YOSO: You Only Speak Once
Secure MPC with Stateless Ephemeral Roles

Craig Gentry[1], Shai Halevi[1], Hugo Krawczyk[1], Bernardo Magri[2], Jesper Buus Nielsen[2⋆], Tal Rabin[1,3], and Sophia Yakoubov[4⋆⋆]

[1] Algorand Foundation
[2] Concordium Blockchain Research Center, Aarhus University
[3] UPenn, USA
[4] Aarhus University, Denmark

Abstract. The inherent difficulty of maintaining stateful environments over long periods of time gave rise to the paradigm of *serverless computing*, where mostly stateless components are deployed on demand to handle computation tasks, and are torn down once their task is complete. Serverless architecture could offer the added benefit of improved resistance to targeted denial-of-service attacks, by hiding from the attacker the physical machines involved in the protocol until after they complete their work. Realizing such protection, however, requires that the protocol only uses stateless parties, where each party sends only one message and never needs to speaks again. Perhaps the most famous example of this style of protocols is the Nakamoto consensus protocol used in Bitcoin: A peer can win the right to produce the next block by running a local lottery (mining) while staying covert. Once the right has been won, it is executed by sending a *single* message. After that, the physical entity never needs to send more messages.

We refer to this as the You-Only-Speak-Once (YOSO) property, and initiate the formal study of it within a new model that we call the YOSO model. Our model is centered around the notion of *roles*, which are stateless parties that can only send a single message. Crucially, our modelling separates the protocol design, that only uses roles, from the role-assignment mechanism, that assigns roles to actual physical entities. This separation enables studying these two aspects separately, and our YOSO model in this work only deals with the protocol-design aspect.

We describe several techniques for achieving YOSO MPC; both computational and information theoretic. Our protocols are synchronous and provide guaranteed output delivery (which is important for application domains such as blockchains), assuming honest majority of roles in every time step. We describe a practically efficient computationally-secure

⋆ Partially funded by The Concordium Foundation; The Danish Independent Research Council under Grant-ID DFF-8021-00366B (BETHE); The Carlsberg Foundation under the Semper Ardens Research Project CF18-112 (BCM).

⋆⋆ Funded by the European Research Council (ERC) under the European Unions's Horizon 2020 research and innovation programme under grant agreement No 669255 (MPCPRO).

protocol, as well as a proof-of-concept information theoretically secure protocol.

Keywords. Blockchains, Secure MPC, Stateless Parties, YOSO.

1 Introduction

A somewhat surprising feature of our networked world is just how hard it is to keep a working stateful execution environment over long periods of time. Even in non-adversarial settings, it is a major challenge to keep a server operational and connected through software updates, local physical events, and global infrastructure interruptions. This becomes even harder in adversarial environments. Consider for example a network adversary targeting a specific protocol, watching the communication network and mounting a targeted denial of service (DoS) attack on any machine that sends a message in this protocol. In high-stake environments, one also must worry about near-instant malicious compromise, unleashed by well equipped adversaries with a stash of zero-day exploits.

One approach for mitigating this issue is the paradigm of *serverless computing*, where mostly-stateless components are deployed on demand to handle computation tasks, and are torn down once their task is complete. In addition to economic benefits, a protocol built from such components could offer better resistance against strong adversaries by hiding the physical machines that play a role in the protocol, until after they complete their work and send their messages. To realize this protection, however, the protocol must utilize only stateless components, making it harder to design.

Perhaps the best-known example of this style of protocol is the Nakamoto consensus protocol used in Bitcoin [19]. A salient property of the Bitcoin design is that a peer can win the right to produce the next block by running a local lottery (mining), while staying covert. Once the right has been won, it is executed by sending a *single* message. After that, the physical entity never needs to send another message. Another example is the Algorand consensus protocol [8] with its player-replaceability property.

In this work we initiate a formal study of protocols of this style, which we refer to as You-Only-Speak-Once (YOSO). An important conceptual contribution of our work is the (relatively) clean modeling of such protocols, centered around their use of *roles* (which is the name we use for those one-time stateless parties). Crucially, our modeling separates the protocol design using roles from the role-assignment functionality that assigns the roles to actual physical machines.

This separation lets us study the protocol design problem on its own, freeing us from having to specify the role-assignment implementation which is necessarily very system dependent: a proof-of-work blockchain will have very different role-assignment mechanisms from a proof-of-stake blockchain, and a traditional cloud environment will use yet different mechanisms. However, all these systems could use the same protocol for secure computation once the roles have been properly assigned. On the technical side we make the following contributions:

– We present a formal model for defining and studying such protocols, called the YOSO model, which in particular codifies the separation between role-assignment and protocol execution and formally defines the notion of only speaking once. The YOSO model is cast within the UC framework [5] and therefore can draw on the existing body of research on UC security. An overview of the model is provided in Section 2. For a more detailed treatment see the full version of the paper [14].
– We also devise tools for working in the YOSO model, and describe two different secure MPC protocols. Our main solution presented in Section 3 is an information theoretic proof-of-concept protocol that provides statistical security.[5] Additionally, in the full version [14] we also describe a computationally-secure protocol. Both protocols are synchronous and provide guaranteed output delivery (which is important for our application domain), assuming an honest majority of roles in every protocol step.
– We show that an information theoretic secure YOSO MPC can be compiled into a natural UC secure protocol running on a toy model of a blockchain with role assignment. This is meant as a sanity check of the abstract role-based YOSO model. It shows that protocols developed in this model can indeed be compiled to practice. We show that if we start with a static-secure (analogously, adaptive-secure) YOSO protocol, we can get a static-secure (analogously, adaptive-secure) UC protocol with essentially the same corruption threshold.

1.1 The YOSO Model

We introduce the YOSO model to make it easy to start studying YOSO MPC independently of blockchain and role assignment.

Role-based computation In the YOSO model, participants in protocols are called *roles* rather than parties or nodes or machines. The reason for the name "roles" is that we usually think of these one-time parties as playing some role in a protocol. Some examples of roles include "Party #3 in the 2nd VSS protocol on the 8th round", "the prover in the 6th NIZK", etc. Formally, a role is just a stateless party that can only send a single message before it is destroyed, and a protocol is an interaction between roles. Throughout this manuscript we use the following terminology:

Roles: are abstract formal entities that perform the protocol actions and communicate with other roles.

Nodes/Machines: refer to stateful long-living entities that the adversary can identify and target for corruption. These can be physical or virtual machines, that would typically have some identifying characteristics such as an IP address that can be used by the adversary to attack them.

[5] As we explain below, the restrictions of working in the YOSO model are so severe that a priory it was not clear to us that information-theoretical security is even possible in the "$2t + 1$ regime". Indeed this work began as an attempt to prove that no such protocols exist.

We sometimes use the term *parties*, but only in informal discussions and in contexts where the distinction between roles and machines is immaterial.

Importantly, roles are detached from machines, and mapping of machines to roles happens at execution time. A protocol in the YOSO model will inevitably be executed alongside a role-assignment functionality, and the security of the protocol will rely on the guarantees provided by that functionality. Ideally, this assignment should be unknown to the attacker until after the machine plays its role and sends a message, hence limiting the adversary's ability to target the role for corruption.

The YOSO model can be used with different role-assignment functionalities with different guarantees. In this work we mainly consider a simple random-assignment functionality: it assigns each role to a random machine from among a universe of available ones, and hides that assignment from the adversary (unless the chosen machine is already corrupted). An adversary that corrupts machines will therefore be unable to predict which roles will be corrupted; upon corruption of a machine the adversary will be handed the random roles that are mapped to that machine. This allows for a simplified view of the adversary where all corruptions are random.

1.2 MPC in the YOSO Model

A compelling motivation for these protocols is scalable computation in the presence of an adaptive fail-stop adversary (a powerful DoS adversary, as noted earlier). Imagine a large number — perhaps millions — of nodes that want to engage in a secure computation in the presence of such an adversary. Assuming that the DoS adversary cannot take down more than some threshold of the nodes, then running an MPC protocol among all of them would yield the desired result. However, running classical MPC protocols among a large number of nodes is expensive. All of the nodes typically need to communicate with all of their peers, creating a prohibitive communication load. YOSO MPC enables the computation to be run by a small subset of the nodes, with an independent subset — or committee — participating in every round. YOSO MPC thwarts an adaptive DoS adversary because the adversary is unable to predict which fail-stops will be useful to foil the security; thus it creates the opportunity for execution of the protocol with small committees resulting in communication that is sub-linear in the number of nodes in the network.

As a more concrete example of a scenario where such scalable computation would be necessary, consider "MPC as a service". That is, an outsourced computation service where clients submit inputs for a joint computation so that the privacy of the inputs and the correctness of the output are guaranteed, even if a fraction of the provider's servers are adversarially controlled. However, while full corruption of servers is expensive, dedicated denial of service against targeted servers is an easier attack to carry out, and the protocol should be able to withstand it. YOSO MPC offers a solution that remains secure under these realistic conditions.

Role Assignment for YOSO MPC. In order to reap the benefits of such scalable YOSO MPC, it is important to assign YOSO MPC roles to machines in a scalable way without revealing the role assignment before the roles need to speak. Furthermore, the assigned machines should be able to receive secret messages (even while the message senders do not know their identities). This is challenging since, being able to speak only once, the machine having won a role cannot first make a public key available, and then receive messages and execute its role in the protocol. This would involve speaking at least twice.

One solution that was recently proposed by Benhamouda *et al.* [3] involves the use of nominating committees: each machine has a public key for an encryption scheme allowing the rerandomization of public keys. For each role R there will be a delegator role D. (We call R the delegate, and D the delegator.) First a machine is assigned a delegator role D using, e.g., cryptographic sortition (or just by solving some puzzle). Then the delegator D will pick, uniformly at random, another machine to play the delegate role R. It will take that machine's public key pk_i, rerandomize it into $\widetilde{pk_i}$, and publish $\widetilde{pk_i}$. Note that $\widetilde{pk_i}$ does not reveal the identity of the machine that was assigned to R; however, it enables other roles to send secret messages to the delegate R by encrypting to $\widetilde{pk_i}$. Finally, the delegate R will execute the role.

One drawback of this approach is that the role R will be corrupt if the delegator is corrupt *or* if the delegate is corrupt. This essentially doubles the corruption budget of the adversary. It is an interesting research direction to develop more practical and more secure role assignment mechanisms. However, this is orthogonal to the design of MPC protocols which will be run by the roles, which is the focus of our work. In the full version [14] we give a toy example of compiling a YOSO protocol to run on top of a blockchain with role assignment to illuminate this compelling use case.

