Efficient Secure Multiparty Computation with Identifiable Abort

Carsten Baum[1*], Emmanuela Orsini[2 †], and Peter Scholl[2 ‡]

[1] Department of Computer Science, Aarhus University
[2] Department of Computer Science, University of Bristol

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 roughly the same performance as the SPDZ protocol (Damgård et al., Crypto 2012), requiring $O(n)$ messages to be broadcast for each secure multiplication. 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 and other recent works with applicable efficiency improvements.

Keywords: Secure Multiparty Computation, Identifiable Abort

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+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 desiderable property and intuitively means that either every party receives the output, or else no one does.

*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.

†Supported in part by ERC Advanced Grant ERC-2010-AdG-267188-CRIPTO.

‡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.

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 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.

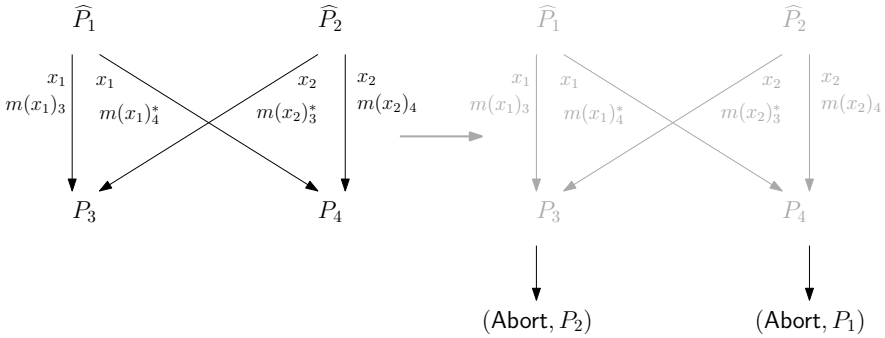

Fig. 1. Counterexample for identifiable abort with pairwise MACs

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

Our Contributions. In this work we present an efficient MPC protocol in the preprocessing model that reactively computes arithmetic circuits over a finite field, providing security with identifiable abort against up to $n-1$ out of n malicious parties. The online phase relies only on efficient, information-theoretic primitives and a broadcast channel, with roughly the same complexity as the SPDZ protocol [DPSZ12]. The offline

phase, which generates correlated randomness, can be instantiated using somewhat homomorphic encryption based on ring-LWE, and allows use of all the relevant optimisations presented in [DPSZ12,DKL+13,BDTZ16].

A first building block towards achieving this goal is our definition of *homomorphic information-theoretic signatures* (HITS). Information-theoretic signature schemes [CR91] cannot have a public verification key (since otherwise an unbounded adversary can easily forge messages), but instead each party holds a private verification key. The main security properties of IT signatures are unforgeability and consistency, meaning that no-one can produce a signature that verifies by one honest party but is rejected by another. Swanson and Stinson [SS11] were the first to formally study and provide security proofs for IT signatures, and demonstrated that many subtle issues can arise in definitions. On the other hand, homomorphic signature schemes [BFKW09,CJL09] feature an additional *homomorphic evaluation* algorithm, which allows certain functions to be applied to signatures. The verification algorithm is then given a signature, a message m and a description of a function f, and verifies that m is the output of f, applied to some previously signed inputs.

We give the first definition of HITS, and the first construction of a *linearly homomorphic* information-theoretic signature scheme, 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. The correlated randomness setup for our protocol requires that each party is provided with random shares of correlated data (multiplication triples), along with signatures on their shares and a private verification key. The underlying signing key is unknown to all parties. Given this setup, the protocol is then similar to the online phase of SPDZ and BDOZ: The parties' inputs are additively shared and authenticated with signatures on shares; then using the homomorphism of signatures, linear computation on shared data can be performed locally, and multiplication is done using preprocessed triples. While it seems straightforward that this modification implies ID-MPC, there are many cases that must be considered. In particular, we must be sure that the properties of the HITS are enough to provide privacy and ensure that no dishonest party is able to accuse an honest party of cheating.

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, and to achieve identifiable abort we require a modified version of the distributed decryption procedure from [DPSZ12].

