Constant-Round Asynchronous Multi-Party Computation

Sandro Coretti[1], Juan Garay[2], Martin Hirt[1], and Vassilis Zikas[3]

[1]Department of Computer Science, ETH Zurich, {`corettis,hirt,vzikas`}`@inf.ethz.ch`
[2]Yahoo Labs, `garay@yahoo-inc.com`
[3]Department of Computer Science, RPI, `vzikas@cs.rpi.edu`

February 25, 2016

Abstract

Secure multi-party computation (MPC) allows several mutually distrustful parties to securely compute a joint function of their inputs and exists in two main variants: In *synchronous* MPC parties are connected by a synchronous network with a global clock, and protocols proceed in *rounds* with strong delivery guarantees, whereas *asynchronous* MPC protocols can be deployed even in networks that deliver messages in an arbitrary order and impose arbitrary delays on them.

The two models—synchronous and asynchronous—have to a large extent developed in parallel with results on both feasibility and asymptotic efficiency improvements in either track. The most notable gap in this parallel development is with respect to round complexity. In particular, although under standard assumptions on a synchronous communication network (availability of secure channels and broadcast), synchronous MPC protocols with (exact) constant rounds have been constructed, to the best of our knowledge, thus far no constant-round asynchronous MPC protocols are known, with the best protocols requiring a number of rounds that is linear in the multiplicative depth of the arithmetic circuit computing the desired function.

In this work we close this gap by providing the first constant-round asynchronous MPC protocol. Our protocol is optimally resilient (i.e., it tolerates up to $t < n/3$ corrupted parties), adaptively secure, and makes black-box use of a pseudo-random function. It works under the standard network assumptions for protocols in the asynchronous MPC setting, namely, a complete network of point-to-point (secure) asynchronous channels with eventual delivery and asynchronous Byzantine agreement (aka consensus). We provide formal definitions of these primitives and a proof of security in the Universal Composability framework.

1 Introduction

In secure multi-party computation (MPC), a set of n parties $p_1, \ldots, p_n$, each holding some private input, wish to jointly compute a function on these inputs in a fashion such that even up to t colluding adversarial parties are unable to obtain any information beyond what they can extract from their inputs and outputs or to affect the computation in any way other than contributing their desired inputs. The problem of MPC has been studied in the two important settings of *synchronous* and *asynchronous* networks, respectively.

MPC protocols for the synchronous setting assume a network in which parties proceed in rounds, with the guarantee that messages sent by any party in any given round are delivered to all recipients in the same round. Consequently, in all such protocols the parties are assumed to be (at least partially) synchronized, i.e., to be in the same round at all times.

In real-world networks, such as the Internet, this synchrony assumption corresponds to assuming that the parties have (partially) synchronized clocks and communicate over channels with a known upper-bounded latency. The synchronous structure is then imposed by "timeouts," i.e., in each round the parties wait for an amount T of time defined by their estimate of when other parties send their messages and the bound on the network latency. If their estimate is accurate and their clocks are indeed synchronized, this will ensure that parties receive all messages sent to them from honest senders before the end of the round (timeout). Thus, after time T has passed, they can safely assume that if a message was expected for the current round but has not been received, then the sender must be adversarial. The security of synchronous protocols heavily relies on this assumption. In fact, many of them would become completely insecure if there is even a single delayed message. As a result, the round length T must typically be set much higher than the average transmission time.

A natural question is therefore to study the security one can obtain if no synchrony assumption is made but merely under the assumption that messages sent by honest parties are *eventually* delivered.[1] In particular, messages sent by parties can be reordered arbitrarily and delayed by arbitrary (albeit finite) amounts of time in such an asynchronous network. Note that one could consider even more pessimistic networks where the adversary can block messages sent by honest parties; this is for example the case in the base network assumed in Canetti's UC framework [14]. In such networks, however, protocols cannot be guaranteed to (eventually) terminate as the adversary can delay the computation indefinitely.

In asynchronous MPC protocols parties do not wait until a round times out. Rather, as soon as a party has received enough messages to compute its next message[2], it computes that message, sends it, and moves on. In that sense, asynchronous MPC protocols are "opportunistic" and terminate essentially as quickly as the network allows. Hence, they can be much faster than their synchronous counterparts depending on the network latency.[3]

In this work, unless explicitly stated otherwise, whenever we refer to the *asynchronous (communication) model*, we mean the above asynchronous model with eventual delivery.

On the importance of round complexity. The inherent need for waiting until each round times out clearly makes round complexity an important consideration for the performance of *synchronous* MPC protocols. Indicatively, Schneider and Zohner [34] have shown that as the latency between machines increases, the cost of each round becomes more and more significant.

Despite their opportunistic nature, round complexity is just as important a consideration for *asynchronous* protocols, since a protocol's round complexity can be a more relevant efficiency metric than, for example, its bit complexity. Indeed, at the conceptual/theoretical level, having constant-round

---

[1]The eventual-delivery assumption is supported by the fact that whenever a message is dropped or delayed for too long, Internet protocols typically resend that message.

[2]What "enough" means is concretely specified by the party's protocol.

[3]This speed up, however, does not come for free, as it inevitably allows the adversary to exclude some of the honest parties' inputs from being considered in the computation.

protocols allows us to use them as sub-routines in a higher level protocol without blowing up (asymptotically) the round complexity of the calling protocol, while at the practical level, communication time is often dominated by the round-trip times in the network and not by the size of the messages. For example, it takes about the same amount of time to transmit a byte and a megabyte, while sending a message from A to B over many intermediate nodes, computing something at B, and sending an answer back to A may take a (comparatively) long time.

Our contributions. In this paper, we first formalize the asynchronous model with *eventual delivery* in the universal composability (UC) framework [14], introduce a suitable formal notion of *asynchronous round complexity*, and formulate the basic communication resources (such as asynchronous secure channel and asynchronous Byzantine agreement [A-BA]) as ideal functionalities in that model. (See Section 3.)

We then present the—to the best of our knowledge—first *constant-round* MPC protocol for this asynchronous setting (i.e., a protocol whose round complexity is independent of the multiplicative depth of the evaluated circuit and the number n of parties) based on standard assumptions, namely, the existence of pseudo-random functions (PRFs).[4] The protocol is UC-secure in the secure-channels model with A-BA, and makes *black-box* use of the underlying PRF, tolerating a computationally bounded, *adaptive* adversary actively corrupting up to $t < n/3$ parties, which is optimal for this setting.[5]

At a high level, here is how we construct our constant round protocol. First, we devise a constant-depth circuit for computing the keys, masked values, and (shares of the) garbled gates needed for a distributed evaluation of a Yao garbled circuit that encodes the function the parties wish to compute. This circuit is then evaluated by means of a linear-round (in the depth of the circuit and in n) asynchronous protocol. However, this circuit is Boolean whereas all existing asynchronous protocols evaluate arithmetic circuits. To deal with this mismatch we devise an asynchronous protocol for computing Boolean circuits by appropriately adapting the protocol by Ben-Or, Kelmer, and Rabin [9]. Any party who receives the output from the evaluation of the Boolean circuit uses it to encrypt shares of each garbled gate, which it sends to all other parties. Finally, each party *locally* evaluates the (distributed) garbled circuit by decrypting incoming encrypted shares of each gate and reconstructing the function table of the gate as soon as sufficiently many consistent shares have arrived until all gates are evaluated. Once all gates are evaluated in this fashion, the party is in possession of the output. The protocol and its analysis are presented in Section 4.

Related work. Beaver, Micali, and Rogaway [3] were the first to provide a constant-round MPC protocol in the synchronous stand-alone model. (Refer to Appendix A for a more detailed and historical account of the development of MPC protocols in both the synchronous and asynchronous settings, together with the tools that are used in each setting.) Their protocol is secure in the computational setting and tolerates an adaptive adversary who actively corrupts up to $t < n/2$ parties. The complexity of [3] was improved by Damgård and Ishai [19], who provided the first constant-round protocol making black-box use of the underlying cryptographic primitive (a pseudo-random generator). Importantly, both [3] and [19] assume a broadcast channel, an assumption essential for obtaining constant-round MPC. Indeed, as proved in [22, 20], it is impossible to implement such a broadcast channel from point-to-point communication in a constant number of rounds, and although expected constant-round broadcast protocols exist in the literature (e.g., [21, 30]), using them to instantiate calls within the constructions of [3] or [19] would not yield an expected constant-round protocol [7]. The intuitive

---

[4] We note that in the synchronous setting, constant round MPC can be achieved from threshold fully homomorphic encryption (TFHE) [2]. Although we are not aware of any published work in the asynchronous setting, the synchronous methodology could in principle also be used in our eventual-delivery framework, with the additional requirement (besides TFHE) of a CRS (for NIZKs) to achieve UC-security. In contrast, our work provides a constant-round UC-secure MPC protocol from standard assumptions.

[5]Refer to the discussion in related work below on the necessity of this bound in the asynchronous setting.

reason—formally argued by Ben-Or and El-Yaniv [7]—is that the process of running n such broadcast protocols (even in parallel) does not terminate in an expected constant number of rounds.

The model of *asynchronous* communication with eventual delivery was considered early on in seminal works on fault-tolerant distributed computing, e.g., [23], (although to our knowledge this paper is the first to formalize this capability in the UC framework). The study of optimally resilient MPC in this type of asynchronous networks was initiated by Ben-Or, Canetti, and Goldreich [6], who proved that any function can be computed by a perfectly secure asynchronous protocol if and only if at most $t < n/4$ parties are corrupted. Following that result, Ben-Or, Kelmer, and Rabin [9] showed that if a negligible error probability is allowed, the bound $t < n/3$ is necessary and sufficient for asynchronous MPC.[6] More recently, Hirt *et al.* [27, 28] provided computationally secure solutions (i.e., protocols tolerating a computationally bounded adversary) and Beerliová and Hirt [4] perfectly secure solutions with improved communication complexity.

The above asynchronous protocols are secure—according to simpler, stand-alone security definitions—if one assumes point-to-point communication and an A-BA protocol. Similarly to their synchronous counterparts, all the above protocols—even assuming an A-BA primitive—have round complexity linear in the multiplicative depth of the arithmetic circuit that computes the function, as they follow the standard gate-by-gate evaluation paradigm.

We note in passing that although in the synchronous setting BA implies broadcast, this is not the case in the asynchronous setting. Indeed, Canetti and Rabin [16] provide an asynchronous BA protocol tolerating $t < n/3$ malicious parties, which if every honest party terminates at the latest after a poly-logarithmic number of rounds, securely implements asynchronous BA except with negligible probability. A broadcast protocol with similar guarantees is provably impossible [23], and existence of an asynchronous BA protocol which terminates in a strict constant number of rounds would contradict the impossibility from [22, 20]. Similarly to the synchronous case, although solutions for asynchronous BA with expected constant number of rounds exist [12, 16], using them in the above asynchronous protocol to replace invocations to asynchronous BA would not yield an expected constant-round MPC protocol [7].[7]

2   Model and Building Blocks

We denote the player set by $\mathcal{P} = \{p_1, \ldots, p_n\}$ and consider a computationally bounded *adaptive t-adversary*, i.e., the adversary gets to corrupt up to t parties dynamically during the protocol execution and depending on its protocol view. The most common network model for the execution of asynchronous protocols is secure channels with eventual delivery, where the adversary is allowed to delay the delivery of any message by an arbitrary but finite amount of time, i.e., he is not able to block messages sent among honest parties. Moreover, as argued in the introduction, existing asynchronous protocols rely on an additional resource, namely, an asynchronous version of Byzantine agreement (A-BA) instead of a broadcast channel, and such a resource is even necessary to obtain an (exact) constant number of rounds. We formalize this model and formulate the ideal functionalities corresponding to these communication resources separately in Section 3.

