# Distributed Cryptography as a Service

**Elisaweta Masserova**

CMU-CS-24-151

September 2024

Computer Science Department
School of Computer Science
Carnegie Mellon University
Pittsburgh, PA 15213

**Thesis Committee:**
Bryan Parno, Co-Chair
Vipul Goyal, Co-Chair (NTT Research and Carnegie Mellon University)
Elaine Shi
Antigoni Polychroniadou (J.P. Morgan AI Research)
Tal Rabin (Amazon Web Services and University of Pennsylvania)

*Submitted in partial fulfillment of the requirements*
*for the degree of Doctor of Philosophy.*



The work presented in this thesis was supported in part by a gift from Bosch, Protocol Labs Cryptonet Network Grant RFP-013 "Stateless Distributed Randomness Generation", NSF Grants No. 1801369 and 2224279, and by the CONIX Research Center, one of six centers in JUMP, a Semiconductor Research Corporation (SRC) program sponsored by DARPA.
Any opinions, findings, and conclusions or recommendations expressed in this thesis are those of the author and do not necessarily reflect the views of these sponsors.

**Keywords:** Secure Multi-Party Computation, Distributed Systems, Blockchains

*Für Gabi*

## Abstract

Today's world is undeniably data-driven. The explosion of the Internet has generated vast volumes of data, and the advent of machine learning has unlocked captivating applications that thrive on this data. In such a world, it is evident that the ability to store, transmit, and process data securely is paramount. Distributing trust is one of the fundamental cryptographic principles that enable such security, and it is at the core of key cryptographic tools such as multi-party computation (MPC) and randomness generation. As the demand for secure and reliable cryptographic solutions grows, there is increasing interest in offering distributed protocols as a service.

Such services are typically expected to run continuously for long periods of time, requiring significant resource commitments from all participating parties. One approach to mitigate this issue is to design distributed cryptographic protocols that are *stateless*. In stateless protocols, parties are easily replaceable, and can contribute to the execution by participating only for a short time, without committing to a long-term computation.

In this work, we study stateless multi-party computation. We start by introducing a stateless MPC protocol which does not require parties to be online at the same time and requires *no interaction* between the participants. We construct this protocol in the blockchain model and under the assumption of what we call *Conditional Storage and Retrieval* (CSaR) systems. In our next step, we eliminate the CSaR requirement and design stateless MPC without relying on this assumption. More concretely, we focus on the recently introduced *You Only Speak Once* (YOSO) paradigm. In this model participating parties are allowed to send only a single message; i.e., they speak only once. We improve the state of the art in YOSO MPC by designing a protocol with better communication complexity than the currently known solutions. Then, we focus on improving the efficiency of special-purpose YOSO MPC. Specifically, we consider the task of distributed randomness generation, and design a suite of protocols, each balancing different trade-offs in terms of underlying assumptions, efficiency, and corruption threshold. We develop these protocols in the model of *YOSO with worst-case corruptions* ($\text{YOSO}^{\mathsf{WCC}}$), which is even stronger than the original YOSO MPC.

## Acknowledgments

I am forever grateful for the help of many wonderful people who accompanied me on the journey that led to this thesis.

First and foremost, this work would not be possible without my advisors, Bryan Parno and Vipul Goyal.

Bryan's approach to research, advising, and life never ceases to amaze me. He is incredibly dedicated, systematic, always present and empathetic. I learned so many things from him – how to write, which questions to ask, and how to find meaning in research, even when everything seems pointless. Bryan, it has been an absolute pleasure working with you over the past six years.

Vipul pushed me to become a better researcher, offering support whenever needed and bringing boundless enthusiasm – whether when I showed a (very simple!) tool I programmed, or embarked on a new project. Without his technical knowledge and his guidance, I wouldn't be the cryptographer I am today.

Both Bryan and Vipul always supportive of my varied research interests, and I am very grateful for the freedom to simply dive into any new project that captured my curiosity, knowing my advisors would have my back.

Sincere thanks go to Elaine Shi, who, since joining CMU, has become my inofficial third advisor. I am fascinated by Elaine's dedication to her work and, above all, to her students. I am fairly confident that I spent more hours working in Elaine's office than I did in my own. Elaine's presence at CMU enriched not only my life, but many others. The CMU crypto seminar, which brought so many minds together, would likely not have happened without her.

I would like to thank my thesis committee members, Antigoni Polychroniadou and Tal Rabin. Your insightful feedback and comments have truly shaped this thesis, and I am immensely grateful for your advice. It would not have been the same without you.

Special thanks to the members of the CMU cryptography and security community – Wenting, Aayush, Steve, Aymeric, Jay, Travis, Abhiram, Yi, Sydney, Josh, Chanhee, Mike, Pratap, Yifan, Justin, Alper, Orestis, Ke, Hao, Nikhil, Mingxun, Andrew, Afonso, and many others. I learned so much from you and you made my time in Pittsburgh truly memorable. Special shout-out to Ke – my partner in crime in exploring Pittsburgh. I already miss our coffee dates! Hao and Andrew – how will I ever play Spirit Island without you guys? Yifan – you taught me so much about crypto, thanks for being so patient with

all the questions I had. Aymeric and Abhiram – I will always remember our peaceful afternoons chatting over tea. Jay – thank you for being the person I knew I could always talk to about virtually anything.

Thank you to the people I met during my two summers at J.P.Morgan – Antigoni and Daniel, working with you was truly a crucial point in my PhD. I was surprised by how much I enjoyed diving into MPC, and I owe much of that to you two. Akira, thank you for teaching me so much about zero-knowledge, and (especially!) thank you for agreeing to learn machine learning together. Alex, Harish – thanks for sharing your ideas and your knowledge with me, I can't wait to finally start working with you! Yiping – I couldn't have wished for a better partner to explore the incredible NYC cuisine with. Chenkai, Nikolas, Yue, Sahar – thank you for all our discussions, formal and less formal ones. You made my summers in NY absolutely unforgettable.

To the incredible team behind the "YOSO Randomness Generation" project – João, Chen-Da, Pratik, and Aravind – thank you! Working with you has never felt like work. I learned a lot from each of you. To many more projects together! Special thanks to Chen-Da for introducing me to problems I really enjoy thinking about and reminding me of what I'm good at ("Lisa, attack!"). Ours was one of the most fun projects I worked on in the past six years.

Thank you to each of my amazing co-authors from other projects. Special shout-out to Akshay – your vast knowledge of crypto fascinates me time and time again. Deepali – you taught me how to communicate ideas to those outside of my area. Hao – I don't know how often we had to write a Rapidash rebuttal, but with you, it never felt like a chore (though I do hope we finally make it soon).

Thank you to my Pittsburgh friends, who made my time here so much brighter – Yang, Max, Philipp, Dominik, and Conni. Thank you to my friends back in Germany and Ukraine – Ainara, Jenny, Geraldine, Jérôme, Isi, Emi, Dennis, Natalia, Kamilla, Nastya, and many more – I always knew I could count on each of you. Noemi – shout-out for being an amazing (conference) travel buddy!

A heartfelt thank you to my amazing mother for her unwavering love and support throughout my life. And finally, to a group of people I've always considered family, even though we are not related – Naikis, Gilbert, Heinrich, and meine liebe Gabi, who this thesis is dedicated to. None of this would have been possible without you. Thank you.

# Contents

# List of Figures

# List of Tables

# Chapter 1

# Introduction

*It's a dangerous business, Frodo, going out your door. You step onto the road, and if you don't keep your feet, there's no knowing where you might be swept off to.*

J.R.R. Tolkien, The Lord of the Rings

Each day, we rely on digital services to handle our communication, access essential information, and manage our money and health records. In this increasingly data-centric world, access to secure and reliable digital services is crucial. At the same time, as our dependence on digital communication and computation increases, traditional centralized approaches to manage data start to show cracks. While centralization allows for highly efficient systems, centralized systems are vulnerable to single points of failure. This can be an issue even in non-adversarial settings – just think how often you encounter bugs, network outages, and software glitches!

The problems become even more critical when there is malicious intent, especially when the stakes are high. Denial of service (DoS), zero-day exploits, and social engineering attacks are being used by powerful malicious actors, sometimes even governments, to steal private data or disrupt processes and bring organizations to their knees.

Decentralization, one of the key principles of modern cryptography, helps. By distributing trust among multiple parties in a clever way, distributed systems are far more resilient against compromise than their centralized counterparts. Indeed, there is growing demand for distributed cryptographic protocols such as secure multi-party computation. However, if these protocols were offered *as a service*, they would be expected to run continuously for extended periods of time, which would, in turn, demand non-trivial resource commit-

ments from participating parties. One way to alleviate this issue is to design distributed cryptographic protocols in which parties are easily replaceable, so a party can leave the computation at any given time, without committing their resources long-term. An easy way to obtain this property is to design protocols which are *stateless*, in the sense that parties do not maintain local private state. In such protocols, parties are easily replaceable as there is no need to transfer sensitive information to the successor when a party leaves and another takes over the execution.

In this thesis, we advance the state of the art in (mostly) stateless distributed cryptographic protocols. As a first step, we focus on the problem of secure multi-party computation (MPC). MPC protocols allow mutually distrusting parties to compute an output of a function on their private inputs. Remarkably, MPC protocols are designed in a way that, aside from the intended output, no information is leaked about the inputs. This opens up an entire realm of new applications. For example, two banks could employ an MPC protocol to collaboratively run fraud detection mechanisms on *joint* data, without disclosing the respective confidential data to each other. This long-standing cryptographic problem has led to a prolific line of research, with many works focused on improving the communication, computation, and round complexities of MPC. However, most works focus on stateful protocols, which are not particularly well-suited for long-term operation. With this in mind, we ask the following question:

*Can we design a stateless MPC protocol under (not necessarily standard, but still) realistic assumptions, where parties participate only for a single round?*

In Chapter 2 we show that this is indeed possible. In our solution we utilize blockchains and *Conditional Storage and Retrieval* (CSaR) systems, an abstraction for (conditional) secret storage. Our MPC protocol achieves guaranteed output delivery, meaning that the adversary cannot prevent honest parties from obtaining the output. This property is essential when offering MPC as a service.

As CSaR systems can be somewhat complex, in Chapter 3 we design stateless MPC protocols without relying on this assumption. For this, we focus on the new *You Only Speak Once* (YOSO) model. This paradigm has been recently introduced by Gentry et al. [55], and it is tailored to large-scale stateless environments such as blockchains. It introduces an abstraction of so-called roles, which can be seen as stateless parties that only send a single message; i.e., they *speak only once*. The role-assignment mechanism assigns roles to the physical machines that execute these, and it is typically abstracted away from the protocol itself. The latter then relies on roles to perform the needed functionality (for example,

MPC).

In YOSO, our goal is to advance the state of the art in terms of communication complexity. Towards this, we use techniques from traditional stateful MPC with good communication complexity to design a YOSO MPC protocol in the preprocessing model, where the communication complexity of the online phase is better than the state of the art by a linear factor.

Finally, in Chapter 4 we complete our exploration of the YOSO paradigm by shifting our focus from generic YOSO MPC to a specialized multi-party functionality: distributed randomness generation. Here, a number of mutually distrusting parties, each with its own local source of randomness, work together to produce public, unpredictable, and unbiased randomness. Such randomness is a key component in numerous financial and cryptographic applications. Traditionally, it was obtained via trusted third parties. However, as this approach has the same issues as all centralized protocols, recently there has been an increased effort to obtain *distributed* randomness generation protocols.

In our work we provide such randomness generation protocols in the setting of *YOSO with worst-case corruptions* ($\text{YOSO}^{\mathsf{WCC}}$), a strengthening of the original YOSO MPC model, which was recently introduced by Nielsen, Ribeiro and Obremski [92]. Both models are similar in that they utilize stateless roles and allow these to speak only once. However, MPC protocols in the original YOSO model typically assume that the adversary's best strategy is to corrupt machines *at random*. The intuition for this is the following: Assuming that the role-assignment mechanism is secure and hides the assignment before the parties send their single protocol message, and assuming that parties erase their state before sending their message, the adversary simply does not know which party is the next to participate. Hence, it also does not know which party to corrupt! However, this places high trust assumptions on the role-assignment mechanism, which in turn makes such protocols hard to design. Indeed, known role-assignment protocols are fairly complex, and tend to compromise either in terms of efficiency [56] or resiliency [21]. Allowing for worst-case corruptions, as is done in $\text{YOSO}^{\mathsf{WCC}}$, allows us to reduce the trust on the role-assignment mechanism.

In our work we provide a suite of $\text{YOSO}^{\mathsf{WCC}}$ distributed randomness generation protocols in the computational setting, which offer different trade-offs in terms of efficiency, corruption threshold, and underlying assumptions. We complement our results by studying lower bounds for such computationally secure protocols.

In the remainder of this chapter we give a technical overview of this thesis.

## 1.1 Blockchains Enable Non-Interactive MPC

In our first contribution we provide a solution for MPC which achieves the property that each MPC participant who supplies inputs but does not wish to receive the output can go offline after sending only a single message. Additionally, parties who execute the MPC protocol do not need to interact with one another, are stateless, and it is sufficient if they participate only for short periods of time. We further provide variations of our protocol that offer further desirable properties such as guaranteed output delivery.

To obtain such non-interactive MPC, we assume a public key infrastructure (PKI). Additionally, we rely what we call *conditional storage and retrieval systems* (CSaRs). Roughly, CSaR systems allow a user to submit a secret along with a release condition. Later, if a (possibly different) user is able to satisfy this release condition, the secret is privately sent to this user. During the process, the secrets cannot be modified and no information is leaked about the secrets. We use blockchains to design a system called *extractable witness encryption on a blockchain* (eWEB) which fits this abstraction. Finally, we assume the existence of append-only bulletin boards that allow parties to publish data and receive a confirmation in return that the data was published. As both eWEB and bulletin boards can be realized using blockchains [43, 61, 79], for simplicity, in the following we will state that we design our protocols in the blockchain model.

In our construction, we distinguish between parties who supply inputs (dubbed *contributors*) and parties who wish to receive outputs (dubbed *evaluators*). At a high level, the construction transforms any MPC protocol into one which requires no interaction between the participants by having the contributors "encrypt" the next-message functions for each round of MPC while hardcoding their private inputs. These encryptions are then stored using CSaR. During the MPC execution, the evaluators can then decrypt the next-message functions round-by-round (and party-by-party) in a controlled way to eventually obtain the output of the protocol. To ensure that privacy of the intermediate outputs holds, the next-message functions of the underlying MPC are modified slightly to provide *encryptions* of the state (instead of the state in plain) that must be passed to the next round. Importantly, it is not necessary for an evaluator to contribute to every round of the underlying MPC – it is sufficient to contribute for one round (then, another evaluator can take over).

We further discuss ways to optimize our protocol. Towards this, we note that the *combined communication- and state complexity* of the underlying MPC is particularly important in our construction. This is because both the communication- and state complexities of the underlying MPC translate into the number of CSaR storage- and retrieval requests (and

thus communication complexity) in our overall construction. While there are a number of works optimizing communication complexity of MPC, optimizing internal state complexity has been largely overlooked. We design a scheme in the plain model which relies on multi-key fully homomorphic encryption (MFHE), probabilistically checkable proofs, collision-resistant hash functions, and IND-CPA secure public key encryption. Its combined communication- and state complexity is independent of the function that we are computing.

By combining this MPC protocol with our main construction we obtain an MPC protocol in the blockchain model which in addition to being stateless has the property that the communication complexity of this protocol is independent of the function that is being computed using this MPC protocol.

Finally, we design an MPC scheme which requires only a single message from the contributors with the additional property of *guaranteed output delivery*, i.e., the adversary cannot prevent honest parties from obtaining the output. For this, we in particular rely on the underlying protocol having guaranteed output delivery as well (and thus require the majority of the MPC contributors to be honest).

## 1.2 Towards Scalable YOSO MPC via Packed Secret-Sharing

In the previous section we described non-interactive MPC using blockchains, which can be seen as a stateless protocol under the assumption of a CSaR. As CSaR systems can get somewhat complex, in our next contribution we move to fully stateless distributed protocols which do not rely on the CSaR assumption. Towards this, we explore the YOSO paradigm [55], which is tailored to large-scale stateless computing environments such as blockchains.

The YOSO model is built around the notion of *roles*, which are stateless parties that send out only a *single* message. MPC is being performed by committees of such roles, where the size of the committees is much smaller than the number of parties in the system. Furthermore, the information about the party-role assignment of the next committee is typically hidden from the adversary. Such a paradigm has an interesting advantage: It enables MPC protocols which utilize only *small* committees (think only a few thousands out of a million total participants) and are secure in the face of a very powerful adversary who can potentially corrupt many parties (say half a million parties). In particular, the corruption budget of the adversary can be big enough to corrupt the *entire* MPC committee.

Somewhat surprisingly, assuming that parties erase their state after they have spoken, security holds even if the adversary is *adaptive*, meaning that it can corrupt any parties at any time, subject to the corruption budget.

The intuition here is that the adversary does not know *who* to corrupt – participants of the future committees are hidden until they speak, and corrupting them *after* they have spoken does not bring the adversary any advantage, as they will not participate in the protocol again[1].

While the YOSO paradigm is very appealing, the restriction to send out only a single message makes designing the MPC protocols in this model challenging, which leads to complex protocols with high communication complexity. Currently, the communication complexity per gate of the state-of the art YOSO MPC protocol in the computational setting is $O(n^2)$ (can be amortized to $O(n)$ given a wide circuit). In our work, we improve the communication complexity of YOSO MPC in order to bring it closer to practicality.

In more detail, we explore the *online/offline* paradigm, which has been extensively utilized in the standard MPC literature to achieve improvements (in particular, in terms of communication complexity) of the MPC protocols. We further focus on the setting where the YOSO committees not only have an honest majority (which is the standard assumption in this line of work), but there is a *gap* proportional to the committee size between the number of honest parties and the number of corrupt parties. In more detail, let $n$ be the committee size, and let $t$ be the amount of corruptions in the committee. In our work we consider the setting in which the committee satisfies $t < n(\frac{1}{2} - \epsilon)$, for some constant $\epsilon > 0$.

We obtain our results by adapting techniques from traditional (non-YOSO) MPC literature to the YOSO setting. Here, we consider Turbopack [50] as our starting point. Turbopack provides MPC with abort with constant online communication complexity. To obtain a YOSO MPC protocol that satisfies our goals in terms of security and efficiency, we need to solve a few technical issues. First, Turbopack is inherently stateful and requires multiple rounds; in order to adapt it to the YOSO setting we need a way to pass secret protocol state to future parties in a way that does not break security. Next, we note Turbopack is in the preprocessing model, and it crucially assumes that during the online phase, the parties know certain secrets that were obtained during the offline phase. However, in the YOSO world, parties who participate in the online phase are not the same as the ones who prepared the preprocessing material. More importantly, in YOSO we cannot even

[1]Technically, it is possible that a physical machine fulfills multiple roles. However, a machine that has spoken does not have a higher chance of participating in a later committee than any other machine in the system.

assume that the parties who execute the offline phase know which parties will be performing the online phase. Thus, we need to find an efficient way to pass the preprocessing secrets to the online parties, and do so *without knowing who these parties will be*. Finally, we note that Turbopack achieves only security with abort, while our goal is to provide guaranteed output delivery.

To solve these issues, we carefully combine the techniques from Turbopack with the ideas from the well-known CDN protocol [45]. This allows us to design a YOSO MPC protocol that has guaranteed output delivery and scales much better as $n$ grows, in contrast to the case when $\epsilon = 0$. In particular, we show how to achieve an online phase with communication complexity that is *independent* of the committee size $n$.

## 1.3 Improved YOSO Randomness Generation with Worst-Case Corruptions

In our final chapter, we delve deeper into the YOSO paradigm, narrowing our focus from general-purpose MPC to the specialized multi-party functionality of distributed randomness generation. In more detail, we consider the following problem: Given $n$ mutually distrusting parties, each with its own local source of randomness, we wish to generate a public random bit. The generated bit must be unpredictable and unbiasable, even though some $t$ of the parties can be corrupt.

We design our protocols in the setting of YOSO with worst-case corruptions, a recently introduced model [92] which is specifically tailored to the problem of randomness generation. Just as in the original YOSO model, in YOSO with worst-case corruptions ($YOSO^{\mathsf{WCC}}$) we distinguish between stateless "roles" and physical machines which may run for a long time and retain state. The $n$ participating roles $P_1, \ldots, P_n$ are executed in a linear fashion one after the other, and the adversary is allowed to corrupt any $t$ of them prior to the start of the protocol. Upon its execution, $P_i$ can publicly broadcast a value $x_i$ and send secret values $s_{i,j}$ to each "future" role $P_j$, i.e., any $P_j$ with $j > i$. After the execution of the last role $P_n$, anyone should be able to obtain an unbiased public random bit by applying a publicly known and deterministic extraction function to the broadcasted values $(x_1, \ldots, x_n)$.

