Linear Overhead Optimally-resilient Robust MPC Using Preprocessing

Ashish Choudhury[1], Emmanuela Orsini[2], Arpita Patra[3], and Nigel. P. Smart[2]

[1] International Institute of Information Technology Bangalore, India.
ashish.choudhury@iiitb.ac.in
[2] Dept. Computer Science, University of Bristol, Bristol, United Kingdom.
Emmanuela.Orsini@bristol.ac.uk,nigel@cs.bris.ac.uk
[3] Department of Computer Science and Automation, Indian Institute of Science, India.
arpita@csa.iisc.ernet.in

Abstract. We present a new technique for robust secret reconstruction with $\mathcal{O}(n)$ communication complexity. By applying this technique, we achieve $\mathcal{O}(n)$ communication complexity per multiplication for a wide class of robust practical Multi-Party Computation (MPC) protocols. In particular our technique applies to robust threshold computationally secure protocols in the case of $t < n/2$ in the pre-processing model. Previously in the pre-processing model, $\mathcal{O}(n)$ communication complexity per multiplication was only known in the case of computationally secure non-robust protocols in the dishonest majority setting (i.e. with $t < n$) and in the case of perfectly-secure robust protocols with $t < n/3$. A similar protocol was sketched by Damgård and Nielsen, but no details were given to enable an estimate of the communication complexity. Surprisingly our robust reconstruction protocol applies for both the synchronous and asynchronous settings.

1 Introduction

Secure MPC is a fundamental problem in secure distributed computing [28,23,6,12]. An MPC protocol allows a set of n mutually distrusting parties with private inputs to securely compute a joint function of their inputs, even if t out of the n parties are corrupted. Determining the communication complexity of MPC in terms of n, is a task which is both interesting from a theoretical and a practical standpoint. It is a folklore belief that the complexity should be essentially $\mathcal{O}(n)$ per multiplication in the computation. However, "most" *robust* secret-sharing based MPC protocols which are practical have complexity $\mathcal{O}(n^2)$.

To understand the problem notice that apart from the protocols for entering parties inputs and determining parties outputs, the main communication task in secret-sharing based MPC protocols is the evaluation of the multiplication gates (we assume a standard arithmetic circuit representation of the function to be computed for purely expository reasons, in practice other representations may be better). If we consider the classic information-theoretic passively secure sub-protocol for multiplication gates when $t < n/2$ (locally multiply the shares, reshare and then recombine) we require $\mathcal{O}(n^2)$ messages per multiplication gate [6,22]. This is because each party needs to send the shares representing its local multiplication to every other party, thus requiring $\mathcal{O}(n^2)$ messages, and hence $\mathcal{O}(n^2)$ bits if we only look at complexity depending on n.

Even if we look at such protocols in the pre-processing model, where the so-called "Beaver multiplication triples" are produced in an offline phase [2], and we are primarily concerned about the communication complexity of the online phase, a similar situation occurs. In such protocols, see for example [16], the standard multiplication sub-protocol is for each party to broadcast a masking of their shares of the gate input values to every other party. This again has $\mathcal{O}(n^2)$ communication complexity.

In the SPDZ protocol [19], for the case of *non-robust*[4] maliciously secure MPC (with abort) in the dishonest majority setting (i.e. with $t < n$), an online communication complexity of $\mathcal{O}(n)$ was achieved. This is attained by replacing the broadcast communication of the previous method with the following trick. For each multiplication gate one party is designated as the "reconstructor". The broadcast round is then replaced by each party sending their masked values to the reconstructor, who then reconstructs the value and then sends it to each party. This requires exactly $2 \cdot n$ messages being sent, and is hence $\mathcal{O}(n)$. However, this protocol is only relevant in the dishonest majority setting as any dishonest behaviour of any party is subsequently detected via the SPDZ MAC-checking procedure, in which case the protocol aborts. Our goal is to achieve such a result for robust protocols in the *pre-processing model*.

Related Work: With $t < n/3$, information-theoretically secure an online protocols with $\mathcal{O}(n)$ communication per multiplication are presented in [18]. There the basic idea is a new method of reconstructing a batch of $\Theta(n)$ secret-shared values with $\mathcal{O}(n^2)$ communication complexity, thus providing a linear overhead. However, the method is tailor-made only for $t < n/3$ (as it is based on the error-correcting capability of the Reed-Solomon (RS) codes) and will not work with $t < n/2$. Hence with $t < n/2$ in the computational setting, a new technique to obtain $\mathcal{O}(n)$ *online complexity* is needed. In [18] a similar protocol in the pre-processing model is also sketched, which uses the designatred reconstructor idea (similar to the idea used in SPDZ, discussed above). The protocol is only sketched, and appears to require $O(t)$ rounds to identify the faulty shares; as opposed to our method which requires no additional rounds.

In [24], a computationally-secure MPC protocol with $t < n/2$ and communication complexity $\mathcal{O}(n)$ per multiplication is presented. The protocol is not designed in the pre-processing model, but rather in the player-elimination framework, where the circuit is divided into segments and each segment is evaluated "optimistically", assuming no fault will occur. At the end of the segment evaluation, a detection protocol is executed to identify whether the segment is evaluated correctly and if any inconsistency is detected, then a fault-localization protocol is executed. The fault-localization process identifies a pair of parties, with at least one of them being corrupted. The pair is then neglected for the rest of the protocol execution and the procedure is repeated. There are several drawbacks of this protocol. The protocol cannot be adapted to the pre-processing model; so the benefits provided by the pre-processing based MPC protocols

[4] An MPC protocol is called robust if the honest parties obtain the correct output at the end of the protocol irrespective of the behaviour of the corrupted parties, otherwise it is called non-robust.

(namely efficiently generating circuit-independent raw materials for several instances of the computation in parallel) cannot be obtained. The protocol also makes expensive use of zero-knowledge (ZK) machinery throughout the protocol and it does not seem to be adaptable to the asynchronous setting with $\mathcal{O}(n)$ communication complexity. Our techniques on the other hand are focused on efficient protocols in the pre-processing model. For example we use ZK tools only in the offline phase, and our online methods are easily applicable to the asynchronous communication setting[5], which models real-world networks like the Internet more appropriately than the synchronous communication setting.

In [7], an information-theoretically secure MPC protocol in the pre-processing model with $t < n/2$ and $\mathcal{O}(n)$ communication complexity per multiplication is presented. Both the offline and online phase of [7] are designed in the dispute control framework [3], which is a generalisation of the player-elimination technique and so like other papers in the same framework it is not known if the protocol can be made to work in the more practical asynchronous communication setting. Moreover since their online phase protocol is in the dispute control framework, it requires $\mathcal{O}(n^2 + \mathcal{D})$ rounds of interaction in the online phase, where $\mathcal{D}$ is the multiplicative depth of the circuit. This is unlike other MPC protocols in the pre-processing model whose online phase requires only $\mathcal{O}(\mathcal{D})$ rounds of interaction [18,4,8,19]. Our technique for the online phase protocol does not deploy any player-elimination/dispute-control techniques and so requires fewer rounds than [7]. And our online phase can be executed even in the asynchronous setting with $t < n/2$ and $\mathcal{O}(n)$ communication complexity. Imagine a scenario involving a large number of parties, participating from various parts of the globe. Clearly (an asynchronous) online protocol with less number of communication rounds is desirable here and so our online phase protocol will fit the bill appropriately. In the non-preprocessing model, information-theoretically secure MPC protocols with "near linear" amortized communication complexity but *non-optimal resilience* are presented in [1,17,21]. Namely the overall communication complexity of these protocols are $\mathcal{O}\left(\text{polylog}(n, C) \cdot C\right)$, where C is the circuit size. While the protocol of [17] is perfectly-secure and can tolerate upto $t < (1/3 - \epsilon) \cdot n$ corruptions where $0 < \epsilon < 1/3$, the protocols in [1,21] are statistical with resilience $t < (1/2 - \epsilon) \cdot n$ where $0 < \epsilon < 1/2$. The central idea in these protocols is to take advantage of the non-optimal resilience by deploying packed secret-sharing, where "several" values are secret shared simultaneously via a single sharing instance. None of the protocols are known to work in asynchronous settings and all of them heavily rely on the fact that there are more honest parties than just $1/2$ (making them non-optimal in terms of resilience).

Finally we note that an asynchronous MPC protocol with $t < n/3$ and $\mathcal{O}(n)$ communication complexity in the pre-processing model is presented in [15]. However the online phase protocol of [15] is based on the $\mathcal{O}(n)$ reconstruction method of [4,18] with $t < n/3$ and hence cannot be adapted to the $t < n/2$ setting.

Our Contribution: We present a computationally-secure method to obtain $\mathcal{O}(n)$ communication complexity for the online phase of robust MPC protocols with $t < n/2$.

[5] We stress that we are interested only in the online complexity. Unlike our online phase, our offline phase protocol cannot be executed in a completely asynchronous setting with $t < n/2$.

We are focused on protocols which could be practically relevant, so we are interested in suitable modifications of protocols such as VIFF [16], BDOZ [8] and SPDZ [19]. Our main contribution is a trick to robustly reconstruct a secret with an amortized communication complexity of $\mathcal{O}(n)$ messages. Assuming our arithmetic circuit is suitably wide, this implies an $\mathcal{O}(n)$ online phase when combined with the standard method for evaluating multiplication gates based on pre-processed Beaver triples.

To produce this sub-protocol we utilize the error-correcting capability of the underlying secret-sharing scheme when error positions are already known. To detect the error positions we apply the the pair-wise BDOZ MACs from [8]. The overall sub-protocol is highly efficient and can be utilized in practical MPC protocols. Interestingly our reconstruction protocol also works in the asynchronous setting. Thus we obtain a practical optimization in both synchronous and asynchronous setting.

Before proceeding we pause to examine the communication complexity of the offline phase of protocols such as SPDZ. It is obvious that in the case of a computationally secure offline phase one can easily adapt the somewhat homomorphic encryption (SHE) based offline phase of SPDZ to the case of Shamir secret sharing when $t < n/2$. In addition one can adapt it to generate SPDZ or BDOZ style MACs. And this is what we exactly do to implement our offline phase in the synchronous setting. In [19] the offline communication complexity is given as $\mathcal{O}(n^2/s)$ in terms of the number of messages sent, where s is the "packing" parameter of the SHE scheme. As shown in the full version of [20], assuming a cyclotomic polynomial is selected which splits completely modulo the plaintext modulus p, the packing parameter grows very slowly in terms of the number of parties (for all practical purposes it does not increase at all). In addition since s is in the many thousands, for all practical purposes the communication complexity of the offline phase is $\mathcal{O}(n)$ in terms of the number of messages. However, each message is $\mathcal{O}(s)$ and so the bit communication complexity is still $\mathcal{O}(n^2)$.

