# Efficient Constant-Round MPC with Identifiable Abort and Public Verifiability

Carsten Baum[1⋆], Emmanuela Orsini[2⋆⋆], Peter Scholl[1⋆⋆⋆], and Eduardo Soria-Vazquez[1†]

[1] Aarhus University, Denmark
[2] imec-COSIC, KU Leuven, Belgium

**Abstract.** Recent years have seen a tremendous growth in the interest in secure multiparty computation (MPC) and its applications. While much progress has been made concerning its efficiency, many current, state-of-the-art protocols are vulnerable to *Denial of Service attacks*, where a cheating party may prevent the honest parties from learning the output of the computation, whilst remaining anonymous. The security model of *identifiable abort* aims to prevent these attacks, by allowing honest parties to agree upon the identity of a cheating party, who can then be excluded in the future. Several existing MPC protocols offer security with identifiable abort against a dishonest majority of corrupted parties. However, all of these protocols have a round complexity that scales linearly with the depth of the circuit (and are therefore unsuitable for use in high latency networks) or use cryptographic primitives or techniques that have a high computational overhead.

In this work, we present the first efficient MPC protocols with identifiable abort in the dishonest majority setting, which run in a constant number of rounds and make only black-box use of cryptographic primitives. Our main construction is built from highly efficient primitives in a careful way to achieve identifiability at a low cost. In particular, we avoid the use of public-key operations outside of a setup phase, incurring a relatively low overhead on top of the fastest currently known constant-round MPC protocols based on garbled circuits. Our construction also avoids the use of adaptively secure primitives and heavy zero-knowledge machinery, which was inherent in previous works. In addition, we show how to upgrade our protocol to achieve *public verifiability* using a public bulletin board, allowing any external party to verify correctness of the computation or identify a cheating party.

⋆ Supported by the European Research Council (ERC) under the European Unions' Horizon 2020 research and innovation programme under grant agreement No 669255 (MPCPRO) as well as the BIU Center for Research in Applied Cryptography and Cyber Security in conjunction with the Israel National Cyber Bureau in the Prime Minister's Office. Part of this work was done while the author was at Bar Ilan University.

⋆⋆ Supported in part by ERC Advanced Grant ERC-2015-AdG-IMPaCT.

⋆⋆⋆ Supported in part by the Danish Independent Research Council under Grant-ID DFF-6108-00169 (FoCC) and an Aarhus University Research Foundation (AUFF) starting grant.

† Supported by the Carlsberg Foundation under the Semper Ardens Research Project CF18-112 (BCM).

# 1 Introduction

Secure Multi-Party Computation (MPC) is a general term for techniques which allow a set of $n$ parties to compute a function $f$ on their private inputs such that only the output of the function becomes known. Using MPC as a tool to achieve security generally comes with an inherent slowdown over insecure solutions, so using the right MPC protocol with suitable properties is crucial in order to foster adoption in practice. For certain requirements, it is even known that MPC is impossible to achieve.

For example, while in the *honest majority* setting, where more than half of the parties are honest, MPC for any function is possible, when there is a *dishonest majority* it is well-known that *fairness* for MPC is impossible, in general [15]. The fairness property means that if any corrupted party learns the output then all the honest parties do as well, so a dishonest party cannot withhold the output from the other parties. To work around this impossibility, most MPC protocols for dishonest majority settle for the weaker notion of *security with abort*, which allows the adversary to abort the protocol, possibly after learning the output.

However, a major downside of this model is that it does not protect against denial-of-service attacks. This motivates the stronger model of *MPC with identifiable abort*, or ID-MPC, where if the adversary aborts then the honest parties will agree upon the identity of a cheating party. This allows the honest parties to exclude cheaters and re-run the aborting protocol, and it can also be combined with a distributed ledger (such as in [35]) to achieve monetary fairness (see e.g. [4] for an overview). The concept of ID-MPC was first implicitly considered in the context of covert security, and more formally studied in later works [16, 29].

A related, desirable property of an MPC protocol is *public verifiability* [3, 45], which allows any external party to verify the correctness of some claimed outputs of the protocol by, for instance, inspecting public values posted to a bulletin board. This is important for settings where the computation is of particular interest to the public, for example, it may be desirable for the results of a research study on private medical data to be publicly verifiable. It is also relevant to the client-server setting, where many clients outsource a computation to a set of non-colluding servers and wish to verify the result, without interacting with the servers.

As well as security properties like the above, an important aspect when choosing an MPC protocol is its efficiency. This can be measured in terms of number of rounds of communication, total communication complexity (i.e. amount of data sent over the network), and computational overhead (compared with computing the function in the clear). In this work, we consider the problem of efficiently constructing MPC in the dishonest majority setting providing security with identifiable abort and public verifiability, in a constant number of rounds of interaction.

## 1.1 Previous Work on Constant-Round MPC, Identifiable Abort and Public Verifiability

*Constant-Round MPC.* The main tool for building constant-round MPC is garbled circuits, which were introduced by Yao [49] for 2-party secure computation. Garbled circuits were generalized to the multi-party setting by Beaver, Micali and Rogaway [6],

who constructed a constant-round MPC protocol (called "BMR") that can support a dishonest majority of participants. The BMR protocol makes heavy, non-black-box use of a pseudorandom generator, so is inefficient in practice.

Subsequently, constant-round MPC making only *black-box* use of cryptographic primitives was presented by Damgård and Ishai [18], for the honest majority setting, and extended to the case of a dishonest majority by Ishai et al. [30]. Later, more efficient black-box solutions with active security for dishonest majority were introduced by Lindell et al. [36, 37], who used somewhat homomorphic encryption in a preprocessing phase of the protocols. Currently, the most efficient protocols are those by Wang et al. [48] and Hazay et al. [26], which use oblivious transfer (OT) instead of homomorphic encryption, and can be instantiated very efficiently using the TinyOT-protocol [40, 21] based on fast OT extension techniques [28, 31].

*ID-MPC in the Dishonest Majority Setting.* The seminal MPC protocol of Goldreich, Micali and Wigderson [24] can be combined with any public-coin zero-knowledge proof system to obtain ID-MPC for dishonest majority, and the same holds for the BMR protocol [6] to achieve a constant round complexity. However, the resulting protocols make extensive, non-black-box use of cryptographic primitives and are not practical. Additionally, also [41] implies a constant-round ID-MPC scheme that is not black-box (and secure in the stand-alone setting as observed by [7]. More recently, there has been interest in concretely efficient ID-MPC. Ishai, Ostrovsky and Zikas [29] presented an ID-MPC protocol in the preprocessing model, where a trusted dealer gives the parties some correlated randomness, with information-theoretic security. They also gave a general compiler that allows removing the trusted dealer, leading to the first ID-MPC protocol making only black-box use of cryptographic primitives, namely, an adaptively secure oblivious transfer protocol and a broadcast channel. Concurrent to this work, Brandt et al. [9] studied the feasibility of ID-MPC from lower-cardinality primitives as well as the relation of the conflict graph to identifiable abort. Their work is orthogonal to ours, as we are interested in concrete and practical constructions.

Baum et al. [5] also construct ID-MPC in the preprocessing model, with better concrete efficiency, by combining a variant of the BDOZ protocol [8] with information-theoretic signatures, and homomorphic encryption for the preprocessing. Other works [17, 46] have added identifiability to the practical SPDZ protocol [19], obtaining more efficient results in a similar setting. These works, while concretely quite practical, all require a number of rounds of interaction that scales *linearly* with the multiplicative depth of the circuit being evaluated.

*MPC with Public Verifiability.* The idea of secure computation with public verifiability was first introduced in the two-party setting for covert security by Asharov and Orlandi [2]. Subsequent works [33, 27] later improved upon the efficiency of their construction, and in particular the size of the cheating certificate, for which the work of Hong et al. [27] requires $< 400$ bytes for $128$ bit security.

The notion of public verifiability for actively secure dishonest majority MPC (with potentially all parties being corrupted) has been introduced independently by Baum et al. [3] and Schoenmakers and Veeningen [45]. Their work ensures privacy if at least one party is honest and correctness for any level of corruption. In subsequent works, [5, 17]

independently showed how to combine public verifiability and identifiable abort for general computations where either the correctness of the output is attested or a cheater will be found by a third party. Both works rely on expensive tools in a preprocessing phase (lattice-based encryption for large fields), have a circuit depth-dependent round complexity and have not been implemented in practice. Another, more general approach for publicly verifiable MPC with identifiable abort was given in [32] where the authors presented a general compiler based on the approach of [29].

### 1.2 Contributions

In this work, we present the first *concretely efficient* and *constant-round* MPC protocols that provide security with identifiable abort and public verifiability in the dishonest majority setting. Note that all our protocols are in the setting of static corruptions.

Our results for identifiable abort assume access to a *broadcast channel*, while for public verifiability we need a public *bulletin board*, and in both cases we count round complexity by assuming that their consumes a single round. In practice, if using an authenticated broadcast protocol [20, 43] to implement this, each broadcast requires $\Omega(n)$ rounds of point-to-point messages [23]. Alternatively, broadcast can be realized using a bulletin board or blockchain, giving a constant number of rounds of interaction with this functionality. Note that it seems difficult to avoid the use of broadcast, since MPC with identifiable abort itself implies secure broadcast [16].

We first establish the feasibility of ID-MPC with constant round complexity, with black-box use of cryptographic primitives.

**Theorem 1.1 (informal)** *There exists an ID-MPC protocol for securely realizing any functionality in a constant number of rounds, given black-box access to an adaptively secure oblivious transfer protocol and a pseudorandom function.*

Next, our main result is a more *concretely efficient* protocol, with greatly reduced communication complexity and allowing optimizations like efficient OT extension and free-XOR gates.

**Theorem 1.2 (informal)** *There exists an ID-MPC protocol for securely realizing any functionality in a constant number of rounds, given black-box access to a statically secure oblivious transfer protocol and a circular 2-correlation robust hash function.*

Interestingly, and unlike the previous result, in this construction we manage to avoid the need for adaptively secure OT, allowing our protocol to use efficient OT extensions [28], which are impossible with adaptive security in the standard model as showed by Lindell and Zarosim [39]. This means that Theorems 1.1 and 1.2 are incomparable from a feasibility perspective, since although constructions of adaptively secure OT are known from standard assumptions, it cannot be built from static OT in a black-box manner [38].

Finally, we show how to upgrade the above protocol to achieve public verifiability using a public bulletin board.

**Theorem 1.3 (informal)** *Assuming additionally a secure public bulletin board, there is a black-box ID-MPC protocol with public verifiability, with a constant number of rounds of interaction with the bulletin board.*

We obtain our first feasibility result with a variant of the Damgård-Ishai protocol [18] for constant-round honest majority MPC, tailored for the dishonest majority setting using information-theoretic signatures [13]. We then obtain a protocol with identifiable abort by combining this with a transformation from [29], which needs an adaptively secure OT protocol. While our construction achieves static security, we want to remark that it is possible to construct an adaptively secure constant-round ID-MPC protocol by applying the [29] transform to the [30] protocol. This approach, on the other hand, will make non-black box use of the underlying PRF by the [29] compiler whereas our construction is fully black-box.

Our second protocol is much more attractive from a practical perspective, since it builds upon recent, optimized MPC protocols that offer active security with (non-identifiable) abort using BMR-style garbled circuits [26, 48]. We also support the free-XOR technique [34], by assuming a suitable circular 2-correlation robust hash function [14]. Our core idea is a lightweight method of adding identifiability to the MPC protocol of Hazay, Scholl and Soria-Vazquez [26], which creates a BMR garbled circuit using OT and any non-constant round MPC protocol[3]. We obtain our efficient method in two steps: firstly, we devise a cheater identification procedure for the online phase, based on opening a circuit-independent number of additively homomorphic commitments. The cheater identification is highly efficient as this is the only necessary interaction and because no heavy cryptographic tools such as zero-knowledge proofs are necessary. Secondly, we show how to modify the preprocessing phase of [26] to produce the necessary committed values in an identifiable way. To achieve the latter, we improve techniques by Ishai, Ostrovsky and Zikas [29] to avoid the use of adaptively secure OT. Our approach in doing so might be of independent interest.

*Concrete Efficiency.* We now expand on the concrete efficiency of our protocols and compare them with existing constant-round, non-identifiable protocols, as illustrated in Table 1. Note that the current most practical, constant-round MPC protocols are all obtained by combining garbling circuits with the so-called 'TinyOT' protocol [40], which combines OT extension and additive secret sharing with information-theoretic MACs over $\mathbb{F}_2$. The TinyOT part turns out to be the dominant, overall cost in the protocols, in terms of communication complexity. The parameter $B$ in Table 1 is related to a statistical security parameter used in cut-and-choose in TinyOT, and in practice is around 3–6. Using the most efficient multi-party variant of TinyOT [48] has a communication complexity of $O(n^2 B\kappa)$ bits per AND gate. The most efficient constant-round protocols have roughly the same communication complexity as TinyOT.

Our efficient protocol from Sections 3–4 uses TinyOT in a similar way to previous works, with the difference that we also use homomorphic commitments to obtain

[3] It is plausible that one could alternatively instantiate [36] with [5] as preprocessing, though this appears to yield a slower protocol as already the non-identifiable preprocessing of [36] has a larger overhead ($4n + 5$ SPDZ multiplications vs. 1 TinyOT-AND) plus the constructed circuit does not benefit from Free-XOR.

| Protocol | ID/PV | Based on | Assumptions | Communication |
|---|---|---|---|---|
| [26] | ✗ | OT + [30] | OT, free-XOR | $O((n^2\kappa + \mathsf{poly}(n))\|C\|)$ |
| [26] | ✗ | TinyOT | OT, free-XOR | $O(n^2B^2\kappa\|C\|)$ |
| [48] | ✗ | Optimized TinyOT | OT, free-XOR | $O(n^2B\kappa\|C\|)$ |
| Full version | ✓ / ✗ | [18] + [29] | adaptive OT, PRF | $\mathsf{bc}(\Omega(n^4 \cdot \|C\|))$ |
| Sections 3, 4, 5 | ✓ / ✓ | TinyOT + hom. commit. | OT, free-XOR | $O(n^2B\kappa\|C\|) + \mathsf{bc}(n^2\kappa\|C\|)$ |

**Table 1.** Efficiency of constant-round MPC protocols with and without identifiable abort, for a circuit with $|C|$ AND gates. ID/PV means identifiability or public verifiability. Communication complexity measured in total number of bits transmitted across the network; $\mathsf{bc}(n)$ is the cost of securely broadcasting $O(n)$ bits. The 'free-XOR' assumption is a circular 2-correlation robust hash function [14]

identifiability. While most constructions of publicly verifiable homomorphic commitments use public-key style assumptions like discrete log, we are able to get away with a weaker form of homomorphic commitment that only allows a bounded number of openings. This variant be based on any extractable commitment scheme [12], and the main computational cost is PRG evaluations and encodings of an error-correcting code, which can be implemented very efficiently, so we expect only a small computational overhead on top of the non-identifiable protocols. Additionally, the introduced communication overhead from these commitments (per gate) is expected to be a factor 2-3 over the communication that is necessary to perform the String-Oblivious Transfer required to garble a gate as in [26].