Intuitively, any traditional stateful protocol for randomness generation can be translated into one in the YOSO$^{\mathsf{WCC}}$ setting: Given an $r$-round stateful protocol for $n$ participants and $t$ corruptions, we can "linearize" it by using $r$ roles to implement the behavior of the $i$-th party over $r$ rounds. To mimic the stateful behavior of the round-based protocol, each

role in YOSO$^{\mathsf{WCC}}$ sends the adjusted "state" to the role which implements the behavior of the next round of the same stateful party. Clearly, if the round-based protocol is secure, its YOSO$^{\mathsf{WCC}}$ counterpart is secure as well. Unfortunately, this approach has subpar resilience: It tolerates only $t$ corruptions while requiring $n * r$ roles.

In our work, we propose multiple YOSO$^{\mathsf{WCC}}$ randomness generation protocols in the computational setting with trade-offs in terms of communication, resiliency, and required assumptions. At the core of our constructions lies the well-known *commit-and-reveal* paradigm: Intuitively, we let roles "commit" to their randomness, and forward the reveal information to the roles in the future who help with opening these values. After $t + 1$ roles have committed their randomness, each random value is opened and the output is computed by taking the xor of the revealed values. By using $t + 1$ roles who contribute randomness, we ensure that at least one contributed random value is coming from an honest party. Further, the commitments to the values ensure that a malicious party cannot change the value they committed to after seeing the opening values of honest parties. While this already looks like a secure solution, there is a subtlety: We must ensure that the committed values are *guaranteed* to be opened. This is because otherwise an adversarial party could simply commit to, say, bit 1, and later decide whether to flip the outcome of the xor by revealing the opening or not, and do so after seeing honest values.

To ensure this final property, in our first protocol we rely on a cryptographic primitive known as publicly verifiable secret sharing (PVSS). In PVSS, a dealer can share its secret among $n$ parties in a way that any $t+1$ parties can reconstruct the secret, but any $t$ parties have no information about the secret. Public verifiability of PVSS ensures that anyone (even non-recipients) can verify that the dealer performed the sharing correctly, and there indeed exists a *unique* secret which can be later reconstructed by any set of $t + 1$ recipient parties. By relying on PVSS with some further properties, we fix the issue above and ensure that the adversary is not able to bias the outcome.

As PVSS requires setup and somewhat heavy cryptographic assumptions, in our next construction we adopt *verifiable secret sharing* (VSS), a primitive which is similar to PVSS, except that it does not provide public verifiability. Instead, VSS provides a "strong commitment" property, which ensures that the shares of the honest parties define a secret (which could be $\perp$). We design our next randomness generation protocols based on a YOSO$^{\mathsf{WCC}}$-friendly variation of a VSS primitive, which we dub *split-dealer VSS*.

Finally, we design a protocol which assumes non-interactive perfectly binding commitments and reduces the number of roles in the setting without setup, though at the cost of exponential communication and computation complexity. We complement our results by

studying lower bounds (in terms of role complexity) for computationally secure YOSO$^{\mathsf{WCC}}$ protocols without setup.

### 1.3.1 Summary

To summarize, in this thesis we advance the state of the art in stateless secure multi-party computation. We study stateless MPC under various models and assumptions – from CSaRs to YOSO with its role-assignment. We then narrow our focus and explore a specialized MPC functionality – randomness generation. Last but not least, in addition to being stateless, most our protocols require parties to speak only once and provide guaranteed output delivery, a highly desirable property in MPC as a service.

# Chapter 2

# Blockchains Enable Non-Interactive MPC

*What, like it's hard?*

Elle Woods
Legally Blonde

We start our exploration of distributed cryptography as a service by using blockchains to design a protocol which achieves non-interactive MPC, a powerful primitive which is known to be impossible in the standard model.

The work presented in this chapter is based primarily on a joint work with Vipul Goyal, Bryan Parno, and Yifan Song, which has been published at TCC 2021 [65]. As the main author, I formalized the key primitive which we call conditional storage and retrieval systems (CSaR), and designed a secure CSaR-based protocol which enables MPC where participants do not need to interact with each other. Additionally, this chapter includes the eWEB protocol, an instantiation of a CSaR system, which I designed during another joint work with the same co-authors along with Abhiram Kothapalli. This work has been published at PKC 2022 [64].

## 2.1 Introduction

Secure Multiparty Computation (MPC) [59, 104] enables parties to evaluate an arbitrary function in a secure manner, i.e., without revealing anything besides the outcome of the

computation. MPC is increasingly important in the modern world and allows people to securely accomplish a number of difficult tasks. Obtaining efficient MPC protocols is thus a relevant problem and it has indeed been extensively studied [54, 59, 104]. One important question is the round complexity of MPC schemes. In the semi-honest case, in 1990, Beaver et al. [15] gave the first constant-round MPC protocol for three or more parties. A number of works [63, 80, 93] aiming to analyze and reduce round complexity followed, both in the semi-honest and fully malicious models. In 2016, Garg et al. [54] proved that four rounds are necessary to achieve secure MPC in the fully malicious case in the plain model. Four round MPC protocols have been recently proposed [10, 29, 41], resolving the questions of round complexity.

Unfortunately, solutions that achieve even the optimal round complexity are still problematic for many applications since these solutions typically require synchronous communication from the participants – imagine for example the U.S. voting process. If the voting is conducted via secure multi-party computation, all participants are required to be online at the same time. It is unrealistic to assume that all of the eligible U.S. voters can be persuaded to be online at the same time on the Election Day. In this work, we rely on blockchains to achieve MPC that *does not require participants to be online at the same time or interact with each other.*

Such non-interactive solutions advance the state of the art of secure multi-party computation, opening up a whole new realm of possible applications. For example, *passive data collection* for privacy preserving collaborative machine learning becomes possible. Federated learning is already used to train machine learning models for the keyboards of mobile devices for the purposes of autocorrect and predictive typing [2]. Unfortunately, using off-the-shelf MPC protocols to perform such training securely is not straight-forward. Not all smartphones are online at the same time and it might even be unknown how many devices will end up participating. In contrast, off-the-shelf MPC protocols typically assume that all (honest) participants are indeed online during some time period, and the number of participants is known. This leads us to the following question:

*Can we construct a secure MPC protocol which does not require the parties to be online at the same time and guarantees privacy and correctness even if all but one of the parties are fully malicious? Is it possible to design a protocol which requires only a single round of participation from the parties supplying the inputs, and allows the parties to go offline after the first round if they are not interested in learning the output?*

Consider such a protocol in the use case outlined above – each smartphone could

independently send a single message to a server, and at the end of the collection period the server would obtain the model trained on the submitted inputs, all while preserving the privacy of the gathered inputs.

### 2.1.1 Our Results

In our work, we provide a solution for MPC which achieves the property that each MPC participant who supplies inputs but does not wish to receive the output can go offline after the first round. The participants are not required to interact with each other. We additionally provide variations of our protocol that offer further desirable properties.

Before we provide the formal theorem statements, we discuss the protocol execution model and the notation.

In our work, we assume the existence of append-only bulletin boards that allow parties to publish data and receive an unforgeable confirmation that the data was published in return. Furthermore, we assume a public key infrastructure (PKI). Finally, we rely on conditional storage and retrieval systems (CSaRs, see Section 2.3 for details). Roughly, CSaR systems allow a user to submit a secret along with a release condition. Later, if a (possibly different) user is able to satisfy this release condition, the secret is *privately* sent to this user. Intuitively, during the process, the secrets cannot be modified and no information is leaked about the secrets. We require that CSaRs are used as ideal functionalities. We give an instantiation of a CSaR system which we call *eWEB* – extractable witness encryption on a blockchain. As eWEB relies on blockchains, and bulletin boards can be realized using blockchains as well [43, 61, 79], for simplicity, we will state that we design our protocols in the blockchain model. Finally, we assume IND-CCA secure public key encryption, and digital signatures.

In our construction, we distinguish between parties who supply inputs (dubbed *MPC contributors*) and parties who wish to receive outputs (dubbed *evaluators*). Our construction is a protocol transforming an MPC scheme $\pi$ into another scheme $\pi'$. The contributors in $\pi'$ are exactly the participants in $\pi$. The evaluators can (but do not have to) be entirely different parties from those who contribute inputs in $\pi$.

We are now ready to introduce our first result:

**Theorem 1. (Informal)** *Any MPC protocol $\pi$ secure against fully-malicious adversaries can be transformed into another MPC protocol $\pi'$ in the blockchain model that provides security with abort against fully-malicious adversaries and does not require participants*

*to be online at the same time. The MPC contributors are required to participate for only a single round (the evaluators might be required to participate for multiple rounds). The adversary is allowed to corrupt as many MPC contributors in $\pi'$ as it is allowed to corrupt participants in $\pi$. The adversary is allowed to corrupt any number of evaluators.*

In addition to this result, we discuss ways to optimize our construction. To this end, we explain why the combined communication and state complexity (where state complexity is the amount of data that parties maintain between the different rounds of the protocol execution) of the underlying MPC protocol is of a particular importance in our construction. Briefly, both the communication and state complexities of the underlying MPC translate directly into the number of CSaR storage and retrieval requests in our overall construction. We describe a protocol in the plain model which relies on multi-key fully homomorphic encryption (MFHE). Its combined communication and state complexity is independent of the function that we are computing. While optimizing communication complexity has received considerable attention in the community in the past few years, optimizing internal state complexity has been largely overlooked. We believe that this particular problem might be exciting on its own. In our construction which optimizes the combined communication and state complexity, we assume multi-key fully homomorphic encryption, and collision-resistant hash functions. The result that we achieve here is the following:

**Theorem 2. (Informal)** *Let $f$ be an $N$-party function. Protocol 14 is an MPC protocol computing $f$ in the standard model and secure against fully malicious adversaries corrupting up to $t < N$ parties. Its combined communication and state complexity depends only on the security parameter, number of parties, and input and output sizes. In particular, the combined communication and state complexity is independent of the function $f$.*

Using this MPC protocol in combination with our first construction, under the assumptions that we rely on in our main construction and in the MPC construction with optimized communication and state complexity, we achieve the following:

**Corollary 1. (Informal)** *There exists an MPC protocol $\pi'$ in the blockchain model which provides security with abort against fully-malicious adversaries and does not require participants to be online at the same time. The MPC contributors are required to participate for a single round (the evaluators might be required to participate for multiple rounds). Furthermore, the number of calls to CSaR in this protocol is independent of the function that is being computed using this MPC protocol* [1].

Finally, we achieve an MPC protocol which requires only a single round of participation

[1] A prior version of this paper erroneously stated that the communication complexity (instead of the number of CSaR calls) is independent of the function being computed.

from MPC contributors with the additional property of *guaranteed output delivery*, meaning that adversarial parties cannot prevent honest parties from receiving the output. For this, we in particular rely on the underlying protocol having guaranteed output delivery as well (and thus requiring the majority of the MPC contributors to be honest). We rely on the same assumptions (PKI, CSaRs, append-only bulletin boards etc.) as the ones used in our main construction. The formal result that we achieve is the following:

**Theorem 3. (Informal)** *Any MPC protocol $\pi$ that is secure against fully-malicious adversaries and provides guaranteed output delivery can be transformed into another MPC protocol $\pi'$ in the blockchain model that provides security with guaranteed output delivery against fully-malicious adversaries and does not require participants to be online at the same time. The MPC contributors and evaluators are required to participate for only a single round. The adversary is allowed to corrupt as many MPC contributors in $\pi'$ as it is allowed to corrupt participants in $\pi$. The adversary is allowed to corrupt all but one evaluator.*

### 2.1.2 Technical Overview

In this work, we propose an MPC protocol that does not require participants to be present at the same time. In order to do so, we rely on the following cryptographic building blocks – garbled circuits [18, 104, 106], CSaRs, and bulletin boards with certain properties. Before we introduce the construction idea, we elaborate on each of these primitives.

Roughly, a garbling scheme allows one to "encrypt" (garble) a circuit and its inputs such that when evaluating the garbled circuit only the output is revealed. In particular, no information about the inputs of other parties or intermediate values is revealed by the garbled circuit or during its evaluation. In our construction we use Yao's garbled circuits [104, 106].

In our construction, we rely on bulletin boards which allow parties to publish strings on an append-only log. It must be hard to modify or erase contents from this log. Additionally, we require that parties receive a confirmation ("proof of publish") that the string was published and that other parties can verify this proof. Such bulletin boards have been extensively used in prior works [43, 61, 79] and as pointed out by these works can be realized both from centralized systems such as the Certificate Transparency project [1] and decentralized systems such as proof-of-stake or proof-of-work blockchains.

Finally, we define the CSaR primitive. Roughly, this primitive allows for the distributed and secure storage and retrieval of secrets and realizes the following ideal functionality:

- Upon receiving a secret along with a release condition and an identifier, if the identifier was not used before, the secret is stored and all participants are notified of a valid secret storage request. The release condition is simply an NP statement.
- Upon receiving an (identifier, witness) from a user, the ideal functionality checks whether a secret with this identifier exists and if so, whether the given witness satisfies the release condition of this secret record. If so, the secret is sent to the user who submitted the release request.

We can instantiate the CSaR with *eWEB*, which stands for "Extractable Witness Encryption on a Blockchain". Roughly, it allows users to encode a secret along with a release condition and store the secret on a blockchain. Once a user proves that they are able to satisfy the release condition, blockchain miners jointly and privately release the secret to this user. Along the way, no single party is able to learn any information about the secret.

**Our construction.** By relying on bulletin boards, Yao's garbled circuits and CSaRs, we are able to *transform any secure MPC protocol $\pi$ into another secure MPC protocol $\pi'$ that provides security with abort and does not require participants to be online at the same time.* At a high level, our idea is as follows: first, each contributor (party who supplies inputs in the protocol) $P$ in the MPC protocol $\pi$ garbles the next-message function for each round of $\pi$. Then, $P$ stores the garbled circuits as well as the garbled keys with a CSaR using carefully designed release conditions. Note that each party $P$ is able to do so individually, without waiting for any information from other parties and can go offline afterwards. Once all contributors have stored their data with the CSaR, one or more "evaluators" (parties who wish to receive the output) interact with the CSaR and use the information stored by the MPC contributors in order to retrieve the garbled circuits and execute the original protocol $\pi$. The group of the contributors and the group of evaluators do not need to be the same – in fact, these groups can even be disjoint. *The evaluators might change from round to round.*

Note that while the high-level overview is simple, there are a number of technical challenges that we must overcome in the actual construction due to its non-interactive nature. For example, since the security of Yao's construction relies on the fact that for each wire only a single key is revealed, we must ensure that each honest garbled circuit is executed only on a *single* set of inputs. The adversary also must not trick a garbled circuit of some honest party $A$ into thinking that a message broadcast by some party $C$ is message $m$, and tricking a garbled circuit of another honest party $B$ into thinking that $C$ in fact broadcast message $m' \neq m$. Furthermore, we must ensure that it is hard to execute the

protocol "out of order", i.e., an adversary cannot execute some round $i$ prior to round $j$ where $i > j$. Such issues do not come up in the setting where parties are online during the protocol execution and able to witness messages broadcast by other parties.

We solve these issues by utilizing bulletin boards, carefully constructing the release conditions for the garbled circuits and the wire keys, and modifying the next-message functions which must be garbled by the contributors.

Note that the next-message functions from round two onward take as inputs messages produced by the garbled circuits in prior rounds. At the time when the MPC contributors are constructing their circuits, the inputs of other parties are not known, and thus it is not possible to predict which wire key (the one corresponding to 0 or the one corresponding to 1) will be needed during the protocol execution. At the same time, one cannot simply make both wire keys public since the security of the garbled circuit crucially relies on the fact that for each wire only a single wire key can be revealed. We solve this problem by storing both wire keys with the CSaR, utilizing bulletin boards, and requiring the evaluators to publish the output of the garbled circuits of each round. Then, (part of) the CSaR release condition for the wire key corresponding to bit $b$ on some wire $w$ of some party's garbled circuit for round $i$ is that the message from round $i-1$ is published and contains bit $b$ at position $w$. This way we ensure that while both options for wire $w$ are "obtainable", only the wire key for bit $b$ (the one that is needed for the execution) is revealed.

Next, note that in our construction we specifically rely on Yao's garbled circuits. Yao's construction satisfies the so-called "selective" notion of security, which requires the adversary to choose its inputs before it sees the garbled circuit (in contrast to the stronger "adaptive" notion of security which would allow the adversary to choose its inputs *after* seeing the garbled circuits [17]). We ensure that the selective notion of security is sufficient for our construction by requiring that not only the wire keys, but also the garbled circuits are stored with the CSaR. The release conditions both for the garbled circuit for some round $i$ and all its wire keys require a proof that all messages for rounds 1 up to and including round $i-1$ are published by the evaluators. This way, the evaluators are required to "commit" to the inputs before receiving the selectively secure garbled circuits, which achieves the same effect as adaptive garbled circuits.

As outlined above, we must ensure that it is hard for the adversary to trick the garbled circuit produced by some honest party $A$ into accepting inputs from another honest party $B$ that were not produced by $B$'s circuits. We accomplish this by modifying the next-message function of every party $A$ as follows: in addition to every message $m$ that is produced by some party $B$, the next-message function takes as input a signature $\sigma$ on $m$ as well and

verifies that the signature is correct. If this is not the case for any of the input messages, the next-message function outputs $\perp$. Otherwise, the next-message function proceeds as usual and in addition to outputting the resulting message it outputs the signature of party $A$ on this message.

Our end goal is to reduce the security of our construction to the security of the underlying MPC protocol $\pi$. While utilizing bulletin boards and introducing signatures is a good step forward, we must be careful when designing the CSaR release conditions. The adversary could sign multiple messages for each corrupted contributor in $\pi$, publish these messages on the bulletin board and thus receive multiple keys for some wires. To prevent this, the CSaR release condition must consider only the *very first* message published for round $i-1$ on the bulletin board. This way, we ensure that there is only a single instance of the MPC running (only a single wire key is released for each circuit): even if the adversary is able to sign multiple messages on behalf of various MPC contributors, only the very first message published on the bulletin board for a specific round will be used by the CSaR system to release the wire keys for the next round.

The ideas outlined above are the main ideas in our protocol. We now elaborate on a few additional details:

The next-message function of the protocol typically outputs not only the message for the next round, but also the state which is used in the next round. It is assumed that this state is kept private by the party. In our case, the output of the next-message function will be output by the garbled circuit and thus made available to the evaluator. To ensure that the state is kept private, we further modify the next-message function to add an encryption step at the end: the state is encrypted under the public key of the party who is executing this next-message function. To ensure that the state can be used by the garbled circuit of the party in the next round, we add a state decryption step at the beginning of the next-message function of that round. Similar to the public output of the next-message function, we compute a signature on the encryption of the state and verify this signature in the garbled circuit of the next round.

Note that in the construction outlined above, we use some secret information which does not depend on the particular execution but still must be kept private (secret keys of the parties used for the decryption of the state, signing keys used to sign the output of the next-message function etc.). This information is hard-coded in the garbled circuits. We explain how this can be done in Section 2.4.

Finally, note that we require the following property from the underlying protocol $\pi$:

given a transcript of execution of $\pi$, the output of $\pi$ can be publicly computed. As we note in Section 2.4, this property can be easily achieved by slightly modifying the original protocol $\pi$.

We provide all protocol details and outline optimizations in Section 2.4 and give the formal construction in Protocols 6, 7 and 8. The formal security proof is done by providing a simulator for the construction and proving that an interaction with the simulator in the ideal world is indistinguishable from the interaction with an adversary in the real world.

To summarize, using the construction sketched above we achieve the following result:

**Theorem 4. (Informal)** *Protocols 6, 7 and 8 transform any MPC protocol $\pi$ secure against fully-malicious adversaries into another MPC protocol $\pi'$ in the blockchain model that provides security with abort against fully-malicious adversaries and does not require participants to be online at the same time. The MPC contributors are required to participate for only a single round (the evaluators might be required to participate for multiple rounds). The adversary is allowed to corrupt as many MPC contributors in $\pi'$ as it is allowed to corrupt participants in $\pi$. The adversary is allowed to corrupt any number of evaluators.*

In addition to our main protocol that requires only one message from the MPC contributors and does not require any additional functionality from the CSaR participants apart from the core CSaR functionality itself (storing and releasing secrets), we provide a number of variations that have further desirable properties, such as guaranteed output delivery. We now outline these further contributions.

**Improving Efficiency.** The efficiency of our construction is strongly tied to the efficiency of the underlying MPC protocol $\pi$. Note that in our construction each input wire key of each garbled circuit is stored with the CSaR, and the inputs of the garbled circuits are exactly messages exchanged between the parties *as well as the state information passed from previous rounds*. Thus, the communication and state complexities translate directly into the number of CSaR store and release operations that the MPC contributors, as well as later the evaluators, must make. In order to reduce the number of CSaR invocations, we describe an MPC protocol which optimizes the *combined communication and internal-state complexity*. While communication complexity is typically considered to be one of the most important properties of an MPC protocol, state complexity receives relatively little attention. Our main construction shows that there are indeed use cases where *both* the communication and the state complexity matter, and we initiate a study of the combined state and communication complexity.

Specifically, we introduce an MPC protocol in which the combined communication and

state complexity is independent of the function we are computing. We achieve it in two steps: we start with a protocol secure against semi-malicious adversaries [2] which at the same time has communication and state complexity which is independent of the function that is being computed. Then, we extend it to provide fully malicious security while taking care to retain the attractive communication and state complexity properties in the process.