As our online phase also works in the asynchronous setting, we explore how the offline phase, and the interaction between the offline and online phases can be done asynchronously. For this we follow the VIFF framework [16], which implements the offline phase asynchronously with $t < n/3$ via the pseudo-random secret sharing, assuming a single synchronization point between the offline and online phases. Following the same approach, we show how the interaction between our offline and online phase can be handled asynchronously with $t < n/2$. However we require an additional technicality for $t < n/2$ to deal with the issue of agreement among the parties at the end of asynchronous offline phase. Specifically, we either require "few" synchronous rounds or a non-equivocation mechanism at the end of offline phase to ensure agreement among the parties. We stress that once this is done then the online phase protocol can be executed in a completely asynchronous fashion with $t < n/2$.

2 Preliminaries

We assume a set of parties $\mathcal{P} = \{P_1, \ldots, P_n\}$, connected by pair-wise authentic channels, and a centralized static, active PPT adversary $\mathcal{A}$ who can corrupt any $t < n/2$ parties. For simplicity we assume $n = 2t + 1$, so that $t = \Theta(n)$. The functionality that the parties wish to compute is represented by an arithmetic circuit over a finite field $\mathbb{F}$, where $|\mathbb{F}| > n$. We denote by μ and κ the statistical and cryptographic security parameter respectively. A negligible function in κ (μ) will be denoted by $\mathsf{negl}(\kappa)$ ($\mathsf{negl}(\mu)$), while $\mathsf{negl}(\kappa, \mu)$ denotes a function which is negligible in both κ and μ. We use both information-theoretic and public-key cryptographic primitives in our protocols. The security of the information theoretic primitives are parameterised with μ, while that of cryptographic primitives are parametrised with κ. We assume $\mathbb{F} = \mathrm{GF}(p)$, where p is a prime with $p \approx 2^{\mu}$, to ensure that the statistical security of our protocol holds with all but $\mathsf{negl}(\mu)$ probability. Each element of $\mathbb{F}$ can be represented by μ bits. For vectors $A = (a_1, \ldots, a_m)$ and $B = (b_1, \ldots, b_m)$, $A \otimes B$ denotes the value $\sum_{i=1}^{m} a_i b_i$. The ith element in a vector A is denoted as $A[i]$ and (i, j)th element in a matrix A as $A[i, j]$.

2.1 Communication Settings

In this paper we consider two communication settings. The first setting is the popular and simple, but less practical, synchronous channel setting, where the channels are synchronous and there is a strict upper bound on the message delays. All the parties in this setting are assumed to be synchronized via a global clock. Any protocol in this setting operates as a sequence of rounds, where in every round: A party first performs some computation, then they send messages to the others parties over the pair-wise channels and broadcast any message which need to be broadcast; this stage is followed by receiving both the messages sent to the party by the other parties over the pair-wise channels and the messages broadcast by the other parties. Since the system is synchronous, any (honest) party need not have to wait endlessly for any message in any round. Thus the standard behaviour is to assume that if a party does not receive a value which it is supposed to receive or instead it receives a "syntactically incorrect" value, then the party simply substitutes a default value (instead of waiting endlessly) and proceeds further to the next round.

The other communication setting is the more involved, but more practical, asynchronous setting; here the channels are asynchronous and messages can be arbitrarily (but finitely) delayed. The only guarantee here is that the messages sent by the honest parties will eventually reach their destinations. The order of the message delivery is decided by a *scheduler*. To model the worst case scenario, we assume that the scheduler is under the control of the adversary. The scheduler can only schedule the messages exchanged between the honest parties, without having access to the "contents" of these messages. As in [5,10], we consider a protocol execution in this setting as a sequence of *atomic steps*, where a single party is *active* in each step. A party is activated when it receives a message. On receiving a message, it performs an internal computation and then possibly sends messages on its outgoing channels. The order of the atomic steps are controlled by the scheduler. At the beginning of the computation, each party will be in a special *start* state. A party is said to *terminate/complete* the computation if it reaches a *halt* state, after which it does not perform any further computation. A protocol execution is said to be complete if all the honest parties terminate the computation.

It is easy to see that the asynchronous setting models real-world networks like the Internet (where there can be arbitrary message delays) more appropriately than the synchronous setting. Unfortunately, designing protocol in the asynchronous setting is complicated and this stems from the fact that we cannot distinguish between a corrupted sender (who does not send any messages) and a slow but honest sender (whose messages are arbitrarily delayed). Due to this the following unavoidable but inherent phenomenon is always present in any asynchronous protocol: at any stage of the protocol, no (honest) party can afford to receive communication from *all* the n parties, as this may turn out to require an endless wait. So as soon as the party hears from $n-t$ parties, it has to proceed to the next stage; but in this process, communication from t potentially honest parties may get ignored.

2.2 Primitives

Linearly-homomorphic Encryption Scheme (HE). We assume an IND-CPA secure linearly-homomorphic public-key encryption scheme set-up for every $P_i \in \mathcal{P}$ with message space $\mathbb{F}$; a possible instantiation could be the BGV scheme [9]. Under this set-up, P_i will own a secret decryption key $\mathbf{dk}^{(i)}$ and the corresponding encryption key $\mathbf{pk}^{(i)}$ will be publicly known. Given $\mathbf{pk}^{(i)}$, a plaintext x and a randomness r, anyone can compute a ciphertext $\mathsf{HE}.\mathbf{c}(x) \overset{def}{=} \mathsf{HE.Enc}_{\mathbf{pk}^{(i)}}(x, r)$ of x for P_i, using the encryption algorithm $\mathsf{HE.Enc}$, where the size of $\mathsf{HE}.\mathbf{c}(x)$ is $\mathcal{O}(\kappa)$ bits. Given a ciphertext $\mathsf{HE}.\mathbf{c}(x) = \mathsf{HE.Enc}_{\mathbf{pk}^{(i)}}(x, \star)$ and the decryption key $\mathbf{dk}^{(i)}$, P_i can recover the plaintext $x = \mathsf{HE.Dec}_{\mathbf{dk}^{(i)}}(\mathbf{c}_x)$ using the decryption algorithm $\mathsf{HE.Dec}$. The encryption scheme is assumed to be *linearly homomorphic*: given two ciphertexts $\mathsf{HE}.\mathbf{c}(x) = \mathsf{HE.Enc}_{\mathbf{pk}^{(i)}}(x, \star)$ and $\mathsf{HE}.\mathbf{c}(y) = \mathsf{HE.Enc}_{\mathbf{pk}^{(i)}}(y, \star)$, there exists an operation, say $\oplus$, such that $\mathsf{HE}.\mathbf{c}(x) \oplus \mathsf{HE}.\mathbf{c}(y) = \mathsf{HE.Enc}_{\mathbf{pk}^{(i)}}(x + y, \star)$. Moreover, given a ciphertext $\mathsf{HE}.\mathbf{c}(x) = \mathsf{HE.Enc}_{\mathbf{pk}^{(i)}}(x, \star)$ and a public constant c, there exists some operation, say $\odot$, such that $c \odot \mathsf{HE}.\mathbf{c}(x) = \mathsf{HE.Enc}_{\mathbf{pk}^{(i)}}(c \cdot x, \star)$.

Information-theoretic MACs: We will use information-theoretically secure MAC, similar to the one used in [8]. Here a random pair $\mathsf{K} = (\alpha, \beta) \in \mathbb{F}^2$ is selected as the MAC key and the MAC tag on a value $a \in \mathbb{F}$, under the key K is defined as $\mathsf{MAC}_{\mathsf{K}}(a) \overset{def}{=} \alpha \cdot a + \beta$. The MACs will be used as follows: a party P_i will hold some value a and a MAC tag $\mathsf{MAC}_{\mathsf{K}}(a)$, while party P_j will hold the MAC key K. Later when P_i wants to disclose a to P_j, it sends a along with $\mathsf{MAC}_{\mathsf{K}}(a)$; P_j verifies if a is consistent with the MAC tag with respect to its key K. A *corrupted* party P_i on holding the MAC tag on a message gets one point on the straight-line $y = \alpha x + \beta$ and it leaves one degree of freedom on the polynomial. Therefore even a computationally unbounded P_i cannot recover K completely. So a corrupted P_i cannot reveal an incorrect value $a' \neq a$ to an honest P_j without getting caught, except with probability $\frac{1}{|\mathbb{F}|} \approx 2^{-\mu} = \mathsf{negl}(\mu)$, which is the probability of guessing a second point on the straight-line. We call two MAC keys $\mathsf{K} = (\alpha, \beta)$ and $\mathsf{K}' = (\alpha', \beta')$ *consistent* if $\alpha = \alpha'$. Given two consistent MAC keys $\mathsf{K} = (\alpha, \beta)$ and $\mathsf{K}' = (\alpha, \beta')$ and a public constant c, we define the following operations on MAC keys:

$$\mathsf{K} + \mathsf{K}' \overset{def}{=} (\alpha, \beta + \beta'), \quad \mathsf{K} + c \overset{def}{=} (\alpha, \beta + \alpha c) \quad \text{and} \quad c \cdot \mathsf{K} \overset{def}{=} (\alpha, c \cdot \beta).$$

Given two consistent MAC keys K, K' and a value c, the following *linearity* properties hold for the MAC:

– **Addition:**

$$\mathsf{MAC}_{\mathsf{K}}(a) + \mathsf{MAC}_{\mathsf{K}'}(b) = \mathsf{MAC}_{\mathsf{K}+\mathsf{K}'}(a+b).$$

– **Addition/Subtraction by a Constant:**

$$\mathsf{MAC}_{\mathsf{K}-c}(a+c) = \mathsf{MAC}_{\mathsf{K}}(a) \quad \text{and} \quad \mathsf{MAC}_{\mathsf{K}+c}(a-c) = \mathsf{MAC}_{\mathsf{K}}(a).$$

– **Multiplication by a constant:**

$$c \cdot \mathsf{MAC}_{\mathsf{K}}(a) = \mathsf{MAC}_{c\cdot\mathsf{K}}(c \cdot a).$$

2.3 The Various Sharings

We define following two types of secret sharing.

