

# Efficient Secure Multiparty Computation with Identifiable Abort

Carsten Baum<sup>1\*</sup>, Emmanuela Orsini<sup>2 †</sup>, and Peter Scholl<sup>2 ‡</sup>

<sup>1</sup> Department of Computer Science, Aarhus University  
cbaum@cs.au.dk

<sup>2</sup> Department of Computer Science, University of Bristol  
{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), and 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) and other recent works with applicable efficiency improvements.

**Keywords:** Secure Multiparty Computation, Identifiable Abort

---

\*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, within which part of this work was performed; and also from the CFEM research center (supported by the Danish Strategic Research Council) within which part of this work was performed.

<sup>†</sup>Supported in part by ERC Advanced Grant ERC-2010-AdG-267188-CRIPTO.

<sup>‡</sup>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.

# Table of Contents

Efficient Secure Multiparty Computation with Identifiable Abort .....	1
<i>Carsten Baum, Emmanuela Orsini, and Peter Scholl</i>	
1 Introduction .....	3
2 Preliminaries .....	6
2.1 Notation .....	6
2.2 Model .....	6
2.3 Broadcast Channel .....	7
3 Homomorphic Information-Theoretic Signatures .....	7
4 Construction of HITS .....	10
4.1 Security of Theorem 1 .....	11
4.2 Lower Bounds on IT Signatures .....	14
5 Online Phase for Efficient MPC with Identifiable Abort .....	15
5.1 Security .....	17
5.2 An Optimised Protocol .....	20
5.3 Security .....	21
6 Preprocessing with Identifiable Abort .....	22
6.1 Modified Functionality $\mathcal{F}_{\text{Prep}}^*$ .....	23
6.2 SHE Scheme Requirements .....	23
6.3 Basic Subprotocols .....	24
6.4 Creating the Preprocessing Data .....	24
6.5 Security of Theorem 3 .....	26
7 Efficiency Evaluation .....	28
7.1 Preprocessing Cost .....	30
8 From MPC with Identifiable Abort to MPC with Public Verifiability. ....	31
9 Efficient MPC with Identifiable Abort over Small Fields .....	32
9.1 Packing Shares into Codewords .....	32
9.2 Evaluating the Circuit .....	33
9.3 Identifiable Abort and Efficiency .....	34

# 1 Introduction

Multiparty Computation (MPC) deals with the problem of jointly computing a function among a set of mutually distrusting parties with some security guarantees such as correctness of the output and privacy of the input. 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<sup>+</sup>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 share, all parties first provide their shares of both the value and the 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, Sec. 3.6]), depicted in Fig. 1, shows that this is not sufficient.

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  $m(x_1)_4^*$ .

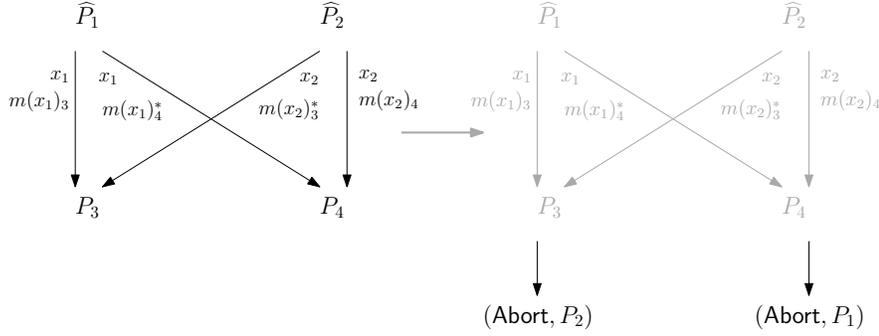


Fig. 1. Counterexample for identifiable abort with pairwise MACs

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  $(\text{Abort}, P_3)$ , 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<sup>+</sup>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.

Our definition of HITS is slightly different to [SS11], as we only consider a single, honest signer, instead of allowing any user to sign messages. This simplifies the definitions, but requires taking

some additional care in proving the consistency property, which in our case is no longer implied by unforgeability.

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<sup>1</sup> [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. We show that 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 an 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 Section 8, we present a simple modification of our ID-MPC scheme that implies *public verifiability*: 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. In 9, we present an information-theoretic MPC protocol in the preprocessing model that is optimised for a different range of field sizes and which is related to the MiniMACS protocol [DZ13]. This MPC protocol also uses HITS but can tolerate lower security guarantees than our main protocol.

**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  $O(\sigma)$  times less efficient than ours, to achieve soundness error  $2^{-\sigma}$ . Note that Ishai et al. also use IT signatures for authenticating values, similarly to our usage, but without the homomorphic property that allows

---

<sup>1</sup>This is not needed in SPDZ, because a simple ‘broadcast with abort’ technique can be performed in just two rounds.

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<sup>+</sup>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 Section 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 Section 4 describes our construction. Our information-theoretic ID-MPC protocol in the preprocessing model is presented in Section 5, followed by the preprocessing using SHE in Section 6. In Section 7 we evaluate the efficiency of our protocols, compared with the previous state of the art. In Section 8 we show how to upgrade the security of our ID-MPC protocol to be publicly verifiable by an external party, and we give an ID-MPC construction over small fields in Section 9.

## 2 Preliminaries

### 2.1 Notation

Throughout this work, we denote by  $\kappa$  and  $\lambda$  the statistical, resp. computational, security parameters, and we use the standard definition of negligible (denoted by  $\text{negl}(\kappa)$ ) and overwhelming function from [Gol01]. We use bold lower case letters for vectors, i.e.  $\mathbf{v}$  and refer to the  $i$ th 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  $\{1, \dots, n\}$ . The  $n$  parties in the protocol are denoted as  $\mathcal{P} = \{P_1, \dots, P_n\}$ , 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  $[\mathcal{F}]_{\perp}^{\text{ID}}$  to be the functionality with the same behaviour as  $\mathcal{F}$ , except that at any time the adversary may send a special command  $(\text{Abort}, P_i)$ , where  $i \in \mathcal{I}$ , which causes  $[\mathcal{F}]_{\perp}^{\text{ID}}$  to output  $(\text{Abort}, P_i)$  to all parties.

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

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 require 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 to achieve this, 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  $\ell$ -bit messages. Hirt and Raykov [HR14] presented a protocol that reduces the communication cost to  $O(\ell n)$  when  $\ell$  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<sup>2</sup> 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].

---

<sup>2</sup>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.

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

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

$\sigma \leftarrow \text{Sign}(m, \text{sk})$  is a deterministic algorithm that takes as input a message  $m \in \mathcal{M}$  and signing key  $\text{sk}$ , and outputs a signature  $\sigma$ .

$\sigma \leftarrow \text{Eval}(f, (\sigma_1, \dots, \sigma_\ell))$  homomorphically evaluates the function  $f \in \mathcal{F}$  on a list of signatures  $(\sigma_1, \dots, \sigma_\ell)$ .

$0/1 \leftarrow \text{Ver}(m, \sigma, f, \text{vk}_j)$  takes as input a message  $m$ , a signature  $\sigma$ , a function  $f$  and  $P_j$ 's verification key  $\text{vk}_j$ , 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 ( $(w, \tau)$ -security).** A HITS scheme  $(\text{Gen}, \text{Sign}, \text{Ver}, \text{Eval})$  is  $(w, \tau)$ -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  $\pi_i(m_1, \dots, m_\ell) = m_i$ . Then we require that for every pair  $(\text{sk}, \text{vk})$  output by  $\text{Gen}$ , for any  $(m_1, \dots, m_\ell) \in \mathcal{M}^\ell$ , and for all  $i \in [\ell], j \in [n]$ ,

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

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

$$\text{Ver}(f(m_1, \dots, m_\ell), \text{Eval}(f, (\text{Sign}(m_1, \text{sk}), \dots, \text{Sign}(m_\ell, \text{sk}))), f, \text{vk}_j) = 1.$$

**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  $(\text{sk}, \text{vk}) \leftarrow \text{Gen}(1^\kappa, w)$  and sends  $\{\text{vk}_i\}_{i \in \mathcal{I}}$  to the adversary.
2.  $\mathcal{A}$  may query  $\mathcal{C}$  adaptively up to a maximum of  $w$  times for signatures. Let  $m_1, \dots, m_w$  be the list of messages queried to  $\mathcal{C}$ .

3.  $\mathcal{A}$  outputs a function  $f \in \mathcal{F}$ , a list of indices  $\{i_1, \dots, i_\ell\} \subseteq [w']$  in ascending order, a target message  $m^*$  and a signature  $\sigma^*$ .
4.  $\mathcal{A}$  wins if  $m^* \neq f(m_{i_1}, \dots, m_{i_\ell})$  and there exists  $j \in [n] \setminus \mathcal{I}$  for which

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

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

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

**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 <sup>3</sup> if there exist  $i, j \in [n] \setminus \mathcal{I}$  such that

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

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

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

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  $\sigma$  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  $m_1, \dots, m_\ell$  other than that given by  $f(m_1, \dots, m_\ell)$ . 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  $(\text{Gen}, \text{Sign}, \text{Eval}, \text{Ver})$  is evaluation private if there exists a PPT algorithm Sim that, for every  $(\text{sk}, \text{vk}) \leftarrow \text{Gen}$ , for every function  $f \in \mathcal{F}$ , for all messages  $m_1, \dots, m_\ell$  with  $m = f(m_1, \dots, m_\ell)$ ,  $\sigma_i = \text{Sign}(m_i, \text{sk})$  and  $\sigma = \text{Eval}(f, \sigma_1, \dots, \sigma_\ell)$ , computes

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

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.

---

<sup>3</sup>Note there is no requirement that  $m^* \neq f(m_{i_1}, \dots, m_{i_\ell})$ .

## 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  $\mathcal{F}$  to be the set of all affine transformations  $f : \mathbb{F}^w \rightarrow \mathbb{F}$  (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.

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

1. Sample  $\hat{\alpha}_1, \dots, \hat{\alpha}_n \leftarrow \mathbb{F}$  and  $\hat{\beta}_{i,1}, \dots, \hat{\beta}_{i,n} \leftarrow \mathbb{F}$  for each  $i \in [w]$ .
2. For each verifier  $P_j$ , sample  $\mathbf{v}_j = (v_{j,1}, \dots, v_{j,n}) \leftarrow \mathbb{F}^n$  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  $\mathbf{sk} = \left( \left\{ \hat{\alpha}_r, \{\hat{\beta}_{i,r}\}_{i \in [w]} \right\}_{r=1}^n \right)$ ,  $\mathbf{vk} = \left( \mathbf{v}_j, \alpha_j, \{\beta_{j,i}\}_{i \in [w]} \right)_{j=1}^n$ .

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

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

**Eval( $f, (\boldsymbol{\sigma}_1, \dots, \boldsymbol{\sigma}_w)$ ):** Let  $f : \mathbb{F}^w \rightarrow \mathbb{F}$  be defined by

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

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:

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

**Ver( $m, \boldsymbol{\sigma}, f, \mathbf{vk}_j$ ):** 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

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

where  $\boldsymbol{\sigma}|_r$  indicates the  $r$ th component of  $\boldsymbol{\sigma}$ . If the check passes output 1, otherwise 0.