In more detail, we start with the MPC construction by Brakerski et al. [29] which is based on multi-key fully homomorphic encryption (MFHE) and achieves semi-malicious security. We note that the communication and state complexity of this construction depends only on the security parameters, the number of parties, and the input and output sizes. In particular, note that the construction's combined communication and state complexity is independent of the function we are computing.

Our next step is to extend this construction so that it provides security against malicious adversaries. For this, we propose to use the zero-knowledge protocol proposed by Kilian [82] that relies on probabilistically checkable proofs (PCPs) and allows a party $P$ to prove the correctness of some statement $x$ to the prover $V$ using a witness $w$. Along the way, we need to make minor adjustments to Kilian's construction because its state complexity is unfortunately too high for our purposes – in particular, in the original construction, the entire PCP string is stored by the prover to be used in later rounds. After making a minor adjustment – recomputing the PCP instead of storing it – to the construction to address this issue, we use this scheme after each round of the construction by Brakerski et al. in order to prove the correct execution of the protocol by the parties. The resulting construction achieves fully malicious security, and its communication and state complexities are still independent of the function that we are computing.

We provide the details of the construction and analyse its security and communication/state complexity properties in Section 2.9 with the formal protocol description in Protocol 14. In this protocol, we assume the existence of an MFHE scheme with circular security and the existence of a collision-resistant hash functions. We are able to achieve the following result which may be of independent interest:

**Lemma 1. (Informal)** *Let $f$ be an $N$-party function. Protocol 14 is an MPC protocol computing $f$ in the plain (authenticated broadcast) model and secure against fully malicious adversaries corrupting up to $t < N$ parties. Its communication and state complexity depend only on security parameters, number of parties, and the input and output sizes. In particular, the communication and state complexity of Protocol 14 is independent of the function $f$.*

[2] Intuitively, semi-malicious adversaries can be viewed as semi-honest adversaries which are allowed to freely choose their random tapes.

Using this MPC protocol in combination with our first construction, under the assumptions that we rely on in our main construction and in the MPC construction with optimized communication and state complexity, we achieve the following:

**Corollary 2. (Informal)** *There exists an MPC protocol $\pi'$ in the blockchain model that has adversarial threshold $t < N$, provides security with abort against fully-malicious adversaries and does not require participants to be online at the same time. Only a single message is required from the MPC contributors (the evaluators might be required to produce multiple messages). Furthermore, the number of calls to CSaR of this protocol is independent of the function that is being computed using this MPC protocol.*

**Non-Interactive MPC with Guaranteed Output Delivery (GoD).** We need to modify our construction in order to provide guaranteed output delivery. In order to achieve GoD, we require the protocol $\pi$ to have the GoD property as well, and thus the majority of the participants in $\pi$ (recall that these are exactly the contributors in $\pi'$) must be honest [3]. While making this change (in addition to a few minor adjustments) would be enough to guarantee GoD in our construction in the setting with only a single evaluator, it is certainly not sufficient when there are multiple evaluators, some of them dishonest. This is due to the following issue: since we must prevent an adversary from executing honest garbled circuits on multiple different inputs, we cannot simply allow each evaluator to execute garbled circuits on the inputs of its choosing. In particular, the CSaR release conditions must ensure that for each wire only a *single* key is revealed. In our first construction this results in the malicious evaluator being able to prevent an honest evaluator from executing the garbled circuits as intended by submitting an invalid first message for any round. Thus, to ensure guaranteed output delivery while maintaining secrecy, we must ensure that a malicious evaluator posting a wrong message does *not* prevent an honest evaluator from posting a correct message and using it for the key reveal. In particular, we will ensure that only a *correct* (for a definition of "correctness" explained below) message can be used for the wire key reveal.

Note that the inputs to the garbled circuits depend on the evaluators' outputs from the previous rounds. Checking the "correctness" of the evaluators' outputs is not entirely straight-forward since an honest execution of a garbled circuit which was submitted by a dishonest party might produce outputs which look incorrect (for example, have invalid signatures). Thus, simply letting the CSaR system check the signatures on the messages supplied by the evaluators might result in an honest evaluator being denied the wire keys for the next round.

[3] Note that there is no such restriction on the evaluators in $\pi'$.

In our GoD construction we overcome this issue largely using the following adjustments:

- all initial messages containing garbled circuits and wire keys are required to be posted before some deadline.
- we use a CSaR with public release (whenever a secret is released, it is released publicly and the information can be viewed by anyone).
- we ensure that it is possible to distinguish between the case where the evaluator is being dishonest, and the case where the evaluator is being honest, but the contributor in $\pi$ supplied invalid garbled circuits or keys, or did not supply some required piece of information.

We achieve the last point by designing the CSaR release condition in a way that it verifies that the evaluator's output can be explained by the information stored by the contributors in $\pi$. In particular, as part of the CSaR's release condition, we require a proof of correct execution for the garbled circuit outputs. The relation that the CSaR system is required to check in this case is roughly as follows: "The execution of the garbled circuit $GC$ on the wire keys $\{k_i\}_{i\in I}$ results in the output provided by $E$. Here, the garbled circuit $GC$ is the circuit, and $\{k_i\}_{i\in I}$ are the keys for this circuit reconstructed using the values published by the CSaR which are present on the proof of publish supplied by $E$". Note that due to the switch to the CSaR with public release, the wire keys used for the computation are indeed accessible to the CSaR system after their first release.

Similar to our first construction, we eventually reduce the security of the new protocol to the security of the original protocol. In addition to our first construction however, since the CSaR system is now able to verify messages submitted by the evaluators, honest evaluators are always able to advance in the protocol execution. This insight allows us to ensure that honest evaluators do not need to abort with more than a negligible probability along the way. Thus, if the underlying protocol $\pi$ achieves guaranteed output delivery, the protocol we propose achieves guaranteed output delivery as well.

We give full details of the GoD construction in Section 2.6. The statement about our GoD construction is given below.

**Lemma 2. (Informal)** *Any MPC protocol $\pi$ which is secure against fully-malicious adversaries and provides guaranteed output delivery can be transformed into another MPC protocol $\pi'$ in the blockchain model that provides security with guaranteed output delivery against fully-malicious adversaries and does not require participants to be online at the same time. The MPC contributors are required to participate for only a single round (the evaluators might be required to participate for multiple rounds). The adversary is allowed to corrupt as many MPC contributors in $\pi'$ as it is allowed to corrupt participants in $\pi$. The*

*adversary is allowed to corrupt any number of evaluators.*

### 2.1.3 Related Work

Closest to our work is the line of research that studies non-interactive multiparty computation [51, 73, 75], initiated in 1994 by Feige et al. [51], in which a number of parties submit a single message to a server (evaluator) that, upon receiving all of the messages, computes the output of the function. In their work, Feige et al. allow the messages of the parties to be dependent on some shared randomness that must be unknown to the evaluator. Unfortunately, this means that if the evaluator is colluding with one or more of the participants, the scheme becomes insecure. Overcoming this restriction, Halevi et al. [75] started a line of work on non-interactive *collusion-resistant* MPC. Their model of computation required parties to interact *sequentially* with the evaluator (in particular, the order in which the clients connect to the evaluator is known beforehand). Beimel et al. [16] and Halevi et al. [74] subsequently removed the requirement of sequential interaction. Further improving upon these results, the work of Halevi et al. [73] removed the requirement of a complex correlated randomness setup that was present in a number of previous works [16, 60, 74]. Halevi et al. [73] work in a public-key infrastructure (PKI) model in combination with a common random string. As the authors point out, PKI is the minimal possible setup that allows one to achieve the best-possible security in this setting, where the adversary is allowed to corrupt the evaluator and an arbitrary number of parties and learn nothing more than the so-called "residual function", which is the original function restricted to the inputs of the honest parties. In particular, this means that the adversary is allowed to learn the outcome of the original function on *every possible* choice of adversarial inputs.

Our work differs from the line of work on non-interactive MPC described above in a number of aspects. In contrast to those works, our construction is not susceptible to the adversary learning the residual function – roughly because the adversary must effectively "commit" to its input, and the CSaR system ensures that the adversary only receives a single set of wire keys per honest garbled circuit (the set of wire keys that aligns with the adversarial input). Additionally, in our work the parties do not need to directly communicate with the evaluator. In fact, in our construction that ensures guaranteed output delivery, any party can *spontaneously* decide to become an evaluator and still receive the result – there is no need to rerun the protocol in this case.

Related to us are also the works on reusable non-interactive secure computation (NISC) [5, 9, 12, 33, 38], initiated by Ishai et al. [77]. Intuitively, reusable NISC allows a receiver to

publish a reusable encoding of its input $x$ in a way that allows any sender to let the receiver obtain $f(x, y)$ for any $f$ by sending only a single message to the receiver. In our work, we focus on a multi-party case, where a party that does not need the output is not required to wait for other parties to submit their inputs.

Recently, Benhamouda and Lin [23] proposed a model called *multiparty reusable Non-Interactive Secure Computation (mrNISC) Market* that beautifully extends reusable NISC to the multiparty setting. In this model, parties first commit their inputs to a public bulletin board. Later, the parties can compute a function on-the-fly by sending a public message to an evaluator. An adversary who corrupts a subset of parties learns nothing more about the secret inputs of honest parties than what it can derive from the output of the computation. Importantly, the bulletin board commitments are reusable, and the security guarantee continues to hold even if there are multiple computation sessions. In their work, Benhamouda and Lin mention that any one-round construction is susceptible to the residual attacks and thus slightly relax the non-interactive requirement in order to solve this problem. Indeed, their construction can be viewed as a 2-round MPC protocol with the possibility to reuse messages of the first round for multiple computations. Our scheme shows that when using blockchains it is indeed possible to provide a construction that requires only a single round of interaction from the parties supplying the input and is nonetheless not susceptible to residual attacks.

Concurrent to our work, Almashaqbeh et al. [6] recently published a manuscript which focuses on designing non-interactive MPC protocols which use blockchains to provide *short term* security without residual leakage. They focus on the setting where the inputs of all but one of the parties are public. In this setting, designing one-round MPC can be done easily by having all parties send their input to the only party which holds the secret input. This party can then compute the output and distribute it to other parties. The authors are able to extend the setting to the two-party semi-honest private input setting where one round protocols for the party not getting the output can be easily designed as well. While our protocol provides a worst-case security guarantee, they focus on an incentive-based notion of security. While both constructions bypass the residual leakage issue, their security guarantees might degrade with time. The key challenge in their setting is fairness / guaranteed output delivery which they solve using an incentive-based model of security. Hence their work is essentially unrelated to ours.

Finally, two blockchain-based works ([42] and [55]) focus on improving the flexibility of the MPC protocols. Choudhuri et al. [42] proposed the notion of *fluid* MPC, while Gentry et al. [55] proposed YOSO MPC (which we will discuss in detail in later chapters). Similar

to us, these constructions allow the MPC participants to leave after the first round if they are not interested in learning the output. However, to execute the MPC protocol both Choudhuri et al. and Gentry et al. require a number of committees of different parties which interact with each other, and each committee must provide honest majority. Our protocol preserves privacy of inputs even if there is a single evaluator who is dishonest.

## 2.2 Preliminaries

In this section we briefly discuss cryptographic building blocks used in our system.

### 2.2.1 MPC

In our work we consider MPC that allow a set of parties $\mathcal{P} = \{P_1, \ldots, P_n\}$ to securely compute the output of some function $f$. We specifically consider MPC protocols in the broadcast model [4], where all parties have access to a broadcast channel and each round consists of parties broadcasting messages to other parties that participate in the protocol. An MPC protocol specifies for each party and each round the so-called next-message function, which defines the computation that is performed by that particular party in that round, as well as the message that the party broadcasts in that round and the state that is passed to the next round. More formally:

**Definition 1.** *Given an interactive broadcast-only d-round MPC protocol, the* ***next-message function*** *for round $i$ of party $P_j$ is the function $(m_j^i, s_j^i) \leftarrow f(x_j, r_j^i, m^i, s_j^{i-1})$, where $x_j$ is $P_j$'s input in the MPC protocol, $r_j^i$ is the local randomness used by party $P_j$ in round $i$, $m^i = m_1^{i-1} \| m_2^{i-1} \| \ldots \| m_n^{i-1}$ is the concatenation of messages received by each party in round $i-1$ (note that since we consider a broadcast protocol all parties receive the same message), $s_j^i$ is an auxiliary state information output by $P_j$ in round $i$ ($s_j^0 = \perp$), and $m_j^i$ is the message output by $P_j$ in round $i$.*

Note: we assume that if a message from round $k < i-1$ is needed in round $i$, it is incorporated in all of $P_j$'s state messages from $s_j^{k+1}$ to $s_j^{i-1}$.

Regarding the security of the MPC protocol, we consider the standard simulation-based notion. In the ideal world parties interact with the ideal functionality $\mathcal{F}_{\text{MPC}}$, described in Functionality 2.1. In the real world, parties engage in the real-world MPC protocol $\pi$ in

[4]Note that we will relax this requirement later, also allowing MPC protocols which use secure point-to-point channels. See Section 2.4 for details.

the presence of an adversary $A$, who is allowed to corrupt a set $I \subset [n]$ of parties and may follow an arbitrary polynomial-time strategy. Security of $\pi$ is defined as follows:

**Definition 2.** *A protocol $\pi$ is said to securely compute $F$ with abort if for every PPT adversary $A$ in the real world, there exists a PPT adversary $S$, such that for any set of corrupted parties $I \subset [n]$ with $|I| \leq t$ (where $t$ is the adversarial threshold), every initial input vector $(x_1, \ldots, x_n)$, and every security parameter $\lambda$, it holds that*

$$\{\texttt{IDEAL}_{f,S(z),I}(1^\lambda, (x_1, \ldots, x_n))\} =_c \{\texttt{REAL}_{\pi,A(z),I}(1^\lambda, (x_1, \ldots, x_n))\},$$

*where $z \in \{0,1\}^*$ is the auxiliary input, $\texttt{IDEAL}_{f,S(z),I}$ denotes the output of the interaction of the adversary $S(z)$ (who corrupts parties in $I$) with the ideal functionality (this output consists of the output of the adversary $S(z)$ as well as the outputs of the honest parties), and $\texttt{REAL}_{\pi,A(z),I}$ denotes the output of protocol $\pi$ given the adversary $A(z)$ who corrupts parties in $I$ (this output consists of the output of the adversary $A(z)$ as well as the outputs of the honest parties).*

Finally, in our constructions we additionally assume that the underlying protocol $\pi$ has the property that given the transcript of the protocol execution, the output can be publicly computed (as defined in [78]):

**Definition 3** (Publicly Recoverable Output)**.** *Given a transcript $\tau$ of an execution of a protocol $\pi$, there exists a function* Eval *such that the output of the protocol $\pi$ for all parties is given by $y = \mathsf{Eval}(\tau)$.*

**Functionality 2.1:** $\mathcal{F}_{\text{MPC}}$

1. Let the set of MPC participants be $\mathcal{P} = \{P_1, \ldots, P_n\}$.
2. Let $x_i$ denote the input of the party $P_i \in \mathcal{P}$.
3. The adversary $S$ selects a set $I \subset [n]$ of corrupted parties.
4. Each honest party $P_i$ sends its input $x_i^* = x_i$ to $\mathcal{F}_{\text{MPC}}$. For each corrupted party $P_j$, the adversary may select any value $x_j^*$ and send it to $\mathcal{F}_{\text{MPC}}$.
5. $\mathcal{F}_{\text{MPC}}$ computes $F(x_1^*, \ldots, x_n^*) = (y_1, \ldots, y_n)$ and sends $\{y_i\}_{i \in I}$ to the adversary.
6. The adversary sends either `abort` or `continue` to $\mathcal{F}_{\text{MPC}}$.
   - If the adversary sent `abort`, $\mathcal{F}_{\text{MPC}}$ sends $\perp$ to each honest party.
   - Otherwise, $\mathcal{F}_{\text{MPC}}$ sends $y_i$ to each honest party $P_i$.
7. Each honest party $P_i$ outputs the message it received from $\mathcal{F}_{\text{MPC}}$. Each adversarial party can output an arbitrary PPT function of the adversary's view.

In one of our constructions, we consider MPC protocols which provide *guaranteed output*

*delivery*. In that case the security of protocol $\pi$ is defined the same way as before, except that the ideal functionality is now $\mathcal{F}_{\text{MPC-GoD}}$, described in Functionality 2.2.

**Functionality 2.2:** $\mathcal{F}_{\text{MPC-GoD}}$

1. Let the set of MPC participants be $\mathcal{P} = \{P_1, \ldots, P_n\}$.
2. Let $x_i$ denote the input of the party $P_i \in \mathcal{P}$.
3. The adversary $S$ selects a set $I \subset [n]$ of corrupted parties.
4. Each honest party $P_i$ sends its input $x_i^* = x_i$ to $\mathcal{F}_{\text{MPC-GoD}}$. For each corrupted party $P_j$, the adversary may select any value $x_j^*$ and send it to $\mathcal{F}_{\text{MPC-GoD}}$.
5. $\mathcal{F}_{\text{MPC}}$ computes $F(x_1^*, \ldots, x_n^*) = (y_1, \ldots, y_n)$, substituting each missing value by some default value.
6. $\mathcal{F}_{\text{MPC-GoD}}$ sends $y_i$ to each party $P_i$.
7. Each honest party $P_i$ outputs the message it received from $\mathcal{F}_{\text{MPC-GoD}}$. Each adversarial party can output an arbitrary PPT function of the adversary's view.

### 2.2.2 Yao's Grabled Circuits

One of the core building blocks in our construction are Yao's garbled circuits that allow secure two-party computation [104, 106]. In the following, we provide definitions for the garbling process as well as the security of garbling scheme (taken verbatim from [41]):

**Definition 4** (Garbling scheme)**.** *A garbling scheme for circuits is a tuple of PPT algorithms* $\mathtt{GC} := (\mathtt{Gen}, \mathtt{Garble}, \mathtt{Eval})$ *such that:*

- $(\{\mathtt{lab}^{w,b}\}_{w\in\mathtt{inp},b\in\{0,1\}}) \leftarrow \mathtt{Gen}(1^\lambda, \mathtt{inp})$ : $\mathtt{Gen}$ *takes the security parameter* $1^\lambda$ *and length of input for the circuit as input and outputs a set of input labels* $\{\mathtt{lab}^{w,b}\}_{w\in\mathtt{inp},b\in\{0,1\}}$.
- $\bar{C} \leftarrow \mathtt{Garble}(C, (\{\mathtt{lab}^{w,b}\}_{w\in\mathtt{inp},b\in\{0,1\}})$ : $\mathtt{Garble}$ *takes as input a circuit* $C : \{0,1\}^{\mathtt{inp}} \rightarrow \{0,1\}^{\mathtt{out}}$ *and a set of input labels* $\{\mathtt{lab}^{w,b}\}_{w\in\mathtt{inp},b\in\{0,1\}}$ *and outputs the garbled circuit* $\bar{C}$.
- $y \leftarrow \mathtt{Eval}(\bar{C}, \mathtt{lab}^x)$ : $\mathtt{Eval}$ *takes as input the garbled circuit* $\bar{C}$*, input labels* $\mathtt{lab}^x$ *corresponding to the input* $x \in \{0,1\}^{\mathtt{inp}}$ *and outputs* $y \in \{0,1\}^{\mathtt{out}}$.

*The garbling scheme satisfies the following properties:*

1. ***Correctness:*** *For any circuit* $C$ *and input* $x \in \{0,1\}^{\mathtt{inp}}$,

$$Pr[C(x) = \mathtt{Eval}(\bar{C}, \mathtt{lab}^x)] = 1,$$

*where* $(\{\mathtt{lab}^{w,b}\}_{w\in\mathtt{inp},b\in\{0,1\}}) \leftarrow \mathtt{Gen}(1^\lambda, \mathtt{inp})$ *and* $\bar{C} \leftarrow \mathtt{Garble}(C, \{\mathtt{lab}^{w,b}\}_{w\in\mathtt{inp},b\in\{0,1\}})$.

2. ***Selective Security:*** *There exists a PPT simulator* $\mathtt{Sim}_{\mathtt{GC}}$ *such that, for any PPT adversary* $\mathcal{A}$*, there exists a negligible function* $\mu(\cdot)$ *such that*

$$|Pr[\mathtt{Experiment}_{\mathcal{A},\mathtt{Sim}_{\mathtt{GC}}}(1^\lambda, 0) = 1] - Pr[\mathtt{Experiment}_{\mathcal{A},\mathtt{Sim}_{\mathtt{GC}}}(1^\lambda, 1) = 1]| \leq \mu(\lambda)$$

*where the experiment* $\mathtt{Experiment}_{\mathcal{A},\mathtt{Sim}_{\mathtt{GC}}}(1^\lambda, b)$ *is defined as follows:*

(a) *The adversary* $\mathcal{A}$ *specifies the circuit* $C$ *and an input* $x \in \{0,1\}^{\mathtt{inp}}$ *and gets* $\bar{C}$ *and* $\hat{\mathtt{lab}}$*, which are computed as follows:*

- *If* $b = 0$*:*
  - $(\{\mathtt{lab}^{w,b}\}_{w\in\mathtt{inp},b\in\{0,1\}}) \leftarrow \mathtt{Gen}(1^\lambda, \mathtt{inp})$
  - $\bar{C} \leftarrow \mathtt{Garble}(C, (\{\mathtt{lab}^{w,b}\}_{w\in\mathtt{inp},b\in\{0,1\}})$
- *If* $b = 1$*:*
  - $(\bar{C}, \hat{\mathtt{lab}}) \leftarrow \mathtt{Sim}_{\mathtt{GC}}(1^\lambda, C(x))$
- *The adversary outputs a bit* $b'$*, which is the output of the experiment.*

We note that Yao's protocol achieves selective security. Very roughly, the security of the party producing the garbled circuit relies on the fact that for each wire of the circuit, only a single garbled key is revealed, and thus the only information the other party gets is the (garbled) output. We refer to the work of Lindell and Pinkas for the details of the construction as well as the security proof [87].