We now present some basic tools we use in our protocol.

---

[6]The necessity of the $t < n/3$ bound follows from the result by Canetti *et al.* [6, 13], who argue that this bound is necessary for fail-stop adversaries; it also applies to computational security and assuming A-BA. Moreover, note that in the asynchronous setting, all feasibility bounds are worse by an additive term of t compared to the synchronous setting. Intuitively, this stems from the fact that honest parties cannot distinguish between messages by other honest parties being delayed and messages by corrupted parties not being sent. Thus, in particular, perfectly secure asynchronous MPC is possible only if $t < n/4$.

[7]Nonetheless, [7] does describe an alternative way of obtaining several asynchronous BA protocols that are guaranteed to all terminate in expected constant number of rounds.

Secret sharing. Our construction makes use of Shamir's secret sharing scheme [35], which allows to encode a secret into n shares such that any subset of t shares gives no information about the secret and such that from any subset of $t + 1$ shares the secret can be reconstructed.

For a sharing of a secret s, let $[s]_i$ denote the ith share. A set of shares are called *t-consistent* if they lie on a polynomial of degree at most t. For a tuple of secrets $s = (s_1, \ldots, s_\ell)$, denote—in slight abuse of notation—by $[s]_i := ([s_1]_i, \ldots, [s_\ell]_i)$ the tuple of the ith shares of the values and refer to it as the ith share of s. A set of such tuples is called *t-consistent* if the property holds for all components.

A linear-round asynchronous MPC protocol. In [9], Ben-Or, Kelmer, and Rabin constructed a protocol, call it π_{BKR}, that computes an arbitrary n-party function f in an asynchronous environment assuming asynchronous point-to-point (secure) channels and asynchronous BA.[8] The protocol follows the gate-by-gate evaluation paradigm [8, 26, 17], where the function to be evaluated is represented as an arithmetic circuit over a sufficiently large finite field, and the computation proceeds by evaluating sequentially the gates of depth one, then the gates of depth two, and so on. The evaluation of each gate requires a constant number of (asynchronous) rounds,[9] thus making the round complexity of the overall protocol linear in the depth of the circuit.

π_{BKR} was designed for a simpler, stand-alone security definition, which only ensures sequential composition. In the next section we show how it can be cast in our eventual-delivery model to give UC-security guarantees.

3   A UC Model for Asynchronous Computation with Eventual Message Delivery

In this section we formalize the asynchronous network model with eventual message delivery in the UC framework. We start with the basic task of point-to-point communication.

Asynchronous communication with eventual delivery. Our formulation of communication with eventual delivery within the UC framework builds on ideas from [31]. In particular, we capture such communication by allowing the parties access to multi-use bilateral secure channels, where a sender $p_i \in \mathcal{P}$ can input a messages to be delivered to some recipient $p_j \in \mathcal{P}$; messages are buffered and delivered in an order specified by the adversary.

To ensure that when p_s and p_r are honest, the adversary cannot delay the delivery of submitted messages arbitrarily, we make the following modifications: We first turn the UC secure channels functionality to work in a "fetch message" mode, where the channel delivers the message to its intended recipient p_j if and only if p_j asks to receive it by issuing a special "fetch" command. If the adversary wishes to delay the delivery of some message, he needs to submit to the channel functionality an integer value T—the *delay*. This will have the effect of the channel ignoring the first T fetch attempts following the reception of the sender's message. Importantly, we require the adversary send the delay T in unary notation; this will ensure that the delay will be bounded by the adversary's running time,[10] and thus a polynomial environment will be able to observe the transmission through its completion. To allow the adversary freedom in scheduling delivery of messages, we allow him to input delays more than once, which are added to the current delay amount. (If the adversary wants to deliver the message in the next activation, all he needs to do is submit a negative delay.)

The detailed specification of secure channels with eventual delivery, denoted $\mathcal{F}_{\text{A-SMT}}$, is given in Figure 1. In the description, we denote by $\vec{M}$ a vector of strings. We also use $||$ to denote the operation which adds a string to $\vec{M}$; concretely, if $\vec{M} = (m_1, \ldots, m_\ell)$, then $\vec{M}||m := (m_1, \ldots, m_\ell, m)$

Functionality $\mathcal{F}_{\text{A-SMT}}(p_s, p_r)$

Initialize $D := 0$ and $\vec{M} := (\texttt{eom})$, where $\texttt{eom}$ is a special "end-of-messages" symbol.

- Upon receiving a message $(\texttt{send}, m)$ from p_s set $D := D + 1$ and $\vec{M} := (m, mid)||\vec{M}$, where mid is a unique message ID, and send (D, mid) to the adversary.
- Upon receiving a message $(\texttt{fetch})$ from p_r:
    1. Set $D := D - 1$.
    2. If $D = 0$ and $\vec{M} = ((m_1, mid_1), \ldots, (m_\ell, mid_\ell), \texttt{eom})$ then set $\vec{M} := ((m_2, mid_2), \ldots, (m_\ell, mid_\ell), \texttt{eom})$, and send the message m_1 to p_r (otherwise, no message is sent and the activation is given back to the environment, as defined in the UC framework).
- Upon receiving a message $(\texttt{delay}, T)$ from the adversary, if T is a valid delay (i.e., it encodes an integer in unary notation), set $D := \max\{1, D + T\}$; otherwise, ignore the message.
- Upon receiving a message $(\texttt{permute}, \pi)$ from the adversary, if $\pi : [|\vec{M}| - 1] \to [|\vec{M}| - 1]$ is a permutation over $[|\vec{M}| - 1]$, then set $\vec{M} := \vec{M}' = ((m_{\pi(1)}, mid_{\pi(1)}), \ldots, (m_{\pi(\ell)}, mid_{\pi(\ell)}), \texttt{eom})$.
- (*Adaptive message replacement*) Upon receiving a message $(p_s, ((m_1, mid_1), \ldots, (m_{\ell'}, mid_{\ell'})), T')$ from $\mathcal{A}$, if p_s is corrupted and $D > 0$ and T' is a valid delay, then set $D = \max\{1, T'\}$ and set $\vec{M} := ((m_1, mid_1), \ldots, (m_{\ell'}, mid_{\ell'}), \texttt{eom})$.

Figure 1: Asynchronous secure channel with eventual delivery

and $m||\vec{M} = (m, m_1, \ldots, m_\ell)$.

We refer to the model in which every two parties p_i and p_j in $\mathcal{P}$ have access to an independent instance of $\mathcal{F}_{\text{A-SMT}}(p_i, p_j)$ as the $\mathcal{F}_{\text{A-SMT}}$-*hybrid model*. An asynchronous protocol in such a model proceeds as follows: Whenever a party p_j gets activated, if its current protocol instructions include sending some message m to some other party p_j, then the party inputs $(\texttt{send}, m)$ to $\mathcal{F}_{\text{A-SMT}}(p_i, p_j)$; otherwise, p_i sends a $\texttt{fetch}$ message to every channel $\mathcal{F}_{\text{A-SMT}}(p_j, p_i)$, $j \in [n]$ in a round-robin fashion, i.e., if in the previous activation it sent a $(\texttt{fetch})$ message to $\mathcal{F}_{\text{A-SMT}}(p_j, p_i)$, then it sends a $(\texttt{fetch})$ message to $\mathcal{F}_{\text{A-SMT}}(p_{(j \bmod n)+1}, p_i)$.

Remark 1 (On permuting messages). *Our formulation of an asynchronous channel captures the somewhat pessimistic view of asynchronous communication, implicit in many works on asynchronous distributed protocols, where the adversary has full scheduling power and can, in particular, reorder the messages sent by any party as he wishes. One could attempt to emulate a network which does not allow for reordering of the messages—the so-called first-in-first-out (FIFO) channel—by adding appropriate (publicly known) message identifiers and instructing the parties to wait until a specific identifier is delivered before outputting messages with other identifiers. However, we note that such an emulating protocol would be distinguished from the original when, for example, we consider an adversary that introduces no delay and an environment that inputs two messages in a row and corrupts the receiver as soon as the first message is supposed to have been delivered.*

Asynchronous secure function evaluation (SFE). In an asynchronous environment, it is impossible to get guaranteed (eventual) termination and input completeness, i.e, take into account all inputs in the computation of the function (cf. [31] and early work on fault-tolerant distributed computing). The reason is that if honest parties wait until the inputs of all parties are delivered, then the adversary can make them wait indefinitely for corrupted parties to give their inputs (honest parties have no way

---

[8][9] also assumes A-Cast to get a more efficient solution, but as argued in the introduction, A-Cast can be easily reduced to asynchronous BA in two rounds.

[9]Note that in each such round the parties might invoke the asynchronous BA resource.

[10]We refer to [14] for a formal definition of running time in the UC framework.

Functionality $\mathcal{F}_{\text{A-SFE}}^{f}(\mathcal{P})$

$\mathcal{F}_{\text{A-SFE}}^{f}$ proceeds as follows, given a function $f : (\{0,1\}^* \cup \{\bot\})^n \times R \to (\{0,1\}^*)^n$ and a player set $\mathcal{P}$. For each $i \in \mathcal{P}$, initialize variables x_i and y_i to a default value $\bot$ and a current delay $D_i := 0$. Additionally, set $\mathcal{I} := \mathcal{H}$. (Recall that $\mathcal{H}$ denotes the set of honest parties)

- Upon receiving message $(\text{no-input}, \mathcal{P}')$ from the adversary, if $|\mathcal{P}'| \leq |\mathcal{P} \setminus \mathcal{H}|$ and no party has received an output (output, y) yet, then set $\mathcal{I} = \mathcal{H} \setminus \mathcal{P}'$; otherwise ignore this message.
- Upon receiving input (input, v) from party $p_i \in \mathcal{P}$ (or from the adversary in case p_i is corrupted), do the following:
    - If some party (or the adversary) has received an output (output, y), then ignore this message; otherwise, set $x_i := v$.
    - If $x_i \neq \bot$ for every $p_i \in \mathcal{I}$, then compute $(y_1, \ldots, y_n) = f((x_1', \ldots, x_n'), r)$ for a uniformly random r, where $x_i' = x_i$ for $p_i \in \mathcal{I} \cup (\mathcal{P} \setminus \mathcal{H})$ and $x_i' = \bot$ for all other parties.
    - Send (input, i) to the adversary.
- Upon receiving (delay, p_i, T), from the adversary, set $D_i := D_i + T$.
- Upon receiving message (fetch) from party $p_i \in \mathcal{P}$, if y_i has not yet been set (i.e., $y_i = \bot$) then ignore this message, else do:
    - Set $D_i := D_i - 1$
    - If $D_i = 0$, send (output, y_i) to p_i.

Figure 2: Asynchronous SFE with eventual delivery

of distinguishing between an honest sender whose message is delayed and a corrupt sender who did not send a message). Thus, to ensure eventual termination, the parties cannot afford to wait for input from more than $n - t$ parties, as the t remaining parties might be the corrupted ones. Therefore, protocols for asynchronous computation of a multi-party function f on inputs $x_1, \ldots, x_n$ from parties $p_1, \ldots, p_n$ end up computing the function $f|_{\mathcal{P}'}(x_1, \ldots, x_n) = f(x_1', \ldots, x_n')$ for some $\mathcal{P}' \subseteq \mathcal{P}$ with $|\mathcal{P}'| = t$, where $x_i' = x_i$ if $p_i \notin \mathcal{P}'$, and otherwise a default value (denoted $\bot$).