**Theorem 1.** *Let  $\kappa \in \mathbb{N}$  be the security parameter,  $\mathbb{F}$  be a finite field. Moreover, define  $\mathcal{M} := \mathbb{F}$  and  $\mathcal{F}$  be the set of affine maps from  $\mathbb{F}^w$  to  $\mathbb{F}$ , then the tuple of algorithms  $(\text{Gen}, \text{Sign}, \text{Eval}, \text{Ver})$  is a  $(w, 2/|\mathbb{F}|)$ -secure HITS with evaluation privacy.*

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

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

#### 4.1 Security of Theorem 1

*Proof (of Theorem 1).* We prove that our construction satisfies all the properties required for a HITS (Definitions 3 and 4).

*Signing and evaluation correctness.* Signing correctness is implied by evaluation correctness for  $f = \pi_i$ , so we show the latter.

Let  $w, n \in \mathbb{N}^+$  and  $j \in [n]$  be arbitrarily chosen. Moreover, let  $(\mathbf{sk}, \mathbf{vk}) \leftarrow \text{Gen}(1^\kappa, w)$ , take arbitrary  $(m_1, \dots, m_w) \in \mathcal{M}^w$  and let  $f \in \mathcal{F}$  be defined by  $f(m_1, \dots, m_w) = \sum_{i=1}^w \mu_i \cdot m_i + c$ , for some  $\mu_1, \dots, \mu_w, c \in \mathbb{F}$ . Applying the Sign algorithm for  $i = 1, \dots, w$ , we obtain

$$\sigma_i = \left( \hat{\alpha}_r \cdot m_i + \hat{\beta}_{i,r} \right)_{r=1}^n.$$

Using these signatures and  $f$ , Eval then computes  $\sigma^f = \sum_{i=1}^w \mu_i \cdot \sigma_i$ . The verification algorithm will compute

$$\beta_j = \sum_{i=1}^w \mu_i \cdot \beta_{j,i} - c \cdot \alpha_j = \sum_{i=1}^w \mu_i \sum_{r=1}^n \hat{\beta}_{i,r} \cdot v_{j,r} - c \cdot \alpha_j.$$

Therefore

$$\begin{aligned} \beta_j + \alpha_j \cdot f(m_1, \dots, m_w) &= \sum_{i=1}^w \mu_i \sum_{r=1}^n \hat{\beta}_{i,r} \cdot v_{j,r} - c \cdot \alpha_j + \\ &\quad \sum_{r=1}^n \hat{\alpha}_r \cdot v_{j,r} \sum_{i=1}^w \mu_i \cdot m_i + \alpha_j \cdot c \\ &= \sum_{i=1}^w \mu_i \sum_{r=1}^n v_{j,r} \cdot \left( \hat{\beta}_{i,r} + \hat{\alpha}_r \cdot m_i \right) \\ &= \sum_{r=1}^n v_{j,r} \sum_{i=1}^w \mu_i \cdot \sigma_i|_r \\ &= \sum_{r=1}^n v_{j,r} \cdot \sigma^f|_r \end{aligned}$$

and the algorithm will output 1.

*Unforgeability.* Before we start proving the unforgeability and consistency of the scheme, we prove a bound on the adversary's uncertainty about  $\mathbf{sk}$ . In particular, we show that there are always at least  $|\mathbb{F}|$  vectors  $(\hat{\alpha}'_1, \dots, \hat{\alpha}'_n)$  that are consistent with the adversary's view, and all are equally likely.

**Lemma 1.** *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.  $\mathcal{C}$  samples  $(\mathbf{sk}, \mathbf{vk}) \leftarrow \text{Gen}(1^\kappa, w)$  and sends  $\{\mathbf{vk}_j\}_{j \in \mathcal{I}}$  to the adversary.
2.  $\mathcal{A}$  may query  $\mathcal{C}$  adaptively up to a maximum of  $w$  times for signatures.
3.  $\mathcal{A}$  outputs  $(\hat{\alpha}'_1, \dots, \hat{\alpha}'_n)$ .

Then  $\Pr[(\hat{\alpha}'_1, \dots, \hat{\alpha}'_n) = (\hat{\alpha}_1, \dots, \hat{\alpha}_n)] \leq 1/|\mathbb{F}|$  when the probability is taken over the randomness of  $\text{Gen}$  and  $\mathcal{A}$ .

*Proof.* In Step 1 of the game,  $\mathcal{A}$  obtains

$$\{\mathbf{vk}_j\}_{j \in \mathcal{I}} = \{v_{j,1}, \dots, v_{j,n}, \alpha_j, \{\beta_{j,i}\}_{i \in [w]}\}_{j \in \mathcal{I}}.$$

It then in Step 2 provides messages  $m_1, \dots, m_w \in \mathbb{F}$  and obtains

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

Collect all of  $\mathcal{A}$ 's information about the  $\hat{\alpha}_r, \hat{\beta}_{i,r}$  into a system of linear equations. For simplicity, consider the setting where  $w = 1$  and  $\mathcal{I} = \{1, \dots, n-1\}$ . We obtain

$$\begin{pmatrix} v_{1,1} & \cdots & v_{1,n} & 0 & \cdots & 0 \\ v_{2,1} & \cdots & v_{2,n} & 0 & \cdots & 0 \\ \vdots & \ddots & \vdots & \vdots & \ddots & \vdots \\ v_{n-1,1} & \cdots & v_{n-1,n} & 0 & \cdots & 0 \\ 0 & \cdots & 0 & v_{1,1} & \cdots & v_{1,n} \\ \vdots & \ddots & \vdots & \vdots & \ddots & \vdots \\ 0 & \cdots & 0 & v_{n-1,1} & \cdots & v_{n-1,n} \\ m & \cdots & 0 & 1 & \cdots & 0 \\ \vdots & \ddots & \vdots & \vdots & \ddots & \vdots \\ 0 & \cdots & m & 0 & \cdots & 1 \end{pmatrix} \cdot \begin{pmatrix} \hat{\alpha}_1 \\ \vdots \\ \hat{\alpha}_n \\ \hat{\beta}_{1,1} \\ \vdots \\ \hat{\beta}_{1,n} \end{pmatrix} = \begin{pmatrix} \alpha_1 \\ \vdots \\ \alpha_{n-1} \\ \beta_{1,1} \\ \vdots \\ \beta_{n-1,1} \\ \boldsymbol{\sigma}_1|_1 \\ \vdots \\ \boldsymbol{\sigma}_1|_n \end{pmatrix}$$

The system has  $3n - 2$  equations but only  $2n$  variables. Fortunately the matrix on the left is not full-rank: Assuming  $m \neq 0$ , the first  $n - 1$  rows can be used to obtain the next  $n - 1$  rows using the last  $n$  rows, so we can safely remove them.<sup>4</sup> If  $m = 0$ , then the  $n$  leftmost columns are 0 for the last  $2n - 1$  rows, out of which at most  $n$  can be linearly independent.

Therefore, the matrix has rank at most  $2n - 1$  for  $2n$  unknowns and  $\mathcal{A}$  cannot recover the correct sk. Instead, by fixing a certain  $\hat{\alpha}'_i$  there exists at least one  $(\hat{\alpha}'_1, \dots, \hat{\alpha}'_n)$  compatible with this choice. The same applies in the case of  $w$  signatures where  $\mathcal{A}$  will obtain a matrix with  $(w + 1) \cdot n - 1$  independent rows, but has to solve for  $(w + 1) \cdot n$  unknown values. Since all the  $\hat{\alpha}_i$  were chosen uniformly at random each such choice of  $\hat{\alpha}'_i$  yields a candidate vector that is equally likely.  $\square$

In the definition of the unforgeability game,  $\mathcal{A}$  chooses a function  $f$  by fixing  $\mu_1, \dots, \mu_l, c$ . It is easy to see that the choice of  $c$  does not make any difference, so assume that  $c = 0$  for simplicity. Let  $m_{i_1}, \dots, m_{i_l}$  be the messages queried to  $\mathcal{C}$  and  $\boldsymbol{\sigma}_{i_1}, \dots, \boldsymbol{\sigma}_{i_l}$  the responses. We conclude that, for all  $j \in [n] \setminus \mathcal{I}$ , the equation

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

with

$$\boldsymbol{\sigma}^f = \sum_{k=1}^l \mu_k \cdot \boldsymbol{\sigma}_{i_k} \quad \text{and} \quad m^f = \sum_{k=1}^l \mu_k \cdot m_{i_k} \quad \text{and} \quad \beta_j^f = \sum_{k=1}^l \mu_k \cdot \beta_{j,i_k},$$

<sup>4</sup>Multiply the first row with  $-m$ , the  $2n - 1$ st row with  $v_{1,1}$  and add these together. Then continue with row 2n,  $2n + 1$ , etc.

must hold due to evaluation correctness. Now  $\mathcal{A}$  picks  $m^* \in \mathbb{F}$  such that  $m^* \neq f(m_{i_1}, \dots, m_{i_l})$  and a signature  $\sigma^*$ , and assume for the sake of contradiction that  $\mathcal{A}$  wins the game with probability  $p > 1/|\mathbb{F}|$ . Then

$$\beta_j^f + \alpha_j \cdot m^* = \sum_{r=1}^n v_{j,r} \cdot \sigma^*|_r \quad (2)$$

holds for some  $j \in [n] \setminus \mathcal{I}$ . We can subtract (2) from (1) and obtain

$$\begin{aligned} 0 &= \sum_{r=1}^n v_{j,r} \cdot \left( \frac{\sigma^f|_r - \sigma^*|_r}{m^f - m^*} \right) - \alpha_j \\ &= \sum_{r=1}^n v_{j,r} \cdot \left( \frac{\sigma^f|_r - \sigma^*|_r}{m^f - m^*} - \hat{\alpha}_r \right). \end{aligned} \quad (3)$$

$\mathcal{A}$  can make Equation 3 hold by either providing the trivial solution or another vector in the kernel of the map. In Lemma 1 we established that the view of  $\mathcal{A}$  contains at least  $|\mathbb{F}|$  candidates for  $(\hat{\alpha}_1, \dots, \hat{\alpha}_n)$ , all of which being equally likely. Therefore, finding the trivial solution with probability  $> 1/|\mathbb{F}|$  contradicts Lemma 1. On the other hand, by the rank nullity theorem, the linear map induced by the right-hand side of (3) has a kernel of dimension  $n - 1$ . All  $v_{j,1}, \dots, v_{j,n}$  were chosen uniformly at random and are unknown by  $\mathcal{A}$ , so Equation (3) will hold with probability  $|\mathbb{F}^{n-1}|/|\mathbb{F}^n| = 1/|\mathbb{F}|$ . Hence the overall probability is  $2/|\mathbb{F}| - 1/|\mathbb{F}|^2$ .

*Consistency.* Assume that  $\mathcal{A}$  can break consistency with probability  $p > 1/|\mathbb{F}|$ . By definition of the game, it must then hold that at least one of  $\sigma^* \neq \sum_{k=1}^l \mu_k \cdot \sigma_{i_k}$  or  $m^* \neq f(m_{i_1}, \dots, m_{i_l})$ : If  $m^* = f(m_{i_1}, \dots, m_{i_l})$  and  $\sigma^* = \sum_{k=1}^l \mu_k \cdot \sigma_{i_k}$  then by evaluation correctness,  $\text{Ver}(m^*, \sigma^*, f, \mathbf{vk}_j) = 1, \forall j \in [n]$ , which contradicts the winning condition.