In Section 7, 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.

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 higly 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 [IKOS07] for proving the correctness of each message in zero-knowledge. Although recent work by Giacomelli et al. [GMO16] shows that this technique in fact can be efficient for certain applications, it is not known whether this is true in the above setting. Note that Ishai et al. also use IT signatures for authenticating values, similarly to our usage, but without the homomorphic property that allows our protocol to be efficient. For the preprocessing stage, they describe

an elegant transformation that converts any protocol for implementing any correlated randomness setup in the OT-hybrid model into one with identifiable abort, which makes black-box use of an OT protocol. Again, unfortunately this method is not particularly practical, mainly because it requires the OT protocol to be secure against an adaptive adversary, which is much harder to achieve than statically secure OT [LZ13].

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

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

Organisation. In 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 and proves its security. 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 show how to upgrade the security of our ID-MPC protocol to be publicly verifiable by an external party.

2 Preliminaries

2.1 Notation

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

2.2 Model

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

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

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

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

2.3 Broadcast Channel

Our protocols require all messages to be sent across 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 ℓ-bit messages. Hirt and Raykov [HR14] presented a protocol that reduces the communication cost to $O(\ell n)$ when ℓ is large enough. We are not aware of any works analysing the practicality of these protocols, so we suggest this as an important direction for future research, along with investigating the extent to which our protocols can reduce use of the broadcast primitive.

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[1] 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 homomorphism to produce arbitrary related signatures, the verification algorithm must be given a function, and then verifies that the signed message is a valid output of the function on some previously signed messages.

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

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

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

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

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

- $0/1 \leftarrow \mathsf{Ver}(m, \sigma, f, \mathsf{vk}_j)$ *takes as input a message m, a signature σ, a function f and P_j's verification key vk_j, and checks that m is the valid, signed output of f.*

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

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

[1] 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.

To define security, we allow $\mathcal{A}$ to make use of the signing oracle $\mathcal{O}^S$: this takes a message m as input and outputs $\sigma = \mathsf{Sign}(m, \mathsf{sk})$. The number w of queries that an adversary is allowed to make to this oracle is defined in parameters of the scheme. Note that we could also have a stronger security notion using a verification oracle, but this is not needed for our application.

We then define security as follows:

Definition 3 (w-security). *A* HITS *scheme* $(\mathsf{Gen}, \mathsf{Sign}, \mathsf{Ver}, \mathsf{Eval})$ *is* w-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 identify function* $\mathsf{id}_i(m_1, \ldots, m_\ell) = m_i$. *Then we require that for every pair* $\mathsf{sk}, \mathbf{vk}$ *output by* Gen, *for any* $(m_1, \ldots, m_\ell) \in \mathcal{M}^\ell$, *and for all* $i \in [\ell], j \in [n]$,

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

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

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

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

1. *Let* $(\mathsf{sk}, \mathbf{vk}) \leftarrow \mathsf{Gen}(1^\kappa, w)$ *and send* $\{\mathsf{vk}_i\}_{i \in \mathcal{I}}$ *to the adversary.*
2. $\mathcal{A}$ *may query the oracle* $\mathcal{O}^S$ *up to a maximum of* w *times. Let* Q *be the set of messages queried to* $\mathcal{O}^S$.
3. $\mathcal{A}$ *outputs a function* $f \in \mathcal{F}$, *a list of input messages* $(m_1, \ldots, m_\ell) \in Q^\ell$, *a target message* m^* *and a signature* σ^*.
4. $\mathcal{A}$ *wins if* $m^* \neq f(m_1, \ldots, m_\ell)$ *and there exists* $j \in [n] \setminus \mathcal{I}$ *for which*

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

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

$$\Pr[\mathcal{A} \text{ wins}] = \mathsf{negl}(\kappa).$$

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* [2] *if there exist* $i, j \in [n] \setminus \mathcal{I}$ *such that*

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

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

$$\Pr[\mathcal{A} \text{ wins}] = \mathsf{negl}(\kappa).$$

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

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

[2] Note there is no requirement that $m^* \neq f(m_1, \ldots, m_\ell)$.