Definition 1 ($[\cdot]$-sharing). *We say a value $s \in \mathbb{F}$ is $[\cdot]$-shared among $\mathcal{P}$ if there exists a polynomial $p(\cdot)$ of degree at most t with $p(0) = s$ and every (honest) party $P_i \in \mathcal{P}$ holds a share $s_i \overset{def}{=} p(i)$ of s. We denote by $[s]$ the vector of shares of s corresponding to the (honest) parties in $\mathcal{P}$. That is, $[s] = \{s_i\}_{i=1}^n$.*

Definition 2 ($\langle\cdot\rangle$-sharing). *We say that a value $s \in \mathbb{F}$ is $\langle\cdot\rangle$-shared among $\mathcal{P}$ if s is $[\cdot]$-shared among $\mathcal{P}$ and every (honest) party P_i holds a MAC tag on its share s_i for a key K_{ji} held by every P_j. That is, the following holds for every pair of (honest) parties $P_i, P_j \in \mathcal{P}$: party P_i holds MAC tag $\mathsf{MAC}_{\mathsf{K}_{ji}}(s_i)$ for a MAC key K_{ji} held by P_j. We denote by $\langle s\rangle$ the vector of such shares, MAC keys and MAC tags of s corresponding to the (honest) parties in $\mathcal{P}$. That is, $\langle s\rangle = \big\{s_i, \{\mathsf{MAC}_{\mathsf{K}_{ji}}(s_i), \mathsf{K}_{ij}\}_{j=1}^n\big\}_{i=1}^n$.*

While most of our computations are done over values that are $\langle\cdot\rangle$-shared, our efficient public reconstruction protocol for $\langle\cdot\rangle$-shared values will additionally require a tweaked version of $\langle\cdot\rangle$-sharing, where there exists some designated party, say P_j; and the parties hold the shares and the MAC tags in an encrypted form under the public key $\mathbf{pk}^{(j)}$ of an HE scheme, where P_j knows the corresponding secret key $\mathbf{dk}^{(j)}$. We stress that the shares and MAC tags will not be available in clear. More formally:

Definition 3 ($\langle\langle\cdot\rangle\rangle_j$-sharing). *Let $s \in \mathbb{F}$ and $[s] = \{s_i\}_{i=1}^n$ be the vector of shares corresponding to an $[\cdot]$-sharing of s. We say that s is $\langle\langle\cdot\rangle\rangle_j$-shared among $\mathcal{P}$ with respect to a designated party P_j, if every (honest) party P_i holds an encrypted share $\mathsf{HE}.\mathbf{c}(s_i)$ and encrypted MAC tag $\mathsf{HE}.\mathbf{c}(\mathsf{MAC}_{\mathsf{K}_{ji}}(s_i))$ under the public key $\mathbf{pk}^{(j)}$, such that P_j holds the MAC keys K_{ji} and the secret key $\mathbf{dk}^{(j)}$. We denote by $\langle\langle s\rangle\rangle_j$ the vector of encrypted shares and encrypted MAC tags corresponding to the (honest) parties in $\mathcal{P}$, along with the MAC keys and the secret key of P_j. That is, $\langle\langle s\rangle\rangle_j = \Big\{\{\mathsf{HE}.\mathbf{c}(s_i), \mathsf{HE}.\mathbf{c}(\mathsf{MAC}_{\mathsf{K}_{ji}}(s_i))\}_{i=1}^n, \{\mathsf{K}_{ji}\}_{i=1}^n, \mathbf{dk}^{(j)}\Big\}$.*

Private Reconstruction of $\langle\cdot\rangle$ and $\langle\langle\cdot\rangle\rangle$-shared Value Towards a Designated Party. Note that with $n = 2t+1$, a $[\cdot]$-shared value cannot be robustly reconstructed towards a designated party just by sending the shares, as we cannot do error-correction. However, we can robustly reconstruct a $\langle\cdot\rangle$-sharing towards a designated party, say P_j, by asking the parties to send their shares, along with MAC tags to P_j, who then identifies the correct shares with high probability and reconstructs the secret. A similar idea can be used to reconstruct an $\langle\langle s\rangle\rangle_j$-sharing towards P_j. Now the parties send encrypted shares and MAC tags to P_j, who decrypts them before doing the verification. We call the resultant protocols $\mathsf{RecPrv}(\langle s\rangle, P_j)$ and $\mathsf{RecPrvEnc}(\langle\langle s\rangle\rangle_j)$ respectively, which are presented in Fig. 1. We stress that while $\langle s\rangle$ can be reconstructed towards *any* P_j, $\langle\langle s\rangle\rangle_j$ can be reconstructed only towards P_j, as P_j alone holds the secret key $\mathbf{dk}^{(j)}$ that is required to decrypt the shares and the MAC tags.

Protocol $\mathsf{RecPrv}(\langle s\rangle, P_j)$

- Every party $P_i \in \mathcal{P}$ sends its share s_i and the MAC tag $\mathsf{MAC}_{\mathsf{K}_{ji}}(s_i)$ to the party P_j.
- Party P_j on receiving the share s'_i and the MAC tag $\mathsf{MAC}'_{\mathsf{K}_{ji}}(s_i)$ from P_i computes $\mathsf{MAC}_{\mathsf{K}_{ji}}(s'_i)$ and verifies if $\mathsf{MAC}_{\mathsf{K}_{ji}}(s'_i) \overset{?}{=} \mathsf{MAC}'_{\mathsf{K}_{ji}}(s_i)$.
- If the verification passes then P_j considers s'_i as a valid share.
- Once $t+1$ valid shares are obtained, using them P_j interpolates the sharing polynomial and outputs its constant term as s.

Protocol $\mathsf{RecPrvEnc}(\langle\langle s\rangle\rangle_j)$

- Every party $P_i \in \mathcal{P}$ sends $\mathsf{HE}.\mathbf{c}(s_i)$ and $\mathsf{HE}.\mathbf{c}(\mathsf{MAC}_{\mathsf{K}_{ji}}(s_i))$ to the party P_j.
- Party P_j, on receiving these values, computes $s'_i = \mathsf{HE.Dec}_{\mathbf{dk}^{(j)}}(\mathsf{HE}.\mathbf{c}(s_i))$ and $\mathsf{MAC}'_{\mathsf{K}_{ji}}(s_i) = \mathsf{HE.Dec}_{\mathbf{dk}^{(j)}}(\mathsf{HE}.\mathbf{c}(\mathsf{MAC}_{\mathsf{K}_{ji}}(s_i)))$. The rest of the steps are the same as for $\mathsf{RecPrv}(\star, P_j)$.

Fig. 1. Protocols for Reconstructing a $\langle\cdot\rangle$-sharing and $\langle\langle\cdot\rangle\rangle$-sharing Towards a Designated Party

It is easy to see that if P_j is honest, then P_j correctly reconstructs the shared value in protocol RecPrv as well as in $\mathsf{RecPrvEnc}$, except with probability at most $\frac{t}{|\mathbb{F}|} \approx \mathsf{negl}(\mu)$. While protocol RecPrv has communication complexity $\mathcal{O}(\mu \cdot n)$ bits, protocol $\mathsf{RecPrvEnc}$ has communication complexity $\mathcal{O}(\kappa \cdot n)$ bits. Also note that both the protocols will work in the asynchronous setting. We argue this for RecPrv (the same argument will work for $\mathsf{RecPrvEnc}$). The party P_j will eventually receive the shares of s from at least $n - t = t + 1$ honest parties, with correct MACs. These $t+1$ shares are enough for the robust reconstruction of s. So we state the following lemma for RecPrv. Similar statements hold for protocol $\mathsf{RecPrvEnc}$. Thus we have the following Lemmas.

Lemma 1. *Let s be $\langle\cdot\rangle$-shared among the parties $\mathcal{P}$. Let P_j be a specific party. Protocol* RecPrv *achieves the following in the synchronous communication setting:*

- *Correctness: Except with probability* $\mathsf{negl}(\mu)$*, an honest P_j reconstructs s.*
- *Communication Complexity: The communication complexity is $\mathcal{O}(\mu \cdot n)$ bits.*

Lemma 2. *Let s be $\langle\cdot\rangle$-shared among the parties $\mathcal{P}$. Let P_j be a specific party. Protocol* RecPrv *achieves the following in the asynchronous communication setting:*

– *Correctness & Communication Complexity: Same as in Lemma 1.*
– *Termination: If every honest party participates in* RecPrv*, then an honest P_j will eventually terminate.*

Linearity of Various Sharings. All of the previously defined secret sharings are linear, which for ease of exposition we shall now overview. We first define what is meant by key consistent sharings.

Definition 4 (Key-consistent $\langle\cdot\rangle$ and $\langle\langle\cdot\rangle\rangle_j$ Sharings). *Two $\langle\cdot\rangle$-sharings $\langle a\rangle$ and $\langle b\rangle$ are said to be key-consistent if every (honest) P_i holds consistent MAC keys for every P_j across both the sharings.*

Sharings $\langle\langle a\rangle\rangle_j$ and $\langle\langle b\rangle\rangle_j$ with respect to a designated P_j are called key-consistent if P_j holds consistent MAC keys for every P_i across both the sharings, and the encryptions are under the same public key of P_j.

Linearity of $[\cdot]$-sharings: Given $[a] = \{a_i\}_{i=1}^n$ and $[b] = \{b_i\}_{i=1}^n$ and a public constant c, we have:

– *Addition*: To compute $[a+b]$, every party P_i needs to locally compute $a_i + b_i$,

$$[a] + [b] = [a+b] = \{a_i + b_i\}_{i=1}^n .$$

– *Addition by a Public Constant*: To compute $[c+a]$, every party P_i needs to locally compute $c + a_i$,

$$c + [a] = [c+a] = \{c + a_i\}_{i=1}^n .$$

– *Multiplication by a Public Constant*: To compute $[c \cdot a]$, every party P_i needs to locally compute $c \cdot a_i$,

$$c \cdot [a] = [c \cdot a] = \{c \cdot a_i\}_{i=1}^n .$$

Linearity of $\langle\cdot\rangle$-sharing: Given $\langle a\rangle = \left\{a_i, \{\mathsf{MAC}_{\mathsf{K}_{ji}}(a_i), \mathsf{K}_{ij}\}_{j=1}^n\right\}_{i=1}^n$, and $\langle b\rangle = \left\{b_i, \{\mathsf{MAC}_{\mathsf{K}'_{ji}}(b_i), \mathsf{K}'_{ij}\}_{j=1}^n\right\}_{i=1}^n$ that are key-consistent and a publicly-known constant c, we have:

– *Addition*: To compute $\langle a+b\rangle$, every party P_i needs to locally compute $a_i + b_i$, $\{\mathsf{MAC}_{\mathsf{K}_{ji}}(a_i) + \mathsf{MAC}_{\mathsf{K}'_{ji}}(b_i)\}_{j=1}^n$ and $\{\mathsf{K}_{ij} + \mathsf{K}'_{ij}\}_{j=1}^n$,

$$\langle a\rangle + \langle b\rangle = \langle a+b\rangle = \left\{a_i + b_i, \{\mathsf{MAC}_{\mathsf{K}_{ji}+\mathsf{K}'_{ji}}(a_i + b_i), \mathsf{K}_{ij} + \mathsf{K}'_{ij}\}_{j=1}^n\right\}_{i=1}^n .$$

– *Addition by a Public Constant*: To compute $\langle c+a\rangle$, every party P_i needs to locally compute $c + a_i$, In addition recall that $\mathsf{MAC}_{\mathsf{K}_{ji}-c}(a_i + c) = \mathsf{MAC}_{\mathsf{K}_{ji}}(a_i)$. Hence we assign $\mathsf{MAC}_{\mathsf{K}_{ji}}(a_i)$ to $\mathsf{MAC}_{\mathsf{K}_{ji}-c}(a_i + c)$ and compute $\{\mathsf{K}_{ij} - c\}_{j=1}^n$.

$$c + \langle a\rangle = \langle c+a\rangle = \left\{c + a_i, \{\mathsf{MAC}_{\mathsf{K}_{ji}-c}(a_i + c), \mathsf{K}_{ij} - c\}_{j=1}^n\right\}_{i=1}^n .$$

– *Multiplication by a Public Constant*: To compute $\langle c \cdot a \rangle$, every party P_i needs to locally compute $c \cdot a_i$, $\{c \cdot \mathsf{MAC}_{\cdot \mathsf{K}_{ji}}(a_i)\}_{j=1}^n$ and $\{c \cdot \mathsf{K}_{ij}\}_{j=1}^n$,

$$c \cdot \langle a \rangle = \langle c \cdot a \rangle = \left\{c \cdot a_i, \{\mathsf{MAC}_{c \cdot \mathsf{K}_{ji}}(c \cdot a_i), c \cdot \mathsf{K}_{ij}\}_{j=1}^n\right\}_{i=1}^n.$$

<u>*Linearity of $\langle\langle \cdot \rangle\rangle_j$-sharings:*</u> Given $\langle\langle a \rangle\rangle_j = \left\{\{\mathsf{HE}.\mathbf{c}(a_i), \mathsf{HE}.\mathbf{c}(\mathsf{MAC}_{\mathsf{K}_{ji}}(a_i)), \}_{i=1}^n, \{\mathsf{K}_{ji}\}_{i=1}^n, \mathbf{dk}^{(j)}\right\}$ and $\langle\langle b \rangle\rangle_j = \left\{\{\mathsf{HE}.\mathbf{c}(b_i), \mathsf{HE}.\mathbf{c}(\mathsf{MAC}_{\mathsf{K}_{ji}}(b_i))\}_{i=1}^n, \{\mathsf{K}'_{ji}\}_{i=1}^n, \mathbf{dk}^{(j)}\right\}$ that are key-consistent we can add the sharings via the operation

$$\begin{aligned}\langle\langle a \rangle\rangle_j + \langle\langle b \rangle\rangle_j &= \langle\langle a + b \rangle\rangle_j \\ &= \Big\{\{\mathsf{HE}.\mathbf{c}(a_i + b_i), \mathsf{HE}.\mathbf{c}(\mathsf{MAC}_{\mathsf{K}_{ji}+\mathsf{K}'_{ji}}(a_i + b_i))\}_{i=1}^n, \\ &\qquad \{\mathsf{K}_{ji} + \mathsf{K}'_{ji}\}_{i=1}^n, \mathbf{dk}^{(j)}\Big\}\end{aligned}$$

So to compute $\langle\langle a + b \rangle\rangle_j$, every party $P_i \in \mathcal{P}$ needs to locally compute the values $\mathsf{HE}.\mathbf{c}(a_i) \oplus \mathsf{HE}.\mathbf{c}(b_i)$ and $\mathsf{HE}.\mathbf{c}(\mathsf{MAC}_{\mathsf{K}_{ji}}(a_i)) \oplus \mathsf{HE}.\mathbf{c}(\mathsf{MAC}_{\mathsf{K}'_{ji}}(b_i))$, while party P_j needs to compute $\{\mathsf{K}_{ji} + \mathsf{K}'_{ji}\}_{i=1}^n$.

Generating $\langle\langle \cdot \rangle\rangle_j$-sharing from $\langle \cdot \rangle$-sharing. In our efficient protocol for public reconstruction of $\langle \cdot \rangle$-shared values, we come across the situation where there exists: a value r known only to a designated party P_j, a publicly known encryption $\mathsf{HE}.\mathbf{c}(r)$ of r, under the public key $\mathbf{pk}^{(j)}$, and a $\langle \cdot \rangle$-sharing $\langle a \rangle = \left\{a_i, \{\mathsf{MAC}_{\mathsf{K}_{ji}}(a_i), \mathsf{K}_{ij}\}_{j=1}^n\right\}_{i=1}^n$. Given the above, the parties need to compute a $\langle\langle \cdot \rangle\rangle_j$ sharing:

$$\langle\langle r \cdot a \rangle\rangle_j = \left\{\{\mathsf{HE}.\mathbf{c}(r \cdot a_i), \mathsf{HE}.\mathbf{c}(\mathsf{MAC}_{r \cdot \mathsf{K}_{ji}}(r \cdot a_i))\}_{i=1}^n, \{r \cdot \mathsf{K}_{ji}\}_{i=1}^n, \mathbf{dk}^{(j)}\right\}$$

of $r \cdot a$. Computing the above needs only local computation by the parties. Specifically, each party $P_i \in \mathcal{P}$ locally computes the values $\mathsf{HE}.\mathbf{c}(r \cdot a_i) = a_i \odot \mathsf{HE}.\mathbf{c}(r)$ and

$$\mathsf{HE}.\mathbf{c}(\mathsf{MAC}_{r \cdot \mathsf{K}_{ji}}(r \cdot a_i)) = \mathsf{HE}.\mathbf{c}(r \cdot \mathsf{MAC}_{\mathsf{K}_{ji}}(a_i)) = \mathsf{MAC}_{\mathsf{K}_{ji}}(a_i) \odot \mathsf{HE}.\mathbf{c}(r),$$

since $r \cdot \mathsf{MAC}_{\mathsf{K}_{ji}}(a_i) = \mathsf{MAC}_{r \cdot \mathsf{K}_{ji}}(a_i \cdot r)$. Finally party P_j locally computes $\{r \cdot \mathsf{K}_{ji}\}_{i=1}^n$.

3 Public Reconstruction of $\langle \cdot \rangle$-sharings with a Linear Overhead

We present a new protocol to publicly reconstruct $n(t+1)\frac{\kappa}{\mu} = \Theta(\frac{n^2\kappa}{\mu})$ $\langle \cdot \rangle$-shared values with communication complexity $\mathcal{O}(\kappa \cdot n^3)$ bits. So the amortized communication overhead for public reconstruction of one $\langle \cdot \rangle$-shared value is linear in n i.e. $\mathcal{O}(\mu \cdot n)$ bits. For a better understanding of the ideas used in the protocol, we first present a protocol $\mathsf{RecPubSimple}$ to publicly reconstruct $n(t+1)$ $\langle \cdot \rangle$-shared values with communication complexity $\mathcal{O}(\kappa \cdot n^3)$ bits. We will then extend this protocol for $n(t+1)\frac{\kappa}{\mu}$ secrets while retaining the same communication complexity; the resulting protocol is called RecPub.

Let $\{\langle a^{(i,j)} \rangle\}_{i=1,j=1}^{n,t+1}$ be the $\langle \cdot \rangle$-sharings, which need to be publicly reconstructed. The naive way of achieving the task is to run $\Theta(n^3)$ instances of RecPrv, where $\Theta(n^2)$

instances are run to reconstruct all the values to a single party. This method has communication complexity $\mathcal{O}(\kappa \cdot n^4)$ bits and thus has a quadratic overhead. Our approach outperforms the naive method, and works for both synchronous as well as asynchronous setting; for simplicity we first explain the protocol assuming a synchronous setting.

Let A be an $n \times (t+1)$ matrix, with (i,j)th element as $a^{(i,j)}$. Let $A_i(x)$ be a polynomial of degree t defined over the values in the ith row of A; i.e. $A_i(x) \overset{def}{=} A[i,1]+A[i,2]x+\ldots, A[i,t+1]x^t$. Let B denote an $n \times n$ matrix and $B[i,j] \overset{def}{=} A_i(j)$, for $i,j \in \{1,\ldots,n\}$. Clearly A can be recovered given any $t+1$ columns of B. We explain below how to reconstruct at least $t+1$ columns of B to all the parties with communication complexity $\mathcal{O}(\kappa \cdot n^3)$ bits. In what follows, we denote ith row and column of A as A_i and A^i respectively, with a similar notation used for the rows and columns of B.

Since B_i is linearly dependent on A_i, given $\langle\cdot\rangle$-sharing of A_i, it requires only local computation to generate $\langle\cdot\rangle$-sharings of the elements in B_i. Specifically, $\langle B[i,j]\rangle = \langle A[i,1]\rangle + \langle A[i,2]\rangle \cdot j + \ldots + \langle A[i,t+1]\rangle \cdot j^t$. Then we reconstruct the elements of A to all the parties in two steps. First B^i is reconstructed towards P_i using n instances of RecPrv with an overall cost $\mathcal{O}(\mu \cdot n^3)$ bits. Next each party P_i sends B^i to all the parties, requiring $\mathcal{O}(\mu \cdot n^3)$ bits of communication. If every P_i behaves honestly then every party would possess B at the end of the second step. However a corrupted P_i may not send the correct B^i. So what we need is a mechanism that allows an honest party to detect if a corrupted party P_i has sent an incorrect B^i. Detecting is enough, since every (honest) party is guaranteed to receive correctly the B^i columns from $t+1$ honest parties. Recall that $t+1$ correct columns of B are enough to reconstruct A.