First suppose that  $m^* = f(m_1, \dots, m_l)$ , so  $\sigma^* \neq \sum_{k=1}^l \mu_k \cdot \sigma_{i_k}$ . Then by the winning condition, it must hold that

$$0 = \sum_{r=1}^n v_{j,r} \cdot \underbrace{\left( \sigma^*|_r - (m^* - c) \cdot \hat{\alpha}_r - \sum_{k=1}^l \mu_k \cdot \hat{\beta}_{i_k,r} \right)}_{=x_r}, \quad (4)$$

for at least one  $j \in [n] \setminus \mathcal{I}$ . Suppose at least one of the values  $x_1, \dots, x_n$  is non-zero. Then due to the random choice of the  $v_{j,r}$  the above sum will only be 0 with probability at most  $1/|\mathbb{F}|$  (as in the Unforgeability proof). Thus, it must hold that  $x_1 = x_2 = \dots = x_n = 0$  and therefore

$$\forall r \in [n] : (m^* - c) \cdot \hat{\alpha}_r + \sum_{k=1}^l \mu_k \cdot \hat{\beta}_{i_k,r} = \sigma^*|_r.$$

Hence for every  $r$  it must hold that

$$\begin{aligned} \sigma^*|_r &= \hat{\alpha}_r \cdot (m^* - c) + \sum_{k=1}^l \mu_k \cdot \hat{\beta}_{i_k,r} \\ &= \hat{\alpha}_r \cdot (f(m_{i_1}, \dots, m_{i_l}) - c) + \sum_{k=1}^l \mu_k \cdot \hat{\beta}_{i_k,r} \end{aligned}$$

$$\begin{aligned}
&= \hat{\alpha}_r \cdot (\mu_1 \cdot m_{i_1} + \dots \mu_l \cdot m_{i_l}) + \sum_{k=1}^l \mu_k \cdot \hat{\beta}_{i_k, r} \\
&= \sum_{k=1}^l \mu_k \cdot (\hat{\alpha}_r \cdot m_{i_k} + \hat{\beta}_{i_k, r}) \\
&= \sum_{k=1}^l \mu_k \cdot \sigma_{i_k}|_r,
\end{aligned}$$

which contradicts the assumption. We therefore observe that the cheating probability in this case can be bounded by  $1/|\mathbb{F}|$ .

Consider the case where  $m^* \neq f(m_{i_1}, \dots, m_{i_l})$ , then the setting equals the Unforgeability game for which we upper-bounded the success of  $\mathcal{A}$  as  $1/|\mathbb{F}|$  as well. Hence the overall probability is  $3/|\mathbb{F}| - 1/|\mathbb{F}|^2$ .

*Evaluation privacy.* Let  $(\mathbf{sk}, \mathbf{vk}) \leftarrow \text{Gen}(1^\kappa, w)$  and  $f : \mathbb{F}^l \rightarrow \mathbb{F}$  be any linear map defined by:

$$f(x_1, \dots, x_l) = \sum_{i=1}^l \mu_i \cdot x_i + c.$$

Suppose  $m = f(m_1, \dots, m_l)$  for some messages  $m_1, \dots, m_l \in \mathcal{M}$ . Then the algorithm  $\text{Sim}$ , when given  $\mathbf{sk}, m, f$ , creates a signature on  $m, f$  as follows:

1. Compute, for  $r \in [n]$ ,

$$\sigma^*|_r = \sum_{i \in [l]} \mu_i \cdot \hat{\beta}_{i, r} + \hat{\alpha}_r \cdot (m - c).$$

2. Output  $\sigma^*$ .

Now let  $\sigma_i \leftarrow \text{Sign}(\mathbf{sk}, m_i)$ , for  $i \in [l]$ , and  $\sigma = \text{Eval}(f, \sigma_1, \dots, \sigma_l)$ . Note that  $m - c = \sum_i \mu_i \cdot m_i$ , and then we have, for all  $r \in [n]$ ,

$$\sigma^*|_r = \sum_{i \in [l]} \mu_i \cdot \left( \hat{\beta}_{i, r} + \hat{\alpha}_r \cdot m_i \right) = \sum_{i \in [l]} \mu_i \cdot \sigma_i|_r = \sigma|_r,$$

as required. □

## 4.2 Lower Bounds on IT Signatures

After the first impractical unconditionally secure signature scheme proposed by Chaum et al. [CR91], there have been many attempts to modify unconditionally secure *authentication codes* (A-codes) adding to them the extra security properties required by signature schemes. The relation between A-codes and unconditionally secure signatures have been extensively studied by Safavi-Naini et al. [SMY04]. A particular interesting extension of A-codes is given by *multi-receiver authentication codes* (MRA), introduced by Desmedt et al. [DFY92].

Roughly, a MRA-code consists of three sets of participants, a trusted authority (TA), a single honest sender ( $S$ ) and a set of receivers ( $R_1, \dots, R_n$ ). The TA generates secret information  $s$  and  $r_1, \dots, r_n$  for  $S$  and  $R_1, \dots, R_n$ , respectively. In order to send a message  $m$  to  $R_1, \dots, R_n$ ,  $S$  generates

an authenticated message  $(m, \tau)$  by using the secret  $s$ , and then transmits it to the receivers. Each receiver  $R_i$  can individually verify the authenticity of the received message using their private key  $r_i$ .

An  $(\epsilon, t, n)$ -MRA-code is a multi-receiver authentication code where  $n$  is the number of receivers, and for which the probability of success for up to  $t$  colluding corrupt receivers is  $\epsilon$ . The formal security definition for MRA-codes considers two types of attacks:

- A *substitution* attack, in which an adversary substitutes an authenticated message  $(m, \tau)$  with a different message  $(m^*, \tau^*)$  while  $m \neq m^*$ .
- An *impersonation* attack, in which an adversary creates a new message  $(m^*, \tau^*)$  without seeing any messages from the sender.

Information theoretic bounds on the success probability of attacks and combinatorial bounds on the sizes of the key and code spaces are given in [SW99, Sim85]. We can use these bounds to have a measure of the efficiency of our HITS construction. It is indeed possible to prove that HITS satisfies all the conditions for an MRA-code, but with stronger cryptographic properties like consistency, non-repudiation and 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:*

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

In terms of the memory size of signatures, our scheme is  $n$ -bits close to the bound of optimal MRA-codes. This follows from the fact that our definition of unforgeability implies security against substitution and impersonation attacks, and from Theorem 3.5 in [SW99] instantiated with  $P_D \leq \frac{2}{|\mathbb{F}|}$ .

Note that the scheme of [SS11], which is not homomorphic, requires the same memory size as HITS for signatures and signing keys, but 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  $\{P_1, \dots, P_n\}$ , and any HITS scheme  $\text{HITS} = (\text{Gen}, \text{Sign}, \text{Eval}, \text{Ver})$  that satisfies  $(w, \text{negl}(\kappa))$ -security and evaluation privacy from Section 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  $\mathcal{F}_{\text{MPC}}$ , shown in Fig. 2. Note that  $\mathcal{F}_{\text{MPC}}$  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  $[\mathcal{F}_{\text{MPC}}]_{\perp}^{\text{ID}}$  then extends this abort to be possible at any time.

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

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

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

**Add:** On input  $(\text{Add}, id_1, id_2, id_3)$  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  $(\text{Mult}, id_1, id_2, id_3)$  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  $(\text{Output}, id)$  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 **Deliver** or  $(\text{Abort}, P_i)$  for some  $i \in \mathcal{I}$ . If the adversary sends **Deliver** then output  $y$  to all honest parties, otherwise output  $(\text{Abort}, P_i)$ .

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

**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  $a$ :

$$\llbracket a \rrbracket = (a_i, \sigma_{a_i})_{i \in \mathcal{P}},$$

where party  $P_i$  holds  $a_i \in \mathbb{F}$  and  $\sigma_{a_i} = \text{Sign}(a_i, \text{sk})$ , 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,  $\llbracket z \rrbracket = \llbracket x \rrbracket + \llbracket y \rrbracket$ , as follows:

1. Compute  $z_i = x_i + y_i$ .
2. Compute  $\sigma_{z_i} = \text{Eval}(f, (\sigma_{x_i}, \sigma_{y_i}))$ , where  $f(a, b) = a + b$ .
3. Output  $\llbracket z \rrbracket = (z_i, \sigma_{z_i})_{i \in \mathcal{P}}$ .

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.<sup>5</sup>

**Open**( $\llbracket a \rrbracket$ ):

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

<sup>5</sup>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.

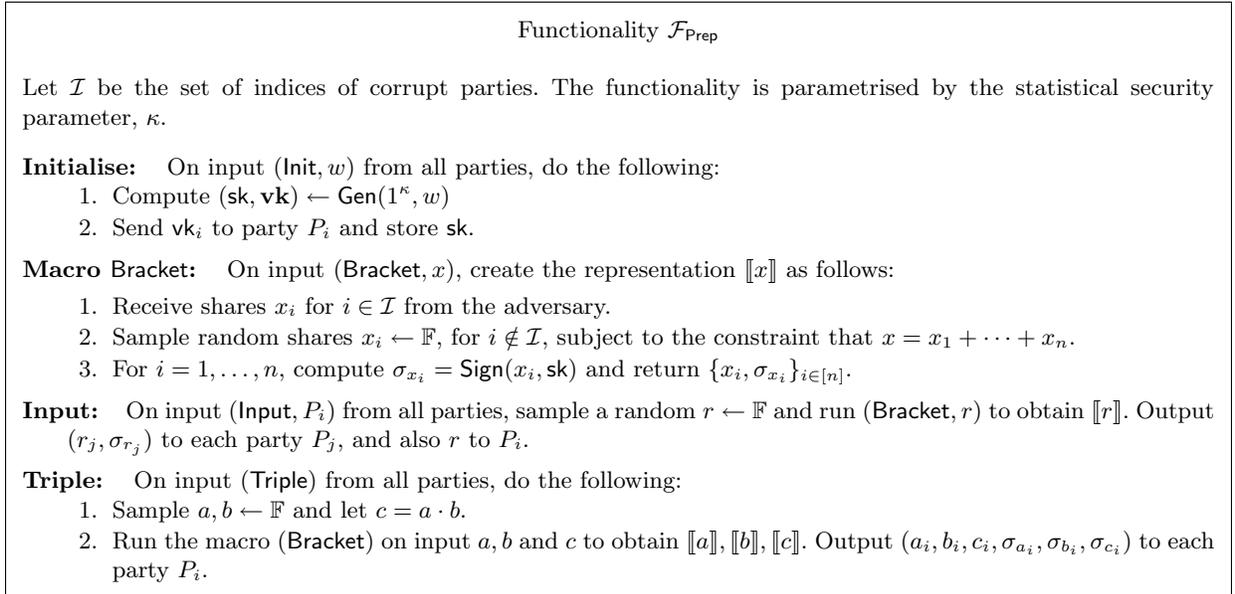
contrary, if there exists an index  $j \in \mathcal{P} \setminus \mathcal{I}$ , where  $\mathcal{I}$  denotes the set of corrupt parties, for which the check does not go through, then the same happens for all honest parties, due to the consistency of HITS.

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

- **Input tuples:** Random shared values  $\llbracket r \rrbracket$ , 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  $\llbracket a \rrbracket, \llbracket b \rrbracket, \llbracket c \rrbracket$ , 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  $\llbracket \cdot \rrbracket$ -representation for computing all linear gates, preprocessed multiplication triples for multiplication using Beaver’s technique [Bea91], and preprocessed input tuples for the inputs.



