Efficient Secure Multiparty Computation with Identifiable Abort

Carsten Baum[1(✉)], Emmanuela Orsini[2], and Peter Scholl[2]

[1] Department of Computer Science, Aarhus University, Aarhus, Denmark
cbaum@cs.au.dk
[2] Department of Computer Science, University of Bristol, Bristol, UK
{emmanuela.orsini,peter.scholl}@bristol.ac.uk

Abstract. We study secure multiparty computation (MPC) in the dishonest majority setting providing security with identifiable abort, where if the protocol aborts, the honest parties can agree upon the identity of a corrupt party. All known constructions that achieve this notion require expensive zero-knowledge techniques to obtain active security, so are not practical.

In this work, we present the first efficient MPC protocol with identifiable abort. Our protocol has an information-theoretic online phase with message complexity $O(n^2)$ for each secure multiplication (where n is the number of parties), similar to the BDOZ protocol (Bendlin et al., Eurocrypt 2011), which is a factor in the security parameter lower than the identifiable abort protocol of Ishai et al. (Crypto 2014). A key component of our protocol is a linearly homomorphic information-theoretic signature scheme, for which we provide the first definitions and construction based on a previous non-homomorphic scheme. We then show how to implement the preprocessing for our protocol using somewhat homomorphic encryption, similarly to the SPDZ protocol (Damgård et al., Crypto 2012).

Keywords: Secure multiparty computation · Identifiable abort

1 Introduction

Multiparty Computation deals with the problem of jointly computing a function among a set of mutually distrusting parties with some security guarantees such as

Full version available at http://eprint.iacr.org/2016/187.pdf

C. Baum—Part of the work was done while visiting University of Bristol. The author acknowledges support from the Danish National Research Foundation and The National Science Foundation of China (under the grant 61061130540) for the Sino-Danish Center for the Theory of Interactive Computation; and also from the CFEM research center (supported by the Danish Strategic Research Council) and the COST Action IC1306.

E. Orsini—Supported in part by ERC Advanced Grant ERC-2010-AdG-267188-CRIPTO.

P. Scholl—Supported in part by EPSRC via grant EP/I03126X, and in part by the DARPA Brandeis program and the US Navy under contract #N66001-15-C-4070.

© International Association for Cryptologic Research 2016
M. Hirt and A. Smith (Eds.): TCC 2016-B, Part I, LNCS 9985, pp. 461–490, 2016.
DOI: 10.1007/978-3-662-53641-4_18

correctness of the output and privacy of the inputs. MPC has been an interesting topic in cryptography for the last 30 years, but while in the past efficiency was the main bottleneck and MPC was exclusively the subject of academic studies, the situation has steadily improved and now even large circuits can be evaluated with acceptable costs in terms of time and space. A key example of this progress is the recent line of work that began with the BDOZ [BDOZ11] and SPDZ [DPSZ12, DKL+13] protocols. These protocols are based on a secret-sharing approach and can provide active security against a dishonest majority, where any number of the parties may be corrupt.

The SPDZ-style protocols work in the *preprocessing model* (or *offline/online* setting), with an offline phase that generates random correlated data independent of the parties' inputs and the function, and an online phase, in which this correlated randomness is used to perform the actual computation. The key advantage of the preprocessing model in SPDZ lies in the efficiency of the online phase, which only uses information-theoretic techniques.

It is a well-known fact that, in the dishonest majority setting, successful termination of protocols cannot be guaranteed, so these protocols simply *abort* if cheating is detected. It was also shown by Cleve in [Cle86] that, unless an honest majority is assumed, it is impossible to obtain protocols for MPC that provide *fairness* and *guaranteed output delivery*. Fairness is a very desirable property and intuitively means that either every party receives the output, or else no-one does.

In this scenario SPDZ-style protocols, and in general all known efficient MPC protocols that allow dishonest majority, are vulnerable to Denial-of-Service attacks, where one or more dishonest parties can force the protocol to abort, so that honest parties never learn the output. They can even do this *after learning the output*, whilst remaining anonymous to the honest parties, which could be a serious security issue in some applications. This motivates the notion of *MPC with identifiable abort* (ID-MPC) [CL14,IOZ14]. Protocols with identifiable abort either terminate, in which case all parties receive the output of the computation, or abort, such that all honest parties agree on the identity of at least one corrupt party. It is clear that, while this property neither guarantees fairness nor output delivery (as it does not prevent a corrupt party from aborting the protocol by refusing to send messages) at the same time it discourages this kind of behaviour because, upon abort, at least one corrupt party will be detected and can be excluded from future computations.

Why Efficient ID-MPC is not Trivial. It is easy to see that the SPDZ protocol is not ID-MPC: Each party holds an additive share x_i of each value x and similarly an additive share $m(x)_i$ of an information-theoretic MAC on x. To open a shared value, all parties provide their shares of both the value and its MAC, and then check validity of the MAC. A dishonest party P_i can make the protocol abort by sending a share $x_i^* \neq x_i$ or $m(x)_i^* \neq m(x)_i$. However, since the underlying value x is authenticated, and not the individual shares, P_i is neither committed to x_i nor $m(x)_i$, so other parties cannot identify who

caused the abort. At first glance, it seems that the [BDOZ11] protocol might satisfy identifiable abort. In this protocol, instead of authenticating x, pairwise MACs are set up so that each party holds a MAC on every other party's share. However, the following counterexample (similar to [Sey12, Sect. 3.6]), depicted in Fig. 1, shows that this is not sufficient.

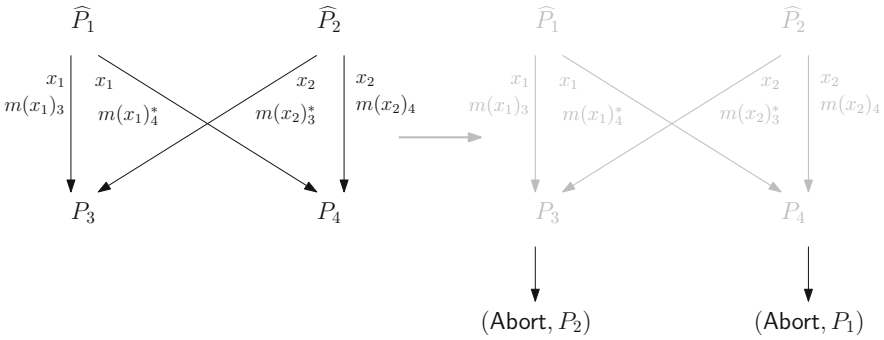

Fig. 1. Counterexample for identifiable abort with pairwise MACs

Let the adversary control parties P_1 and P_2. P_1 sends the correct value x_1 to both remaining parties P_3, P_4, but only the correct MAC $m(x_1)_3$ to P_3. To P_4, he sends an incorrect MAC $m(x_1)_4^*$. Conversely, P_2 will send the incorrect MAC $m(x_2)_3^*$ of his share x_2 to P_3 and the correct $m(x_2)_4$ to P_4. Now both honest parties P_3, P_4 can agree that some cheating happened, but as they do not agree on the identity of the corrupt party they are unable to reliably convince each other who cheated. (Note that a corrupt party could also decide to output (Abort, P_3), confusing matters even further for the honest P_2, P_3.) We conclude that, with an approach based on secret-sharing, special care must be taken so that all honest parties can agree upon the correctness of an opened value.

Our Contributions. In this work we present an efficient MPC protocol in the preprocessing model that reactively computes arithmetic circuits over a finite field, providing security with identifiable abort against up to $n-1$ out of n malicious parties. The online phase relies only on efficient, information-theoretic primitives and a broadcast channel, with roughly the same complexity as the BDOZ protocol [BDOZ11]. The offline phase, which generates correlated randomness, can be instantiated using somewhat homomorphic encryption based on ring-LWE, and allows use of all the relevant optimisations presented in [DPSZ12, DKL+13, BDTZ16].

A first building block towards achieving this goal is our definition of *homomorphic information-theoretic signatures* (HITS). Information-theoretic signature schemes [CR91] cannot have a public verification key (since otherwise an unbounded adversary can easily forge messages), but instead each party holds

a private verification key. The main security properties of IT signatures are unforgeability and consistency, meaning that no-one can produce a signature that verifies by one honest party but is rejected by another. Swanson and Stinson [SS11] were the first to formally study and provide security proofs for IT signatures, and demonstrated that many subtle issues can arise in definitions. On the other hand, homomorphic signature schemes [BFKW09, CJL09] feature an additional *homomorphic evaluation* algorithm, which allows certain functions to be applied to signatures. The verification algorithm is then given a signature, a message m and a description of a function f, and verifies that m is the output of f, applied to some previously signed inputs. We give the first definition of HITS, and the first construction of HITS for affine functions, which is based on the (non-homomorphic) construction from [HSZI00] (proven secure in [SS11]), and has essentially the same complexity.

We then show how to build ID-MPC in the preprocessing model, based on any HITS with some extra basic properties. Our basic protocol is similar to the online phase of SPDZ and BDOZ, based on a correlated randomness setup that produces random shared multiplication triples, authenticated using HITS with an unknown signing key. The downside of this approach is the need for a secure broadcast channel in every round of the protocol. Since broadcast with up to $n-1$ corrupted parties requires $\Omega(n)$ rounds of communication[1] [GKKO07] (assuming a PKI setup), this leads to a round complexity of $\Omega(n \cdot D)$, for depth D arithmetic circuits, and a message complexity of $\Omega(n^3)$ field elements per multiplication gate. The number of broadcast rounds can be reduced to just two—and the total number of rounds to $O(n + D)$—by using an insecure broadcast for each multiplication gate, and then verifying the insecure broadcasts at the end of the protocol in a single round of authenticated broadcast. Additionally, by batching the signature verification at the end of the protocol, we can reduce the message complexity per multiplication to $O(n^2)$. Overall, this gives on online phase that is only around n times slower than the SPDZ protocol, or similar to BDOZ.

In addition, we present a preprocessing protocol that uses somewhat homomorphic encryption to compute the correlated randomness needed for the online phase of our protocol, obtaining security with identifiable abort. The method for creating multiplication triples is essentially the same as [DPSZ12], but creating the additional HITS data is more complex. In addition, we must ensure that the preprocessing protocol has identifiable abort as well.

In the full version of this work, we moreover give two interesting modifications of the scheme: We present an extension of our ID-MPC scheme that implies *verifiable abort*: In the ID-MPC setting, the honest parties agree upon which party is corrupt, but they are not able to convince anyone outside of the computation of this fact. We sketch how our scheme can be modified so that this in fact is possible, using a public bulletin board. We also present an information-theoretic MPC protocol in the preprocessing model that allows use of fields of

[1] This is not needed in SPDZ, because a simple 'broadcast with abort' technique can be performed in just two rounds.

substantially smaller size than in our main protocol (with an approach that is similar to the MiniMAC protocol [DZ13]).

Comparison to Existing Work. The model of identifiable abort was first explicitly defined in the context of covert security, by Aumann and Lindell [AL10]. Cohen and Lindell [CL14] considered the relationships between broadcast, fairness and identifiable abort, and showed that an MPC protocol with identifiable abort can be used to construct secure broadcast. The classic GMW protocol [GMW87] (and many protocols based on this) satisfies the ID-MPC property, but is highly impractical due to the non-black box use of cryptographic primitives.