Parameters of YOSO MPC Protocols. When designing a YOSO MPC protocol there is a number of interesting parameters to consider. In addition to the many "generic" aspects of MPC (such as corruption type and threshold, hardness assumptions, trusted setup, security guarantees, etc.) YOSO MPC protocols have some new parameters in their design.

– *Future/Past Horizon:* When a role speaks, it may send private messages to roles intended to speak in future rounds. The future horizon describes how far into the future a role may need to speak (similarly past horizon is how far back a role may need to listen). The method of assigning roles impacts and is impacted by the future and past horizons and should be taken into consideration. For example, for proof-of-stake systems it is undesirable to assign roles in advance using the current stake distribution. Or if roles are assigned on the fly parties would need to read the history of communication far into the past. One should therefore try to use as short a future/past horizon as possible.
– *Dynamic and Static Execution Time:* Static execution time refers to the ability to know ahead of time when a role would speak in the protocol,

contrasted with the dynamic case where the time to speak is only determined at run-time. As YOSO protocols are ideal for serverless architectures where servers are only running when they need to act, static execution time may save resources (e.g. cloud rental).

A related distinction (in the dynamic case) is whether only the role itself can determine when it is going to speak, or whether it can be determined publicly. (This could make a difference, e.g., in agreement protocols that must accumulate enough votes before moving to the next phase, where we may want to know if we still need to wait for the vote from the role or can we assume that it crashed and will never vote.)

YOSO MPC from Additive Homomorphic Threshold Encryption. Our first technical contribution is a YOSO MPC protocol in the computational setting with guaranteed output delivery in a synchronous model, tolerating a dishonest minority of roles at any given round. Specifically, in every round we will have some number n of roles that will form an honest-majority committee. As stated, it falls to the role-assignment functionality to supply us with committees with honest majority; in this work we allow ourselves to just assume that we have them.

Given a supply of committees with honest majority, our construction is based on the CDN protocol [10]. Informally, CDN requires a system-wide public key pk for an additively homomorphic threshold encryption scheme, where the secret key sk is shared among the committee members (with each member i holding sk_i). The participants then perform the entire computation using additive homomorphism, interspersed with public decryption of masked intermediate values. The protocol uses Beaver triples that are generated on-the-fly to support multiplications; the secret key shares are used to open values in every round of Beaver triple use, and to obtain the computation output at the end.

We note that CDN is already almost a YOSO protocol: the only state the participants need is the secret key shares sk_i, and the only messages that they send are their decryption shares (with the ciphertexts all being public). Providing the participants with shares of the global secret key sk can be done, e.g., using the proactive handover protocol of Benhamouda *et al.* [3], which is a YOSO protocol. In each protocol round, committee members get their decryption shares, and then the committee decrypts the current batch of ciphertexts and reshares sk to the next committee.

To get a YOSO protocol, we also need to generate the Beaver triples YOSO-style. We will use two committees — C_A and C_B — to generate many triples of the form $\big(\mathsf{Enc}(a), \mathsf{Enc}(b), \mathsf{Enc}(ab)\big)$, which will be consumed by future committees during multiplications. We first have members P_i of committee C_A individually choose random a_i's and publish the ciphertexts $\overline{a_i} = \mathsf{Enc}(a_i)$ along with NIZK proofs that these are valid ciphertexts. All parties can use additive homomorphism to obtain $\overline{a}$, an encryption of the sum a of the a_i's. Then members P_j of committee C_B will individually choose random b_j's and set $\overline{b_j} = \mathsf{Enc}(b_j)$, then use additive homomorphism to compute $\overline{c_j}$, an encryption of $b_j a$. P_j then publishes $(\overline{b_j}, \overline{c_j})$, along with proofs that they were generated properly. All parties can use additive homomorphism to obtain $\overline{b}$ and $\overline{c}$, encryptions of the sums b of the b_j's and c of the $b_j a$'s, respectively. $(\overline{a}, \overline{b}, \overline{c})$ form a Beaver triple. Note that as long as all the NIZK proofs are valid and there is at least one honest party in each committee C_A, C_B, the triple is indeed a Beaver triple for the values $a = \sum_i a_i$ and $b = \sum_j b_j$ which are unknown to the adversary.[6]

We describe the complete CDN-based protocol Π_{CDN}, and prove its security, in the full version [14] . For now, we state the following informal theorem.

Theorem. (informal) *Any multiparty function F can be securely implemented by the CDN YOSO protocol in a synchronous network with authenticated broadcast channel, resilient against a fraction $\tau < 1/2$ of random Byzantine corruptions.*

We note that another approach for achieving computational security would be to leverage *fully* homomorphic encryption (FHE). This requires an FHE scheme with a one-message threshold decryption procedure, and also one whose secret key could be maintained proactively using a YOSO protocol. Proactive maintenance of the secret key can be achieved, e.g., using the YOSO handover protocol of Benhamouda *et al.* [3], and one-round decryption can be achieved using the techniques from Asharov *et al.* [1] and Mukherjee-Wichs [18] (after a one-time trusted setup to generate the required evaluation key). In terms of complexity, an FHE-based solution may be more efficient in number of rounds and total communication, but it requires much more local computation, more per-round communication, and a more complicated trusted setup.

YOSO MPC from Information Theoretic Techniques. Our second (and main) technical contribution is a proof-of-concept information theoretic YOSO protocol with guaranteed output delivery in a synchronous model, tolerating any dishonest minority of roles at any given committee. This protocol does not need any trusted setup, but it relies on secure point-to-point channels between roles,[7] as well as a totally-ordered broadcast. One consequence of this protocol is statistically unbiased coin-flip in the YOSO model, which (together with appropriate role-assignment) implies unbiased public randomness in public blockchains via a YOSO protocol.

We begin by observing that YOSO is easy in the semi-honest model, in fact semi-honest BGW [2] is basically already a YOSO protocol. The BGW protocol only uses secret sharing and reconstruction: secret sharing can be done to a future committee (instead of the current one) over point-to-point channels, and reconstruction can be done publicly. When implementing a circuit, each

[6] If we have many honest parties in C_A, C_B (say m of them in each committee), then we can improve efficiency and get $\Omega(m)$ triples at roughly the same bandwidth using standard techniques.

[7] We note again that such secure point-to-point channels would have to be implemented somehow, even though the receiving role may not have been assigned yet to a machine. This task falls to the role-assignment functionality, which we do not specify in this work.

multiplication gate has two committees, one for each round in the multiplication protocol. For a gate with large fan-out, the gate committee will reshare their shares to the committees of all the downstream gates.

It is only when switching to the malicious model that things get hard, as YOSO seems to rule out many common information-theoretic techniques. In particular, patterns such as "committing" to a value and then being challenged on it, or even just using the same secret value in many parts of the protocol, seem to inherently require a party to stick around and speak more than once. The same can be said for cut-and-choose techniques that have a party generating multiple values, being challenged to open (say) half of them, and if they are all valid then the other half is used in the protocol.

It is also easy to see that simplistic solutions such as one party sending all its secret state to another will not help: It would allow the adversary to get this secret value if either the sender or the receiver are corrupted, hence amplifying the adversary's power. A more promising avenue is to let a party share its secret state with future committees (maybe more than one), and have these committees emulate it in the future as needed. However, ensuring that a message from one party is recoverable intact by future committees is challenging; this is essentially a verifiable-secret-sharing (VSS) functionality. Ensuring that the party shares *the same message* to multiple committees poses more challenges still. In Section 3 we address these challenges by gradually developing stronger and stronger primitives that build on each other. Here we just give a hint for some of the observations that enable these tools, and the various steps that go into the construction.

Step 1, Future Broadcast (FBcast*).* In Section 3.2 we describe a Future Broadcast construction that enables a party to prepare a message that should be sent in a future round. This may be complicated in general, since we need to ensure that the message delivered in the future is in fact the message of the party creating it, the kind of authenticity often requires VSS. But in our context we observe that we only need to ensure this authenticity for messages of honest parties, as faulty parties can say whatever they want at any time. Hence, for the FBcast primitive we can assume an honest dealer, which makes the design a lot easier.

Observe that in the computational setting this is straightforward to achieve. A party shares its value using a Shamir secret sharing and also provides every share holder with a digital signature on the share. When the value is reconstructed only shares with valid signatures are taken into the interpolation, if they all lie on a degree-t polynomial then the constant term is taken as the broadcasted message. In the IT setting we show that if the dealer is honest, information theoretic MACs are sufficient to replace digital signatures in this construction.

Step 2, Distributed Commitment (DC*).* In this contruction we want to offer some guarantees for reconstructing a value at a later time also in the case when the dealer is faulty. DC enables a dealer to commit in a distribute manner to a value and at a later time either open the committed value or null. This is exactly the functionality of a commitment in the computational setting, but it is achieved in the IT distributed setting.

To deliver DC we fortify the IT MACs into IT signatures (IT-SIG). An IT-SIG offers a holder of the signature on a value some assurances that in fact the value will be verified when presented. Our techniques build on the VSS interactive tools of Rabin and Ben-Or [22] adjusted to the YOSO model. We transform the IT-SIG from [22] into one where a party knows in advance all the messages that it may need to send in the future. This makes it possible to replace the multiple speaking rounds in the original protocol, by having each party share its future messages using FBcast (Section 3.3). The IT-SIGs provide enough of the digital signature properties for the purpose of realizing distributed commitments (Section 3.4).

*Step 3, Duplicate DC (*DupDC*) and VSS.* Proceeding towards VSS, we again turn to Rabin and Ben-Or [22], who utilize DC to achieve VSS via a cut-and-choose proof. The complication in using in the YOSO model is that in this proof one value needs to be used multiple times. In the YOSO model, this requires creating duplicates of the same committed value, each to be used in a different step of the proof. Letting the dealer run multiple DC's does not work as the dealer might be faulty and share different values. Thus, we would need the dealer to prove that all the committed values are the same. This will create a problem because for the proof to go through the committee holding the sharing would need to talk. Once they talk they have exhausted their one opportunity to speak and now the duplicate of the value has been wasted. Thus, we need to create a mechanism that duplicates values without "wasting" them. Surprisingly, we observe that our DC protocol allows the share holders themselves to create duplicates of the commitment. This avoids the need for additional proofs, the committee of shareholders is mostly honest so all the duplicates will be the same by design (see Section 3.5). Here, yet again, we can make all elements of the proof public, thus informing all parties of the result of the computation. This enables us to finalize the design of the VSS (Section 3.6).