Moreover, by being able to schedule the delivery of messages from honest parties, the adversary can (in worst-case scenarios) choose exactly the set $\mathcal{P}'$. Therefore, the ideal functionality corresponding to asynchronous SFE with eventual termination needs to allow the simulator to choose this set depending on the adversary's strategy. Moreover, the simulator should be allowed to schedule delivery of the outputs depending on the adversary's strategy, but not allowed to delay them arbitrarily. This last requirement can be achieved, as in the case of $\mathcal{F}_{\text{A-SMT}}$, by turning the SFE functionality into a "fetch message"-mode functionality and allowing the simulator to specify a delay on the delivery to every party.

The SFE functionality with the above properties is described in Figure 2. In the description, we use $\mathcal{H} \subseteq \mathcal{P}$ to denote the set of honest parties; note that $\mathcal{H}$ is dynamically updated as the adversary corrupts new parties. Moreover, we use $\mathcal{I}$ to denote the set of honest parties whose input is allowed to be considered in the computation, and require that $|\mathcal{I}| \geq n - 2t$. We provide a generic description of the functionality for an arbitrary number t of corruptions; however, and as implied by classical impossibility results, we are only able to realize it for $t < n/3$ [6].

Asynchronous BA with eventual delivery. The last primitive we describe is (UC) asynchronous BA with eventual message delivery. In such a BA primitive, every party has an input, and we want to ensure the following properties: All honest parties (eventually) output the same value y (consistency), and if all honest parties have the same input x, then this output is $y = x$. Intuitively, asynchronous BA can be cast as a version of asynchronous SFE for the function that looks at the set of received inputs

---

Functionality $\mathcal{F}_{\text{A-BA}}(\mathcal{P})$

For each $i \in \mathcal{P}$, initialize variables x_i and y_i to a default value $\perp$ and a current delay $D_i := 0$. Additionally, set $\mathcal{I} := \mathcal{H}$. (Recall that $\mathcal{H}$ denotes the set of honest parties)

- Upon receiving message (no-input, $\mathcal{P}'$) from the adversary, if $|\mathcal{P}'| \leq |\mathcal{P} \setminus \mathcal{H}|$ and no party has received an output (output, y) yet, then set $\mathcal{I} = \mathcal{H} \setminus \mathcal{P}'$; otherwise ignore this message.
- Upon receiving input (input, v) from party $p_i \in \mathcal{P}$ (or from the adversary in case p_i is corrupted), do the following:
    - If some party (or the adversary) has received an output (output, y), then ignore this message; otherwise, set $x_i := v$.
    - If $x_i \neq \perp$ for every $p_i \in \mathcal{I}$, then set $(y_1, \ldots, y_n) := (y, \ldots, y)$, where if there exists $x \neq \perp$ such that $x_i = x$ for every $p_i \in \mathcal{I}$, then $y = x$; otherwise $y = x_j$, where p_j is the party in $\mathcal{P} \setminus \mathcal{H}$ with the smallest index.
    - Send (input, i) to the adversary.
- Upon receiving (delay, p_i, T), from the adversary, set $D_i := D_i + T$.
- Upon receiving message (fetch) from party $p_i \in \mathcal{P}$, if y_i has not yet been set (i.e., $y_i = \perp$) then ignore this message, else do:
    - Set $D_i := D_i - 1$
    - If $D_i = 0$, send (output, y_i) to p_i.

Figure 3: Asynchronous BA with eventual delivery

and, if all inputs contributed by honest parties are identical[11] and equal to some x, sets the output to x for every party; otherwise, it sets the output to the input of some corrupted party (for example, the first in any ordering, e.g., lexicographic). The formal definition of $\mathcal{F}_{\text{A-BA}}$ is given in Figure 3.

We will refer to the setting where every two parties p_i and p_j in $\mathcal{P}$ have access to an independent instance of $\mathcal{F}_{\text{A-SMT}}(p_i, p_j)$ and, additionally, the parties have access to independent instances of $\mathcal{F}_{\text{A-BA}}(\mathcal{P})$ as the $\{\mathcal{F}_{\text{A-SMT}}, \mathcal{F}_{\text{A-BA}}\}$-*hybrid model*. The execution in the $\{\mathcal{F}_{\text{A-SMT}}, \mathcal{F}_{\text{A-BA}}\}$-hybrid model is analogous to the execution in the $\mathcal{F}_{\text{A-SMT}}$-hybrid model: Whenever a party p_i gets activated, if its current protocol instructions include sending some message m to some other party p_j or inputing a message m' to $\mathcal{F}_{\text{A-BA}}(\mathcal{P})$, then the party inputs (send, m) to $\mathcal{F}_{\text{A-SMT}}(p_i, p_j)$ or m' to $\mathcal{F}_{\text{A-BA}}(\mathcal{P})$, respectively; otherwise, p_i keeps sending (with each activation) a fetch to every channel $\mathcal{F}_{\text{A-SMT}}(p_i, p_j)$, $j \in [n]$ and then to $\mathcal{F}_{\text{A-BA}}(\mathcal{P})$ in a round-robin fashion.

Asynchronous rounds. We now briefly elaborate on the notion of rounds in an asynchronous environment. Unlike the situation in the synchronous case, where rounds are well specified by the protocol, the definition of rounds in an asynchronous setting requires a bit more care. Intuitively, two messages m_i and m_j sent by some party p_i in an asynchronous protocol are considered to be sent in rounds i and j, $j > i$, if m_j is generated by computation which takes as input a message received after p_i sent m_i. Following the above intuition, we define for each p_i and for each point in the protocol execution, the current round in which p_i is to be the number of times p_i alternated between sending (send, m) to some channel $\mathcal{F}_{\text{A-SMT}}(p_i, p_j)$, $p_j \in \mathcal{P}$ (or to the asynchronous BA functionality $\mathcal{F}_{\text{A-BA}}(\mathcal{P})$) and sending (fetch) to some $\mathcal{F}_{\text{A-SMT}}(p_k, p_i)$, $p_k \in \mathcal{P}$ or to $\mathcal{F}_{\text{A-BA}}(\mathcal{P})$. That is, every round (except for the first one)[12] starts by sending a (send, m) to some $\mathcal{F}_{\text{A-SMT}}(p_i, p_j)$ or to $\mathcal{F}_{\text{A-BA}}(\mathcal{P})$ after some (fetch) was sent by p_i and finishes with the first (fetch) command that p_i sends. The round complexity of the protocol is the maximum (over all honest parties) number of rounds that an honest party uses in the protocol execution.

We note in passing that, similarly to [31], the above formulation allows for any party to send

---

[11]Similarly to the SFE case, the adversary might prevent some of the honest parties from providing an input.

[12]The first round starts as soon as the party receives its protocol input from the environment.

several messages in each round: the party buffers the messages and while the buffer is not empty, in each activation the party pops the next message and sends it to its intended recipient.

A UC-secure linear-round MPC protocol with eventual delivery. Finally, we argue the security of protocol π_{BKR} mentioned in Section 2 in our model. π_{BKR} is information-theoretic and proved simulation-based secure, where the simulation is in fact black-box (i.e., the simulator uses the corresponding adversary in a black-box manner) and straight-line (the simulator does not rewind the adversary). Moreover, the protocol tolerates any adaptive t-adversary where $t < n/3$, a bound which is also tight [6]. Thus, by casting π_{BKR} in our UC $\{\mathcal{F}_{\mathrm{A\text{-}SMT}}, \mathcal{F}_{\mathrm{A\text{-}BA}}\}$-hybrid model—where every bilateral message exchange is implemented by the sender p_i and the receiver p_j using (an instance of the) channel $\mathcal{F}_{\mathrm{A\text{-}SMT}}(p_i, p_j)$ and every call to asynchronous BA done by invocation of $\mathcal{F}_{\mathrm{A\text{-}BA}}(\mathcal{P})$—we obtain a protocol for UC securely evaluating $\mathcal{F}_{\mathrm{A\text{-}SFE}}^f(\mathcal{P})$, which is linear in the depth of the circuit computing f. More formally:

Theorem 1 ([9])**.** *Let f be an n-ary function and C be an arithmetic circuit for computing f by parties in $\mathcal{P}$. Then there exists a protocol, π_{BKR}, which UC-securely realizes $\mathcal{F}_{\mathrm{A\text{-}SFE}}^f$ in the $\{\mathcal{F}_{\mathrm{A\text{-}SMT}}, \mathcal{F}_{\mathrm{A\text{-}BA}}\}$-hybrid model tolerating an adaptive t-adversary in a linear (in the depth of the circuit) number of rounds, provided $t < n/3$.*

4 Constant-Round Asynchronous SFE

In this section we present our asynchronous SFE protocol and prove its security and round complexity.

4.1 Description of the Protocol

Let Circ be a Boolean circuit that is to be evaluated in a multi-party computation. In our protocol for securely evaluating the function that Circ computes, denoted $\pi_{\mathrm{A\text{-}SFE}}(\mathsf{Circ}, \mathcal{P})$, the parties first jointly compute a garbled version of Circ (along the lines of [37, 3, 5]); every party then evaluates this garbled circuit locally to obtain the output of the computation. Computing the garbled circuit takes place in two phases: First, the parties evaluate a function $f_{\mathrm{PREP}}^{\mathsf{Circ}}$ (described below) which is represented by a *constant-depth* arithmetic circuit over a finite field using a (non-constant-round) asynchronous MPC protocol. Given the outputs of this function, the parties can then complete the computation of the garbled circuit within one additional asynchronous round.[13] Since the evaluation of the garbled circuit takes place locally and $f_{\mathrm{PREP}}^{\mathsf{Circ}}$ is computed via a constant-depth circuit, the entire protocol is a constant-round protocol.

We define and analyze our protocol in the $\{\mathcal{F}_{\mathrm{A\text{-}SFE}}^{f_{\mathrm{PREP}}^{\mathsf{Circ}}}, \mathcal{F}_{\mathrm{A\text{-}SMT}}\}$-hybrid model. Furthermore we provide a protocol for UC securely realizing the functionality $\mathcal{F}_{\mathrm{A\text{-}SFE}}^{f_{\mathrm{PREP}}^{\mathsf{Circ}}}$ from asynchronous secure channels and BA with eventual delivery based on π_{BKR} [9] (cf. Lemma 2).

Circuit garbling. Before elaborating on the protocol, we describe what the garbled version of Circ looks like.[14] Boolean circuit Circ consists of wires and `NAND` gates.[15] In the garbled version, every wire ω of Circ has a corresponding (secret) random mask m_ω, which is used to hide the real value on that wire. Consequently, every gate g, with input wires α and β and output wire γ, has a special function table T_g that works on masked values. It contains four entries z_γ^{xy}, corresponding to the masked value on the outgoing wire γ under the four possible combinations of masked inputs $x, y \in \{0, 1\}$ on wires

---

[13]Refer to Section 2 for a definition of asynchronous round complexity.

[14]Note that $f_{\mathrm{PREP}}^{\mathsf{Circ}}$ actually computes a "distributed" version of the garbled circuit (described below).

[15]Any (arithmetic or Boolean) circuit can be efficiently transformed into such a circuit.

Function $f_{\text{PREP}}^{\text{Circ}}((b_{\omega_{11}}, \ldots, b_{\omega_{1L_1}}), \ldots, (b_{\omega_{n1}}, \ldots, b_{\omega_{nL_n}}))$

The preparation function is parameterized by a Boolean circuit Circ describing the function to be computed. The wires of Circ are labeled by values $\omega \in \mathbb{N}$. We use Greek letters $\alpha, \beta, \gamma, \omega$ for referring to the wire labels.

INPUT. For every input wire ω, b_ω denotes the corresponding input bit.