After P_i reconstructs B^i, and before it sends the same to party P_j, we allow P_j to obtain a random linear combination of the elements in B^i (via interaction) in a way that the linear combiners are known to no one other than P_j. Later, when P_i sends B^i to P_j, party P_j can verify if the B^i received from P_i is correct or not by comparing the linear combination of the elements of the received B^i with the linear combination that it obtained before. It is crucial to pick the linear combiners randomly and keep them secret, otherwise P_i can cheat with an incorrect B^i without being detected by an honest P_j. In our method, the random combiners for an honest P_j are never leaked to anyone and this allows P_j to reuse them in a latter instance of the public reconstruction protocol. Specifically, we assume the following *one time setup* for $\mathsf{RecPubSimple}$ (which can be done beforehand in the offline phase of the main MPC protocol). Every party P_j holds a secret key $\mathbf{dk}^{(j)}$ for the linearly-homomorphic encryption scheme HE and the corresponding public key $\mathbf{pk}^{(j)}$ is publicly available. In addition, P_j holds a vector R^j of n random combiners and the encryptions $\mathsf{HE}.\mathbf{c}(R^j[1]),\ldots,\mathsf{HE}.\mathbf{c}(R^j[n])$ of the values in R^j under P_j's public key $\mathbf{pk}^{(j)}$ are available publicly. The above setup can be created once and for all, and can be reused across multiple instances of $\mathsf{RecPubSimple}$.

Given the above random combiners in an encrypted form, party P_j can obtain the linear combination $c^{(i,j)} \overset{def}{=} \sum_{l=1}^{n} B^i[l]R^j[l]$ of the elements of B^i as follows. First note that the parties hold $\langle\cdot\rangle$-sharing of the elements of B^i. If the linear combiners were publicly known, then the parties could compute $\langle c^{(i,j)}\rangle = \sum_{l=1}^{n} R^j[l]\langle B^i[l]\rangle$ and reconstruct $c^{(i,j)}$ to party P_j using RecPrv. However since we do *not* want to disclose the combiners, the above task is performed in an encrypted form, which is doable since the combiners are encrypted under the linearly-homomorphic PKE. Specifically, given encryptions $\mathsf{HE}.\mathbf{c}(R^j[l])$ under $\mathbf{pk}^{(j)}$ and sharings $\langle B^i[l] \rangle$, the parties first generate $\langle\langle R^j[l] \cdot B^i[l] \rangle\rangle_j$ for every P_j (recall that it requires only local computation). Next the parties linearly combine the sharings $\langle\langle R^j[l] \cdot B^i[l] \rangle\rangle_j$ for $l = 1, \ldots, n$ to obtain $\langle\langle c^{(i,j)} \rangle\rangle_j$, which is then reconstructed towards party P_j using an instance of $\mathsf{RecPrvEnc}$. In total n^2 such instances need to be executed, costing $\mathcal{O}(\kappa \cdot n^3)$ bits. Protocol $\mathsf{RecPubSimple}$ is presented in Fig. 2.

Protocol $\mathsf{RecPubSimple}(\{\langle a^{(i,j)} \rangle\}_{i=1,j=1}^{n,t+1})$

Each $P_j \in \mathcal{P}$ holds R^j and the encryptions $\mathsf{HE}.\mathbf{c}(R^j[1]), \ldots, \mathsf{HE}.\mathbf{c}(R^j[n])$, under P_j's public key $\mathbf{pk}^{(j)}$, are publicly known. Let A be the matrix of size $n \times (t+1)$, with (i,j)th entry as $a^{(i,j)}$, for $i \in \{1, \ldots, n\}$ and $j \in \{1, \ldots, t+1\}$. We denote the ith row and column of A as A_i and A^i respectively. Let $A_i(x) \overset{def}{=} a^{(i,1)} + \ldots + a^{(i,t+1)}x^t$ for $i \in \{1, \ldots, n\}$. Let B be the matrix of size $n \times n$, with the (i,j)th entry as $B[i,j] \overset{def}{=} A_i(j)$ for $i, j \in \{1, \ldots, n\}$. We denote the ith row and column of B as B_i and B^i respectively. The parties do the following to reconstruct A:

- **Computing $\langle \cdot \rangle$-sharing of every element of B:** For $i, j \in \{1, \ldots, n\}$, the parties compute $\langle B[i,j] \rangle = \langle A[i,1] \rangle + j \cdot \langle A[i,1] \rangle + \ldots + j^t \cdot \langle A[i,t+1] \rangle$.
- **Reconstructing B^i towards P_i:** For $i \in \{1, \ldots, n\}$, the parties execute $\mathsf{RecPrv}(\langle B[1,i] \rangle, P_i), \ldots, \mathsf{RecPrv}(\langle B[n,i] \rangle, P_i)$ to enable P_i robustly reconstruct B^i.
- **Reconstructing $B^i \otimes R^j$ towards P_j:** Corresponding to each $P_i \in \mathcal{P}$, the parties execute the following steps, to enable each $P_j \in \mathcal{P}$ to obtain the random linear combination $c^{(i,j)} \overset{def}{=} B^i \otimes R^j$:
  - The parties first compute $\langle\langle R^j[l] \cdot B^i[l] \rangle\rangle_j$ from $\mathsf{HE}.\mathbf{c}(R^j[l])$ and $\langle B^i[l] \rangle$ for $l \in \{1, \ldots, n\}$ and then compute $\langle\langle c^{(i,j)} \rangle\rangle_j = \sum_{l=1}^{n} \langle\langle R^j[l] \cdot B^i[l] \rangle\rangle_j$.
  - The parties execute $\mathsf{RecPrvEnc}(\langle\langle c^{(i,j)} \rangle\rangle_j)$ to reconstruct $c^{(i,j)}$ towards P_j.
- **Sending B^i to all:** Each $P_i \in \mathcal{P}$ sends B^i to every $P_j \in \mathcal{P}$. Each P_j then reconstructs A as follows:
  - On receiving $\bar{B}^i$ from P_i, compute $c'^{(i,j)} = \bar{B}^i \otimes R^j$ and check if $c^{(i,j)} \overset{?}{=} c'^{(i,j)}$. If the test passes then P_j considers $\bar{B}^i$ as the valid i^{th} column of the matrix B.
  - Once $t+1$ valid columns of B are obtained by P_j, it then reconstructs A.

Fig. 2. Robustly Reconstructing $\langle \cdot \rangle$-shared Values with $\mathcal{O}(\kappa \cdot n)$ Communication Complexity

The correctness and communication complexity of the protocol are stated in Lemma 3, which follows in a straight forward fashion from the protocol description and the detailed protocol overview. The security of the protocol will be proven, in the full version, in conjunction with the online phase of our MPC protocol.

Lemma 3. *Let* $\{\langle a^{(i,j)} \rangle\}_{i=1,j=1}^{n,t+1}$ *be a set of* $n(t+1)$ *shared values which need to be publicly reconstructed by the parties. Then given a setup* $(\mathbf{pk}^{(1)}, \mathbf{dk}^{(1)}), \ldots, (\mathbf{pk}^{(n)}, \mathbf{dk}^{(n)})$ *for the linearly-homomorphic encryption scheme* HE *for the* n *parties and encryptions* $\mathsf{HE}.\mathbf{c}(R^j[1]), \ldots, \mathsf{HE}.\mathbf{c}(R^j[n])$ *of* n *random values in* R^j *on the behalf of* *each party $P_j \in \mathcal{P}$, with only P_j knowing the random values, protocol* RecPubSimple *achieves the following in the synchronous communication setting:*

– *Correctness: Except with probability* $\mathsf{negl}(\kappa, \mu)$*, every honest party reconstructs* $\{a^{(i,j)}\}_{i=1,j=1}^{n,t+1}$.
– *Communication Complexity: The communication complexity is* $\mathcal{O}(\kappa \cdot n^3)$ *bits.*

From $\mathcal{O}(\kappa \cdot n)$ to $\mathcal{O}(\mu \cdot n)$ Amortized cost of Reconstruction. We note that the amortized complexity of reconstructing one secret via RecPubSimple is $\mathcal{O}(\kappa \cdot n)$, where κ is the cryptographic security parameter. To improve the amortized cost to $\mathcal{O}(\mu \cdot n)$, we make the following observation on the communication in RecPubSimple. There is a scope to amortize part of the communication to reconstruct more than $n(t+1)$ secrets. This leads to a trick that brings down the amortized communication complexity per secret to $\mathcal{O}(\mu \cdot n)$ bits. We call our new protocol RecPub. which starts with $\frac{\kappa}{\mu}$ batches of secrets where each batch consists of $n(t+1)$ secrets. For each batch, RecPub executes exactly the same steps as done in RecPubSimple except for the step involving the reconstruction of $B^i \otimes R^j$. RecPub keeps the communication cost of this step unperturbed by taking a random linear combination of $\frac{\kappa}{\mu}$ B^i columns together. Therefore RecPub still needs private reconstruction of n^2 $\langle\langle \cdot \rangle\rangle_j$-shared values and a communication of $\mathcal{O}(\kappa \cdot n^3)$ bits for this step. For the rest of the steps, the communication complexity of RecPub will be $\frac{\kappa}{\mu}$ times the communication complexity of the same steps in RecPubSimple. Since RecPubSimple requires $\mathcal{O}(\mu \cdot n^3)$ bits of communication for the rest of the steps, the communication complexity of RecPub will turn out to be $\mathcal{O}(\kappa \cdot n^3)$ bits of communication overall. Since the number reconstructed secrets are $n(t+1)\frac{\kappa}{\mu}$, RecPub offers an amortized cost of $\mathcal{O}(\mu \cdot n)$ bits per secret. The formal specification of protocol RecPub is in Fig. 3.

We note that RecPub takes random linear combination of $\frac{\kappa n}{\mu}$ values. So the one time set up has to be enhanced where every P_j now holds $\frac{\kappa n}{\mu}$ random combiners, and the encryptions of them are available in public. Namely R^j is a vector of $\kappa n/\mu$ random values and the encryptions $\mathsf{HE}.\mathbf{c}(R^j[1]), \ldots, \mathsf{HE}.\mathbf{c}(R^j[\kappa n/\mu])$ done under P_j's public key $\mathbf{pk}^{(j)}$ are available publicly. We thus have the following Lemma.