The most relevant previous work is by Ishai et al. [IOZ14], who formally studied constructing identifiable abort, and presented a general compiler that transforms any semi-honest MPC protocol in the preprocessing model into a protocol with identifiable abort against malicious adversaries. Their protocol is information-theoretic, and makes use of the 'MPC-in-the-head' technique of Ishai et al. [IKOS07] for proving the correctness of each message in zero-knowledge. Although recent work by Giacomelli et al. [GMO16] shows that this technique can be efficient for certain applications, we show that when applied to ID-MPC as in [IOZ14], the resulting protocol is around $O(\kappa)$ times less efficient than ours, to achieve soundness error $2^{-\kappa}$. Note that Ishai et al. also use IT signatures for authenticating values, similarly to our usage, but without the homomorphic property that allows our protocol to be efficient. For the preprocessing stage, they describe an elegant transformation that converts any protocol for implementing any correlated randomness setup in the OT-hybrid model into one with identifiable abort, which makes black-box use of an OT protocol. Again, unfortunately this method is not particularly practical, mainly because it requires the OT protocol to be secure against an adaptive adversary, which is much harder to achieve than statically secure OT [LZ13].

Several other works have used similar primitives to information-theoretic signatures for various applications. In [CDD+99], a primitive called *IC signatures* is used for adaptively secure multiparty computation. These are very similar to what we use and their construction is linearly homomorphic, but the opening stage requires every party to broadcast values, whereas in our HITS only the sender broadcasts a message. Moreover, IC signatures are required to handle the case of a corrupted dealer (which we do not need, due to trusted preprocessing), and this leads to further inefficiencies. In [IOS12], a *unanimously identifiable commitment scheme* is presented, which is used to construct identifiable secret sharing; this has similarities to a simplified form of IC signatures, but is not linear.

Finally, we note that when the number of parties is constant, it is possible to achieve a relaxed notion of fairness, called *partial fairness*, in the dishonest majority setting by allowing a non-negligible distinguishing probability by the environment [BLOO11].

Organisation. In Sect. 2, we describe the model and some basic preliminaries, and also discuss the need for and use of a broadcast channel in our protocols. Section 3 introduces the definition of homomorphic information-theoretic signatures, and Sect. 4 describes our construction. Our information-theoretic ID-MPC protocol in the preprocessing model is presented in Sect. 5, followed by the preprocessing using SHE in Sect. 6. In Sect. 7 we evaluate the efficiency of our protocols, compared with the previous state of the art.

2 Preliminaries

2.1 Notation

Throughout this work, we denote by κ and λ the statistical, resp. computational, security parameters, and we use the standard definition of negligible (denoted by $\mathsf{negl}(\kappa)$) and overwhelming function from [Gol01]. We use bold lower case letters for vectors, i.e. $\mathbf{v}$ and refer to the ith element of a vector $\mathbf{v}$ as $\mathbf{v}|_i$. The notation $x \leftarrow S$ will be used for the uniform sampling of x from a set S, and by $[n]$ we mean the set $\{1, \ldots, n\}$. The n parties in the protocol are denoted as $\mathcal{P} = \{P_1, \ldots, P_n\}$, while the adversary is denoted by $\mathcal{A}$, and has control over a subset $\mathcal{I} \subset [n]$ of the parties. We also sometimes let $\mathcal{P}$ denote the index set $[n]$, depending on the context.

2.2 Model

We prove our protocols secure in the universal composability (UC) model of Canetti [Can01], with which we assume the reader has some familiarity. Our protocols assume a single static, active adversary, who can corrupt up to $n-1$ parties at the beginning of the execution of a protocol, forcing them to behave in an arbitrary manner. We assume a *synchronous* communication model, where messages are sent in rounds, and a *rushing adversary*, who in each round, may receive the honest parties' messages before submitting theirs.

We use the UC definition of *MPC with identifiable abort*, or ID-MPC, from Ishai et al. [IOZ14]. Given any UC functionality $\mathcal{F}$, define $[\mathcal{F}]_{\perp}^{\mathtt{ID}}$ to be the functionality with the same behaviour as $\mathcal{F}$, except that at any time the adversary may send a special command (Abort, P_i), where $i \in \mathcal{I}$, which causes $[\mathcal{F}]_{\perp}^{\mathtt{ID}}$ to output (Abort, P_i) to all parties.

Definition 1 ([IOZ14]). *Let $\mathcal{F}$ be a functionality and $[\mathcal{F}]_{\perp}^{\mathtt{ID}}$ the corresponding functionality with identifiable abort. A protocol Π* securely realises *$\mathcal{F}$ with identifiable abort if Π securely realises the functionality $[\mathcal{F}]_{\perp}^{\mathtt{ID}}$.*

As noted in [IOZ14], the UC composition theorem [Can01] naturally extends to security with identifiable abort, provided that the higher-level protocol always respects the abort behaviour of any hybrid functionalities.

2.3 Broadcast Channel

Our protocols require use of a secure broadcast channel. Since Cohen and Lindell showed that MPC with identifiable abort can be used to construct a broadcast channel [CL14], and it is well known that secure broadcast is possible if and only if there are fewer than $n/3$ corrupted parties, it is not surprising that we assume this (the protocols in [IOZ14] require the same).

In practice, we suggest the broadcast primitive is implemented using *authenticated broadcast*, which exists for any number of corrupted parties, assuming a PKI setup. For example, the classic protocol of Dolev and Strong [DS83] uses digital signatures, and Pfitzmann and Waidner [PW92] extended this method to the information-theoretic setting. Both of these protocols have complexity $O(\ell n^2)$ when broadcasting ℓ-bit messages. Hirt and Raykov [HR14] presented a protocol that reduces the communication cost to $O(\ell n)$ when ℓ is large enough. We are not aware of any works analysing the practicality of these protocols, so we suggest this as an important direction for future research.

3 Homomorphic Information-Theoretic Signatures

In this section, we define the notion of *homomorphic information-theoretic signatures* (HITS). It differs slightly from standard cryptographic signatures: First and foremost, in the information-theoretic setting, a signature[2] scheme must have a distinct, private verification key for each verifying party. This is because we define security against computationally unbounded adversaries, hence a verifier could otherwise easily forge signatures. Secondly, allowing homomorphic evaluation of signatures requires taking some additional care in the definitions. To prevent an adversary from exploiting the homomorphism to produce arbitrary related signatures, the verification algorithm must be given a function, and then verifies that the signed message is a valid output of the function on some previously signed messages.

With this in mind, our definition therefore combines elements from the IT signature definition of Swanson and Stinson [SS11], and (computational) homomorphic signature definitions such as [BFKW09, CJL09, GVW15].

Definition 2 (Homomorphic Information-Theoretic Signature). *A* homomorphic information-theoretic signature (HITS) *scheme for the set of verifiers* $\mathcal{P} = \{P_1, \ldots, P_n\}$*, function class* $\mathcal{F}$ *and message space* $\mathcal{M}$*, consists of a tuple of algorithms* $(\mathsf{Gen}, \mathsf{Sign}, \mathsf{Ver}, \mathsf{Eval})$ *that satisfy the following properties:*

$(\mathsf{sk}, \mathbf{vk}) \leftarrow \mathsf{Gen}(1^\kappa, w)$ *takes as input the (statistical) security parameter* κ *and an upper bound* $w \in \mathbb{N}$ *on the number of signatures that may be created, and outputs the signing key* sk *and vector of the parties' (private) verification keys,* $\mathbf{vk} = (\mathsf{vk}_1, \ldots, \mathsf{vk}_n)$.

[2] We want to put forward that the name *signature* for the primitive in question can be somewhat misleading, as it shares properties with commitments and MACs. Nevertheless we decided to use the term for historical reasons.

$\sigma \leftarrow \mathsf{Sign}(m, \mathsf{sk})$ *is a deterministic algorithm that takes as input a message* $m \in \mathcal{M}$ *and signing key* sk*, and outputs a signature* σ*.*
$\sigma \leftarrow \mathsf{Eval}(f, (\sigma_1, \ldots, \sigma_\ell))$ *homomorphically evaluates the function* $f \in \mathcal{F}$ *on a list of signatures* $(\sigma_1, \ldots, \sigma_\ell)$*.*
$0/1 \leftarrow \mathsf{Ver}(m, \sigma, f, \mathsf{vk}_j)$ *takes as input a message* m*, a signature* σ*, a function* f *and* P_j*'s verification key* vk_j*, and checks that* m *is the valid, signed output of* f*.*

Remark 1. HITS schemes can generally be defined to operate over *data sets*. Multiple data sets can be handled by tagging each dataset with a unique identifier and restricting operations to apply only to signatures with the same tag. However, for our application we only require a single dataset, which simplifies the definition.

Remark 2. To streamline the definition even more, we consider a setting where there is only one signer, who is honest. This leads to a definition that is conceptually simpler than the IT signature definition of Swanson and Stinson [SS11], which considers a group of users who can all sign and verify each others' messages.

We then define security as follows:

Definition 3 (w, τ-security). *A HITS scheme* $(\mathsf{Gen}, \mathsf{Sign}, \mathsf{Ver}, \mathsf{Eval})$ *is* (w, τ)-secure *for a class of functions* $\mathcal{F}$ *and message space* $\mathcal{M}$ *if it satisfies the following properties:*

Signing correctness: *Let* $\ell \leq w$ *and define for* $i \in [\ell]$ *the projection function* $\pi_i(m_1, \ldots, m_\ell) = m_i$*. Then we require that for every pair* $(\mathsf{sk}, \mathbf{vk})$ *output by* Gen*, for any* $(m_1, \ldots, m_\ell) \in \mathcal{M}^\ell$*, and for all* $i \in [\ell], j \in [n]$*,*

$$\mathsf{Ver}(m_i, \mathsf{Sign}(m_i, \mathsf{sk}), \pi_i, \mathsf{vk}_j) = 1.$$

Evaluation correctness: *For every pair* $\mathsf{sk}, \mathbf{vk}$ *output by* Gen*, for every function* $f \in \mathcal{F}$*, for all messages* $(m_1, \ldots, m_\ell) \in \mathcal{M}^\ell$*, and for all* $j \in [n]$*,*

$$\mathsf{Ver}\left(f\left(m_1, \ldots, m_\ell\right), \mathsf{Eval}\left(f, \left(\mathsf{Sign}(m_1, \mathsf{sk}), \ldots, \mathsf{Sign}(m_\ell, \mathsf{sk})\right)\right), f, \mathsf{vk}_j\right) = 1.$$

Unforgeability: *Let* $\mathcal{I} \subsetneq [n]$ *be an index set of corrupted verifiers, and define the following game between a challenger* $\mathcal{C}$ *and an adversary* $\mathcal{A}$*:*

1. $\mathcal{C}$ *computes* $(\mathsf{sk}, \mathbf{vk}) \leftarrow \mathsf{Gen}(1^\kappa, w)$ *and sends* $\{\mathsf{vk}_i\}_{i \in \mathcal{I}}$ *to the adversary.*
2. $\mathcal{A}$ *may query* $\mathcal{C}$ *adaptively up to a maximum of* w *times for signatures. Let* $m_1, \ldots, m_{w'}$ *be the list of messages queried to* $\mathcal{C}$*.*
3. $\mathcal{A}$ *outputs a function* $f \in \mathcal{F}$*, a list of indices* $\{i_1, \ldots, i_\ell\} \subseteq [w']$ *in ascending order, a target message* m^* *and a signature* σ^**.*
4. $\mathcal{A}$ *wins if* $m^* \neq f(m_{i_1}, \ldots, m_{i_\ell})$ *and there exists* $j \in [n] \backslash \mathcal{I}$ *for which*

$$\mathsf{Ver}(m^*, \sigma^*, f, \mathsf{vk}_j) = 1.$$

A scheme is unforgeable if for any subset of corrupted verifiers $\mathcal{I} \subsetneq [n]$ *and for any adversary* $\mathcal{A}$*,*

$$\Pr[\mathcal{A}\ wins] \leq \tau(|\mathcal{M}|, \kappa).$$

Consistency: *The security game for consistency is identical to the unforgeability game, except for the final step (the winning condition), which becomes: 4.* $\mathcal{A}$ *wins*[3] *if there exist* $i, j \in [n] \backslash \mathcal{I}$ *such that*

$$\mathsf{Ver}(m^*, \sigma^*, f, \mathsf{vk}_i) = 1 \qquad \textit{and} \qquad \mathsf{Ver}(m^*, \sigma^*, f, \mathsf{vk}_j) = 0.$$

A scheme satisfies consistency *if for any set* $\mathcal{I} \subsetneq [n]$ *and for any* $\mathcal{A}$ *playing the above modified game,*

$$\Pr[\mathcal{A}\ wins] \leq \tau(|\mathcal{M}|, \kappa).$$

Note that evaluation correctness implies signing correctness, but we state two separate properties for clarity. The *consistency* (or *transferability*) property guarantees that a corrupted party cannot create a signature σ that will be accepted by one (honest) verifier but rejected by another. In [SS11], a reduction from consistency to unforgeability is given. However, their definition of IT signatures considers a group of users who are all signers and verifiers, any of whom may be corrupted. In our setting, there is a single, honest signer, so consistency is no longer implied and must be defined separately.