**Fig. 4.** Ideal functionality for the preprocessing phase.

## 5.1 Security

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

Protocol  $\Pi_{\text{Online}}$

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

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

**Input:** For party  $P_i$  to input  $x \in \mathbb{F}$ , the parties call  $\mathcal{F}_{\text{Prep}}$  with  $(\text{Input}, P_i)$  to obtain a mask value  $\llbracket r \rrbracket$ , and  $P_i$  also obtains  $r$ :

1.  $P_i$  sets  $m = r - x$  and broadcasts  $m$ .
2. All parties locally compute  $\llbracket x \rrbracket = \llbracket r \rrbracket - m$ .

**Add:** On input  $(\llbracket x \rrbracket, \llbracket y \rrbracket)$ , parties locally compute  $\llbracket x + y \rrbracket = \llbracket x \rrbracket + \llbracket y \rrbracket$ .

**Multiply:** On input  $(\llbracket x \rrbracket, \llbracket y \rrbracket)$ , the parties do the following:

1. Take one multiplication triple  $(\llbracket a \rrbracket, \llbracket b \rrbracket, \llbracket c \rrbracket)$  from  $\mathcal{F}_{\text{Prep}}$ , compute  $\llbracket \epsilon \rrbracket = \llbracket x \rrbracket - \llbracket a \rrbracket$ ,  $\llbracket \rho \rrbracket = \llbracket y \rrbracket - \llbracket b \rrbracket$ .
2. Call  $\text{Open}(\llbracket \epsilon \rrbracket)$  and  $\text{Open}(\llbracket \rho \rrbracket)$ .
3. Locally compute  $\llbracket z \rrbracket = \llbracket c \rrbracket + \epsilon \cdot \llbracket b \rrbracket + \rho \cdot \llbracket a \rrbracket + \epsilon \cdot \rho$

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

**Fig. 5.** Operations for Secure Function Evaluation with Identifiable Abort

*Proof.* Let  $\mathcal{A}$  be a malicious PPT real adversary attacking the protocol  $\Pi_{\text{Online}}$ , we construct an ideal adversary  $\mathcal{S}$  with access to  $\mathcal{F}_{\text{MPC}}$  which simulates a real execution of  $\Pi_{\text{Online}}$  with  $\mathcal{A}$  such that no environment  $\mathcal{Z}$  can distinguish the ideal process with  $\mathcal{S}$  and  $\mathcal{F}_{\text{MPC}}$  from a real execution of  $\Pi_{\text{Online}}$  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  $\mathcal{F}_{\text{Prep}}$  towards the adversary  $\mathcal{A}$ . Note that  $\mathcal{S}$  knows all the data given to the adversary and the simulated signing key  $\text{sk}^*$  of HITS, so can generate a valid signature for any message.

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

- For  $i \in \mathcal{P} \setminus \mathcal{I}$ ,  $\mathcal{S}$  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  $[\mathcal{F}_{\text{MPC}}]_{\perp}^{\text{ID}}$ .

*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  $[\mathcal{F}_{\text{MPC}}]_{\perp}^{\text{ID}}$ .

In a multiplication gate, for each call to  $\text{Open}$ ,  $\mathcal{S}$  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  $(t_j, \sigma_{t_j})$  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  $i \in \mathcal{P} \setminus \mathcal{I}$  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  $(\text{Abort}, P_j)$  to  $[\mathcal{F}_{\text{MPC}}]_{\perp}^{\text{ID}}$ . Otherwise it proceeds. <sup>6</sup>

<sup>6</sup>If there is more than invalid share then we always abort with the smallest index where the check fails.

*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  $\sigma_{y_{i_0}^*} = \text{Sim}(\text{sk}^*, y_{i_0}^*, f)$ , 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  $y_j, \sigma_{y_j}$ . If  $y_j = y_j^*$  and  $\sigma_{y_j} = \sigma_{y_j^*}$  for all  $j \in \mathcal{I}$  then  $\mathcal{S}$  sends **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  $\text{negl}(\kappa)$ .

During *initialisation*, the simulator honestly runs an internal copy of  $\mathcal{F}_{\text{Prep}}$ , 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  $\mathcal{F}_{\text{Prep}}$  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  $\mathcal{F}_{\text{Prep}}$ , 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  $\mathcal{F}_{\text{Prep}}$ , so are perfectly indistinguishable. To argue indistinguishability of the signatures, we need to use the evaluation privacy property. We must prove that

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

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

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

where  $g$  is the linear function evaluated to get the values  $t_i$  and  $t_i^*$ , and  $\text{sk}$  and  $\text{sk}^*$  are respectively the real-world and ideal-world secret keys. Since  $(t_i, \text{sk})$  and  $(t_i^*, \text{sk}^*)$  are identically distributed in both the executions, then the same holds for  $\sigma_{t_i}$  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  $\text{sk}^*$ , 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.  $\square$

## 5.2 An Optimised Protocol

When instantiated with our HITS scheme from Section 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  $\Pi_{bc}$  be the UC protocol for authenticated broadcast used in the protocol. We make the following assumption about its structure: in the first round of  $\Pi_{bc}$ , the sender (with input  $x$ ) sends  $x$ , and nothing more, to all parties.<sup>7</sup> Let the remainder of the protocol be denoted  $\Pi'_{bc}$ .

We now modify the protocol  $\Pi_{\text{Online}}$  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  $\Pi'_{bc}(B_i)$  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

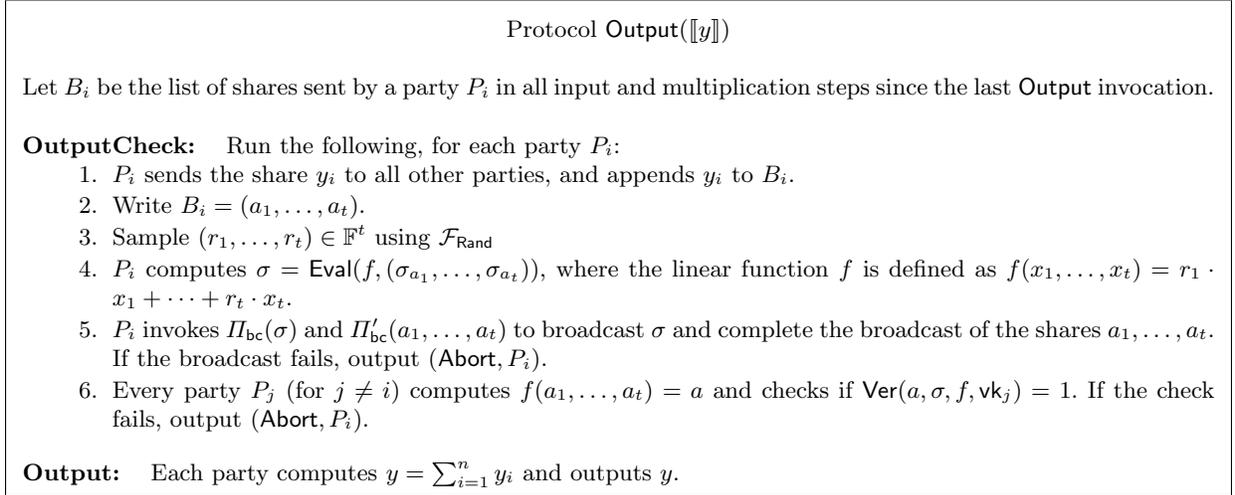
---

<sup>7</sup>Almost any broadcast protocol can be easily converted into this form. For example, the Dolev-Strong broadcast [DS83] begins with the sender sending  $(x, \text{Sign}(x))$  to all parties; we split this up into one round for  $x$  and one round for  $\text{Sign}(x)$ .

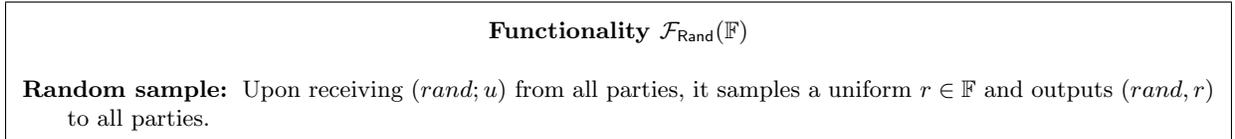
one output gate in the optimised protocol is  $O(D + n)$ . The total number of field elements sent over the network is no more than <sup>8</sup>

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

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



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



**Fig. 7.** Functionality  $\mathcal{F}_{\text{Rand}}$

### 5.3 Security

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  $[\mathcal{F}_{\text{MPC}}]_{\perp}^{\text{ID}}$  to obtain the result  $y$ , then adjusts one honest party's share and signature (using the evaluation privacy algorithm) to fix the

<sup>8</sup>Excluding the cost of  $\mathcal{F}_{\text{Rand}}$ , which can be implemented using standard techniques such as a hash-based commitment scheme in the random oracle model.

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  $\Pi_{bc}$  for each broadcast. If any broadcast fails for a corrupt sender  $P_j$  then  $\mathcal{S}$  sends `(Abort,  $P_j$ )` to  $[\mathcal{F}_{MPC}]_{\perp}^{\text{ID}}$ . 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  $\Pi_{bc}$  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  $B'_i = (a'_1, \dots, a'_t)$  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  $\mathcal{F}_{\text{Prep}}$  with identifiable abort, based on somewhat homomorphic encryption. The protocol is based on the SPDZ preprocessing [DPSZ12,DKL<sup>+</sup>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  $\mathcal{F}_{\text{Prep}}$  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 $\mathcal{F}_{\text{Prep}}^*$

The  $\mathcal{F}_{\text{Prep}}$  functionality from Section 5 is completely black-box with respect to the HITS scheme used. In this section, we implement preprocessing specifically for the scheme HITS from Section 4. We also make one small modification to the initialisation of  $\mathcal{F}_{\text{Prep}}$ , 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.

On input  $(\text{Init}, n_M, n_I)$  from all parties, set  $w = n \cdot (n_M + n_I)$  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  $(\text{sk}, \mathbf{vk}) \leftarrow \text{HITS.Gen}(1^\kappa, w)$ , except using the provided values  $\{\mathbf{v}_j\}_{j \in \mathcal{I}}$  to compute  $\{\mathbf{vk}_j\}_{j \in \mathcal{I}}$ , instead of sampling fresh values.
3. Send  $\mathbf{vk}_i$  to party  $P_i$  and store  $\text{sk}$ .