CREATE RANDOM VALUES. For each wire ω do:
1. For each $i \in [n]$ choose a random sub-key $k_{\omega,0}^i \in \mathbb{F}^n$.
   Set $k_{\omega,0} := (k_{\omega,0}^1, \ldots, k_{\omega,0}^n)$.
2. For each $i \in [n]$ choose a random sub-key $k_{\omega,1}^i \in \mathbb{F}^n$.
   Set $k_{\omega,1} := (k_{\omega,1}^1, \ldots, k_{\omega,1}^n)$.
3. Choose random mask $m_\omega \in \{0,1\}$.

INPUT WIRES. For every input wire ω do:
1. Compute masked value $z_\omega := b_\omega \oplus m_\omega$.
2. Choose corresponding key $k_\omega := k_{\omega,z_\omega}$.

COMPUTE MASKED FUNCTION TABLES. For every gate g with wires α, β, γ do:
1. For every $x, y \in \{0,1\}$ do:

   (a) Compute masked value $z_\gamma^{xy} := ((x \oplus m_\alpha) \text{ NAND } (y \oplus m_\beta)) \oplus m_\gamma$.
   (b) Choose corresponding key $k_\gamma^{xy} := k_{\gamma,z_\gamma^{xy}}$.
   (c) Set $t_g^{xy} := (z_\gamma^{xy}, k_\gamma^{xy})$ and $T_g := (t_g^{00}, t_g^{01}, t_g^{10}, t_g^{11})$.

2. Compute a Shamir sharing of T_g (i.e., of every entry).

OUTPUT. Proceed as follows:
  (Public outputs) Output the following values to *all* players:

   1. For every *input* wire ω: the masked value z_ω and the corresponding key k_ω.

  (Private outputs) Output the following values to every $p_i \in \mathcal{P}$:

   1. For every wire ω: the subkeys $k_{\omega,0}^i$ and $k_{\omega,1}^i$.
   2. For every gate g: the ith share $[T_g]_i$ of T_g.
   3. For every output wire ω: the mask m_ω if p_i is to learn that output.

Figure 4: The description of function $f_{\text{PREP}}^{\text{Circ}}$ corresponding to the distributed version of circuit garbling.

α and β. Each entry is obtained by unmasking the inputs, applying the gate function (NAND), and re-masking the result with the mask of the outgoing wire. That is,

$$z_\gamma^{xy} = ((x \oplus m_\alpha) \text{ NAND } (y \oplus m_\beta)) \oplus m_\gamma,$$

for $x, y \in \{0,1\}$.

The entries of each function table need to be protected so that only the one entry necessary to evaluate the circuit can be accessed. To that end, for each wire ω there are two (secret) keys $k_{\omega,0}$ and $k_{\omega,1}$. In the function tables T_g, each entry z_γ^{xy} is now augmented by the corresponding key $k_{\gamma,z_\gamma^{xy}}$ of the outgoing wire γ. The pair $t_g^{xy} := (z_\gamma^{xy}, k_{\gamma,z_\gamma^{xy}})$ is encrypted with $k_{\alpha,x}$ and $k_{\beta,y}$ under the "tweak" (g, x, y). The resulting ciphertexts

$$c_g^{xy} := \text{Enc}_{k_{\alpha,x}, k_{\beta,y}}^{g,x,y} \left(t_g^{xy} \right) = \text{Enc}_{k_{\alpha,x}, k_{\beta,y}}^{g,x,y} \left(z_\gamma^{xy}, k_{\gamma,z_\gamma^{xy}} \right)$$

make up the garbled function table

$$C_g := (c_g^{00}, c_g^{01}, c_g^{10}, c_g^{11}),$$

9

where $(\mathsf{Enc}_{k_1,k_2}^T(\cdot), \mathsf{Dec}_{k_1,k_2}^T(\cdot))$ is a tweakable dual-key cipher. A suitable such cipher can be realized using a PRF [5].[16]

In order to be compatible with the garbled function tables, inputs to the circuit must be garbled as well. That is, for the input bit b_ω on input wire ω, the garbled input is $z_\omega := b_\omega \oplus m_\omega$ and the corresponding key is $k_\omega := k_{\omega,z_\omega}$.

With the garbled inputs and function tables, any party can (locally) evaluate the circuit as follows: Given the masked values and the corresponding keys of the incoming wires of some gate, the party can decrypt the corresponding row, obtaining the masked value on the outgoing wire and the corresponding key. In the end, the values on the output wires can be unblinded if the corresponding masks are known.

Distributed encryption. Given the input bits b_ω for all input wires ω, computing the garbled circuit could be described by a constant-depth circuit, since the garbled function tables can be computed in parallel after choosing the wire masks and keys. This circuit, however, would be rather large since it entails evaluating the cipher. Therefore, to avoid evaluating the cipher within the asynchronous MPC, the parties use the distributed-encryption technique by Damgård and Ishai [19]: Instead of computing $\mathsf{Enc}_{k_1,k_2}^T(m)$ for a message m, two keys k_1 and k_2, and a tweak T, the parties first jointly choose $2n$ subkeys $k_1^1, \ldots, k_1^n$ and $k_2^1, \ldots, k_2^n$, compute a Shamir sharing of $m = ([m]_1, \ldots, [m]_n)$, open $[m]_i$ as well as k_1^i and k_2^i to p_i for every i, and then each party encrypts its share $[m]_i$ of m using its two subkeys k_1^i and k_2^i and sends the resulting ciphertext $\mathsf{Enc}_{k_1^i,k_2^i}^T([m]_i)$ to all parties.

In order to decrypt, a party in possession of all keys recovers the shares by decrypting the ciphertexts received from other players and waits until it has $2t + 1$ t-consistent shares, which it uses to reconstruct m.[17]

Asynchronously evaluating Boolean circuits. Protocol π_{BKR} [9], which we wish to use to realize $\mathcal{F}_{\mathrm{A\text{-}SFE}}^{f_{\mathrm{PREP}}^{\mathsf{Circ}}}$, evaluates arithmetic circuits over fields with more than two elements; the circuit representing $f_{\mathrm{PREP}}^{\mathsf{Circ}}$, however, is Boolean. Thus, in order to evaluate it via π_{BKR} we transform it into an (arithmetic) circuit over an extension field F of GF(2), denoted $C_{f_{\mathrm{PREP}}^{\mathsf{Circ}}}^F$, by having every NAND gate with inputs $x, y \in \{0, 1\}$ replaced by the computation $1 - xy$, which can be implemented using addition and multiplication over the extension field F. The above transformation, however, works only if all the inputs to the circuit corresponding to bits in the Boolean circuit are either 0 or 1 in the corresponding field F. For the honest parties this is easy to enforce: they encode a 0 (resp., 1) input bit into the 0 (resp., 1) element of F. The adversary, however, might try to cheat by giving "bad" inputs, namely, inputs in $F \setminus \{0, 1\}$. We now show an explicit construction to ensure that the adversary cannot give any value other than 0 or 1, resulting in a simple adaptation of protocol π_{BKR}.[18]

Before describing the solution we recall the reader how π_{BKR} evaluates a given circuit. We omit several low-level details and keep the description at the level which is appropriate for a formal description of our adaptation; the interested reader is referred to [9] for further details. π_{BKR} follows the *gate-by-gate evaluation paradigm* [26, 8]: The circuit is evaluated in a gate-by-gate fashion, where the invariant is that after the evaluation of each gate (in fact, of each bulk of gates that are at the same multiplicative depth), the output of the gates is verifiably shared among all the parties. In fact, [9] defines the notion of *Ultimate Secret Sharing* (USS) which is a version of VSS that is appropriate for asynchronous computation with $t < n/3$;[19] More concretely, the first step is to process all input gates

---

[16]The security required from such a cipher is roughly semantic security even if one of the keys is known (see [5] for more details). Moreover, we assume a canonical injective mapping of triples (g, x, y) to the tweak space of the cipher.

[17]Our protocol ensures that each party eventually receives these many encrypted shares (see below).

[18]In principle, the arithmetic circuit "re-compiler" technique by Genkin *et al.* [24] could also be used for this purpose, although it is not shown to work for π_{BKR} nor be constant-depth. In addition, the functionality of the re-compiler is richer, as it allows to restrict possible malicious strategies during the evaluation of the circuit, which is not needed here.

[19]USS is an adaptation of the bivariate-polynomial sharing technique [8, 33] to the asynchronous setting.

Protocol $\pi_{\text{A-SFE}}(\mathsf{Circ}, \mathcal{P})$: Code for p_i

First, mark all gates as *unevaluated*. Initialize empty variables z_ω and k_ω for every wire ω and m_ω for every output wire ω accessible to p_i. Initialize $\phi := 0$ (phase indicator). Then, proceed as follows:

- Upon first activation with input b, input b to $\mathcal{F}_{\text{A-SFE}}^{f_{\text{PREP}}^{\text{Circ}}}$.
- Upon later activations:

  - If $\phi = 0$, check if output from $\mathcal{F}_{\text{A-SFE}}^{f_{\text{PREP}}^{\text{Circ}}}$ received. If not, output (output) to $\mathcal{F}_{\text{A-SFE}}^{f_{\text{PREP}}^{\text{Circ}}}$ and become inactive. Otherwise, encrypt every gate g, with wires α, β, γ, as follows:

    1. Output by functionality includes:
       (a) Subkeys $k_{\alpha,0}^i$ and $k_{\alpha,1}^i$ as well as $k_{\beta,0}^i$ and $k_{\beta,1}^i$.
       (b) Function table share $[T_g]_i = ([t_g^{00}]_i, [t_g^{01}]_i, [t_g^{10}]_i, [t_g^{11}]_i)$.
    2. For $x, y \in \{0, 1\}$, compute $c_g^{xy,i} := \mathsf{Enc}_{k_{\alpha,x}^i, k_{\beta,y}^i}^{g,x,y}\left([t_g^{xy}]_i\right)$.
    3. Send $C_g^i := (c_g^{00,i}, c_g^{01,i}, c_g^{10,i}, c_g^{11,i})$ to all parties by invocation of $\mathcal{F}_{\text{A-SMT}}(p_i, p_i), j \in [n]$.

    Further, for all input wires ω, set z_ω and k_ω, to the values output by $\mathcal{F}_{\text{A-SFE}}^{f_{\text{PREP}}^{\text{Circ}}}$. Similarly, set the masks m_ω for the (accessible) output wires to the values output by $\mathcal{F}_{\text{A-SFE}}^{f_{\text{PREP}}^{\text{Circ}}}$. Set $\phi := 1$.

  - If $\phi = 1$, upon reception of any encryption, proceed as follows for every unevaluated gate g, with incoming wires α and β and outgoing wire γ:

    1. Let z_α, z_β, and z_γ be the masked bits and k_α, k_β, and k_γ the keys of the incoming wires α and β and of the outgoing wire γ. If z_α and z_β, are not defined yet, skip this gate; else:
       (a) For each ciphertext $C_g^j = (c_g^{00,j}, c_g^{01,j}, c_g^{10,j}, c_g^{11,j})$ from a party p_j, decrypt $t_g^{xy,j} := \mathsf{Dec}_{k_\alpha^j, k_\beta^j}^{g,x,y}\left(c_g^{xy,j}\right)$ for $x = z_\alpha$ and $y = z_\beta$, thereby recovering j^{th} shares of z_γ and of every entry of $k_\gamma = (k_\gamma^1, \ldots, k_\gamma^n)$.
       (b) Check if z_γ and the entries of k_γ can be safely computed by interpolation, i.e., if there are at least $2t + 1$ t-consistent shares for each value. If not, skip this gate. Otherwise, interpolate and mark g as *evaluated*.