To eventually complete the design of the MPC we would also need duplicates of the VSS as the same value might go into multiple gates and the committee holding the value can only speak once. Luckily, we can derive the duplicates of the VSS directly from the duplicates of the DC.

*Step 4, Augmented VSS (*AugVSS*).* We need one more level of sharing which we call Augmented VSS. In this level of sharing we add the property that not only is a secret s shared via VSS but also that all the shares that define the sharing of s are VSSed. This will enable the MPC.

Step 5, Secure-MPC. Once we have AugVSS, getting information-theoretic secure-MPC can be done using standard techniques that need to be adapted to the YOSO model. We maintain the variant throughout the computation that the values on the wires are AugVSS. Hence we prove:

Theorem. (informal) *Any multiparty function F can be securely implemented by an information-theoretic YOSO protocol in a synchronous network with broadcast and secure point-to-point channels, resilient against a fraction $\tau < 1/2$ of random Byzantine corruptions. The protocol additionally tolerates any number of chosen, Byzantine corruptions of input roles and output roles.*

It is crucial, for practical purposes, that we can tolerate chosen corruptions of input roles and output roles. Often the inputs and outputs are given by known clients that could more easily be targeted by an attack.

Epilogue, Public Randomness. The cut-and-choose protocols in our design are described using access to public randomness (which defines the challenges in those protocols). But where can we get this public randomness? Producing true randomness in a distributed setting seems to require MPC, creating a circular problem. Yet, we can show that our protocols remain secure when using *unpredictable* (high min-entropy) values, rather than truly random ones. Producing public unpredictable values in the honest-majority setting is much easier, and can even be done in a YOSO fashion. Thus, we can complete the MPC without the need for true randomness.

Of course, once we are able to get full-blown MPC, we can use it to produce completely uniform public randomness. This in particular solves the problem of obtaining public uniform randomness on a public blockchain using a YOSO protocol, a problem that was explored by a few previous works [6,7].

On the impossibility of Garay et al. [12]. In [12] it was shown that any protocol in the information theoretic model with a sublinear message complexity (in the number of parties) cannot withstand adaptive corruptions of a fraction equal or greater than $1 - \sqrt{0.5}$ of the total number of parties. Yet, we claim that our IT protocol can withstand less than $n/2$ adaptive corruptions. This is not a contradiction. Our proof proceeds in two steps. In the first we prove that our IT protocol is adaptively secure without the assumption of sublinear message complexity. In the second part, when we prove the protocol that has sublinear message complexity, we need to combine our IT protocol with some role-assignment mechanism. This inevitably takes our protocol out of the IT model, making the lower bound of [12] not applicable.[8]

YOSO can be Realized. Our YOSO protocols are abstract in that they only consider abstract roles; we abstract away role assignment and machines. To show that protocols designed in our abstract YOSO model can be used in practice, we show how to compile these asbtract protocols into concrete protocols that use physical machines, assuming an underlying role-assignment service. To that end, we define a simple UC functionality $\mathcal{F}_{\mathrm{RA}}$, modeling a system with role

[8] Specifically, the implementation of our communication channels which are needed to enable the solution can only be achieved in the computational setting (in our specific case we assume a PKI and more).

assignment: That functionality "spits out" a sequence of random public keys, where the corresponding secret key is known by a random, secret node in the system.

Assuming access to this role-assignment functionality, in addition to a broadcast channel and point-to-point channels *between physical machines in the system* (modeled as ideal functionalities $\mathcal{F}_{\mathrm{BC}}$, $\mathcal{F}_{\mathrm{SPP}}$), we show how to compile any abstract protocol Π in the YOSO model into a concrete protocol in the UC hybrid model with functionalities $\mathcal{F}_{\mathrm{RA}}$, $\mathcal{F}_{\mathrm{BC}}$, and $\mathcal{F}_{\mathrm{SPP}}$. (These functionalities can then be implemented using an underlying blockchain, e.g., as described in [3].)

We prove two results: (1) We show that an abstract YOSO protocol Π that IT YOSO-implements a secure function evaluation of F against t *random*, static corruptions, can be compiled using hybrid functionalities $\mathcal{F}_{\mathrm{BC}}$ and $\mathcal{F}_{\mathrm{SPP}}$ into a UC secure protocol Π' for the $\mathcal{F}_{\mathrm{RA}}$-hybrid model that tolerates ρ *chosen*, static corruptions for any $\rho < t$. (2) We show the same for adaptive security.

We can get security against chosen corruptions from security against random corruptions because the adversary does not know the role-to-machine association chosen by $\mathcal{F}_{\mathrm{RA}}$. Intuitively, corrupting a machine just corrupts random roles.

1.3 Related Work

Protocols built out of ephemeral one-time roles became popular over the last decade with the emergence of public blockchains, whose defining feature is not relying on long-term participants with fixed identities. In particular, starting with Nakamoto's consensus protocol [19], these protocols became popular for achieving agreement in different settings, e.g., [17,20,8,4].

Only very recently did we start seeing attempts at using this style of protocols for other cryptographic tasks: Benhamouda *et al.* [3] described how to use such protocols for long-term maintenance of secrets on public blockchains, and mentioned the possibility of using these secrets for various tasks, including for general-purpose secure computation. Blum *et al.* [4] described how to implement input-free protocols in this model (such as coin tossing), and also described informally an FHE-based solution for functions with input (similar to the one sketched in Section 1.2 above).

Choudhuri *et al.* [9] described general-purpose secure-MPC protocols of this style (that they call *fluid*), where the participants need to volunteer for roles (in our terminology we would call it a volunteer-based role-assignment functionality). Such protocols can be tweaked and casted as YOSO protocols with a volunteer-based role assignment. However, the protocols of [9] only guarantee security with abort, making their use extremely fragile as a single corruption can abort the protocol. Moreover, volunteer-based role assignment seems susceptible to an adversary filling the volunteering parties with faulty parties by volunteering many times.

2 YOSO for the Working Cryptographer

The YOSO model can be cast within the UC framework [5] by identifying the roles in YOSO protocols with the party identifiers of the UC framework. This means that the roles are executed by the UC model, which completely abstracts away how these roles are actually assigned to physical machines; in fact, there is not even a notion of physical machines left. We then introduce a notion of random corruptions that are out of the control of the adversary. This can be used to model a set of roles which, in the now abstracted away real world, are hidden inside random physical machines, and the adversary can corrupt machines of its choosing.

Below we always use the term roles rather than parties, just to stress that we are in the YOSO model. This terminology is for didactic purposes only; a role in our formal model is identical to a party in the normal UC framework. The "speak once" aspect is enforced by our execution model, as we now explain.

2.1 YOSO Wrappers

To force roles to only speak once, we are explicitly "yosofying" them with a YOSO wrapper. Namely, our execution model postulates a wrapper around each role, that kills it immediately after the first time that it speaks. When that happens, the wrapper sends a Spoke token to the environment, the adversary and all its sub-routines (sub-protocols and ideal functionalities). Thereafter it responds with a Spoke token to the environment whenever activated, and only sends Spoke to the sub-routines that it is connected to.

Defining what it means for a role to "speak for the first time" is somewhat nontrivial. The main issue to tackle is whether sending messages to functionalities constitute speaking. To see the issue, consider a protocol Π (that implements some functionality $\mathcal{F}$), in which a role R must listen for many incoming messages before deciding to send a message. In this case, the $\mathcal{F}$-hybrid model could have the role R sending its input to $\mathcal{F}$ very early, but the implementation would have R actually speaking much later.

To account for that, we let functionalities reply to parties with the special Spoke token. The functionality can freely choose when to send this token, and the YOSO wrapper will kill the role as soon as it receives a Spoke token from any functionality. For example, a communication-channel functionality will reply with a Spoke token as soon as a party sends anything on it, while a higher-level functionality may trigger a Spoke token based on some input from the adversary. Note that when a communication channel outputs Spoke to a role, the role will pass it on to all its sub-routines and then its environment/outer protocol. Hence the entire composed role will be crashed.

We denote the "yosofied" role R by $\mathsf{YoS}(R)$, and the protocol that we get by yosofying all the roles in Π is denoted by $\mathsf{YoS}(\Pi)$.

2.2 Random Corruptions

In addition to the usual corruptions of the UC model we also model random corruptions in the YOSO model — that is, corruptions out of the control of the adversary.

We do this without changing the UC framework itself. Recall that in UC a corruption is implemented by the adversary just writing $(\texttt{corrupt}, cp)$ on the backdoor tape of the party, where cp is some auxiliary information like the type of corruption: Byzantine, semi-honest, *et cetera.* There is no explicit mechanism in UC for limiting how many parties are corrupted or with which flavor. However, we often choose to analyze protocols under a restricted set of corruptions. This is simple to do by only quantifying over adversaries adhering to this restriction. This is easy to formulate for settings like "only semi-honest corruptions" or "at most a minority of the parties". However, it seems to be trickier for random corruptions: if the adversary corrupts a role R, how can we know that R was chosen at random? We need a precise meaning for this in order to be able to make precise security claims. For this purpose, we introduce a simple notion called the corruption controller ($\mathcal{CC}$), that runs as part of the environment. If an adversary wants to do a random corruption, it asks the environment, which will pass the request to the $\mathcal{CC}$. Then, the $\mathcal{CC}$ will sample the corruption and inform the adversary which role was corrupted (via the environment). If the environment sees the adversary is not respecting the decision of the $\mathcal{CC}$, then the environment will make a random guess in the security game. This enforces that no distinguishing advantage comes from executions violating the will of the $\mathcal{CC}$. We then only prove security under the class of environments having such a $\mathcal{CC}$ and using it as intended. We call this the class of controlled environments.

These random corruptions can be mixed freely with other corruption types, but it is illustrative to consider a generalization of the usual adversary structures to random corruptions. We codify the corruption power of the adversary by means of a *corruption structure.*

Let Role be the set of (names of) roles in the system. A corruption structure on Role is a set of probability distributions over $2^{|\mathsf{Role}|}$. A static adversary would choose at the beginning of the execution a specific corruption distribution $C \in \mathcal{C}$ and give it to the $\mathcal{CC}$ via the environment. Then the $\mathcal{CC}$ samples $c \leftarrow C$ and give it to the adversary via the environment, and each role $\mathsf{R} \in \mathsf{Role}$ can now be corrupted if $\mathsf{R} \in c$. Note that a corruption structure with only point distributions (i.e. with a single probability-one pattern $c \in C$) corresponds exactly to standard static corruptions with these allowed patterns, coinciding with the notion of general adversary structure of Hirt and Maurer [15]. We stress that corruption structure represents our *assumption* about the corruption power of the adversary when designing the protocol. It is up to the role-assignment functionality to ensure that realistic adversaries will be unlikely to exceed this power.

