happens:
$- \forall i \in [n] : (\mathsf{msg}, \perp) = \mathtt{LRec}(\mathsf{s}_i, \{\mathsf{s}'_j\}_{j \in S})$
$- \forall i, j \in S \setminus C \ : \emptyset \neq \mathsf{L}_i = \mathsf{L}_j \subseteq C$,
where $(\mathsf{msg}_i, \mathsf{L}_i) \leftarrow \mathtt{LRec}(\mathsf{s}_i, \{\mathsf{s}'_k\}_{k \in S})$ and $(\mathsf{msg}_j, \mathsf{L}_j) \leftarrow \mathtt{LRec}(\mathsf{s}_j, \{\mathsf{s}'_k\}_{k \in S})$

Fig. 17. Security game for local identifiability.

UISSwPPS

```
Share(msg):
    1. Compute $\{\mathsf{s}_i\}_{i\in[n]\setminus O} \leftarrow \mathtt{UISS.Share}(\mathsf{msg})$;
    2. For each $i \in [n] \setminus O$, compute $\left(\mathsf{com}_i^1, \ldots, \mathsf{com}_i^n, \mathsf{dec}_i\right) \leftarrow \mathtt{Commit}(\mathsf{s}_i)$;
    3. If $i \in O$, then $\mathsf{s}_i^{\mathsf{pub}} := \perp$; otherwise, $\mathsf{s}_i^{\mathsf{pub}} := (\mathsf{s}_i, \mathsf{dec}_i)$;
    4. Set $\mathsf{s}_i^{\mathsf{priv}} := (\{\mathsf{com}_j^i\}_{j\in[n]\setminus O}, \mathsf{s}_i^{\mathsf{pub}})$.
LRec($\mathsf{s}_i^{\mathsf{priv}}, \{\mathsf{s}_j^{\mathsf{pub}}\}_{j\in S}$):
    1. Let $\mathsf{L}_i := \emptyset$;
    2. For each $j \in S \setminus O$, if $\mathtt{Open}(\mathsf{com}_j^i, \mathsf{dec}_j) \neq \mathsf{s}_j$, then $\mathsf{L}_i = \mathsf{L}_i \cup \{j\}$;
    3. If $\mathsf{L}_i \neq \emptyset$, then return $(\perp, \mathsf{L}_i)$;
    4. If $i \in O$, then
        (a) Set $\mathsf{s}' := \mathsf{s}_j$ where $j = \min(S)$;
        (b) Return $(\overline{\mathsf{msg}}, \perp)$, where $\overline{\mathsf{msg}} = \mathtt{UISS.Rec}(\mathsf{s}', \{\mathsf{s}_j\}_{j\in S})$.
    5. Otherwise, if $i \notin O$, then return $(\overline{\mathsf{msg}}, \perp)$, where $\overline{\mathsf{msg}} = \mathtt{UISS.Rec}(\mathsf{s}_i, \{\mathsf{s}_j\}_{j\in S})$.
Rec($\{\mathsf{s}_j^{\mathsf{pub}}\}_{j\in S}$):
    1. For each $i \in S$, compute $(\mathsf{msg}_i, \mathsf{L}_i) = \mathtt{UISS.Rec}(\mathsf{s}_i, \{\mathsf{s}_j\}_{j\in S})$;
    2. If $\exists\, i, j$ s.t. $\mathsf{msg}_i \neq \mathsf{msg}_j$ output $\perp$, otherwise output $\mathsf{msg}_k$ with $k = \min(S)$.
```

Fig. 18. UISSwPPS construction.

5.2 Construction

Theorem 7. *Let $\mathbb{A}_{n,t}^O$ be a threshold access structure with k observers, where $k < n$. Let* ($\mathtt{UISS.Share}, \mathtt{UISS.Rec}$) *be a UISS for $\mathbb{A}_{m,t}$, where $m = n - k$. Let* ($\mathtt{Commit}, \mathtt{Open}$) *be a n-party UIC. Then, the construction in Figure 18 is a UISSwPPS for $\mathbb{A}_{n,t}^O$.*

Proof. We will now proceed to prove all the properties of the UISSwPPS.