Additionally, we require that signatures output by the Eval algorithm do not reveal any information on the input messages $m_1, \ldots, m_\ell$ other than that given by $f(m_1, \ldots, m_\ell)$. This is similar to the concept of *context hiding* [GVW15] in the computational setting, and is captured by the following definition.

Definition 4 (Evaluation privacy). *A HITS scheme* $(\mathsf{Gen}, \mathsf{Sign}, \mathsf{Eval}, \mathsf{Ver})$ *is* evaluation private *if there exists a PPT algorithm* Sim *that, for every* $(\mathsf{sk}, \mathbf{vk}) \leftarrow \mathsf{Gen}$*, for every function* $f \in \mathcal{F}$*, for all messages* $m_1, \ldots, m_\ell$ *with* $m = f(m_1, \ldots, m_\ell)$*,* $\sigma_i = \mathsf{Sign}(m_i, \mathsf{sk})$ *and* $\sigma = \mathsf{Eval}(f, \sigma_1, \ldots, \sigma_\ell)$*, computes*

$$\mathsf{Sim}(\mathsf{sk}, m, f) = \sigma.$$

Intuitively, this means that any valid signature that comes from Eval can also be computed without knowing the original inputs to f, so is independent of these. This definition is simpler than that of [GVW15], as our signing algorithm is restricted to be deterministic, so we require equality rather than an indistinguishability-based notion.

4 Construction of HITS

We now describe our construction of homomorphic information-theoretic signatures. The message space $\mathcal{M}$ is a finite field $\mathbb{F}$. We restrict the function class $\mathcal{F}$

[3] There is no requirement that $m^* \neq f(m_{i_1}, \ldots, m_{i_\ell})$.

to be the set of all affine transformations $f : \mathbb{F}^w \to \mathbb{F}$ (where w is the maximum number of signatures that can be produced). The general case of affine functions with fewer than w inputs can be handled by using a default value, $\perp$, for the unused input variables. Note also that the signing algorithm is *stateful*, and must keep track of how many messages have been signed previously.

$\mathsf{Gen}(1^\kappa, w)$**:** The key generation algorithm is as follows:

1. Sample $\hat{\alpha}_1, \ldots, \hat{\alpha}_n \leftarrow \mathbb{F}$ and $\hat{\beta}_{i,1}, \ldots, \hat{\beta}_{i,n} \leftarrow \mathbb{F}$ for each $i \in [w]$.
2. For each verifier P_j, sample $\mathbf{v}_j = (v_{j,1}, \ldots, v_{j,n}) \leftarrow \mathbb{F}^n$ and compute

$$\alpha_j = \sum_{r=1}^{n} \hat{\alpha}_r \cdot v_{j,r} \quad \text{and} \quad \beta_{j,i} = \sum_{r=1}^{n} \hat{\beta}_{i,r} \cdot v_{j,r} \quad \text{for } i \in [w].$$

3. Output $\mathsf{sk} = \left(\left\{ \hat{\alpha}_r, \{\hat{\beta}_{i,r}\}_{i\in[w]} \right\}_{r=1}^{n} \right)$, $\mathbf{vk} = \left(\mathbf{v}_j, \alpha_j, \{\beta_{j,i}\}_{i\in[w]} \right)_{j=1}^{n}$.

$\mathsf{Sign}(m, \mathsf{sk})$**:** To sign the i-th message, m, (for $i \leq w$) the signer computes the vector

$$\boldsymbol{\sigma}_i = \left(\hat{\alpha}_r \cdot m + \hat{\beta}_{i,r} \right)_{r=1}^{n}.$$

$\mathsf{Eval}(f, (\boldsymbol{\sigma}_1, \ldots, \boldsymbol{\sigma}_w))$: Let $f : \mathbb{F}^w \to \mathbb{F}$ be defined by

$$f(x_1, ..., x_w) = \mu_1 \cdot x_1 + \cdots + \mu_w \cdot x_w + c,$$

with $\mu_i, c \in \mathbb{F}$. The new signature $\boldsymbol{\sigma}$ is obtained by evaluating f, excluding the constant term, over every component of the input signatures:

$$\boldsymbol{\sigma} = \mu_1 \cdot \boldsymbol{\sigma}_1 + \cdots + \mu_w \cdot \boldsymbol{\sigma}_w \in \mathbb{F}^n.$$

$\mathsf{Ver}(m, \boldsymbol{\sigma}, f, \mathsf{vk}_j)$**:** First use f to compute the additional verification data

$$\beta_j = \sum_{i=1}^{w} \mu_i \cdot \beta_{j,i} - c \cdot \alpha_j.$$

Then check that

$$\beta_j + \alpha_j \cdot m = \sum_{r=1}^{n} \boldsymbol{\sigma}|_r \cdot v_{j,r}.$$

If the check passes output 1, otherwise 0.

Theorem 1. *Let $\mathbb{F}$ be a finite field, $\mathcal{M} := \mathbb{F}$ and $\mathcal{F}$ be the the set of affine maps from $\mathbb{F}^w$ to $\mathbb{F}$, then the tuple of algorithms* $(\mathsf{Gen}, \mathsf{Sign}, \mathsf{Eval}, \mathsf{Ver})$ *is a* $(w, 3/|\mathbb{F}|)$*-secure* HITS *with evaluation privacy.*

Proof. See the full version of this work. □

As an immediate consequence of the previous theorem, we have:

Corollary 1. *Let* $|\mathbb{F}| > 2^\kappa, w = \mathsf{poly}(\kappa)$ *then* $(\mathsf{Gen}, \mathsf{Sign}, \mathsf{Eval}, \mathsf{Ver})$ *is a* $(\mathsf{poly}(\kappa), \mathsf{negl}(\kappa))$*-secure HITS with evaluation privacy.*

Proposition 1. *Let* $\mathcal{SG}, \mathcal{SK}$ *and* $\mathcal{VK}$ *be the domains of the signatures, signing and verification keys, respectively. Our* HITS *has the following memory sizes:*

$$|\mathcal{SG}| = |\mathbb{F}|^n, \qquad |\mathcal{SK}| = |\mathbb{F}|^{n(w+1)}, \qquad |\mathcal{VF}| = |\mathbb{F}|^{n(w+3)}.$$

In terms of signature size, our scheme is n-bits close to the lower bound for MRA-codes, which HITS is a special case of (see the full version of this work for more details). Note that the scheme of [SS11], which is not homomorphic, requires the same memory size as HITS for signatures and signing keys, but has slightly smaller verification keys ($|\mathbb{F}|^{n(w+2)-1}$). We do not know if the signature schemes described in [SS11] and our HITS are optimal in terms of the memory size of the keys, or if the need for larger verification keys in HITS is due to the homomorphic property of the scheme.

5 Online Phase for Efficient MPC with Identifiable Abort

In this section we describe our information-theoretic protocol for secure multiparty computation with identifiable abort in the preprocessing model. We assume a set of n parties $\mathcal{P} = \{P_1, \ldots, P_n\}$, and any HITS scheme $\mathsf{HITS} = (\mathsf{Gen}, \mathsf{Sign}, \mathsf{Eval}, \mathsf{Ver})$ that satisfies $(w, \mathsf{negl}(\kappa))$-security and evaluation privacy from Sect. 3, and supports homomorphic evaluation of linear functions over a message space of a finite field $\mathbb{F}$.

Our protocol performs reactive computation of arithmetic circuits over $\mathbb{F}$, using correlated randomness from a preprocessing setup, similarly to the BDOZ and SPDZ protocols [BDOZ11, DPSZ12]. Correctness, privacy and identifiable abort are guaranteed by the security properties of HITS. The functionality that we implement is $\mathcal{F}_{\mathsf{MPC}}$, shown in Fig. 2. Note that $\mathcal{F}_{\mathsf{MPC}}$ already contains an explicit command for identifiable abort in the output stage, since it models an unfair execution where the adversary can abort after learning the output. The modified functionality $\left[\mathcal{F}_{\mathsf{MPC}}\right]^{\mathtt{ID}}_{\perp}$ then extends this abort to be possible at any time.

Authenticated Secret Sharing. Our protocol is based on authenticated additive secret sharing over the finite field $\mathbb{F}$, and we use the following notation to represent a shared value a:

$$[\![a]\!] = \left(a_i, \sigma_{a_i}\right)_{i \in \mathcal{P}},$$

where party P_i holds $a_i \in \mathbb{F}$ and $\sigma_{a_i} = \mathsf{Sign}(a_i, \mathsf{sk})$, such that $\sum_{i \in \mathcal{P}} a_i = a$.

By the linearity of the secret sharing scheme and HITS we can easily define addition of two shares, $[\![z]\!] = [\![x]\!] + [\![y]\!]$, as follows:

1. Compute $z_i = x_i + y_i$.

Functionality $\mathcal{F}_{\mathsf{MPC}}$

Let $\mathcal{I}$ be the set of indices of corrupt parties.

Input: On input $(\mathsf{Input}, P_i, id, x)$ from P_i and $(\mathsf{Input}, P_i, id)$ from all other parties, with id a fresh identifier and $x \in \mathbb{F}$, store (id, x).

Add: On input $(\mathsf{Add}, id_1, id_2, id_3)$ from all parties (where id_1, id_2 are present in memory), retrieve (id_1, x), (id_2, y) and store $(id_3, x + y)$.

Multiply: On input $(\mathsf{Mult}, id_1, id_2, id_3)$ from all parties (where id_1, id_2 are present in memory), retrieve (id_1, x), (id_2, y) and store $(id_3, x \cdot y)$.

Output: On input (Output, id) from all parties (where id is present in memory), retrieve (id, y) and send y to the adversary. Wait for the adversary to input either $\mathsf{Deliver}$ or (Abort, P_i) for some $i \in \mathcal{I}$. If the adversary sends $\mathsf{Deliver}$ then output y to all honest parties, otherwise output (Abort, P_i).