### 2.2.3 DPSS

A key building block of our CSaR instantiation is a dynamic proactive secret sharing scheme (DPSS), which allows a party to distribute shares of a secret to $n$ parties. The scheme ensures that an adversary in control of some threshold number of parties $t$ will learn no information about the secret. Over the course of running the protocol the set of parties holding the secret is constantly changing, and the adversary might "release" some parties (corresponding to users who regain control of their systems) and corrupt new ones.

A DPSS scheme consists of the following three phases.

**Setup.** In each setup phase, one or more independent clients secret-share a total of $m$ secrets to a set of $n$ parties, known as a committee, denoted by $\mathcal{C} = \{P_1, \ldots, P_n\}$. After each setup phase, each committee member holds one share for each secret $s$ distributed during this phase.

**Hand-off.** As the protocol runs, the hand-off phase is periodically invoked. In the hand-off phase, the sharing of each secret is passed from the old committee, $\mathcal{C}$, to a new

committee, $\mathcal{C}'$. This process reflects parties leaving and joining the committee. After the hand-off phase, all parties in the old committee delete their shares, and all parties in the new committee hold a sharing for each secret $s$. The hand-off phase is particularly challenging, since during the hand-off a total of $2t$ parties may be corrupted ($t$ parties in the old committee and $t$ parties in the new committee).

**Reconstruction.** When a client (which need not be one who participated in the setup phase) asks for the reconstruction of a specific secret, that client and all parties in the current committee engage in a reconstruction process to allow the client learn the secret.

#### 2.2.3.1 DPSS Security Definition

A DPSS scheme is required to satisfy two properties, *robustness* and *secrecy*. At a high-level, *robustness* requires that it should always be possible to recover the secret. *Secrecy* requires that an adversary should not learn any further information about the secret beyond what has been learned before running the protocol.

**Robustness.** For any PPT adversary $\mathcal{A}$ with corruption threshold $t$ it holds that after each setup phase, there exists a fixed secret $s^\star$ for each sharing distributed during this phase. If the setup phase was executed by an honest client, this secret is the same as the one chosen by this client. When at some point an honest client asks for a reconstruction of this secret, the client receives the correct secret $s^\star$.

**Secrecy.** For any PPT adversary $\mathcal{A}$ with corruption threshold $t$, there exists a simulator $\mathcal{S}$ with access to our security model $\mathsf{Ideal}_{\text{safe}}$ (described in Ideal Secrecy), such that the view of $\mathcal{A}$ interacting with $\mathcal{S}$ is computationally indistinguishable from the view in the real execution.

**Ideal Secrecy:** $\mathsf{Ideal}_{\text{safe}}$

1. $\mathsf{Ideal}_{\text{safe}}$ receives the secrets from the clients in the setup phase.
2. When an honest client asks for the reconstruction of some specific secret $s^\star$, $\mathsf{Ideal}_{\text{safe}}$ sends $s^\star$ to that client.

**Definition 5.** *A dynamic proactive secret-sharing scheme is secure if for any PPT adversary $\mathcal{A}$ and threshold $t$, it satisfies both* ***robustness*** *and* ***secrecy****.*

### 2.2.4 Non-interactive zero-knowledge proofs

Non-interactive zero-knowledge proofs (NIZKs) allow one party (the prover) prove validity of some statement to another party (the verifier), in a way that nothing except for the validity of the statement is revealed and no interaction between the prover and the verifier is needed. More formally, as defined by Groth [67]:

Let $R$ be an efficiently computable binary relation. For pairs $(x, w) \in R$ we call $x$ the statement and $w$ the witness.

A proof system for relation $R$ consists of a key generation algorithm $KeyGen$, a prover $P$ and a verifier $V$. The key generation algorithm produces a CRS $\sigma$. The prover takes as input $(\sigma, x, w)$ and produces a proof $\pi$. The verifier takes as input $(\sigma, x, \pi)$ and outputs 1 if the proof is acceptable and 0 otherwise.

**Definition 6** (Proof system)**.** *$(KeyGen, P, V)$ is a proof system for $R$ if it satisfies the following properties:*

- *Perfect completeness. For all adversaries $\mathcal{A}$ holds:*
$$Pr[\sigma \leftarrow KeyGen(1^k); (x, w) \leftarrow \mathcal{A}(\sigma); \pi \leftarrow P(\sigma, x, w) : V(\sigma, x, \pi) = 1 \text{ if } (x, w) \in R] = 1$$
- *Perfect soundness. For all adversaries $\mathcal{A}$ holds:*
$$Pr[\sigma \leftarrow KeyGen(1^k); (x, \pi) \leftarrow \mathcal{A}(\sigma) : V(\sigma, x, \pi) = 0 \text{ if } x \notin L] = 1$$

**Definition 7** (Proof of knowledge)**.** *We call a proof system $(KeyGen, P, V)$ a proof of knowledge for $R$ if there exists a polynomial time extractor $E = (E_1, E_2)$ such that for all adversaries $\mathcal{A}$ holds:*

$$Pr[\sigma \leftarrow KeyGen(1^k) : \mathcal{A}(\sigma) = 1] = Pr[(\sigma, \epsilon) \leftarrow E_1(1^k) : \mathcal{A}(\sigma) = 1], \text{ and}$$
$$Pr[(\sigma, \epsilon) \leftarrow E_1(1^k); (x, \pi) \leftarrow \mathcal{A}(\sigma); w \leftarrow E_2(\sigma, \epsilon, x, \pi) : V(\sigma, x, \pi) = 0 \text{ or } (x, w) \in R] = 1$$

**Definition 8** (Non-interactive zero-knowledge proof, NIZK)**.** *A non-interactive proof system $(KeyGen, P, V)$ is a NIZK for $R$ if there exists a polynomial time simulator $S = (S_1, S_2)$, which satisfies the following property:*

***(Unbounded) computational zero-knowledge.*** *For all PPT adversaries $\mathcal{A}$ holds*

$$Pr[\sigma \leftarrow KeyGen(1^k) : \mathcal{A}^{P(\sigma,\cdot,\cdot)}(\sigma) = 1] \approx Pr[(\sigma, \tau) \leftarrow S_1(1^k) : \mathcal{A}^{S(\sigma,\tau,\cdot,\cdot)}(\sigma) = 1],$$

*where $f(k) \approx g(k)$ means that there exists a negligible function $negl(k)$ s.t. $|f(k) - g(k)| < negl(k)$*

Finally, in our eWEB protocol we require that if after seeing some number of simulated proofs the adversary is able to produce a new valid proof, we are able to extract a witness

from this proof. This property is called simulation sound extractability [67]:

**(Unbounded) simulation sound extractability.** A NIZK proof of knowledge $(KeyGen, P,$
$V, S_1, S_2, E_1, E_2)$ is simulation sound extractable, if for all PPT adversaries $\mathcal{A}$ holds

$$Pr[(\sigma, \tau, \epsilon) \leftarrow SE_1(1^k); (x, \pi) \leftarrow \mathcal{A}^{S_2(\sigma,\tau,\cdot)}(\sigma, \epsilon); w \leftarrow E_2(\sigma, \epsilon, x, \pi) : (x, \pi) \notin Q \text{ and } (x, w) \notin R \text{ and } V(\sigma, x, \pi) = 1] \approx 0,$$

where $SE_1$ is an algorithm that outputs $(\sigma, \tau, \epsilon)$ such that it is identical to $S_1$ when restricted to the first two parts $(\sigma, \tau)$, and $Q$ is a list of simulation queries and responses (i.e., simulated proofs).

### 2.2.5 Hash functions

To reduce communication cost of our eWEB protocol, we use hash functions. In the following, we formally define a hash function family and the collision-resistance property that we rely on in our construction.

**Definition 9.** *Hash function family*

*A family of functions $H = \{h_i : D_i \to R_i\}_{i \in I}$ is a hash function family, if the following holds:*

***Easy to sample.*** *There exists a PPT algorithm Gen outputting $i$ s.t. $h_i$ is a random member of $H$.*

***Easy to evaluate.*** *There exists a PPT algorithm Eval s.t. for all $i \in I$ and $x \in D_i$, $Eval(i, x) = h_i(x)$.*

***Compression.*** *For all $i \in I$ holds $|R_i| < |D_i|$.*

We say a hash function family is a collision resistant hash function (CRHF) if the following condition holds:

**Collision-resistance.** Foll all PPT $\mathcal{A}$, security parameter $k$ and a negligible function $negl(\cdot)$ holds:

$$Pr[i \leftarrow Gen(k); (x, x') \leftarrow \mathcal{A}(i) : x \neq x' \text{ and } h_i(x) = h_i(x')] \leq negl(k)$$

### 2.2.6 Append-only Bulletin Boards

In our construction, we rely on public bulletin boards. Specifically, we require that the bulletin boards allows parties to publish arbitrary strings and receive a confirmation (dubbed *"proof of publish"*) that the string was published in return.

Following the approach of Kaptchuk et al [79], we assume that parties publish their strings as part of a public chain of values, and abstract the bulletin board syntax as follows:

- $(\texttt{post}, \sigma) \leftarrow \texttt{Post}(\texttt{M})$. Intuitively, when a party wishes to post some data $M$ on the public chain, the `Post` function is called. This call results in `post` (which consists of $M$, as well as additional data which identifies this data record on the chain) being appended to the chain. The tuple $(\texttt{post}, \sigma)$, where $\sigma$ is the proof of publish, is returned. We assume that a proof of publish is public and can be retrieved for already published posts as well.
- $\{0, 1\} \leftarrow \texttt{Verify}(\texttt{post}, \sigma)$. The public verification algorithm takes as input a supposedly published record `post` as well as a proof of publish $\sigma$, and verifies that the record `post` has indeed been published.

Security-wise, we require that the contents of the bulletin board are hard to erase or modify and that the proof of publish is unforgeable. Specifically, we require that up to a negligible probability it is impossible to come up with a pair $(\texttt{post}, \sigma)$ such that $\texttt{Verify}(\texttt{post}, \sigma) = 1$, unless this pair has been generated through a call to the `Post` algorithm. This property holds even if the adversary is given an oracle that posts arbitrary strings on the bulletin board on the behalf of the adversary.

Such bulletin boards have been extensively investigated in prior works [43, 61, 79]. While specific syntax details of the bulletin board abstraction slightly vary throughout these works, they all ensure that parties are able to post arbitrary strings on an append-only log, and the proof of publish cannot be forged. These works also point out that bulletin boards with the properties described above already exist in practice. They can be realized from centralized systems such as the Certificate Transparency project [1], and from the decentralized systems such as proof-of-work or proof-of-stake blockchains.

### 2.2.7 MPC in the Presence of Contributors and Evaluators

In the following, we formally define the security of the functionality which we want to achieve. Recall that we consider two sets of parties – MPC contributors who supply inputs

and MPC evaluators who wish to obtain the output.

We consider the simulation-based notion of security. In the ideal world, parties interact with the ideal functionality $\mathcal{F}_{\text{eval-MPC}}$, described in Figure 2.3. Note that the difference to the standard ideal functionality for MPC with abort (described in Figure 2.1) is that we distinguish between contributors and evaluators.

In the real world, parties execute the protocol $\pi$ in the presence of an adversary $A$. The adversary $A$ is allowed to corrupt a set of contributors $I \subset [n]$ as well as a set of evaluators $I' \subset [n']$. $A$ is allowed to send messages in place of corrupted parties and can follow an arbitrary polynomial-time strategy.

Security of $\pi$ is defined as follows:

**Definition 10.** *A protocol $\pi$ is said to securely compute $F$ with abort in the presence of contributors and evaluators if for every PPT adversary $A$ in the real world, there exists a PPT adversary $S$, such that for any set of corrupted evaluators $I' \subset [n']$, any set of contributors $I \subset [n]$ with $|I| \leq t$ (where $t$ is the adversarial threshold), every initial input vector $(x_1, \ldots, x_n)$, and every security parameter $\lambda$, it holds that*

$$\{\texttt{IDEAL}_{f,S(z),I}(1^\lambda, (x_1, \ldots, x_n))\} =_c \{\texttt{REAL}_{\pi,A(z),I}(1^\lambda, (x_1, \ldots, x_n))\},$$

*where $z \in \{0,1\}^*$ is the auxiliary input, $\texttt{IDEAL}_{f,S(z),I}$ denotes the output of the interaction of the adversary $S(z)$ (who corrupts parties in $I$) with the ideal functionality $\mathcal{F}_{\text{eval-MPC}}$ (this output consists of the output of the adversary $S(z)$ as well as the outputs of the honest parties), and $\texttt{REAL}_{\pi,A(z),I}$ denotes the output of the interaction between the adversary $A(z)$ who corrupts parties in $I$ and the honest parties in the protocol $\pi$ (this output consists of the output of the adversary $A(z)$ as well as the outputs of the honest parties).*

In one of our constructions, we consider MPC protocols which provide *guaranteed output delivery*. In that case the security of protocol $\pi$ is defined the same way as before, except that the ideal functionality is now $\mathcal{F}_{\text{eval-MPC-GoD}}$, described in Functionality 2.4.

### 2.2.8 Multi-Key FHE with Distributed Setup

Our construction of an MPC scheme which combined communication and state complexity is independent of the function being computed is based on the MPC protocol of Brakerski et al. [29], which in turn utilizes multi-key fully homomorphic encryption scheme with distributed setup. In the following, we formally define this primitive (in large parts taken verbatim from Brakerski et al. [29]).

**Functionality 2.3:** $\mathcal{F}_{\text{eval-MPC}}$

1. We distinguish between the set of MPC contributors $\mathcal{P} = \{P_1, \ldots, P_n\}$ and the set of evaluators $\mathcal{E} = \{E_1, \ldots, E_{n'}\}$. These sets can be, but do not need to be disjoint.
2. Let $x_i$ denote the input of the party $P_i \in \mathcal{P}$.
3. The adversary $S$ selects a set of contributors $I \subset [n]$ to corrupt.
4. The adversary $S$ selects a set of evaluators $I' \subset [n']$ to corrupt.
5. Each honest party $P_i$ sends its input $x_i^* = x_i$ to $\mathcal{F}_{\text{eval-MPC}}$. For each corrupted party $P_j$, the adversary may select any value $x_j^*$ and send it to $\mathcal{F}_{\text{eval-MPC}}$.
6. $\mathcal{F}_{\text{eval-MPC}}$ computes $F(x_1^*, \ldots x_n^*)$ and sends $F(x_1^*, \ldots x_n^*)$ to the adversary.
7. The adversary sends either `abort` or `continue` to $\mathcal{F}_{\text{eval-MPC}}$.
   - If the adversary send `abort`, $\mathcal{F}_{\text{eval-MPC}}$ sends $\perp$ to all honest evaluators.
   - Otherwise, $\mathcal{F}_{\text{eval-MPC}}$ sends $F(x_1^*, \ldots x_n^*)$ to each honest evaluator.
8. Each honest evaluator outputs the message it received from $\mathcal{F}_{\text{eval-MPC}}$. Each adversarial party can output an arbitrary PPT function of the adversary's view.

**Functionality 2.4:** $\mathcal{F}_{\text{eval-MPC-GoD}}$

1. We distinguish between the set of MPC contributors $\mathcal{P} = \{P_1, \ldots, P_n\}$ and the set of evaluators $\mathcal{E} = \{E_1, \ldots, E_{n'}\}$. These sets can be, but do not need to be disjoint.
2. Let $x_i$ denote the input of the party $P_i \in \mathcal{P}$.
3. The adversary $S$ selects a set of contributors $I \subset [n]$ to corrupt.
4. The adversary $S$ selects a set of evaluators $I' \subset [n']$ to corrupt.
5. Each honest party $P_i$ sends its input $x_i^* = x_i$ to $\mathcal{F}_{\text{eval-MPC}}$. For each corrupted party $P_j$, the adversary may select any value $x_j^*$ and send it to $\mathcal{F}_{\text{eval-MPC}}$.
6. $\mathcal{F}_{\text{eval-MPC}}$ computes $F(x_1^*, \ldots x_n^*)$ and sends it to the adversary as well as each honest evaluator.
7. Each honest evaluator outputs the message it received from $\mathcal{F}_{\text{eval-MPC-GoD}}$. Each adversarial party can output an arbitrary PPT function of the adversary's view.

**Definition 11** (Multi-key fully homomorphic encryption scheme, MFHE)**.** *An MFHE scheme with distributed setup consists of the procedures* (MFHE.DistSetup, MFHE.Keygen, MFHE.Encrypt, MFHE.Decrypt, MFHE.Eval)*, defined as follows:*

- $\mathsf{params}_i \leftarrow \mathsf{MFHE.DistSetup}(1^\kappa, 1^N, i)$: On input the security parameter $\kappa$ and number of users $N$, outputs the system parameters for the $i$-th player $\mathsf{params}_i$. Let $\mathsf{params} = \{\mathsf{params}_i\}_{i\in[N]}$.
- $(\mathsf{pk}, \mathsf{sk}) \leftarrow \mathsf{MFHE.Keygen}(\mathsf{params}, \mathsf{i})$: On input $\mathsf{params}$ and entry number $i$ the key generation algorithm outputs a public/secret key pair $(\mathsf{pk}, \mathsf{sk})$.
- $c \leftarrow \mathsf{MFHE.Encrypt}(\mathsf{pk}, x)$: On input $pk$ and a plaintext message $x \in \{0,1\}^*$ output a "fresh ciphertext" $c$. (We assume for convenience that the ciphertext includes also the respective public key.)
- $\hat{c} := \mathsf{MFHE.Eval}(\mathsf{params}; \mathcal{C}; (c_1, \ldots, c_l))$: On input a (description of a) Boolean circuit $\mathcal{C}$ and a sequence of fresh ciphertexts $(c_1, \ldots, c_l)$, output an "evaluated ciphertext" $\hat{c}$. (Here we assume that the evaluated ciphertext includes also all the public keys from the $c_i$'s.)
- $x := \mathsf{MFHE.Decrypt}((\mathsf{sk}_1, \ldots, \mathsf{sk}_N), \hat{\mathsf{c}})$: On input an evaluated ciphertext $\hat{c}$ (with $N$ public keys) and the corresponding $N$ secret keys $(\mathsf{sk}_1, \ldots, \mathsf{sk}_N)$, output the message $x \in \{0,1\}^*$.

The scheme is correct if for every circuit $\mathcal{C}$ on $N$ inputs and any input sequence $x_1, \ldots, x_N$ for $\mathcal{C}$, we set $\mathsf{params}_i \leftarrow \mathsf{MFHE.DistSetup}(1^\kappa, 1^N, i)$, $\mathsf{params} = \{\mathsf{params}_i\}_{i\in[N]}$, and then generate $N$ key-pairs and $N$ ciphertexts $(pk_i, sk_i) \leftarrow \mathsf{MFHE.Keygen}(\mathsf{params})$ and $c_i \leftarrow \mathsf{MFHE.Encrypt}(\mathsf{pk_i}, x_i)$, then we get

$$\mathsf{MFHE.Decrypt}((\mathsf{sk}_1, \ldots, \mathsf{sk}_N), \mathsf{MFHE.Eval}(\mathsf{params}; \mathcal{C}; (c_1, \ldots, c_N))) = \mathcal{C}(x_1, \ldots, x_N)$$

except with negligible probability (in $\kappa$) taken over the randomness of all these algorithms.

In the work of Brakerski et al. the following two properties are needed of the multi-key FHE schemes: first, the decryption procedure consists of a "local" partial-decryption procedure $ev_i \leftarrow \mathsf{MFHE.PartDec}(\hat{c}, \mathsf{sk_i})$ that only takes one of the secret keys and outputs a partial decryption share, and a public combination procedure that takes these partial shares and outputs the plaintext, $x \leftarrow \mathsf{MFHE.FinDec}(ev_1, \ldots, ev_N, \hat{c})$. Another property that is needed is the ability to simulate the decryption shares. Specifically, there exists a PPT simulator $S^T$, that gets for input:

- the evaluated ciphertext $\hat{c}$,
- the output plaintext $x := \mathsf{MFHE.Decrypt}((\mathsf{sk}_1, \leftarrow, \mathsf{sk}_N), \hat{c})$,
- a subset $I \subset [N]$, and all secret keys except the one for $I$, $\{\mathsf{sk_j}\}_{j\in[N]\setminus I}$.