Regarding communication complexity, the main overhead in our protocol comes from creating and broadcasting homomorphic commitments to the $O(n \cdot |C|)$ wire keys in a BMR garbled circuit. We minimize this cost by using the efficient homomorphic commitments mentioned above, which have only a small constant communication overhead. Using this scheme, the overhead of commitments is not much more than the cost incurred from having each party broadcast its shares of the garbled circuit ($4n^2 \cdot \kappa|C|$ bits) at the end of our preprocessing phase. We remark that this broadcast step is not needed in non-identifiable protocols [48, 26], which can get away with reconstructing the garbled circuit towards a single party who then sends the sum of all shares.

To compare with existing non-constant round protocols such as [5, 46], we remark that these use lattice-based preprocessing. Such preprocessing is much more computationally expensive than our lightweight techniques based on OT extension. In terms of broadcasts, the offline phase of [5] has $O(n^3|C|\kappa)$ broadcast complexity, which is worse than our protocol. [46] does not describe the offline phase in detail, but it likely requires $O(n\kappa|C|)$ broadcasts for threshold decryption of the homomorphic encryption scheme. Regarding round complexity, even with the factor $n$ overhead when implementing broadcast, our protocol likely performs significantly better for complex functionalities with high-depth circuits. In general, [5, 46, 17] are for arithmetic circuits and likely applicable in different scenarios than ours, making a direct comparison difficult.

### 1.3 Technical Overview

In this overview, we assume some familiarity with garbled circuits and their use in MPC. For a more thorough introduction, we refer to the full version.

**Feasibility of constant-round ID-MPC.** To first establish a feasibility result, we use a variant of the garbling scheme from [18] combined with information-theoretic signatures [13, 25, 47], together with a compiler for sampling functionalities with identifiable abort from [29]. Although this construction is quite natural, we are not aware of it being described before.

In a little more detail, [18] is based on a garbling scheme where, similarly to BMR, when evaluating the garbled circuit, for each wire we obtain a vector of keys $(K_w^1, \ldots, K_w^n)$, where the component $K_w^i$ is known to party $P_i$. The garbling uses a specialized encryption scheme, which encrypts $K_w^i$ by first producing *verifiable secret shares* (VSS) $(K_w^i[1], \ldots, K_w^i[n])$ of $K_w^i$, and then encrypting each share $K_w^i[j]$ under the corresponding input wire key components of $P_j$, as:

$$E_{K_u,K_v}(K_w^i) := \begin{pmatrix} \mathsf{H}(K_u^1, K_v^1) \oplus K_w^i[1] \\ \vdots \\ \mathsf{H}(K_u^n, K_v^n) \oplus K_w^i[n] \end{pmatrix}$$

This is amenable to secure computation in a black-box way, as $P_j$ can input the hash values $H(K_u^j, K_v^j)$ to the protocol, and as long as the majority of these hash values are correct, which is guaranteed by an honest majority, the VSS allows correct reconstruction of $K_w^i$.

We adapt this to the dishonest majority setting by replacing VSS with additive secret-sharing and information-theoretic signatures. Roughly, we consider a preprocessing functionality which samples additive shares of each $K_w^i$ and augments each share with a signature under a signing key that no-one gets, while also allowing corrupt parties to choose their hash values for each gate. This suffices to obtain ID-MPC in an online phase, since if any corrupt party uses an incorrect hash value then the corresponding signature on their share will no longer verify.

To realize the preprocessing phase which outputs authenticated shares of the garbled circuit, we apply the compiler from [29], which transforms a protocol for any sampling functionality that is secure with abort, into one with identifiable abort. We remark that in the preprocessing functionality, the size of each garbled gate is $O(n^3 \cdot \kappa)$ bits, and the communication complexity of the protocol to generate this is at least $\Omega(n^4\kappa)$ due to overheads in [29], so this approach is not practical.

For space reasons, the complete description of these protocols can be found in the full version.

**Concretely efficient ID-MPC with BMR.** As mentioned before, our protocol follows the same approach of [26] ('HSS') based upon BMR garbled circuits. In BMR garbling, the vector of output wire keys $(K_w^1, \ldots, K_w^n)$ of a gate $g$ is directly encrypted under the input wire keys, with

$$E_{K_u,K_v}(K_w) := \bigoplus_{j=1}^{n} \mathsf{H}(g, K_u^j, K_v^j) \oplus (K_w^1, \ldots, K_w^n)$$

When using free-XOR with BMR, each pair keys on a wire is of the form $(K_{w,0}, K_{w,1} = K_{w,0} \oplus R)$ for some fixed string $R = (R^1, \ldots, R^n)$, with $R^i$ known to $P_i$. When garbling an AND gate with input wires $u, v$ and output wire $w$, we need to produce the 4 rows

$$\begin{aligned} \mathtt{circ}_{g,a,b} = &\bigoplus_{j=1}^{n} \mathsf{H}(g, K_{u,a}^j, K_{v,b}^j) \oplus (K_{w,0}^1, \ldots, K_{w,0}^n) \\ &\oplus (R^1, \ldots, R^n) \cdot ((\lambda_u \oplus a)(\lambda_v \oplus b) \oplus \lambda_w), \end{aligned} \tag{1}$$

for $(a, b) \in \{0, 1\}^2$, where $\lambda_u, \lambda_v, \lambda_w$ are the secret wire masks assigned to each wire.

In the HSS protocol, to generate additive shares of the above, each party $P_i$ first samples all of their key components and global string $R^i$, as well as secret shares of all the wire masks. Then, a generic MPC protocol for binary circuits is used to compute shares of the wire mask products $\lambda_u \cdot \lambda_v$, and shares of the products between each wire mask and every global string $R^i$ are computed using OT. This allows the parties to obtain additive shares of the entire garbled circuit, since each hash value in (1) can be computed locally by party $P_j$. If any party uses an incorrect hash value, it was shown in [26] that this would result in an abort in the online phase with overwhelming probability, since each party can check correctness when decrypting a gate by checking for the presence of one of their own key components.

*Identifiable online phase.* Adding identifiable abort to BMR is more challenging than with [18], since if any error is introduced to the hash values in (1), we have no direct way of knowing which party introduced it. Note that if the parties were committed to *the entirety* of the shares of the garbled circuit (i.e. all of (1)) then this would be straightforward: they could simply broadcast their shares, then attempt to run the online phase; if any party sends an incorrect share then the protocol aborts with overwhelming probability, and in our case everyone could then open their commitments to prove they behaved honestly. Unfortunately, we do *not* know how to efficiently create commitments to all of the shares, since in particular each share contains a hash value $\mathsf{H}(g, K_{u,a}^j, K_{v,b}^j)$, and it seems challenging to reliably commit to these without resorting to proving statements about hash function computations in zero-knowledge.

Instead, we observe that it is actually enough if each party is given commitments to *partial shares* of the garbled gates, namely, shares of the whole of (1) *except for* the hash values. To see this, consider that some party aborts at gate $g$ in the computation. If $g$ is the *first* (in topological order) such gate where the parties detect an inconsistency, then it must hold that the *preceding* gates were correctly garbled. This means that the wire keys from the previous gate can be used to compute the correct $\mathsf{H}(\cdot)$ values by every party. Hence, we can verify the garbling of $g$ by opening the commitments to the partial shares, then reconstructing the shares that should have been sent by 'filling in' the remaining parts of the garbled gate that were not committed to. Finally, the resulting shares can be compared with the shares that were actually sent, allowing us to detect a cheating party.

We therefore rely on a preprocessing functionality that adds XOR-homomorphic commitments to all the wire keys and shares of the bit-string products. Since the commitments are homomorphic, this easily allows computing commitments to the partial shares as required.

*Identifiable preprocessing phase.* Our first challenge with the preprocessing is to create the necessary commitments to the bit-string products in a reliable way. We show that without identifiability, this can be done without too much difficulty, using a consistency check based on a technique adapted from [26].

Next, the main challenge is to make the whole preprocessing identifiable. One possible approach would be to simply apply the same IOZ transformation we used for the protocol based on Damgård-Ishai, to convert a protocol $\Pi_{\mathsf{Prep}}$ that realizes the preprocessing functionality $\mathcal{F}_{\mathsf{Prep}}$ with abort into a new protocol $\Pi^{\mathsf{ID}}_{\mathsf{Prep}}$ that is identifiable. Unfortunately, this transformation has two main drawbacks: Firstly, the protocol $\Pi_{\mathsf{Prep}}$ needs to compute not only the outputs of $\mathcal{F}_{\mathsf{Prep}}$, but *authenticated secret shares* of these outputs, where each share has an information-theoretic signature attached to it; since IT signatures have a multiplicative $\Omega(n)$ storage overhead, this adds a significant cost burden to the protocol. Secondly, $\Pi_{\mathsf{Prep}}$ needs to be secure against *adaptive corruptions*, which is in general much harder to achieve than static corruptions; in particular, it rules out the use of efficient OT extensions unless we rely on the programmable random oracle model [39, 10].

We work around these issues with careful modifications to the [29] transformation, which are tailored specifically to our preprocessing phase. We first briefly recall the idea behind IOZ. To construct $\Pi^{\mathsf{ID}}_{\mathsf{Prep}}$, first each party commits to its randomness in $\Pi_{\mathsf{Prep}}$, and then if $\Pi_{\mathsf{Prep}}$ aborts, everyone simply opens their randomness, which is safe as the preprocessing phase is independent of the parties' inputs. The main challenge when proving security of this approach is that if the protocol aborts, the simulator needs to be able to convincingly open the honest parties' random tapes to the adversary, explaining the previously simulated protocol messages. This leads to the above two issues, since (1) if the protocol aborts *after* a corrupt party has seen its outputs, the simulator may not be able to produce honest parties' outputs that match, and (2) the simulator may not be able to come up with convincing honest parties' random tapes, since the previous honest parties' messages were simulated independently of the actual outputs from $\mathcal{F}^{\mathsf{ID}}_{\mathsf{Prep}}$ In IOZ, (1) is resolved by producing an authenticated secret-sharing of the outputs, and (2) is resolved by requiring $\Pi_{\mathsf{Prep}}$ to be adaptively secure.

In our work, we address (1) by ensuring that an abort is only possible in $\Pi_{\mathsf{Prep}}$ *before* the ideal functionality $\mathcal{F}_{\mathsf{Prep}}$ has delivered outputs to the honest parties. This means there is no danger of inconsistencies between the simulated honest parties' outputs and those seen by the distinguisher. Our method of resolving (2) is more complex. First, consider a simulation strategy where when running $\Pi_{\mathsf{Prep}}$ within $\Pi^{\mathsf{ID}}_{\mathsf{Prep}}$, the simulator simply performs an honest run of $\Pi_{\mathsf{Prep}}$ on random inputs. If $\Pi_{\mathsf{Prep}}$ later aborts, there is no problem opening the random tapes of honest parties', since the simulator knows these. The problem now is that the simulator can no longer extract any corrupt parties' inputs which may have to be sent to $\mathcal{F}_{\mathsf{Prep}}$, or ensure the corrupt parties get the corrupt output sent by $\mathcal{F}_{\mathsf{Prep}}$. To work around this, we combine $\Pi_{\mathsf{Prep}}$ with a homomorphic commitment scheme, and require that every party commits to all values used in $\Pi_{\mathsf{Prep}}$;

we ensure consistency of these commitments with the values in $\Pi_{\mathsf{Prep}}$ with a simple test where we open random linear combinations of the commitments, and modify the (reactive) protocol $\Pi_{\mathsf{Prep}}$ to open the same combinations. If the homomorphic commitment scheme is UC secure with identifiable abort, then the simulator can use this to extract and open the values in $\Pi_{\mathsf{Prep}}$, allow us to prove security of the whole protocol. A suitable commitment scheme can be efficiently constructed, building upon any (non-homomorphic) extractable commitment and a PRG [12].

We apply the above blueprint to the preprocessing phase of HSS, which performs multiplications between random bits, as well as between bits and random, fixed strings, to produce additive shares of the garbled circuit. With our transformation, the parties actually end up producing *homomorphic commitments* to shares of some (but not all) parts of the garbled circuit; namely, they are committed to the wire keys and the shares of the bit-string products from (1).

**Achieving public verifiability.** Public verifiability with identifiable abort requires not only that a party from the protocol can identify a cheater, but anyone can do so (or verify correctness of the result) by simply inspecting some messages posted to a public bulletin board. Adding this to our efficient construction requires modifying both the preprocessing and online phases of the protocol. First, we modify our preprocessing method so that the underlying protocol that is secure with abort satisfies a property called *public detectability*, which requires that an external verifier, who is given the random tapes of all parties in the protocol and all broadcast messages, can detect whether any cheating occurred and identify a corrupted party if so. This is similar to the concept of $\mathcal{P}$*-verifiability* used in IOZ [29], but removes the requirement that the verifier is also given the view of one honest party. We then show that any suitable, secure protocol can be transformed to be publicly detectable, with a simple transformation that is similar to the $\mathcal{P}$-verifiable transformation from [29]. Using the publicly detectable protocol in our identifiable preprocessing phase, and replacing the broadcast channel with a bulletin board, we obtain a publicly verifiable preprocessing protocol with identifiable abort.

To add public verifiability to the online phase, we need to ensure that an external evaluator can detect any cheating in the garbled circuit, given only the public transcript. It turns out that in case of abort, almost all of the computation done by an honest party when detecting a cheater relies only on public information; the only exception is the 0/1 wire values that are obtained when evaluating the garbled circuit, which each party computes by looking at its private keys. To allow an external verifier to compute these values, we modify the preprocessing with a variant of the point-and-permute technique, which encodes these values as the last bit in the corresponding key on that wire. Now if the protocol aborts, and the entire transcript of broadcast messages has been posted to the public bulletin board, the verifier has all the information that is needed to detect any inconsistency and identify a cheating party.