    If all gates have been evaluated, output $z_\omega \oplus m_\omega$ for all (accessible) output wires ω.

Figure 5: Our constant-round asynchronous SFE protocol in the $\{\mathcal{F}_{\text{A-SFE}}^{f_{\text{PREP}}^{\text{Circ}}}, \mathcal{F}_{\text{A-SMT}}\}$-hybrid model

in parallel (i.e., receive inputs from all parties); this step finishes with every party holding a share of the input of each party p_i. As already mentioned, due to asynchrony, the inputs of some, up to t, (honest) parties might not be considered. The set Core of these parties whose inputs are considered (the so-called *core set* [9, 6]) is decided by π_{BKR} (and agreed upon by all parties) during the evaluation of the input gates, while the input of the parties not in the core set is set to a default value, in our case to 0 (i.e., a default USS of 0 is adopted as a sharing of these parties' inputs [9]). Once any party has agreed on the core set parties giving input, it goes on to the evaluation of the next gate (in fact, of all gates which are one level deeper in the circuit in parallel).

Our modification to π_{BKR} is as follows. For any party p_j, as soon as p_j has processed all input gates (i.e., holds shares of inputs of all parties in the core set and default shares for the remaining parties), and before any other gate of the arithmetic circuit is computed, p_j does the following: For each party we denote by x_i' the value that is (eventually) shared as p_i's input when all parties have evaluated the corresponding input gate, and denote by $[x_i']_j$ p_j's share of this value.[20]

Now, instead of continuing to process the original circuit $C_{f_{\text{PREP}}^{\text{Circ}}}^F$, we use the following trick from [10],

---

[20]By the USS property, at this point p_i is committed to x_i' but the adversary has no information on it, i.e., the adversary holds random shares of a USS of x_i.

which will allows us to enforce zero/one inputs. Each party uses the shared values x_i' to compute the circuit evaluating the following function: output $\vec{c} = (c_1, \ldots, c_n)$, where $c_i = x_i' - x_i'^2$ for each p_i. Each party that received the output $\vec{c}$ does the following:[21] For each p_i, if $c_i \neq 0$, then the parties replace the sharing of x_i' with a default sharing of 0. (That is, as soon as p_j receives the vector $\vec{c}$, for each i with $c_i \neq 0$ p_i replaces his share $[x_i]_j$ of x_i' with a default sharing of 0.) Once a party has completed this step (and replaced his local shares), he continues the execution of π_{BKR} with the modified shares to compute the remainder of the circuit $C^F_{f^{\mathsf{Circ}}_{\mathrm{PREP}}}$.

We denote the above modification of protocol π_{BKR} (in the $\mathcal{F}_{\text{A-BA}}(\mathcal{P})$-hybrid world where calls to A-BA are replaced by invocations of $\mathcal{F}_{\text{A-BA}}(\mathcal{P})$) with the above mechanism for enforcing inputs in $\{0, 1\}$ by $\pi^{0/1}_{\mathrm{BKR}}$. In order to evaluate the (Boolean) circuit for $f^{\mathsf{Circ}}_{\mathrm{PREP}}$, the parties execute $\pi^{0/1}_{\mathrm{BKR}}$ encoding their inputs and outputs with the following trivial encoding: An input-bit 0 (resp., 1) is encoded as the 0 (resp., 1) element in F, and output 0 (resp., 1) in F is decoded back to the bit 0 (resp., 1). The following lemma states the achieved security.

Lemma 2. *Protocol $\pi^{0/1}_{\mathrm{BKR}}$ for evaluating the circuit $C^F_{f^{\mathsf{Circ}}_{\mathrm{PREP}}}$ with the above trivial encoding UC-securely realizes $\mathcal{F}^{f^{\mathsf{Circ}}_{\mathrm{PREP}}}_{\text{A-SFE}}$.*

sketch. First note that if the inputs of all (honest and corrupted) parties are 0 or 1 (in the arithmetic field F), then the (decoded) output of $C^F_{f^{\mathsf{Circ}}_{\mathrm{PREP}}}$ is the the same as the output of the (Boolean) circuit for $f^{\mathsf{Circ}}_{\mathrm{PREP}}$ since all NAND gates with inputs $x, y \in \{0, 1\}$ are computed by $1 - xy$. Next, we argue that $\pi^{0/1}_{\mathrm{BKR}}$ forces the inputs of the adversary to be 0 or 1 and does not modify the inputs of honest parties. Indeed, an honest party p_i in the core set will share inputs $x_i' \in \{0, 1\}$ and therefore $c_i = 0$, which means that his input sharing is not modified by $\pi^{0/1}_{\mathrm{BKR}}$. The same holds for any corrupted party that shares $x_i' = 0$ or $x_i' = 1$. On the other hand, any corrupted party sharing a value other than 0 or 1 will result into an output $c_i \neq 0$ (since the non-zero elements in F form a multiplicative group of order $|F| - 1$) and its input will be set to 0.

Note that the eventual termination of π_{BKR} ensures that all parties will eventually receive the output vector c and will therefore resume the computation of the original circuit $C^F_{f^{\mathsf{Circ}}_{\mathrm{PREP}}}$, which (also due to the eventual termination of π_{BKR}) will terminate. The simulation of $\pi^{0/1}_{\mathrm{BKR}}$ is easily reduced to the simulation of π_{BKR}: The evaluation of the extra component that computes the c_i's can be easily simulated as they are random sharings of 0 for all honest parties in the core set, and for corrupted parties they are functions of the sharing of x_i' that the adversary creates in the input-processing phase, which for corrupted parties is fully simulatable. For the rest of the simulation, the simulator simply uses the π_{BKR} simulator. Thus the indistinguishability of the simulation follows from the security of π_{BKR}. □

Putting things together. The detailed description of protocol $\pi_{\text{A-SFE}}(\mathsf{Circ}, \mathcal{P})$ is presented in Figure 5. As already said, we describe the protocol in the $\mathcal{F}^{f^{\mathsf{Circ}}_{\mathrm{PREP}}}_{\text{A-SFE}}$-hybrid model, where $\mathcal{F}^{f^{\mathsf{Circ}}_{\mathrm{PREP}}}_{\text{A-SFE}}$ can be replaced with $\pi^{0/1}_{\mathrm{BKR}}$ using Lemma 2 and the universal composition theorem. At a high-level, the protocol proceeds as follows: In the first phase, the parties send their inputs to the functionality $\mathcal{F}^{f^{\mathsf{Circ}}_{\mathrm{PREP}}}_{\text{A-SFE}}$. The (randomized) function $f^{\mathsf{Circ}}_{\mathrm{PREP}}$ chooses the random masks, the subkeys, and computes Shamir sharings of the masked function tables (which are the values that need to be encrypted). Moreover, based on the inputs, it computes the masked value and the corresponding key of every input wire. The formal specification of $f^{\mathsf{Circ}}_{\mathrm{PREP}}$ can be found in Figure 4. The fact that $\mathcal{F}^{f^{\mathsf{Circ}}_{\mathrm{PREP}}}_{\text{A-SFE}}$ can be evaluated by a constant-depth circuit is illustrated in Figure 6 in Appendix C, which provides a diagram describing the structure of such a circuit. Each of the rectangles corresponds to a collection of independent constant-depth circuits that are evaluated in parallel.

---

[21] Observe that the eventual delivery property ensures that every party will eventually receive the output.

In the second phase of the protocol, as soon as a party receives output from $\mathcal{F}_{\text{A-SFE}}^{f_{\text{PREP}}^{\text{Circ}}}$, it encrypts all the shares obtained using the appropriate subkeys and sends the resulting ciphertexts to all parties, as shown in Figure 5. Then, it proceeds to locally evaluate the gates. For each gate, the party waits for ciphertexts from other parties and decrypts them. For a specific entry in the function table, the party has to wait until it has $2t + 1$ t-consistent shares of that entry (see again Figure 5).[22] Note that since all of the at least $2t + 1$ honest parties are guaranteed to obtain an output from $\mathcal{F}_{\text{A-SFE}}^{f_{\text{PREP}}^{\text{Circ}}}$, they will all properly encrypt their function tables and send out the resulting ciphertexts. Therefore, the wait for $2t + 1$ t-consistent shares is finite.

4.2 Analysis of the Protocol

Theorem 3. *Let* Circ *be a given boolean circuit and* f_{Circ} *be the n-party function computed by* Circ. *Protocol* $\pi_{\text{A-SFE}}(\text{Circ}, \mathcal{P})$ *securely realizes* $\mathcal{F}_{\text{A-SFE}}^{f_{\text{Circ}}}$ *in the* $\{\mathcal{F}_{\text{A-SFE}}^{f_{\text{PREP}}^{\text{Circ}}}, \mathcal{F}_{\text{A-SMT}}\}$-*hybrid model tolerating an adaptive adversary who corrupts up to $t < n/3$ of the parties and making black-box use of a PRF.*

A full proof of Theorem 3 can be found in Appendix B. Here we only provide a high-level sketch.

sketch. The output of each party from the evaluation of $\mathcal{F}_{\text{A-SFE}}^{f_{\text{PREP}}^{\text{Circ}}}$ contains (among other things) a t-out-of-n sharing of the garbled circuit for computing function f_{Circ}. After receiving the output from $\mathcal{F}_{\text{A-SFE}}^{f_{\text{PREP}}^{\text{Circ}}}$ and encrypting as described in Figure 5, the only time the parties have to wait is for the encryptions of $2t + 1$ t-consistent shares of garbled function-table entries from other parties. Since all of the at least $2t + 1$ honest parties are guaranteed to obtain an output from $\mathcal{F}_{\text{A-SFE}}^{f_{\text{PREP}}^{\text{Circ}}}$, they will all properly encrypt their function tables and send out the resulting ciphertexts at some point. Therefore, the wait for $2t + 1$ t-consistent shares is finite.

Moreover, the adversary cannot make an honest party accept a wrong value for any entry of the garbled gate: Observe that in any set of $2t + 1$ shares that a party receives, at least $t + 1$ are from honest parties. These $t + 1$ shares uniquely define the degree-t sharing polynomial F and, therefore, they can only be combined with correct shares (as output by $\mathcal{F}_{\text{A-SFE}}^{f_{\text{PREP}}^{\text{Circ}}}$). This implies that wrong shares sent by the adversary cannot make any honest party choose any other polynomial than F.

The simulator $\mathcal{S}$ for an adversary $\mathcal{A}$ roughly proceeds as follows: It emulates towards $\mathcal{A}$ the behavior of $\mathcal{F}_{\text{A-SFE}}^{f_{\text{PREP}}^{\text{Circ}}}$ and the channels $\mathcal{F}_{\text{A-SMT}}$. The security of the circuit-garbling technique and that of Shamir sharings allows $\mathcal{S}$ to perfectly simulate the entries of the garbled function tables that one would decrypt during a local evaluation of the garbled circuit, even *without* knowing the actual inputs. Moreover, the security of the double-key cipher ensures that the remaining entries are hidden, and can thus be replaced by dummy values (which can again be done without knowing the inputs) without causing a noticeable difference in the view of $\mathcal{A}$. $\qquad\qquad\square$

We now turn to the analysis of the protocol's round complexity. It is straightforward to verify that our protocol (assuming hybrid $\mathcal{F}_{\text{A-SFE}}^{f_{\text{PREP}}^{\text{Circ}}}$) needs only two communication rounds for each party p_i: one round in which p_i sends its inputs to the functionality and receives its outputs, and one round in which p_i sends all its corresponding encryptions and receives the encryptions of other parties. Moreover, the function $f_{\text{PREP}}^{\text{Circ}}$ can be represented by an arithmetic circuit Prep over a finite field $\mathbb{F}$ with constant (multiplicative) depth: The players first jointly generate the subkeys and the masks. A straightforward method for generating a random field element (such as the subkeys) is to take a random input from every party and computing the sum. Generating a random bit $b \in \{0, 1\} \subseteq \mathbb{F}$ can be reduced to generating random field elements as shown by Genkin *et al.* [24]. Given the masks and the subkeys, computing the function table and a Shamir sharing thereof can clearly be done in constant depth and, most importantly, in parallel for every gate.