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

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

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

4 Construction of HITS

We now describe our construction of linearly homomorphic information-theoretic signatures. The message space $\mathcal{M}$ is a finite field $\mathbb{F}$, where $|\mathbb{F}| \geq 2^\kappa$. We restrict the function class $\mathcal{F}$ to be the set of all linear transformations $f : \mathbb{F}^w \to \mathbb{F}$ (where w is the maximum number of signatures that can be produced). The general case of linear funtions with fewer than w inputs can be handled by using a default value, $\perp$, for the unused input variables. Note also that the signing algorithm is *stateful*, and must keep track of how many messages have been signed previously.

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

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

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

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

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

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

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

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

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

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

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

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

Then check that

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

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

4.1 Security

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

Proof. 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 $f = \mathsf{id}_i$, so we show the latter.

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

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

Using these signatures and f, Eval then computes $\boldsymbol{\sigma}^f = \sum_{i=1}^w \mu_i \cdot \boldsymbol{\sigma}_i$. 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

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

and the algorithm will output 1.

Unforgeability. Before we start proving the unforgeability and consistency of the scheme, we prove that the adversary does not have enough information to reconstruct sk at any point.

Lemma 1. *Let $\mathcal{I} \subsetneq [n]$ be an index set of corrupted verifiers, and define the following game for an adversary $\mathcal{A}$:*

1. *Let $(\mathsf{sk}, \mathbf{vk}) \leftarrow \mathsf{Gen}(1^\kappa, w)$ and send $\{\mathsf{vk}_j\}_{j \in \mathcal{I}}$ to the adversary.*
2. *$\mathcal{A}$ may query the oracle $\mathcal{O}^S$ up to a maximum of w times.*
3. *$\mathcal{A}$ outputs a value sk'.*

Then $\Pr[\mathsf{sk}' = \mathsf{sk}] \leq 1/|\mathbb{F}|$ when the probability is taken over the randomness of Gen and $\mathcal{A}$.

Proof. In the first step of the game, $\mathcal{A}$ obtains

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

It then queries $\mathcal{O}^S$ on messages $m_1, ..., m_w \in \mathbb{F}$ and obtains

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

We now show that this information is not sufficient to recover the $\hat{\alpha}_r, \hat{\beta}_{j,i}$.

Collect all of $\mathcal{A}$'s information about the $\hat{\alpha}_r, \hat{\beta}_{i,r}$ into a system of linear equations. For now, consider the setting where $w = 1$ and $\mathcal{I} = \{1, ..., n-1\}$. 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. [3] 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 for that reason.

Therefore, the matrix has rank at most $2n-1$ for $2n$ unknowns and $\mathcal{A}$ cannot recover all $\hat{\alpha}_r, \hat{\beta}_{i,r}$ values. 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 $\hat{\alpha}_i, \hat{\beta}_i$ were chosen uniformly at random the claim follows. □

In the definition of the unforgeability game, $\mathcal{A}$ chooses a function f by fixing $\mu_1, ..., \mu_w, c$. It is easy to see that the choice of c does not make any difference, so assume that $c = 0$ for simplicity. Let $m_1, \ldots, m_w$ be the messages queried to $\mathcal{O}^S$ and $\boldsymbol{\sigma}_1, \ldots, \boldsymbol{\sigma}_w$ the responses. We conclude that, for all $j \in [n] \setminus \mathcal{I}$, the equation

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

with

$$\boldsymbol{\sigma}^f = \sum_{i=1}^{w} \mu_i \cdot \boldsymbol{\sigma}_i \quad \text{and} \quad m^f = \sum_{i=1}^{w} \mu_i \cdot m_i \quad \text{and} \quad \beta_j^f = \sum_{i=1}^{w} \mu_i \cdot \beta_{j,i},$$

must hold due to evaluation correctness. Now $\mathcal{A}$ picks $m^* \in \mathbb{F}$ such that $m^* \neq f(m_1, ..., m_w)$ and a signature $\boldsymbol{\sigma}^*$, and assume for the sake of contradiction that $\mathcal{A}$ wins the game, so

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

[3] 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.