Notice that our public cheater identification is protocol-specific and does not require heavy NIZK machinery. This differentiates it from [32] who gave a general compiler that achieves publicly verifiable ID-MPC, but where the generated "cheating certificate" is a NIZK that has to re-compute the next-message function of the compiled protocol. That means that compiling a BMR-style protocol using their approach might require

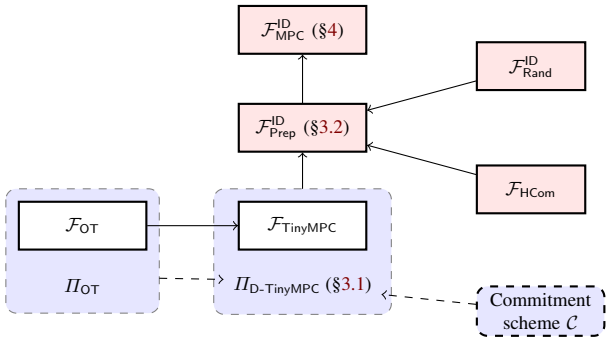


**Fig. 1.** Illustration of our efficient protocol with identifiable abort.

giving a zero-knowledge proof of correct garbling of the whole circuit, whereas our certificate just requires a few commitments to be opened.

**Paper Outline.** In Figure 1 we show the relationship between our protocols and functionalities in our main construction with identifiable abort. Section 3.1 contains our publicly detectable transformation, used for both identifiable abort and public verifiability, and instantiation from the OT-based preprocessing phase of [26]. Section 3.2 describes our identifiable preprocessing protocol, which uses the publicly detectable $\Pi_{\mathsf{D\text{-}TinyMPC}}$ in a non-black-box way (but with black-box use of its next-message function), and combines this with homomorphic commitments. In Section 4, we present the main MPC protocol with identifiable abort, which uses $\mathcal{F}_{\mathsf{Prep}}^{\mathsf{ID}}$ to create and then evaluate a BMR garbled circuit, with identifiable abort. In Section 5, we describe how to modify the previous protocol to additionally obtain public verifiability, using a bulletin board instead of a broadcast channel.

## 2 Preliminaries

Let $\kappa$ (resp. $s$) denote the computational (resp. statistical) security parameter. We let $\mathcal{P} = \{P_1, \ldots, P_n\}$ be the set of parties involved in any particular protocol/functionality, and $\mathcal{V}$ be a verifier which might check $\mathcal{P}$'s computation at a later point. Among those parties, we denote by $\mathcal{I} \subset \mathcal{P}$ the set of corrupted parties and by $\overline{\mathcal{I}} = \mathcal{P} \setminus \mathcal{I}$ the honest parties. Let $C_f$ be a circuit computing the function $f : \mathbb{F}_2^{n_{\mathtt{in}}} \to \mathbb{F}_2^{n_{\mathtt{out}}}$ with $n_{\mathtt{in}}$ inputs and $n_{\mathtt{out}}$ outputs. To ease the reading, we drop the dependence on $f$, when it is clear from the context. We will define the disjunct sets $\mathtt{input}_1, \ldots, \mathtt{input}_n \subset [n]$ as the inputs which each party in $\mathcal{P}$ provides to the circuit $C$, so $P_i$ provides the inputs in $\mathtt{input}_i$. The circuit $C$ has the set of AND gates $G$, for which we denote the extended set $G^{\mathtt{ext}} := G \times \mathbb{F}_2^2$. For $\tau \in G^{\mathtt{ext}}$, we usually denote $\tau = (g, a, b)$ where $g$ is the AND gate in question and $a, b \in \mathbb{F}_2$ are used to point to a specific entry in $g$'s (garbled) truth table.

### 2.1 Security Model and Primitives

We will prove security of our protocols in the universal composability (UC) framework [11]. We consider a static, active adversary corrupting up to $n-1$ parties. To achieve our goals, we will make use of multiple primitives, whose ideal functionalities we now introduce.

**Identifiable Abort Version of Functionalities.** In order to be able to rigorously discuss our protocols, we now formalize what it is to enhance their ideal functionalities $\mathcal{F}$ to support identifiable abort, which we denote by $\mathcal{F}^{\mathsf{ID}}$ and describe in Figure 2. As showed in [29], the UC composition theorem extends to security with identifiable abort in a straightforward way.

**Functionality $\mathcal{F}^{\mathsf{ID}}$**

Let $\mathcal{F}$ be a functionality which runs with parties $\mathcal{P} = \{P_1, \ldots, P_n\}$ and an adversary $\mathcal{A}$ who corrupts a subset $\mathcal{I} \subset \mathcal{P}$ of parties. $\mathcal{F}^{\mathsf{ID}}$ is exactly as $\mathcal{F}$, with the following extra command:

**Abort:** At any time, $\mathcal{A}$ can send a special command $(\texttt{Abort}, \mathcal{J})$ where $\mathcal{J} \subseteq \mathcal{I}, \mathcal{J} \neq \emptyset$. The functionality then stores $\mathcal{J}$, sends $(\texttt{Abort}, \mathcal{J})$ to parties in $\mathcal{P} \setminus \mathcal{I}$ and terminates the execution of any current command.

**Fig. 2.** Extending a functionality $\mathcal{F}$ to its identifiable abort version $\mathcal{F}^{\mathsf{ID}}$.

An $\mathcal{F}^{\mathsf{ID}}$ functionality is exactly as $\mathcal{F}$, but additionally allows the adversary to send a message $(\texttt{Abort}, \mathcal{J})$ at any point of time, where $\mathcal{J}$ denotes a non-empty set of dishonest parties. Upon receiving this message, the functionality ceases all computation and outputs the set $\mathcal{J}$ to all honest parties. The main points of identifiable abort are that (i) The adversary cannot abort without revealing the identity of at least one corrupt party; and (ii) All honest parties interacting with $\mathcal{F}^{\mathsf{ID}}$ agree on the revealed corrupted parties.

**Coin Tossing.** Coin tossing is used by a set of parties to fairly sample a number of coins according to a fixed distribution. In this work we will use an identifiable version of it, $\mathcal{F}_{\mathsf{Rand}}^{\mathsf{ID}}$, meaning that either all computing parties learn the sampled coins or, otherwise, the honest parties agree on a subset of dishonest parties who cheated in the sampling process. The standard $\mathcal{F}_{\mathsf{Rand}}$ functionality is described in the full version.

**Secure Broadcast.** Our work will crucially rely on the use of *secure* (or, *authenticated*) broadcast, which is a standard functionality given in the full version of the paper. Nevertheless, in order to achieve protocols with identifiable abort, we need to enhance the description of this functionality to $\mathcal{F}_{\mathsf{Broadcast}}^{\mathsf{ID}}$ as previously described. This is not a problem, as all standard protocols for $\mathcal{F}_{\mathsf{Broadcast}}$ such as [20, 43] are already identifiable. Under the assumption of a Public Key Infrastructure, implementing $\mathcal{F}_{\mathsf{Broadcast}}$ requires $\Omega(n)$ rounds of communication and signatures [23]. If the parties have access to an authenticated bulletin board, $\mathcal{F}_{\mathsf{Broadcast}}$ can be achieved with a single call to the board.

**Functionality $\mathcal{F}_{\mathsf{HCom}}$**

$\mathcal{F}_{\mathsf{HCom}}$ is parameterized by $\kappa \in \mathbb{N}$. $\mathcal{F}_{\mathsf{HCom}}$ interacts with a sender $P_S \in \mathcal{P}$, where the remaining parties $\mathcal{P} \setminus \{P_S\}$ act as receivers. $\mathcal{A}$ may corrupt any subset $\mathcal{I} \subsetneq \mathcal{P}$ and at any point it may send a message $(\texttt{Abort}, \mathcal{J})$ with $\emptyset \neq \mathcal{J} \subseteq \mathcal{I}$, upon which the functionality sends $(\texttt{Abort}, \mathcal{J})$ to $\mathcal{P}$ and halts.

**Commit:** Upon receiving $(\texttt{Commit}, \texttt{cid}, M)$ from $P_S$, where $M \in \mathbb{F}_2^\kappa$, save $(\texttt{cid}, M)$ locally and send $(\texttt{Commit-Recorded}, \texttt{cid})$ to $\mathcal{P}$ and $\mathcal{A}$. Every further message with this $\texttt{cid}$ to **Commit** is ignored.

**Add:** Upon receiving $(\texttt{Add}, \texttt{cid}_1, \texttt{cid}_2, \texttt{cid}_3)$ by $P_S$, where $(\texttt{cid}_1, M_1), (\texttt{cid}_2, M_2)$ are stored but not $\texttt{cid}_3$, add $(\texttt{cid}_3, M_1 + M_2)$ to the list and send $(\texttt{Add-Recorded}, \texttt{cid}_1, \texttt{cid}_2, \texttt{cid}_3)$ to $\mathcal{P}$ and $\mathcal{A}$.

**Open:** Upon receiving the first $(\texttt{Open}, \texttt{cid})$ by $P_S$ where $(\texttt{cid}, M)$ was previously stored, ignore all future messages to **Commit**. Send $(\texttt{Open}, \texttt{cid}, M)$ to all parties in $\mathcal{P}$ and $\mathcal{A}$.

**Fig. 3.** Functionality $\mathcal{F}_{\mathsf{HCom}}$ for homomorphic multiparty commitment with delayed verifiability.

**Homomorphic Commitments.** In this work, we use homomorphic commitments. These allow a sender to commit to a message $M$ at a certain time, such as to later open $M$ to a set of receivers. The properties required from commitment schemes are that (i) $M$ remains hidden to the receivers until the opening (hiding); and (ii) the sender can only open $M$ and no other value to the receivers, once committed (binding). We further require that the commitment scheme is homomorphic, meaning that the sender can open any linear combination of commitments that it made without revealing anything but the combined output. The functionality $\mathcal{F}_{\mathsf{HCom}}$ is described in Figure 3.

To efficiently implement $\mathcal{F}_{\mathsf{HCom}}$ we would like to use the homomorphic commitment scheme of Cascudo et al. [12], but it turns out that this is not possible directly. The problem is that $\mathcal{F}_{\mathsf{HCom}}$ (which we use throughout this work) allows to perform multiple rounds of **Add** and **Open**, whereas [12] permits to perform only one call to the interface **Open**. In the full version, we present a slightly weaker functionality $\mathcal{F}_{\mathsf{WHComm}}$ having multiple rounds of **Open** but not **Add**. We show that this is sufficient for our application and also how this weaker functionality can then be implemented using the protocol in [12].

## 3 Preprocessing Phase

Here we describe our preprocessing phase with identifiable abort. At a high level, we proceed in two steps: first, we describe a protocol with the weaker property of public detectability, and then we bootstrap it to a preprocessing protocol with identifiable abort using homomorphic commitments.

### 3.1 Publicly Detectable MPC with (Non-Identifiable) Abort

We start this section by recalling the notion of the (deterministic) *next message function*, $\mathsf{nmf}_{\Pi}^{i}$, of a party $P_i$ in an $n$-party protocol $\Pi$ that is executed in a limited number

of rounds, say $\rho$. Given the VIEW of $P_i$ at the beginning of round $h$, where $h \leq \rho$, i.e. the set $\text{VIEW}_h^i = (i, X, X_i, \mathsf{Rnd}_i, (M_{i,1}, \ldots, M_{i,h}))$, where $i$ identifies party $P_i$, $X$ is the common public input, $X_i$ and $\mathsf{Rnd}_i$ are $P_i$'s private input and randomness respectively, and $(M_{i,1}, \ldots, M_{i,h})$ are the messages received by $P_i$ in the first $h$ rounds, then $\mathsf{nmf}_{\Pi}^i(\text{VIEW}_h^i) = M_{h+1}^i$ are the messages that $P_i$ has to send in round $h+1$. In particular, $\mathsf{nmf}_{\Pi}^i(\text{VIEW}_\rho^i) = Y_i$, where $Y_i$ is $P_i$'s output, and $\text{VIEW}^i = \text{VIEW}_\rho^i$. In other words, the messages sent by each party $P_i$ at each round are deterministically specified as a function of $P_i$'s inputs and random coins, and messages received by $P_i$ in previous rounds.

We can now introduce the notion of *public detectability*. It is similar to that of $\mathcal{P}$-verifiability given in [29]. However, whereas the notion of $\mathcal{P}$-verifiability in that work was conceived with identifiable abort in mind, public detectability will allow us to implement functionalities not only achieving identifiable abort, but also public verifiability if required (see Section 5).

**Definition 3.1 (Public detectability)** *Let $\Pi$ be a protocol in the CRS model and $\mathcal{D}$ a deterministic poly-time algorithm, called the* detector*, which takes as inputs the CRS, the inputs and random tape of all parties in $\mathcal{P}$ involved in the execution of $\Pi$, and any message sent over an authenticated broadcast channel during the execution of $\Pi$. We say that the protocol $\Pi$ is* publicly detectable *if the detector $\mathcal{D}$ outputs a non-empty subset $\mathcal{J} \subset \mathcal{P}$ corresponding to (some of) the parties that did not honestly execute $\Pi$, if any such subset $\mathcal{J}$ exists.*

Notice there is a gap between the public detectability and identifiable abort properties: the latter requires that, upon abort, the adversary does not learn anything about the honest parties' inputs, beyond of what is deducible from the functionalities' output; on the other hand, running the detector requires access to all the input and random tape of $\mathcal{P}$. However, we will show, in Section 3.2, that public detectability is almost enough to define our preprocessing with identifiable abort, which we will later on extend to a public verifiable one in Section 5. At a high level, the main idea is that, since the goal of the preprocessing phase is to produce random correlated values that will be used in a very efficient online evaluation, during such a phase parties have not yet provided their private inputs, so, if the protocol aborts, it is enough for every party to run the detector on their own. The privacy of the overall MPC protocol is not affected then, due to the absence of the (actual) private inputs.

We now show how to turn any protocol $\Pi$ that UC-realises an ideal functionality $\mathcal{F}$ in the CRS model with static security, into a protocol $\Pi_V$ realising the same functionality with public detectability. Given the protocol $\Pi$, and a binding and hiding commitment scheme $\mathcal{C} = (\texttt{Commit}, \texttt{Reveal})$, we apply the following changes to $\Pi$.

- Before any step of $\Pi$ is executed, each party securely broadcasts a commitment to their input and random tape using the commitment scheme.[4]

[4] This part of the transformation is not actually needed in order to achieve public detectability, but it will simplify the way we use transformed protocols later on in order to achieve identifiable abort and public verifiability.

– In case of any broadcast communication, execute the protocol $\Pi$ using instead an authenticated broadcast functionality $\mathcal{F}_{\mathsf{Broadcast}}$ .
– Each pairwise communication between a sender $P_S$ and a receiver $P_R$, such that $\{P_S, P_R\} \subseteq \mathcal{P}$, is implemented by first securely broadcasting a commitment $c(M^S)$ to the message $M^S$ that has to be sent, followed by a private opening of it towards the receiving party. If $P_R$ does not receive the correct opening from $P_S$, then the receiver securely broadcasts a message asking for the opening of $c(M^S)$. The sender has to reply with that information, using also secure broadcast.[5] If the broadcasted reply is a correct opening, parties in $\mathcal{P}$ retake the computation, otherwise they abort.