Combining the above and Theorem 3 with Theorem 1 yields the following corollary:

---

[22]Note that using the Berlekamp-Welch algorithm, this can be achieved efficiently.

Corollary 4. *Let* Circ *be a given boolean circuit and* f_{Circ} *be the n-party function computed by* Circ. *There exists a constant-round protocol which securely realizes* $\mathcal{F}_{\mathrm{A\text{-}SFE}}^{f_{\mathsf{Circ}}}$ *in the* $\{\mathcal{F}_{\mathrm{A\text{-}BA}}, \mathcal{F}_{\mathrm{A\text{-}SMT}}\}$*-hybrid model tolerating an adaptive adversary who corrupts up to* $t < n/3$ *of the parties and making black-box use of a PRF.*

References

[1] Proceedings of the Twentieth Annual ACM Symposium on Theory of Computing, 2-4 May 1988, Chicago, Illinois, USA. ACM (1988)

[2] Asharov, G., Jain, A., López-Alt, A., Tromer, E., Vaikuntanathan, V., Wichs, D.: Multiparty computation with low communication, computation and interaction via threshold FHE. In: Pointcheval, D., Johansson, T. (eds.) EUROCRYPT 2012. LNCS, vol. 7237, pp. 483–501. Springer, Berlin, Germany, Cambridge, UK (Apr 15–19, 2012)

[3] Beaver, D., Micali, S., Rogaway, P.: The round complexity of secure protocols (extended abstract). In: STOC. pp. 503–513. ACM (1990)

[4] Beerliová-Trubíniová, Z., Hirt, M.: Simple and efficient perfectly-secure asynchronous mpc. In: Kurosawa, K. (ed.) ASIACRYPT. Lecture Notes in Computer Science, vol. 4833, pp. 376–392. Springer (2007)

[5] Bellare, M., Hoang, V.T., Rogaway, P.: Foundations of garbled circuits. In: the ACM Conference on Computer and Communications Security, CCS'12, Raleigh, NC, USA, October 16-18, 2012. pp. 784–796 (2012), http://doi.acm.org/10.1145/2382196.2382279

[6] Ben-Or, M., Canetti, R., Goldreich, O.: Asynchronous secure computation. In: STOC. pp. 52–61 (1993)

[7] Ben-Or, M., El-Yaniv, R.: Resilient-optimal interactive consistency in constant time. Distributed Computing 16(4), 249–262 (2003), http://dx.doi.org/10.1007/s00446-002-0083-3

[8] Ben-Or, M., Goldwasser, S., Wigderson, A.: Completeness theorems for non-cryptographic fault-tolerant distributed computation (extended abstract). In: STOC [1], pp. 1–10

[9] Ben-Or, M., Kelmer, B., Rabin, T.: Asynchronous secure computations with optimal resilience (extended abstract). In: PODC. pp. 183–192 (1994)

[10] Boneh, D., Goh, E.J., Nissim, K.: Evaluating 2-DNF formulas on ciphertexts. In: Kilian, J. (ed.) TCC 2005. LNCS, vol. 3378, pp. 325–341. Springer, Berlin, Germany, Cambridge, MA, USA (Feb 10–12, 2005)

[11] Bracha, G.: An asynchronou [(n-1)/3]-resilient consensus protocol. In: Probert, R.L., Lynch, N.A., Santoro, N. (eds.) 3rd ACM PODC. pp. 154–162. ACM Press, Vancouver, British Columbia, Canada (Aug 27–29, 1984)

[12] Cachin, C., Kursawe, K., Shoup, V.: Random oracles in Constantinople: Practical asynchronous byzantine agreement using cryptography. Journal of Cryptology 18(3), 219–246 (Jul 2005)

[13] Canetti, R.: Studies in Secure Multiparty Computation and Applications. Ph.D. thesis, Weizmann Institute of Technology, (http://www.wisdom.weizmann.ac.il/ oded/PSX/ran-phd.pdf (6 1995)

[14] Canetti, R.: Universally composable security: A new paradigm for cryptographic protocols. In: 42nd FOCS. pp. 136–145. IEEE Computer Society Press, Las Vegas, Nevada, USA (Oct 14–17, 2001)

[15] Canetti, R., Feige, U., Goldreich, O., Naor, M.: Adaptively secure multi-party computation. In: 28th ACM STOC. pp. 639–648. ACM Press, Philadephia, Pennsylvania, USA (May 22–24, 1996)

[16] Canetti, R., Rabin, T.: Fast asynchronous byzantine agreement with optimal resilience. In: 25th ACM STOC. pp. 42–51. ACM Press, San Diego, California, USA (May 16–18, 1993)

[17] Chaum, D., Crépeau, C., Damgård, I.: Multiparty unconditionally secure protocols (extended abstract). In: STOC [1], pp. 11–19

[18] Choi, S.G., Katz, J., Malozemoff, A.J., Zikas, V.: Efficient three-party computation from cut-and-choose. Cryptology ePrint Archive, Report 2014/128 (2014), `http://eprint.iacr.org/2014/128`

[19] Damgård, I., Ishai, Y.: Constant-round multiparty computation using a black-box pseudorandom generator. In: Shoup, V. (ed.) CRYPTO. Lecture Notes in Computer Science, vol. 3621, pp. 378–394. Springer (2005)

[20] Dolev, D., Strong, H.R.: Authenticated algorithms for byzantine agreement. SIAM J. Comput. 12(4), 656–666 (1983), `http://dx.doi.org/10.1137/0212045`

[21] Feldman, P., Micali, S.: Optimal algorithms for byzantine agreement. In: 20th ACM STOC. pp. 148–161. ACM Press, Chicago, Illinois, USA (May 2–4, 1988)

[22] Fischer, M.J., Lynch, N.A.: A lower bound for the time to assure interactive consistency. Inf. Process. Lett. 14(4), 183–186 (1982), `http://dx.doi.org/10.1016/0020-0190(82)90033-3`

[23] Fischer, M.J., Lynch, N.A., Paterson, M.: Impossibility of distributed consensus with one faulty process. In: Fagin, R., Bernstein, P.A. (eds.) Proceedings of the Second ACM SIGACT-SIGMOD Symposium on Principles of Database Systems, March 21-23, 1983, Colony Square Hotel, Atlanta, Georgia, USA. pp. 1–7. ACM (1983), `http://doi.acm.org/10.1145/588058.588060`

[24] Genkin, D., Ishai, Y., Prabhakaran, M., Sahai, A., Tromer, E.: Circuits resilient to additive attacks with applications to secure computation. In: Symposium on Theory of Computing, STOC 2014, New York, NY, USA, May 31 - June 03, 2014. pp. 495–504 (2014), `http://doi.acm.org/10.1145/2591796.2591861`

[25] Goldreich, O.: Foundations of Cryptography: Basic Applications, vol. 2. Cambridge University Press, Cambridge, UK (2004)

[26] Goldreich, O., Micali, S., Wigderson, A.: How to play any mental game or a completeness theorem for protocols with honest majority. In: STOC. pp. 218–229. ACM (1987)

[27] Hirt, M., Nielsen, J.B., Przydatek, B.: Cryptographic asynchronous multi-party computation with optimal resilience (extended abstract). In: Cramer, R. (ed.) EUROCRYPT. Lecture Notes in Computer Science, vol. 3494, pp. 322–340. Springer (2005)

[28] Hirt, M., Nielsen, J.B., Przydatek, B.: Asynchronous multi-party computation with quadratic communication. In: Aceto, L., Damgård, I., Goldberg, L.A., Halldórsson, M.M., Ingólfsdóttir, A., Walukiewicz, I. (eds.) ICALP (2). Lecture Notes in Computer Science, vol. 5126, pp. 473–485. Springer (2008)

[29] Hirt, M., Zikas, V.: Adaptively secure broadcast. In: Gilbert, H. (ed.) EUROCRYPT 2010. LNCS, vol. 6110, pp. 466–485. Springer, Berlin, Germany, French Riviera (May 30 – Jun 3, 2010)

[30] Katz, J., Koo, C.Y.: On expected constant-round protocols for byzantine agreement. In: Dwork, C. (ed.) CRYPTO 2006. LNCS, vol. 4117, pp. 445–462. Springer, Berlin, Germany, Santa Barbara, CA, USA (Aug 20–24, 2006)

[31] Katz, J., Maurer, U., Tackmann, B., Zikas, V.: Universally composable synchronous computation. In: Sahai, A. (ed.) TCC 2013. LNCS, vol. 7785, pp. 477–498. Springer, Berlin, Germany, Tokyo, Japan (Mar 3–6, 2013)

[32] Pfitzmann, B., Waidner, M.: Unconditional byzantine agreement for any number of faulty processors. In: Finkel, A., Jantzen, M. (eds.) STACS 92, 9th Annual Symposium on Theoretical Aspects of Computer Science, Cachan, France, February 13-15, 1992, Proceedings. Lecture Notes in Computer Science, vol. 577, pp. 339–350. Springer (1992), `http://dx.doi.org/10.1007/3-540-55210-3_195`

[33] Rabin, T., Ben-Or, M.: Verifiable secret sharing and multiparty protocols with honest majority (extended abstract). In: 21st ACM STOC. pp. 73–85. ACM Press, Seattle, Washington, USA (May 15–17, 1989)

[34] Schneider, T., Zohner, M.: GMW vs. Yao? efficient secure two-party computation with low depth circuits. In: Sadeghi, A.R. (ed.) FC 2013. LNCS, vol. 7859, pp. 275–292. Springer, Berlin, Germany, Okinawa, Japan (Apr 1–5, 2013)

[35] Shamir, A.: How to share a secret. Commun. ACM 22(11), 612–613 (1979)

[36] Yao, A.C.C.: Protocols for secure computations (extended abstract). In: FOCS. pp. 160–164. IEEE (1982)

[37] Yao, A.C.C.: How to generate and exchange secrets (extended abstract). In: FOCS. pp. 162–167. IEEE (1986)

A   History and Related Work (cont'd)

Here we provide a fuller account of related work and put our results in perspective. To give a more complete picture, we start by discussing the development of MPC protocols in the synchronous setting, and then contrast it with the development in the asynchronous setting. Along the way we also discuss the tools (e.g., setup assumptions and communication resources) that are used in each setting.

Starting with Yao's seminal paper [36], which introduced the problem of MPC and provided the first solution, a long line of interesting results proved feasibility bounds for *synchronous* networks in various adversarial settings. Goldreich, Micali, and Wigderson [26, 25] proved that under computational assumptions (the existence of enhanced trapdoor permutations), any n-party function can be securely computed if and only if up to $t < n$ parties are corrupted passively or up to $t < n/2$ actively. Corresponding bounds for information-theoretic security were shown by Ben-Or, Goldwasser, and Wigderson [8], who proved that perfect security is possible if and only if the adversary corrupts up to $t < n/2$ parties passively or up to $t < n/3$ actively. Similar bounds where concurrently proved by Chaum, Crépeau, and Damgård [17] for the case where a negligible error-probability is allowed and were later improved by Rabin and Ben-Or [33] to achieve optimal resiliency $t < n/2$.