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

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

In Lemma 1 we ruled out that $\mathcal{A}$ can compute all of the $\hat{\alpha}_r$ and therefore cannot provide the trivial solution to the above equation. Then by the rank nullity theorem, the linear map induced by the right-hand side of (3) has a kernel of dimension $n-1$. All $v_{j,1}, ..., v_{j,n}$ were chosen uniformly at random and are unknown by $\mathcal{A}$, so Equation (3) will hold with probability $|\mathbb{F}^{n-1}|/|\mathbb{F}^n| = 1/|\mathbb{F}|$ which is negligible in κ. Note that the above argument generalises to an arbitrary $\mathcal{I}$.

Consistency. Assume that $\mathcal{A}$ can break consistency with non-negligible probability. By definition of the game, it must then hold that at least one of $m^* \neq f(m_1, ..., m_w)$ or $\boldsymbol{\sigma}^* \neq \sum_{i=1}^{w} \mu_i \cdot \boldsymbol{\sigma}_i$: If $m^* = f(m_1, ..., m_w)$ and $\boldsymbol{\sigma}^* = \sum_{i=1}^{w} \mu_i \cdot \boldsymbol{\sigma}_i$ then by evaluation correctness, $\mathsf{Ver}(m^*, \boldsymbol{\sigma}^*, f, \mathsf{vk}_j) = 1, \forall j \in [n]$, which contradicts the winning condition.

Suppose $m^* \neq f(m_1, ..., m_w)$, then we can simply run $\mathcal{A}$ in the unforgeability game and, by definition, it will win with non-negligible probability.

Otherwise, we have $m^* = f(m_1, ..., m_w)$ but $\boldsymbol{\sigma}^* \neq \sum_{i=1}^{w} \mu_i \cdot \boldsymbol{\sigma}_i$. Then by the winning condition, it must hold that

$$0 = \sum_{r=1}^{n} v_{j,r} \cdot \Bigg(\underbrace{\boldsymbol{\sigma}^*|_r - (m^* - c) \cdot \hat{\alpha}_r - \sum_{i=1}^{w} \mu_i \cdot \hat{\beta}_{i,r}}_{=x_r} \Bigg), \tag{4}$$

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

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

Hence for every r it must hold that

$$\begin{aligned} \boldsymbol{\sigma}^*|_r &= \hat{\alpha}_r \cdot (m^* - c) + \sum_{i=1}^{w} \mu_i \cdot \hat{\beta}_{i,r} \\ &= \hat{\alpha}_r \cdot (f(m_1, ..., m_w) - c) + \sum_{i=1}^{w} \mu_i \cdot \hat{\beta}_{i,r} \\ &= \hat{\alpha}_r \cdot (\mu_1 \cdot m_1 + \ldots \mu_w \cdot m_w) + \sum_{i=1}^{w} \mu_i \cdot \hat{\beta}_{i,r} \\ &= \sum_{i=1}^{w} \mu_i \cdot (\hat{\alpha}_r \cdot m_i + \hat{\beta}_{i,r}) \\ &= \sum_{i=1}^{w} \mu_i \cdot \boldsymbol{\sigma}_i|_r, \end{aligned}$$

which contradicts the assumption.

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

$$f(x_1, \ldots, x_w) = \sum_{i=1}^{w} \mu_i \cdot x_i + c.$$

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

1. Compute, for $r \in [n]$,
$$\boldsymbol{\sigma}^*|_r = \sum_{i \in [w]} \mu_i \cdot \hat{\beta}_{i,r} + \hat{\alpha}_r \cdot (m - c).$$
2. Output $\boldsymbol{\sigma}^*$.