When considering adaptive corruptions several choices are possible. We consider two in this work called sample corruptions and point corruptions. In sample corruptions the adversary gives a distribution on a set of roles and gets one of them corrupted, within some bound. In point corruptions the adversary can ask permission to corrupt a given role with some limited probability. If the corruption fails the role stays honest forever after. It is interesting future work to explore the relation between different notions of random corruptions.

2.3 YOSO Security

The notion of a protocol realizing a functionality is borrowed from the UC model. Namely, we say that Π YOSO-realizes (implements) $\mathcal{F}$ for some class of environments (possibly using random corruptions) if $\mathsf{YoS}(\Pi)$ UC-realizes $\mathcal{F}$. The considered class of environments should be a subset of the controlled environments.

It is easy to see that UC composition still holds for controlled environments. If an environment is composed with a protocol or simulator to define a new environment, as happens in the proof of the UC theorem, then this composed environment still uses the $\mathcal{CC}$ of the original one. The same holds when one composes an environment with a simulator. Therefore we get UC composition also for controlled environments.

YOSO composition then follows directly from UC composition. Let Π be a protocol for the $\mathcal{G}$-hybrid model and assume that Π YOSO-realises $\mathcal{F}$. Assume that Γ YOSO-realises $\mathcal{G}$. As usual in the UC framework let $\Pi^{\mathcal{G}\to\Gamma}$ be the protocol Π with calls to $\mathcal{G}$ replaced by calls to Γ. It follows that $\Pi^{\mathcal{G}\to\Gamma}$ YOSO-realises $\mathcal{F}$. To see this, note that the premises give us that $\mathsf{YoS}(\Pi)$ UC-realises $\mathcal{F}$ and that $\mathsf{YoS}(\Gamma)$ UC-realises $\mathcal{G}$. By the usual UC theorem we get that $\mathsf{YoS}(\Pi)^{\mathcal{G}\to\mathsf{YoS}(\Gamma)}$ UC-realises $\mathcal{F}$. Then use that by construction $\mathsf{YoS}(\Pi)^{\mathcal{G}\to\mathsf{YoS}(\Gamma)} = \mathsf{YoS}(\Pi^{\mathcal{G}\to\Gamma})$. This follows by the way the YoS wrapper passes around the SPOKE token to shut down entire composed parties.

2.4 Common Features, Functionalities, and Models

Synchrony. To simplify the treatment of synchronous clocks, we assume that in every round the environment sends a TICK message to all the roles *and also to all the functionalities* and the adversary, in addition to any other inputs that it wants to provide them. We use the model in [16] for this.

Communication Channels and PKI. We assume an authenticated broadcast channel denoted $\mathcal{F}_{\mathsf{BC}}$, and usually also secure point-to-point channels $\mathcal{F}_{\mathsf{SPP}}$ (or at least authenticated channels $\mathcal{F}_{\mathsf{PP}}$). These functionalities are defined more or less as usual in the UC framework, except that in our case they return a SPOKE token to any role immediately in the step following the receipt of message from it.[9] These functionalities are formally presented in the full version [14]. We also sometimes use a PKI functionality, which is specified in Figure 1.

YOSO Secure Function Evaluation. We consider secure function evaluation in the YOSO model. We assume that the roles of a protocol Π are divided into

[9] We allow a role to send messages on multiple channels in the same step, then it will receive SPOKE tokens from all of them in the next step.

On the first input $\textsc{Tick}$ sample $(pk_{\mathsf{R}}, \mathsf{sk}_{\mathsf{R}}) \leftarrow \mathsf{Gen}$ for all $\mathsf{R} \in \mathsf{Correct} \cup \mathsf{Crash}$. Output pk to $\mathcal{O}$. For all $\mathsf{R} \in \mathsf{Leaky}$ output sk_{R} to $\mathcal{O}$. For each $\mathsf{R} \in \mathsf{Malicious}$ query $\mathcal{O}$ to get the keys $(pk_{\mathsf{R}}, \mathsf{sk}_{\mathsf{R}})$ for R. Then for each $\mathsf{R} \in \mathsf{Correct}$ output $(\mathsf{sk}_{\mathsf{R}}, \{\ pk_{\mathsf{R}'}\}_{\mathsf{R}' \in \mathsf{Role}})$ to R.

Fig. 1: The ideal functionality $\mathcal{F}_{\mathsf{Gen}}$ for a very simple PKI setup with key generator Gen.

input roles, output roles and computation roles. The input roles receive inputs from the environment and the output roles will deliver the outputs back. The computation nodes carry out intermediary steps of the computation and do not interact with the environment.

As usual for UC-like models, to formulate the assertion that a function F could be computed securely we need to wrap that function by a compatible functionality $\mathcal{F}_{\mathrm{MPC}}^{F}$, as described in[14]. Importantly, we assume that the roles receiving the output *do not speak in an implementation* (so $\mathcal{F}_{\mathrm{MPC}}^{F}$ never sends $\textsc{Spoke}$ tokens to the output roles). Otherwise these output roles would not be able to contribute the result to the higher-level protocol.

By default, we assume that the roles receiving the inputs and the roles giving the outputs can be corrupted using the usual chosen corruptions. This is reasonable since in most of the meaningful high-level protocols, like elections, the inputs to the protocol are given by known machines that might be subject to targeted DoS attacks. Computation nodes however, are only subject to random corruptions; when running in the "real world" with a concrete role assignment mechanism, we get to execute computation roles on random machines.

We then say that Π YOSO securely implements F with a fraction τ random corruptions if Π implements $\mathcal{F}_{\mathrm{MPC}}^{F}$ against any number of chosen corruptions of input roles and output roles and random corruptions of up to a fraction τ of the computation roles.

The IT YOSO Model. We define the standard IT YOSO model to be the model with broadcast and secure point-to-point channels, unbounded environments, and poly-time protocols, ideal functionalities and simulators.

The Computational YOSO Model. The computational YOSO model is equipped with an authenticated broadcast channel, perhaps authenticated point-to-point channels, a PKI functionality (such as the one from Figure 1), and poly-time environments, protocols, ideal functionalities and simulators.

3 The Information-Theoretic $t < \frac{n}{2}$ MPC Protocol

In this section we describe an MPC protocol in the information theoretic YOSO model for a fraction $\tau < 1/2$ of random Byzantine corruptions.

Theorem 1. *For any multiparty function F, there exists a poly-time protocol Π described below running with the network $(\mathcal{F}_{\mathsf{BC}}, \mathcal{F}_{\mathsf{SPP}})$ which YOSO-realizes* *the ideal functionality $\mathcal{F}_{\mathrm{MPC}}^{F}$ in the information theoretic YOSO model. The protocol tolerates any number of chosen, Byzantine corruptions of input roles and output roles, and for any $\tau < 1/2$ it tolerates adaptive, Byzantine, random τ-point-corruptions of computation nodes.*

Recall that the reason we allow chosen corruptions of input roles and output roles is that in a real-life setting we cannot reasonably assume that it is unknown which machines will give input or get the outputs. So input and output roles could be targeted. On the other hand, we want to model that computation roles are run on random, secret machines, so we only allow random corruptions of computation nodes. Recall that τ-point corruptions just means that the adversary can point to a role R and ask for a corruption. Then the role is made corrupted with probability τ, and with probability $1-\tau$ it will remain honest forever after. The type of random corruption it not essential for our proof. The reason why we prove security against point corruptions is that this is the type of corruption needed for the compilation result shown in the full version [14].

Below we will phrase the protocol in terms of disjoint *committees* of size n. We call the roles in a committee *parties*. Let c be the number of committees that we need. We then start with $N = cn$ computation roles $\mathsf{R}_1, \ldots, \mathsf{R}_N$. We call the committees $\mathsf{C}_1, \ldots, \mathsf{C}_c$ where $\mathsf{C}_j = \{\mathsf{P}_1^j, \ldots, \mathsf{P}_n^j\}$ and $\mathsf{P}_i^j = \mathsf{R}_{i+(j-1)n}$. We call P_i^j *party i in committee j*. Notice that this grouping of roles into committees is static. This does not affect security as the adversary cannot bias corruption towards a specific committee. Each party is still subject only to τ-point corruptions. If we set $\tau < 1/2$ then we can clearly pick n large enough that we can conclude from a tail bound that all committees have at most $t < n/2$ corrupted parties except with negligible probability. For the rest of the section we then assume that this has been done. From this point on the only assumption we need for security is that each committee has $t < n/2$ corrupted parties.

Note that we allow any number of corruptions among input roles and output roles. However, input roles and output roles are not part of committees, so this does not violate the honest majority assumption for committees.

Our protocol is adaptively secure. We will, however, below mainly prove static security and only briefly discuss adaptive security. The reason is that for point corruptions, the distinction between adaptive corruptions and static corruptions is minimal. An adaptive point corruption just means that the adversary chooses to be oblivious to whether a party is corrupt or not until the point corruption. This gives it no new powers over static corruptions. Note, in particular, that corruption control component $\mathcal{CC}$ could sample before the UC execution starts for each role R_i a bit b_i which is 1 with probability τ. If later the adversary does a point corruption of R_i it will become corrupted if and only if $b_i = 1$. Therefore, even in the adaptive case, the corruptions can be thought of as being static: they were chosen before the execution started. The only complication in proving adaptive security compared to proving static security is then that in the adaptive case, the simulator will not know b_i until the adversary does a point corruption of R_i. Below we phrase the proof in terms of static security. The proof can be adapted to the adaptive case using standard techniques.

The challenge in designing an information-theoretic MPC protocol in the YOSO model is in replacing the actions of parties that interact and speak multiple times in regular MPC protocols with parties (more precisely, roles) that speak only once. For this we introduce several tools and components for YOSO adaptation that may be useful for other protocols as well. A first such tool is *Future Broadcast (*FBcast*)* that allows a party P, that in the standard model would speak in several rounds, to send its future messages to future roles that will transmit the messages (either privately or through broadcast) when the time for those messages to be delivered comes. For example, consider a non-YOSO protocol where a party P transmits a message m at round i and a message m' at round $i+3$. In the YOSO adaptation, the role representing the actions of P in round i will transmit m at round i and also, in the same round, apply $\mathsf{FBcast}(m')$ to pass message m' to a role that will speak m' in round $i+3$. Note that this procedure is possible only in cases where the future message is known in advance. An interesting point to observe is that correctness of FBcast (in particular, in terms of correctness of messages sent "into the future"), needs only be guaranteed for original senders of m' that are honest as faulty ones can choose to speak any message of their choice whenever they speak. The sender P_i^j uses $\mathsf{FBcast}(m')$ to replace its own sending of m' in the future. In the emulated protocol a corrupt P_i^j could send $m'' \neq m'$ at this future point. So it is tolerable that $\mathsf{FBcast}(m')$ may open to $m'' \neq m'$ in the future when P_i^j is corrupt.