Fig. 2. Ideal functionality for reactive MPC in the finite field $\mathbb{F}$.

2. Compute $\sigma_{z_i} = \mathsf{Eval}(f, (\sigma_{x_i}, \sigma_{y_i}))$, where $f(a, b) = a + b$.
3. Output $[\![z]\!] = \big(z_i, \sigma_{z_i}\big)_{i \in \mathcal{P}}$.

Note that if $\sigma_{x_i}, \sigma_{y_i}$ are already outputs of the Eval algorithm, then f should instead be defined to include the linear function that was applied to these inputs previously, and Eval applied to those inputs. However, this is just a technicality and in practice, each homomorphic addition can be computed on-the-fly. We can also define addition or multiplication of shared values by constants, using Eval in a similar way.[4]

$\mathsf{Open}([\![a]\!])$:

1. Every party $P_i \in \mathcal{P}$ broadcasts (a_i, σ_{a_i}).
2. Each party P_i runs $\mathsf{Ver}(a_j, \sigma_{a_j}, f, \mathsf{vk}_i)$, for each $j \neq i$. If for some j the check fails, P_i outputs (Abort, P_j), otherwise it outputs $a = \sum_{i \in \mathcal{P}} a_i$.

Fig. 3. Procedure for opening an authenticated, shared value.

In Fig. 3 we define the basic subprotocol used to open authenticated, shared values. Each time the command Open is called, parties check the correctness of the opened value using the Ver algorithm. For each share, the intuition is that if the corresponding signature is verified, then the share is correct with overwhelming probability due to the unforgeability of the scheme; on the contrary,

[4] For addition with a constant, only one party (say P_1) needs to adjust their share. Signatures stay the same, as the verification algorithm accounts for the constant term in the affine function.

if there exists an index $j \in \mathcal{P} \backslash \mathcal{I}$, where $\mathcal{I}$ denotes the set of corrupt parties, for which the check does not go through, then the same happens for all honest parties, due to the consistency of HITS.

Preprocessing Requirements. The preprocessing functionality, $\mathcal{F}_{\mathsf{Prep}}$, is shown in Fig. 4. It generates a set of HITS keys $(\mathsf{sk}, \mathbf{vk})$ and gives each party a verification key, whilst no-one learns the signing key. The functionality then computes two kinds of authenticated data, using sk:

- **Input tuples:** Random shared values $[\![r]\!]$, such that one party, P_i, knows r. This is used so that P_i can provide input in the online phase.
- **Multiplication triples:** Random shared triples $[\![a]\!], [\![b]\!], [\![c]\!]$, where $a, b \leftarrow \mathbb{F}$ and $c = a \cdot b$.

Note that corrupted parties can always choose their own shares of authenticated values, instead of obtaining random shares from the functionality.

Protocol. Our protocol, shown in Fig. 5, is based on the idea of securely evaluating the circuit gate by gate in a shared fashion, using the linearity of the $[\![\cdot]\!]$-representation for computing all linear gates, preprocessed multiplication triples for multiplication using Beaver's technique [Bea91], and preprocessed input tuples for the inputs.

Functionality $\mathcal{F}_{\mathsf{Prep}}$

Let $\mathcal{I}$ be the set of indices of corrupt parties. The functionality is parametrised by the statistical security parameter, κ.

Initialise: On input (Init, w) from all parties, do the following:
1. Compute $(\mathsf{sk}, \mathbf{vk}) \leftarrow \mathsf{Gen}(1^\kappa, w)$.
2. Send vk_i to party P_i and store sk.

Macro $\mathsf{Bracket}$**:** On input $(\mathsf{Bracket}, x)$, create the representation $[\![x]\!]$ as follows:
1. Receive shares x_i for $i \in \mathcal{I}$ from the adversary.
2. Sample random shares $x_i \leftarrow \mathbb{F}$, for $i \notin \mathcal{I}$, subject to the constraint that $x = x_1 + \cdots + x_n$.
3. For $i = 1, \ldots, n$, compute $\sigma_{x_i} = \mathsf{Sign}(x_i, \mathsf{sk})$ and return $\{x_i, \sigma_{x_i}\}_{i \in [n]}$.

Input: On input (Input, P_i) from all parties, sample a random $r \leftarrow \mathbb{F}$ and run $(\mathsf{Bracket}, r)$ to obtain $[\![r]\!]$. Output (r_j, σ_{r_j}) to each party P_j, and also r to P_i.

Triple: On input (Triple) from all parties, do the following:
1. Sample $a, b \leftarrow \mathbb{F}$ and let $c = a \cdot b$.
2. Run the macro $(\mathsf{Bracket})$ on input a, b and c to obtain $[\![a]\!], [\![b]\!], [\![c]\!]$. Output $(a_i, b_i, c_i, \sigma_{a_i}, \sigma_{b_i}, \sigma_{c_i})$ to each party P_i.

Fig. 4. Ideal functionality for the preprocessing phase.

Protocol Π_{Online}

Let n_M be the number of secure multiplications to be performed and n_I the total number of inputs.

Initialise: The parties call $\mathcal{F}_{\mathsf{Prep}}$ with $(\mathsf{Init}, n \cdot (3n_M + n_I))$. If $\mathcal{F}_{\mathsf{Prep}}$ outputs (Abort, P_i), the parties output (Abort, P_i) and halt.

Input: For party P_i to input $x \in \mathbb{F}$, the parties call $\mathcal{F}_{\mathsf{Prep}}$ with (Input, P_i) to obtain a mask value $[\![r]\!]$, and P_i also obtains r:
1. P_i sets $m = r - x$ and broadcasts m.
2. All parties locally compute $[\![x]\!] = [\![r]\!] - m$.

Add: On input $([\![x]\!], [\![y]\!])$, parties locally compute $[\![x + y]\!] = [\![x]\!] + [\![y]\!]$.

Multiply: On input $([\![x]\!], [\![y]\!])$, the parties do the following:
1. Take one multiplication triple $([\![a]\!], [\![b]\!], [\![c]\!])$ from $\mathcal{F}_{\mathsf{Prep}}$, compute $[\![\epsilon]\!] = [\![x]\!] - [\![a]\!]$, $[\![\rho]\!] = [\![y]\!] - [\![b]\!]$.
2. Call $\mathsf{Open}([\![\epsilon]\!])$ and $\mathsf{Open}([\![\rho]\!])$.
3. Locally compute $[\![z]\!] = [\![c]\!] + \epsilon \cdot [\![b]\!] + \rho \cdot [\![a]\!] + \epsilon \cdot \rho$.

Output: To output a share $[\![y]\!]$, the parties call $\mathsf{Open}([\![y]\!])$. If for some $i \in \mathcal{P}$ the check fails, output (Abort, i) and halt, otherwise accept y as a valid output.

Fig. 5. Operations for secure function evaluation with identifiable abort.

5.1 Security

Theorem 2. *In the $\mathcal{F}_{\mathsf{Prep}}$-hybrid model, the protocol Π_{Online} implements $\left[\mathcal{F}_{\mathsf{MPC}}\right]_{\perp}^{\mathtt{ID}}$ with statistical security against any static active adversary corrupting up to $n - 1$ parties.*

Proof. Let $\mathcal{A}$ be a malicious PPT real adversary attacking the protocol Π_{Online}, we construct an ideal adversary $\mathcal{S}$ with access to $\mathcal{F}_{\mathsf{MPC}}$ which simulates a real execution of Π_{Online} with $\mathcal{A}$ such that no environment $\mathcal{Z}$ can distinguish the ideal process with $\mathcal{S}$ and $\mathcal{F}_{\mathsf{MPC}}$ from a real execution of Π_{Online} with $\mathcal{A}$. $\mathcal{S}$ starts by invoking a copy of $\mathcal{A}$ and running a simulated interaction of $\mathcal{A}$ with $\mathcal{Z}$.

After describing the simulator we will argue indistinguishability of the real and ideal worlds. Let $\mathcal{I}$ be the index set of corrupt parties, simulation proceeds as follows:

Simulating the **Initialise** *Step.* The simulator $\mathcal{S}$ honestly emulates $\mathcal{F}_{\mathsf{Prep}}$ towards the adversary $\mathcal{A}$. Note that $\mathcal{S}$ knows all the data given to the adversary and the simulated signing key sk^* of HITS, so can generate a valid signature for any message.

Simulating the **Input** *Step.* We distinguish two cases:

– For $i \in \mathcal{P} \backslash \mathcal{I}$, $\mathcal{S}$ emulates towards $\mathcal{A}$ a broadcast of a random value $m \in \mathbb{F}$, and proceeds according to the protocol with the next simulated random input

tuple, r. Thereafter, $\mathcal{S}$ computes $x = r - m$ and stores x, the dummy, random input for honest P_i.
– For $i \in \mathcal{I}$, $\mathcal{S}$ receives from the adversary the message m, and retrieves the next random input tuple r. It then computes $x = r - m$ and inputs it to the $\left[\mathcal{F}_{\mathsf{MPC}}\right]_{\perp}^{\mathtt{ID}}$.

Simulating the **Circuit Evaluation.** For linear gates, the simulator does not have to simulate any message on the behalf of the honest parties. $\mathcal{S}$ updates the internal shares and calls the respective procedure in $\left[\mathcal{F}_{\mathsf{MPC}}\right]_{\perp}^{\mathtt{ID}}$.

In a multiplication gate, for each call to Open, $\mathcal{S}$ receives all the corrupt shares $(t_j^*, \sigma_{t_j^*})$ from $\mathcal{A}$, and computes and sends the shares and signatures for the dummy honest parties as in the protocol. Let (t_j, σ_{t_j}) be the values that $\mathcal{S}$ expects from the dishonest P_j, based on previous computations and the simulated preprocessing data. $\mathcal{S}$ checks for all the $(t_j^*, \sigma_{t_j^*})$ received from $\mathcal{A}$ and for all $i \in \mathcal{P} \backslash \mathcal{I}$ that $t_j = t_j^*$ and $\sigma_{t_j} = \sigma_{t_j^*}$. If the check does not pass for some $j \in \mathcal{I}$ then $\mathcal{S}$ sends (Abort, P_j) to $\left[\mathcal{F}_{\mathsf{MPC}}\right]_{\perp}^{\mathtt{ID}}$. Otherwise it proceeds.[5]

Simulating the **Output** *Step.* The simulator sends (Output) to the functionality and gets the result y back. Let y' be the output value that the simulator has computed using dummy, random inputs on behalf of the honest parties. Then it picks an honest party P_{i_0}, and modifies its share as $y_{i_0}^* = y_{i_0} + (y - y')$, then uses the evaluation privacy algorithm to compute $\sigma_{y_{i_0}^*} = \mathsf{Sim}(\mathsf{sk}^*, y_{i_0}^*, f)$, where f is the same linear function that has been applied to obtain $\sigma_{y_{i_0}}$, and sends the honest shares and signatures to the adversary. It then receives $(y_j^*, \sigma_{y_j^*})_{j \in \mathcal{I}}$ from the adversary, while expecting y_j, σ_{y_j}. If $y_j = y_j^*$ and $\sigma_{y_j} = \sigma_{y_j^*}$ for all $j \in \mathcal{I}$ then $\mathcal{S}$ sends $\mathsf{Deliver}$ to the functionality; otherwise it sends (Abort, P_j) for the lowest j that failed and halts.