Now let $\boldsymbol{\sigma}_i \leftarrow \mathsf{Sign}(\mathsf{sk}, m_i)$, for $i \in [w]$, and $\boldsymbol{\sigma} = \mathsf{Eval}(f, \boldsymbol{\sigma}_1, ..., \boldsymbol{\sigma}_w)$. Note that $m - c = \sum_i \mu_i \cdot m_i$, and then we have, for all $r \in [n]$,

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

as required. □

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 $\{P_1, \ldots, P_n\}$, and any HITS scheme $\mathsf{HITS} = (\mathsf{Gen}, \mathsf{Sign}, \mathsf{Eval}, \mathsf{Ver})$ that satisfies w-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 $\mathcal{F}_{\mathsf{MPC}}$, shown in Fig. 2. Note that $\mathcal{F}_{\mathsf{MPC}}$ already contains an explicit command for identifiable abort in the output stage, since it models an unfair execution where the adversary can abort after learning the output. The modified functionality $[\mathcal{F}_{\mathsf{MPC}}]^{\mathsf{ID}}_{\perp}$ then extends this abort to be possible at any time.

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

- **Input tuples:** Random shared values $[\![r]\!]$, such that one party, P_i, knows r. This is used so that P_i can provide input in the online phase.

- **Multiplication triples:** Random shared triples $[\![a]\!], [\![b]\!], [\![c]\!]$, where $a, b \leftarrow \mathbb{F}$ and $c = a \cdot b$.

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

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

[4] For addition with a constant, all parties adjust their signatures but only one party, say P_1, needs to adjust their share.

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

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

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

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

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

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

Fig. 4. Ideal functionality for the preprocessing phase.

Protocol Π_{Online}

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

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

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

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

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

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

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

5.1 Security Proof

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

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

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

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

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

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

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

In a multiplication gate, for each call to Open, $\mathcal{S}$ receives all the corrupt shares $(t_j^*, \sigma_{t_j^*})$ from $\mathcal{A}$, and computes and sends the shares and signatures for the dummy honest parties as in the protocol. Let (t_j, σ_{t_j}) be the values that $\mathcal{S}$ expects from the dishonest P_j, based on previous computations and the simulated preprocessing data. $\mathcal{S}$ checks for all the $(t_j^*, \sigma_{t_j^*})$ received from $\mathcal{A}$ and for all $i \in \mathcal{P} \setminus \mathcal{I}$ 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 (Abort, P_j) to $[\mathcal{F}_{\mathsf{MPC}}]_{\perp}^{\mathtt{ID}}$. Otherwise it proceeds. [5]

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

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

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

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

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

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

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

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

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

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

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

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

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

5.2 Batch Signature Verification

Protocol $\mathsf{SignCheck}$

Usage: Let $\{a_j\}_{j\in[t]}$ be all the values that party P_i has revealed during the protocol to all the other parties.

$\mathsf{SignCheck}((a_1, \ldots, a_t), P_i)$**:**

1. Sample $(r_1, \ldots, r_t) \in \mathbb{F}^t$ using $\mathcal{F}_{\mathsf{Rand}}$
2. P_i computes $\sigma = \mathsf{Eval}(f, (\sigma_{a_1}, \ldots, \sigma_{a_t}))$, where the linear function f is defined as $f(x_1, \ldots, x_t) = r_1 \cdot x_1 + \cdots + r_t \cdot x_t$.
3. Every party P_j (for $j \neq i$) computes $f(a_1, \ldots, a_t) = a$ and checks if $\mathsf{Ver}(a, \sigma, f, \mathsf{vk}_j) = 1$. If the check fails, output (Abort, P_i).

Fig. 6. Method to check a batch of signatures on revealed values

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. We can reduce this to $O(n)$ — the same cost as [DPSZ12] — 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 broadcast shares without the corresponding signatures. We then check a random linear combination of each parties' signatures just before every output stage. The total communication cost of the protocol is now $n_I + 2nn_M + O(n^2)$ field elements to be broadcast, where n_I is the total number of private inputs and n_M the number of secure multiplications in the circuit being evaluated. Meanwhile, the storage cost (for the preprocessing data) is $O(n(n_M + n_I))$ per party, which is a factor $O(n)$ worse than SPDZ, since we need additional verification data for every party's share. It is an interesting open problem to see if this can be improved.