Correctness. Follows by inspection.
Privacy. Follows by the privacy of UIC and UISS.
Adaptive Local Identifiability. Suppose for contradiction that the adversary $\mathcal{A}$ wins the game in Figure 3.

Reduction R. Let us first recall that a public share $\mathsf{s}_j^{\mathsf{pub}}$ of our UISSwPPS is of the form $\mathsf{s}_j^{\mathsf{pub}} = (\mathsf{s}_j, \mathsf{dec}_j)$ and the corresponding private share is of the form $\mathsf{s}_j^{\mathsf{priv}} = (\{\mathsf{com}_j^i\}_{j\in[n]\setminus O}, \mathsf{s}_j^{\mathsf{pub}})$. We show a reduction R that will act as a proxy between the challenger $\mathcal{C}$ of the game described in Figure 16 and $\mathcal{A}$ which is playing the game in Figure 3.

R picks a random index $i^* \in [n]$ and a random message $\mathsf{msg} \in \{0,1\}^*$. She then computes $\{\mathsf{s}_i\}_{i\in[n]\setminus O} \leftarrow \mathtt{UISS.Share}(\mathsf{msg})$ and sends s_{i^*} to $\mathcal{C}$ obtaining back dec_{i^*}. R sets $\mathsf{s}_{i^*}^{\mathsf{pub}} = (\mathsf{s}_{i^*}, \mathsf{dec}_{i^*})$; for all $i \in [n]\setminus\{i^*\}$, R computes $\mathsf{s}_i^{\mathsf{pub}}$ honestly (Figure 18). R receives the corrupted set C from $\mathcal{A}$ and forward it to $\mathcal{C}$; $\mathcal{C}$ sends back the set $\{\mathsf{com}_{i^*}^j\}_{j\in[n]}$. R aborts if $i^* \notin C$ or $i^* \in O$. Otherwise, R uses the honestly generated shares and the ones received from $\mathcal{C}$ to compute the private shares honestly. R sends $\{\mathsf{s}_j^{\mathsf{priv}}\}_{j\in C}$ to $\mathcal{A}$ receiving the set $\{\tilde{\mathsf{s}}_j^{\mathsf{pub}}\}_{j\in C}$. R retrieves dec' from $\tilde{\mathsf{s}}_{i^*}^{\mathsf{pub}}$, which she forwards to $\mathcal{C}$. We finish the proof observing that since by hypothesis $\mathcal{A}$ has a non-negligible probability of winning the game in Figure 3, R has a non-negligible advantage of winning the game in Figure 16 (in particular, R provides dec' which will make cause some — but not all — parties to abort). The reduction has a loss of $\frac{1}{n}$.

If a list L_i is empty then step 3 in of $\mathtt{LRec}$ (Figure 18) is not executed, and by the correctness of UIC and UISS we are guaranteed that a correct message is reconstructed.

Publicly Detectable Failures. Suppose for contradiction that with non-negligible probability $\mathcal{A}$ wins the game in Figure 4. Then $\exists S \in \mathbb{A}^{O}_{n,t}$ (which contains an index not corrupted by $\mathcal{A}$) s.t. $\mathtt{Rec}$ returns a message $\overline{\mathsf{msg}} \notin \{\bot, \mathsf{msg}\}$. If this is the case $\mathcal{A}$ could be used to break the local identifiability of UISS.

Reduction R. Let us first recall that the public share $\mathsf{s}_j^{\mathsf{pub}}$ of our UISSwPPS is of the form $\mathsf{s}_j^{\mathsf{pub}} = (\mathsf{s}_j, \mathsf{dec}_j)$. We show a reduction R that will act as a proxy between the challenger $\mathcal{C}$ of the game described in Figure 17 and $\mathcal{A}$ which is playing the game in Figure 4. R receives the corrupted set C from $\mathcal{A}$ and forward it to $\mathcal{C}$ receiving back the shares $\{\mathsf{s}_j\}_{j\in C}$. R acting as a challenger for $\mathcal{A}$ computes $\mathtt{Commit}$ on input s_j for $j \in C$ to complete the public shares $\{\mathsf{s}_j^{\mathsf{pub}}\}_{j\in C}$ that R forwards to $\mathcal{A}$. R receives from $\mathcal{A}$ the set $\{\tilde{\mathsf{s}}_j^{\mathsf{pub}}\}_{j\in C}$ and uses it to constructs the set $\{\tilde{\mathsf{s}}_j\}_{j\in C}$ that she forwards to $\mathcal{C}$. We finish the proof observing that since by hypothesis $\mathcal{A}$ has a non-negligible probability of winning the game in Figure 4 R has a non-negligible advantage of winning the game in Figure 17 (in particular, R provides shares that are able to make reconstruct a different message from the one shared).