The simulator produces as output simulated partial evaluation decryption shares: $\{\tilde{ev}_i\}_{i\in I} \leftarrow S^T(x, \hat{c}, I, \{\mathsf{sk_j}\}_{j\in[N]\setminus I})$. We want the simulated shares to be statistically close to the shares produced by the local partial decryption procedures using the keys $\{sk_i\}_{i\in I}$, even conditioned

on all the inputs of $S^T$. A scheme is simulatable if it has local decryption and a simulator as described here.

Brakerski et al. require that semantic security for the $i$-th party holds even when all $\{\mathsf{params}_j\}_{j\in[N]\setminus i}$ are generated adversarially and possibly depending on $\mathsf{params}_i$.

They consider a rushing adversary that chooses $N$ and $i \in [N]$, then it sees $\mathsf{params}_i$ and produces $\mathsf{params}_j$ for all $j \in [N] \setminus \{i\}$. After this setup, the adversary is engaged in the usual semantic-security game, where it is given the public key, chooses two messages and is given the encryption of one of them, and it needs to guess which one was encrypted.

Simulation of the decryption shares is defined as before, but now the evaluated ciphertext is produced by the honest party interacting with the same rushing adversary (and statistical closeness holds even conditioned on everything that the adversary sees).

## 2.3 CSaRs

In our work, we rely on what we call *conditional storage and retrieval systems* (*CSaR*s) that allow for a secure storage- and retrieval of secrets. In this section, we formalize the CSaR primitive and discuss its instantiation.

### 2.3.1 Defining CSaRs

Informally, CSaRs allow a user to store a secret with a CSaR along with a release condition. The user's secret is released if and only if this condition is satisfied. While such systems could be realized via a trusted third party, they can also be obtained using a set of parties with the guarantee that some sufficiently large subset of these parties is honest. A user can then distribute its secret between the set of parties, and the CSaR's security guarantee ensures that no subset of parties that is smaller than a defined threshold can use its secret shares to gain information about the secret. We capture this informal notion by giving a formal ideal functionality in Figure 2.5.

Based on the ideal functionality, we define the security of a CSaR scheme as follows:

**CSaR Security.** For any PPT adversary $\mathcal{A}$ there exists a PPT simulator $\mathcal{S}$ with access to our security model $\mathsf{Ideal}_{\mathrm{CSaR}}$ (described in Ideal CSaR), such that the view of $\mathcal{A}$ interacting with $\mathcal{S}$ is computationally indistinguishable from the view in the real execution.

**Figure 2.5: Ideal CSaR:** $\mathsf{Ideal}_{\mathrm{CSaR}}$

1. **SecretStore** Upon receiving an (identifier, release condition, secret) tuple $\tau = (id, F, s)$ from a client $P$, $\mathsf{Ideal}_{\mathrm{CSaR}}$ checks whether $id$ was already used. If not, $\mathsf{Ideal}_{\mathrm{CSaR}}$ stores $\tau$ and notifies all participants that a valid storage request with the identifier $id$ and the release condition $F$ has been received from a client $P$. Here, the release condition is an NP statement.
2. **SecretRelease** Upon receiving an (identifier, witness) tuple $(id, w)$ from some client $C$, $\mathsf{Ideal}_{\mathrm{CSaR}}$ checks whether there exists a record with the identifier $id$. If so, $\mathsf{Ideal}_{\mathrm{CSaR}}$ checks whether $F(w) = true$, where $F$ is the release condition corresponding to the secret with the identifier $id$. If so, $\mathsf{Ideal}_{\mathrm{CSaR}}$ sends the corresponding secret $s$ to client $C$.

### 2.3.2 Instantiating CSaRs

We now provide an implementation of a CSaR system which we dub eWEB. [5] Its key building block is a DPSS scheme used in a black-box way. The initial committee are miners who mined the most recent $n$ blocks in the underlying blockchains.

Given a secret message $M$ and a release condition $F$, the depositor stores the release condition $F$ on the blockchain and secret-shares $M$ among the miners using the secret storage (setup) algorithm of the DPSS scheme.

During the protocol's periodically executed hand-off phase, the secrets are passed from the miners of the old committee to the miners of the new committee using the DPSS hand-off algorithm. The new committee consists of the miners who mined the most recent $n$ blocks. This keeps the size of the committee constant and allows all parties to determine the current committee by looking at the blockchain state. It is possible that some committee members receive more information about the secrets than others — roughly, if a party mined $m$ out of the last $n$ blocks, this party receives $\frac{m}{n}$ of all the shares. This reflects the distribution of the computing power (for POW blockchains) or stake (for POS blockchains) in the system [61].

To retrieve a stored secret, a requester $U$ needs to prove that they are eligible to do so.

[5] Note that another viable candidate for a CSaR instantiation is the system proposed by Benhamouda et al. [21]. While it does not formally explain how the secrets can be stored to and retrieved from the blockchain given a specific release condition, it should be possible to extend their system by the techniques we use in eWEB.

This poses a challenge. An insecure solution is to just send a valid witness $w$ ($F(w) = true$) to the miners. One obvious problem with this solution is that a malicious miner can use the provided witness to construct a new secret release request and retrieve the secret himself. To solve this problem, instead of sending the witness in clear, the user *proves that they know a valid witness*. Thus, while the committee members are able to check the validity of the request and privately release the secret to $U$, the witness remains hidden. In our scheme we rely on non-interactive zero knowledge proofs (NIZKs) [26]. Such proofs allow one party (the prover) to prove validity of some statement to another party (the verifier), such that nothing except for the validity of the statement is revealed. In eWEB we specifically use simulation extractable non-interactive zero knowledge *proofs of knowledge*, which allow the prover convince the verifier that they know a witness to some statement. Note that extractability can be added to any NIZK [3, 84]. We use NIZKs for relation $R = \{(pk, w) \mid F(w) = true \text{ and } pk = pk\}$, where $F(\cdot)$ is the release condition specified by the depositor and $pk$ is the public key of user $U$ and is used to identify the user eligible to receive the secret. After the miners verify the validity of the request, they engage in the DPSS's secret reconstruction with requester $U$ to release the secret to $U$.

We provide the full secret storage protocol in Figure 3. The hand-off protocol is given in Figure 4. The secret release protocol is in Figure 5. Note that the asymptotics of eWEB match those of the underlying DPSS scheme. Below, we elaborate on additional details of our construction.

#### 2.3.2.1 Subtleties of Point-to-Point Channels

Instead of assuming secure point-to-point channels, we rely on authenticated encryption and Protocols 1 and 2, executed whenever a message needs to be securely sent from one party to another. It is used for all messages exchanged in eWEB, including the underlying DPSS protocol. Whenever a party receives an encrypted message, it performs an authentication check to ensure that a ciphertext received from some party was generated by that party. This prevents the following malleability issue - a malicious user desiring to learn a secret with the identifier $id$ could generate a new secret storage request with a function $\tilde{F}$ for which he knows a witness, copy the DPSS messages sent by the user who created the storage request $id$ to the miners and later prove his knowledge of a witness for $\tilde{F}$ to release the corresponding secret. Without the authentication check, our scheme would be insecure, and our security proof would not go through.

**Protocol 1** MessagePreparation

1. For a message $m$ to be sent by party $P_s$ to party $P_r$, $P_s$ computes the ciphertext $c \leftarrow Enc_{pk_r}(m|pid_s)$, where $pk_r$ is the public key of $P_r$ and $pid_s$ is the party identifier of $P_s$.
2. $P_s$ prepends the storage identifier $id$ of his request and sends the tuple $(id, c)$ to $P_r$.

**Protocol 2** AuthenticatedDecryption

1. Upon receiving a tuple $(id, c)$ from party $P_s$ over an authenticated channel, the receiving party $P_r$ decrypts $c$ using its secret key $sk$ to obtain $m \leftarrow Dec_{sk}(c)$.
2. $P_r$ verifies that $m$ is of the form $m'|pid_s$ for some message $m'$, where $pid_s$ is the identifier of party $P_s$.
3. If the verification check fails, $P_r$ stops processing $c$ and outputs an error message.

#### 2.3.2.2 Storage Identifiers

Each storage request has a unique identifier $id$. This can be, e.g., the address of this particular transaction in the blockchain. It is used for practical reasons, and is not relevant for the security of our construction.

#### 2.3.2.3 Handling Large Secrets

Since the secret itself might be very large, it is also possible to first encrypt the secret using a symmetric encryption scheme, store the ciphertext publicly off chain and then secret-share the symmetric key instead. Also, we store request parameters (such as release conditions or proofs) off-chain, saving only the hash of the message on-chain.

**Proving security.** In the following, we briefly outline why the eWEB system satisfies the CSaR security notion. We sketch out the simulator $S$ that has access to the parties' secrets only via the ideal CSaR functionality and has the property that no PPT adversary can distinguish between interaction with the simulator and the interaction with the honest parties. $S$ in particular relies on the DPSS simulator $S_{DPSS}$ while simulating $\mathsf{Ideal}_{\text{safe}}$ for $S_{DPSS}$ and the NIZK simulator given by the zero-knowledge property of the NIZK scheme.

Before we describe the simulation of each eWEB subprotocol, we note that in the following $S$ internally stores all information published on the blockchain and aborts if whenever during the execution it notices that the information it retrieved from the blockchain is inconsistent with the internal copy.

Similarly, whenever according to the protocol $S$ is required to store some data off-chain

**Protocol 3** SecretStore

1. The depositor executes NIZK's $\mathsf{KeyGen}$ protocol to obtain a CRS: $\sigma \leftarrow \mathsf{KeyGen}(1^k)$.
2. The depositor computes hash $\mathsf{requestHash} \leftarrow H(F|\sigma)$, and publishes $\mathsf{requestHash}$ on the blockchain. Let $id$ be the storage identifier of the published request.
3. The depositor stores the tuple $(id, F|\sigma)$ offchain.
4. The depositor and the current members of the miner committee engage in the DPSS **Setup Phase**.
5. Each committee member retrieves $\mathsf{requestHash}$ from the blockchain, $F|\sigma$ from the offchain storage, and verifies that $\mathsf{requestHash}$ is indeed the hash of $F|\sigma$:

$$\mathsf{requestHash} \stackrel{?}{=} H(F|\sigma)$$

   If this is not the case, the committee member aborts.
6. $C_i$ stores $(id, \mathsf{dpss\text{-}data}_i)$ internally, where $\mathsf{dpss\text{-}data}_i$ is the data obtained from the DPSS **Setup Phase**.

**Protocol 4** SecretsHandoff

1. For each secret storage identifier $id$, the miners of the old and the new committee engage in the DPSS **Handoff Phase** for the corresponding secret. Let $\mathsf{dpss\text{-}data}_i^{id}$ denote the resulting DPSS data corresponding to the storage identifier $id$ of party $C_i$ of the new committee after the handoff phase.
2. For each secret storage identifier $id$, each miner of the new committee stores $(id, \mathsf{dpss\text{-}data}_i^{id})$ internally.

and the hash of it on-chain, $S$ additionally stores the data internally. Whenever according to the protocol $S$ is required to verify the correctness of this off-chain data by comparing its hash to the on-chain hash, $S$ instead directly compares the data to the one stored internally.

Additionally, whenever $S$ needs to send a message from an honest party to an honest party, it sends an encryption of a zero string of the according length instead.

**Simulating SecretStore.** We distinguish between the following cases:

- The client storing the secret is honest.
- The client storing the secret is malicious.

In the first case, instead of generating NIZK's CRS honestly, $S$ uses NIZK's simulator to generate the CRS $\sigma$. Then, $S$ generates the hash of the request, publishes it on the blockchain and stores the request data offchain as specified by the eWEB protocol. Additionally, $S$ stores the request data internally. Then, $S$ (acting as $\mathsf{Ideal}_{\text{safe}}$) notifies the

**Protocol 5** SecretRelease

1. To request the release of a secret with identifier $id$, the requester retrieves $\mathsf{requestHash}$ from the blockchain, $F|\sigma$ from off-chain storage, and verifies that $\mathsf{requestHash}$ is indeed the hash of $F|\sigma$:
$$\mathsf{requestHash} \stackrel{?}{=} H(F|\sigma)$$
If this is not the case, the requester aborts.
2. The requester computes a NIZK proof of knowledge of the witness for $F$ and his identifier $p_{id}$:
$$\pi \leftarrow P(\sigma, p_{id}, w),$$
3. The requester computes hash of the storage identifier, his identifier and the proof to obtain $\mathsf{requestHash}^* \leftarrow H(id|p_{id}|\pi)$ and publishes $\mathsf{requestHash}^*$ on blockchain. Let $id^*$ be the identifier of the published request.
4. The requester stores $(id^*, id|p_{id}|\pi)$ offchain.
5. Each committee member retrieves $\mathsf{requestHash}^*$ from the blockchain request with the identifier $id^*$, $id|p_{id}|\pi$ from the offchain storage, and verifies that:
$$\mathsf{requestHash}^* \stackrel{?}{=} H(id|p_{id}|\pi)$$
If not, the committee member aborts.
6. Each committee member retrieves $\mathsf{requestHash}$ from the blockchain request with the identifier $id$, $F|\sigma$ from the offchain storage, and verifies that:
$$\mathsf{requestHash} \stackrel{?}{=} H(F|\sigma)$$
If not, the committee member aborts.
7. Each committee member $C_i$ retrieves its share of the secret, $\mathsf{dpss\text{-}data}_i$, from its internal storage.
8. Each committee member $C_i$ checks if $\pi$ is a valid proof using the NIZK's verification algorithm $V$:
$$V(\sigma, p_{id}, \pi) \stackrel{?}{=} true$$
If so, $C_i$ and party $p_{id}$ engage in the DPSS **Reconstruction** using $\mathsf{dpss\text{-}data}_i$.

DPSS simulator of an honest secret storage request to simulate the DPSS setup phase (stopping the execution for a committee party whenever the request verification check did not go through).

In the second case, $S$ follows the eWEB protocol to verify the hashes and stores the

obtained data internally. Additionally, $S$ uses the DPSS simulator and passes messages between the adversary and the DPSS simulator (for those parties whose hash verification did not fail) and if the DPSS simulator subsequently extracts the secret and stores it in $\mathsf{Ideal}_{\text{safe}}$, $S$ stores it internally and in $\mathsf{Ideal}_{\text{CSaR}}$ (with the given release condition).

**Simulating SecretsHandoff.** $S$ uses the DPSS simulator to simulate the handoff phase.

**Simulating SecretRelease.** We again distinguish between the following cases:

- The client requesting the secret is honest.
- The client requesting the secret is malicious.

In the first case, $S$ follows the protocol for the request hash verification and if the offchain data is successfully verified $S$ uses the NIZK simulator to generate the required proof of knowledge. Then, $S$ continues to follow the eWEB protocol to generate the hash of the secret release request, publish it on the blockchain and store the required information offchain. For each honest committee member, $S$ continues to follow the protocol to retrieve the secret release request hash and verify the offchain data. Then, $S$ uses the DPSS simulator for the secret reconstruction process.

In the second case, we additionally distinguish between the following cases:

- The client who stored the secret is honest.
- The client who stored the secret is malicious.

In the first case, $S$ follows the protocol up to the step when it must engage with the client in the DPSS reconstruction phase. If the requester passed all verification checks, $S$ uses the witness extractor on the submitted proof and use the retrieved witness to retrieve the secret from the $\mathsf{Ideal}_{\text{CSaR}}$ and store it in $\mathsf{Ideal}_{\text{safe}}$ and uses the DPSS simulator for the last step.

In the second case, $S$ simply uses the DPSS simulator while accessing internally stored secret if necessary (acting as $\mathsf{Ideal}_{\text{safe}}$).

We now outline why no PPT adversary $\mathcal{A}$ is able to distinguish the view of interaction with the simulator $S$ constructed above from the view of interacting with honest parties in the real world. For this, we establish a series of hybrids such that any two consecutive hybrids are indistinguishable.

**$\text{Hybrid}_0$:** This hybrid corresponds to the execution in the real world. The simulator $S$ controls all honest parties and follows the protocol.

**$\text{Hybrid}_1$:** In this hybrid, $S$ switches from honestly generating the CRS and the proofs

to using the NIZK's simulator. By the unbounded zero-knowledge property of the NIZK, the adversary can detect the difference at most with a negligible probability.

**Hybrid$_2$:** In this hybrid, $S$ switches from generating the CRS using the NIZK's simulator given by the unbounded zero-knowledge property to generating the CRS given by the simulation sound extractability property of the NIZK. Again, the adversary can detect the difference at most with a negligible probability due to the simulation sound extractability property.

**Hybrid$_3$:** In this hybrid, $S$ internally stores all messages published on the blockchain and aborts whenever a message it retrieved from the blockchain during the execution at a later point is not consistent with the internal copy. Since we assume that is hard to modify or erase posts on the blockchain, $S$ aborts only with a negligible probability.

**Hybrid$_4$:** In this hybrid, $S$ additionally internally stores the information that is normally being hashed with the hash being published on chain and data being stored off chain. $S$ aborts whenever during the execution the information stored offchain is not consistent with $S$'s internal copy, but still passes the hash verification check. Since we assume that the hash function is collision-resistant, $S$ aborts only with a negligible probability.

**Hybrid$_5$:** In this hybrid, all encrypted messages sent between honest parties are changed to encryptions of zero strings of the same length. Due to the multi-message IND-CCA security of the encryption scheme, the adversary can detect the difference at most with a negligible probability.

**Hybrid$_6$:** In this hybrid, $S$ switches to using the DPSS simulator while honestly simulating $\mathsf{Ideal}_{\text{safe}}$ for it by keeping a list $L$ of secrets and adding secrets to this list whenever honest parties wish to store a secret or whenever the DPSS simulator wishes to store a secret. Here, $S$ simulates the point-to-point channels of the DPSS protocol in the same way as outlined in the proof given in the eWEB paper. By the security of the DPSS scheme, the adversary notices the difference with at most a negligible probability.

**Hybrid$_7$:** In this hybrid, $S$ changes the time when honest secrets are stored in $L$ – instead of storing them when the honest user wishes to store a secret, $S$ stores them only when a party wishes to see the secret and is able to provide a valid release request. Note that the only case when the secrets in $L$ are accessed by $S$ is when a client requests a reconstruction and is able to satisfy the release condition. Thus, nothing changes in this case.

**Hybrid$_8$:** In this hybrid, $S$ switches to using $\mathsf{Ideal}_{\text{CSaR}}$ to retrieve honest users' secrets.

Whenever the DPSS simulator wishes to retrieve an honest secret from $\mathsf{Ideal}_{\text{safe}}$, $S$ uses the proof of knowledge property of the NIZK to extract a witness from the adversarial proof. Then, $S$ sends the witness to $\mathsf{Ideal}_{\text{CSaR}}$ and (if the witness is correct) stores the obtained secret in $L$ for the DPSS simulator to use. By the unbounded simulation soundness property of the NIZK, the extracted witness satisfies the release condition. Thus, the adversary is able to detect a difference at most with a negligible probability.

### 2.3.3 CSaR-PR

For one of our constructions we rely on a CSaR variation which releases the secrets not privately to a single user, but publicly to everyone. We call this variation CSaR with public release (*CSaR-PR*), and introduce the ideal functionality in Figure 2.6.

**Figure 2.6: Ideal CSaR-PR:** $\mathsf{Ideal}_{\text{CSaR-PR}}$

1. **SecretStore** Upon receiving an (identifier, release condition, secret) tuple $\tau = (id, F, s)$ from a client $P$, $\mathsf{Ideal}_{\text{CSaR-PR}}$ checks whether $id$ was already used. If not, $\mathsf{Ideal}_{\text{CSaR-PR}}$ stores $\tau$ and notifies all participants that a valid storage request with the identifier $id$ and the release condition $F$ has been received from a client $P$.
2. **SecretRelease** Upon receiving an (identifier, witness) tuple $(id, w)$ from some client $C$, $\mathsf{Ideal}_{\text{CSaR-PR}}$ checks whether there exists a record with the identifier $id$. If so, $\mathsf{Ideal}_{\text{CSaR-PR}}$ checks whether $F(w) = true$, where $F$ is the release condition corresponding to the secret with the identifier $id$. If so, $\mathsf{Ideal}_{\text{CSaR-PR}}$ broadcasts the secret.

**CSaR-PR Security.** For any PPT adversary $\mathcal{A}$ there exists a PPT simulator $\mathcal{S}$ with access to our security model $\mathsf{Ideal}_{\text{CSaR-PR}}$ (described in Ideal CSaR-PR), such that the view of $\mathcal{A}$ interacting with $\mathcal{S}$ is computationally indistinguishable from the view in the real execution.

**CSaR-PR Instantiation.** CSaR-PR can be instantiated with a primitive that is very similar to our eWEB protocol, except that the user requesting the release of the secret provides the witness for the secret release condition publicly, allowing everyone to reconstruct the secret.

In more detail, to achieve public secret release, we make the following changes to the eWEB scheme:

- Instead of using NIZK proofs, the witness is submitted in clear.
- Instead of engaging in the DPSS **Reconstruction phase** with the user who provided a valid witness, miners post the corresponding reconstruction data off-chain and the signed hash of this data on-chain.
- Users retrieve the hashes from blockchain and verify the signatures. Using the retrieved information, users get the data from the off-chain storage, check the validity using the verified hashes and execute the DPSS **Reconstruction phase** to reconstruct the secret.