The protocol $\mathsf{SignCheck}$ (Fig. 6) either succeeds or outputs (Abort, P_i) if some inconsistent message/signature pair is found. The evaluation correctness and security properties of HITS guarantee that passing this check implies:

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

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

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

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

where a'_i are the actual values the adversary opened to, a_i are the original signed values and r_i are uniformly random values sampled from $\mathcal{F}_{\mathsf{Rand}}$ (Fig. 7)[6]. It is easy to see that if at least one $a_i \neq a'_i$ then 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 cheating with overwhelming probability.

6 Preprocessing with Identifiable Abort

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

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

- The protocol to be compiled is assumed to consist only of calls to an ideal OT functionality and a broadcast channel. This means that any pairwise communication must be performed via the OT functionality and so is very expensive.

- The transformation requires first computing an authenticated secret sharing of the required output, and then opening this to get the output. In our case, the output of $\mathcal{F}_{\mathsf{Prep}}$ is already secret shared and authenticated, so intuitively, this step seems unnecessary.

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

Even though there has recently been some progress in efficiently implementing preprocessing for MPC using OT [FKOS15], the first and third points above in particular seem to imply that the method of Ishai et al. will not be efficient. Instead, we show how to modify the original SPDZ preprocessing [DPSZ12] so that the protocol produces the new preprocessing data we need, and ensures security against identifiable abort. In particular, the key differences between our protocol and [DPSZ12] are:

- We use a distributed decryption protocol that is guaranteed to produce the correct plaintext, which ensures that all outputs are correct at the end of the preprocessing. The original SPDZ protocol allowed for possible adversarial errors in some MAC or triple values defined in the output of $\mathcal{F}_{\mathsf{Prep}}$, provided that these would be detected in the online phase. In our case, any cheating must be identifiable, so we cannot allow these errors.

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

- Zero-knowledge proofs used for proving correct ciphertext generation must be *public-coin* (i.e. the verifier's messages are its random coin tosses) to ensure that all parties can consistently verify them. (This is already the case for the proofs in [DPSZ12], but is not necessarily required.)

- Instead of producing shares of MACs on the preprocessing data, we provide each party with individual signatures and verification keys on their shares, using a private distributed decryption protocol that is secure with identifiable abort.

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

The $\mathcal{F}_{\mathsf{Prep}}$ 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 $\mathcal{F}_{\mathsf{Prep}}$, shown in Fig. 8, which simplifies our preprocessing protocol by not requiring the adversary's verification data, $\mathbf{v}_j$, to be uniformly random. The following proposition shows that the scheme, and therefore online phase, remain secure with this modification.

Proposition 1. *The scheme* HITS *remains secure when* Gen *is modified to allow adversarial inputs, as in* $\mathcal{F}^*_{\mathsf{Prep}}$.

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

On input $(\mathsf{Init}, n_M, n_I)$ 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 $(\mathsf{sk}, \mathbf{vk}) \leftarrow \mathsf{HITS.Gen}(1^\kappa, w)$, except using the provided values $\{\mathbf{v}_j\}_{j\in\mathcal{I}}$ to compute $\{\mathsf{vk}_j\}_{j\in\mathcal{I}}$, instead of sampling fresh values.
3. Send vk_i to party P_i and store sk.

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

6.2 SHE Scheme Requirements

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

$$\langle x \rangle = \mathsf{Enc}(x).$$

The binary operators $\boxplus, \boxtimes$ then guarantee that

$$\langle x + y \rangle = \langle x \rangle \boxplus \langle y \rangle$$
$$\langle x \cdot y \rangle = \langle x \rangle \boxtimes \langle y \rangle$$

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+13], a ring-LWE variant of the BGV scheme [BGV12] is practical for this purpose, and this also allows homomorphic operations to be batched for greater efficiency.

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

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

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

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

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

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

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

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

Fig. 9. SHE distributed key generation and decryption functionality

6.3 Basic Subprotocols

Fig. 10 shows 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 $\mathcal{F}_{\mathsf{KeyGenDec}}$, instead of a possibly faulty method. This means that there is no way the adversary can introduce errors into triples, so we avoid the need for the pairwise sacrificing procedure from [DPSZ12], where half of the triples are wasted to check correctness. The main other difference in our protocol, compared to [DPSZ12], is that we need to setup verification keys for the signature scheme and create signatures on every share, which is more complex than setting up simple MACs.