Consistent Failures. Suppose for contradiction that $\mathcal{A}$ wins the game in Figure 5. Then $\exists S \in \mathbb{A}^{O}_{n,t}$ s.t. $\mathtt{Rec}(\{\mathsf{s}_j^{\mathsf{pub}}\}_{j\in S})$ outputs $\bot$.

Looking at $\mathtt{Rec}$ defined in Figure 18 if it returns $\bot$, then it must be the case that there exist i, j s.t. $\mathsf{msg}_i = \mathtt{UISS.Rec}(\mathsf{s}_i, \{\mathsf{s}_k\}_{k\in S})$, $\mathsf{msg}_j = \mathtt{UISS.Rec}(\mathsf{s}_j, \{\mathsf{s}_k\}_{k\in S})$ and $\mathsf{msg}_j \neq \mathsf{msg}_i$, since UISS is perfect correct we can conclude that at least one share, say s_j with $j \in S$, was changed by $\mathcal{A}$. Considering the public share $\mathsf{s}_j^{\mathsf{pub}}$ of our UISSwPPS are of the form $\mathsf{s}_j^{\mathsf{pub}} = (\mathsf{s}_j, \mathsf{dec}_j)$ we observe that $\mathcal{A}$ could have left dec_i unchanged or not. In the first case from the correctness of UIC we can conclude that $\mathtt{LRec}$ outputs $\bot$; in the second case from the binding with agreement on abort property of UIC follows that $\mathtt{LRec}$ outputs $\bot$ (the reduction follows very closely to the one described for adaptive local identifiability).

Predictable Failure.

Predictable failure for $\mathtt{LRec}$. Roughly speaking, the predictable failure property requires the existence of an algorithm $\mathtt{SLRec}$ which "predicts" the output of the adversary when she run $\mathtt{LRec}$. Below we show the algorithm $\mathtt{SLRec}$.

$\mathtt{SLRec}(C, S, i^*, \{\mathsf{s}_j^{\mathsf{pub}}\}_{j\in[n]}, \{\mathsf{s}_j^{\mathsf{priv}}\}_{j\in C}, \{\tilde{\mathsf{s}}_j^{\mathsf{pub}}\}_{j\in S\cap C})$ computes the following steps:

```
1. Parse $\mathsf{s}_j^{\mathsf{priv}}$ as $(\{\mathsf{com}_k^j\}_{k\in[n]\setminus O}, \mathsf{s}_j^{\mathsf{pub}})$ and $\mathsf{s}_j^{\mathsf{pub}}$ as $(\mathsf{s}_j, \mathsf{dec}_j)$ for $j \in C$.
2. Parse $\tilde{\mathsf{s}}_j^{\mathsf{pub}}$ as $(\tilde{\mathsf{s}}_j, \mathsf{dec}_j)$ for $j \in S \cap C$.
3. Let $\mathsf{L}_{i^*} := \emptyset$;
4. If $i^* \in C$ :
   (a) For each $j \in C \setminus O$, if $\mathtt{Open}(\mathsf{com}_j^{i^*}, \mathsf{dec}_j) = \bot$, then $\mathsf{L}_{i^*} = \mathsf{L}_{i^*} \cup \{j\}$.
   (b) If $\mathsf{L}_{i^*} \neq \emptyset$: $(b_{i^*}, \mathsf{L}_{i^*}) = \mathtt{UISS.SRec}(C, S, i^*, \{\mathsf{s}_j\}_{j\in C}, \{\tilde{\mathsf{s}}_j\}_{j\in S\cap C})$.
   (c) If $\mathsf{L}_{i^*} = \emptyset$ output $(1, \bot)$ otherwise output $(0, \mathsf{L}_{i^*})$
5. Otherwise:
   (a) For randomly selected $k \in C$: for each $j \in C\setminus O$, if $\mathtt{Open}(\mathsf{com}_j^i, \mathsf{dec}_j) \neq \mathsf{s}_j$, then $\mathsf{L}_k = \mathsf{L}_k \cup \{j\}$.
   (b) If $\mathsf{L}_k = \emptyset$ output $(1, \bot)$ otherwise output $(0, \mathsf{L}_k)$
```