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

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

*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.  $\square$

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

## 6.2 SHE Scheme Requirements

As in SPDZ, we use a *threshold somewhat homomorphic encryption* scheme  $\text{SHE} = (\text{Gen}, \text{Enc}, \text{Dec}, \boxplus, \boxtimes)$  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  $\langle x \rangle = \text{Enc}(x)$ . The binary operators  $\boxplus, \boxtimes$  then guarantee that

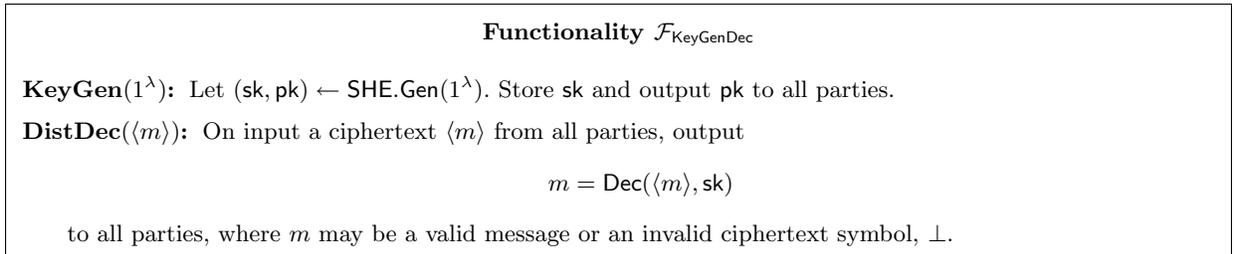
$$\langle x + y \rangle = \langle x \rangle \boxplus \langle y \rangle \quad \text{and} \quad \langle x \cdot y \rangle = \langle x \rangle \boxtimes \langle y \rangle,$$

for some suitable choice of randomness in the output ciphertexts. For our purposes, these homomorphic operations only need to support evaluation of circuits with polynomially many additions and multiplicative depth 1. As was shown in [DPSZ12,DKL<sup>+</sup>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  $\Pi_{\text{ZKPoK}}$ , 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  $\mathcal{F}_{\text{Rand}}$  (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  $\mathcal{F}_{\text{KeyGenDec}}$  (Fig. 9). To achieve security with identifiable abort in our preprocessing protocol, note that the distributed decryption method modelled in  $\mathcal{F}_{\text{KeyGenDec}}$  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  $\Pi_{\text{ZKPoK}}$  proof used for ciphertext generation. Efficient ZK proofs for actively secure LWE-based key generation and distributed decryption were also given in [AJL<sup>+</sup>12], which can be adapted to the ring-LWE setting.



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

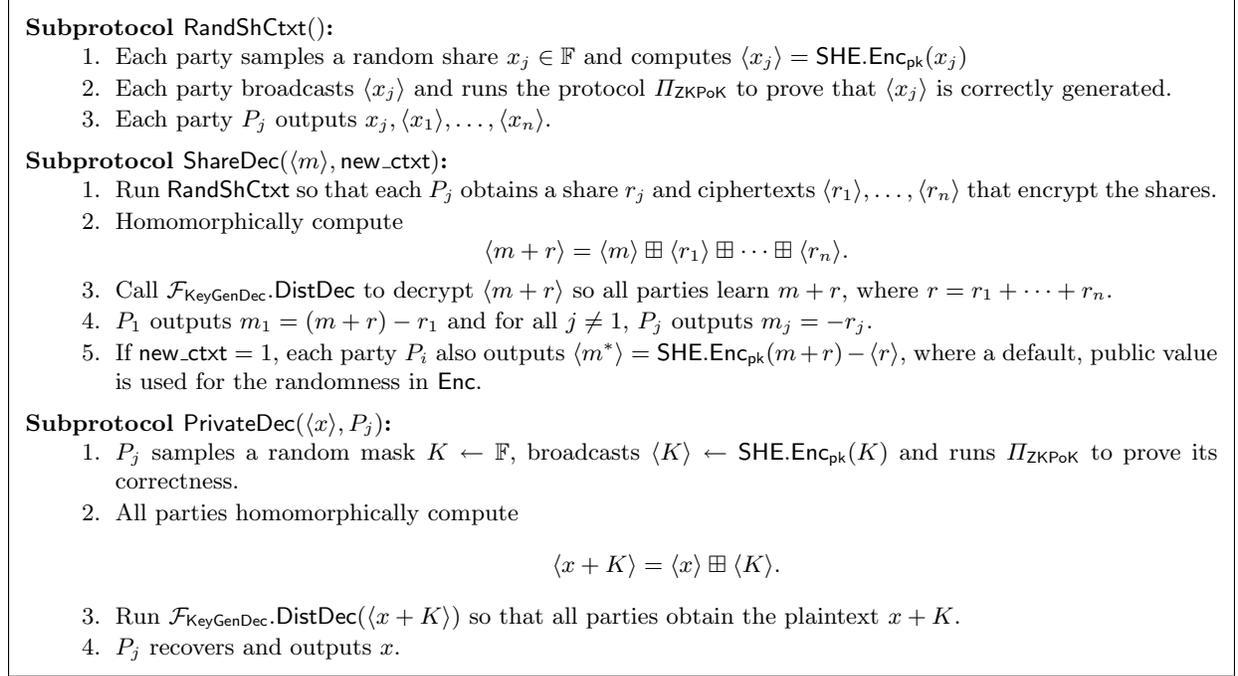
### 6.3 Basic Subprotocols

In Fig. 10 we describe some basic subprotocols for generating and decrypting ciphertexts. The **RandShCtxt** subprotocol creates  $n$  public ciphertexts encrypting uniformly random shares, where each party holds one share. The **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 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  $\langle x \rangle$  to only  $P_i$ . 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 Fig. 11 and Fig. 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  $\mathcal{F}_{\text{KeyGenDec}}$ , 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.



**Fig. 10.** Subprotocols for the preprocessing protocol using SHE

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

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

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 homomorphically compute

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

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

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

**Protocol  $\Pi_{\text{Prep}}$**

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

$$\begin{aligned} \langle \hat{\alpha}_r \rangle &= \langle \hat{\alpha}_r^1 \rangle \boxplus \dots \boxplus \langle \hat{\alpha}_r^n \rangle \\ \langle \hat{\beta}_{r,i} \rangle &= \langle \hat{\beta}_{r,i}^1 \rangle \boxplus \dots \boxplus \langle \hat{\beta}_{r,i}^n \rangle \quad \text{for } i \in [w] \end{aligned}$$

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

$$\begin{aligned} \langle \alpha_j \rangle &= \bigoplus_{r=1}^n (\langle \hat{\alpha}_r \rangle \boxtimes \langle v_{j,r} \rangle) \\ \langle \beta_{j,i} \rangle &= \bigoplus_{r=1}^n (\langle \hat{\beta}_{r,i} \rangle \boxtimes \langle v_{j,r} \rangle) \end{aligned}$$

6. Run the subprotocol  $\text{PrivateDec}(\langle \alpha_j \rangle, P_j)$  and  $\text{PrivateDec}(\langle \beta_{j,i} \rangle, P_j)$  for  $i \in [w]$  and  $j \in [n]$ , so that each party  $P_j$  gets their verification key  $\text{vk}_j$ .
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,  $\text{vk}_j = (\mathbf{v}_j, \alpha_j, \{\beta_{j,i}\}_{i \in [w]})$ .

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

### 6.5 Security of Theorem 3

As in [DPSZ12], to simplify the proof of security for our protocol we make the following assumption about the underlying SHE scheme. Note that this holds for the BGV scheme and its variants, under the (ring)-LWE assumption, and trivially implies the standard notion of IND-CPA security.

**Definition 5.** *We say the scheme SHE has the meaningless keys property if there exists an algorithm  $\text{Gen}^*$  that outputs a meaningless public key,  $\text{pk}^*$ , such that for  $\text{pk} \leftarrow \text{Gen}(1^\lambda)$ ,  $\text{pk}^* \leftarrow \text{Gen}^*(1^\lambda)$  and any message  $x$ :*

$$\begin{aligned} \text{Enc}(\text{pk}^*, x) &\stackrel{s}{\approx} \text{Enc}(\text{pk}^*, 0) \\ \text{pk} &\stackrel{c}{\approx} \text{pk}^*, \end{aligned}$$

where the distributions are taken over the randomness of  $\text{Gen}, \text{Gen}^*$ , and  $\stackrel{s}{\approx}, \stackrel{c}{\approx}$  are the usual definitions of statistical and computational indistinguishability.

*Proof.* (of Theorem 3) We give a sketch of the mechanics of a simulator  $\mathcal{S}$ , such that no environment  $\mathcal{Z}$  can distinguish between an interaction with the real protocol  $\Pi_{\text{Prep}}$ , and an interaction with  $\mathcal{S}$  and the ideal functionality  $[\mathcal{F}_{\text{Prep}}^*]_{\perp}^{\text{ID}}$ . There are two main components to the argument for indistinguishability. Firstly, all of the SHE ciphertexts must be indistinguishable in both worlds; this is similar to the proof of [DPSZ12, Thm. 3], and uses the meaningless keys property of SHE. The second, main component of the proof is to show that the signatures and verification keys received by the parties are indistinguishable in both worlds.

**Protocol  $\Pi_{\text{Prep}}$  (continued)**

**Subprotocol  $\text{Auth}(x_j, \langle x_j \rangle, P_j)$ :** 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  $\text{cnt} := 1$ .

1. All parties homomorphically compute, for  $r \in [n]$ ,

$$\langle \sigma|_r \rangle = \langle \hat{\alpha}_r \rangle \boxtimes \langle x_j \rangle \boxplus \langle \hat{\beta}_{r, \text{cnt}} \rangle$$

2. Run  $\text{PrivateDec}(\langle \sigma_r \rangle, P_j)$  for  $r \in [n]$  so that  $P_j$  obtains the signature on  $x_j$ ,

$$\sigma = (\hat{\alpha}_1 \cdot x_j + \hat{\beta}_{1, \text{cnt}}, \dots, \hat{\alpha}_n \cdot x_j + \hat{\beta}_{n, \text{cnt}})$$

3. Set  $\text{cnt} := \text{cnt} + 1$

**Triple:** Creates a single, authenticated multiplication triple.

1. The parties run  $\text{RandShCtxt}$  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

$$\langle c \rangle = (\langle a_1 \rangle \boxplus \dots \boxplus \langle a_n \rangle) \boxtimes (\langle b_1 \rangle \boxplus \dots \boxplus \langle b_n \rangle)$$

3. The parties run  $\text{ShareDec}(\langle c \rangle, 1)$  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  $\text{Auth}(x_j, \langle x_j \rangle, P_j)$  so  $P_j$  obtains  $\sigma_{x_j} = \text{Sign}(x_j, \text{sk})$ .
5. Output the authenticated triple  $(\llbracket a \rrbracket, \llbracket b \rrbracket, \llbracket c \rrbracket)$ .

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

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

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

The simulator  $\mathcal{S}$  first generates an SHE key pair  $(\text{pk}, \text{SHE.sk}) \leftarrow \text{SHE.Gen}(1^\lambda)$  and sends  $\text{pk}$  to all parties. Simulate the rest of the setup phase by honestly following the protocol, until it comes to the final stage of simulating calls to  $\mathcal{F}_{\text{KeyGenDec}}.\text{PrivateDec}$ , to decrypt the verification keys. At this point, use the SHE key  $\text{sk}$  to decrypt the adversary's ciphertexts containing the verification data  $\mathbf{v}_j$  for  $j \in \mathcal{I}$ , and send these to  $[\mathcal{F}_{\text{Prep}}^*]_{\perp}^{\text{ID}}$ . Receive from  $[\mathcal{F}_{\text{Prep}}^*]_{\perp}^{\text{ID}}$  the remaining verification data  $\alpha_j, \{\beta_{j,i}\}_{i \in [w]}$  and send this to the adversary, simulating the output of  $\mathcal{F}_{\text{KeyGenDec}}.\text{PrivateDec}$  in step 6.

Simulating the triple and input generation protocols (and their subroutines) proceeds similarly: the simulator sends ciphertexts on behalf of honest parties as in the protocol, and decrypts the adversary's ciphertexts containing shares of triples  $a_i, b_i$  and sends these to  $\mathcal{F}_{\text{Prep}}^*$ , along with the shares of  $c_i$  that are extracted from ciphertexts received in the  $\text{ShareDec}$  procedure. In response, the simulator receives from  $\mathcal{F}_{\text{Prep}}^*$  signatures on all corrupted parties' shares, and uses these to simulate the final calls to  $\mathcal{F}_{\text{KeyGenDec}}.\Pi_{\text{PubDec}}$  during the  $\text{Auth}$  subroutine.