Indistinguishability. Now we prove that the all the simulated transcripts and the honest parties' outputs are identically distributed to the real transcripts and output in the view of the environment $\mathcal{Z}$, except with probability $\mathsf{negl}(\kappa)$.

During *initialisation*, the simulator honestly runs an internal copy of $\mathcal{F}_{\mathsf{Prep}}$, so the simulation of this step is perfect. In the *input* step, the values m broadcast by honest parties are uniformly random in both cases, as they are masked by a one-time uniformly random value from $\mathcal{F}_{\mathsf{Prep}}$ that is unknown to $\mathcal{Z}$.

In the *multiplication* step, the parties call the command Open. Honest shares and signatures are simulated as in the protocol, using the simulated data from the emulation of $\mathcal{F}_{\mathsf{Prep}}$, and applying the Eval algorithm. The broadcast shares are all uniformly distributed in both worlds, as the shares are always masked by fresh random values from $\mathcal{F}_{\mathsf{Prep}}$, so are perfectly indistinguishable. To argue indistinguishability of the signatures, we need to use the evaluation privacy property.

[5] If there is more than invalid share then we always abort with the smallest index where the check fails.

We must prove that

$$\sigma_{t_i} \overset{s}{\approx} \sigma_{t_i^*},$$

where $\{\sigma_{t_i^*}\}_{i \notin \mathcal{I}}$ are the simulated ideal-world signatures, and $\{\sigma_{t_i}\}_{i \notin \mathcal{I}}$ are the real-world signatures, for some honest parties' shares $\{t_i\}_{i \notin \mathcal{I}}$.

Since σ_{t_i} and $\sigma_{t_i^*}$ are both valid signatures output from Eval, evaluation privacy guarantees that there exists an algorithm Sim such that:

$$\sigma_{t_i} = \mathsf{Sim}(\mathsf{sk}, t_i, g) \quad \text{and} \quad \sigma_{t_i^*} = \mathsf{Sim}(\mathsf{sk}^*, t_i^*, g),$$

where g is the linear function evaluated to get the values t_i and t_i^*, and sk and sk^* are respectively the real-world and ideal-world secret keys. Since (t_i, sk) and (t_i^*, sk^*) are identically distributed in both the executions, then the same holds for σ_{t_i} and $\sigma_{t_i^*}$. Note that it is crucial here that $\mathcal{S}$ computes $\sigma_{t_i^*}$ using Eval and the function g, rather than creating a fresh signature using sk^*, otherwise indistinguishability would not hold.

We also must consider the abort behaviour of the Open procedure in the two worlds. If during any opening, $\mathcal{A}$ attempts to open a fake value then it will always be caught in the simulation, whereas it succeeds if it is able to forge a corresponding signature in the real protocol. Hence, if the ideal protocol aborts with the identity of some corrupt party P_i, then the same happens in the real world, except with negligible probability due to unforgeability. The consistency property of HITS ensures that if one honest party outputs (Abort, P_i) in the protocol, then all the honest parties will output the same, except with negligible probability.

Now, if the real or simulated protocol proceeds to the last step, $\mathcal{Z}$ observes the *output* value y, and the corresponding honest parties' shares, together with their signatures. The honest shares are consistent with y and the signatures are correctly generated in both worlds. Again, to argue indistinguishability of the signatures we can use the evaluation privacy property of HITS. Hence $\mathcal{Z}$'s view of the honest parties' messages in the last step has the same distribution in the real and hybrid execution.

Finally, we must argue indistinguishability of the outputs in both worlds. In the ideal world, the output y is a correct evaluation on the inputs, so the only way the environment can distinguish is to produce an incorrect output in the real world. This can only happen if a corrupt party sending an incorrect share that is successfully verified. However, as we have seen before, if the adversary attempts to open a fake value, during the input, multiplication or output step, then it will be caught with overwhelming probability, by the unforgeability and consistency properties of HITS. □

5.2 An Optimised Protocol

When instantiated with our HITS scheme from Sect. 4, the online phase protocol above requires $O(n^2)$ field elements to be broadcasted per secure multiplication.

Since each authenticated broadcast requires $O(n)$ rounds, this gives a communication complexity of at least $O(n^3)$ field elements per multiplication and $O(D \cdot n)$ rounds overall, where D is the multiplicative depth of the arithmetic circuit. We now describe an optimised variant of our protocol, which reduces the number of rounds to $O(D + n)$ and the communication cost per multiplication to $O(n^2)$.

Reducing the Number of Broadcasts. Let Π_{bc} be the UC protocol for authenticated broadcast used in the protocol. We make the following assumption about its structure: in the first round of Π_{bc}, the sender (with input x) sends x, and nothing more, to all parties.[6] Let the remainder of the protocol be denoted Π'_{bc}.

We now modify the protocol Π_{Online} so that whenever a party P_i is supposed to broadcast a value x_i in the Open subprotocol, P_i instead sends x_i to all parties, and appends x_i to a list B_i. Note that the **Input** stage still requires broadcast, as otherwise it seems difficult for the simulator to extract a corrupted party's input. The **Output** stage is then modified so that first, each party runs $\Pi'_{\mathsf{bc}}(B_i)$ to complete the broadcasts that were initialised in the previous rounds. With this change, there are only two broadcast rounds and each multiplication gate requires just one round of communication, reducing the overall number of rounds to $O(D + n)$.

Batching the Signature Verification. We can reduce the number of field elements sent during a multiplication to $n - 1$ per party by checking all signatures together in the **Output** stage of the protocol, rather than during the circuit evaluation. This means that during the computation, the parties only send shares without the corresponding signatures. We then check a random linear combination of each parties' signatures just before every output stage.

The complete protocol for the optimised output stage is given in Fig. 6. Since there are only two authenticated broadcast rounds, the number of rounds for computing a depth D circuit with one output gate in the optimised protocol is $O(D + n)$. The total number of field elements sent over the network is no more than[7]

$$n_I \cdot \mathsf{bc}(1) + 2n(n-1) \cdot n_M + n \cdot \mathsf{bc}(n + 2n_M + 1),$$

where n_I is the total number of private inputs, n_M the number of secure multiplications and $\mathsf{bc}(m)$ the cost of broadcasting m elements using Π_{bc}. Meanwhile, the storage cost (for the preprocessing data) is $O(n(n_M + n_I))$ per party.

A drawback of this optimization is that in comparison to Π_{Online} a party that sends corrupt signatures σ will now only be caught *after* $\mathcal{A}$ *learns the output.*

[6] Almost any broadcast protocol can be easily converted into this form. For example, the Dolev-Strong broadcast [DS83] begins with the sender sending $(x, \mathsf{Sign}(x))$ to all parties; we split this up into one round for x and one round for $\mathsf{Sign}(x)$.

[7] Excluding the cost of $\mathcal{F}_{\mathsf{Rand}}$, which can be implemented using standard techniques such as a hash-based commitment scheme in the random oracle model.

Protocol $\mathsf{Output}(\llbracket y \rrbracket)$

Let B_i be the list of shares sent by a party P_i in all input and multiplication steps since the last Output invocation.

OutputCheck: Run the following, for each party P_i:

1. P_i sends the share y_i to all other parties, and appends y_i to B_i.
2. Write $B_i = (a_1, \ldots, a_t)$.
3. Sample $(r_1, \ldots, r_t) \in \mathbb{F}^t$ using $\mathcal{F}_{\mathsf{Rand}}$.
4. P_i computes $\sigma = \mathsf{Eval}(f, (\sigma_{a_1}, \ldots, \sigma_{a_t}))$, where the linear function f is defined as $f(x_1, \ldots, x_t) = r_1 \cdot x_1 + \cdots + r_t \cdot x_t$.
5. P_i invokes $\Pi_{\mathsf{bc}}(\sigma)$ and $\Pi'_{\mathsf{bc}}(a_1, \ldots, a_t)$ to broadcast σ and complete the broadcast of the shares $a_1, \ldots, a_t$. If the broadcast fails, output (Abort, P_i).
6. Every party P_j (for $j \neq i$) computes $f(a_1, \ldots, a_t) = a$ and checks if $\mathsf{Ver}(a, \sigma, f, \mathsf{vk}_j) = 1$. If the check fails, output (Abort, P_i).

Output: Each party computes $y = \sum_{i=1}^{n} y_i$ and outputs y.

Fig. 6. Output stage of the optimised online protocol.

Functionality $\mathcal{F}_{\mathsf{Rand}}(\mathbb{F})$

Random sample: Upon receiving $(rand; u)$ from all parties, it samples a uniform $r \in \mathbb{F}$ and outputs $(rand, r)$ to all parties.

Fig. 7. Functionality $\mathcal{F}_{\mathsf{Rand}}$ that provides random values to all parties.

We stress that this is according to the definition of *identifiable abort* (which does not specify when the abort signal must be sent), but different from Π_{Online} where such behaviour would immediately be detected.

Security of the Modified Online Phase. We now argue security of the new online protocol, describing the key differences compared with the previous protocol. In the simulation, the simulator $\mathcal{S}$ now cannot determine whether a corrupt party has sent the correct message during the **Multiply** command, since the signatures are not sent here. Instead, this must be detected in the output stage when the broadcasts and signatures are checked.

In the **OutputCheck** stage, $\mathcal{S}$ first calls the functionality $\left[\mathcal{F}_{\mathsf{MPC}}\right]_{\perp}^{\mathtt{ID}}$ to obtain the result y, then adjusts one honest party's share and signature (using the evaluation privacy algorithm) to fix the correct output as before, and sends the honest shares to the adversary. For the remainder of this stage, the simulator acts as in the protocol for the honest parties, computing the random linear combination of signatures using Eval, and then runs the simulator of Π_{bc} for each broadcast. If any broadcast fails for a corrupt sender P_j then $\mathcal{S}$ sends (Abort, P_j) to $\left[\mathcal{F}_{\mathsf{MPC}}\right]_{\perp}^{\mathtt{ID}}$. If all broadcasts succeed, $\mathcal{S}$ checks the signatures and sends (Abort, P_j) if the

signature of P_j does not verify. Note that an incorrect broadcast can lead to an honest party's signature being incorrect, so it is important that the broadcasts are checked first here.

Indistinguishability of all shares sent up until the **Output** stage follows from uniformity of the preprocessing data, as in the previous protocol. The security of the Π_{bc} simulator guarantees indistinguishability of step 5, in particular that all parties agree upon the sets of shares B_i that were sent by each party P_i during the protocol.

If the broadcasts succeed then the honest parties' signatures will always be correctly generated, and the evaluation privacy property of HITS guarantees they are identically distributed. The environment therefore can only distinguish between the worlds by causing the output, y, to be incorrect. Suppose a corrupt party P_i broadcasts the values $B_i' = (a_1', \ldots, a_t')$ in the protocol, and $a_j \neq a_j'$ for at least one j, where a_j is the original signed value that P_i was supposed to send. Then if the verification in step 5 of the output stage succeeds, the correctness and security properties of HITS guarantee that:

$$\sum_{i=1}^{t} (a_i - a_i') \cdot r_i = 0,$$

It is easy to see that the probability of this check passing is $1/|\mathbb{F}|$, as the values r_i are unknown to the adversary at the time of choosing a_i', so the check prevents an incorrect output with overwhelming probability.

6 Preprocessing with Identifiable Abort

This section describes a practical protocol for securely implementing $\mathcal{F}_{\mathsf{Prep}}$ with identifiable abort, based on somewhat homomorphic encryption. The protocol is based on the SPDZ preprocessing [DPSZ12,DKL+13], but the cost is around n^2 times higher due to the larger amount of preprocessing data needed for the HITS data in our online phase.