Lemma 4. *Let* $\{\langle a^{(i,j,k)} \rangle\}_{i=1,j=1,k=1}^{n,t+1,\kappa/\mu}$ *be a set of* $n(t+1)\frac{\kappa}{\mu}$ *shared values which need to be publicly reconstructed by the parties. Then given a setup* $(\mathbf{pk}^{(1)}, \mathbf{dk}^{(1)}), \ldots,$ $(\mathbf{pk}^{(n)}, \mathbf{dk}^{(n)})$ *for the linearly-homomorphic encryption scheme* HE *for the* n *parties and encryptions* $\mathsf{HE}.\mathbf{c}(R^j[1]), \ldots, \mathsf{HE}.\mathbf{c}(R^j[\kappa n/\mu])$ *of* $\kappa n/\mu$ *random values in* R^j *on the behalf of each party* $P_j \in \mathcal{P}$*, with only* P_j *knowing the random values, protocol* RecPub *achieves the following in the synchronous communication setting:*

– *Correctness: Except with probability* $\mathsf{negl}(\kappa, \mu)$*, every honest party reconstructs* $\{a^{(i,j,k)}\}_{i=1,j=1,k=1}^{n,t+1,\kappa/\mu}$.
– *Communication Complexity: The communication complexity is* $\mathcal{O}(\kappa \cdot n^3)$ *bits.*

Protocol RecPubSimple **and** RecPub **in the Asynchronous Setting:** Consider the protocol RecPubSimple, and note that the steps involving interaction among the parties are during the instances of RecPrv and RecPrvEnc. All the remaining steps involve only

Protocol $\mathsf{RecPub}(\{\langle a^{(i,j,k)}\rangle\}_{i=1,j=1,k=1}^{n,t+1,\kappa/\mu})$

Each $P_j \in \mathcal{P}$ holds $R^j = (R^j[1], \ldots, R^j[\kappa n/\mu])$ and the encryptions $\mathsf{HE}.\mathbf{c}(R^j[1]), \ldots, \mathsf{HE}.\mathbf{c}(R^j[\kappa n/\mu])$, under P_j's public key $\mathbf{pk}^{(j)}$, are publicly known. For a k that varies over $1, \ldots, \kappa/\mu$, the set of secrets $\{a^{(i,j,k)}\}_{i=1,j=1}^{n,t+1}$ is denoted as the kth batch of secrets. Let $\mathcal{A}^{(k)}$ be the matrix of size $n \times (t+1)$ consisting of the kth batch i.e., the (i,j)th entry of $\mathcal{A}^{(k)}$ is $a^{(i,j,k)}$, for $i \in \{1, \ldots, n\}$ and $j \in \{1, \ldots, t+1\}$. We denote the ith row and column of $\mathcal{A}^{(k)}$ as $\mathcal{A}_i^{(k)}$ and $\mathcal{A}^{(k)^i}$ respectively. We define $\mathcal{A}_i^{(k)}(x) \overset{def}{=} \mathcal{A}^{(k)}[i,1] + \mathcal{A}^{(k)}[i,2] \cdot x + \ldots + \mathcal{A}^{(k)}[i,t+1] \cdot x^t$ for $i \in \{1, \ldots, n\}$. Let $\mathcal{B}^{(k)}$ be the matrix of size $n \times n$, with the (i,j)th entry as $\mathcal{B}^{(k)}[i,j] \overset{def}{=} \mathcal{A}_i^{(k)}(j)$ for $i,j \in \{1, \ldots, n\}$. We denote the ith row and column of $\mathcal{B}^{(k)}$ as $\mathcal{B}_i^{(k)}$ and $\mathcal{B}^{(k)^i}$ respectively. We denote the concatenation of the ith column of all the $\mathcal{B}^{(k)}$s as B^i i.e. $B^i = \left[(\mathcal{B}^{(1)^i})^{tr}, \ldots, (\mathcal{B}^{(\frac{\kappa}{\mu})^i})^{tr}\right]$ where $(\cdot)^{tr}$ denotes vector transpose. The parties do the following to reconstruct $\mathcal{A}^{(1)}, \ldots, \mathcal{A}^{(\frac{\kappa}{\mu})}$:

- **Computing $\langle\cdot\rangle$-sharing of elements of $\mathcal{B}^{(k)}$ for $k = 1, \ldots, \kappa/\mu$:** Same as in $\mathsf{RecPubSimple}$.
- **Reconstructing $\mathcal{B}^{(k)^i}$ towards P_i for $k = 1, \ldots, \kappa/\mu$:** Same as in $\mathsf{RecPubSimple}$. Party P_i holds B^i at the end of this step.
- **Reconstructing $B^i \otimes R^j$ towards P_j:** Corresponding to each $P_i \in \mathcal{P}$, the parties execute the following steps, to enable each $P_j \in \mathcal{P}$ to obtain the random linear combination $c^{(i,j)} \overset{def}{=} B^i \otimes R^j = \sum_{l=1}^{\kappa n/\mu} R^j[l]B^i[l]$:
  - The parties first compute $\langle\langle R^j[l]B^i[l]\rangle\rangle_j$ from $\mathsf{HE}.\mathbf{c}(R^j[l])$ and $\langle B^i[l]\rangle$ for $l \in \{1, \ldots, \frac{\kappa n}{\mu}\}$ and then compute $\langle\langle c^{(i,j)}\rangle\rangle_j = \sum_{l=1}^{\frac{\kappa n}{\mu}} \langle\langle R^j[l]B^i[l]\rangle\rangle_j$.
  - The parties execute $\mathsf{RecPrvEnc}(\langle\langle c^{(i,j)}\rangle\rangle_j)$ to reconstruct $c^{(i,j)}$ towards P_j.
- **Sending B^i to all:** Every party $P_i \in \mathcal{P}$ sends B^i to every party $P_j \in \mathcal{P}$. Each party P_j then reconstructs $\mathcal{A}^{(1)}, \ldots, \mathcal{A}^{(\frac{\kappa}{\mu})}$ as follows:
  - On receiving $\bar{B}^i$ from P_i, compute $c'^{(i,j)} = \bar{B}^i \otimes R^j$ and check if $c^{(i,j)} \overset{?}{=} c'^{(i,j)}$. If the test passes then P_j interprets $\bar{B}^i$ as $\left[(\mathcal{B}^{(1)^i})^{tr}, \ldots, (\mathcal{B}^{(\frac{\kappa}{\mu})^i})^{tr}\right]$ and considers $\mathcal{B}^{(k)^i}$ as the valid i^{th} column of the matrix $\mathcal{B}^{(k)}$ for $k = 1, \ldots, \kappa/\mu$.
  - For $k = 1, \ldots, \kappa/\mu$, once $t+1$ valid columns of $\mathcal{B}^{(k)}$ are obtained by P_j, it then reconstructs $\mathcal{A}^{(k)}$.

Fig. 3. Robustly Reconstructing $\langle\cdot\rangle$-shared Values with $\mathcal{O}(\mu \cdot n)$ Communication Complexity

local computation by the parties. As the instances of RecPrv and $\mathsf{RecPrvEnc}$ eventually terminate for each honest party, it follows that $\mathsf{RecPubSimple}$ eventually terminates for each honest party in the asynchronous setting. Similar arguments hold for RecPub, so we get the following lemma.

Lemma 5. *Protocol* RecPub *achieves the following in the asynchronous communication setting:*

– *Correctness & Communication Complexity: Same as in Lemma 4*
– *Termination: Every honest party eventually terminates the protocol.*

4 Linear Overhead Online Phase Protocol

Let $f : \mathbb{F}^n \rightarrow \mathbb{F}$ be a publicly known function over $\mathbb{F}$, represented as an arithmetic circuit C over $\mathbb{F}$, consisting of M multiplication gates. Then using our efficient reconstruction protocol RecPub enables one to securely realize the standard ideal functionality $\mathcal{F}_f$ (see Fig. 4 for an explicit functionality) for the MPC evaluation of the circuit C, in the $\mathcal{F}_{\text{PREP}}$-hybrid model, with communication complexity $\mathcal{O}(\mu \cdot (n \cdot M + n^2))$ bits, thus providing a linear overhead per multiplication gate. More specifically, assume that the parties have access to an ideal pre-processing and input processing functionality $\mathcal{F}_{\text{PREP}}$, which creates the following *one-time* setup: **(i)** Every P_j holds a secret key $\mathbf{dk}^{(j)}$ for the linearly-homomorphic encryption scheme HE and the corresponding public key $\mathbf{pk}^{(j)}$ is available publicly. In addition, each P_j holds n random combiners $R^{(j)} = (r^{(j,1)}, \ldots, r^{(j,n)})$ and the encryptions $\mathsf{HE}.\mathbf{c}(r^{(j,1)}), \ldots, \mathsf{HE}.\mathbf{c}(r^{(j,n)})$ of these values under P_j's public key are publicly available. **(ii)** Each P_i holds α_{ij}, the α-component of all its keys for party P_j (recall that for key-consistent sharings every P_i has to use the same α-component for all its keys corresponding to P_j). The above setup can be reused across multiple instances of Π_{ONLINE} and can be created once and for all. In addition to the one-time setup, the functionality also creates at least M random $\langle\cdot\rangle$-shared multiplications triples (these are not reusable and have to be created afresh for every execution of Π_{ONLINE}) and $\langle\cdot\rangle$-shared inputs of the parties. Functionality $\mathcal{F}_{\text{PREP}}$ is presented in Fig. 5. In $\mathcal{F}_{\text{PREP}}$, the ideal adversary specifies all the data that the corrupted parties would like to hold as part of the various sharings generated by the functionality. Namely it specifies the shares, MAC keys and MAC tags. The functionality then completes the sharings while keeping them consistent with the data specified by the adversary.

Functionality $\mathcal{F}_f$

$\mathcal{F}_f$ interacts with the parties $P_1, \ldots, P_n$ and the adversary $\mathcal{S}$ and is parametrized by an n-input function $f : \mathbb{F}^n \rightarrow \mathbb{F}$ represented as an arithmetic circuit C.

– Upon receiving $(i, x^{(i)})$ from every party $P_i \in \mathcal{P}$ where $x^{(i)} \in \mathbb{F}$, the functionality computes $y = C(x^{(1)}, \ldots, x^{(n)})$, sends y to all the parties and the adversary $\mathcal{S}$ and halts. Here C denotes the arithmetic circuit over $\mathbb{F}$ representing f.

Fig. 4. The Ideal Functionality for Computing a Given Function

Using $\mathcal{F}_{\text{PREP}}$ we design a protocol Π_{ONLINE} (see Fig. 6) which realizes $\mathcal{F}_f$ in the synchronous setting and provides *universal composability* (UC) security [11,8,19,14]. The protocol is based on the standard Beaver's idea of securely evaluating the circuit in a shared fashion using pre-processed shared random multiplication triples [2] and shared inputs. Namely, the parties evaluate the circuit C in a $\langle\cdot\rangle$-shared fashion by maintaining the following invariant for each gate in the circuit. Given a $\langle\cdot\rangle$-sharing of the inputs of the gate, the parties generate an $\langle\cdot\rangle$-sharing of the output of the gate. Maintaining the invariant for linear gates requires only local computation, thanks to the linearity property of the $\langle\cdot\rangle$-sharing. For multiplication gates, the parties deploy a

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

The functionality interacts with the parties in $\mathcal{P}$ and the adversary $\mathcal{S}$ as follows. Let $\mathcal{C} \subset \mathcal{P}$ be the set of corrupted parties.