It remain to observe that if $\mathcal{A}$ has a non-negligible probability of winning in the predictable failure game, then when $i \in C$ we could construct a reduction that wins binding with agreement on abort property of UIC; otherwise (when $i \notin C$) we could construct a reduction that wins the predictable failure game of UISS.

Predictable failure for $\mathtt{Rec}$. Roughly speaking, the predictable failure property requires the existence of an algorithm $\mathtt{SRec}$ which "predicts" the output of the adversary when she run $\mathtt{Rec}$. Therefore we show how the algorithm $\mathtt{SRec}$ works, intuitively $\mathtt{SRec}$ relys on the algorithms $\mathtt{UISS.SRec}$ which it exists by the predictable failure property of UISS.

The algorithm $\mathtt{SRec}(C, S, \{\mathsf{s}_j^{\mathsf{pub}}\}_{j\in C}, \{\tilde{\mathsf{s}}_j^{\mathsf{pub}}\}_{j\in S\cap C})$ computes the following steps:

```
– Parse $\mathsf{s}_j^{\mathsf{pub}}$ as $(\mathsf{s}_j, \mathsf{dec}_j)$ for $j \in C$.
– Parse $\tilde{\mathsf{s}}_j^{\mathsf{pub}}$ as $(\tilde{\mathsf{s}}_j, \tilde{\mathsf{dec}}_j)$ for $j \in S \cap C$.
– For each $i \in C$, compute $b_i = \mathtt{UISS.SRec}(C, S, i, \{\mathsf{s}_j\}_{j\in C}, \{\tilde{\mathsf{s}}_j\}_{j\in S\cap C})$;
– If $\exists\ j$ s.t. $b_j = 0$ output 0, otherwise output 1.
```

It remain to observe that if $\mathcal{A}$ has a non-negligible probability of winning in the predictable failure game, then we could construct a reduction that wins the predictable failure game of UISS.

References

Bea90. Donald Beaver. Multiparty protocols tolerating half faulty processors. In Gilles Brassard, editor, *CRYPTO'89*, volume 435 of *LNCS*, pages 560–572. Springer, Heidelberg, August 1990.

BMMMQ20. Nicholas-Philip Brandt, Sven Maier, Tobias Müller, and Jörn Müller-Quade. Constructing secure multi-party computation with identifiable abort. *IACR Cryptol. ePrint Arch.*, 2020:153, 2020.

Cle86. Richard Cleve. Limits on the security of coin flips when half the processors are faulty (extended abstract). In *18th ACM STOC*, pages 364–369. ACM Press, May 1986.

CvT95. Claude Crépeau, Jeroen van de Graaf, and Alain Tapp. Committed oblivious transfer and private multi-party computation. In Don Coppersmith, editor, *CRYPTO'95*, volume 963 of *LNCS*, pages 110–123. Springer, Heidelberg, August 1995.

IOS12. 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, Heidelberg, March 2012.

IOZ14. Yuval Ishai, Rafail Ostrovsky, and Vassilis Zikas. Secure multi-party computation with identifiable abort. In Juan A. Garay and Rosario Gennaro, editors, *CRYPTO 2014, Part II*, volume 8617 of *LNCS*, pages 369–386. Springer, Heidelberg, August 2014.

Kil88. Joe Kilian. Founding cryptography on oblivious transfer. In *20th ACM STOC*, pages 20–31. ACM Press, May 1988.

Rab81. Michael O Rabin. How to exchange secrets with oblivious transfer. *Technical Memo*, 1981. https://www.iacr.org/museum/rabin-obt/obtrans-eprint187.pdf.

RB89. Tal Rabin and Michael Ben-Or. Verifiable secret sharing and multiparty protocols with honest majority (extended abstract). In *21st ACM STOC*, pages 73–85. ACM Press, May 1989.