We first explain in more detail why the generic preprocessing method of Ishai et al. [IOZ14] does not lead to an efficient protocol. They presented a method to transform any protocol for a correlated randomness setup in the OT-hybrid model into a protocol that is secure with identifiable abort. Although their compiled protocol only requires black-box use of an OT protocol, it is impractical for a number of reasons:

- The protocol to be compiled is assumed to consist only of calls to an ideal OT functionality and a broadcast channel. This means that any pairwise communication must be performed via the OT functionality and so is very expensive.
- The transformation requires first computing an authenticated secret sharing of the required output, and then opening this to get the output. In our case, the output of $\mathcal{F}_{\mathsf{Prep}}$ is already secret shared and authenticated, so intuitively, this step seems unnecessary.

– Their security proof requires the underlying OT protocol to be adaptively secure. This is much harder to achieve in practice, and rules out the use of efficient OT extensions [LZ13].

6.1 Modified Functionality $\mathcal{F}^*_{\mathsf{Prep}}$

The $\mathcal{F}_{\mathsf{Prep}}$ functionality from Sect. 5 is completely black-box with respect to the HITS scheme used. In this section, we implement preprocessing specifically for the scheme HITS from Sect. 4. We also make one small modification to the initialisation of $\mathcal{F}_{\mathsf{Prep}}$, shown in Fig. 8, which simplifies our preprocessing protocol by not requiring the adversary's verification data, $\mathbf{v}_j$, to be uniformly random. The following proposition shows that the scheme, and therefore online phase, remain secure with this modification.

Proposition 2. *The scheme* HITS *remains secure when* Gen *is modified to allow adversarial inputs, as in* $\mathcal{F}^*_{\mathsf{Prep}}$.

Proof. This easily follows by inspection of the scheme. Notice that the signing and verification algorithms for honest parties are independent of the values $\{\mathbf{v}_j\}_{j\in\mathcal{I}}$, so changing the distribution of these cannot cause an honest party to accept an invalid signature or reject a valid signature. □

We now show how to use somewhat homomorphic encryption to perform the preprocessing with identifiable abort.

6.2 SHE Scheme Requirements

As in SPDZ, we use a *threshold somewhat homomorphic encryption* scheme $\mathsf{SHE} = (\mathsf{Gen}, \mathsf{Enc}, \mathsf{Dec}, \boxplus, \boxtimes)$ to generate the preprocessing data. The scheme must have a message space of $\mathbb{F}$, and we represent ciphertexts known to all parties with the notation $\langle x\rangle = \mathsf{Enc}(x)$. The binary operators $\boxplus, \boxtimes$ then guarantee that

$$\langle x+y\rangle = \langle x\rangle \boxplus \langle y\rangle \quad \text{and} \quad \langle x\cdot y\rangle = \langle x\rangle \boxtimes \langle y\rangle,$$

for some suitable choice of randomness in the output ciphertexts. For our purposes, these homomorphic operations only need to support evaluation of circuits

On input $(\mathsf{Init}, n_M, n_I)$ from all parties, set $w = n\cdot(3\cdot n_M + n_I)$ and do the following:

1. Wait for the adversary to input $\mathbf{v}_j \in \mathbb{F}^n$, for each $j \in \mathcal{I}$.
2. Compute $(\mathsf{sk}, \mathbf{vk}) \leftarrow \mathsf{HITS.Gen}(1^\kappa, w)$, except using the provided values $\{\mathbf{v}_j\}_{j\in\mathcal{I}}$ to compute $\{\mathsf{vk}_j\}_{j\in\mathcal{I}}$, instead of sampling fresh values.
3. Send vk_i to party P_i and store sk.

Fig. 8. Initialise command of $\mathcal{F}^*_{\mathsf{Prep}}$.

with polynomially many additions and multiplicative depth 1. As was shown in [DPSZ12,DKL+13], a ring-LWE variant of the BGV scheme [BGV12] is practical for this purpose, and this also allows homomorphic operations to be batched for greater efficiency.

In addition, we require the following interactive protocols that will be used for the preprocessing.

Zero Knowledge Proof of Plaintext Knowledge. A protocol Π_{ZKPoK}, which is a public-coin zero-knowledge proof of knowledge of the message and randomness that makes up a ciphertext. When used in our preprocessing protocol, all parties will sample the public verifier's messages with a coin-tossing functionality $\mathcal{F}_{\mathsf{Rand}}$ (see Fig. 7), so that the proofs are verified by all parties.

Distributed Key Generation and Decryption. The distributed key generation protocol outputs a public key to all parties, and an additively shared secret key. The distributed decryption protocol then allows the parties to decrypt a public ciphertext so that all parties obtain the output. These requirements are modelled in the functionality $\mathcal{F}_{\mathsf{KeyGenDec}}$ (Fig. 9). To achieve security with identifiable abort in our preprocessing protocol, note that the distributed decryption method modelled in $\mathcal{F}_{\mathsf{KeyGenDec}}$ always outputs a *correct* decryption, unlike the method in SPDZ [DPSZ12], which allows a corrupted party to introduce additive errors into the output. The SPDZ method can easily be modified to achieve this, by including a zero-knowledge proof, similar to the Π_{ZKPoK} proof used for ciphertext generation. Efficient zero-knowledge proofs for actively secure LWE-based key generation and distributed decryption were also given in [AJL+12], which can be adapted to the ring-LWE setting.

6.3 Basic Subprotocols

In Fig. 10 we describe some basic subprotocols for generating and decrypting ciphertexts. The $\mathsf{RandShCtxt}$ subprotocol creates n public ciphertexts encrypting uniformly random shares, where each party holds one share. The $\mathsf{ShareDec}$ subprotocol takes a public ciphertext $\langle m \rangle$, encrypting m, and performs distributed decryption in such a way that each party learns only a random, additive

Functionality $\mathcal{F}_{\mathsf{KeyGenDec}}$

KeyGen(1^λ)**:** Let $(\mathsf{sk}, \mathsf{pk}) \leftarrow \mathsf{SHE.Gen}(1^\lambda)$. Store sk and output pk to all parties.

DistDec$(\langle m \rangle)$**:** On input a ciphertext $\langle m \rangle$ from all parties, output

$$m = \mathsf{Dec}(\langle m \rangle, \mathsf{sk})$$

to all P_i, where m may be a valid message or an invalid ciphertext symbol $\perp$.

Fig. 9. SHE distributed key generation and decryption functionality.

share of m. If the flag new_ctxt is set to 1 then ShareDec additionally outputs a fresh encryption of the message m to all parties. This is used to ensure that SHE only needs to evaluate circuits of multiplicative depth 1. The PrivateDec subprotocol is another variant of distributed decryption that decrypts the ciphertext $\langle x \rangle$ only to P_i. Note that the private decryption protocol used in [DPSZ12] is not suitable here, as it involves parties all sending a single message to P_j; in the identifiable abort setting, this would allow P_j to claim that an honest party P_i sent an invalid message, as the messages are not verifiable by all parties. To get around this, our PrivateDec protocol only uses broadcasted messages that are verifiable by all parties using the public-coin zero-knowledge proofs.

6.4 Creating the Preprocessing Data

The complete preprocessing protocol is shown in Figs. 11 and 12. To create a multiplication triple, each party must obtain random, additive shares (a_i, b_i, c_i) such that $c = a \cdot b$, along with signatures on these shares. Creating shares of triples is essentially identical to the method in [DPSZ12], except we use the correct distributed decryption command of $\mathcal{F}_{\mathsf{KeyGenDec}}$, instead of a possibly faulty method. This means that there is no way the adversary can introduce errors into triples, so we avoid the need for the pairwise sacrificing procedure from [DPSZ12], where half of the triples are wasted to check correctness. The main other difference in our protocol, compared to [DPSZ12], is that we need to setup verification keys for the signature scheme and create signatures on every share, which is more complex than setting up simple MACs.

The setup phase begins by using RandShCtxt to create random, additive shares of the signing key values $\hat{\alpha}_r, \hat{\beta}_{r,i}$, and each party P_j's private verification values $v_{j,r}$, along with ciphertexts encrypting the signing key shares and verification data, in steps 2–3. Next, in steps 4–5, the homomorphism of SHE is used to compute ciphertexts encrypting the signing key, and then ciphertexts encrypting the verification key values $\alpha_j, \beta_{j,i}$ for party P_j, for $i \in [w]$. These verification keys are then privately decrypted to each party.

Given encryptions of the signing key, an encrypted share can be authenticated by homomorphic evaluation of the signing algorithm, followed by private decryption of the signature to the relevant party, as seen in the subprotocol Auth (Fig. 11). Recall that in our scheme, a signature on x_j is given by

$$\boldsymbol{\sigma} = \left(\hat{\alpha}_r \cdot x_j + \hat{\beta}_r\right)_{r=1}^{n},$$

where $\hat{\alpha}_r, \hat{\beta}_r$ are uniformly random elements of the secret key. (Note we have dropped the subscript i on $\hat{\beta}$ here.) For party P_j to obtain a signature on the share x_j, where all parties already know the ciphertext $\langle x_j \rangle$, all parties homomophically compute

$$\langle \boldsymbol{\sigma}|_r \rangle = (\langle \alpha_r \rangle \boxtimes \langle x_j \rangle) \boxplus \langle \beta_r \rangle,$$

for $r \in [n]$, and use private distributed decryption to output $\boldsymbol{\sigma}$ to P_j.

Subprotocol $\mathsf{RandShCtxt}()$**:**

1. Each party samples a random share $x_j \in \mathbb{F}$ and computes $\langle x_j \rangle = \mathsf{SHE.Enc}_{\mathsf{pk}}(x_j)$.
2. Each party broadcasts $\langle x_j \rangle$ and runs the protocol Π_{ZKPoK} to prove that $\langle x_j \rangle$ is correctly generated.
3. Each party P_j outputs $x_j, \langle x_1 \rangle, \ldots, \langle x_n \rangle$.

Subprotocol $\mathsf{ShareDec}(\langle m \rangle, \mathsf{new\_ctxt})$**:**

1. Run $\mathsf{RandShCtxt}$ so that each P_j obtains a share r_j and ciphertexts $\langle r_1 \rangle, \ldots, \langle r_n \rangle$ that encrypt the shares.
2. Homomorphically compute
$$\langle m + r \rangle = \langle m \rangle \boxplus \langle r_1 \rangle \boxplus \cdots \boxplus \langle r_n \rangle.$$
3. Call $\mathcal{F}_{\mathsf{KeyGenDec}}.\mathsf{DistDec}$ to decrypt $\langle m + r \rangle$ so all parties learn $m + r$, where $r = r_1 + \cdots + r_n$.
4. P_1 outputs $m_1 = (m + r) - r_1$ and for all $j \neq 1$, P_j outputs $m_j = -r_j$.
5. If $\mathsf{new\_ctxt} = 1$, each party P_i also outputs $\langle m^* \rangle = \mathsf{SHE.Enc}_{\mathsf{pk}}(m + r) - \langle r \rangle$, where a default, public value is used for the randomness in Enc.