It is easy to prove that the protocol $\Pi$, modified as above, is publicly detectable.

**Lemma 3.2** *Let $\Pi$ be a protocol that realises an ideal functionality $\mathcal{F}$ with static security in the* CRS *model with broadcast and pairwise communication, and $\mathcal{C}$ a standalone-secure commitment scheme. The protocol $\Pi_V$ described above is publicly detectable and realises the functionality $\mathcal{F}$ in the $\{$CRS,$\mathcal{F}_{\mathsf{Broadcast}}\}$-hybrid model.*

**Publicly Detectable Preprocessing.** In our preprocessing phase, we use the functionality $\mathcal{F}_{\mathsf{TinyMPC}}$ (Figure 4), which is a standard functionality for secret sharing-based MPC for binary circuits augmented with the command **MultBitString** that allows multiplying a bit by a fixed string known to one party. This functionality is exactly what is needed to securely preprocess a BMR garbled circuit [26] with abort, where the fixed strings play the roles of the global $R^i$ strings in the garbled circuit; this can be efficiently implemented using a TinyOT-like protocol, for example [40, 21, 26], in the $\mathcal{F}_{\mathsf{OT}}$-hybrid model.

We can apply the transformation above to obtain a publicly detectable protocol $\Pi_{\mathsf{D\text{-}TinyMPC}}$, if we have a protocol $\Pi_{\mathsf{TinyMPC}}$ that implements the functionality $\mathcal{F}_{\mathsf{TinyMPC}}$ in the CRS model. Such a protocol can be efficiently obtained by implementing the OT functionality with OT extension [31, 1], with base OTs realized in the CRS model [42]. Thus, we obtain the following corollary.

**Corollary 3.3** *Let $\mathcal{C}$ be a commitment scheme and $\Pi_{\mathsf{TinyMPC}}$ a protocol that UC-realises the functionality $\mathcal{F}_{\mathsf{TinyMPC}}$ in the CRS model. The protocol $\Pi_{\mathsf{D\text{-}TinyMPC}}$ (described in in the full version) is publicly detectable and it securely realises the functionality $\mathcal{F}_{\mathsf{TinyMPC}}$ in the $\{\mathcal{F}_{\mathsf{Broadcast}}$, CRS$\}$-hybrid model.*

### 3.2 Implementing the Preprocessing with Identifiable Abort

We now combine the detectable protocol $\Pi_{\mathsf{D\text{-}TinyMPC}}$ with homomorphic commitments, $\mathcal{F}_{\mathsf{HCom}}$, to obtain a preprocessing protocol with identifiable abort. Our preprocessing functionality $\mathcal{F}_{\mathsf{Prep}}^{\mathsf{ID}}$ is described in the full version of this work. It essentially performs the same computations as $\mathcal{F}_{\mathsf{TinyMPC}}$, except the output shares of the bit-string multiplication are now committed with homomorphic commitments, modelled in the functionality by allowing them to be added together and opened. Another key difference

[5] This does not break security, because such a situation can only occur if $P_S$ or $P_R$ are corrupted, in which case $\mathcal{A}$ would obtain $M^S$ anyway.

**Functionality $\mathcal{F}_{\mathsf{TinyMPC}}$**

The functionality runs with parties $P_1, \ldots, P_n$ and an adversary $\mathcal{A}$. It has a list of corrupt parties $\mathcal{I}$ which it obtains from $\mathcal{A}$.
Angle brackets $\langle x \rangle$ denote a secret $x \in \mathbb{F}_2$ stored by the functionality, together with a public identifier. The inputs to every command below are public inputs that must be provided by all parties (where in this case, the notation $\langle x \rangle$ refers only to the *identifier* of the secret value $x$).

**Init:** On input $(\texttt{Init})$ from all parties, if $(\texttt{Init})$ was received before then do nothing. For each $i \in [n]$, if $i \in \mathcal{I}$ then receive $R^i \in \mathbb{F}_2^\kappa$ from $\mathcal{A}$, otherwise sample a random $R^i \leftarrow \mathbb{F}_2^\kappa$. Send $R^i$ to party $P_i$ and store the strings $R^i$.

**Input:** On input $(\texttt{Input}, P_i, \langle x \rangle)$ from all parties and $(\texttt{Input}, P_i, \langle x \rangle, x)$ from $P_i$, where $x \in \mathbb{F}_2$, store $x$.

**Add:** On input $(\texttt{Add}, \langle z \rangle, \langle x \rangle, \langle y \rangle)$ from all parties, where the two bits $x$ and $y$ were previously stored, store $z = x + y$

**Mult:** On input $(\mathsf{Multiply}, \mathbb{F}_2, \langle \bar{x} \rangle, \langle x_1 \rangle, \langle x_2 \rangle)$ from all parties, where $x_1, x_2$ were stored previously, store $\bar{x} = x_1 \cdot x_2$.

**MultBitString:** On input $(\texttt{MultBitString}, \langle x \rangle, P_i)$ from all parties, where $x$ was stored previously:

1. $\mathcal{A}$ inputs $W^j \in \mathbb{F}_2^\kappa$ for each $P_j \in \mathcal{I}$.
2. Sample $W^j \leftarrow \mathbb{F}_2^\kappa$ for $j \in \overline{\mathcal{I}}$ subject to the constraint that $x \cdot R^i = \sum_{j \in [n]} W^j$.
3. Send $W^j$ to party $P_j$.

**Open:** On input $(\texttt{PublicOutput}, \langle x_1 \rangle, \ldots, \langle x_m \rangle)$ from all parties, where $x_i, i \in [m]$, have been stored previously:

1. Send $(\texttt{Deliver}, x_1, \ldots, x_m)$ to $\mathcal{A}$.
2. If $\mathcal{A}$ sends $\texttt{Abort}$, forward $\texttt{Abort}$ to all parties and halt. Otherwise send $(\texttt{Output}, x_1, \ldots, x_m)$ to all parties.

**Fig. 4.** Functionality $\mathcal{F}_{\mathsf{TinyMPC}}$ for Bit-MPC.

is that the outputs are only delivered at the very end of the protocol. After the initial outputs are sent to the parties, the only further allowed command is to open values from the homomorphic commitment scheme. This is because in the security proof for our preprocessing protocol, the simulator can always equivocate $\mathcal{F}_{\mathsf{HCom}}$, whereas it cannot equivate the simulation of $\Pi_{\mathsf{D\text{-}TinyMPC}}$ when it is still possible for an abort to occur (which would require opening the honest parties' random tapes to the adversary).

The protocol $\Pi^{\mathsf{ID}}_{\mathsf{Prep}}$, implementing $\mathcal{F}^{\mathsf{ID}}_{\mathsf{Prep}}$ and described in Figure 5, uses the publicly detectable version of $\Pi_{\mathsf{TinyMPC}}$ (from Corollary 3.3) for all the $\mathbb{F}_2$-arithmetic, i.e. to perform secure additions and multiplications on bits, as well as to obtain secret-shares of the product of secret bits with the strings $R^i$. The protocol uses two copies of the homomorphic commitment functionality (which we name $\mathcal{F}_{\mathsf{HCom}}$ and $\mathcal{F}^{\mathtt{Bit}}_{\mathsf{HCom}}$). The first copy is used to create commitments to values in $\mathbb{F}_2^\kappa$, such as the fixed strings $R^i$ as well as the additive shares of all the bit-string products of secret bits with $R^i$. We furthermore employ a consistency check to verify that the committed bit-string shares are correct, which is shown in Figure 6. The functionality $\mathcal{F}^{\mathtt{Bit}}_{\mathsf{HCom}}$ is used to additionally commit to the bits which are used in $\Pi_{\mathsf{D\text{-}TinyMPC}}$, and we use a second consistency check to verify

that these two sets of bits stored in $\Pi_{\mathsf{D\text{-}TinyMPC}}$ and $\mathcal{F}_{\mathsf{HCom}}^{\mathtt{Bit}}$ are the same; this can be found in Figure 7. We also use this functionality to open bit values during the output phase. We remark that the necessity of using $\mathcal{F}_{\mathsf{HCom}}^{\mathtt{Bit}}$ for both of this is an artefact of the proof and we leave it as an interesting open problem to remove $\mathcal{F}_{\mathsf{HCom}}^{\mathtt{Bit}}$ (together with the consistency check $\Pi_{\mathsf{CheckBit}}$) while retaining a provably-secure protocol.

In the case of abort, we will reveal all random tapes and committed messages of $\Pi_{\mathsf{D\text{-}TinyMPC}}$ and test which party has sent inconsistent messages and when. Interestingly, we can do that without requiring adaptive primitives (in comparison to previous works): The simulation of $\Pi_{\mathsf{D\text{-}TinyMPC}}$ in the security proof is only ever checked using the public detectability if no output of $\mathcal{F}_{\mathsf{Prep}}^{\mathsf{ID}}$ has been revealed yet. Therefore we do not have to ever equivocate the random tape during the simulation of $\Pi_{\mathsf{D\text{-}TinyMPC}}$ - revealing the tape used by the simulator is enough. This is exactly where previous work [29] required adaptivity of the underlying primitives, which we in our case can then avoid. To prove consistency of the committed shares of bit-string products, we use the following lemma. Its statement and proof are very similar to [26, Lemma 3.1] and we only provide it here for completeness. The proof of the lemma is given in the full version.

**Lemma 3.4** *If the protocol $\Pi_{\mathsf{BitStringMult}}$ does not abort, then the committed values $R^j, W^i_{\tau,j}$ produced by $\Pi_{\mathsf{BitStringMult}}$ satisfy*

$$\sum_{i=1}^{n} W^i_{\tau,j} = x_\tau \cdot R^j,$$

*where $R^j$ was computed in the **Init** phase and $\langle x_1 \rangle, \ldots, \langle x_m \rangle$ were input to $\Pi_{\mathsf{BitStringMult}}$, except with probability* $\max(\varepsilon, 2^{-s})$.

We use the lemma below to verify that bits committed in both $\mathcal{F}_{\mathsf{HCom}}^{\mathtt{Bit}}$ and $\Pi_{\mathsf{D\text{-}TinyMPC}}$ are consistent. We omit the proof, which is essentially a simplified version of the proof of Lemma 3.4.

**Lemma 3.5** *If the protocol $\Pi_{\mathsf{CheckBit}}$ does not abort, then the inputs $\langle x_1 \rangle, \ldots, \langle x_m \rangle$, $[x^i_1]^{\mathtt{Bit}}_c, \ldots, [x^i_m]^{\mathtt{Bit}}_c$ satisfy*

$$\sum_{i=1}^{n} x^i_j = x_j,$$

*except with probability* $\max(\varepsilon, 2^{-s})$.

These allow us to prove security of our construction.

**Theorem 3.6** *Assuming a secure broadcast channel, protocol $\Pi_{\mathsf{Prep}}^{\mathsf{ID}}$ implements $\mathcal{F}_{\mathsf{Prep}}^{\mathsf{ID}}$ in the $\{\text{CRS}, \mathcal{F}_{\mathsf{HCom}}, \mathcal{F}_{\mathsf{HCom}}^{\mathtt{Bit}}, \mathcal{F}_{\mathsf{Rand}}\}$-hybrid model with security against a static, active adversary corrupting at most $n-1$ parties.*

The proof follows a simulation-based argument, along the lines described in Section 1.3. It is given in the full version.

**The Preprocessing Protocol $\Pi^{\mathsf{ID}}_{\mathsf{Prep}}$**

The protocol runs with parties $P_1, \ldots, P_n$. It runs two instances of the homomorphic commitment scheme which we denote as $\mathcal{F}_{\mathsf{HCom}}$ and $\mathcal{F}^{\mathtt{Bit}}_{\mathsf{HCom}}$.
NOTATION: We use $\langle x \rangle$ to denote that $x \in \mathbb{F}_2$ is stored by $\Pi_{\mathsf{D\text{-}TinyMPC}}$, $[x]^{\mathtt{Bit}}_i$ to denote that $x \in \mathbb{F}_2$ is stored in $\mathcal{F}^{\mathtt{Bit}}_{\mathsf{HCom}}$ where $P_i$ is the sender, and $[X]_i$ to denote that $X \in \mathbb{F}_2^\kappa$ is stored in $\mathcal{F}_{\mathsf{HCom}}$ where $P_i$ is the sender. The parties maintain a list $\mathsf{PubOutputs}$ which is initially empty.

PHASE I: COMPUTATION

**Init:** The parties call $\Pi_{\mathsf{D\text{-}TinyMPC}}$ with input $(\mathtt{Init})$ and each obtain a string $R^i \in \mathbb{F}_2^\kappa$. Then they each commit to $R^i$ using $\mathcal{F}_{\mathsf{HCom}}$.

**Sample:** Party $P_i$ samples $x \leftarrow \mathbb{F}_2$ or $X \leftarrow \mathbb{F}_2^\kappa$. The parties do one of:
- $x \in \mathbb{F}_2$: use the $(\mathtt{Input})$ command of $\Pi_{\mathsf{D\text{-}TinyMPC}}$ to obtain $\langle x \rangle$. $P_i$ then commits to $x$ by calling $\mathcal{F}^{\mathtt{Bit}}_{\mathsf{HCom}}$ on input $(\mathtt{Commit}, \mathtt{id}, x)$, obtaining $[x]^{\mathtt{Bit}}_i$.
- $X \in \mathbb{F}_2^\kappa$: $P_i$ commits to $X$ by calling $\mathcal{F}_{\mathsf{HCom}}$, to obtain $[X]_i$.

**AddBit:** To add two bits $\langle x \rangle$ and $\langle y \rangle$, parties use the **Add** command of $\Pi_{\mathsf{D\text{-}TinyMPC}}$. If commitments $\{[x]^{\mathtt{Bit}}_i, [y]^{\mathtt{Bit}}_i\}_i$ are also stored in $\mathcal{F}^{\mathtt{Bit}}_{\mathsf{HCom}}$, use the **Add** command of $\mathcal{F}^{\mathtt{Bit}}_{\mathsf{HCom}}$.

**AddString:** To add the committed strings $[X]_i$ and $[Y]_i$, the parties use $\mathcal{F}_{\mathsf{HCom}}$.

**Mult:**
- To multiply two bits $\langle x \rangle$ and $\langle y \rangle$, the parties use the **Multiply** command of $\Pi_{\mathsf{D\text{-}TinyMPC}}$.
- To multiply the bit $\langle x \rangle$ with the strings $R^1, \ldots, R^n$, the parties run the subprotocol $\Pi_{\mathsf{BitStringMult}}$.

**Public Output (bit):** On input $(\mathtt{PublicOutput}, [x]^{\mathtt{Bit}}_1, \ldots, [x]^{\mathtt{Bit}}_n)$ from all parties, append $\{[x]^{\mathtt{Bit}}_i\}_i$ to the list $\mathsf{PubOutputs}$.