To ensure security with identifiable abort, we require that if at any stage during the simulation, a proof of correct ciphertext generation ( $\Pi_{\text{ZKPoK}}$ ) fails for  $P_j$  where  $j \in \mathcal{I}$ , then  $\mathcal{S}$  sends  $(\text{Abort}, P_j)$  to  $[\mathcal{F}_{\text{Prep}}^*]_{\perp}^{\text{ID}}$  and halts. Since  $\Pi_{\text{ZKPoK}}$  is the only place in which the protocol can abort, we do not need to consider identifiable abort elsewhere.

*Indistinguishability.* The argument for indistinguishability of the triple shares, SHE keys and ciphertexts in the two worlds is similar to that in [DPSZ12], which shows how to use an algorithm

that distinguishes between the two worlds with non-negligible probability to break the meaningless keys property of the SHE scheme. Since our usage of SHE with zero-knowledge proofs is almost identical to that work, we do not describe the reduction here. Note that our scenario is slightly simpler than [DPSZ12], as our distributed decryption method is guaranteed to be correct, so we do not have to handle any errors there.

We must also show that the verification keys and signatures generated are indistinguishable in both worlds. In the setup phase, the signing key values  $\hat{\alpha}_j, \hat{\beta}_{j,i}$  are never seen by the environment; also, since each of these values is calculated as the sum of random shares contributed by all parties, they will be uniformly distributed, even if the adversarial shares are not. This means the underlying signing key is identically distributed in both worlds, and hence so are the honest parties' verification keys. The corrupted parties' verification keys are also identically distributed, because the adversary's contributions,  $\mathbf{v}_j$ , are sent to the functionality in the ideal world so that the keys are computed in exactly the same way.

Finally, we must consider the distributions of the signatures. It is easy to see by inspection of the protocol that in the real world, all signatures are correctly generated according to the `Sign` algorithm, using the underlying signing key that is unknown to all parties. On the other hand, in the ideal world, the signatures are generated by  $[\mathcal{F}_{\text{Prep}}]_{\perp}^{\text{ID}}$  using `Sign` and the signing key of  $[\mathcal{F}_{\text{Prep}}]_{\perp}^{\text{ID}}$ . Since we have already shown that the signature keys are identically distributed in both worlds, it follows that the distributions of signatures are the same also, and so all signatures will correctly verify in both worlds.  $\square$

## 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.

**Cost of Broadcast.** For MPC with identifiable abort, we denote the cost of broadcasting  $m$  field elements by  $\text{bc}(m)$ . 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<sup>+</sup>12]. This means each broadcast costs  $O(n)$  messages, with a one-time  $O(n^2)$  cost to verify these at the end.

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<sup>+</sup>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  $\alpha_i$ . 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  $\beta_{a_1}^i, \dots, \beta_{a_n}^i$  and  $n$  MAC values  $m_1(a_i), \dots, m_n(a_i)$ . 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 ZK 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 ZK proofs.

Each ZK 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  $m = O(\sigma)$  for statistical security  $\sigma$ , whereas [GMO16] use  $m = 3$ , but require repeating the proof  $\sigma$  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  $\sigma$  signatures (or  $\sigma \cdot n$  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  $\Pi_{\text{SCP}}$  protocol of [IOZ14]. According to [GMO16, Sec. 4.2], the proof size is at least  $2 \cdot \sigma \cdot \log_2(|\mathbb{F}|)$ , for a proof with soundness  $2^{-\sigma}$ , 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  $t = O(\sigma)$  field elements in the  $\Pi_{\text{SCP}}$  protocol, so would not have significantly better complexity.

**Comparison of the Online Phases.** The complexities in Table 1 for our protocol can be derived from the analysis in Section 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.

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)	$\sigma n$	$\sigma n$	$\sigma \cdot \text{bc}(1)$	$\sigma n \cdot \text{bc}(1) + 2n \cdot \text{bc}(n)$	$\sigma n \cdot \text{bc}(1) + 2n \cdot \text{bc}(n)$	$O(D \cdot n)$
<b>Ours</b>	$2n + 1$	$6n + 3$	$\text{bc}(1)$	$2n(n - 1)$	$n \cdot \text{bc}(n + 2N_M + 1)$	$O(D + n)$

**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  $\sigma$  is a statistical security parameter.

## 7.1 Preprocessing Cost

For preprocessing, the main factor affecting computation and communication costs in [DPSZ12,DKL<sup>+</sup>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  $n$  ZK proofs, and since there are  $n$  signatures per shared value (one per share) this gives a total of  $O(n^3)$  ZK 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 is 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  $\mathcal{F}_{\text{Prep}}$  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].

## 8 From MPC with Identifiable Abort to MPC with Public Verifiability.

In a practical setting, it can be necessary to raise the deterrent to malicious behavior by e.g. allowing so-called public verifiability (see e.g. [AO12]). That means that an honest party can convince an external third party (e.g. a judge) that another party cheated in the computation (that is called *accountability*), whereas a dishonest party cannot blame an honest party (a property named *defamation-freeness*). In addition, we require that the third party does not learn anything about the inputs of the honest parties (in case that it is controlled by  $\mathcal{A}$ ), which we call *privacy-preserving*.

In more formal terms, let  $\Pi(\mathcal{P}, \mathcal{A}, \mathcal{I}, 1^\kappa, (x_k)_{k \notin \mathcal{I}})$  denote the random variable describing a transcript for a protocol with  $n$  parties  $\mathcal{P} = \{P_1, \dots, P_n\}$ , out of which the ones in  $\mathcal{I}$  are controlled by  $\mathcal{A}$  and where the honest parties have inputs  $x_k$  as mentioned above. We assume that there exists a PPT algorithm  $\text{Jud}$  that, on input  $(n, 1^\kappa, \pi, j)$ , outputs a bit  $b$  and has the following properties:

**Accountability:** If  $i \in \mathcal{P} \setminus \mathcal{I}, j \in \mathcal{I}$  then

$$\Pr \left[ \text{Jud}(n, 1^\kappa, \pi, j) = 1 \mid \pi \leftarrow \Pi(\mathcal{P}, \mathcal{A}, \mathcal{I}, 1^\kappa, (x_k)_{k \notin \mathcal{I}}) \wedge \right. \\ \left. P_i \text{ outputs (Abort, } P_j) \right] = 1 - \text{negl}(\kappa)$$

**Defamation-Free:** If  $j \notin \mathcal{I}$  then

$$\Pr \left[ \text{Jud}(n, 1^\kappa, \pi, j) = 0 \mid \pi \leftarrow \Pi(\mathcal{P}, \mathcal{A}, \mathcal{I}, 1^\kappa, (x_k)_{k \notin \mathcal{I}}) \right] = 1 - \text{negl}(\kappa)$$

Moreover, no corrupt  $\text{Jud}'$  should be able to learn anything about the parties' inputs:

**Privacy-Preserving:** For all  $k \notin \mathcal{I}$  let  $\hat{x}_k \leftarrow \mathcal{M}$  be drawn uniformly at random from the message space. Then, for any PPT algorithm  $\text{Jud}'$  and  $i \notin \mathcal{I}$  it holds that

$$\Pr[\text{Jud}'(n, 1^\kappa, \pi, \cdot) = x_i \mid \pi \leftarrow \Pi(\mathcal{P}, \mathcal{A}, \mathcal{I}, 1^\kappa, (x_k)_{k \notin \mathcal{I}})] = 1 \\ \approx \\ \Pr[\text{Jud}'(n, 1^\kappa, \pi, \cdot) = x_i \mid \pi \leftarrow \Pi(\mathcal{P}, \mathcal{A}, \mathcal{I}, 1^\kappa, (\hat{x}_k)_{k \notin \mathcal{I}})] = 1$$

This third property is necessary to exclude the *trivial* solution where a party  $P_i$  allows  $\text{Jud}$  to check the transcript by opening a commitment to its key  $\text{vk}_i$ , as this would reveal the input  $P_i$  to  $\text{Jud}$ .

This notion of public verifiability extends the idea of [BDO14] where the external party (which the authors call the *Auditor*) can inspect the transcript for correctness after the computation is done. In their model and protocol, one is not able to identify a dishonest party, whereas we will now sketch how to modify our protocol such that this indeed can be done.

In a first step, replace the broadcast channel with a bulletin board<sup>9</sup>  $\mathcal{B}$ . That means that there is a resource available to which each party can send messages, such that these sent messages cannot be altered after they have been published. Moreover,  $\mathcal{B}$  must be publicly readable in particular by  $\text{Jud}$ , and each message must be accompanied by a signature of the sending party. This transcript of all messages does not yield privacy problems, since  $\text{Jud}$  only obtains information that  $\mathcal{A}$  learns as well.

Now, the  $\text{Ver}$  algorithm will be run such that it generates verification keys for  $2n$  parties. As before, we privately give  $\text{vk}_i$  to  $P_i$  but additionally post commitments to  $\text{vk}_{n+1}, \dots, \text{vk}_{2n}$  on  $\mathcal{B}$ . The

<sup>9</sup>Such a bulletin board can be constructed e.g. using the blockchain of Bitcoin.

opening information to the commitment of  $\mathbf{vk}_{n+i}$  is given to  $P_i$ . Moreover, we demand that each commitment is signed by all parties (this can be done during the offline phase). We want to point out that only having one such additional key is not sufficient, because this key would then be known to  $\mathcal{A}$  and the consistency of the HITS only applies when such an additional key is unknown.

If a party  $P_i$  detects cheating of some  $P_j$ ,  $P_i$  can simply send the opening information for  $\mathbf{vk}_{n+i}$  to Jud, which by the consistency of the HITS will end up with the same conclusion as  $P_i$ . Moreover, the same property ensures that a dishonest  $P_j$  cannot defame an honest  $P_i$ . As for privacy, this is ensured because the only information that  $\mathcal{A}$  controlling Jud can obtain is the same as for a HITS instance with  $2n$  parties, where  $2n - 1$  can be corrupted.

We leave a more detailed description of such a protocol as future work.

## 9 Efficient MPC with Identifiable Abort over Small Fields

In Section 5 we introduced an MPC protocol whose security relies on the size of the field  $\mathbb{F}$ . In particular, Corollary 1 requires that  $\log |\mathbb{F}| - 2 \geq \kappa$  if all properties of the HITS scheme should hold with overwhelming probability. A standard approach over smaller fields is to extend the  $\llbracket \cdot \rrbracket$ -representation to contain multiple signatures with different keys. This construction unfortunately does not work well in conjunction with small fields: If  $|\mathbb{F}| = O(1)$  then one necessarily needs  $\Omega(\kappa)$  signatures on each share to achieve error probability  $< 2^{-\kappa}$ .

We will now show how to adapt a technique due to Damgård and Zakarias [DZ13] that uses a dedicated preprocessing to work with HITS. Their approach only works for circuits that are not too thin (i.e. they need a certain number of gates that can be evaluated in parallel) but then permits (in the best case) constant overhead for MPC without identifiable abort. They assume that the circuit comes in layers, where each layer only consists of addition or multiplication gates only. We now revisit and extend their construction and obtain ID-MPC over small fields with  $O(1)$  signatures per share.