The above works assume point-to-point secure communication and a broadcast channel and, under these assumptions, are secure even against an adaptive adversary [15]. However, in [26, 25] both these resources can be implemented with adaptive security assuming a public-key infrastructure and non-committing encrpytion [15, 29]. Similarly, the broadcast channel in [8, 17, 33] can be emulated by an adaptively secure broadcast protocol [29].[23] The round complexity of all the above protocols in the malicious multi-party setting—even with the assumption of a broadcast channel—is linear in the multiplicative depth of the arithmetic circuit corresponding to the function the parties aim to compute.

Beaver, Micali, and Rogaway [3] were the first to provide a constant-round MPC protocol in the synchronous stand-alone model. Their protocol is secure in the computational setting and tolerates an adaptive adversary who actively corrupts up to $t < n/2$ parties. The complexity of [3] was improved

---

[23]Because [33] tolerates even $n/3 \leq t < n/2$ corrupted parties, the emulation of broadcast would require an additional setup of information-theoretic pseudo-signatures [32].

by Damgård and Ishai [19], who provided the first constant-round protocol making black-box use of the underlying cryptographic primitive (a pseudo-random generator). Importantly, both [3] and [19] assume a broadcast channel, an assumption essential for obtaining constant-round MPC. Indeed, as proved in [22, 20], it is impossible to implement such a broadcast channel from point-to-point communication in a constant number of rounds, and although expected constant-round broadcast protocols exist in the literature (e.g., [21, 30]), using them to instantiate calls within the constructions of [3] or [19] would not yield an expected constant-round protocol [7]. The intuitive reason—formally argued by Ben-Or and El-Yaniv [7]—is that the process of running n such broadcast protocols (even in parallel) does not terminate in an expected constant number of rounds.

The model of *asynchronous* communication with eventual delivery was considered early on in seminal works on fault-tolerant distributed computing (e.g., [23]). The study of optimally resilient MPC in such an asynchronous network was initiated by Ben-Or, Canetti, and Goldreich [6], who proved that any function can be computed by a perfectly secure asynchronous protocol if and only if at most $t < n/4$ parties are corrupted. Following that result, Ben-Or, Kelmer, and Rabin [9] showed that if a negligible error probability is allowed, the bound $t < n/3$ is necessary and sufficient for asynchronous MPC.[24] More recently, Hirt *et al.* [27, 28] provided computationally secure solutions (i.e., protocols tolerating a computationally bounded adversary) and Beerliová and Hirt [4] perfectly secure solutions with improved communication complexity.

The above asynchronous protocols are secure if one assumes point-to-point communication and an A-BA protocol. Similarly to their synchronous counterparts, all the above protocols—even assuming an A-BA primitive—have round complexity linear in the multiplicative depth of the arithmetic circuit that computes the function, as they follow the standard gate-by-gate evaluation paradigm.

We note in passing that although in the synchronous setting BA implies broadcast, this is not the case in the asynchronous setting. Indeed, Canetti and Rabin [16] provide an asynchronous BA protocol tolerating $t < n/3$ malicious parties, which if every honest party terminates at the latest after a poly-logarithmic number of rounds, securely implements asynchronous BA except with negligible probability. A broadcast protocol with similar guarantees is provably impossible [23], and existence of an asynchronous BA protocol which terminates in a strict constant number of rounds would contradict the impossibility from [22, 20]. Similarly to the synchronous case, although solutions for asynchronous BA with expected constant number of rounds exist [12, 16], using them in the above asynchronous protocol to replace invocations to asynchronous BA would not yield an expected constant-round MPC protocol [7].[25]

Finally, if one gives up the requirement that the broadcast protocol (eventually) terminates when the sender is corrupted (this results in a primitive known as *A-Cast* [11]), then one can implement it even in a constant number of rounds. (In fact, A-Cast can be easily reduced to asynchronous BA by having the sender send his input to all parties, who then forward this input as soon as it is received to the asynchronous BA primitive).

B Full Security Proof

Theorem 3. *Let* Circ *be a given boolean circuit and* f_{Circ} *be the n-party function computed by* Circ. *Protocol* $\pi_{\text{A-SFE}}(\mathsf{Circ}, \mathcal{P})$ *securely realizes* $\mathcal{F}^{f_{\mathsf{Circ}}}_{\text{A-SFE}}$ *in the* $\{\mathcal{F}^{f^{\mathsf{Circ}}_{\text{PREP}}}_{\text{A-SFE}}, \mathcal{F}_{\text{A-SMT}}\}$*-hybrid model tolerating an adaptive adversary who corrupts up to* $t < n/3$ *of the parties and making black-box use of a PRF.*

---

[24]The necessity of the $t < n/3$ bound follows from the result by Canetti *et al.* [6, 13], who argue that this bound is necessary for fail-stop adversaries; it also applies to computational security and assuming A-BA. Moreover, note that in the asynchronous setting, all feasibility bounds are worse by an additive term of t compared to the synchronous setting. Intuitively, this stems from the fact that honest parties cannot distinguish between messages by other honest parties being delayed and messages by corrupted parties not being sent. Thus, in particular, perfectly secure asynchronous MPC is possible only if $t < n/4$.

[25]Nonetheless, [7] does describe an alternative way of obtaining several asynchronous BA protocols that are guaranteed to all terminate in expected constant number of rounds.

Proof. We prove the security of the protocol in the setting where the parties have hybrid access to the asynchronous secure channels $\mathcal{F}_{\text{A-SMT}}$ and a functionality $\mathcal{F}_{\text{A-SFE}}^{f_{\text{PREP}}^{\text{Circ}}}$ for computing the function $f_{\text{PREP}}^{\text{Circ}}$ (Figure 4). The security of the protocol follows then from the universal composition theorem, by replacing the ideal computation of $f_{\text{PREP}}^{\text{Circ}}$ with the asynchronous MPC protocol from [9].

First we argue correctness of the protocol's output and guaranteed termination. The output of each party from the evaluation of $\mathcal{F}_{\text{A-SFE}}^{f_{\text{PREP}}^{\text{Circ}}}$ contains (among other things) a t-out-of-n sharing of the garbled circuit for computing function f_{Circ}. After receiving the output from $\mathcal{F}_{\text{A-SFE}}^{f_{\text{PREP}}^{\text{Circ}}}$ and encrypting as described in Figure 5, the only time the parties have to wait is for the encryptions of $2t+1$ t-consistent shares of function-table entries from other parties. Since all of the at least $2t + 1$ honest parties are guaranteed to obtain an output from $\mathcal{F}_{\text{A-SFE}}^{f_{\text{PREP}}^{\text{Circ}}}$, they will all properly encrypt their function tables and send out the resulting ciphertexts at some point. Therefore, the wait for $2t + 1$ t-consistent shares is finite.

Moreover, the adversary cannot make an honest party accept a wrong value for any entry of the garbled gate: Observe that in any $2t + 1$ shares that a party receives, at least $t + 1$ have to be from honest parties. These $t + 1$ shares uniquely define the degree-t sharing polynomial F and, therefore, they can only be combined with correct shares (as output by $\mathcal{F}_{\text{A-SFE}}^{f_{\text{PREP}}^{\text{Circ}}}$). This implies that wrong shares sent by the adversary cannot make any honest party choose any other polynomial than F.

The simulator $\mathcal{S}$. Let $\mathcal{A}$ be an adversary attacking the protocol. We will use ω_{j_ℓ} to denote the ℓth input wire of p_j. The simulator $\mathcal{S}$ uses $\mathcal{A}$ in a straight-line black-box manner where it emulates towards $\mathcal{A}$ the behavior of $\mathcal{F}_{\text{A-SFE}}^{f_{\text{PREP}}^{\text{Circ}}}$ and all $\mathcal{F}_{\text{A-SMT}}$ channels. Moreover, the simulator keeps for each honest party p_i a simulated view for p_i, which is initially empty. The core component of the simulation lies in simulating the values that the adversary sees of a garbled circuit. However, $\mathcal{S}$ does not know the inputs of honest parties. Hence, $\mathcal{S}$ has to come up with a sharing of a "fake" circuit which is indistinguishable from the original circuit. We handle this as follows: $\mathcal{S}$ uses 0 for each of the honest parties' input bits. While this will cause the output to be different than in the real execution (where the correct inputs are used) $\mathcal{S}$ can correct this *by modifying the masks of the output gates of the circuit* to correspond to the correct output. The security of the encryption scheme guarantees that these modified selection bits do not provide the adversary with any distinguishing advantage. The details of the simulation follow (w.l.o.g. assume that the input wires are labeled as $1, \ldots, N$):

- For each (simulated) honest party p_i, the simulator initializes p_i's phase indicator $\tilde{\phi}_i := 0$. $\mathcal{S}$ also sets $\tilde{\mathcal{H}}$ to be the set of honest parties ($\tilde{\mathcal{H}}$ is updated with any new corruption requests) and initializes $\tilde{\mathcal{P}}' := \emptyset$.

- First $\mathcal{S}$ internally emulates an invocation of $\mathcal{F}_{\text{A-SFE}}^{f_{\text{PREP}}^{\text{Circ}}}$. When a corrupted party p_j hands input b for input-wire ω_{j_ℓ}, if $\tilde{\phi}_i = 0$ for all $i \in [n]$, then the simulator sets $\tilde{b}_{\omega_{j_\ell}} = b$ and hands $\tilde{b}_{\omega_{j_\ell}}$ to its ideal functionality $\mathcal{F}_{\text{A-SFE}}^{f_{\text{Circ}}}$; otherwise, $\mathcal{S}$ ignores the new input. If $\mathcal{A}$ requests to corrupt a new party p_i, then $\mathcal{S}$ corrupts this party, and if p_i has already received input $b_{\omega_{i_\ell}}$ for any wire ω_{i_ℓ} from the environment, then $\mathcal{S}$ hands $b_{\omega_{i_\ell}}$ to $\mathcal{A}$.

- Any delay $\mathcal{A}$ sends to $\mathcal{S}$ for $\mathcal{F}_{\text{A-SFE}}^{f_{\text{PREP}}^{\text{Circ}}}$, $\mathcal{S}$ takes it into account in its internal emulation of $\mathcal{F}_{\text{A-SFE}}^{f_{\text{PREP}}^{\text{Circ}}}$. Any delay D_i (for an honest party p_i) which $\mathcal{A}$ sends to $\mathcal{S}$ for delaying a secure channel $\mathcal{F}_{\text{A-SMT}}(p_j, p_i)$, the simulator takes it into account in the simulation of this channel.

- If, at any point in the simulation of $\mathcal{F}_{\text{A-SFE}}^{f_{\text{PREP}}^{\text{Circ}}}$, $\mathcal{A}$ sends $(\texttt{NoInput}, \mathcal{P}')$ to $\mathcal{S}$, then if $|\mathcal{P}'| \leq |\mathcal{P} \setminus \mathcal{H}|$ and $\tilde{\phi}_i := 0$ for all $i \in [n]$, the simulator sets $\tilde{\mathcal{P}}' := \mathcal{P}'$ and sends $(\texttt{NoInput}, \tilde{\mathcal{P}}')$ to its own ideal functionality $\mathcal{F}_{\text{A-SFE}}^{f_{\text{PREP}}^{\text{Circ}}}$ (otherwise this message is ignored).