**Delayed Outputs:** On input $(\mathtt{DelayedOutputs})$ from all parties:
- Run **Check** of the subprotocol $\Pi_{\mathsf{CheckBit}}$ on input all of the bits committed to during **Sample**, to check their consistency.
- Use the $(\mathtt{Open})$ command of $\mathcal{F}^{\mathtt{Bit}}_{\mathsf{HCom}}$ to output all the $x_i$ values, for every tuple $\{[x]^{\mathtt{Bit}}_i\}^n_{i=1} \in \mathsf{PubOutputs}$, and compute the public value $x = x_1 \oplus \cdots \oplus x_n$.
- Each party $P_i$ outputs the list of public openings, together with all private values that $P_i$ stored during **Sample** or the bit-string **Mult** step.

PHASE II: FINAL OUTPUT

**Public Output (String):** To output the committed string $[X]_j$, $P_j$ uses the $\mathtt{Open}$ command of $\mathcal{F}_{\mathsf{HCom}}$. If $P_j$ fails to open any commitment, the parties output $(\mathtt{Abort}, P_j)$.

**Abort:** If the $\Pi_{\mathsf{D\text{-}TinyMPC}}$ protocol aborts:
1. Every party opens its commitments to its random tape of $\Pi_{\mathsf{D\text{-}TinyMPC}}$.
2. Run the detector $\mathcal{D}$ of $\Pi_{\mathsf{D\text{-}TinyMPC}}$. If $\mathcal{D}$ outputs that $\mathcal{J} \subset [n]$ cheated then output $(\mathtt{Abort}, \mathcal{J})$.

**Fig. 5.** The preprocessing protocol $\Pi^{\mathsf{ID}}_{\mathsf{Prep}}$.

### 3.3 Efficiency Analysis

The main overhead of our preprocessing protocol, compared with the non-identifiable protocol which we build upon [26], is due to the use of secure broadcast in the compila-

**Subprotocol $\Pi^m_{\mathsf{BitStringMult}}$**

The subprotocol uses the functionalities $\mathcal{F}_{\mathsf{HCom}}$, $\mathcal{F}_{\mathsf{Rand}}$, and the protocol $\Pi_{\mathsf{D\text{-}TinyMPC}}$.
We let $s$ denote a statistical security parameter.
INPUTS: Bits $\langle x_1\rangle, \ldots, \langle x_m\rangle$, and strings $[R^1]_1, \ldots, [R^n]_n$, where party $P_i$ has $R^i$.
OUTPUT: Shares of the bit-string products $W_{\tau,j} = x_\tau \cdot R^j$, for $\tau \in [m], j \in [n]$, and commitments to every party's share of $W_{\tau,j}$ under $\mathcal{F}_{\mathsf{HCom}}$.

**I: Init:** The parties sample $s$ additional random bits.

1. Each $P_i$ calls **Input** on $\Pi_{\mathsf{D\text{-}TinyMPC}}$ with $s$ random bits $(\hat{x}^i_1, \ldots, \hat{x}^i_s)$. Compute the shared bits $\langle \hat{x}_\tau \rangle = \sum_{i\in[n]} \langle \hat{x}^i_\tau \rangle$ using $\Pi_{\mathsf{D\text{-}TinyMPC}}$.
2. Write $X = (x_1, \ldots, x_m)$ and $\hat{X} = (\hat{x}_1, \ldots, \hat{x}_s)$ and define $\langle X\rangle, \langle \hat{X}\rangle$ accordingly.

**II: Multiply:** For each $j \in [n]$, the parties do as follows:

1. Call $\Pi_{\mathsf{D\text{-}TinyMPC}}$ on input $(\texttt{MultBitString})$ to obtain random shares $W^i_{\tau,j}$ of $W_{\tau,j} = x_\tau \cdot R^j$, and shares $\hat{W}^i_{\tau,j}$ of $\hat{x}_\tau \cdot R^j$.
2. Write $W^i_j \in (\mathbb{F}_2^\kappa)^m$ as $P_i$'s shares of $X \cdot R^j$, and $\hat{W}^i_j \in (\mathbb{F}_2^\kappa)^s$ for the shares of $\hat{X} \cdot R^j$.

**III: Commit:** Each party $P_i$ commits to $W^i_j$ and $\hat{W}^i_j$ using $\mathcal{F}_{\mathsf{HCom}}$, for each $j \in [n]$.

**IV: Check:** The parties check correctness of the commitments as follows:

1. The parties call $\mathcal{F}_{\mathsf{Rand}}$ to sample a seed for a uniformly random, $\varepsilon$-almost 1-universal linear hash function, $\mathbf{H} \in \mathbb{F}_2^{s\times m}$.
2. All parties compute the vector:
$$\langle C_x\rangle = \mathbf{H} \cdot \langle X\rangle \ + \langle \hat{X}\rangle \in \mathbb{F}_2^s$$
and open $C_x$ using **Open** of $\Pi_{\mathsf{D\text{-}TinyMPC}}$. If $\Pi_{\mathsf{D\text{-}TinyMPC}}$ aborts, the parties run the **Abort** phase of $\Pi^{\mathsf{ID}}_{\mathsf{Prep}}$.
3. For each $i \in [n]$, the parties use $\mathcal{F}_{\mathsf{HCom}}$ to obtain commitments to the vectors in $(\mathbb{F}_2^\kappa)^s$:
$$[C^i_j]_i = \mathbf{H} \cdot [W^i_j]_i + [\hat{W}^i_j]_i \ \text{ for } j \neq i, \text{ and } \quad [C^i_i]_i = \mathbf{H} \cdot [W^i_i]_i + [\hat{W}^i_i]_i + C_x \cdot [R^i]_i.$$
Each $P_i$ then opens its commitments to $C^i_j$, for $j \in [n]$.
4. All parties check that, for each $j \in [n]$, $\sum_{i=1}^n C^i_j = 0$. If any check fails, the parties go to the **Abort** phase below.
5. The parties output the shares $W^i_{\tau,j}$, and commitments $[W^i_{\tau,j}]_i$.

**Abort:** If Step 4 of **Check** fails, the parties do as follows:

1. If $\mathcal{F}_{\mathsf{Rand}}$ outputs $(\texttt{Abort}, \mathcal{J})$ then all parties output this. If not, continue.
2. Every party opens its commitments to its random tape of $\Pi_{\mathsf{D\text{-}TinyMPC}}$.
3. Using the opened random tapes, transcript and CRS of $\Pi_{\mathsf{D\text{-}TinyMPC}}$, compute each party's shares $W^i_j$ and $\hat{W}^i_j$, which were obtained after the **Multiply** step.
4. Let $\mathcal{J} \subset [n]$ be the set of indices $i \in [n]$ for which $C^i_j \neq \mathbf{H} \cdot W^i_j + \hat{W}^i_j$.
5. Output $(\texttt{Abort}, \mathcal{J})$.

**Fig. 6.** Subprotocol $\Pi^m_{\mathsf{BitStringMult}}$ to check consistency of committed bit-string multiplications.

tion with Lemma 3.2, and the use of homomorphic commitments for every wire in the

**Subprotocol $\Pi_{\mathsf{CheckBit}}$**

The subprotocol uses the functionalities $\mathcal{F}_{\mathsf{HCom}}^{\mathtt{Bit}}$, $\mathcal{F}_{\mathsf{Rand}}$, and the protocol $\Pi_{\mathsf{D\text{-}TinyMPC}}$.
NOTATION: We use $\langle x \rangle$ to denote that $x \in \mathbb{F}_2$ is stored by $\Pi_{\mathsf{D\text{-}TinyMPC}}$, and $[x]_c^{\mathtt{Bit}}$ to denote $x \in \mathbb{F}_2$ that is stored in $\mathcal{F}_{\mathsf{HCom}}^{\mathtt{Bit}}$. We let $s$ denote a statistical security parameter.
INPUTS: Bits $\langle x_1 \rangle, \ldots, \langle x_m \rangle$ stored using $\Pi_{\mathsf{D\text{-}TinyMPC}}$, and $\mathcal{F}_{\mathsf{HCom}}^{\mathtt{Bit}}$ commitments $[x_1^i]_c^{\mathtt{Bit}}, \cdots, [x_m^i]_c^{\mathtt{Bit}}$, for $i \in [n]$, where $P_i$ committed to the values $x_j^i$.
OUTPUT: The protocol succeeds if $x_j = \sum_i x_j^i$, for $j \in [m]$.

**Check:**

1. Each party $P_i$ samples $s$ random bits $\hat{x}_1, \ldots, \hat{x}_s \leftarrow \mathbb{F}_2$.
2. $P_i$ inputs $\hat{x}_j^i$ into $\Pi_{\mathsf{D\text{-}TinyMPC}}$, and commits to $\hat{x}_j^i$ with $\mathcal{F}_{\mathsf{HCom}}^{\mathtt{Bit}}$, for $j \in [m]$.
3. Using $\mathcal{F}_{\mathsf{Rand}}$, the parties sample a random $\varepsilon$-almost 1-universal hash function $\mathbf{H} \in \mathbb{F}_2^{s \times m}$.
4. Writing $[X^i]_c^{\mathtt{Bit}} = ([x_1^i]_c^{\mathtt{Bit}}, \ldots, [x_m^i]_c^{\mathtt{Bit}})$, $[\hat{X}^i]_c^{\mathtt{Bit}} = ([\hat{x}_1^i]_c^{\mathtt{Bit}}, \ldots, [\hat{x}_s^i]_c^{\mathtt{Bit}})$, and similarly for $\langle X \rangle, \langle \hat{X} \rangle$, compute
$$[C_1^i]_c^{\mathtt{Bit}} = \mathbf{H} \cdot [X^i]_c^{\mathtt{Bit}} + [\hat{X}^i]_c^{\mathtt{Bit}}, \quad \langle C_2 \rangle = \mathbf{H} \cdot \langle X \rangle + \langle \hat{X} \rangle$$
5. The parties open $C_1^i$ and $C_2$ using $\mathcal{F}_{\mathsf{HCom}}^{\mathtt{Bit}}$ and $\Pi_{\mathsf{D\text{-}TinyMPC}}$, respectively, and check that $\sum_i C_1^i = C_2$. If the check fails, the parties go to **Abort**.

**Abort:**

1. All parties open their $\langle x_j \rangle$-shares as $x_j^i$ using $\Pi_{\mathsf{D\text{-}TinyMPC}}$, and each $P_i$ opens its $[x_j^i]_c^{\mathtt{Bit}}$ values as $y_j^i$ using $\mathcal{F}_{\mathsf{HCom}}^{\mathtt{Bit}}$.
2. Let $\mathcal{J} \subset [n]$ be the set of indices $i$ for which there exists a $j \in [m]$ such that $x_j^i \neq y_j^i$.
3. Output $(\mathtt{Abort}, \mathcal{J})$.

**Fig. 7.** Subprotocol $\Pi_{\mathsf{CheckBit}}$ to check consistency of committed bits.

garbled circuit. We now discuss these costs in more detail, and describe an optimization to reduce the use of broadcast.

We first observe that we can run an *optimistic* version of $\Pi_{\mathsf{Prep}}^{\mathsf{ID}}$, without the additional broadcasts in Lemma 3.2. If this optimistic offline protocol ends successfully (i.e. no abort occurs during Phase I), then the online phase will not require the identifiability property of $\Pi_{\mathsf{Prep}}^{\mathsf{ID}}$. In case of an error in the optimistic $\Pi_{\mathsf{Prep}}^{\mathsf{ID}}$ instance, we then re-run the preprocessing with new randomness for each party and using the identifiable version with broadcast. Observe that this may not identify a cheater in case the second protocol succeeds, but this does not contradict the definition of identifiable abort as the adversary now just forces the honest parties to use more resources. At the same time, the fact that the real inputs at this point were not used for any computation and due to the use of new randomness by each party, privacy is preserved.

Regarding the homomorphic commitments, note that the dominating cost is during the bit-string multiplication ($\Pi_{\mathsf{D\text{-}TinyMPC}}$), where each of the $n$ parties must commit to $n \cdot m$ strings of length $\kappa$, where $m$ is approximately the number of AND gates. Computationally, this overhead is very cheap. For instance, based on the implementation from [44] (which is similar to the non-interactive scheme we use) as a rough estimate for computation, then for $n = 5$ parties and the AES circuit with 6800 AND gates, we

estimate the cost of generating commitments in a WAN setting is around 0.3 seconds. Compared with [48], this is only around a 10% overhead. Therefore, the main cost introduced in this stage is likely having to securely broadcast these commitments. Since for large circuits this involves the broadcast of very long strings, broadcast "extension" techniques such as [22] may be useful to reduce the complexity.

## 4 Online Phase

The online phase of our computation is modeled after [26]. For completeness, in the full version we outline the main idea of constant-round MPC, so here we focus on the details necessary in order to achieve identifiable abort. HSS uses a BMR-approach to achieve constant-round MPC, where the parties can identify locally if their output was correct or not. In order to identify a cheater, we will perform a reconstruction of a faulty gate in case of an error in that gate. This reconstruction, as it turns out, does not reveal any information beyond the regular protocol transcript. To be able to perform such a reconstruction we will use the new properties of our enhanced preprocessing functionality $\mathcal{F}_{\mathsf{Prep}}^{\mathsf{ID}}$, namely that it also provides a verification mechanism for keys. This reconstruction consists of opening commitments to the input keys to the gate which all parties used for evaluation as well as the supposed output key using $\mathcal{F}_{\mathsf{Prep}}^{\mathsf{ID}}$.