The security of this instantiation can be proven analogously to that of the original CSaR and eWEB constructions.

## 2.4 Our Non-Interactive MPC Construction

We now present our first construction - given an MPC protocol $\pi$, we use Yao's garbled circuits as well as a CSaR to transform it into an MPC protocol $\pi'$ that does not require parties to be online at the same time and only requires a single message from the contributors in $\pi$. The contributors in $\pi$ do not need to interact with each other. First, we briefly outline the assumptions we make and define the adversarial model.

**Assumptions.** We assume a public-key infrastructure and the existence of a CSaR. To distinguish between concurrent executions of the protocol, we give each computation a unique identifier $id$, and we assume that the evaluators know the public keys of the parties eligible to contribute in the protocol $\pi$. We assume the existence of a bulletin board modeled as an append-only log that provides a *proof of publish* which cannot be (efficiently) forged. Finally,we assume IND-CCA secure public key encryption.

For the ease of presentation, we assume the following about the MPC protocol $\pi$: (a) it is in a broadcast model, and (b) it has a single output which is made public to all participants in the last round [6].

**Adversary model.** We consider a computationally bounded, fully malicious, static adversary $\mathcal{A}$. Once an adversary corrupts a party it remains corrupted: the adversary is not allowed to adaptively corrupt previously honest parties.

[6]Note that these are not real limitations: if a protocol has several outputs, some of which cannot be made public, the MPC functionality broadcasts the encryption of a party's output under that party's public key. Additionally, later in this section we discuss how protocols with point-to-point channels can be supported in the broadcast model.

### 2.4.1 Construction Overview

Intuitively, there are two main steps in the protocol. In the first step, the parties (dubbed "contributors") prepare the garbled circuits (and keys) and store these with the CSaR. In the second step, one or more parties (we dub them "evaluators") use the garbled circuits to execute the original protocol $\pi$.

**Step 1. Preparing Garbled Circuits and Keys.** Each party $P_j$ that wishes to participate (contribute inputs) in $\pi$ starts by garbling the slightly modified next-message functions of each round of $\pi$. Typically, the next-message function takes as input some subset of the following: the secret input of the party, local randomness of the party for that particular round, the messages received in the previous rounds, some secret state passed along from the previous round. The output consists of the message that is broadcast as well as the state that is passed to the next round. We make the following modifications: in each round $i$, instead of the state $s_j^i$ that is passed to the next round, the function outputs the encryption $c_j^i$ of the state as well as a signature $sigpr_j^i$ over this encryption. Additionally, the modified next-message function outputs the public message $m_j^i$ that is supposed to be broadcast by $P_j$ in this round, as well as the signature $sigpub_j^i$ over this message. The secret key as well as the signature key of $P_j$ are hard-coded in the circuit (we explain how it can be done later in this section). Prior to executing the original next-message function, the modified function decrypts the state using the hard-coded secret key of $P_j$ and verifies the signatures on each public message as well as the signature on the state passed in from previous round. Intuitively, these modifications are due to the following reasons:

- The state of the party is passed in an *encrypted state* because the state information is assumed to be private in the original MPC construction.
- The parties need to *sign* their messages (and *verify* signatures on the messages passed as inputs) since we must prevent the adversary from tricking an honest party into acceptance of a message that is supposedly generated by another honest party, but in reality is mauled by the adversary.

Once the garbled circuits are prepared, $P_j$ stores the garbled circuits with CSaR. Note that the next-round functions in particular take messages produced by *other parties* as inputs. Thus, there is no way for the party to know at the time the garbled circuits are constructed, whether the key corresponding to bit 0 or the key corresponding to bit 1 will be chosen for some wire $w$. To allow an evaluator to execute the garbled circuits anyway, $P_j$ additionally stores both wire keys for each input wire with CSaR, each with a separate CSaR request. This needs to be done for every single round, since in any particular round

the inputs will depend on the messages produced by the garbled circuits of other parties in the previous round.

Intuitively, in order to be able to reduce the security of this protocol to the security of the original MPC protocol, we need to ensure not only that the adversary is not able to maul messages of the honest parties and see the parties' private information, but also that the protocol is executed in order and there is only a single instance of the protocol running. This is ensured by carefully constructing conditions that must be met in order to release the garbled circuits and wire keys. In order to release a garbled circuit for some round $i$, a party needs to provide a proof that the execution of the protocol up to and including round $i-1$ is finalized. In order to release a wire key corresponding to bit $b$ on a wire corresponding to position $p$ of the input to some garbled circuit, a party needs to additionally provide a proof that the input bit to position $p$ in this circuit is indeed bit $b$. In the following, we first explain how the protocol is executed, and then explain how exactly the release conditions look like.

**Step 2. Executing $\pi$.** Once all required information is stored, an evaluator $E$ can execute the original MPC protocol $\pi$. It is not required that $E$ is one of the parties participating in the protocol $\pi$ and in fact, there can be multiple evaluators (for simplicity, we refer to all of them as "$E$"). $E$ executes the garbled circuits round-by-round. Once $E$ has executed all garbled circuits for a certain round, $E$ publishes the concatenation of the outputs of these circuits on a bulletin board. Then, $E$ uses the proof of publishing of this message in order to release the garbled circuits as well as the wire keys of the next round.

**First round optimization.** Note that the message broadcast by the parties in the first round of the protocol $\pi$ does not require any information from the other participants in the MPC protocol. Thus, instead of storing the garbled circuits for the first round, we let the parties publish their first message (and the signature on it) directly. The secret state that needs to be passed to the second round is hard-coded in the garbled circuit of the second round.

**Release conditions.** As described above, after the execution of all garbled circuits of the certain round, the evaluator is tasked with publishing the (concatenation of the) outputs of these circuits. This published message servers as a commitment to the evaluator's execution of this round, and this is what is needed to release the gabled circuits of the next round. We additionally require that the length of each published message is the same as expected by the protocol (corresponds to the number of input wires), and the correct length requirement holds for every part of this message (i.e., the public message, the signature over it, the state, and the signature over the state for each contributing party). In order to

ensure that there is only a *single* evaluation of the original MPC running, only the *very first* published message that is of a correct form (i.e., satisfies the length requirements) can be used as the witness to release garbled circuits and keys of a certain round. We call such messages *authoritative* messages. Formally, the authoritative message of round $d > 1$ is a published message that satisfies the following conditions:

- Message is of the form $(id, d, m)$, where $m$ is of the form $(m_1^d \| \cdots \| m_n^d \| sigpub_1^d \| \cdots \| sigpub_n^d \| c_1^d \| \cdots \| c_n^d \| sigpr_1^d \| \cdots \| sigpr_n^d)$. This corresponds to the concatenated output of the garbled circuits of round $d$: public messages followed by signatures over each public message, and encryptions of state followed by signatures over each ciphertext.
- each $m_j^d$, $c_j^d$, $sigpub_j^d$, $sigpr_j^d$ has correct length.
- This is the first published message that satisfies the requirements above.

Due to our first round optimization the authoritative message of the first round is slightly different. In particular, there are up to $n$ authoritative messages for the first round – one for each contributing party. Formally, an authoritative message of round $d = 1$ from party $P_k$ is a published message that satisfies the following conditions:

- Message is of the form $(id, 1, k, m_k^1, sigpub_k^1)$.
- $m_k^1$ and $sigpub_k^1$ both have correct length.
- This is the first published message that satisfies the requirements above.

In terms of authoritative messages, the release conditions can be now defined as follows: in order to release the garbled circuits for round $i$, we require that all authoritative messages for rounds 1 up to and including round $i - 1$ are published. In order to release the wire key for some bit $b$ of an input wire $w$ of a garbled circuit the authoritative message of the previous round must contain bit $b$ at the same position $w$.

**Identifying secrets.** In order for the evaluator to know the identifiers of the secrets it must request from CSaR, we require that upon storing the secrets (i.e., garbled circuits and wire keys), the contributors choose their CSaR secret identifiers (appending their own party identifier to the secret in order to ensure that it has not been used before) and publish those identifiers on the bulletin board (we assume messages can't be posted or stored by a party pretending to be another party). For readability purposes, further we exclude this detail from the construction description.

**Removing point-to-point channels.** While in our construction we assume that the original MPC protocol is in a broadcast model, it is very common for MPC protocols to assume secure point-to-point channels. We can handle such protocols as well since an MPC

| x | w | out |
|---|---|---|
| 0 | 0 | $K_0$ |
| 0 | 1 | $K_0$ |
| 1 | 0 | $K_0$ |
| 1 | 1 | $K_1$ |

| x | out |
|---|---|
| 0 | $K_0$ |
| 1 | $K_0$ |

**Figure 2.7:** On the left, we show the computation of the AND-gate in Yao's construction. Given the garbled keys of $x$ and $w$, depending on whether they correspond to zero or one, the doubly-encrypted ciphertext contains $K_0$ or $K_1$. On the right, we show the computation for the AND-gate if $w = 0$. In this case, both ciphertexts contain $K_0$.

protocol that assumes point-to-point channels can be easily converted to a protocol in a broadcast model, see "Subtleties of Point-to-Point Channels" in Section 2.3.2.

**Hardcoding secret inputs.** As mentioned above, some of the information used in the modified next-message function (such as the secrets of the parties, their secret keys etc.) is hardcoded in the circuit. Say the hardcoded input wire is $w$, and its value is (bit) $b$. Then, the party preparing the garbled circuit that uses $w$ does so as follows: whenever one of the inputs to a gate is $w$, the party removes the wire corresponding to $w$ from the circuit and computes the values in the ciphertexts using bit $b$ only (instead of computing the output both for $w = 0$ and $w = 1$). We give an example for the computation of the AND-Gate in Figure 2.7. For security purposes, it is important that we do *not* perform any circuit optimizations based on the value of $w$.

**Notation.** In the following, we denote party $P_j$'s public and secret encryption key pair as $(pk_j, sk_j)$. We denote party $P_j$'s signature and verification keys as $sigk_j$ and $verk_j$. By $m_j^i$ we denote messages that are generated by the party $P_j$ in the $i$-th round.

**Further Details.** Note that eWEB, the construction that we use as the instantiation of the CSaR, assumes a CRS. This requirement can be removed in our case by simply allowing each participant in the protocol $\pi$ to prepare the CRS on its own. From a security standpoint, this is unproblematic – we only wish to protect the secrets of honest clients, and if a client is honest, it will generate the CRS honestly as well [7].

Additionally, we note that in eWEB the party storing the secret is required to send multiple messages. In order to ensure that in our MPC protocol a single message from the

[7]Note that this change reduces the efficiency of the eWEB system – instead of batching secrets from different clients, only secrets from a single client can be processed together now.

MPC participant is sufficient and the parties can go offline after sending this message, we slightly modify the eWEB construction. Roughly, in eWEB miners are tasked with jointly preparing a random value $r$ s.t. each miner knows a share of $r$. The user then publishes the value $s + r$ (where $s$ denotes the secret to be stored), and the miners compute their shares of $s$ by subtracting their shares of $r$ from $s + r$. Along the way, the commitments to the sharing of $s$ are made public. We modify it as follows: the user simply publishes the commitments to the sharing of $s$ and sends shares of $s$ (along with the witnesses) to the miners who then verify the correctness of the shares and witnesses.

Finally, note that we require that the original protocol $\pi$ has the publicly recoverable output property (see Definition 3). For security with abort, this property can be easily achieved as follows: first, all parties broadcast the output. Then, if all parties broadcasted the same value, this value is taken as the output. Otherwise, protocol is considered to be aborted. In the following, for simplicity we assume that protocol $\pi$ has the publicly recoverable output property and $\mathsf{Eval}$ denotes the algorithm used to retrieve the output from the transcript.

The full construction is given in Protocols 6 and 7 (preparation of the garbled circuits and keys), as well as Protocol 8 (execution phase).

**Security Analysis.** Intuitively, correctness of the construction as well as the secrecy of the honest parties' inputs follow from the correctness as well as security properties of the underlying cryptographic primitives as well as the original protocol $\pi$. We formally show security by providing a simulator in the ideal model and showing that no PPT adversary can distinguish between interaction with the simulator and the interaction with the honest parties. Intuitively, we rely on the security of the cryptographic primitives used in our construction to show that the adversary is not able to use a garbled circuit from an honest party in a "wrong" way. In particular, the adversary cannot trick an honestly produced garbled circuit into accepting wrong inputs from other honest parties i.e., inputs that were not produced using the garbled circuits or published (for the first message) by those parties directly, or claim that a required message from some honest party is missing. Additionally, there is no way for the adversary to execute honest garbled circuits for the same round on inconsistent inputs (or execute a single honest garbled circuit multiple times on a different inputs) since only the authoritative message published for a single round is considered valid. We then rely on the security of the original protocol $\pi$. We give the formal proof in Section 2.5.

**Protocol 6** NON-INTERACTIVE MPC – *CircuitPreparationPhase*

1. $P_j$ computes the output $(m_j^1, s_j^1)$ of the first round of $\pi$. $P_j$ computes the signature $sigpub_j^1$ on the message $(id, 1, j, m_j^1)$ using its signing key $sigk_j$. $P_j$ posts $\mathtt{M_j^1} = (id, 1, j, m_j^1, sigpub_j^1)$ on the bulletin board.
2. $P_j$ produces Yao's garbled circuits $\{GC_j^i\}$ for each round $i > 1$ based on the circuit $C_j^i$ of the next-message function $f_j^i$ of the original MPC protocol $\pi$:

$$(\{\mathtt{lab}_j^{w,b,i}\}_{w\in\mathtt{inp_j^i},b\in\{0,1\}}) \leftarrow \mathtt{Gen}(1^\lambda, \mathtt{inp_j^i})$$

$$GC_j^i \leftarrow \mathtt{Garble}(C_j^i, (\{\mathtt{lab}_j^{w,b,i}\}_{w\in\mathtt{inp_j^i},b\in\{0,1\}})$$

   Here, $\mathtt{inp_j^i}$ is the length of the input to the circuit $C_j^i$. This circuit takes as input messages $\{m_k^{i-1}\}_{k=1}^n$ published by the parties in the previous round along with the signatures $\{sigpub_k^{i-1}\}_{k=1}^n$ of these messages, and the encryption $c_j^{i-1}$ of the secret state passed by $P_j$ from the previous round as well as the signature $sigpr_j^{i-1}$ over this ciphertext. All of $P_j$'s keys, input $x_j$ and randomness $r_j^i$ are hardcoded in the circuit. The verification and public keys of other participants are also hardcoded in the circuit. For the circuit of the second round, the secret state passed from the first round is also hardcoded in the circuit. The circuit decrypts the secret state and, if the ciphertext was correctly authenticated, executes the next message function of the current round:
   (a) If $i = 2$, proceed to step 2(c).
   (b) Verify the signature on the tuple $(id, i-1, j, c_j^{i-1})$ using $verk_j$. If this check fails, stop the execution and output $\perp$.
   (c) Verify the signature on the tuple $(id, i-1, z, m_z^{i-1})$ from party $P_z$. If any verification check fails, stop the execution and output $\perp$.
   (d) Compute $s_j^{i-1} = Dec_{sk_j}(c_j^{i-1})$.
   (e) Obtain $(m_j^i, s_j^i)$ by executing $f_j^i(x_j, r_j^i, m^i, s_j^{i-1})$, where $m^i = m_1^{i-1} \| \cdots \| m_n^{i-1}$.
   (f) Compute the signature $sigpub_j^i$ on the public message $(id, i, j, m_j^i)$ using the signing key $sigk_j$.
   (g) Compute the encryption of the state $c_j^i = Enc_{pk_j}(s_j^i)$.
   (h) Compute the signature $sigpr_j^i$ on the tuple $(id, i, j, c_j^i)$ including the encryption of state using the signing key $sigk_j$.
   (i) Output $(m_j^i, sigpub_j^i, c_j^i, sigpr_j^i)$.
3. $P_j$ securely stores garbled circuits $GC_j^i$ for all rounds $i > 1$ using a CSaR. The witness needed to release the garbled circuit of round $i$ is a valid proof of publishing of all authoritative messages from round 1 and up to and including round $i-1$.

**Protocol 7** NON-INTERACTIVE MPC – *KeyStoragePhase*

1. Securely store input wire keys $(\{\mathtt{lab}_j^{w,b,2}\}_{w\in\mathtt{inp}_j^2,b\in\{0,1\}})$ for the circuit of the second round using CSaR. For each party $P_k$ whose first round message is needed for the computation, the witness required to decrypt the wire key corresponding to the $i$-th bit of the input being 0 (resp. 1) is a **valid proof of publishing** of the following:
   (a) All of the authoritative messages of the first round.
   (b) $i$-th bit of the authoritative message of round 1 of Party $P_k$ is 0 (resp. 1).
2. Securely store input wire keys $(\{\mathtt{lab}_j^{w,b,d}\}_{w\in\mathtt{inp}_j^d,b\in\{0,1\}})$ for the circuit of the $d$-th $(d \geq 3)$ round using CSaR. The witness needed to decrypt the wire key corresponding to the $i$-th bit of the input being 0 (resp. 1) is a **valid proof of publishing** of the following:
   (a) All of the authoritative messages of the first $d-1$ rounds.
   (b) $i$-th bit of the authoritative message of round $d-1$ is 0 (resp. 1).

## 2.5 Security Proof - Main Construction

Formally, we show that our construction supports the MPC functionality $\mathcal{F}_{\text{eval-MPC}}$ described in Functionality 2.3.

We do so by constructing the simulator $S$ using the CSaR simulator $S_{CSaR}$, the garbled circuit simulator $S_{GC}$, and the simulator of the original MPC protocol $S_{MPC}$. Intuitively, our end goal is to arrive at the point where we only have access to the honest parties' secrets via the ideal functionality $\mathcal{F}_{\text{eval-MPC}}$ – then we have shown that the parties' secrets are safe. In order to use the CSaR simulator (which needs access to an ideal functionality $\mathsf{Ideal}_{\text{CSaR}}$), $S$ simulates $\mathsf{Ideal}_{\text{CSaR}}$ by itself. In order to use the garbled circuit simulator $S_{GC}$ which takes as input the output that it needs to compute, $S$ uses the messages output by the MPC simulator $S_{MPC}$. Note that whenever we use the simulator $S_{GC}$, we also give it as input the circuit representation of the next-message function of the according round (for the according party) as specified by our construction. For ease of presentation, we skip this detail in the following proof.

We construct the simulator $S$ as follows:

$S$ starts by choosing the public and secret keys $(pk_j, sk_j)$, as well as the signing and verification keys $(sigk_j, verk_j)$ for the honest parties, and initializing an empty list $L$ which will be later used for the secret storage when simulating $\mathsf{Ideal}_{\text{CSaR}}$. Then, $S$ starts the real-world adversary $\mathcal{A}$, and gives $\mathcal{A}$ the public and verification keys. Additionally, $S$ starts the CSaR simulator $S_{CSaR}$. $S$ also starts the simulator $S_{MPC}$ for MPC functionality $f$ and

**Protocol 8** NON-INTERACTIVE MPC – *ExecutionPhase*

1. The evaluator $E$ uses messages $(id, 1, z, m_z^1, sigpub_z^1)$ posted on the bulletin board by each party $P_z$ as the proof of publishing to get the garbled circuits (and keys) for the second round stored in CSaR by each participant in $\pi$. Then, $E$ computes the outputs $(m_j^2, sigpub_j^2, c_j^2, sigpr_j^2)$ of the second round by executing the garbled circuits.
2. If an authoritative message of the second round was not published on the bulletin board yet, set $m = (m_1^2 \| \cdots \| m_n^2 \| sigpub_1^2 \| \cdots \| sigpub_n^2 \| c_1^2 \| \cdots \| c_n^2 \| sigpr_1^2 \| \cdots \| sigpr_n^2)$, publish $\mathtt{M^2} = (id, 2, m)$:
$$(\mathtt{post}^2, \sigma^2) \leftarrow \mathtt{Post}(\mathtt{M}^2)$$
and use the proof of publish $\sigma^2$ as the witness to decrypt the wire keys and the garbled circuits of the next round. If an authoritative message $(id, 2, m')$ was published on the bulletin board, use its proof of publishing as the witness if $m' = m$. Otherwise, stop the execution and output $\perp$.
3. In each following round $d \geq 3$, $E$ executes each garbled circuit published by party $P_z$ for round $d-1$. Then, $E$ concatenates the outputs and checks if there is a message on the bulletin board for this round. If there is no such message, $E$ posts the computed output $\mathtt{M^d} = (id, d, m_1^{d-1} \| \cdots \| m_n^{d-1} \| sigpub_1^{d-1} \| \cdots \| sigpub_n^{d-1} \| c_1^{d-1} \| \cdots \| c_n^{d-1} \| sigpr_1^{d-1} \| \cdots \| sigpr_n^{d-1})$:
$$(\mathtt{post}^\mathtt{d}, \sigma^d) \leftarrow \mathtt{Post}(\mathtt{M}^\mathtt{d})$$
and uses the proof of publishing $\sigma^d$ as witness to obtain input keys and garbled circuits of the next round. Otherwise, if a message for this round is already published and is the same as the one computed by $E$, $E$ uses the proof of publishing of this message as the witness. If it is not the same message as the one computed by $E$, $E$ aborts the execution.
4. Let $\tau$ denote the resulting transcript of execution of $\pi$. $E$ outputs $\mathsf{Eval}(\tau)$ as the result.

secure protocol $\pi$. Whenever $S_{MPC}$ sends a message to its ideal functionality, $S$ forwards this message to its own ideal functionality (and vice versa).