### 9.1 Packing Shares into Codewords

We assume that the field  $\mathbb{F}$  permits the construction of an  $[m, k, d]$  linear systematic block code (where  $m, k, d = O(\kappa)$ ). That means that we can represent each vector  $\mathbf{x} \in \mathbb{F}^k$  using the efficiently computable monomorphism  $C : \mathbb{F}^k \rightarrow \mathbb{F}^m$  such that  $\forall \mathbf{x}, \mathbf{y} \in \mathbb{F}^k : \mathbf{x} \neq \mathbf{y} \Rightarrow \#(C(\mathbf{x}) - C(\mathbf{y})) \geq d$  where  $\#(\cdot)$  is the Hamming weight of a vector. This map is systematic if  $\mathbf{x}$  appears in  $C(\mathbf{x})$  in the first  $k$  coordinates. We moreover assume that there exists a  $\text{poly}(\kappa)$ -time procedure to decide for a given  $\mathbf{c} \in \mathbb{F}^m$  if  $\mathbf{c}$  lies in the image of  $C$ . If so, then we will denote this as  $\mathbf{c} \in C$  or say that  $\mathbf{c}$  is a *codeword*. Note that we will denote with  $C$  both the map and the image of it.

An interesting notion to study is the so-called *Schur transform* of a code, for which we consider  $C^* = \text{span}\{C(\mathbf{x}) \odot C(\mathbf{y}) \mid \mathbf{x}, \mathbf{y} \in \mathbb{F}^k\}$  where  $\odot$  is the coordinate-wise product of two vectors.  $C^*$  is itself a linear  $[m, k', d']$  code where  $k' \geq k, d' \leq d$ . We make the assumption that  $d, d' \in O(\kappa)$ . Codes that allow for such parameters are e.g. the Reed-Solomon code or Algebraic Geometry codes.

Similar to our previous definition of the  $\llbracket \cdot \rrbracket$ -representation we will now introduce the  $\llbracket \cdot \rrbracket_m$ -representation as follows: Assume that there are  $m$  sets of keys  $\{(\mathbf{sk}_t, (\mathbf{vk}_{1,t}, \dots, \mathbf{vk}_{n,t}))\}_{t \in [m]}$  for  $m$  independent instances of HITS. We will represent a *vector*  $\mathbf{x} \in \mathbb{F}^k$  by letting  $\mathbf{x}_1, \dots, \mathbf{x}_n$  be chosen uniformly at random subject to the constraint that  $\mathbf{x} = \sum_i \mathbf{x}_i$ . Each party  $P_i$  will secretly hold  $\mathbf{x}_i$ . Then

$$\llbracket \mathbf{x} \rrbracket_m := \left( C(\mathbf{x}_1), \dots, C(\mathbf{x}_n), (\sigma_{\mathbf{sk}_t}(C(\mathbf{x}_1)|_t), \dots, \sigma_{\mathbf{sk}_t}(C(\mathbf{x}_n)|_t))_{t \in [m]} \right).$$

In this representation one puts a signature  $\sigma_{\text{sk}_t}(C(\mathbf{x}_i))|_t$  on each coordinate of  $C(\mathbf{x}_i)$  that  $P_i$  holds, where (as in Section 5)  $P_i$  also holds this signature.

It is immediate that addition of two such representations  $\llbracket \mathbf{x} \rrbracket_m, \llbracket \mathbf{y} \rrbracket_m$  can be done by a coordinate-wise addition. We obtain the following algorithm, where each party  $P_i$  does the following:

1. Set  $\mathbf{z}_i = \mathbf{x}_i + \mathbf{y}_i$
2. For  $t \in [m]$  compute  $\sigma_{\text{sk}_i}(C(\mathbf{z}_i)|_t) \leftarrow \text{Eval}(f, (\sigma_{\text{sk}_i}(C(\mathbf{x}_i))|_t), \sigma_{\text{sk}_i}(C(\mathbf{y}_i))|_t))$  where  $f(a, b) = a + b$
3. Output  $\llbracket \mathbf{z} \rrbracket_m := \left( C(\mathbf{z}_1), \dots, C(\mathbf{z}_n), (\sigma_{\text{sk}_t}(C(\mathbf{z}_1))|_t), \dots, \sigma_{\text{sk}_t}(C(\mathbf{z}_n))|_t)_{t \in [m]} \right)$

To add a constant vector  $\mathbf{c} \in \mathbb{F}^k$  to  $\llbracket \mathbf{x} \rrbracket_m$  we let a fixed party (let's assume  $P_1$ ) set  $C(\mathbf{x}_1) \leftarrow C(\mathbf{c}) + C(\mathbf{x}_1)$ . Afterwards, all parties use  $\text{Eval}$  to add the constant  $C(\mathbf{c})|_t$  by adjusting  $\sigma_{\text{sk}_1}(C(\mathbf{x}_1))|_t$  and the function they evaluate during  $\text{Ver}$ .

Given a random representation  $\llbracket \mathbf{r} \rrbracket_m$ , where  $\mathbf{r}$  is known to  $P_i$ , this already allows a party  $P_i$ , to privately input a vector  $\mathbf{x}$  into the computation as in our original ID-MPC scheme (this  $\llbracket \mathbf{r} \rrbracket_m$  will come from the preprocessing, as before).

We can moreover repurpose the  $\text{Open}$  algorithm to work on  $\llbracket \mathbf{x} \rrbracket_m$ :

1. Every party  $P_i$  broadcasts  $(C(\mathbf{x}_i), (\sigma_{\text{sk}_t}(C(\mathbf{x}_i))|_t)_{t \in [m]})$ .
2. Each party  $P_i$  runs  $\text{Ver}(C(\mathbf{x}_j)|_t, \sigma_{\text{sk}_t}(C(\mathbf{x}_j))|_t, f, \mathbf{vk}_{i,t})$ , for each  $j \neq i, t \in [m]$  and checks that  $C(\mathbf{x}_i)$  is a codeword of  $C$ . If for some  $j, t$  the check fails or the vector is not a codeword,  $P_i$  outputs  $(\text{Abort}, P_j)$ .
3. Output  $\mathbf{x} = \sum_{i \in \mathcal{P}} \mathbf{x}_i$  where  $\mathbf{x}_i$  is obtained from decoding  $C(\mathbf{x}_i)$ .

The opening is now successful iff  $P_j$  has verified all signatures and if  $C(\mathbf{x}_i)$  is indeed a codeword of  $C$ . As in [DZ13] we obtain correctness since  $\mathcal{A}$  must have successfully forged signatures for at least  $d$  positions (the minimal distance of  $C$ ). Due to our choice, the probability of successfully forging  $d$  signatures is  $|\mathbb{F}|^{-d} = |\mathbb{F}|^{-O(\kappa)}$ . As before, the properties of the HITS allow us to identify a cheater in case of abort.

## 9.2 Evaluating the Circuit

Following the approach of Section 5 we want to be able to multiply constant vectors  $\mathbf{c}$  into a  $\llbracket \cdot \rrbracket_m$ -representation, which then permits multiplication of these representations using the Beaver circuit randomization technique. This can be done by coordinate-wise multiplication of the vectors of  $\llbracket \mathbf{x} \rrbracket_m$  with  $C(\mathbf{c})$  plus carefully adjusting the signatures. But the resulting representation does not have codewords from  $C$ , but from  $C^*$  instead. Let us write  $\llbracket \mathbf{x} \rrbracket_m^*$  in the case where we have a  $\llbracket \cdot \rrbracket_m$ -representation where the codewords are in the Schur transform. In order to continue the computation we have to turn each such  $\llbracket \cdot \rrbracket_m^*$  back into a  $\llbracket \cdot \rrbracket_m$ -representation containing the exact same value. To achieve this we assume that the preprocessing outputs pairs  $(\llbracket \mathbf{r} \rrbracket_m^*, \llbracket \mathbf{r} \rrbracket_m)$ , where  $\mathbf{r}$  was chosen uniformly at random. This gives rise to an interactive multiplication protocol for constants that is similar to the multiplication protocol of  $\Pi_{\text{Online}}$ :

1. Each  $P_i$  sets  $\mathbf{z}_i \leftarrow C(\mathbf{x}_i) \odot C(\mathbf{c})$
2. For  $t \in [m]$  each  $P_i$  computes  $\sigma_{\text{sk}_i}(\mathbf{z}_i|_t) \leftarrow \text{Eval}(f_{C(\mathbf{c})|_t}, (\sigma_{\text{sk}_i}(C(\mathbf{x}_i))|_t))$  where  $f_b(a) = b \cdot a$
3. Set  $\llbracket \mathbf{z} \rrbracket_m^* \leftarrow \left( \mathbf{z}_1, \dots, \mathbf{z}_n, (\sigma_{\text{sk}_t}(\mathbf{z}_1)|_t), \dots, \sigma_{\text{sk}_t}(\mathbf{z}_n)|_t)_{t \in [m]} \right)$
4. Open  $\llbracket \mathbf{z} \rrbracket_m^* - \llbracket \mathbf{r} \rrbracket_m^*$  using the above algorithm and reconstruct  $\mathbf{x} - \mathbf{r}$ . If  $\text{Open}$  fails then abort.

5. Compute  $\llbracket c \odot \mathbf{x} \rrbracket_m \leftarrow C(\mathbf{x} - \mathbf{r}) + \llbracket \mathbf{r} \rrbracket_m$ .

As outlined in  $\Pi_{\text{Online}}$  this protocol can now be used to multiply  $\llbracket \cdot \rrbracket_m$ -representations using multiplication triples.

An additional complication (in comparison to  $\Pi_{\text{Online}}$ ) is that we may have to reorder values inside  $\llbracket \mathbf{x} \rrbracket_m$  or between multiple representations (or duplicate them), due to the wiring between layers of the circuit. For simplicity, we assume that all values of a layer fit into one representation (but the generalization is straightforward): Observe that each such *rewiring* corresponds to applying a linear transformation  $f$  to the vector of the representation. We demand from the preprocessing to obtain a pair  $(\llbracket \mathbf{r} \rrbracket_m, \llbracket f(\mathbf{r}) \rrbracket_m)$  for each such layer. To compute  $\llbracket f(\mathbf{x}) \rrbracket_m$  we then use the following algorithm:

1. Compute and open  $\llbracket \mathbf{x} - \mathbf{r} \rrbracket_m$  using the above open algorithm and reconstruct  $\mathbf{x} - \mathbf{r}$ . If **Open** fails then abort.
2. Each party locally computes  $\mathbf{y} \leftarrow f(\mathbf{x} - \mathbf{r})$ .
3. Set  $\llbracket f(\mathbf{x}) \rrbracket_m \leftarrow \mathbf{y} + \llbracket f(\mathbf{r}) \rrbracket_m$ .

In order to generate these pairs  $(\llbracket \mathbf{r} \rrbracket_m, \llbracket f(\mathbf{r}) \rrbracket_m)$  the preprocessing must be aware of the actual circuit that is evaluated in the online phase. This dependence can be resolved by using programmable sorting networks (see [DZ13] for details) instead of the fixed maps from the actual circuit.

### 9.3 Identifiable Abort and Efficiency

A careful inspection shows that our above protocol follows the same outline as  $\Pi_{\text{Online}}$  and that the subprotocols use similar techniques, with the difference being the representation switching and the inter-layer permutation. Both of these are variants of the **Mult** protocol of  $\Pi_{\text{Online}}$ . Hence, security again reduces to being able to securely perform **Open**. As we argued above, with a sufficient parameter choice we can ensure that opening  $\llbracket \cdot \rrbracket_m, \llbracket \cdot \rrbracket_m^*$ -representations can be done such that cheating is detected, based on  $d, d'$  being large enough.