There are multiple cases in which aborts can happen during the online phase, which entails different ways of how the cheater must be identified. These cases are as follows:

1. *A set of parties can stop to communicate*. As our protocol (apart from $\mathcal{F}_{\mathsf{Prep}}^{\mathsf{ID}}$) only uses broadcast communication, such behaviour identifies cheaters directly.
2. *The adversary can manipulate a gate in such a way that an honest party sees an error.* This is the most straightforward error from garbling and can directly be detected by reconstructing the gate. In the protocol, we detect this in Steps 5, 6 of **Abort**. There, we open the keys which correspond to the output of the gate in $\mathcal{F}_{\mathsf{Prep}}^{\mathsf{ID}}$ that we would expect to see, based on **Init**. We can then compare the opening with the actual outputs which were derived based on $\texttt{share}_\tau^j$ (the garbling information) from each party, which directly shows whenever $\mathcal{A}$ garbled a circuit wrongly towards any party.
3. *The adversary may send complaints, even though a gate was garbled correctly.* In this case we reconstruct the gate normally and, as no error occurs, we identify complaining parties as cheaters. This is the end of Step 6 of **Abort**.
4. *The adversary may send complaints about incorrect gates that are outside the "active path".* Here we will show that the honest parties agree on the active path, i.e. on the rows of each garbled table which they decrypt during evaluation or - equivalently - the public values $\Lambda_w$. Thus, honest parties can identify complaints outside the active path as cheating, which is done in Step 2 of **Abort**.
5. *The adversary can garble a gate incorrectly for a dishonest party and let that party not report this.* In this case, the protocol will only abort at the next AND gate $g'$ into which the output of the wrongly garbled gate $g$ is fed. In such a case we will see a difference between the keys that the honest parties obtained as input of $g$ and the keys that are opened by the dishonest party that did not send a `Conflict`-message. We do this by opening the committed input keys $[K^j_{u,\Lambda_u}]_j, [K^j_{v,\Lambda_v}]_j$ in Step 5 via

**The MPC Protocol - $\Pi^{\mathsf{ID}}_{\mathsf{MPC}}$ (Initialization)**

COMMON INPUTS: A hash function $\mathsf{H} : G \times \mathbb{F}_2^{2\kappa} \to \mathbb{F}_2^{n\kappa}$ and a circuit $C_f$ representing the function $f$. Let the input wires of a gate be labeled $u$ and $v$, and the output wire be $w$. Let $\Lambda_u$ and $\Lambda_v$ be the public values on the input wires. The protocol uses an instance of $\mathcal{F}^{\mathsf{ID}}_{\mathsf{Prep}}$.
PRIVATE INPUTS: $\rho = (\rho_1, \ldots, \rho_{n_{\mathtt{in}}})$, where $\{\rho_h\}_{h \in \mathtt{input}_i}$ is party $P_i$'s input.
If a set of parties $\mathcal{J}$ does not send messages during the protocol, then each party outputs $(\mathtt{Abort}, \mathcal{J})$ and stops. If $\mathcal{F}^{\mathsf{ID}}_{\mathsf{Prep}}$ at any point outputs $(\mathtt{Abort}, \mathcal{J})$ then each party outputs $(\mathtt{Abort}, \mathcal{J})$ and stops.

**Init:**

1. Each party $P_i$ sends $(\mathtt{Init})$ to $\mathcal{F}^{\mathsf{ID}}_{\mathsf{Prep}}$, which in turn outputs $[R^1]_1, \ldots, [R^n]_n$.
2. Passing topologically through all the wires $w \in W$ of the circuit:
   - If $w \in \mathtt{input}_h$:
     (a) Each $P_i$ sends $\mathtt{Sample}(\mathbb{F}_2, P_h)$ to $\mathcal{F}^{\mathsf{ID}}_{\mathsf{Prep}}$ which outputs $\langle \lambda_w \rangle$.
     (b) For each $j \in [n]$ each $P_i$ sends $\mathtt{Sample}(\mathbb{F}_2^\kappa, P_j)$ to $\mathcal{F}^{\mathsf{ID}}_{\mathsf{Prep}}$ which outputs $[K^j_{w,0}]_j$.
     (c) For each $j \in [n]$ each $P_i$ sends $\mathtt{Add}([K^j_{w,0}]_j, [R^j]_j)$ to $\mathcal{F}^{\mathsf{ID}}_{\mathsf{Prep}}$ which outputs $[K^j_{w,1}]_j$.
   - If $w$ is the output of an AND gate with input wires $u, v$:
     (a) Each $P_i$ sends $\mathtt{Sample}(\mathbb{F}_2, \perp)$ to $\mathcal{F}^{\mathsf{ID}}_{\mathsf{Prep}}$ which outputs $\langle \lambda_w \rangle$.
     (b) Each $P_i$ sends $\mathsf{Multiply}(\langle \lambda_u \rangle, \langle \lambda_v \rangle)$ to $\mathcal{F}^{\mathsf{ID}}_{\mathsf{Prep}}$ which outputs $\langle \lambda_{uv} \rangle$.
     (c) For each $j \in [n]$ each $P_i$ sends $\mathtt{Sample}(\mathbb{F}_2^\kappa, P_j)$ to $\mathcal{F}^{\mathsf{ID}}_{\mathsf{Prep}}$ which outputs $[K^j_{w,0}]_j$.
     (d) For each $j \in [n]$ each $P_i$ sends $\mathtt{Add}([K^j_{w,0}]_j, [R^j]_j)$ to $\mathcal{F}^{\mathsf{ID}}_{\mathsf{Prep}}$ which outputs $[K^j_{w,1}]_j$.
   - If $w$ is the output of a XOR gate, and $u$ and $v$ its input wires:
     (a) Each $P_i$ sends $\mathtt{Add}(\langle \lambda_u \rangle, \langle \lambda_v \rangle)$ to $\mathcal{F}^{\mathsf{ID}}_{\mathsf{Prep}}$ which outputs $\langle \lambda_w \rangle$.
     (b) Each $P_i$ for each $j \in [n]$ sends $\mathtt{Add}([K^j_{u,0}]_j, [K^j_{v,0}]_j)$ to $\mathcal{F}^{\mathsf{ID}}_{\mathsf{Prep}}$ which outputs $[K^j_{w,0}]_j$ and $\mathtt{Add}([K^j_{w,0}]_j, [R^j]_j)$ to $\mathcal{F}^{\mathsf{ID}}_{\mathsf{Prep}}$ which outputs $[K^j_{w,1}]_j$ respectively.
3. For each $\tau \in G^{\mathtt{ext}}$ the parties use **Add** of $\mathcal{F}^{\mathsf{ID}}_{\mathsf{Prep}}$ to compute $\langle d_\tau \rangle = (\langle \lambda_u \rangle \oplus a) \cdot (\langle \lambda_v \rangle \oplus b) \oplus \langle \lambda_w \rangle$ from $\langle \lambda_w \rangle, \langle \lambda_{uv} \rangle, \langle \lambda_u \rangle$ and $\langle \lambda_v \rangle$. Then for each $j \in [n]$ each party $P_i$ sends $\mathtt{MultBitString}(\langle d_\tau \rangle, [R^j]_j)$ to $\mathcal{F}^{\mathsf{ID}}_{\mathsf{Prep}}$ so that the parties obtain $[W^1_{\tau,j}]_1, \ldots, [W^n_{\tau,j}]_n$ respectively.
4. The parties first send $(\mathtt{PublicOutput}, \langle \lambda_w \rangle)$ to $\mathcal{F}^{\mathsf{ID}}_{\mathsf{Prep}}$ for each output wire $w \in \mathtt{out}$ of $C_f$. Afterwards, the parties send $(\mathtt{DelayedOutputs})$ to $\mathcal{F}^{\mathsf{ID}}_{\mathsf{Prep}}$ so that each party $P_i$ obtains $\{\lambda_w\}_{w \in \mathtt{out}}$ and $\{\lambda_w\}_{w \in \mathtt{input}_i}$ as well as $R^i$, $K^i_{w,0}$ and $W^i_{\tau,j}$ as defined above.

**Fig. 8.** The MPC protocol - $\Pi^{\mathsf{ID}}_{\mathsf{MPC}}$ (Initialization).

$\mathcal{F}^{\mathsf{ID}}_{\mathsf{Prep}}$ and comparing these to the keys which are the inputs of the "faulty" gate $g$ as evaluated by every party during the circuit evaluation. This is done in Step 6 of **Abort**.

Throughout the protocol, which is outlined in Figure 8, Figure 9 and Figure 10 we let $K^i_{w,0}$ be the 0-key of $P_i$ for the wire $w$, $R^i$ be the global difference used by this party

**The MPC Protocol - $\Pi^{\mathsf{ID}}_{\mathsf{MPC}}$ (Computation)**

**Input:**

1. For all input wires $\mathtt{in} \in \mathtt{input}_i$ with input from $P_i$ the party computes $\Lambda_{\mathtt{in}} = \rho_{\mathtt{in}} \oplus \lambda_{\mathtt{in}}$. Then, $P_i$ broadcasts $\Lambda_{\mathtt{in}}$ to all parties.
2. Upon receiving $\Lambda_{\mathtt{in}}$ for all inputs of $C_f$ each party $P_i$ broadcasts $K^i_{\mathtt{in},\Lambda_{\mathtt{in}}}$ for all $\mathtt{in} \in \mathtt{input}$. Denote all these input keys as $\overline{K}^j_{\mathtt{in},\Lambda_{\mathtt{in}}}$ for $j \in [n]$.

**Garble:**

1. For all $\tau \in G^{\mathtt{ext}}$ with $\tau = (g, a, b)$ each $P_i$ defines

$$\mathtt{share}^i_\tau \leftarrow \mathsf{H}(g, K^i_{u,a}, K^i_{v,b}) \oplus (W^i_{\tau,1}, \ldots, W^i_{\tau,n}) \oplus (0, \ldots, 0, K^i_{w,0}, \ldots, 0).$$

2. Each $P_i$ broadcasts $\mathtt{share}^i_\tau$ to all parties, who set $\mathtt{circ}_\tau = \bigoplus_{j\in[n]} \mathtt{share}^j_\tau$.

**Circuit Evaluation:** Passing through the circuit topologically, the parties can now locally compute the following operations for each gate $g$ with input wires $u, v$, public values $\Lambda_u, \Lambda_v$ and keys $\overline{K}^j_{u,\Lambda_u}, \overline{K}^j_{v,\Lambda_v}$:

1. If $g$ is a XOR gate, set the public value on the output wire to be $\Lambda_w = \Lambda_u \oplus \Lambda_v$. In addition, for every $j \in [n]$, each party computes $\overline{K}^j_{w,\Lambda_w} = \overline{K}^j_{u,\Lambda_u} \oplus \overline{K}^j_{v,\Lambda_v}$.
2. If $g$ is an AND gate , then each party computes:

$$(\overline{K}^1_{w,\Lambda_w}, \ldots, \overline{K}^n_{w,\Lambda_w}) = \mathtt{circ}_{g,\Lambda_u,\Lambda_v} \oplus \left( \bigoplus\nolimits_{j\in[n]} \mathsf{H}(g, \overline{K}^j_{u,\Lambda_u}, \overline{K}^j_{v,\Lambda_v}) \right)$$

3. If $\overline{K}^i_{w,\Lambda_w} \notin \{K^i_{w,0}, K^i_{w,1}\}$, then $P_i$ broadcasts $(\mathtt{Conflict}, g, \Lambda_u, \Lambda_v)$ and enters **Abort**. Otherwise, it sets $\Lambda_w = c$ if $\overline{K}^i_{w,\Lambda_w} = K^i_{w,c}$.
4. The output key of $g$ is defined to be $(\overline{K}^1_{w,\Lambda_w}, \ldots, \overline{K}^n_{w,\Lambda_w})$ and the public value $\Lambda_w$.

**Output:** Everyone obtained a public value $\Lambda_{\mathtt{out}}$ for every circuit-output wire $\mathtt{out}$. Each party can obtain the actual output $Y = (y_1, \ldots, y_{n_{\mathtt{out}}})$ as $y_{\mathtt{out}} = \Lambda_{\mathtt{out}} \oplus \lambda_{\mathtt{out}}$, where $\lambda_w$ was obtained during **Init**.

**Fig. 9.** The MPC protocol - $\Pi^{\mathsf{ID}}_{\mathsf{MPC}}$ (Computation).

and define $K^i_{w,1} = K^i_{w,0} \oplus R^i$. Overlined keys $\overline{K}$ are those obtained in the evaluation of the circuit. We present the proof of the following theorem in the full version.

**Theorem 4.1** *Let* $\mathsf{H}$ *be a circular 2-correlation robust hash function. Assuming a secure broadcast channel, protocol* $\Pi^{\mathsf{ID}}_{\mathsf{MPC}}$ *implements the functionality* $\mathcal{F}^{\mathsf{ID}}_{\mathsf{MPC}}$ *in the* $\mathcal{F}^{\mathsf{ID}}_{\mathsf{Prep}}$*-hybrid model with security against a static, active adversary corrupting at most* $n-1$ *parties.*

*Optimistic online phase.* We can define an optimistic version of $\Pi^{\mathsf{ID}}_{\mathsf{MPC}}$, for which we analyse its best and worst case complexity. In this variant the use of broadcast is replaced with that of point-to-point channels, with the exception of the broadcast of the $\Lambda_{\mathtt{in}}$ values. We require this broadcast in order to extract the inputs of the malicious parties and the unique active path for the evaluation of the garbled circuit. As MPC with identifiable abort implies secure broadcast [16], it seems natural that we cannot avoid broadcast in our protocol, even optimistically. Furthermore, broadcasting $\Lambda_{\mathtt{in}}$ is also necessary in the original [26] construction.

**The MPC Protocol - $\Pi^{\mathsf{ID}}_{\mathsf{MPC}}$ (Abort)**

**Abort:** Receive messages of the type $(\texttt{Conflict}, g, \Lambda_u, \Lambda_v)$ from the parties where $g$ is an AND-gate. Ignore all double or other messages. Then each party does the following checks:

1. Let $\{\overline{K}^j_{\texttt{in},\Lambda_{\texttt{in}}}\}_{\texttt{in}\in\texttt{input}}$ be the keys that each party obtained from $P_j$ for all inputs during **Input**. Choose the smallest $g$ in the circuit among all the $\texttt{Conflict}$-messages.
2. Assume that honest parties evaluated the gate $g$ using the public values $\Lambda_u, \Lambda_v$. If a non-empty set of parties $\mathcal{J}$ sent $(\texttt{Conflict}, g, \Lambda'_u, \Lambda'_v)$ with $(\Lambda'_u, \Lambda'_v) \neq (\Lambda_u, \Lambda_v)$, then output $(\texttt{Abort}, \mathcal{J})$.
3. Let $\{\overline{K}^j_{u,\Lambda_u}, \overline{K}^j_{v,\Lambda_v}\}_{j\in[n]}$ be the input keys of $g$ that each party computed during **Circuit Evaluation**. For each $j \in [n]$ recompute $(\overline{K}^j_{w,1}, \ldots, \overline{K}^j_{w,n}) \leftarrow \mathsf{H}(g, \overline{K}^j_{u,\Lambda_u}, \overline{K}^j_{v,\Lambda_v})$. Let $\hat{\mathcal{J}}$ be the set of parties that sent $(\texttt{Conflict}, g, \Lambda_u, \Lambda_v)$.
4. For $\tau = (g, \Lambda_u, \Lambda_v)$ all parties send $\texttt{Add}([K^j_{w,0}]_j, [W^j_{\tau,j}]_j)$ for $j \in [n]$ to $\mathcal{F}^{\mathsf{ID}}_{\mathsf{Prep}}$ to obtain $[D^j_w]_j$.
5. The parties use **Output (String)** of $\mathcal{F}^{\mathsf{ID}}_{\mathsf{Prep}}$ to open the following values:
   (a) Send $(\texttt{PublicOutput}, [K^j_{\texttt{in},\Lambda_{\texttt{in}}}]_j)$ for all $j \in [n], \texttt{in} \in \texttt{input}$ to obtain $K^j_{\texttt{in},\Lambda_{\texttt{in}}}$;
   (b) Send $(\texttt{PublicOutput}, [K^j_{u,\Lambda_u}]_j)$, $(\texttt{PublicOutput}, [K^j_{v,\Lambda_v}]_j)$ for all $j \in [n]$ to obtain $K^j_{u,\Lambda_u}, K^j_{v,\Lambda_v}$ respectively;
   (c) Send $(\texttt{PublicOutput}, [D^j_w]_j)$ if $j = \ell$ and $(\texttt{PublicOutput}, [W^j_{\tau,\ell}]_j)$ otherwise for all $j, \ell \in [n]$ to obtain $D^j_w, W^j_{\tau,\ell}$ respectively.
6. Each party tests
   – For all $j \in [n], \texttt{in} \in \texttt{input}$ that $K^j_{\texttt{in},\Lambda_{\texttt{in}}} \stackrel{?}{=} \overline{K}^j_{\texttt{in},\Lambda_{\texttt{in}}}$;
   – For all $j \in [n]$ that $K^j_{u,\Lambda_u} \stackrel{?}{=} \overline{K}^j_{u,\Lambda_u}$ and $K^j_{v,\Lambda_v} \stackrel{?}{=} \overline{K}^j_{v,\Lambda_v}$.
   – Furthermore, for all $j, \ell \in [n]$
   **if** $j = \ell$ that $D^j_w \stackrel{?}{=} \overline{K}^j_{w,j} \oplus \texttt{share}^j_\tau[j]$;
   **if** $j \neq \ell$ that $W^j_{\tau,\ell} \stackrel{?}{=} \overline{K}^j_{w,\ell} \oplus \texttt{share}^j_\tau[\ell]$.
   Let $\mathcal{J}$ be the set of parties which generated values where one of the above tests does not work out. Then output $(\texttt{Abort}, \mathcal{J})$. If all tests hold then output $(\texttt{Abort}, \hat{\mathcal{J}})$.