As a first application of FBcast, we use it to adapt the IT-SIGs of [22,21] to the YOSO model and then use this YOSOfied primitive to build a Distributed Commitment (DC) protocol in the YOSO model. In it, a party (honest or faulty) commits to a value that it can later choose to reveal or not, but it cannot change the committed value. Furthermore, it is guaranteed that values committed by honest parties are always revealed correctly. We then use DC as an essential ingredient in the design of a YOSO Verifiable Secret Sharing (VSS) scheme which in turn is a central component of our YOSO information-theoretic MPC solution.

In various steps in our protocol we need access to some form of randomness and for clarity of presentation we will assume the presence of a beacon functionality. However, in actuality we need something much weaker than a truly random source to deliver our results, it is enough that the challenge cannot be guessed. Thus, we can have a very simple implementation of the beacon (see full version [14]). We denote this functionality as $\mathcal{F}_{\mathsf{UPBeacon}}$ to reflect that it is an unpredictable beacon. During the analysis we at first assume it returns uniformly random elements. At the end we then return to why it is enough that it is unpredictable and how to implement it.

The solutions presented in this section make essential and repeated use of secret sharing techniques. In all cases, the underlying scheme is Shamir's scheme over a given field, and we assume all committees into which secrets are shared to have at least $t+1$ honest parties where $t+1 > n/2$. Thus, the polynomials defining shares are of degree t.

3.1 Information Theoretic and Homomorphic MAC

Message authentication codes (MAC) are used for verifying the authenticity of messages between a sender and receiver that share a secret key. Following the construction of [22] we have the following two protocols.

Three-party Setting. There exists (i) a sender S holding a message m, it chooses a key K and generates its corresponding MAC tag M computed under a key K; (ii) S sends the pair (m, M) to a receiver R; (iii) S sends the key K to a verifier V. The verification procedure combines the pair (m, M) held by R with the key K held by V.

For our purposes, we consider an information theoretic MAC function with the following properties: (i) producing a correct MAC without knowing the key succeeds with negligible probability even for an unbounded attacker; (ii) message hiding: nothing is learned about the message m from the key K; (iii) homomorphic: the MAC function is homomorphic with respect to appropriate group operations in the following sense. If $M_i = \mathsf{MAC}_{K_i}(m_i), i = 1, 2$, and the keys K_1, K_2 were computed by the same party (they might need to be correlated) then $M_1 + M_2 = \mathsf{MAC}_{K_1 +' K_2}(m_1 + m_2)$.

Such a MAC can be implemented as follows (all elements and operations are over a finite field, e.g., $\mathbb{Z}_p$): $K_i = (a, b_i)$, $M_i = am_i + b_i$ and $K_i +' K_j = (a, b_i + b_j)$. In the sequel, we will say that keys that share the same coefficient a but differ in b_i are *correlated*.

MAC with Distributed Public Verification. In the above setting, to verify a MAC one has to trust V to provide the correct key. In the scenarios in this paper, we often do not trust any single party individually, but rather can only count on committees with a majority of honest participants. Thus, we extend the basic 3-party scheme to one where the role of V is instantiated by an n-party committee $\mathsf{V} = \{V_1, \ldots, V_n\}$. Given a message m that S hands to R, S creates a MAC for m as follows. For $i = 1, \ldots, n$, S chooses keys K_i, computes $M_i = \mathsf{MAC}_{K_i}(m)$, and provides all M_i to R and K_i to V_i. When m needs to be verified, R first broadcasts m and the values M_i. Then, each V_i broadcasts K_i and the value m is accepted (i.e., the MAC validates) if and only if it holds that $M_i = \mathsf{MAC}_{K_i}(m)$ for at least $t + 1$ values of i.

The scheme guarantees that if S follows the protocol and $t + 1 > (n - 1)/2$ members of V are honest, then only a message m originating from S will be accepted. Note that the validation of m is public once R and members of V broadcast their values.

When the MAC in use is homomorphic, we have that if S MACs messages m_1, m_2 in the above way, with the same R and same committee V, then the message $m = m_1 + m_2$ can be validated as follows. R outputs m and $M_i = M_i^{(1)} + M_i^{(2)}$, $i = 1, \ldots, n$, and each V_i outputs $K_i^{(1)} +' K_i^{(2)}$. Here, $M_i^{(1)}, M_i^{(2)}$ are the MAC values received by R for m_1 and m_2, respectively, and $K_i^{(1)}, K_i^{(2)}$ are the keys received by V_i for m_1 and m_2, respectively. We therefore say that this MAC procedure is *homomorphic.*

This protocol is inherently YOSO as each party speaks only once and we refer to it in the following as IT-MAC.

3.2 Future Broadcast

We introduce *Future Broadcast* (FBcast), a fundamental primitive in the YOSO setting that allows an *honest* party P that speaks at time t to prepare a message m for broadcasting at a future time t'. This is accomplished by having P simply secret share m to a committee that will broadcast m at time t', hence bypassing the limitation of speaking only once. To guarantee that the message can be reconstructed (in the case that P is honest and the committee has an honest majority), FBcast implements a robust secret sharing scheme. Namely, a scheme where correct reconstruction is guaranteed as long as the sharing was done correctly and at least $t+1$ honest parties provide their shares (i.e., bad shares from corrupt parties can be identified and eliminated). In settings where digital signatures are available, robust secret sharing is implemented by having the dealer sign its shares. In our information-theoretic setting, we achieve a similar effect using the IT-MAC procedure from Section 3.1 for verifying share integrity.

FBcast.Share (Executed by S on input m)	FBcast.Reveal(with public verification)
Set two n-party committees, ShareHolder and ShareVerifier. 1. Compute a (t, n)-secret sharing $(m_1, \ldots, m_n)$ of m for $t = (n-1)/2$. 2. Generate keys $K_{i,j}$, $1 \leq i, j \leq n$ and compute $M_{i,j} = \mathsf{MAC}_{K_{i,j}}(m_i)$. 3. For $i = 1, \ldots, n$: Send $m_i, M_{i,1}, \ldots, M_{i,n}$ to $\mathsf{ShareHolder}_i$; Send $K_{1,i}, \ldots, K_{n,i}$ to $\mathsf{ShareVerifier}_i$.	1. $\mathsf{ShareHolder}_i$ bcasts $m_i, M_{i,1}, \ldots, M_{i,n}$. 2. $\mathsf{ShareVerifier}_i$ bcasts $K_{1,i}, \ldots, K_{n,i}$. 3. Accept m_i iff $M_{i,j} = \mathsf{MAC}_{K_{i,j}}(m_i)$ for at least $t+1$ of the keys. 4. If there are at least $t+1$ accepted shares and they all define a single polynomial of degree t then output the constant term. Otherwise, output "fail".

Fig. 2: Future Broadcast Protocol

The FBcast protocol is presented in Figure 2. Its first phase, FBcast.Share, is executed by a party S on input message m. It consists of S secret sharing m with a committee ShareHolder where in addition to its share, each $\mathsf{ShareHolder}_i$ receives an IT-MAC of the share computed by S using the above distributed MAC procedure. An additional committee, ShareVerifier, receives the MAC keys from S. When the value m needs to be broadcast in the future, FBcast.Reveal is performed following the distributed verification procedure: the ShareHolder members first broadcast their shares together with their MAC values, followed by a broadcast of keys held by ShareVerifier (note that ShareVerifier must speak after ShareHolder hence requiring two separate committees). Shares that do not pass verification are discarded and if those that remain interpolate to a single polynomial of degree t, the secret is reconstructed, otherwise reconstruction fails.

We denote by $\mathsf{FBcast.Share}_S(m)$ the sharing by S of a value m and $\mathsf{FBcast.Reveal}_S(m)$ the revealing of m (executed by two committees), and refer to the whole protocol execution as $\mathsf{FBcast}_S(m)$.

Analysis. We show that FBcast satisfies the requirement that if S is honest and used m as input to FBcast.Share then m will be reconstructed when FBcast.Reveal is executed. For this we need to show that only m_i's that originated from S are accepted and that there are sufficiently many accepted shares to interpolate the polynomial. If m_i is accepted then the MAC was verified by a key broadcast by at least one honest ShareVerifier. As S is honest, only m_i's created by S are accepted by an honest party. Furthermore, each share broadcasted by an honest ShareHolder is accepted as there will be at least $t+1$ honest ShareVerifiers whose broadcasted keys satisfy the MAC. By construction, no party speaks twice.

Homomorphism of FBcast. Note that when used with a homomorphic MAC, FBcast inherits the homomorphic property of the distributed MAC scheme from Section 3.1. We denote this fact as $\mathsf{FB}_P(m_1) + \mathsf{FB}_P(m_2) = \mathsf{FB}_P(m_1 + m_2)$ for any messages m_1 and m_2 shared by the same party P. Yet, as the keys need to be correlated the creator of the MAC needs to know in advance what two values will be added. This is easily achievable in our protocols.

3.3 Homomorphic IT-SIG

Our protocols would benefit from a signature functionality in order to construct a VSS protocol. Of course in the information theoretic setting we cannot achieve the full properties of a signature, but we can achieve enough of the functionality to deliver the result. The property which we need is the following. Assume again the setting from the IT-MAC (Section 3.1). We would want to assure R that the message that it holds will be accepted by the committee V. In essence, that it has a "signature" on the message that it holds.

Unlike the transformation of the basic IT-MAC from [22] that did not require modification to comply with the YOSO model, the IT-SIG construction from that paper does require changes as it has interaction. Our protocol IT-SIG is described in Figure 3. It consists of two phases, IT-SIG.Setup and IT-SIG.Reveal. In IT-SIG.Setup, a sender S provides a receiver R with a value m and also provides verification information to a committee V of n verifiers $V_1, \ldots, V_n$. The goal is for R to disclose m in the IT-SIG.Reveal phase in a way that allows to *publicly verify* the correctness of m with the help of committee V and with the following guarantees, assuming that V contains an honest majority:

– If S and R are honest then the correct value m is disclosed and verified during IT-SIG.Reveal and no information on m is revealed prior to that.
– If both S and R are corrupt we make no requirement at all.
– If only S is corrupt, at the end of IT-SIG.Setup, R holds a value m' that will pass verification in IT-SIG.Reveal.