Note that the CSaR simulator $S_{CSaR}$ is running during the whole execution of $S$, and requires access to the ideal functionality $\mathsf{Ideal}_{\mathrm{CSaR}}$. We first explain how exactly $S$ is simulating the CSaR infrastructure, and then explain how $S$ is simulating each phase of the protocol.

**Simulation of the CSaR infrastructure.**

- $S$ honestly simulates $\mathsf{Ideal}_{\mathrm{CSaR}}$ by keeping the list $L$ of (identifier, release condition,

secret) tuples and honestly storing messages whenever such requests come from $S_{CSaR}$. Messages stored by the adversary are not only stored, but also released honestly. For an honest message the simulator $S$ decides on the fly whether and what it returns (acting as $\mathsf{Ideal}_{\mathrm{CSaR}}$) (see the description of the next two phases for details). If $S$ is unable to provide a response to a valid release request of an honest message, $S$ aborts.

**Simulation of the CircuitPreparationPhase.**

- In Step 1 of the *CircuitPreparationPhase*, for each honest party $P_j$, $S$ waits until it receives the message $m_j^1$ output by $S_{MPC}$ as the party $P_j$, honestly computes the signature $sigpub_j^1$ over it, and posts $(id, 1, j, m_j^1, sigpub_j^1)$ on the bulletin board.
- In Step 2, to construct garbled circuits for round $i$, $S$ waits until all authoritative messages of the first $i-1$ rounds are published (either by the adversary, or, if applicable, by $S$ itself). Then, $S$ forwards each message $m_z^{i-1}$ that is supposed to represent the public message of a corrupted party $P_z$ in the authoritative message of round $i-1$ to $S_{MPC}$ (as if coming from $P_z$).

  To construct the garbled circuit of an honest party $P_j$ in round $i$, $S$ uses the garbled circuit simulator $S_{GC}$. If $P_j$'s garbled circuit is supposed to use an input $m$ according to our protocol, $S$ verifies that the authoritative message of round $i-1$ contains a valid signature on $m$. If not, $S$ uses the garbled circuit simulator on the input $\perp$. If all the required signatures are valid, $S$ uses the garbled circuit simulator on the input $(m_j^i, sigpub_j^i, c_j^i, sipr_j^i)$. Here, $m_j^i$ is the output of $S_{MPC}$ for party $P_j$ in round $i$, $sigpub_j^i$ is the signature honestly computed by $S$ over $(id, i, j, m_j^i)$, $c_j^i$ the encryption of a zero string, and $sipr_j^i$ the signature honestly computed by $S$ over $(id, i, j, c_j^i)$.
- When an honest party is supposed to store a garbled circuit as specified by Step 3, $S$ simulates the SecretStorage step of $\mathsf{Ideal}_{\mathrm{CSaR}}$ by informing $S_{CSaR}$ that a secret has been stored with the release condition as specified by Step 3 of the *CircuitPreparationPhase*. Once a valid release request for this garbled circuit has been submitted to $S_{CSaR}$, $S$ checks whether it was able to construct a garbled circuit according to the procedure outlined in the simulation of Step 2, and if so, honestly simulates $\mathsf{Ideal}_{\mathrm{CSaR}}$ by storing the constructed garbled circuit in the list $L$ and then honestly releasing it to $S_{CSaR}$. Otherwise, $S$ aborts.

**Simulation of the KeyStoragePhase.**

- For all wire keys that must be stored according to Step 1 and Step 2 of the *KeyStoragePhase*, $S$ simulates the SecretStorage step $\mathsf{Ideal}_{\mathrm{CSaR}}$ by informing $S_{CSaR}$ that a secret has been stored with the release condition as specified by Step 1 and Step 2 of the *KeyStoragePhase*. Once a valid release request has been submitted for a wire key,

$S$ checks whether it was able to create the requested garbled key according to the procedure outlined in Step 2 of the simulation of the CircuitPreparationPhase, and if so, $S$ honestly simulates $\mathsf{Ideal}_{\mathrm{CSaR}}$ by storing this garbled wire key in the list $L$ and then honestly releasing it to $S_{CSaR}$. Otherwise, $S$ aborts.

**Simulation of the ExecutionPhase.**

- Note that the evaluator does not possess any secrets. Thus, $S$ simply follows the procedure outlined in Protocol 8, sending its CSaR release requests to $S_{CSaR}$.

Now, we prove that no PPT adversary $\mathcal{A}$ is able to distinguish the view of interaction with the simulator $S$ constructed above from the view of interacting with honest parties in the real world. We prove it by establishing a series of hybrids such that any two consecutive hybrids are indistinguishable. We denote the advantage of the adversary in distinguishing the between the $\mathbf{Hybrid}_{i-1}$ and $\mathbf{Hybrid}_i$ by $\epsilon_i$, and define the hybrids as follows:

**$\mathbf{Hybrid_0}$:** This hybrid corresponds to the execution in the real world. The simulator $S$ controls all honest parties and follows the protocol.

**$\mathbf{Hybrid_1}$:** The simulator $S$ behaves the same as in the previous protocol, except that it switches from honestly executing the CSaR protocol to using the CSaR simulator $S_{CSaR}$ while honestly simulating $\mathsf{Ideal}_{\mathrm{CSaR}}$ by himself. In more detail, $S$ passes messages to and from $S_{CSaR}$ and the dishonest parties, as well as simulates $\mathsf{Ideal}_{\mathrm{CSaR}}$ by storing a list $L$ of (identifier, release condition, secret) tuples as follows:

- Upon receiving a secret $s$ and a release condition $F$ from $S_{CSaR}$, the simulator $S$ stores $(id, F, s)$, where $id$ is the identifier of the CSaR request.
- Whenever an honest message needs to be stored, $S$ honestly stores it in $L$ and notifies $S_{CSaR}$.
- Whenever $S_{CSaR}$ queries $\mathsf{Ideal}_{\mathrm{CSaR}}$ for a secret with the identifier $id$, the simulator $S$ checks whether the entry with the identifier $id$ exists, and if so, whether the given witness satisfies the release condition of this entry. If so, the simulator looks up the list $L$ of stored tuples and returns the corresponding secret $s$ to $S_{CSaR}$.

**Lemma 3.** *For the hybrids* $\boldsymbol{Hybrid_1}$ *and* $\boldsymbol{Hybrid_0}$ *holds:* $\epsilon_1 \leq negl_1(n)$.

*Proof.* Intuitively, this holds by the security of the CSaR protocol. In more detail, assume that there exists an adversary $\mathcal{A}$ able to distinguish between $\mathbf{Hybrid_1}$ and $\mathbf{Hybrid_0}$. Then, we can construct an adversary $\mathcal{B}$ for the CSaR protocol. $\mathcal{B}$ starts by choosing the public and secret keys for the honest parties, sends the public keys to $\mathcal{A}$, constructs the garbled circuits as specified by Protocol 6, and posts the messages of the first round on the bulletin board.

Whenever a CSaR message needs to be passed from $\mathcal{A}$ to an honest party, $\mathcal{B}$ forwards it to its challenger. Whenever the challenger passes an CSaR message to a dishonest party, $\mathcal{B}$ forwards it to $\mathcal{A}$. Additionally, $\mathcal{B}$ passes messages to and from honest clients and the challenger. Now, if $\mathcal{B}$'s challenger uses the real CSaR protocol, the game $\mathcal{A}$ is in is exactly $\mathbf{Hybrid_0}$, while if $\mathcal{B}$'s challenger uses the simulator, the game $\mathcal{A}$ is in is exactly $\mathbf{Hybrid_1}$. Thus, $\mathcal{B}$'s advantage is at least the same as the advantage of $\mathcal{A}$. Since the advantage of $\mathcal{B}$ is negligible by the security of the CSaR protocol we use, the advantage of $\mathcal{A}$ must be negligible. □

**$\mathbf{Hybrid_2}$:** The simulator behaves the same as in the previous round, except that it changes the way $\mathsf{Ideal}_{\mathrm{CSaR}}$ is simulated. Specifically, instead of saving the honest parties' secrets (wire keys, garbled circuits) in list $L$ at the time specified by the protocol, the simulator stores each secret in $L$ only when the simulator asks for this secret and is able to provide a valid witness for the corresponding release condition.

**Lemma 4.** *For the hybrids* $\boldsymbol{Hybrid_2}$ *and* $\boldsymbol{Hybrid_1}$ *holds:* $\epsilon_2 = 0$.

*Proof.* Note that in $\mathsf{Ideal}_{\mathrm{CSaR}}$, access to a secret is needed only upon a valid release request for this secret. Thus, it makes no difference whether the message is stored in $L$ at the time that is specified by the protocol or only upon a valid release request. □

**$\mathbf{Hybrid_3}$:** The simulator $S$ behaves the same as in the previous round, except that it aborts if $S_{CSaR}$ provides a valid release request for an honest input wire key or an honest garbled circuit that does not satisfy one of the requirements outlined in Protocols 6 and 7 based on $S$'s own view of the computation, i.e, corresponds to bit $1 - b_p$ at some position $p$ in the string, when the authoritative message's bit in this position is $b_p$, or corresponds to some position which does not have a recorded authoritative message on the bulletin board yet). We denote this by $\mathsf{abort}_1$.

**Lemma 5.** *For the hybrids* $\boldsymbol{Hybrid_3}$ *and* $\boldsymbol{Hybrid_2}$ *holds:* $\epsilon_3 \leq negl_3(n)$

*Proof.* Note that the simulator $S$ aborts only if $S_{CSaR}$ provides a valid request for some wire key $z$ such that one of the release requirements either of Protocol 6 or of Protocol 7 does not hold. Since we know that $S_{CSaR}$ provides a valid request, and at the same time the release requirement does not hold based on $S$'s view of the bulletin board, it means that as a part of its execution, the adversary $\mathcal{A}$ is able to provide a forged proof of publishing. If $\mathcal{A}$ is able to do so with a non-negligible probability, we can use it to forge a proof of publishing with non-negligible probability as well. □

**$\mathbf{Hybrid_4}$:** The simulator $S$ behaves the same as in the previous hybrid, except that it now uses the garbled circuit simulator $S_{GC}$ instead of honestly constructing the garbled circuit. Specifically, once the adversary published its authoritative message for round $i-1$, the simulator $S$ honestly computes the output of the garbled circuit (consisting of the public message, the encrypted state, and the signatures) of an honest party $P_j$ in round $i$ using the authoritative message from round $i-1$, as well as honest party's input, public and secret keys, signature and verification keys, and (for the garbled circuit of the second round) the honest parties' state from the first round. Denote the output of party $P_j$'s garbled circuit of round $i$ by $out_i^j$. Then, $S$ uses the garbled circuit simulator to construct the circuit for round $i$ of an honest party $P_j$: $gc_i^j = S_{GC}(out_i^j)$ and uses the result instead of the actual garbled circuit.

**Lemma 6.** *For the hybrids* $\boldsymbol{Hybrid_4}$ *and* $\boldsymbol{Hybrid_3}$ *holds:* $\epsilon_4 \leq negl_4(n)$.

*Proof.* Technically, this is a series of hybrids where the circuits are replaced one after the other (starting with the circuits of the first round). Note that at this point the adversarial input is guaranteed to be known before the circuit is constructed. Thus, $S$ is always able to correctly compute the output, and the statement holds by the selective security of the garbled circuit construction. □

**$\mathbf{Hybrid_5}$:** The simulator behaves the same as in the previous hybrid, except that the simulation of the garbled circuits is done a bit differently. Specifically, we change the input we provide to the garbled circuit simulator $S_{GC}$: instead of using an encryption of the state, we use an encryption of zeroes (the signature is computed on this encryption of zeroes).

**Lemma 7.** *For the hybrids* $\boldsymbol{Hybrid_5}$ *and* $\boldsymbol{Hybrid_4}$ *holds:* $\epsilon_5 \leq negl_5(n)$.

*Proof.* Technically, this is a series of hybrids where the encryptions of the state are replaced one after the other. Note that at this point, the simulator $S$ does not use the secret keys of the honest parties anymore. By the security of the encryption scheme, in each hybrid, the distribution of the input we give to the garbled circuit simulator $S_{GC}$ is computationally indistinguishable from the input in the previous hybrid. Thus, the input distribution of the adversary does not change as well. Therefore, each two consecutive hybrids (and thus $\mathbf{Hybrid_4}$ and $\mathbf{Hybrid_5}$ as well) are indistinguishable. □

**$\mathbf{Hybrid_6}$:** The simulator $S$ behaves the same as in the previous hybrid, except that it aborts if the adversary posts an authoritative message for some round $i$ such that some honest part of it (public message or state that is supposed to be produced by the honest party) is not consistent with what the simulator expects based on the authoritative message

of the previous round, but still *has a valid signature*. Specifically, the simulator computes the output of the honest party's garbled circuit on the authoritative message of the previous round, and checks whether this message is the same as what is given in the authoritative message of the current round. We denote this by $\mathsf{abort}_2$.

**Lemma 8.** *For the hybrids* $\boldsymbol{Hybrid_6}$ *and* $\boldsymbol{Hybrid_5}$ *holds:* $|\epsilon_6 - \epsilon_5| \leq negl_6(n)$

*Proof.* Note that if the adversary is able to post such message, we can use this adversary to construct an adversary against the unforgeability of the signature scheme that we use. Thus, this situation can occur only with some negligible probability.

In more detail, this is a series of hybrids where in each hybrid we target one honest party at a time. Assume that there exists an adversary $\mathcal{A}$ able to distinguish between two consecutive hybrids that differ only in the fact that the adversary posts an unexpected, correctly signed message for some honest party $P_j$. Then we can construct an adversary $\mathcal{B}$ against the security of the signature scheme. $\mathcal{B}$ starts by choosing the public and secret keys of the honest parties (except for the signature/verification key of party $P_j$ - those keys are chosen by $\mathcal{B}$'s challenger), sends the public keys to $\mathcal{A}$, constructs the garbled circuits, posts messages of the first round on the bulletin board etc. as specified by the description of the simulator in the previous hybrid. Whenever $\mathcal{B}$ needs to sign a message for $P_j$, it uses the signature oracle. Now, if $\mathcal{A}$ outputs an unexpected correctly signed message for $P_j$, $\mathcal{B}$ can use this message to present the forgery to its own challenger. Thus, $\mathcal{B}$'s advantage is at least the same as the advantage of $\mathcal{A}$. Since the advantage of $\mathcal{B}$ is negligible by the security of the signature scheme we use, the advantage of $\mathcal{A}$ must be negligible as well. □

**Hybrid$_7$:** The simulator $S$ behaves the same as in the previous hybrid, except that if the adversary posts an authoritative message for some round $i-1$ such that some part of it does not have a valid signature, the simulator changes the way that the garbled circuits for round $i$ that use this partial message is generated: instead of computing the output using the inputs and then using the garbled circuit simulator as is done in the previous hybrid, the simulator $S$ computes the garbled circuit directly as $S_{GC}(\perp)$.

**Lemma 9.** *For the hybrids* $\boldsymbol{Hybrid_7}$ *and* $\boldsymbol{Hybrid_6}$ *holds:* $|\epsilon_7 - \epsilon_6| = 0$

*Proof.* Note that the garbled circuit of an honest party would have output $\perp$ anyway due to the signature verification check. Thus, nothing has changed.

□

**Hybrid$_8$:** The simulator $S$ behaves the same as in the previous hybrid, except that if

the adversary posts a message for some round $i-1$ such that some honest part of it is not consistent with the simulator's expectations based on the authoritative message of the previous round, the simulator changes the way that the garbled circuits for round $i-1$ that use this honest message are generated: instead of computing the output using the inputs and then using the garbled circuit simulator as is done in $\mathbf{Hybrid_9}$, the simulator $S$ computes the garbled circuit directly as $S_{GC}(\perp)$.

**Lemma 10.** *For the hybrids* $\boldsymbol{Hybrid_8}$ *and* $\boldsymbol{Hybrid_7}$ *holds:* $|\epsilon_8 - \epsilon_7| = 0$

*Proof.* Note that at this point, due to the steps made in the previous two hybrids, we know that the signature on the changed message is invalid. Thus, any honest garbled circuit that uses this changed message will output $\perp$ either as part of Step 2b) or part of Step 2c) of Protocol 6. Thus, the input we feed into the garbled circuit simulator does not change.

□

**$\mathbf{Hybrid_9}$:** Consider the garbled circuits of the honest parties that were computed *not* using the garbled circuit simulator with the input $\perp$. Note that $S$ currently computes those garbled circuits by using the garbled circuit simulator $S_{GC}$ on the output that $S$ honestly computed based on the authoritative message of the previous round as well as the honest party's input and state from the previous round. In this hybrid, we remove the requirement of knowing the honest party's input and state. Specifically, the simulator behaves the same as in the previous hybrid, except that instead of honestly computing the output using the honest parties' inputs and states, it relies on the simulator $S_{MPC}$ of the original MPC protocol $\pi$ to retrieve the public messages that are supposed to be output by those garbled circuits that were not already generated by the garbled circuit simulator using the input $\perp$. Note that in $\mathbf{Hybrid_5}$ we already changed the encryption of state to encryption of zeroes, so once we retrieved the public messages, we are done.

**Lemma 11.** *For the hybrids* $\boldsymbol{Hybrid_9}$ *and* $\boldsymbol{Hybrid_8}$ *holds:* $|\epsilon_9 - \epsilon_8| \leq negl_9(n)$

*Proof.* At this point, note the the adversary is not able to misbehave more than it can in the execution of the protocol $\pi$. Thus, the indistinguishability of the hybrids holds by the security of the original MPC protocol $\pi$. □

Note that in the last hybrid, the simulator does not need the honest parties' inputs to simulate the execution.

## 2.6 Guaranteed Output Delivery

In this section, we provide an extension of our main construction that ensures guaranteed output delivery, meaning that the corrupted parties cannot prevent honest parties from receiving their output.

In order to provide guaranteed output delivery, the first step is to build upon an MPC protocol $\pi$ that also has this property. However, note that this change by itself is not sufficient – a malicious evaluator could still disrupt the execution of our original construction by simply providing an authoritative message that contains an invalid signature and thus forcing honest garbled circuits to abort. It is clear that we cannot simply accept such invalid signatures. Thus, further modifications are required. In general, compared to our main protocol we make the following changes:

- The original MPC protocol must have the guaranteed output delivery property.
- We introduce a deadline by which all initial messages must be posted. In the following, we denote this deadline by $\tau$.
- Signatures on the messages are verified not by the garbled circuits, but rather by the CSaR parties as part of the CSaR request. The signature is computed on the whole message, rather than separately for the public and state parts of the next-message function's output.
- We use CSaR with public release, which is similar to CSaR, but instead of privately releasing secret shares to the user, the parties release the shares publicly (e.g., by posting them on the bulletin board).
- As a part of the release condition, the garbled circuits and wire keys of the current round (that were previously published on the bullet board) are used to check whether the message submitted by the evaluator is indeed the output of the garbled circuit in question. Only if this is the case (i.e., the evaluator acted honestly) is the evaluator allowed to receive the next wire keys. The evaluator uses a proof of publishing of the garbled circuits and the wire keys released by the CSaR to prove the correctness of the computation. Roughly the following statement is checked: "The execution of the garbled circuit $GC$ on the wire keys $\{k_i\}_{i\in I}$ results in the output provided by $E$. Here, the garbled circuit $GC$ is the circuit, and $\{k_i\}_{i\in I}$ are the keys for this circuit reconstructed using the published values of the CSaR present on the proof of publish supplied by $E$".
- If a message from the first round was not published, or a garbled circuit or wire key from some party was not stored with CSaR, the evaluator needs to prove that with

respect to the genesis block, by deadline $\tau$ indeed no such message was stored. We call such proof a "proof of missing message".

- In the cases described in the last two points, the CSaR releases default wire keys (encoding "$\perp$") for each garbled circuit that is supposed to use the missing message.

In order to allow for an easy verification of the evaluator's claims of invalid garbled circuits, we use CSaR with public release (CSaR-PR, see Figure 2.6), which is the same as CSaR, except that the witness is supplied by the client that wishes to receive the secrets publicly, and the secrets (garbled circuits and wire keys in our case) are released publicly as well (as long as the release condition is satisfied). Such CSaR-PR can be instantiated with the PublicWitness construction presented in the eWEB work. For simplicity, in the following we assume that the public release of the computation result is permitted. If the application requires that only a certain party obtains the function result, it can be easily supported by changing the output of the function that is being computed to the *encryption* of this output under that party's public key.