**Fig. 10.** The MPC protocol - $\Pi^{\mathsf{ID}}_{\mathsf{MPC}}$ (Abort).

The main advantage of avoiding the use of broadcast in an instance of $\Pi^{\mathsf{ID}}_{\mathsf{MPC}}$ is in the reconstruction of the garbled circuit (Step 2 of **Garble**). In our optimistic version we can put the circuit together by having each party send $\texttt{share}^i_\tau$ to a single party $P_i$, which will then send the reconstructed circuit to everyone else, as in [26]. Note that replacing the broadcast here and in Step 2 of **Input**, only allows an adversary to introduce additive errors to the circuit, which can be easily derived by computing the difference between an evaluation of the garbled circuit with the correct key and an incorrect one.

When the honest parties encounter an error that would trigger the execution of **Abort**, they instead repeat the execution of $\Pi^{\mathsf{ID}}_{\mathsf{MPC}}$ from Step 2 of **Input**, this time using broadcast as indicated in the protocol boxes. Any party refusing to engage in the re-run can be trivially identified as corrupted by all honest parties. Hence, the best-case com-

plexity is of $O(\kappa n|C|)+\mathsf{bc}(I)$, where $I$ is the number of input wires, and the worst-case complexity is $O(\kappa n|C|) + \mathsf{bc}(I\kappa + n^2\kappa|C|)$.

Finally, we observe that this additional optimistic evaluation of a garbled circuit with additive errors enables an extra attempt for the adversary to guess the value $R^i$ of each honest party. As $R^i \in \{0,1\}^\kappa$, this has no impact on the protocol security.

## 5 Achieving Public Verifiability

We conclude by presenting how the previously introduced protocols and functionalities can be transformed into publicly verifiable counterparts. To model public verifiability formally, we assume (similar to [3]) the existence of a third party $\mathcal{V}$ who will not be part of $\mathcal{P}$. As a matter of fact, $\mathcal{V}$ does not need to be online or even exist while all the other protocol steps are running.

The notion of public verifiability which we achieve in this work differs from [3, 46, 17] in multiple respects: while it still allows $\mathcal{V}$ to establish the correctness of the output value, our model requires that at least one honest party is present during the MPC protocol. This is in contrast to the aforementioned works which could guarantee correctness even if all parties are corrupted. Similar to [17] (albeit different from the other works) $\mathcal{V}$ will be able to identify cheaters during the verification phase (if they existed). Note that our model lends itself to be applicable in e.g. the client-server setting in order to prevent corrupted servers from announcing false outcomes, or when MPC is integrated with distributed ledgers.

More formally, let $\mathcal{F}^{\mathsf{ID}}$ be the identifiable abort version of any functionality $\mathcal{F}$, as presented in Section 2.1. We denote as $\mathcal{F}^{\mathsf{PV}}$ an extension of $\mathcal{F}^{\mathsf{ID}}$ which further supports public verifiability, as described in Figure 11. In a nutshell, publicly verifiable functionalities incorporate an additional party, the verifier $\mathcal{V}$, who can query the functionality at any point. When doing so, $\mathcal{F}^{\mathsf{PV}}$ replies with all public outputs of $\mathcal{F}$ produced so far and, if there was an abort, the same set of corrupted parties $\mathcal{J}$ that $\mathcal{F}^{\mathsf{ID}}$ would have produced towards the honest parties.

**Functionality $\mathcal{F}^{\mathsf{PV}}$**

$\mathcal{F}^{\mathsf{PV}}$ is exactly as $\mathcal{F}^{\mathsf{ID}}$, with two changes. First, $\mathcal{F}^{\mathsf{PV}}$ runs with an additional party, the verifier $\mathcal{V}$. Second, it has the following extra command:

**Verify:** On input $(\texttt{Verify})$ from $\mathcal{V}$:
- If the functionality has received $(\texttt{Abort}, \mathcal{J})$ from $\mathcal{A}$ in the execution of any other command, return $(\texttt{Abort}, \mathcal{J})$ to $\mathcal{V}$.
- If the functionality has sent any message starting by `PublicOutput`, it forwards that same message to $\mathcal{V}$.

**Fig. 11.** Extending an identifiable-abort functionality $\mathcal{F}^{\mathsf{ID}}$ to its publicly verifiable version $\mathcal{F}^{\mathsf{PV}}$.

Whereas turning functionalities into publicly verifiable ones is pretty straightforward, one has to be more careful about their corresponding protocols. At the core, we

achieve our goal by using $\mathcal{F}_{\mathsf{Broadcast}}^{\mathsf{PV}}$, a publicly verifiable version of secure broadcast. Such variant can be implemented by using an *authenticated* bulletin board $\mathcal{F}_{\mathsf{BulletinBoard}}$ (detailed in the full version): broadcasting is equivalent to writing, and $\mathcal{V}$ can verify any 'sent' message by reading the board.

If considered on its own, adapting $\Pi_{\mathsf{Prep}}^{\mathsf{ID}}$ to implement $\mathcal{F}_{\mathsf{Prep}}^{\mathsf{PV}}$ could be attained mostly by just switching $\mathcal{F}_{\mathsf{Broadcast}}^{\mathsf{ID}}$ to its publicly verifiable version. As all messages would go through $\mathcal{F}_{\mathsf{Broadcast}}^{\mathsf{PV}}$ (either in the clear, or inside commitments), an external verifier could easily find $\mathcal{F}_{\mathsf{Prep}}^{\mathsf{PV}}$'s outputs there if no abort happened. Also, if there was an abort, $\mathcal{V}$ would find in the same place all the necessary information to run the deterministic detector algorithm $\mathcal{D}$ (cf. Definition 3.1) and conclude the same set of corrupted parties $\mathcal{J}$ as the honest parties do.

On the other hand, implementing $\mathcal{F}_{\mathsf{MPC}}^{\mathsf{PV}}$ based on $\mathcal{F}_{\mathsf{Prep}}^{\mathsf{PV}}$ requires substantially more work: the way the **Abort** procedure of $\Pi_{\mathsf{MPC}}^{\mathsf{ID}}$ identifies cheating relies on knowing the *active path* in the garbled circuit corresponding to the $\Lambda_{\mathtt{in}}$ values which were broadcast in the **Input** phase. Unfortunately, determining the active path in $\Pi_{\mathsf{MPC}}^{\mathsf{ID}}$ requires private randomness from any party in $\mathcal{P}$ (namely, Step 3 in Figure 9), which at that point cannot be revealed to $\mathcal{V}$ for running $\mathcal{D}$ because the parties have already provided their private inputs to $\Pi_{\mathsf{MPC}}^{\mathsf{ID}}$. Having all parties announce their active path does not solve the problem, as it is unclear to $\mathcal{V}$ which such path could be trusted. We now explain how to modify both $\Pi_{\mathsf{Prep}}^{\mathsf{ID}}$ and $\Pi_{\mathsf{MPC}}^{\mathsf{ID}}$ in order to achieve public verifiability.

### 5.1 Public Active Paths

To make the active path recognisable for $\mathcal{V}$, we use the well-known technique of fixing the last bit of a key to be its external wire value $\Lambda_w$, i.e., we require that each $K_{w,0}^i$ has as last bit 0 and each $K_{w,1}^i$ has as last bit 1[6]. The latter can be achieved by requiring that the last bit of $R^i$ is 1. We formalize this in the ideal functionality $\mathcal{F}_{\mathsf{Prep}}^{\mathsf{PV}}$ by requiring that vectors generated by **Sample** and **Init** have their last bit set to 0 and 1, respectively. For the sake of formality, we denote the resulting modified functionality as $\mathcal{F}_{\widetilde{\mathsf{Prep}}}^{\mathsf{PV}}$.

In order to UC-implement $\mathcal{F}_{\widetilde{\mathsf{Prep}}}^{\mathsf{PV}}$, we first modify $\Pi_{\mathsf{Prep}}^{\mathsf{ID}}$ so that all messages are sent via $\mathcal{F}_{\mathsf{Broadcast}}^{\mathsf{PV}}$. Notice that $\mathcal{F}_{\mathsf{HCom}}$ and $\mathcal{F}_{\mathsf{Rand}}$ can be easily implemented with public verifiability by having their respective protocols send all communication through $\mathcal{F}_{\mathsf{Broadcast}}^{\mathsf{PV}}$. Furthermore, we require that each party sets the last bit of their vectors according to the previous description. As dishonest parties might not follow this, we add a (cheap) random linear check using $\mathcal{F}_{\mathsf{HCom}}^{\mathsf{PV}}$ to ensure correct behaviour, which is described below:

1. Each $P_i \in \mathcal{P}$ randomly samples $s$ auxiliary masking vectors $A_1^i, \ldots, A_s^i \in \mathbb{F}_2^\kappa$, subject to the constraint that the last bit of each of them is zero. $P_i$ commits to them using $\mathcal{F}_{\mathsf{HCom}}^{\mathsf{PV}}$ as $[A_1^i]_i, \ldots, [A_s^i]_i$.
2. Denote by $[X_1^i]_i, \ldots, [X_r^i]_i$, $r = |G| + n_{\mathtt{in}}$, $P_i$'s (value-zero) keys corresponding to the circuit-input wires and the output wires of AND gates, obtained using **Sample**. Let $[R^i]_i$ be the value obtained during **Init**. Parties in $\mathcal{P}$ call $\mathcal{F}_{\mathsf{Rand}}^{\mathsf{PV}}$ to generate, for every $i \in [n]$, $s \cdot (r+1)$ random bits $b_1^i, \ldots, b_{s(r+1)}^i$.

[6] Technically, in this case we should increase the key length by one bit in order to compensate the loss of entropy, but we omit this in order to simplify our presentation.

3. For $j \in [s]$, $P_i$ uses $\mathcal{F}_{\mathsf{HCom}}^{\mathsf{PV}}$ to compute and open the random linear combination $[Z_j^i]_i = [A_j^i]_i + b_{j+s\cdot r}^i \cdot [R^i]_i + \sum_{k=1}^{r} b_{(j-1)\cdot r+k}^i \cdot [X_k^i]_i$ towards all parties, who check that the last bit of each opened $Z_j^i$ is $b_{r\cdot s+j}^i$. If that is not the case for any $i$ or $j$, they enter the **Abort** procedure.

As the last bit of every $A_j^i, X_k^i$ should be zero and the last bit of $R^i$ should be one and all those values are committed to before the $b$ values are sampled, any corrupted $P_i$ providing wrong values can only pass the $j$-th check with probability at most $1/2$, for every $j \in [s]$. On the other hand, and for an honest $P_i$, the $A_j^i$ masks prevent any leakage on $X_k^i, R^i$ beyond their lasts bits. From this, and Theorem 3.6 it then follows:

**Lemma 5.1** *The transformation to $\Pi_{\mathsf{Prep}}^{\mathsf{ID}}$ outlined above implements $\mathcal{F}_{\widetilde{\mathsf{Prep}}}^{\mathsf{PV}}$ in the $\{\mathcal{F}_{\mathsf{Broadcast}}^{\mathsf{PV}}, \mathcal{F}_{\mathsf{HCom}}^{\mathsf{PV}}, \mathcal{F}_{\mathsf{Rand}}^{\mathsf{PV}}\}$-hybrid model with security against a static, active adversary corrupting at most $n-1$ parties.*

### 5.2 Public Verifiability in the Online Phase

We now explain how to modify the online phase of $\Pi_{\mathsf{MPC}}^{\mathsf{ID}}$ in order to implement $\mathcal{F}_{\mathsf{MPC}}^{\mathsf{PV}}$. As mentioned before, we require that all communication will be done via $\mathcal{F}_{\mathsf{Broadcast}}^{\mathsf{PV}}$. While the **Init**, **Garble**, **Output** and **Abort** phase of our publicly verifiable protocol will be identical to $\Pi_{\mathsf{MPC}}^{\mathsf{ID}}$, we need to introduce some differences in **Input** and **Circuit Evaluation**. We proceed to outline those differences:

**Input:** Add a third step, where parties in $\mathcal{P}$ additionally check if the last bit of $\overline{K}^i_{\mathtt{in},\Lambda_{\mathtt{in}}}$ is identical to $\Lambda_{\mathtt{in}}$ for every $i \in [n]$. Otherwise, they broadcast $(\mathtt{Abort}, \mathcal{J})$, where $\mathcal{J}$ contains every $P_i$ who sent the wrong key.

**Circuit Evaluation:** In Step 3 of this subprotocol, parties in $\mathcal{P}$ additionally check if the last bit of every $\overline{K}^i_{w,\Lambda_w}$ is identical for every $i \in [n]$. If that is not the case, they broadcast $(\mathtt{Conflict}, g, \Lambda_u, \Lambda_v)$ and enter **Abort**.