- **Setup Generation:** On input Setup from the parties in $\mathcal{P}$, the functionality does the following:
  - It creates n public key, secret key pairs $\{\mathbf{pk}^{(i)}, \mathbf{dk}^{(i)}\}_{i=1}^{n}$ of the linearly-homomorphic encryption scheme HE,
  - For each P_i, it selects $\frac{\kappa n}{\mu}$ random values $R^i = (r^{(i,1)}, \ldots, r^{(i,\frac{\kappa n}{\mu})})$, computes $\mathsf{HE}.\mathbf{c}(r^{(i,1)}), \ldots, \mathsf{HE}.\mathbf{c}(r^{(i,\frac{\kappa n}{\mu})})$ under $\mathbf{pk}^{(i)}$,
  - To party P_i it sends
  $$\Big(\mathbf{dk}^{(i)}, (r^{(i,1)}, \ldots, r^{(i,\frac{\kappa n}{\mu})}), \{\mathbf{pk}^{(j)}\}_{j=1}^{n}, \{\mathsf{HE}.\mathbf{c}(r^{(j,1)}), \ldots, \mathsf{HE}.\mathbf{c}(r^{(j,\frac{\kappa n}{\mu})})\}_{j=1}^{n}\Big).$$
  - On the behalf of each honest $P_i \in \mathcal{P} \setminus \mathcal{C}$, it selects n random values $\{\alpha_{ij}\}_{j=1}^{n}$, where the jth value is designated to be used in the MAC key for party P_j. On the behalf of each corrupted party $P_i \in \mathcal{C}$, it receives from $\mathcal{S}$ the α_{ij} values that P_i wants to use in the MAC keys corresponding to the honest party P_j. On receiving, the functionality stores these values.
- **Triple Sharings**: On input $\mathsf{Triples}$ from all the parties in $\mathcal{P}$, the functionality generates $\langle\cdot\rangle$-sharing of χ random multiplication triples in parallel. To generate one such sharing $(\langle a\rangle, \langle b\rangle, \langle c\rangle)$, it does the following:
  - It randomly selects a, b and computes $c = ab$. It then runs 'Single $\langle\cdot\rangle$-sharing Generation' (see below) for a, b and c.
- **Input Sharings**: On input $(x^{(i)}, i, \mathsf{Input})$ from party P_i and (i, Input) from the remaining parties, the functionality runs 'Single $\langle\cdot\rangle$-sharing Generation' (given below) for $x^{(i)}$.

Single $\langle\cdot\rangle$-sharing Generation: The functionality does the following to generate $\langle s\rangle$-sharing for a given value s:

- On receiving the shares $\{s_i\}_{P_i \in \mathcal{C}}$ from $\mathcal{S}$ on the behalf of the corrupted parties, it selects a polynomial $S(\cdot)$ of degree at most t, such that $S(0) = s$ and $S(i) = s_i$ for each $P_i \in \mathcal{C}$. For $P_i \notin \mathcal{P} \setminus \mathcal{C}$, it computes $s_i = S(i)$.
- On receiving $\{\beta_{ij}\}_{P_i \in \mathcal{C},\, P_j \notin \mathcal{C}}$ from $\mathcal{S}$, the second components of the MAC key that $P_i \in \mathcal{C}$ will have for an honest party P_j, it sets $\mathsf{K}_{ij} = (\alpha_{ij}, \beta_{ij})$ where α_{ij} was specified by $\mathcal{S}$ in 'Setup generation' stage. It computes the MAC tag $\mathsf{MAC}_{\mathsf{K}_{ij}}(s_j)$ of s_j for every honest P_j corresponding to the key of every corrupted P_i.
- On receiving MAC tags $\{\mathsf{MAC}_{ij}\}_{P_i \in \mathcal{C},\, P_j \notin \mathcal{C}}$ that the corrupted parties would like to have on their shares s_i corresponding to the MAC key of honest P_j, it fixes the key of P_j corresponding to P_i as $\mathsf{K}_{ji} = (\alpha_{ji}, \beta_{ji})$ where $\beta_{ji} = \mathsf{MAC}_{ij} - \alpha_{ji} \cdot s_i$ and α_{ji} was selected by the functionality in 'Setup generation' stage.
- For every pair of honest parties (P_j, P_k), it chooses the key of P_j as $\mathsf{K}_{jk} = (\alpha_{jk}, \beta_{jk})$ where α_{jk} is taken from 'setup generation phase' and β_{jk} is chosen randomly. It then computes the corresponding MAC tag of P_k as $\mathsf{MAC}_{\mathsf{K}_{jk}}(s_k)$.
- It sends $\big\{s_j, \{\mathsf{MAC}_{\mathsf{K}_{kj}}, \mathsf{K}_{jk}\}_{k=1}^{n}\big\}$ to honest party P_j (no need to send anything to corrupted parties as $\mathcal{S}$ has the data of the corrupted parties already).

Fig. 5. Ideal Functionality for Setup Generation, Offline Pre-processing and Input Processing

shared multiplication triple received from $\mathcal{F}_{\text{PREP}}$ and evaluate the multiplication gate by using Beaver's trick. Specifically, let $\langle p \rangle, \langle q \rangle$ be the sharing corresponding to the inputs of a multiplication gate and let $(\langle a \rangle, \langle b \rangle, \langle c \rangle)$ be the shared random multiplication triple obtained from $\mathcal{F}_{\text{PREP}}$, which is associated with this multiplication gate. To compute an $\langle \cdot \rangle$-sharing of the gate output $p \cdot q$, we note that $p \cdot q = (p - a + a) \cdot (q - b + b) = d \cdot e + d \cdot b + e \cdot a + c$, where $d \overset{def}{=} p - a$ and $e \overset{def}{=} q - b$. So if d and e are publicly known then $\langle p \cdot q \rangle = d \cdot e + d \cdot \langle b \rangle + e \cdot \langle a \rangle + \langle c \rangle$ holds. To make d and e public, the parties first locally compute $\langle d \rangle = \langle p \rangle - \langle a \rangle$ and $\langle e \rangle = \langle q \rangle - \langle b \rangle$ and publicly reconstruct these sharings. Note that even though d and e are made public, the privacy of the gate inputs p and q is preserved, as a and b are random and private. Finally once the parties have the sharing $\langle y \rangle$ for the circuit output, it is publicly reconstructed to enable every party obtain the function output.

To achieve the linear overhead in Π_{ONLINE}, we require that the circuit is "wide" in the sense that at every level there are at least $n(t+1)\frac{\kappa}{\mu}$ independent multiplication gates that can be evaluated in parallel. This is to ensure that we can use our linear-overhead reconstruction protocol RecPub. We note that similar restrictions are used in some of the previous MPC protocols to achieve a linear overhead in the online phase. For example, [18,4,13] requires $\Theta(n)$ independent multiplication gates at each level to ensure that they can use their linear-overhead reconstruction protocol to evaluate these gates. In practice many functions have such a level of parallel multiplication gates when expressed in arithmetic circuit format, and practical systems use algorithms to maximise the level of such parallelism in their execution, see e.g. [27].

The properties of Π_{ONLINE} are stated in Theorem 1, which is proved in the full version. In the protocol, $2M$ $\langle \cdot \rangle$-shared values are publicly reconstructed via RecPub while evaluating the multiplication gates. Assuming that the M multiplication gates can be divided into blocks of $n(t+1)\frac{\kappa}{\mu}$ independent multiplication gates, evaluating the same will cost $\mathcal{O}(\kappa n^3 \cdot \frac{\mu M}{n(t+1)\kappa}) = \mathcal{O}(\mu \cdot n \cdot M)$ bits. The only steps in Π_{ONLINE} which require interaction among the parties are during the instances of the reconstruction protocols, which eventually terminate for the honest parties. Hence we get Theorem 2 for the asynchronous setting.

Theorem 1. *Protocol Π_{ONLINE} UC-securely realizes the functionality $\mathcal{F}_f$ in the $\mathcal{F}_{\text{PREP}}$-hybrid model in the synchronous setting. The protocol has communication complexity $\mathcal{O}(\mu \cdot (n \cdot M + n^2))$ bits.*

Theorem 2. *Protocol Π_{ONLINE} UC-securely realizes the functionality $\mathcal{F}_f$ in the $\mathcal{F}_{\text{PREP}}$-hybrid model in the asynchronous setting. The protocol has communication complexity $\mathcal{O}(\mu \cdot (n \cdot M + n^2))$ bits.*

5 The Various Secure Realizations of $\mathcal{F}_{\text{PREP}}$

Securely Realizing $\mathcal{F}_{\text{PREP}}$ in the Synchronous Setting. In the full version, we present a protocol Π_{PREP} which realizes $\mathcal{F}_{\text{PREP}}$ in the synchronous setting and achieves UC security. The protocol is a straight forward adaptation of the offline phase protocol of [8,19] to deal with Shamir sharing, instead of additive sharing.

Protocol Π_{ONLINE}

Every party $P_i \in \mathcal{P}$ interact with $\mathcal{F}_{\text{PREP}}$ with input Setup, $\mathsf{Triples}$ and $(x^{(i)}, i, \mathsf{Input})$ and receives $\mathbf{dk}^{(i)}$, $\{\mathbf{pk}^{(j)}\}_{j=1}^{n}$, $R^i = (r^{(i,1)}, \ldots, r^{(i, \frac{\kappa n}{\mu})})$, $\{\mathsf{HE}.\mathbf{c}(r^{(j,1)}), \ldots, \mathsf{HE}.\mathbf{c}(r^{(j, \frac{\kappa n}{\mu})})\}_{j=1}^{n}$, its information for multiplication triples $\{(\langle a^{(l)}\rangle, \langle b^{(l)}\rangle, \langle c^{(l)}\rangle)\}_{l=1}^{M}$ and its information for inputs $\{\langle x^{(j)}\rangle\}_{j=1}^{n}$. The honest parties associate the sharing $(\langle a^{(l)}\rangle, \langle b^{(l)}\rangle, \langle c^{(l)}\rangle)$ with the l^{th} multiplication gate for $l \in \{1, \ldots, M\}$ and evaluate each gate in the circuit as follows:

– **Linear Gates**: using the linearity property of $\langle \cdot \rangle$-sharing, the parties apply the linear function associated with the gate on the corresponding $\langle \cdot \rangle$-shared gate inputs to obtain an $\langle \cdot \rangle$-sharing of the gate output.