IT-SIG.Setup	IT-SIG.Reveal
1. On input m, the sender S: (a) Generates keys $K_{i,j}$, $1 \leq i \leq n, 1 \leq j \leq \kappa$ (for security parameter κ), and computes $M_{i,j} = \mathsf{MAC}_{K_{i,j}}(m)$. (b) Transfers $(m, \{M_{i,j}\}_{1\leq i\leq n, 1\leq j\leq\kappa})$ to receiver R and $\{K_{i,j}\}_{1\leq j\leq\kappa}$ to V_i. (c) Executes $\mathsf{FBcast.Share}_S(m)$, and $\mathsf{FBcast.Share}_S(K_{i,j}), 1\leq i\leq n, 1 \leq j \leq \kappa$. 2. Party V_i: (a) Chooses half of the indices at random, denoted by INX_i. (b) Broadcasts $K_{i,j}$ for $j \in INX_i$. (c) Executes $\mathsf{FBcast.Share}_{V_i}(K_{i,j})$, $j \notin INX_i$. 3. Execute $\mathsf{FBcast.Reveal}_S(K_{i,j})$ $j \in INX_i$ for all i; denote by $\bar{K}_{i,j}$ the reconstructed values. 4. If there exist indexes i and j for which $\mathsf{MAC}_{\bar{K}_{i,j}}(m) \neq M_{i,j}$ then R asks that $\mathsf{FBcast.Reveal}_S(m)$ be executed to reveal $\bar{m}$. If $\bar{m} = \perp$ set $\bar{m}$ to a default value.	1. If $\bar{m}$ was revealed in IT-SIG.Setup output this as S's message. 2. R broadcasts $(m, \{M_{i,j}\}_{1\leq i\leq n, j\notin INX_i})$. 3. Set the number of votes for m to be the number of i's for which $\bar{K}_{i,j} \neq K_{i,j}$ for some $j \in INX_i$ from the setup. 4. For all i's not counted in the previous step, execute $\mathsf{FBcast.Reveal}_{V_i}(K_{i,j})$ for $j \notin INX_i$. If $\mathsf{MAC}_{K_{i,j}}(m) = M_{i,j}$ for any one of the recovered values then increment the vote by "1". 5. If vote is at least $t + 1$ then output m as S's message. Otherwise, output $\perp$.

Fig. 3: Information Theoretic SIG

– If only R is corrupt, no value other than the m that originated with S in IT-SIG.Setup can pass verification in IT-SIG.Reveal.

In addition, the protocol needs to satisfy the YOSO model where parties speak only once. We build it so that R speaks only once (either in IT-SIG.Setup or in IT-SIG.Reveal) while in the case of S and the parties in V, from which the logic of the protocol requires more than one message, we resort to FBcast for distributing their future messages so that a different committee broadcasts them when needed, and all parties speak only once.

Analysis. The following assumes an honest majority in committee V and that at most one of R and S is corrupted.

– Corrupt S: We need to show that at the end of IT-SIG.Setup, R holds a value m' that can pass verification in IT-SIG.Reveal. We split our analysis into two cases. First, if a value $\bar{m}$ is revealed during Step 4 of IT-SIG.Setup we set m' to $\bar{m}$ and the rest follows trivially as this value will be outputted in IT-SIG.Reveal. Otherwise, we set m' to the value m received from S and show that m' will have at least $t+1$ votes in IT-SIG.Reveal. Indeed, for each honest V_i, either $\bar{K}_{i,j} \neq K_{i,j}$ for some $j \in INX_i$ and thus their vote is counted; otherwise, it holds that $\mathsf{MAC}_{K_{i,j}}(m) = M_{i,j}$ for all $j \in INX_i$ as R did not complain against these values. Thus, with (overwhelming) probability $1/\binom{\kappa}{\kappa/2}$ due to the cut-and-choose technique, there exists a $j \notin INX_i$

such that $\mathsf{MAC}_{K_{i,j}}(m) = M_{i,j}$, and hence a vote for i will be counted. This guarantees at least $t+1$ votes for the value $m = m'$.

- Corrupt R: In this case we show that only the m that originated with S will pass verification in IT-SIG.Reveal. If the message associated with S is set to the value derived from $\mathsf{FBcast.Reveal}_S(m)$ in the setup, it is certainly a message that originated with S. If it is set to the message published by R, then that message must get $t+1$ "votes". Votes can be generated by corrupt V_i publishing incorrect keys in Step 2b of IT-SIG.Setup; however, there are at most t such corrupt V_i. The only other way to generate a vote for an incorrect m is to forge a MAC M, which happens with negligible probability.
- If S and R are honest, then due to the message hiding property of the MAC function, no information on m is revealed until IT-SIG.Reveal is executed. Indeed, the only case where R requests to broadcast m prior to IT-SIG.Reveal is when the keys broadcasted by S do not verify the MACs; this cannot be the case when S and R are both honest.

Homomorphism of IT-SIGs. The homomorphic properties of the MAC construction from Section 3.1, imply similar properties for IT-SIG in Figure 3 when the underlying MAC function is homomorphic. Namely, if m, m' are messages on which the (same) sender S runs IT-SIG.Setup with the same set V of verifiers and with correlated keys (i.e., corresponding keys use the same coefficient a in the scheme from Section 3.1), then an IT-SIG on $m + m'$ can be verified with committee V using the MAC keys held by V for m and for m'. This homomorphic property is used in an essential way when performing additions/multiplications in an arithmetic circuit as described in Section 3.11. A consequence of the need for correlated keys is that if two messages may need to be added in the future, this fact needs to be known at the time of generating the IT-SIG for both m_1 and m_2. In our application this is always the case as the need for additions is determined by the specific circuit being computed.

3.4 Distributed Commitment (DC)

The FB protocol does not offer any guarantees in the case when the dealer is faulty. Here, we introduce the *distributed commitment protocol* DC that strengthens FB by providing better guarantees when the dealer is corrupt. DC consists of two phases, DC.Commit and DC.Reveal. In DC.Commit, a committer C commits to a value m that may later be revealed in DC.Reveal. More precisely, if C is honest, then as in the case of FB, the revealed value is m, and m is hidden until it is revealed. However, if C is corrupt, the execution of DC.Commit determines a single value m such that the output of DC.Reveal is guaranteed to be either $\perp$ or m (where m itself can be $\perp$). In other words, C can choose to prevent reconstruction, but if it allows for it to happen then it can only be to a value it committed to at the end of DC.Commit. Reconstruction is public, namely, there will be public agreement on the output of DC.Reveal. In essence, this is analogous to a regular commitment in the computational setting where the committer is bound to the value but has the option not to reveal it.

DC.Commit (executed by C on input m)	DC.Reveal
Let ShareHolder and ShareVerifier be two n-party committees. 1. Committer C computes a t-secret sharing of m, $(m_1, \dots, m_n)$ for $t \geq (n-1)/2$. 2. For $i = 1, \dots, n$: C executes IT-SIG.Setup on input m_i with ShareHolder$_i$ as receiver R and the set ShareVerifier acting as the set of verifiers V (same ShareVerifier committee is used in all the invocations).	1. For $i = 1, \dots, n$, run IT-SIG.Reveal with ShareHolder$_i$ as receiver R; let $\bar{m}_i$ to the output of this execution. 2. Take all $\bar{m}_i$ that are not $\perp$ and interpolate a polynomial through these points. If the polynomial is of degree t or less output its constant term, otherwise output $\perp$.

Fig. 4: Distributed Commitment

Protocol DC uses the IT-SIGs (Figure 3) in an essential way. In particular, in Step 3 of DC.Commit, for each m_i, C executes IT-SIG.Setup(m_i) with ShareHolder$_i$ acting as the receiver and with ShareVerifier as the set V of verifiers. The n executions (one for each m_i) are performed in parallel using the same set ShareVerifier in all these executions.

Analysis. We show that at the end of DC.Commit a value m (or $\perp$) is determined, and during DC.Reveal, if C is honest m will be revealed, and if C is corrupt, either m or $\perp$ will be revealed.

In DC.Commit, C executes IT-SIG.Setup with at least $t+1$ honest parties acting as receivers R. For these honest parties, due to the properties of IT-SIG.Setup, it is guaranteed that the value they hold will be accepted in IT-SIG.Reveal. We claim that at the end of DC.Commit, a single value m is committed to, such that the output in DC.Reveal is either m or $\perp$ (where m itself can be $\perp$). To show this, we define m as the constant term of a polynomial of degree at most t interpolated through the set of shares held by the honest parties (this value might be $\perp$ if the points interpolate to a polynomial of a higher degree than t). We now show that if a value is outputted in DC.Reveal it can only be m. When C is honest then only shares that were created by C are accepted and thus the polynomial will interpolate properly during DC.Reveal. If C is faulty we know that at least the shares of the honest parties will be included in the set of shares being interpolated and this is a set of at least $t+1$ shares. Thus, the message which is opened can only be m or $\perp$, with the latter happening only if the shares m_i did not corresponf to points on a polynomial of degree at most t.
We denote by $\mathsf{DC}_P(m)$ the output of the execution of DC.Commit by party P on message m.

Homomorphism of DC. Due to the homomorphic properties of the IT-SIG and FBcast, we have that for any two values m and m' committed by the same honest party P, it holds that $\mathsf{DC}_P(m)+\mathsf{DC}_P(m') = \mathsf{DC}_P(m+m')$. The same considerations for ensuring the homomorphism of IT-SIG described in Section 3.3 hold here too (i.e., the DC operations need to be performed by the same committer using correlated keys). In particular, if this property may be required in the future for two messages m, m', then this fact needs to be known at the time of running DC.Commit on these values (fortunately, for our application this requirement does hold). The question might be raised if we know that m and m' will be added why compute individual DC.Commit for both rather than the sum. In many instances we will need to utilize all three values in different computations.

3.5 Duplicate DC

In our protocols, we often need to use a committed value multiple times, thus requiring the decommitting parties in the DC protocol to act in more than one round, a violation of the YOSO model. One possible solution is for the committer C to commit twice (or more) onto different committees to the same value and provide a proof of equality for the committed values; yet this proof of equality will "waste" the sharing, which is what we need to prevent. Thus, we avoid proofs of equality by having the parties in ShareHolder and ShareVerifier reshare the values that they receive in IT-SIG.Setup. It suffices that honest parties share their shares correctly to guarantee that all duplicates commit to the same value. We are using in an essential way the fact that it is the shareholders and verifiers that reshare their values rather than C, and that we can rely on a majority of honest shareholders.