Trivially, the **Abort** procedure still works for parties in $\mathcal{P}$ the same way it did in $\Pi_{\mathsf{MPC}}^{\mathsf{ID}}$ without the above modifications. For a verifier $\mathcal{V}$ looking at the transcript of this procedure afterwards, things also work out: Due to the fact that wire keys now indicate the corresponding $\Lambda$ value on the wire, up to the smallest $g$ in the circuit among all `Conflict` messages, $\mathcal{V}$ can be sure of having obtained the correct $\Lambda_u, \Lambda_v$ values. Hence, on Step 2, he can perform the same check honest parties in $\mathcal{P}$ did to conclude whether or not the $(\mathtt{Conflict}, g, \Lambda'_u, \Lambda'_v)$ was correct, or if a malicious party trying to cheat. $\mathcal{V}$ cannot perform Step 4, but he can obtain the resulting value on Step 5, because it is a `PublicOutput` of $\mathcal{F}_{\widetilde{\mathsf{Prep}}}^{\mathsf{PV}}$ (cf. Figure 11). The same is true for the other values revealed in that step, and thus $\mathcal{V}$ can perform the whole **Abort** procedure non-interactively. From this, and Theorem 4.1, we can conclude:

**Lemma 5.2** *The transformation to $\Pi_{\mathsf{MPC}}^{\mathsf{ID}}$ outlined above implements $\mathcal{F}_{\mathsf{MPC}}^{\mathsf{PV}}$ in the $\{\mathcal{F}_{\mathsf{Broadcast}}^{\mathsf{PV}}, \mathcal{F}_{\widetilde{\mathsf{Prep}}}^{\mathsf{PV}}\}$-hybrid model with security against a static, active adversary corrupting at most $n-1$ parties.*

## References


1. G. Asharov, Y. Lindell, T. Schneider, and M. Zohner. More efficient oblivious transfer extensions with security for malicious adversaries. In E. Oswald and M. Fischlin, editors, *EUROCRYPT 2015, Part I*, volume 9056 of *LNCS*, pages 673–701. Springer, Heidelberg, Apr. 2015.
2. G. Asharov and C. Orlandi. Calling out cheaters: Covert security with public verifiability. In X. Wang and K. Sako, editors, *ASIACRYPT 2012*, volume 7658 of *LNCS*, pages 681–698. Springer, Heidelberg, Dec. 2012.
3. C. Baum, I. Damgård, and C. Orlandi. Publicly auditable secure multi-party computation. In M. Abdalla and R. D. Prisco, editors, *SCN 14*, volume 8642 of *LNCS*, pages 175–196. Springer, Heidelberg, Sept. 2014.
4. C. Baum, B. David, and R. Dowsley. Insured mpc: Efficient secure computation with financial penalties. Financial Cryptography and Data Security (FC) 2020, 2020. Full version available at https://eprint.iacr.org/2018/942.
5. C. Baum, E. Orsini, and P. Scholl. Efficient secure multiparty computation with identifiable abort. In M. Hirt and A. D. Smith, editors, *TCC 2016-B, Part I*, volume 9985 of *LNCS*, pages 461–490. Springer, Heidelberg, Oct. / Nov. 2016.
6. D. Beaver, S. Micali, and P. Rogaway. The round complexity of secure protocols (extended abstract). In *22nd ACM STOC*, pages 503–513. ACM Press, May 1990.
7. A. Beimel, E. Omri, and I. Orlov. Protocols for multiparty coin toss with dishonest majority. In T. Rabin, editor, *CRYPTO 2010*, volume 6223 of *LNCS*, pages 538–557. Springer, Heidelberg, Aug. 2010.
8. R. Bendlin, I. Damgård, C. Orlandi, and S. Zakarias. Semi-homomorphic encryption and multiparty computation. In K. G. Paterson, editor, *EUROCRYPT 2011*, volume 6632 of *LNCS*, pages 169–188. Springer, Heidelberg, May 2011.
9. N.-P. Brandt, S. Maier, T. Müller, and J. Müller-Quade. Constructing secure multi-party computation with identifiable abort. Cryptology ePrint Archive, Report 2020/153, 2020. https://eprint.iacr.org/2020/153.
10. M. Byali, A. Patra, D. Ravi, and P. Sarkar. Fast and universally-composable oblivious transfer and commitment scheme with adaptive security. Cryptology ePrint Archive, Report 2017/1165, 2017. https://eprint.iacr.org/2017/1165.
11. R. Canetti. Universally composable security: A new paradigm for cryptographic protocols. In *42nd FOCS*, pages 136–145. IEEE Computer Society Press, Oct. 2001.
12. I. Cascudo, I. Damgård, B. David, N. Döttling, R. Dowsley, and I. Giacomelli. Efficient UC commitment extension with homomorphism for free (and applications). In S. D. Galbraith and S. Moriai, editors, *ASIACRYPT 2019, Part II*, volume 11922 of *LNCS*, pages 606–635. Springer, Heidelberg, Dec. 2019.
13. D. Chaum and S. Roijakkers. Unconditionally secure digital signatures. In A. J. Menezes and S. A. Vanstone, editors, *CRYPTO'90*, volume 537 of *LNCS*, pages 206–214. Springer, Heidelberg, Aug. 1991.
14. S. G. Choi, J. Katz, R. Kumaresan, and H.-S. Zhou. On the security of the "free-XOR" technique. In R. Cramer, editor, *TCC 2012*, volume 7194 of *LNCS*, pages 39–53. Springer, Heidelberg, Mar. 2012.
15. R. 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.
16. R. Cohen and Y. Lindell. Fairness versus guaranteed output delivery in secure multiparty computation. In P. Sarkar and T. Iwata, editors, *ASIACRYPT 2014, Part II*, volume 8874 of *LNCS*, pages 466–485. Springer, Heidelberg, Dec. 2014.

17. R. K. Cunningham, B. Fuller, and S. Yakoubov. Catching MPC cheaters: Identification and openability. In J. Shikata, editor, *ICITS 17*, volume 10681 of *LNCS*, pages 110–134. Springer, Heidelberg, Nov. / Dec. 2017.
18. I. Damgård and Y. Ishai. Constant-round multiparty computation using a black-box pseudorandom generator. In V. Shoup, editor, *CRYPTO 2005*, volume 3621 of *LNCS*, pages 378–394. Springer, Heidelberg, Aug. 2005.
19. I. Damgård, V. Pastro, N. P. Smart, and S. Zakarias. Multiparty computation from somewhat homomorphic encryption. In R. Safavi-Naini and R. Canetti, editors, *CRYPTO 2012*, volume 7417 of *LNCS*, pages 643–662. Springer, Heidelberg, Aug. 2012.
20. D. Dolev and H. R. Strong. Authenticated algorithms for byzantine agreement. *SIAM J. Comput.*, 12(4):656–666, 1983.
21. T. K. Frederiksen, M. Keller, E. Orsini, and P. Scholl. A unified approach to MPC with preprocessing using OT. In T. Iwata and J. H. Cheon, editors, *ASIACRYPT 2015, Part I*, volume 9452 of *LNCS*, pages 711–735. Springer, Heidelberg, Nov. / Dec. 2015.
22. C. Ganesh and A. Patra. Broadcast extensions with optimal communication and round complexity. In *Proceedings of the 2016 ACM Symposium on Principles of Distributed Computing*, PODC '16, page 371–380, New York, NY, USA, 2016. Association for Computing Machinery.
23. J. A. Garay, J. Katz, C.-Y. Koo, and R. Ostrovsky. Round complexity of authenticated broadcast with a dishonest majority. In *48th FOCS*, pages 658–668. IEEE Computer Society Press, Oct. 2007.
24. O. Goldreich, S. Micali, and A. Wigderson. How to play any mental game or A completeness theorem for protocols with honest majority. In A. Aho, editor, *19th ACM STOC*, pages 218–229. ACM Press, May 1987.
25. G. Hanaoka, J. Shikata, Y. Zheng, and H. Imai. Unconditionally secure digital signature schemes admitting transferability. In T. Okamoto, editor, *ASIACRYPT 2000*, volume 1976 of *LNCS*, pages 130–142. Springer, Heidelberg, Dec. 2000.
26. C. Hazay, P. Scholl, and E. Soria-Vazquez. Low cost constant round MPC combining BMR and oblivious transfer. In T. Takagi and T. Peyrin, editors, *ASIACRYPT 2017, Part I*, volume 10624 of *LNCS*, pages 598–628. Springer, Heidelberg, Dec. 2017.
27. C. Hong, J. Katz, V. Kolesnikov, W. Lu, and X. Wang. Covert security with public verifiability: Faster, leaner, and simpler. In Y. Ishai and V. Rijmen, editors, *EUROCRYPT 2019, Part III*, volume 11478 of *LNCS*, pages 97–121. Springer, Heidelberg, May 2019.
28. Y. Ishai, J. Kilian, K. Nissim, and E. Petrank. Extending oblivious transfers efficiently. In D. Boneh, editor, *CRYPTO 2003*, volume 2729 of *LNCS*, pages 145–161. Springer, Heidelberg, Aug. 2003.
29. Y. Ishai, R. Ostrovsky, and V. Zikas. Secure multi-party computation with identifiable abort. In J. A. Garay and R. Gennaro, editors, *CRYPTO 2014, Part II*, volume 8617 of *LNCS*, pages 369–386. Springer, Heidelberg, Aug. 2014.
30. Y. Ishai, M. Prabhakaran, and A. Sahai. Founding cryptography on oblivious transfer - efficiently. In D. Wagner, editor, *CRYPTO 2008*, volume 5157 of *LNCS*, pages 572–591. Springer, Heidelberg, Aug. 2008.
31. M. Keller, E. Orsini, and P. Scholl. Actively secure OT extension with optimal overhead. In R. Gennaro and M. J. B. Robshaw, editors, *CRYPTO 2015, Part I*, volume 9215 of *LNCS*, pages 724–741. Springer, Heidelberg, Aug. 2015.
32. A. Kiayias, H.-S. Zhou, and V. Zikas. Fair and robust multi-party computation using a global transaction ledger. In M. Fischlin and J.-S. Coron, editors, *EUROCRYPT 2016, Part II*, volume 9666 of *LNCS*, pages 705–734. Springer, Heidelberg, May 2016.
33. V. Kolesnikov and A. J. Malozemoff. Public verifiability in the covert model (almost) for free. In T. Iwata and J. H. Cheon, editors, *ASIACRYPT 2015, Part II*, volume 9453 of *LNCS*, pages 210–235. Springer, Heidelberg, Nov. / Dec. 2015.

34. V. Kolesnikov and T. Schneider. Improved garbled circuit: Free XOR gates and applications. In L. Aceto, I. Damgård, L. A. Goldberg, M. M. Halldórsson, A. Ingólfsdóttir, and I. Walukiewicz, editors, *ICALP 2008, Part II*, volume 5126 of *LNCS*, pages 486–498. Springer, Heidelberg, July 2008.
35. R. Kumaresan, T. Moran, and I. Bentov. How to use bitcoin to play decentralized poker. In I. Ray, N. Li, and C. Kruegel, editors, *ACM CCS 2015*, pages 195–206. ACM Press, Oct. 2015.
36. Y. Lindell, B. Pinkas, N. P. Smart, and A. Yanai. Efficient constant round multi-party computation combining BMR and SPDZ. In R. Gennaro and M. J. B. Robshaw, editors, *CRYPTO 2015, Part II*, volume 9216 of *LNCS*, pages 319–338. Springer, Heidelberg, Aug. 2015.
37. Y. Lindell, N. P. Smart, and E. Soria-Vazquez. More efficient constant-round multi-party computation from BMR and SHE. In M. Hirt and A. D. Smith, editors, *TCC 2016-B, Part I*, volume 9985 of *LNCS*, pages 554–581. Springer, Heidelberg, Oct. / Nov. 2016.
38. Y. Lindell and H. Zarosim. Adaptive zero-knowledge proofs and adaptively secure oblivious transfer. In O. Reingold, editor, *TCC 2009*, volume 5444 of *LNCS*, pages 183–201. Springer, Heidelberg, Mar. 2009.
39. Y. Lindell and H. Zarosim. On the feasibility of extending oblivious transfer. In A. Sahai, editor, *TCC 2013*, volume 7785 of *LNCS*, pages 519–538. Springer, Heidelberg, Mar. 2013.
40. J. B. Nielsen, P. S. Nordholt, C. Orlandi, and S. S. Burra. A new approach to practical active-secure two-party computation. In R. Safavi-Naini and R. Canetti, editors, *CRYPTO 2012*, volume 7417 of *LNCS*, pages 681–700. Springer, Heidelberg, Aug. 2012.
41. R. Pass. Bounded-concurrent secure multi-party computation with a dishonest majority. In L. Babai, editor, *36th ACM STOC*, pages 232–241. ACM Press, June 2004.
42. C. Peikert, V. Vaikuntanathan, and B. Waters. A framework for efficient and composable oblivious transfer. In D. Wagner, editor, *CRYPTO 2008*, volume 5157 of *LNCS*, pages 554–571. Springer, Heidelberg, Aug. 2008.
43. B. Pfitzmann and M. Waidner. Unconditional byzantine agreement for any number of faulty processors. In *STACS 92*, pages 339–350, 1992.
44. P. Rindal and R. Trifiletti. SplitCommit: Implementing and analyzing homomorphic UC commitments. Cryptology ePrint Archive, Report 2017/407, 2017. http://eprint.iacr.org/2017/407.
45. B. Schoenmakers and M. Veeningen. Universally verifiable multiparty computation from threshold homomorphic cryptosystems. In T. Malkin, V. Kolesnikov, A. B. Lewko, and M. Polychronakis, editors, *ACNS 15*, volume 9092 of *LNCS*, pages 3–22. Springer, Heidelberg, June 2015.
46. G. Spini and S. Fehr. Cheater detection in SPDZ multiparty computation. In A. C. A. Nascimento and P. Barreto, editors, *ICITS 16*, volume 10015 of *LNCS*, pages 151–176. Springer, Heidelberg, Aug. 2016.
47. C. Swanson and D. R. Stinson. Unconditionally secure signature schemes revisited. In S. Fehr, editor, *ICITS 11*, volume 6673 of *LNCS*, pages 100–116. Springer, Heidelberg, May 2011.
48. X. Wang, S. Ranellucci, and J. Katz. Global-scale secure multiparty computation. In B. M. Thuraisingham, D. Evans, T. Malkin, and D. Xu, editors, *ACM CCS 2017*, pages 39–56. ACM Press, Oct. / Nov. 2017.
49. A. C.-C. Yao. How to generate and exchange secrets (extended abstract). In *27th FOCS*, pages 162–167. IEEE Computer Society Press, Oct. 1986.