Subprotocol $\mathsf{PrivateDec}(\langle x \rangle, P_j)$**:**

1. P_j samples a random mask $K \leftarrow \mathbb{F}$, broadcasts $\langle K \rangle \leftarrow \mathsf{SHE.Enc}_{\mathsf{pk}}(K)$ and runs Π_{ZKPoK} to prove its correctness.
2. All parties homomorphically compute
$$\langle x + K \rangle = \langle x \rangle \boxplus \langle K \rangle.$$
3. Run $\mathcal{F}_{\mathsf{KeyGenDec}}.\mathsf{DistDec}(\langle x + K \rangle)$ so that all parties obtain the plaintext $x + K$.
4. P_j recovers and outputs x.

Fig. 10. Subprotocols for the preprocessing protocol using SHE

Theorem 3. *The protocol Π_{Prep} (Figs. 11 and 12) securely realises $\mathcal{F}^*_{\mathsf{Prep}}$ (Figs. 8 and 4) with identifiable abort in the $\mathcal{F}_{\mathsf{KeyGenDec}}$-hybrid model, with computational security.*

Proof. See the full version of this work. □

7 Efficiency Evaluation

We now evaluate the concrete efficiency of our protocol, and compare it with the BDOZ [BDOZ11] and SPDZ [DPSZ12] protocols—which only offer security with abort—and the IOZ protocol [IOZ14], which achieves identifiable abort. First we discuss the complexity of broadcast in the two settings, then we compare the online phases of each protocol, and finally discuss the preprocessing.

Protocol Π_{Prep}

To create n_M triples and n_I input values for n parties, set the parameter $w := n \cdot (3n_M + n_I)$.

Setup: Creates the verification keys and ciphertexts encrypting the signing key.

1. Run $\mathcal{F}_{\mathsf{KeyGenDec}}.\mathsf{KeyGen}$ to obtain an SHE public key pk.
2. Run $\mathsf{RandShCtxt}()$ $2n$ times, so each party P_j obtains random elements $\hat{\alpha}_r^j, v_{j,r}$, for $r = 1, \ldots, n$, and everyone obtains ciphertexts $\langle \hat{\alpha}_r^j \rangle, \langle v_{j,r} \rangle$ encrypting these.
3. Run $\mathsf{RandShCtxt}()$ $w(n)$ times, so party P_j obtains $\hat{\beta}_{r,i}^j$, for $r \in [n]$ and $i \in [w]$, and everyone gets the corresponding ciphertexts.
4. Homomorphically compute, for $r \in [n]$:

$$\langle \hat{\alpha}_r \rangle = \langle \hat{\alpha}_r^1 \rangle \boxplus \cdots \boxplus \langle \hat{\alpha}_r^n \rangle$$
$$\langle \hat{\beta}_{r,i} \rangle = \langle \hat{\beta}_{r,i}^1 \rangle \boxplus \cdots \boxplus \langle \hat{\beta}_{r,i}^n \rangle \qquad \text{for } i \in [w].$$

5. Now compute the encrypted verification keys, for $j \in [n]$:

$$\langle \alpha_j \rangle = \bigboxplus_{r=1}^{n} (\langle \hat{\alpha}_r \rangle \boxtimes \langle v_{j,r} \rangle) \quad \text{and} \quad s\langle \beta_{j,i} \rangle = \bigboxplus_{r=1}^{n} \left(\langle \hat{\beta}_{j,r} \rangle \boxtimes \langle v_{j,r} \rangle \right)$$

6. Run the subprotocol $\mathsf{PrivateDec}(\langle \alpha_j \rangle, P_j)$ and $\mathsf{PrivateDec}(\langle \beta_{j,i} \rangle, P_j)$ for $i \in [w]$ and $j \in [n]$, so that each party P_j gets their verification key vk_j.
7. All parties store the ciphertexts $\langle \hat{\alpha}_r \rangle, \langle \hat{\beta}_{r,i} \rangle$, for $r \in [n]$ and $i \in [w]$, and their private verification keys, $\mathsf{vk}_j = (\mathbf{v}_j, \alpha_j, \{\beta_{j,i}\}_{i \in [w]})$.

Fig. 11. Preprocessing protocol with identifiable abort (Setup).

Cost of Broadcast. For MPC with identifiable abort, we denote the cost of broadcasting m field elements by $\mathsf{bc}(m)$. To be able to identify a cheater, this must be done using authenticated broadcast, which requires a PKI setup. The classic Dolev-Strong broadcast [DS83] has message complexity $O(mn^2)$, or a more recent protocol by Hirt and Raykov costs $O(mn)$ for large enough messages [HR14]. Note that any authenticated broadcast protocol requires $\Omega(n)$ rounds of communication if up to $n-1$ parties may be corrupt [GKKO07], which is considerably more expensive than the standard abort setting.

When security with (non-unanimous) abort is allowed (here, for SPDZ and BDOZ), a simple "broadcast with abort" protocol suffices [GL05]. Here, the broadcaster sends x to everyone, then all other parties resend x and check they received the same value. This can be further optimised by performing trivial, insecure broadcasts, and then at the end of the protocol, doing a single broadcast of the hash of all sent values to verify correctness [DKL+12]. This means each broadcast costs $O(n)$ messages, with a one-time $O(n^2)$ cost to verify these at the end.

Protocol Π_{Prep} (continued)

Subprotocol $\mathsf{Auth}(x_j, \langle x_j \rangle, P_j)$**:** Authenticates the share x_j held by party P_j, where $\langle x_j \rangle$ is a (public) SHE encryption of x_j. Start by initialising a counter $\mathsf{cnt} := 1$.

1. All parties homomorphically compute, for $r \in [n]$,
$$\langle \boldsymbol{\sigma}|_r \rangle = \langle \hat{\alpha}_r \rangle \boxtimes \langle x_j \rangle \boxplus \langle \hat{\beta}_{r,\mathsf{cnt}} \rangle.$$
2. Run $\mathsf{PrivateDec}(\langle \boldsymbol{\sigma}_r \rangle, P_j)$ for $r \in [n]$ so that P_j obtains the signature on x_j,
$$\boldsymbol{\sigma} = (\hat{\alpha}_1 \cdot x_j + \hat{\beta}_{1,\mathsf{cnt}}, \ldots, \hat{\alpha}_n \cdot x_j + \hat{\beta}_{n,\mathsf{cnt}}).$$
3. Set $\mathsf{cnt} := \mathsf{cnt} + 1$

Triple: Creates a single, authenticated multiplication triple.

1. The parties run $\mathsf{RandShCtxt}$ twice to obtain additive shares a_j, b_j and public ciphertexts $\langle a_j \rangle, \langle b_j \rangle$ that encrypt the shares.
2. The parties homomorphically compute the ciphertext
$$\langle c \rangle = (\langle a_1 \rangle \boxplus \cdots \boxplus \langle a_n \rangle) \boxtimes (\langle b_1 \rangle \boxplus \cdots \boxplus \langle b_n \rangle).$$
3. The parties run $\mathsf{ShareDec}(\langle c \rangle, 1)$ to obtain decrypted, random shares of c and a fresh ciphertext $\langle c \rangle$.
4. For each $j \in [n]$ and for each symbol $x \in \{a, b, c\}$, run $\mathsf{Auth}(x_j, \langle x_j \rangle, P_j)$ so P_j obtains $\boldsymbol{\sigma}_{x_j} = \mathsf{Sign}(x_j, \mathsf{sk})$.
5. Output the authenticated triple $([\![a]\!], [\![b]\!], [\![c]\!])$.

Input(P_i)**:** Creates a random, authenticated value $[\![r]\!]$, where r is known to P_i.

1. Run $\mathsf{RandShCtxt}$ to obtain additive shares r_j, and public ciphertexts $\langle r_j \rangle$ that encrypt these shares, for $j \in [n]$.
2. Run $\mathsf{Auth}(r_j, \langle r_j \rangle, P_j)$ for every $j \in [n]$ to obtain $[\![r]\!]$.
3. Compute $\langle r \rangle = \langle r_1 \rangle \boxplus \cdots \boxplus \langle r_n \rangle$ and run $\mathsf{PrivateDec}(\langle r \rangle, P_i)$ so that P_i learns r.

Fig. 12. Preprocessing protocol with identifiable abort (authentication and triple generation).

When opening shared values (such as during multiplication) a more efficient method was described in [DPSZ12], where each party first sends their share to P_1, who then computes the sum and sends the result to all parties. This gives a cost of $2(n-1)$ messages per opening, instead of $n(n-1)$ for the previous method (again, the actual broadcast is verified at the end of the protocol).

SPDZ. In the SPDZ protocol (as in [DKL+13]), an authenticated secret share consists of n additive shares on the secret and n MAC shares, so each party stores two field elements. The preprocessing consists of one authenticated share per input, and three per multiplication triple. In the online phase, each input requires one party to broadcast a single value, for a communication cost of $n-1$

field elements. A multiplication consists of two openings, each of which requires all parties to broadcast a share at a cost of $2n(n-1)$ messages using the protocol described above.

In the output phase of SPDZ, first the shares are opened, then a random linear combination of the MACs is checked, and finally all broadcasts must be checked. The MAC and broadcast checking methods both have a communication cost in $O(n^2)$.

BDOZ. In the BDOZ protocol, each party first obtains a fixed, global MAC key α_i. This is fixed for all shared values, so we ignore this cost. For each shared representation $[a]$, party P_i also stores the share a_i, n local MAC keys $\beta^i_{a_1}, \ldots, \beta^i_{a_n}$ and n MAC values $m_1(a_i), \ldots, m_n(a_i)$. Each of these are a single field element, so we get a total storage cost of $2n+1$ field elements per party for each authenticated shared value.

If we assume an optimised version of the original protocol, so that all parties open their shares a_i using the SPDZ broadcast and then delay MAC checking until the **Output** stage, then the online communication costs are essentially the same as SPDZ.

IOZ. The IOZ online phase takes any semi-honest MPC protocol (with preprocessing), and compiles it to a malicious protocol with identifiable abort, similarly to the GMW paradigm [GMW87]. The compiled protocol has a preprocessing phase that outputs the original semi-honest preprocessing data, authenticated using IT signatures, as well as additional data for zero-knowledge proofs using MPC-in-the-head, which are required for each round of the semi-honest protocol. Using a semi-honest GMW protocol with multiplication triples as a base, the preprocessing data already contains the same number of IT signatures as our protocol, before taking into account the zero-knowledge proofs.

Each zero-knowledge proof requires storing m IT signatures as preprocessing, where m is the number of parties in the MPC-in-the-head method. In [IOZ14], they choose $m = O(\kappa)$ for statistical security level κ, whereas [GMO16] use $m = 3$, but require repeating the proof κ times to get negligible soundness error. Since repeating the proof requires extra preprocessing for each repetition, we obtain a very rough lower bound of storing κ signatures (or $\kappa \cdot n$ field elements) per proof with either approach.

For the communication costs, we only take into account the cost for every party to broadcast one proof, plus the (at least) two signatures that are broadcast in the Π_{SCP} protocol of [IOZ14]. According to [GMO16, Sect. 4.2], the proof size is at least $2 \cdot \kappa \cdot \log_2(|\mathbb{F}|)$, for a proof with soundness $2^{-\kappa}$, generously ignoring the size of the circuit representing the NP-relation being proven and other constant factors. If the IOZ version of MPC-in-the-head is used instead, each proof still requires broadcasting $t = O(\kappa)$ field elements in the Π_{1SCP} protocol, so would not have significantly better complexity.