- At the point where the simulated execution of $\mathcal{F}_{\text{A-SFE}}^{f_{\text{PREP}}^{\text{Circ}}}$ generates the first output towards a (simulated) honest party or the adversary, $\mathcal{S}$ completes the emulation of the internal state of $\mathcal{F}_{\text{A-SFE}}^{f_{\text{PREP}}^{\text{Circ}}}$ as follows: (Observe that $\mathcal{F}_{\text{A-SFE}}^{f_{\text{Circ}}}$, i.e., the actual ideal functionality we are realizing, informs the simulation whenever a new input or an ignored fetch-request is issued by by some honest party; hence, because the simulator knows the exact delays the adversary imposes on the delivery of $\mathcal{F}_{\text{A-SFE}}^{f_{\text{PREP}}^{\text{Circ}}}$, it can simulate the point in which $\mathcal{F}_{\text{A-SFE}}^{f_{\text{PREP}}^{\text{Circ}}}$ will generate output.)

  - For every input wire ω_{i_ℓ} of any (still) honest party p_i, $\mathcal{S}$ sets the corresponding (simulated) input to $\tilde{b}_{\omega_{i_\ell}} := 0$.

  - For every input wire ω_{i_ℓ} of any party $p_i := \tilde{\mathcal{P}}'$, $\mathcal{S}$ (re)sets the corresponding (simulated) input to a default value $\tilde{b}_{\omega_{i_\ell}} := 0$.

  - For every circuit-wire ω, the simulator chooses all the sub-keys and masks $\tilde{m}_\omega$ uniformly at random, as $\mathcal{F}_{\text{A-SFE}}^{f_{\text{PREP}}^{\text{Circ}}}$ would.

  - Furthermore, $\mathcal{S}$ evaluates $f_{\text{PREP}}^{\text{Circ}}$ on the chosen inputs. For each party p_i let $o_{i_1}, \ldots, o_{i_{L_i}}$ denote the labels of the output wires associated to p_i and let $\tilde{z}_{o_{i_\ell}}$ denote the corresponding masked output-bit.

  - $\mathcal{S}$ hands its own functionality $\mathcal{F}_{\text{A-SFE}}^{f_{\text{Circ}}}$ the inputs of corrupted parties and provides enough `fetch` requests to receive the outputs of corrupted parties. For each corrupted party p_i, let $y_{o_{i_1}}, \ldots, y_{o_{i_{Q_i}}}$ denote the outputs that $\mathcal{F}_{\text{A-SFE}}^{f_{\text{Circ}}}$ generates for p_i; for each $\ell \in [Q_i]$, $\mathcal{S}$ (re)sets $\tilde{m}_{o_{i_\ell}} := \tilde{z}_{o_{i_\ell}} \oplus y_{o_{i_\ell}}$; furthermore, $\mathcal{S}$ sets $\tilde{\phi}_i := 1$.

  Analogous to our protocol notation, for every party p_i we denote p_i's simulated output from $\mathcal{F}_{\text{A-SFE}}^{f_{\text{PREP}}^{\text{Circ}}}$ as follows (note that the simulated output of honest parties exists only in the simulator's head):

  1. Subkeys $\tilde{k}_{\alpha,0}^i$ and $\tilde{k}_{\alpha,1}^i$ as well as $\tilde{k}_{\beta,0}^i$ and $\tilde{k}_{\beta,1}^i$.
  2. Function table share $[\tilde{T}_g]_i = ([\tilde{t}_g^{00}]_i, [\tilde{t}_g^{01}]_i, [\tilde{t}_g^{10}]_i, [\tilde{t}_g^{11}]_i)$.
  3. For every output wire o_{i_ℓ} of p_i: the mask $\tilde{m}_{o_{i_\ell}}$ if p_i is to learn that output.
  4. (same for every party) For every *input* wire $\omega \in [N]$: the masked value $\tilde{z}_\omega$ and the corresponding key $\tilde{k}_\omega$

- As soon as in the simulation any (simulated) honest party p_i received his output, the simulator adds this output to his simulated view of p_i, sets $\tilde{\phi}_i := 1$ for all $(x,y) \in \{0,1\}^2$ and computes $\tilde{c}_g^{xy,i} := \text{Enc}_{\tilde{k}_{\alpha,x}^i, \tilde{k}_{\beta,y}^i}^{g,x,y}\left([\tilde{t}_g^{xy}]_i\right)$ for each $(x,y) \in \{0,1\}$. Subsequently, $\mathcal{S}$ computes $\tilde{C}_g^i := (\tilde{c}_g^{00,i}, \tilde{c}_g^{01,i}, \tilde{c}_g^{10,i}, \tilde{c}_g^{11,i})$ and emulates sending $\tilde{C}_g^i$ to all parties (towards the adversary).

- For every (simulated) honest party for which $\tilde{\phi}_i = 1$ and who (in the simulation) receives a new encryption (from another simulated honest party or from the adversary on behalf of corrupted parties), the simulator does the following for each gate g, with wires α, β, γ:

  1. Let $\tilde{z}_\alpha, \tilde{z}_\beta$, and $\tilde{z}_\gamma$, be the masked bits and $\tilde{k}_\alpha, \tilde{k}_\beta$, and $\tilde{k}_\gamma$ the keys of the wires α and β, and γ, respectively. If in the simulated view of p_i $\tilde{k}_\alpha$ and $\tilde{k}_\beta$ are not defined yet, skip this gate; else

     (a) $\mathcal{S}$ keeps track of the number of simulated encryption-shares for g that are delivered to p_i by simulated *honest* parties; denote this number by $\tilde{n}_i^g$. Observe that the simulator can do that, as he is simulating (as above) the point when any honest party sends an encryption to another honest party, and he also knows, by its interaction with $\mathcal{A}$, the accumulated delay. In addition, the simulator keeps track of the number $\tilde{\rho}_i^g$ of *correct*

encryption shares corresponding to adversarial senders ($\mathcal{S}$ marks an encryption share that $\mathcal{A}$ sends as *correct* if it decrypts to the correct plaintext given the corresponding key—the correct plaintext and key are known to $\mathcal{S}$ as they are part of $\mathcal{A}$'s output in the simulated $\mathcal{F}_{\text{A-SFE}}^{f_{\text{PREP}}^{\text{Circ}}}$). As soon as $\tilde{n}_i^g + \tilde{\rho}_i^g \geq 2t + 1$, $\mathcal{S}$ includes the simulated $\tilde{z}_\gamma$ and of $\tilde{k}_\gamma = (k_\gamma^1, \ldots, k_\gamma^n)$.

2. If for all the outgoing wires of output gates corresponding to p_i the key and and mask have been set, then $\mathcal{S}$ instructs the ideal functionality $\mathcal{F}_{\text{A-SFE}}^{f_{\text{Circ}}}$ to deliver p_i's output in the next request by sending an appropriate negative delay so that the accumulated delay becomes 0.

- Any output of the adversary to its environment is forwarded by $\mathcal{S}$ to the environment. Similarly, any message of $\mathcal{Z}$ for the adversary is forwarded by $\mathcal{S}$ to $\mathcal{A}$.

This completes the description of the simulator.

In the following we argue that $\mathcal{S}$ is a good simulator. Towards this direction we follow the idea of the proof from [18]. Concretely, we define the following hybrid experiments HYBRID$_1$ and HYBRID$_2$ as follows:

- HYBRID$_1$ is the experiment where the simulation behave as above, but is given oracle access to the inputs of honest parties and uses them in the emulation of the hybrid $\mathcal{F}_{\text{A-SFE}}^{f_{\text{PREP}}^{\text{Circ}}}$.

- HYBRID$_2$ is similar to HYBRID$_1$ with the difference that instead of counting the correct encryption-shares that a simulated honest party p_i with $\tilde{\phi}_i$ would receive, the simulator uses the actual code of $\tilde{\phi}_i$ for updating its state.

Lemma 5. *The ideal experiment* IDEAL *is indistinguishable from the hybrid experiment* HYBRID$_1$.

sketch. The information seen by the adversary in HYBRID$_1$ is identical as in an experiment where the two party semi-honest protocol by Choi et al. [18, Figure 3] is executed between an honest P_1 (circuit generator) who plays on behalf of honest parties, generates the circuit, and hands it to the adversary, and an adversaria P_2. The strategy of our simulator for simulating this P_1 is identical to the simulation strategy from [18, Theorem 3] with the only difference being the extra timing consideration. Thus by a similar series of hybrids as in [18, Theorem 3] we can prove that HYBRID$_1$ is indistinguishable from the ideal experiment IDEAL. The sequence of hybrids and formal argument is nearly identical to the ones use in the proof in [18, Theorem 3] and is therefore not repeated. $\square$

Lemma 6. *The hybrid experiment* HYBRID$_1$ *is indistinguishable from the hybrid experiment* HYBRID$_2$.

Proof. To prove this statement it suffices to prove that the simulator allows delivery of the output to honest parties at the same time (indeed, any other information exchanged in the protocol is identical in the two hybrids. To this direction we observe that a gate is evaluated by p_i in the protocol if and only p_i receives $2t + 1$ decryptions lying in the correct sharing polynomial. However, all (emulated) decryptions from honest parties will lie on the same sharing polynomial F (of degree t) that $\mathcal{S}$ chose when emulating $\mathcal{F}_{\text{A-SFE}}^{f_{\text{PREP}}^{\text{Circ}}}$. Because $t + 1$ out of the $2t + 1$ shares that p_i awaits for have to be from honest parties (since there are at most t corrupted parties) the set of $2t + 1$ parties whose share is taken into account in the reconstruction must consist of either honest parties or corrupted parties who sent correct encryptions. Thus in HYBRID$_2$ the simulator will consider any gate evaluated as soon as corresponding correct encryptions from n honest and n' corrupted parties with $n + n' = 2t + 1$ arrive which is the same rule as the one used in HYBRID$_1$. $\square$

Lemma 7. *The hybrid experiment* HYBRID$_2$ *is indistinguishable from the real experiment* REAL.

Proof. In HYBRID$_2$ the outputs from the emulation of $\mathcal{F}_{\text{A-SFE}}^{f_{\text{PREP}}^{\text{Circ}}}$ are distributed identically as the outputs from the invocation of $\mathcal{F}_{\text{A-SFE}}^{f_{\text{PREP}}^{\text{Circ}}}$ in the real experiment REAL (since the same inputs are used). Furthermore, the simulator faithfully executes the instructions of any honest p_i in his simulation. Hence HYBRID$_2$ and REAL are identically distributed. $\qquad\square$
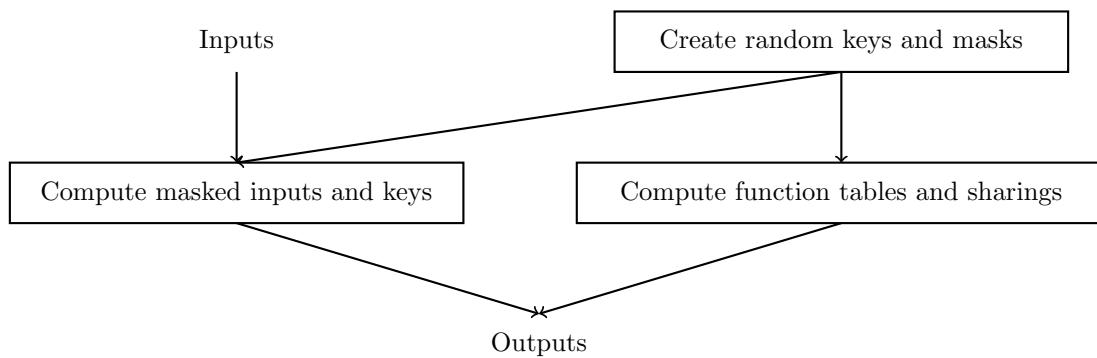
$\hfill\square$

C Figures

Figure 6: Bird's-eye view of the arithmetic circuit Prep computing function $f_{\mathrm{PREP}}^{\mathsf{Circ}}$. Each box represents a constant-depth circuit.