Concerning the efficiency of the sketched scheme, we observe that it only performs  $O(n)$  worse than the original construction by Damgård and Zakarias. Namely, the storage per  $\llbracket \cdot \rrbracket_m$ -representation is  $< 3nm$  field elements (including the  $m$  verification keys), whereas [DZ13] stores  $3m$  elements from  $\mathbb{F}$ . For each unreliable open of [DZ13] one uses broadcast  $n$  times on  $m$  field elements each, whereas we use  $n$  times broadcast on  $m + m \cdot n$  values due to the signatures that we have to open.

The analysis of the network communication of [DZ13] can equally be applied to our setting by plugging in the different overhead from our construction. Using Reed-Solomon codes we achieve a network overhead of  $O(n \cdot \text{polylog}(\kappa))$ , whereas [DZ13] has  $O(\text{polylog}(\kappa))$ .

## References

- AJL<sup>+</sup>12. Gilad Asharov, Abhishek Jain, Adriana López-Alt, Eran Tromer, Vinod Vaikuntanathan, and Daniel Wichs. Multiparty computation with low communication, computation and interaction via threshold FHE. In *Advances in Cryptology - EUROCRYPT 2012 - 31st Annual International Conference on the Theory and Applications of Cryptographic Techniques, Cambridge, UK, April 15-19, 2012. Proceedings*, pages 483–501, 2012.
- AL10. Yonatan Aumann and Yehuda Lindell. Security against covert adversaries: Efficient protocols for realistic adversaries. *J. Cryptology*, 23(2):281–343, 2010.

- AO12. Gilad Asharov and Claudio Orlandi. Calling out cheaters: Covert security with public verifiability. In *Advances in Cryptology—ASIACRYPT 2012*, pages 681–698. Springer, 2012.
- BDO14. Carsten Baum, Ivan Damgård, and Claudio Orlandi. Publicly auditable secure multi-party computation. In *Security and Cryptography for Networks*, pages 175–196. Springer, 2014.
- BDOZ11. Rikke Bendlin, Ivan Damgård, Claudio Orlandi, and Sarah Zakarias. Semi-homomorphic encryption and multiparty computation. In *Advances in Cryptology - EUROCRYPT 2011 - 30th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Tallinn, Estonia, May 15-19, 2011. Proceedings*, pages 169–188, 2011.
- BDTZ16. Carsten Baum, Ivan Damgård, Tomas Toft, and Rasmus Zakarias. Better preprocessing for secure multi-party computation. In *ACNS 2016*. Springer, 2016.
- Bea91. Donald Beaver. Efficient multiparty protocols using circuit randomization. In *Advances in Cryptology—CRYPTO’91*, pages 420–432. Springer, 1991.
- BFKW09. Dan Boneh, David Mandell Freeman, Jonathan Katz, and Brent Waters. Signing a linear subspace: Signature schemes for network coding. In *Public Key Cryptography - PKC 2009, 12th International Conference on Practice and Theory in Public Key Cryptography, Irvine, CA, USA, March 18-20, 2009. Proceedings*, pages 68–87, 2009.
- BGV12. Zvika Brakerski, Craig Gentry, and Vinod Vaikuntanathan. (leveled) fully homomorphic encryption without bootstrapping. In *ITCS*, pages 309–325, 2012.
- BLOO11. Amos Beimel, Yehuda Lindell, Eran Omri, and Ilan Orlov.  $1/p$ -secure multiparty computation without honest majority and the best of both worlds. In *Advances in Cryptology - CRYPTO 2011 - 31st Annual Cryptology Conference, Santa Barbara, CA, USA, August 14-18, 2011. Proceedings*, pages 277–296, 2011.
- Can01. Ran Canetti. Universally composable security: A new paradigm for cryptographic protocols. In *42nd Annual Symposium on Foundations of Computer Science, FOCS 2001, 14-17 October 2001, Las Vegas, Nevada, USA*, pages 136–145, 2001.
- CDD<sup>+</sup>99. Ronald Cramer, Ivan Damgård, Stefan Dziembowski, Martin Hirt, and Tal Rabin. Efficient multiparty computations secure against an adaptive adversary. In *Advances in Cryptology—EUROCRYPT’99*, pages 311–326. Springer, 1999.
- CJL09. Denis Xavier Charles, Kamal Jain, and Kristin E. Lauter. Signatures for network coding. *IJICoT*, 1(1):3–14, 2009.
- CL14. Ran Cohen and Yehuda Lindell. Fairness versus guaranteed output delivery in secure multiparty computation. In *Advances in Cryptology - ASIACRYPT 2014 - 20th International Conference on the Theory and Application of Cryptology and Information Security, Kaoshiung, Taiwan, R.O.C., December 7-11, 2014, Proceedings, Part II*, pages 466–485, 2014.
- Cle86. Richard Cleve. Limits on the security of coin flips when half the processors are faulty (extended abstract). In *Proceedings of the 18th Annual ACM Symposium on Theory of Computing, May 28-30, 1986, Berkeley, California, USA*, pages 364–369, 1986.
- CR91. David Chaum and Sandra Roijackers. Unconditionally-secure digital signatures. In *Advances in Cryptology-CRYPTO’90*, pages 206–214. Springer, 1991.
- DFY92. Yvo Desmedt, Yair Frankel, and Moti Yung. Multi-receiver/multi-sender network security: Efficient authenticated multicast/feedback. In *Proceedings IEEE INFOCOM ’92, The Conference on Computer Communications, Eleventh Annual Joint Conference of the IEEE Computer and Communications Societies, One World through Communications, Florence, Italy, May 4-8, 1992*, pages 2045–2054, 1992.
- DKL<sup>+</sup>12. Ivan Damgård, Marcel Keller, Enrique Larraia, Christian Miles, and Nigel P. Smart. Implementing AES via an actively/covertly secure dishonest-majority MPC protocol. In *SCN*, pages 241–263, 2012.
- DKL<sup>+</sup>13. Ivan Damgård, Marcel Keller, Enrique Larraia, Valerio Pastro, Peter Scholl, and Nigel P. Smart. Practical covertly secure MPC for dishonest majority — or: Breaking the SPDZ limits. In *ESORICS*, pages 1–18, 2013.
- DPSZ12. Ivan Damgård, Valerio Pastro, Nigel P. Smart, and Sarah Zakarias. Multiparty computation from somewhat homomorphic encryption. In Reihaneh Safavi-Naini and Ran Canetti, editors, *Advances in Cryptology - CRYPTO 2012*, volume 7417 of *Lecture Notes in Computer Science*, pages 643–662. Springer, 2012.
- DS83. Danny Dolev and H. Raymond Strong. Authenticated algorithms for byzantine agreement. *SIAM J. Comput.*, 12(4):656–666, 1983.
- DZ13. Ivan Damgård and Sarah Zakarias. Constant-overhead secure computation of boolean circuits using preprocessing. In *Theory of Cryptography*, pages 621–641. Springer, 2013.

- GKKO07. Juan A. Garay, Jonathan Katz, Chiu-Yuen Koo, and Rafail Ostrovsky. Round complexity of authenticated broadcast with a dishonest majority. In *48th Annual IEEE Symposium on Foundations of Computer Science (FOCS 2007), October 20-23, 2007, Providence, RI, USA, Proceedings*, pages 658–668, 2007.
- GL05. Shafi Goldwasser and Yehuda Lindell. Secure multi-party computation without agreement. *J. Cryptology*, 18(3):247–287, 2005.
- GMO16. Irene Giacomelli, Jesper Madsen, and Claudio Orlandi. Zkboo: Faster zero-knowledge for boolean circuits. Cryptology ePrint Archive, Report 2016/163, 2016. <http://eprint.iacr.org/>.
- GMW87. Oded Goldreich, Silvio Micali, and Avi Wigderson. How to play any mental game or A completeness theorem for protocols with honest majority. In *Proceedings of the 19th Annual ACM Symposium on Theory of Computing, 1987, New York, New York, USA*, pages 218–229, 1987.
- Gol01. Oded Goldreich. *The Foundations of Cryptography - Volume 1, Basic Techniques*. Cambridge University Press, 2001.
- GVW15. Sergey Gorbunov, Vinod Vaikuntanathan, and Daniel Wichs. Leveled fully homomorphic signatures from standard lattices. In *Proceedings of the Forty-Seventh Annual ACM Symposium on Theory of Computing, STOC 2015, Portland, OR, USA, June 14-17, 2015*, pages 469–477, 2015.
- HR14. Martin Hirt and Pavel Raykov. Multi-valued byzantine broadcast: The  $t < n$  case. In *Advances in Cryptology - ASIACRYPT 2014 - 20th International Conference on the Theory and Application of Cryptology and Information Security, Kaoshiung, Taiwan, R.O.C., December 7-11, 2014, Proceedings, Part II*, pages 448–465, 2014.
- HSZI00. Goichiro Hanaoka, Junji Shikata, Yuliang Zheng, and Hideki Imai. Unconditionally secure digital signature schemes admitting transferability. In *Advances in Cryptology—ASIACRYPT 2000*, pages 130–142. Springer, 2000.
- IKOS07. Yuval Ishai, Eyal Kushilevitz, Rafail Ostrovsky, and Amit Sahai. Zero-knowledge from secure multiparty computation. In *Proceedings of the 39th Annual ACM Symposium on Theory of Computing, San Diego, California, USA, June 11-13, 2007*, pages 21–30, 2007.
- IOS12. Yuval Ishai, Rafail Ostrovsky, and Hakan Seyalioglu. Identifying cheaters without an honest majority. In *Theory of Cryptography - 9th Theory of Cryptography Conference, TCC 2012, Taormina, Sicily, Italy, March 19-21, 2012. Proceedings*, pages 21–38, 2012.
- IOZ14. Yuval Ishai, Rafail Ostrovsky, and Vassilis Zikas. Secure multi-party computation with identifiable abort. In *Advances in Cryptology—CRYPTO 2014*, pages 369–386. Springer, 2014.
- LZ13. Yehuda Lindell and Hila Zarosim. On the feasibility of extending oblivious transfer. In *TCC*, pages 519–538, 2013.
- PW92. Birgit Pfitzmann and Michael Waidner. Unconditional byzantine agreement for any number of faulty processors. In *STACS 92, 9th Annual Symposium on Theoretical Aspects of Computer Science, Cachan, France, February 13-15, 1992, Proceedings*, pages 339–350, 1992.
- Sey12. Hakan Ali-John Seyalioglu. *Reducing Trust When Trust is Essential*. PhD thesis, University of California, Los Angeles, 2012. <https://escholarship.org/uc/item/7301296m>.
- Sim85. Gustavus J Simmons. Authentication theory/coding theory. In *Proceedings of CRYPTO 84 on Advances in Cryptology*, pages 411–431, New York, NY, USA, 1985. Springer-Verlag New York, Inc.
- SMY04. Reihaneh Safavi-Naini, Luke McAven, and Moti Yung. General group authentication codes and their relation to "unconditionally-secure signatures". In *Public Key Cryptography - PKC 2004, 7th International Workshop on Theory and Practice in Public Key Cryptography, Singapore, March 1-4, 2004*, pages 231–247, 2004.
- SS11. Colleen M Swanson and Douglas R Stinson. Unconditionally secure signature schemes revisited. In *Information Theoretic Security*, pages 100–116. Springer, 2011.
- SW99. Reihaneh Safavi-Naini and Huaxiong Wang. Multireceiver authentication codes: Models, bounds, constructions, and extensions. *Inf. Comput.*, 151(1-2):148–172, 1999.