Subprotocol RandShCtxt():

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

Subprotocol ShareDec($\langle m \rangle$, new_ctxt):

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

Subprotocol PrivateDec($\langle x \rangle, P_j$):

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

Fig. 10. Subprotocols for the preprocessing protocol using SHE

Protocol Π_{Prep}

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

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

1. Run $\mathcal{F}_{\mathsf{KeyGenDec}}$.KeyGen to obtain an SHE public key pk.
2. Run RandShCtxt() $2n$ times, so each party P_j obtains random elements $\hat{\alpha}_r^j, v_{j,r}$, for $r = 1, \ldots, n$, and everyone obtains ciphertexts $\langle \hat{\alpha}_r^j \rangle, \langle v_{j,r} \rangle$ encrypting these.
3. Run RandShCtxt() $w(n)$ times, so party P_j obtains $\hat{\beta}_{r,i}^j$, for $r \in [n]$ and $i \in [w]$, and everyone gets the corresponding ciphertexts.
4. Homomorphically compute, for $r \in [n]$:
$$\langle \hat{\alpha}_r \rangle = \langle \hat{\alpha}_r^1 \rangle \boxplus \cdots \boxplus \langle \hat{\alpha}_r^n \rangle$$
$$\langle \hat{\beta}_{r,i} \rangle = \langle \hat{\beta}_{r,i}^1 \rangle \boxplus \cdots \boxplus \langle \hat{\beta}_{r,i}^n \rangle \qquad \text{for } i \in [w]$$
5. Now compute the encrypted verification keys, for $j \in [n]$:
$$\langle \alpha_j \rangle = \boxplus_{r=1}^{n} \left(\langle \hat{\alpha}_{j,r} \rangle \boxtimes \langle v_{j,r} \rangle \right)$$
$$\langle \beta_{j,i} \rangle = \bigboxplus_{r=1}^{n} \left(\langle \hat{\beta}_{j,r} \rangle \boxtimes \langle v_{j,r} \rangle \right)$$
6. Run the subprotocol PrivateDec($\langle \alpha_j \rangle, P_j$) and PrivateDec($\langle \beta_{j,i} \rangle, P_j$) for $i \in [w]$ and $j \in [n]$, so that each party P_j gets their verification key vk_j.
7. All parties store $\langle \hat{\alpha}_i \rangle, \langle \hat{\beta}_i \rangle$, for $r \in [n]$, and their private verification keys, $\mathsf{vk}_j = (\mathbf{v}_j, \alpha_j, \{\beta_{j,i}\}_{i \in [w]}$.

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

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

Protocol Π_{Prep} (continued)

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

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

Triple: Creates a single, authenticated multiplication triple.

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

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

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

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

ciphertexts encrypting the signing key, and then ciphertexts encrypting the verification key values $\alpha_j, \beta_{j,i}$ for party P_j, for $i \in [w]$. These verification keys are then privately decrypted to each party.

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

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

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

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

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

6.5 Security

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* Gen^* *that outputs a meaningless public key,* pk^**, such that for* $\mathsf{pk} \leftarrow \mathsf{Gen}(1^\lambda), \mathsf{pk}^* \leftarrow \mathsf{Gen}^*(1^\lambda)$ *and any message* x*:*

$$\mathsf{Enc}(\mathsf{pk}^*, x) \overset{s}{\approx} \mathsf{Enc}(\mathsf{pk}^*, 0)$$
$$\mathsf{pk} \overset{c}{\approx} \mathsf{pk}^*,$$

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

Theorem 3. *The protocol* Π_{Prep} *securely realises* $\mathcal{F}^*_{\mathsf{Prep}}$ *with identifiable abort in the* $\mathcal{F}_{\mathsf{KeyGenDec}}$*-hybrid model, with computational security.*

Proof (Sketch). 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 Π_{Prep}, and an interaction with $\mathcal{S}$ and the ideal functionality $\left[\mathcal{F}^*_{\mathsf{Prep}}\right]^{\mathtt{ID}}_{\perp}$. 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.