Table 1. Comparison of the storage and communication costs of the protocols, measured in number of field elements. $N = n_I + 2n_M$ (where n_I is number of inputs, n_M is number of multiplications), D is the multiplicative depth of the circuit, and κ is a statistical security parameter

Protocol	Prep. storage		Online Comms.			Rounds
	Input	Mult.	Input	Mult.	Output	
SPDZ	2	6	$n-1$	$4(n-1)$	$O(n^2)$	$O(D)$
BDOZ	$2n+1$	$6n+3$	$n-1$	$4(n-1)$	$O(n^2)$	$O(D)$
IOZ (at least)	κn	κn	$\kappa \cdot \mathsf{bc}(1)$	$\kappa n \cdot \mathsf{bc}(1) + 2n \cdot \mathsf{bc}(n)$	$\kappa n \cdot \mathsf{bc}(1) + 2n \cdot \mathsf{bc}(n)$	$O(D \cdot n)$
Ours	$2n+1$	$6n+3$	$\mathsf{bc}(1)$	$2n(n-1)$	$n \cdot \mathsf{bc}(n + 2n_M + 1)$	$O(D+n)$

Comparison of the Online Phases. The complexities in Table 1 for our protocol can be derived from the analysis in Sect. 5.2. We have ignored storage costs for the $\mathbf{v}_j, \alpha_j$ parts of the verification keys, as these are independent of the number of signatures. Our protocol is roughly a factor of n times worse than SPDZ in terms of storage and communication cost, and has similar costs to BDOZ, bar the requirement for two rounds of authenticated broadcast. Compared with the IOZ protocol, we improve by at least a multiplicative factor in the security parameter, as well as a greatly reduced number of broadcast rounds.

7.1 Preprocessing Cost

For preprocessing, the main factor affecting computation and communication costs in [DPSZ12,DKL+13] is the number of zero-knowledge proofs of correct ciphertext generation that are required, so this is what we measure in our protocol.

The main cost of our preprocessing protocol, compared with [DPSZ12], is to produce the signatures and verification keys for each shared value, instead of MACs as in SPDZ. The **Setup** phase of our protocol (Fig. 11) generates verification keys, whose size depends on the number of signatures. Ignoring any costs independent of the number of signatures, this requires n calls to RandShCtxt for each signature. Each RandShCtxt call requires n zero-knowledge proofs, and since there are n signatures per shared value (one per share) this gives a total of $O(n^3)$ zero-knowledge proofs per multiplication triple or input tuple. This dominates the cost of creating the n signatures for each shared value, which is in $O(n^2)$.

In contrast, SPDZ shared MAC values only require $O(n)$ proofs each, so our protocol requires $O(n^2)$ more proofs than SPDZ in the preprocessing phase. It as an interesting problem to see if this can be reduced, although it seems that with IT signatures a factor of at least $O(n)$ is inherent, due to the signature size.

For comparison, note that the IOZ preprocessing transformation, which is based on any protocol in the OT-hybrid model, uses a verifiable OT protocol which broadcasts a message for every message of the OT protocol, adding an $O(n)$ overhead on top of the OT-hybrid protocol. When accounting for producing the larger amount of preprocessing data needed for the online phase, this gives

an overall overhead of $O(n^2)$, the same as ours. However, it seems unlikely that an OT-based protocol for $\mathcal{F}_{\mathsf{Prep}}$ could be much more efficient than using SHE, mainly because the need for adaptive security in IOZ prevents the use of efficient OT extensions [LZ13].

References

[AJL+12] Asharov, G., Jain, A., López-Alt, A., Tromer, E., Vaikuntanathan, V., Wichs, D.: Multiparty computation with low communication, computation and interaction via threshold FHE. In: Pointcheval, D., Johansson, T. (eds.) EUROCRYPT 2012. LNCS, vol. 7237, pp. 483–501. Springer, Heidelberg (2012)

[AL10] Aumann, Y., Lindell, Y.: Security against covert adversaries: efficient protocols for realistic adversaries. J. Cryptology **23**(2), 281–343 (2010)

[BDOZ11] Bendlin, R., Damgård, I., Orlandi, C., Zakarias, S.: Semi-homomorphic encryption and multiparty computation. In: Paterson, K.G. (ed.) EUROCRYPT 2011. LNCS, vol. 6632, pp. 169–188. Springer, Heidelberg (2011)

[BDTZ16] Baum, C., Damgård, I., Toft, T., Zakarias, R.: Better preprocessing for secure multiparty computation. In: Manulis, M., Sadeghi, A.-R., Schneider, S. (eds.) ACNS 2016. LNCS, vol. 9696, pp. 327–345. Springer, Heidelberg (2016). doi:10.1007/978-3-319-39555-5_18

[Bea91] Beaver, D.: Efficient multiparty protocols using circuit randomization. In: Feigenbaum, J. (ed.) CRYPTO 1991. LNCS, vol. 576, pp. 420–432. Springer, Heidelberg (1992)

[BFKW09] Boneh, D., Freeman, D.M., Katz, J., Waters, B.: Signature schemes for network coding: signing a linear subspace. In: Public Key Cryptography - PKC, pp. 68–87 (2009)

[BGV12] Brakerski, Z., Gentry, C., Vaikuntanathan, V.: (leveled) fully homomorphic encryption without bootstrapping. In: ITCS 2012, pp. 309–325 (2012)

[BLOO11] Beimel, A., Lindell, Y., Omri, E., Orlov, I.: $1/p$-secure multiparty computation without honest majority and the best of both worlds. In: Rogaway, P. (ed.) CRYPTO 2011. LNCS, vol. 6841, pp. 277–296. Springer, Heidelberg (2011)

[Can01] Canetti, R.: Universally composable security: a new paradigm for cryptographic protocols. In: FOCS, pp. 136–145 (2001)

[CDD+99] Cramer, R., Damgård, I.B., Dziembowski, S., Hirt, M., Rabin, T.: Efficient multiparty computations secure against an adaptive adversary. In: Stern, J. (ed.) EUROCRYPT 1999. LNCS, vol. 1592, pp. 311–326. Springer, Heidelberg (1999)

[CJL09] Charles, D.X., Jain, K., Lauter, K.E.: Signatures for network coding. IJICoT **1**(1), 3–14 (2009)

[CL14] Cohen, R., Lindell, Y.: Fairness versus guaranteed output delivery in secure multiparty computation. In: Sarkar, P., Iwata, T. (eds.) ASIACRYPT 2014, Part II. LNCS, vol. 8874, pp. 466–485. Springer, Heidelberg (2014)

[Cle86] Cleve, R.: Limits on the security of coin flips when half the processors are faulty (extended abstract). In: STOC 1986, pp. 364–369 (1986)

[CR91] Chaum, David, Roijakkers, Sandra: Unconditionally-Secure Digital Signatures. In: Menezes, Alfred, J., Vanstone, Scott, A. (eds.) CRYPTO 1990. LNCS, vol. 537, pp. 206–214. Springer, Heidelberg (1991). doi:10.1007/3-540-38424-3_15

[DKL+12] Damgård, I., Keller, M., Larraia, E., Miles, C., Smart, N.P.: Implementing AES via an actively/covertly secure dishonest-majority MPC protocol. In: SCN, pp. 241–263 (2012)
[DKL+13] Damgård, I., Keller, M., Larraia, E., Pastro, V., Scholl, P., Smart, N.P.: Practical covertly secure MPC for dishonest majority – or: breaking the SPDZ limits. In: Crampton, J., Jajodia, S., Mayes, K. (eds.) ESORICS 2013. LNCS, vol. 8134, pp. 1–18. Springer, Heidelberg (2013)
[DPSZ12] Damgård, I., Pastro, V., Smart, N., Zakarias, S.: Multiparty computation from somewhat homomorphic encryption. In: Safavi-Naini, R., Canetti, R. (eds.) CRYPTO 2012. LNCS, vol. 7417, pp. 643–662. Springer, Heidelberg (2012)
[DS83] Dolev, D., Strong, H.R.: Authenticated algorithms for byzantine agreement. SIAM J. Comput. **12**(4), 656–666 (1983)
[DZ13] Damgård, I., Zakarias, S.: Constant-overhead secure computation of boolean circuits using preprocessing. In: Sahai, A. (ed.) TCC 2013. LNCS, vol. 7785, pp. 621–641. Springer, Heidelberg (2013)
[GKKO07] Garay, J.A., Katz, J., Koo, C.-Y., Ostrovsky, R.: Round complexity of authenticated broadcast with a dishonest majority. In: FOCS 2007, pp. 658–668 (2007)
[GL05] Goldwasser, S., Lindell, Y.: Secure multi-party computation without agreement. J. Cryptology **18**(3), 247–287 (2005)
[GMO16] Giacomelli, I., Madsen, J., Orlandi, C.: ZKBoo: faster zero-knowledge for boolean circuits. In: 25th USENIX Security Symposium (USENIX Security 2016), pp. 1069–1083 (2016)
[GMW87] Goldreich, O., Micali, S., Wigderson, A.: How to play any mental game or a completeness theorem for protocols with honest majority. In: STOC 1987, pp. 218–229 (1987)
[Gol01] Goldreich, O.: The Foundations of Cryptography - Basic Techniques, vol. 1. Cambridge University Press, Cambridge (2001)
[GVW15] Gorbunov, S., Vaikuntanathan, V., Wichs, D.: Leveled fully homomorphic signatures from standard lattices. In: STOC 2015, pp. 469–477 (2015)
[HR14] Hirt, M., Raykov, P.: Multi-valued byzantine broadcast: The t¡n case. In: Sarkar, P., Iwata, T. (eds.) ASIACRYPT 2014. LNCS, vol. 8874, pp. 448–465. Springer, Heidelberg (2014). doi:10.1007/978-3-662-45608-8_24
[HSZI00] Hanaoka, G., Shikata, J., Zheng, Y., Imai, H.: Unconditionally secure digital signature schemes admitting transferability. In: Okamoto, T. (ed.) ASIACRYPT 2000. LNCS, vol. 1976, pp. 130–142. Springer, Heidelberg (2000). doi:10.1007/3-540-44448-3_11
[IKOS07] Ishai, Y., Kushilevitz, E., Ostrovsky, R., Sahai, A.: Zero-knowledge from secure multiparty computation. In: STOC 2007, pp. 21–30 (2007)
[IOS12] Ishai, Y., Ostrovsky, R., Seyalioglu, H.: Identifying cheaters without an honest majority. In: Cramer, R. (ed.) TCC 2012. LNCS, vol. 7194, pp. 21–38. Springer, Heidelberg (2012)
[IOZ14] Ishai, Y., Ostrovsky, R., Zikas, V.: Secure multi-party computation with identifiable abort. In: Garay, J.A., Gennaro, R. (eds.) CRYPTO 2014, Part II. LNCS, vol. 8617, pp. 369–386. Springer, Heidelberg (2014)
[LZ13] Lindell, Y., Zarosim, H.: On the feasibility of extending oblivious transfer. In: Sahai, A. (ed.) TCC 2013. LNCS, vol. 7785, pp. 519–538. Springer, Heidelberg (2013)
[PW92] Pfitzmann, B., Waidner, M.: Unconditional byzantine agreement for any number of faulty processors. In: STACS 1992, pp. 339–350 (1992)

[Sey12] Seyalioglu, H.A.-J.: Reducing trust when trust is essential. Ph.D. thesis, University of California, Los Angeles 2012. https://escholarship.org/uc/item/7301296m

[SS11] Swanson, C.M., Stinson, D.R.: Unconditionally secure signature schemes revisited. In: Fehr, S. (ed.) ICITS 2011. LNCS, vol. 6673, pp. 100–116. Springer, Heidelberg (2011)