We define protocol DupDC that allows for the duplication of a DC-committed value m. Let d be the number of duplicates needed. In a first committing phase, DupDC.Commit, committer C runs DC.Commit with a committee ShareHolder, sharing its input m so that $\mathsf{ShareHolder}_i$ receives a share m_i. To generate d duplicates, for each i, $1 \leq i \leq n$, C runs d copies of IT-SIG.Setup on m_i, each copy with an independent set of MAC keys. The same ShareVerifier committee is used for all invocations. The d copies are verified by $\mathsf{ShareHolder}_i$, acting as receiver R, as specified by IT-SIG.Setup. Finally, in the last step of DupDC.Commit, the $\mathsf{ShareHolder}_i$'s and ShareVerifiers execute d independent FBcast.Share for all the values that they holds, onto $2d$ separate committees.

The DupDC.Reveal phase follows DC.Reveal where the opening of m_i is implemented via share reconstruction by one of the d ShareHolder committees to which m_i was shared. Additional information that needs to be broadcast and verified as specified by IT-SIG.Reveal is performed via FBcast.Reveal by the FBcast committees created by $\mathsf{ShareHolder}_i$ during DupDC.Commit.

Analysis. It is straightforward to check that if the original committer C was honest, all duplicated values are correct DC commitments and they will open to the same committed value during DupDC.Reveal. If C is dishonest, but $\mathsf{ShareHolder}_i$ is honest, and verification against a ShareVerifier committee fails during the IT-SIG.Setup actions, then the committed value is set to the one that is FBcast.Reveal as part of Step 4 in IT-SIG.Setup. Otherwise, the value m_i can be reconstructed correctly by any of the d sharings of m_i shared by $\mathsf{ShareHolder}_i$. Since there is a majority ($t + 1$ or more) of honest shareholders in each of the d ShareHolder committees, it is guaranteed that only the committed value or $\perp$ will be reconstructed in each of the d copies.

It follows from the properties of the DC and FBcast protocols. We note that C still has the option of not opening any subset of these duplicate commitments, but all those that will be open will be open to the same value. Note that if the verification fails for one of the duplicates and a value $\bar{m}$ is revealed then it is used for all duplicates.

3.6 Verifiable Secret Sharing Scheme

The distributed commitment DC functionality ensures that the committer, even a corrupt one, is committed to a single value at the end of $\mathsf{DC.Commit}$. However, a corrupt committer can prevent reconstruction of the committed value during $\mathsf{DC.Reveal}$. In our applications, we need a commitment scheme with the property that if the commitment phase is successful then reconstruction of the committed value is guaranteed. We achieve this via *Verifiable Secret Sharing* (VSS), a protocol where a dealer secret shares a value s during a $\mathsf{VSS.Share}$ phase so that s is guaranteed to be reconstructed during $\mathsf{VSS.Reveal}$ from any subset of shareholders that includes $t+1$ honest ones. This is the case even for corrupt dealers that were not disqualified during $\mathsf{VSS.Share}$.

First, we introduce a procedure used in our VSS design as well as part of the MPC protocol. The goal is to guarantee that two parties that are supposed to share the same value s, had in fact done so. We describe the protocol using generic sharing that can be instantiated with any of the sharing protocols discussed in this paper, including DC, VSS, and its variants.

Protocol Share Equality Test.
1. Party P_1 shares two values a_1, ρ_1 and P_2 shares values a_2, ρ_2.
2. Value r is obtained from an unpredictable beacon $\mathcal{F}_{\mathsf{UPBeacon}}$
3. The values $a_1 + r \cdot \rho_1$ and $a_2 + r \cdot \rho_2$ are reconstructed from their sharings.
4. If the reconstruction succeeds and the reconstructed values are equal, conclude the test was successful and $a_1 = a_2$. In any other case reject the test.

It follows using a standard argument that if $a_1 \neq a_2$ then there is at most a single challenge r that will make the proof pass, implying a probability error of $|\mathbb{F}|^{-1}$ for unpredictable r. Therefore, an unpredictability beacon $\mathcal{F}_{\mathsf{UPBeacon}}$ suffices (see the full version [14] for details).

Protocol $\mathsf{VSS.Share}$ proceeds as follows.

1. The dealer D chooses a random polynomial $f(x)$, s.t. $f(0) = s$ and an additional random polynomial $r(x)$, both of degree t. Let the coefficients of $f(x)$ and $r(x)$ be, respectively, f_j, r_j for $0 \leq j \leq t$.
2. Given a set $\mathsf{ShareHolder} = \{P_1, \ldots, P_n\}$, D computes $s_i = f(i), \rho_i = r(i)$ for $1 \leq i \leq n$ and transfers these values privately to P_i.
3. In the same step as above, D performs $\mathsf{DupDC.Commit}$to all the values f_j, r_j. Due to the homomorphic properties of DC, this results in implicit $\mathsf{DC}_D(s_i)$ and $\mathsf{DC}_D(\rho_i)$ sharings (shares of f_j, r_j allow the $\mathsf{ShareHolder}$ committee to compute values s_i, ρ_i for all i).

4. P_i performs $\mathsf{DupDC.Commit}(s_i)$ to obtain two copies of $\mathsf{DC}_{P_i}(s_i)$ (particular applications, such as MPC, may require more copies) and performs $\mathsf{DC}_{P_i}(\rho_i)$, all with homomorphically correlated keys. Additionally, P_i shares the ρ_i to one of the committees to which it duplicates the s_i.
5. Run the above *Equality Test* on the sharings of D and P_i of value s_i and auxiliary ρ_i (in the case of D, the committee uses the implicit DC commitment of s_i, ρ_i).
6. If the values are not equal execute $\mathsf{DC.Reveal}$ of D's sharing of s_i. If it returns $\perp$ disqualify the dealer.

Protocol $\mathsf{VSS.Reveal}$ proceeds as follows.

1. Execute $\mathsf{DC.Reveal}$ for all s_i shared by P_i
2. Interpolate a polynomial using all these share and output the constant term.

Analysis. The VSS protocol needs to ensure that all of the dealer's shares s_i are points on a polynomial of degree at most t and that the value s_i shared by P_i is the same as the one received from D. The former property is enforced via the DC-sharing of polynomial coefficients by D (it ensures the degree of the polynomial and the implicit DC sharing of shares s_i and ρ_i) while the latter uses the equality test to compare the sharings of D and P_i.

Homomorphism of VSS. VSS inherits the homomorphic properties of DC, importantly, in the case of VSS, these properties hold even if the VSS was performed by two different dealers as long as it was done into *the same set of shareholders.* Namely, for two secrets m_1 and m_2, and two dealers D_1 and D_2, we have $\mathsf{VSS}_{D_1}(m_1) + \mathsf{VSS}_{D_2}(m_2) = \mathsf{VSS}(m_1 + m_2)$. Note that the right-hand side VSS is not associated to a specific dealer as it combines sharings of D_1 and D_2. The reason the homomorphism holds across dealers is due to the homomorphic properties of $\mathsf{DC}_{P_i}(\cdot)$ (that only hold for same committer) and the fact that the same P_i's act in both VSS dealings as shareholders.

3.7 Duplicate VSS

As in the case of DC, we also need duplicates of VSS values as a value will need to be part of various computations. Recall that a VSS is a sharing of a value s where each share s_i of the sharing is shared as $\mathsf{DC}_{P_i}(s_i)$. It is easy to see that duplicating the $\mathsf{DC}_{P_i}(s_i)$ commitments results in duplicate VSSs.

3.8 Augmented VSS

In our application, particularly for the multiplication protocol, we need an *Augmented VSS (*AugVSS*)*, where not only the secret given as input is shared with VSS but also the shares resulting from $\mathsf{VSS}(s)$ are shared with VSS.

AugVSS is achieved via the following computation. The dealer D holding a value s defines a polynomial $f(x) = f_t x^t + ... + f_1 x + f_0$ where $f_0 = s$. It carries out $\mathsf{VSS}(f_\ell)$ for $0 \leq \ell \leq t$. Through the homomoprhic properties of the VSS, this implicitly creates a $\mathsf{VSS}(s_i)$ where $s_i = f(i)$.

It can easily be verified that AugVSS is also additively homomorphic, inheriting this property from the homomorphic properties of the VSS. Furthermore, an AugVSS of a value m can be added to a VSS of a value m' creating a VSS sharing of $m + m'$.

3.9 Duplicate AugVSS

Unlike the previous duplications, e.g. duplicate VSS, where we need to simply have another copy of the value, the duplicate AugVSS needs to provide a stronger guarantee. It needs to have a sharing of the same value but with a differnet polynomial. The need for this will become evident when we describe the MPC protocol. AugVSS is modified as follows.

A single duplicate VSS is carried out for the constant term, $\mathsf{DupVSS}(f_0)$. In addition, two sets of values $f_t, ..., f_1$ and $f'_t, ...f'_1$ are chosen. Each set in combination with f_0 defines a different polynomial with the same constant term. The protocol from above is executed on both these sets to create two duplicates. If more copies are needed additional coefficients need to be chosen.

3.10 Proof of Local Multiplication (PLM)

In the following protocol, a prover P shares values a, b and c using VSS and proves that $a \cdot b = c$. The proof uses two committees, C and C'.

1. P performs $\mathsf{VSS}_P(a)$ and $\mathsf{VSS}_P(c)$ onto committee C, and $\mathsf{VSS}_P(b)$ onto committee C'. In addition, P chooses a random value b' and executes $\mathsf{VSS}_P(b')$ onto committee C' and $\mathsf{VSS}_P(a \cdot b')$ onto committee C.
2. Receive random e from $\mathcal{F}_{\mathsf{UPBeacon}}$;
3. Committee C' reconstructs using $\mathsf{VSS.Reveal}$ the value $r = e \cdot b + b'$;
4. Committee C reconstructs using $\mathsf{VSS.Reveal}$ the value $d = r \cdot a - e \cdot c - a \cdot b'$
5. Accept the proof if $d = 0$ and reject otherwise.

It follows using a standard argument that if $c \neq ab$ then $d \neq 0$ except with probability $|\mathbb{F}|^{-1}$. In particular, there is a single e which will let the proof pass. Hence it is enough that e cannot be guessed with non-negligible probability. The rest of the argument for the correctness of the proof follows from the properties of the VSS.

3.11 YOSO MPC

Using the tools developed up to now we can show how to do secure function evaluation (or MPC) in the YOSO model. That is, we are given an arithmetic circuit $\mathcal{C}$, with m secret inputs provided by m parties (roles), and we show how to privately compute the circuit on the inputs, in the YOSO model.