The simulator $\mathcal{S}$ first generates an SHE key pair $(\mathsf{pk}, \mathsf{SHE.sk}) \leftarrow \mathsf{SHE.Gen}(1^\lambda)$ and sends 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 $\mathcal{F}_{\mathsf{KeyGenDec}}.\mathsf{PrivateDec}$, to decrypt the verification keys. At this point, use the SHE key sk to decrypt the adversary's ciphertexts containing the verification data $\mathbf{v}_j$ for $j \in \mathcal{I}$, and send these to $\left[\mathcal{F}^*_{\mathsf{Prep}}\right]^{\mathtt{ID}}_{\perp}$. Receive from $\left[\mathcal{F}^*_{\mathsf{Prep}}\right]^{\mathtt{ID}}_{\perp}$ the remaining verification data $\alpha_j, \{\beta_{j,i}\}_{i\in[w]}$ and send this to the adversary, simulating the output of $\mathcal{F}_{\mathsf{KeyGenDec}}.\mathsf{PrivateDec}$ 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 $\mathcal{F}^*_{\mathsf{Prep}}$, along with the shares of c_i that are extracted from ciphertexts received in the $\mathsf{ShareDec}$ procedure. In response, the simulator receives from $\mathcal{F}^*_{\mathsf{Prep}}$ signatures on all corrupted parties' shares, and uses these to simulate the final calls to $\mathcal{F}_{\mathsf{KeyGenDec}}.\Pi_{\mathsf{PubDec}}$ during the Auth subroutine.

To ensure security with identifiable abort, we require that if at any stage during the simulation, a proof of correct ciphertext generation (Π_{ZKPoK}) fails for P_j where $j \in \mathcal{I}$, then $\mathcal{S}$ sends (Abort, P_j) to $\left[\mathcal{F}^*_{\mathsf{Prep}}\right]^{\mathtt{ID}}_{\perp}$ and halts. Since Π_{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 $[\mathcal{F}_{\mathsf{Prep}}]^{\mathtt{ID}}_{\perp}$ using Sign and the signing key of $[\mathcal{F}_{\mathsf{Prep}}]^{\mathtt{ID}}_{\perp}$. 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.

7 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 $\mathcal{P} = \{P_1, ..., P_n\}$, 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 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

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

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

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

Moreover, no corrupt 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 Jud' and $i \notin \mathcal{I}$ it holds that

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

This third property is necessary to exclude the *trivial* solution where a party P_i allows Jud to check the transcript by opening a commitment to its key vk_i, as this would reveal the input P_i to 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[7] $\mathcal{B}$. 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 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 Jud only obtains information that $\mathcal{A}$ learns as well.

Now, the Ver algorithm will be run such that it generates verification keys for $2n$ parties. As before, we privately give vk_i to P_i but additionally post commitments to $\mathsf{vk}_{n+1}, ..., \mathsf{vk}_{2n}$ on $\mathcal{B}$. The opening information to the commitment of vk_{n+i} is given to P_i. Moreover, we demand that each commitment is signed by all

[7] Such a bulletin board can be constructed e.g. using the blockchain of Bitcoin.

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 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 defamate 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.

References

AJL+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 multiparty computation. *IACR Cryptology ePrint Archive*, 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+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 Roijakkers. Unconditionally-secure digital signatures. In *Advances in Cryptology-CRYPT0'90*, pages 206–214. Springer, 1991.

DKL+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.

FKOS15. Tore Kasper Frederiksen, Marcel Keller, Emmanuela Orsini, and Peter Scholl. A unified approach to MPC with preprocessing using OT. In *Advances in Cryptology - ASIACRYPT 2015 - 21st International Conference on the Theory and Application of Cryptology and Information Security, Auckland, New Zealand, November 29 - December 3, 2015, Proceedings, Part I*, pages 711–735, 2015.

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 on 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.

SS11. Colleen M Swanson and Douglas R Stinson. Unconditionally secure signature schemes revisited. In *Information Theoretic Security*, pages 100–116. Springer, 2011.