– **Multiplication Gates**: M multiplication gates as grouped as a batch of $n \cdot (t+1) \cdot \frac{\kappa}{\mu}$. We explain the evaluation for one batch. Let the inputs to the ith batch be $\{(\langle p^{(l)}\rangle, \langle q^{(l)}\rangle)\}_{l=1}^{n\cdot(t+1)\cdot\frac{\kappa}{\mu}}$ and let $\{(\langle a^{(l)}\rangle, \langle b^{(l)}\rangle, \langle c^{(l)}\rangle)\}_{l=1}^{n\cdot(t+1)\cdot\frac{\kappa}{\mu}}$ be the corresponding associated multiplication triples. To compute $\langle p^{(l)} \cdot q^{(l)}\rangle$, the parties do the following:
  - Locally compute $\langle d^{(l)}\rangle = \langle p^{(l)}\rangle - \langle a^{(l)}\rangle = \langle p^{(l)} - a^{(l)}\rangle$ and $\langle e^{(l)}\rangle = \langle q^{(l)}\rangle - \langle b^{(l)}\rangle_t = \langle q^{(l)} - b^{(l)}\rangle$.
  - Publicly reconstruct the values $\{d^{(l)}\}_{l=1}^{n\cdot(t+1)\cdot\frac{\kappa}{\mu}}$ and $\{e^{(l)}\}_{l=1}^{n\cdot(t+1)\cdot\frac{\kappa}{\mu}}$ using two instances of RecPub.
  - On reconstructing $d^{(l)}, e^{(l)}$, the parties set $\langle p^{(l)} \cdot q^{l)}\rangle = d^{(l)} \cdot e^{(l)} + d^{(l)} \cdot \langle b^{(l)}\rangle + e^{(l)} \cdot \langle a^{(l)}\rangle + \langle c^{(l)}\rangle$.

– **Output Gate**: Let $\langle y\rangle$ be the sharing of the output gate. The parties execute $\mathsf{RecPrv}(\langle y\rangle, P_i)$ for every $P_i \in \mathcal{P}$, robustly reconstruct y and terminate.

Fig. 6. Realizing $\mathcal{F}_f$ with a Linear Overhead in $\mathcal{F}_{\text{PREP}}$-hybrid Model for the Synchronous Setting

Securely Realizing $\mathcal{F}_{\text{PREP}}$ with Abort in the Partial Synchronous Setting. Any secure realization of $\mathcal{F}_{\text{PREP}}$ has to ensure that all the honest parties have an *agreement* on the final outcome, which is impossible in the asynchronous setting with $t < n/2$ [25,26]. Another difficulty in realizing $\mathcal{F}_{\text{PREP}}$ in an asynchronous setting is that it is possible to ensure input provision from only $n - t$ parties to avoid endless wait. For $n = 2t + 1$ this implies that there may be only one honest input provider. This may not be acceptable for most practical applications of MPC. To get rid of the latter difficulty, [16] introduced the following variant of the traditional asynchronous communication setting, which we refer as *partial asynchronous setting*:

- The protocols in the partial asynchronous setting have one *synchronization* point. Specifically, there exists a certain well defined time-out and the assumption is that all the messages sent by the honest parties before the deadline will reach to their destinations within this deadline.
- Any protocol executed in the partial asynchronous setting need not always terminate and provide output to all the honest parties. Thus the adversary may cause the protocol to fail. However it is required that the protocol up to the synchronization point does not release any new information to the adversary.

In the full version we examine how to make Π_{PREP} work in the partial asynchronous setting. We present two solutions; the first which allows some synchronous rounds after the synchronization point, and one which uses a non-equivocation mechanism (which can be implemented using a trusted hardware module).

Acknowledgements

This work has been supported in part by ERC Advanced Grant ERC-2010-AdG-267188-CRIPTO, by EPSRC via grants EP/I03126X and EP/M016803, by DARPA and the US Navy under contract #N66001-15-C-4070, and by the Infosys Foundation.

References

1. J. Baron, K. E. Defrawy, J. Lampkins, and R. Ostrovsky. How to Withstand Mobile Virus Attacks, Revisited. In M. M. Halldórsson and S. Dolev, editors, *PODC*, pages 293–302. ACM, 2014.
2. D. Beaver. Efficient Multiparty Protocols Using Circuit Randomization. In J. Feigenbaum, editor, *CRYPTO*, volume 576 of *Lecture Notes in Computer Science*, pages 420–432. Springer Verlag, 1991.
3. Z. Beerliová-Trubíniová and M. Hirt. Efficient Multi-party Computation with Dispute Control. In S. Halevi and T. Rabin, editors, *TCC*, volume 3876 of *Lecture Notes in Computer Science*, pages 305–328. Springer, 2006.
4. Z. Beerliová-Trubíniová and M. Hirt. Perfectly-Secure MPC with Linear Communication Complexity. In R. Canetti, editor, *TCC*, volume 4948 of *Lecture Notes in Computer Science*, pages 213–230. Springer Verlag, 2008.
5. M. Ben-Or, R. Canetti, and O. Goldreich. Asynchronous Secure Computation. In S. R. Kosaraju, D. S. Johnson, and A. Aggarwal, editors, *STOC*, pages 52–61. ACM, 1993.
6. M. Ben-Or, S. Goldwasser, and A. Wigderson. Completeness Theorems for Non-Cryptographic Fault-Tolerant Distributed Computation (Extended Abstract). In J. Simon, editor, *STOC*, pages 1–10. ACM, 1988.
7. E. Ben-Sasson, S. Fehr, and R. Ostrovsky. Near-Linear Unconditionally-Secure Multiparty Computation with a Dishonest Minority. In R. Safavi-Naini and R. Canetti, editors, *CRYPTO*, volume 7417 of *Lecture Notes in Computer Science*, pages 663–680. Springer, 2012.
8. R. Bendlin, I. Damgård, C. Orlandi, and S. Zakarias. Semi-homomorphic Encryption and Multiparty Computation. In K. G. Paterson, editor, *EUROCRYPT*, volume 6632 of *Lecture Notes in Computer Science*, pages 169–188. Springer, 2011.
9. Zvika Brakerski, Craig Gentry, and Vinod Vaikuntanathan. (leveled) fully homomorphic encryption without bootstrapping. *TOCT*, 6(3):13:1–13:36, 2014.
10. R. Canetti. *Studies in Secure Multiparty Computation and Applications*. PhD thesis, Weizmann Institute, Israel, 1995.
11. R. Canetti. Security and Composition of Multiparty Cryptographic Protocols. *J. Cryptology*, 13(1):143–202, 2000.
12. D. Chaum, C. Crépeau, and I. Damgård. Multiparty Unconditionally Secure Protocols (Extended Abstract). In *STOC*, pages 11–19. ACM, 1988.
13. A. Choudhury, M. Hirt, and A. Patra. Asynchronous Multiparty Computation with Linear Communication Complexity. In Y. Afek, editor, *DISC*, volume 8205 of *Lecture Notes in Computer Science*, pages 388–402. Springer, 2013.
14. A. Choudhury, J. Loftus, E. Orsini, A. Patra, and N. P. Smart. Between a Rock and a Hard Place: Interpolating between MPC and FHE. In K. Sako and P. Sarkar, editors, *ASIACRYPT*, volume 8270, pages 221–240. Springer, 2013.

15. A. Choudhury and A. Patra. Optimally Resilient Asynchronous MPC with Linear Communication Complexity. In S. K. Das, D. Krishnaswamy, S. Karkar, A. Korman, M. Kumar, M. Portmann, and S. Sastry, editors, *ICDCN*, pages 5:1–5:10. ACM, 2015.
16. I. Damgård, M. Geisler, M. Krøigaard, and J. B. Nielsen. Asynchronous Multiparty Computation: Theory and Implementation. In S. Jarecki and G. Tsudik, editors, *PKC*, pages 160–179, 2009.
17. I. Damgård, Y. Ishai, and M. Krøigaard. Perfectly Secure Multiparty Computation and the Computational Overhead of Cryptography. In H. Gilbert, editor, *EUROCRYPT*, volume 6110 of *Lecture Notes in Computer Science*, pages 445–465. Springer, 2010.
18. I. Damgård and J. B. Nielsen. Scalable and Unconditionally Secure Multiparty Computation. In A. Menezes, editor, *CRYPTO*, volume 4622 of *Lecture Notes in Computer Science*, pages 572–590. Springer Verlag, 2007.
19. I. Damgård, V. Pastro, N. P. Smart, and S. Zakarias. Multiparty Computation from Somewhat Homomorphic Encryption. In R. Safavi-Naini and R. Canetti, editors, *CRYPTO*, volume 7417 of *Lecture Notes in Computer Science*, pages 643–662. Springer, 2012.
20. 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 Jason Crampton, Sushil Jajodia, and Keith Mayes, editors, *Computer Security - ESORICS 2013*, volume 8134 of *Lecture Notes in Computer Science*, pages 1–18. Springer, 2013.
21. D. Genkin, Y. Ishai, and A. Polychroniadou. Efficient Multi-party Computation: From Passive to Active Security via Secure SIMD Circuits. In R. Gennaro and M. Robshaw, editors, *CRYPTO*, volume 9216 of *Lecture Notes in Computer Science*, pages 721–741. Springer, 2015.
22. R. Gennaro, M. O. Rabin, and T. Rabin. Simplified VSS and Fact-Track Multiparty Computations with Applications to Threshold Cryptography. In B. A. Coan and Y. Afek, editors, *podc*, pages 101–111. ACM, 1998.
23. O. Goldreich, S. Micali, and A. Wigderson. How to Play any Mental Game or A Completeness Theorem for Protocols with Honest Majority. In *STOC*, pages 218–229. ACM, 1987.
24. M. Hirt and J. B. Nielsen. Robust Multiparty Computation with Linear Communication Complexity. In C. Dwork, editor, *CRYPTO*, volume 4117 of *Lecture Notes in Computer Science*, pages 463–482. Springer, 2006.
25. M. Hirt, J. B. Nielsen, and B. Przydatek. Cryptographic Asynchronous Multi-party Computation with Optimal Resilience (Extended Abstract). In *EUROCRYPT*, LNCS 3494, pages 322–340. Springer Verlag, 2005.
26. M. Hirt, J. B. Nielsen, and B. Przydatek. Asynchronous Multi-Party Computation with Quadratic Communication. In *ICALP*, LNCS 5126, pages 473–485. Springer Verlag, 2008.
27. Marcel Keller, Peter Scholl, and Nigel P. Smart. An architecture for practical actively secure MPC with dishonest majority. In Ahmad-Reza Sadeghi, Virgil D. Gligor, and Moti Yung, editors, *ACM CCS'13*, pages 549–560. ACM, 2013.
28. A. C. Yao. Protocols for Secure Computations (Extended Abstract). In *FOCS*, pages 160–164. IEEE Computer Society, 1982.