Let $\mathcal{C}$ be a given arithmetic circuit with m inputs $x_1, \ldots, x_m$ and gates $g_1, \ldots, g_\ell$. For the YOSO computation of $\mathcal{C}$, we show how to create, given a gate g_i with input values v_{i1}, v_{i2}, both shared with $\mathsf{DupAugVSS}$, a committee C_i that will hold a $\mathsf{DupAugVSS}$ sharing of the output of the gate. In addition, there will be a collection of d duplicates of the AugVSS of the gate's output, where d is the number of gates to which this output enters as an input.

With a lot of attention to details and committee selection we could do the addition of the MPC without interaction. However, to simplify the description of the protocol and to make the addition and multiplication more uniform we will describe things in the same manner.

Gate input setup: As we are looking at a single gate we refer to the committee computing the gate as C. The parties in this committee are $P_1, ..., P_n$. Assume that the value on one input wire is a and the second is b.
The parties in the committee C needs to receive its shares of the values on the input wires. As we assume that the values a and b of input wires are shared using AugVSS this means that the share a_i and b_i of party P_i are shared using a VSS. These values are reconstructed towards P_i. Once P_i receives these two shares it shares them using DupVSS. In addition, P_i proves that it shared the values which it received, and this is done using the proof of equality of sharing from Section 3.6.

Addition: An addition gate can be implemented without interaction. However, for simplicity, we take advantage of the fact that (as needed for multiplication gates) input wires are shared using $\mathsf{DupAugVSS}$, hence we can use the homomophic properties of AugVSS to implement addition.

Multiplication:
1. Party P_i holding shares a_i and b_i of the input wires, shares the value $\gamma_i = a_i \cdot b_i$ using $\mathsf{DupAugVSS}$. The sharing of these values needs to be done onto different committees as specified by the PLM protocol.
2. It executes the PLM protocol to prove that γ_i is the product of its two input shares (Section 3.10).
3. For any i for which the $\mathsf{DupAugVSS}$ or the PLM procedures fail, the committee that holds a_i and b_i uses $\mathsf{VSS.Reveal}$ to publicly reconstruct these values. Later, when the protocol uses the value γ_i, its value is set to the product $a_i \cdot b_i$ of the reconstructed values.
4. The linear combination of the AugVSS of the γ_i's define the AugVSS of $c = a \cdot b = \Sigma_{i=1}^{2t+1} \lambda_i(\gamma_i = a_i \cdot b_i)$. This also creates the $\mathsf{VSS}(c_i) = \Sigma_{j=1}^{2t+1} \lambda_j \mathsf{VSS}(\gamma_{j,i})$ for the appropriate Lagrange coefficients.

Security argument. The multiplication protocol follows the design of [13]. The correctness of the AugVSS sharing of the multiplication $c = a \cdot b$ follows from: (i) the fact that $\mathsf{AugVSS}(\gamma_i)$ completed in a proper manner and its homomorphic properties (ii) the correctness of the PLM; (iii) the public availability of γ_i values for those i where verification failed (these values are available because in AugVSS of the input values of the wires, not only the secret is shared but also its shares). (iv) the existence of Lagrange coefficients λ_i for which $c = a \cdot b = \Sigma_{i=1}^{2t+1} \lambda_i (a_i \cdot b_i)$.

Formalizing security follows standard arguments. In particular, the simulator proceeds as follows. Use the AugVSS's to reconstruct the inputs of the corrupted parties. Input these to $\mathcal{F}_{\text{MPC}}^{F}$ where F denotes the function computed by $\mathcal{C}$. Use dummy inputs of the honest parties in the simulation. Run the simulated protocol honestly with these dummy inputs. When processing an output gate, learn the correct output from $\mathcal{F}_{\text{MPC}}^{F}$. Then from the t simulated shares of the corrupted parties and the output acting as share $t+1$ compute the matching shares of the honest parties. Then send these in the simulation. Furthermore, the simulation of the IT-MAC and IT-SIG are straightforward.

To prove adaptive security the simulator will for each committee C_j start out with a set C_j of size t playing the role of the corrupted parties and will simulate as in the static case with C_j being corrupted. If party P_i^j in C_j becomes corrupted and $\mathsf{P}_i^j \notin C_j$ then the simulator will swap P_i^j with an honest party in C_j and then patch the view of the party to get a simulated state of P_i^j. If P_i^j holds a share on a random, unknown polynomial of degree at most t, the share will just be simulated by a random field element. If P_i^j holds a share on a random, known polynomial of degree at most t, as is the case for a reconstructed output of the computation, then the simulator will know the output and will, with the additional t simulated shares of C_j, have $t+1$ simulated shares. From these it can compute the corresponding simulated share of P_i^j and claim this as the state of P_i^j. In general the adaptive patching follows using standard techniques from MPC and can be done along the lines of [11] where the patching technique is used to prove [2] adaptive secure in the UC model.

References

1. Gilad Asharov, Abhishek Jain, Adriana López-Alt, Eran Tromer, Vinod Vaikuntanathan, and Daniel Wichs. Multiparty computation with low communication, computation and interaction via threshold FHE. In *Advances in Cryptology - EUROCRYPT 2012*, volume 7237 of *Lecture Notes in Computer Science*, pages 483–501. Springer, 2012.
2. Michael Ben-Or, Shafi Goldwasser, and Avi Wigderson. Completeness theorems for non-cryptographic fault-tolerant distributed computation (extended abstract). In Janos Simon, editor, *Proceedings of the 20th Annual ACM Symposium on Theory of Computing, May 2-4, 1988, Chicago, Illinois, USA*, pages 1–10. ACM, 1988.
3. Fabrice Benhamouda, Craig Gentry, Sergey Gorbunov, Shai Halevi, Hugo Krawczyk, Chengyu Lin, Tal Rabin, and Leonid Reyzin. Can a public blockchain keep a secret? In *Theory of Cryptography - 18th International Conference, TCC 2020*, volume 12550 of *Lecture Notes in Computer Science*, pages 260–290. Springer, 2020.
4. Erica Blum, Jonathan Katz, Chen-Da Liu Zhang, and Julian Loss. Asynchronous byzantine agreement with subquadratic communication. *IACR Cryptol. ePrint Arch.*, 2020:851, 2020.
5. Ran Canetti. Universally composable security: A new paradigm for cryptographic protocols. In *42nd FOCS*, pages 136–145. IEEE Computer Society Press, October 2001.

6. Ignacio Cascudo and Bernardo David. SCRAPE: Scalable randomness attested by public entities. In Dieter Gollmann, Atsuko Miyaji, and Hiroaki Kikuchi, editors, *ACNS 17*, volume 10355 of *LNCS*, pages 537–556. Springer, Heidelberg, July 2017.
7. Ignacio Cascudo and Bernardo David. ALBATROSS: publicly attestable batched randomness based on secret sharing. *IACR Cryptol. ePrint Arch.*, 2020:644, 2020.
8. Jing Chen and Silvio Micali. Algorand: A secure and efficient distributed ledger. *Theor. Comput. Sci.*, 777:155–183, 2019.
9. Arka Rai Choudhuri, Aarushi Goel, Matthew Green, Abhishek Jain, and Gabriel Kaptchuk. Fluid MPC: secure multiparty computation with dynamic participants. *IACR Cryptol. ePrint Arch.*, 2020:754, 2020.
10. Ronald Cramer, Ivan Damgård, and Jesper Buus Nielsen. Multiparty computation from threshold homomorphic encryption. In Birgit Pfitzmann, editor, *EUROCRYPT 2001*, volume 2045 of *LNCS*, pages 280–299. Springer, Heidelberg, May 2001.
11. Ivan Damgård and Jesper Buus Nielsen. Adaptive versus static security in the UC model. In *Provable Security - 8th International Conference, ProvSec 2014*, volume 8782 of *Lecture Notes in Computer Science*, pages 10–28. Springer, 2014.
12. Juan A. Garay, Yuval Ishai, Rafail Ostrovsky, and Vassilis Zikas. The price of low communication in secure multi-party computation. In Jonathan Katz and Hovav Shacham, editors, *CRYPTO 2017, Part I*, volume 10401 of *LNCS*, pages 420–446. Springer, Heidelberg, August 2017.
13. Rosario Gennaro, Michael O. Rabin, and Tal Rabin. Simplified VSS and fast-track multiparty computations with applications to threshold cryptography. In Brian A. Coan and Yehuda Afek, editors, *17th ACM PODC*, pages 101–111. ACM, June / July 1998.
14. Craig Gentry, Shai Halevi, Hugo Krawczyk, Bernardo Magri, Jesper Buus Nielsen, Tal Rabin, and Sophia Yakoubov. YOSO: you only speak once / secure MPC with stateless ephemeral roles. *IACR Cryptol. ePrint Arch.*, 2021:210, 2021.
15. Martin Hirt and Ueli M. Maurer. Player simulation and general adversary structures in perfect multiparty computation. *Journal of Cryptology*, 13(1):31–60, January 2000.
16. Jonathan Katz, Ueli Maurer, Björn Tackmann, and Vassilis Zikas. Universally composable synchronous computation. In Amit Sahai, editor, *TCC 2013*, volume 7785 of *LNCS*, pages 477–498. Springer, Heidelberg, March 2013.
17. Silvio Micali. Very simple and efficient byzantine agreement. In Christos H. Papadimitriou, editor, *ITCS 2017*, volume 4266, pages 6:1–6:1, 67, January 2017. LIPIcs.
18. Pratyay Mukherjee and Daniel Wichs. Two round multiparty computation via multi-key FHE. In Marc Fischlin and Jean-Sébastien Coron, editors, *EUROCRYPT 2016, Part II*, volume 9666 of *LNCS*, pages 735–763. Springer, Heidelberg, May 2016.
19. Satoshi Nakamoto. Bitcoin: A peer-to-peer electronic cash system, 2009.
20. Rafael Pass and Elaine Shi. The sleepy model of consensus. In Tsuyoshi Takagi and Thomas Peyrin, editors, *ASIACRYPT 2017, Part II*, volume 10625 of *LNCS*, pages 380–409. Springer, Heidelberg, December 2017.
21. Tal Rabin. Robust sharing of secrets when the dealer is honest or cheating. *J. ACM*, 41(6):1089–1109, 1994.
22. Tal Rabin and Michael Ben-Or. Verifiable secret sharing and multiparty protocols with honest majority (extended abstract). In *Proceedings of the 21st Annual ACM Symposium on Theory of Computing*, pages 73–85. ACM, 1989.