The definition of the *authoritative* message for this construction is a bit different from the definition in our main construction to account for the fact that the signatures and proofs of execution are checked by the CSaR parties. Formally, the authoritative message of round $d > 1$ is a published message that satisfies the following conditions:

- Message is of the form $(id, d, m)$, where $m$ is of the form $(m_1^d \parallel \cdots \parallel m_n^d \parallel c_1^d \parallel \cdots \parallel c_n^d \parallel sig_1^d \parallel \cdots \parallel sig_n^d \| \mathcal{P})$, where $\mathcal{P}$ is some additional proof data, as explained below.
- each $m_j^d$, $c_j^d$, $sig_j^d$ has correct length, and each $sig_j^d$ is a valid signature of $P_d$ on the tuple $(id, d, j, m_j^d, c_j^d)$, and $\mathcal{P}$ contains a proof that for each contributor $P_d$ the output of $P_d$'s garbled circuit for that round is indeed what the evaluator claims this output to be [8]. The following exceptions are allowed:
  1. if a garbled circuit or wire key needed for the evaluation of that garbled circuit from some party $P_j$ is missing and the corresponding message part could not be computed, the evaluator must prove that $P_j$ failed to post the garbled circuit or wire key and the deadline $\tau$ has passed. Recall that in our main construction we require CSaR secret identifiers to be published on the bulletin board (in order for the evaluator to know what secrets it must request from the CSaR). If $P_j$ failed to post the secret identifier, "proof of missing message" is used to prove that this message does not exist. If $P_j$ posted this identifier, but the corresponding

[8] The "proof" simply consists of the whole bulletin board. CSaR retrieves the garbled circuit of $P_j$ and the corresponding wire keys that were published by CSaR on the bulletin board, executes the garbled circuit and checks whether the output is consistent with the message posted by the evaluator.

message is not stored with CSaR, CSaR publicly returned $\perp$ upon evaluator's request to retrieve this message and the proof of this publication is used to prove that the message was not stored. In both cases, wire keys for the default value $\perp$ are released by the CSaR participants as wire keys corresponding to the output of the missing circuit.

2. If a $m_j^d$, $c_j^d$, or $sig_j^d$ has incorrect length, or $sig_j^d$ is not a valid signature of $P_d$ on the tuple $(id, d, j, m_j^d, c_j^d)$, but the evaluator proved that it is indeed the output of $P_d$'s garbled circuit, this still counts as an authoritative message. In this case, wire keys for the default value $\perp$ are released by the CSaR participants as wire keys corresponding to $m_j^d$ and $c_j^d$.

- The deadline $\tau$ has passed at the time of posting.
- This is the first published message that satisfies the requirements above.

Same as in our main construction, there are up to $n$ authoritative messages for the first round – one for each contributing party. Formally, an authoritative message of round $d = 1$ from party $P_k$ is a published message that satisfies the following conditions:

- Message is of the form $(id, 1, k, m_k^1, sig_k^1)$.
- $sig_k^1$ is a $P_k$'s correct signature over $m_k^1$.
- $m_k^1$ has correct length.
- The deadline $\tau$ has not passed at the time of posting.
- This is the first published message that satisfies the requirements above.

If a required authoritative first message from some party $P_j$ is missing, the evaluator must prove that $P_j$ failed to post this message and the deadline $\tau$ has passed ("proof of missing message"). In this case, wire keys for the default value $\perp$ are released by the CSaR participants as wire keys corresponding to that message.

Finally, note that same as in our main construction, we require that the original protocol $\pi$ has the publicly recoverable output property, now with the additional guarantee of output delivery. The publicly recoverable output property with guaranteed output delivery can be easily achieved as follows in a protocol which has guaranteed output delivery: first, all parties broadcast the output. Then, the value that was broadcasted by more than half of the parties is taken as the output. Note that if $\pi$ has guaranteed output delivery, each honest participant in $\pi$ is guaranteed to be able to correctly compute the honest output. Given honest majority among the participants (which we assume in order for $\pi$ to provide the guaranteed output delivery anyway), the protocol outlined above results in a correct output. In the following, for simplicity we assume that protocol $\pi$ has the publicly recoverable output property with guaranteed otput delivery and $\mathsf{Eval}$ denotes the algorithm

used to retrieve the output from the transcript.

The full construction is given in Protocols 9 and 10 (preparation of the garbled circuits and keys), as well as Protocol 11 (execution phase). Just as in our main construction, we show security by providing a simulator that does not have access to the honest parties' secrets and showing that no PPT adversary is able to distinguish the interaction with the simulator from the interaction with the honest parties. However, this time we additionally prove that the guaranteed output delivery property holds for our construction. We provide the formal proof in §2.7.

## 2.7 Proof of Security - GoD Construction

In order to prove security properties of our construction, we again construct the simulator $S$ using the CSaR simulator $S_{CSaR}$, the garbled circuit simulator $S_{GC}$, and the simulator of the original MPC protocol $S_{MPC}$.

$S$ starts by choosing the public and secret keys $(pk_j, sk_j)$, as well as the signature and verification keys $(sigk_j, verk_j)$ for the honest parties, and initializing an empty list $L$ which will be later used for the secret storage when simulating $\mathsf{Ideal}_{\mathrm{CSaR}}$. Then, $S$ starts the real-world adversary $\mathcal{A}$, and gives $\mathcal{A}$ the public and verification keys. Additionally, $S$ starts the CSaR simulator $S_{CSaR}$. $S$ also starts the simulator $S_{MPC}$ for MPC functionality $f$ and secure protocol $\pi$. Whenever $S_{MPC}$ sends a message to its ideal functionality, $S$ forwards this message to its own ideal functionality (and vice versa). Finally, $S$ observes the blockchain and aborts whenever an authoritative message is posted such that it is not consistent with $S$'s expectations based on the authoritative messages of the previous rounds.

First, note that the CSaR simulator $S_{CSaR}$ is running during the whole execution of $S$, and requires access to the ideal functionality $\mathsf{Ideal}_{\mathrm{CSaR}}$. We first explain how exactly $S$ is simulating the CSaR infrastructure, and then explain how $S$ is simulating each phase of the protocol. **Simulation of the CSaR infrastructure.**

- $S$ honestly simulates $\mathsf{Ideal}_{\mathrm{CSaR}}$ by keeping the list $L$ of (identifier, release condition, secret) tuples and honestly storing messages whenever such requests come from $S_{CSaR}$. Messages stored by the adversary are not only stored, but also released honestly. For an honest message that was not stored in $L$ the simulator $S$ decides on the fly whether and what it returns (acting as $\mathsf{Ideal}_{\mathrm{CSaR}}$) (see the description of the next two phases to understand what we mean by this). If $S$ is unable to provide a response to a valid

**Protocol 9** NON-INTERACTIVE MPC WITH GOD – *CircuitPreparationPhase*

1. $P_j$ computes the output $(m_j^1, s_j^1)$ of the first round of the MPC protocol for $F$. $P_j$ computes the signature $sig_j^1$ on the tuple $(id, 1, j, m_j^1)$ using its signing key $sigk_j$. $P_j$ posts $(id, 1, j, m_j^1, sig_j^1)$ on the bulletin board.
2. $P_j$ produces Yao garbled circuit $\{GC_j^i\}$ for each round $i > 1$ based on the circuit $C_j^i$ of the next-message function $f^i$ of the original MPC protocol $\pi$:

$$(\{\mathtt{lab}_j^{w,b,i}\}_{w\in\mathtt{inp}_\mathtt{j}^\mathtt{i},b\in\{0,1\}}) \leftarrow \mathtt{Gen}(1^\lambda, \mathtt{inp}_\mathtt{j}^\mathtt{i})$$

$$GC_j^i \leftarrow \mathtt{Garble}(C_j^i, (\{\mathtt{lab}_j^{w,b,i}\}_{w\in\mathtt{inp}_\mathtt{j}^\mathtt{i},b\in\{0,1\}})$$

   Here, $\mathtt{inp}_\mathtt{j}^\mathtt{i}$ is the length of the input to the circuit $C_j^i$. This circuit takes as input messages $\{m_k^{i-1}\}_{k=1}^n$ published by the parties in the previous round, and the encryption $c_j^{i-1}$ of the secret state passed by $P_j$ from the previous round. All of $P_j$'s keys, input and randomness are hardcoded in the circuit. The verification and public keys of other contributors are also hardcoded in the circuit. For the circuit of the second round, the secret state passed from the first round is hardcoded in the circuit as well. The circuit decrypts the secret state and executes the next message function of the current round:
   (a) Compute $s_j^{i-1} = Dec_{sk_j}(c_j^{i-1})$.
   (b) Obtain $(m_j^i, s_j^i)$ by executing $\tilde{f}(x_j, r_j^i, m^i, s_j^{i-1})$, where $m^i = m_1^{i-1} \parallel \cdots \parallel m_n^{i-1}$.
   (c) Compute the encryption of the state $c_j^i = Enc_{pk_j}(s_j^i)$.
   (d) Compute the signature $sig_j^i$ on the tuple $(id, i, j, m_j^i, c_j^i)$ using the signing key $sigk_j$.
   (e) Output $(m_j^i, c_j^i, sig_j^i)$.
3. $P_j$ securely stores garbled circuits $\{GC_j^i\}$ for all rounds $i > 1$ using CSaR-PR. The witness needed to decrypt the ciphertext of some round $i$ is a valid proof of publishing of all authoritative messages of round 1 and up to (and including) round $i-1$. If $\tau$ was reached and some party did not post its authoritative message of the first round, the witness does not need to include a proof of publishing of the message computed by the garbled circuits of this party. Instead, the witness needs to include a proof of missing message by the deadline $\tau$.

release request of an honest message, $S$ aborts.

**Simulation of the CircuitPreparationPhase.**

- In Step 1 of the *CircuitPreparationPhase*, for each honest party $P_j$, $S$ waits until it receives the message $m_j^1$ output by $S_{MPC}$ as the party $P_j$, honestly computes the

**Protocol 10** NON-INTERACTIVE MPC WITH GOD – *KeyPreparationPhase*

1. Securely store input wire keys $(\{\mathtt{lab}_j^{w,b,2}\}_{w\in\mathtt{inp}_j^2,b\in\{0,1\}})$ for the circuit of the second round using CSaR-PR. For each party $P_k$ whose first round message $m_k^1$ is needed for the computation, the witness required to decrypt the wire key corresponding to the $i$-th bit of the input $m_k^1$ being 0 (resp. 1) is a **valid proof of publishing** with respect to the genesis block of the following:
   (a) Each authoritative message of the first round is published. If a message is missing, the witness needs to include a proof of missing message by deadline $\tau$ instead of that message. For each missing message that is needed in the computation, wire keys for the default value $\perp$ are released.
   (b) $i$-th bit of $m_k^1$ is 0 (resp. 1).
2. Securely store input wire keys $(\{\mathtt{lab}_j^{w,b,d}\}_{w\in\mathtt{inp}_j^d,b\in\{0,1\}})$ for the circuit of the $d$-th $(d \geq 3)$ round using CSaR-PR. Say a message $m_j^{d-1}$ (resp., $c_j^{d-1}$) is needed for the computation. The witness needed to decrypt the wire key corresponding to the $i$-th bit of $m_j^{d-1}$ (resp., $c_j^{d-1}$) being 0 (resp. 1) is a **valid proof of publishing** with respect to the genesis block of the following:
   (a) All authoritative messages of round 1 up to and including round $d-1$ are published (subject to the constraint that $\tau$ is reached and some party did not post its authoritative message of the first round). Recall that an authoritative message is defined in a way that allows for missing or invalid partial messages (given a valid execution proof from the evaluator) – in those cases, for each missing message that is needed in the computation, wire keys for the default value $\perp$ are released.
   (b) $i$-th bit of $m_j^{d-1}$ (resp., $c_j^{d-1}$) is 0 (resp. 1).

signature $sigpub_j^1$ over $(id, 1, j, m_j^1)$, and posts $(id, 1, j, m_j^1, sigpub_j^1)$ on the bulletin board.

- In Step 2, to construct garbled circuits for round $i = 2$, $S$ waits until the deadline $\tau$ has passed and forwards each message $m_z^1$ which is part of $P_z$'s authoritative message of round 1 to $S_{MPC}$ as if coming from $P_z$ (forwarding $\perp$ whenever an authoritative message is missing).

  To construct garbled circuits for rounds $i > 2$, $S$ waits until all authoritative messages of the first $i-1$ rounds are published (either by the adversary, or, if applicable, by $S$ itself), and the deadline $\tau$ has passed. Then, $S$ forwards each message $m_z^{i-1}$ that represents the public message of a corrupted party $P_z$ in the authoritative message of round $i-1$ to $S_{MPC}$ as if coming from $P_z$, forwarding $\perp$ whenever the authoritative

**Protocol 11** NON-INTERACTIVE MPC WITH GOD – *ExecutionPhase*

1. Wait until either deadline $\tau$ has passed.
2. The evaluator $E$ uses messages $(id, 1, z, m_z^1, sigpub_z^1)$ posted on the bulletin board by each party $P_z$ as the proof of publishing to get the garbled circuits (and keys) for the second round stored in CSaR by each participant in $\pi$. Then, $E$ computes the outputs $(m_j^2, sigpub_j^2, c_j^2, sigpr_j^2)$ of the second round by executing the garbled circuits. If for a party $P_j$ any part of the information required to compute the output is missing, output $\perp$ is used in the following.
3. Check whether an authoritative message was published for round 2. If yes, check if this message is consistent with own output and if so, simply use its proof of publish as the witness to decrypt the wire keys of the next round. If the message is not consistent, abort. If the authoritative message is not published yet, publish $(id, 2, m_1^2 \| \cdots \| m_n^2 \| c_1^2 \| \cdots \| c_n^2 \| sig_1^2 \| \cdots \| sig_n^2)$ (appending the proof of execution, as well as proofs of of missing/invalid messages if necessary) and use the proof of publish as the witness.
4. In each following round $d \geq 3$, $E$ executes each garbled circuit published by party $P_z$ for round $d-1$. Then, $E$ checks whether the authoritative message was published for that round and whether this message is consistent with own output and if so, simply uses its proof of publish as the witness to decrypt the wire keys of the next round. If the message is not consistent, $E$ aborts. If the authoritative message is not published yet, $E$ publishes the concatenated output of the garbled circuits along with the proof of execution. In any case, $E$ uses the proof of publish of the authoritative message to release the wire keys and the garbled circuits of the next round.
5. Whenever any needed wire key and/or garbled circuit was missing, $E$ additionally supplies a proof of missing message to decrypt the default wire keys of the next round.
6. Let $\tau'$ denote the resulting transcript of execution of $\pi$. $E$ outputs $\mathsf{Eval}(\tau')$ as the result.

message contained the corresponding proof of missing message or the party's message was invalid (had an invalid signature or length etc).

To construct the garbled circuit of an honest party $P_j$ in round $i$, $S$ uses the garbled circuit simulator $S_{GC}$ on the input $(m_j^i, c_j^i, sig_j^i)$. Here, $m_j^i$ is the output of $S_{MPC}$ for party $P_j$ in round $i$, $c_j^i$ the encryption of a zero string, and $sipr_j^i$ the signature honestly computed by $S$ over $(id, i, j, m_j^i, c_j^i)$.

- When an honest party is supposed to store a garbled circuit as specified by Step 3, $S$ simulates the SecretStorage step of $\mathsf{Ideal}_{\mathrm{CSaR}}$ by informing $S_{CSaR}$ that a secret has been stored with the release condition as specified by Step 3 of the *CircuitPreparationPhase*.

Once a valid release request has been submitted, $S$ checks whether it was able to construct a garbled circuit according to the procedure outlined in the simulation of Step 2, and if so, honestly simulates $\mathsf{Ideal}_{\mathrm{CSaR}}$ by storing the constructed garbled circuit in the list $L$ and then honestly releasing it. Otherwise, $S$ aborts.

**Simulation of the KeyStoragePhase.**

- For all wire keys that must be stored according to Step 1 and Step 2 of the *KeyStoragePhase*, $S$ simulates the SecretStorage step $\mathsf{Ideal}_{\mathrm{CSaR}}$ by informing $S_{CSaR}$ that a secret has been stored with the release condition as specified by Step 1 and Step 2 of the *KeyStoragePhase*. Once a valid release request has been submitted for a wire key, $S$ checks whether it was able to create the requested garbled key according to the procedure outlined in Step 2 of the simulation of the CircuitPreparationPhase, and if so, $S$ honestly simulates $\mathsf{Ideal}_{\mathrm{CSaR}}$ by storing this garbled wire key in the list $L$ and then honestly releasing it. Otherwise, $S$ aborts.

**Simulation of the ExecutionPhase.**

- Note that the evaluator does not possess any secrets. Thus, $S$ simply follows the procedure outlined in Protocol 11, sending its CSaR release requests to $S_{CSaR}$.

Now, we prove that no PPT adversary $\mathcal{A}$ is able to distinguish the view of interaction with the simulator $S$ constructed above from the view of interacting with honest parties in the real world. We start by having the the simulator control the honest parties and honestly follow the protocol, and make gradual changes in order to achieve the simulator described above. We denote the advantage of the adversary in distinguishing the between the $\mathbf{Hybrid}_{i-1}$ and $\mathbf{Hybrid}_i$ by $\epsilon_i$. We define the following hybrids (the detailed proofs for the indistinguishability between the neighboring hybrids are the same as for the corresponding hybrids in our main construction):

**$\mathbf{Hybrid}_0$:** This hybrid corresponds to the execution in the real world. The simulator $S$ controls all honest parties and follows the protocol.

**$\mathbf{Hybrid}_1$:** The simulator $S$ behaves the same as in the previous protocol, except that it switches from honestly executing the CSaR protocol to using the CSaR simulator $S_{CSaR}$ while honestly simulating $\mathsf{Ideal}_{\mathrm{CSaR}}$ by itself. In more detail, $S$ passes messages to and from $S_{CSaR}$ and the dishonest parties, and simulates $\mathsf{Ideal}_{\mathrm{CSaR}}$ by storing a list $L$ of (identifier, release condition, secret) tuples as follows:

- Upon receiving a secret $s$ and a release condition $F$ from $S_{CSaR}$, the simulator stores $(id, F, s)$, where $id$ is the identifier of the CSaR request.

- Whenever an honest message needs to be stored, $S$ honestly stores it in $L$ and notifies $S_{CSaR}$.
- Whenever $S_{CSaR}$ queries $\mathsf{Ideal}_{\mathrm{CSaR}}$ for a secret with the identifier $id$, the simulator $S$ checks whether the entry with the identifier $id$ exists, and if so, whether the given witness satisfies the release condition of this entry. If so, the simulator looks up the list $L$ of stored tuples and returns the corresponding secret $s$ to $S_{CSaR}$ if an entry with the identifier $id$ is in the list.

**Lemma 12.** *For the hybrids* $\boldsymbol{Hybrid_1}$ *and* $\boldsymbol{Hybrid_0}$ *holds* $\epsilon_1 \leq negl_1(n)$ *by the security of the CSaR construction.*

**Hybrid$_2$:** The simulator behaves the same as in the previous round, except that it changes the way $\mathsf{Ideal}_{\mathrm{CSaR}}$ is simulated. Specifically, instead of saving the honest parties' wire keys and garbled circuits in list $L$ at the time specified by the protocol, the simulator stores each secret in $L$ only when $S_{CSaR}$ asks for this secret and is able to provide a valid witness for the corresponding release condition.

**Lemma 13.** *For the hybrids* $\boldsymbol{Hybrid_2}$ *and* $\boldsymbol{Hybrid_1}$ *holds:* $\epsilon_2 = 0$ *(nothing changed between the hybrids).*

**Hybrid$_3$:** The simulator $S$ behaves the same as in the previous round, except that it aborts if an authoritative message for round $i > 1$ is submitted by the adversary which is inconsistent with $S$'s own view of the bulletin board: corresponds to bit $1 - b_p$ at some position $p$ in the string, when based on $S$'s view the authoritative message contains $b_p$ at this position, or corresponds to some position which does not have a recorded authoritative message on the bulletin board yet, or contains a valid proof of missing message for a message that is present on the bulletin board, or contains a proof of publication of garbled circuits/wire keys different from those released by the CSaR according to $S$'s view. We denote this by $\mathsf{abort}_1$.

**Lemma 14.** *For the hybrids* $\boldsymbol{Hybrid_3}$ *and* $\boldsymbol{Hybrid_2}$ *holds:* $\epsilon_3 \leq negl_3(n)$ *by the unforgeability of the proof of publish.*

**Hybrid$_4$:** The simulator $S$ behaves the same as in the previous hybrid, except that it aborts if some honest party's authoritative message for the first round is not the same as expected by the simulator. We denote this by $\mathsf{abort}_2$.

**Lemma 15.** *For the hybrids* $\boldsymbol{Hybrid_4}$ *and* $\boldsymbol{Hybrid_3}$ *holds:* $\epsilon_4 \leq negl_4(n)$

*Proof.* Note that honest parties always publish all required messages, and according to the previous hybrid, the authoritative messages published by the adversary are consistent with $S$'s view of the bulletin board. By the definition of an authoritative message each first

message of a party must contain a valid signature. Thus, if the adversary is able to publish an authoritative message of the first round of some honest party such that is not the same as expected by the simulator, it means that the adversary is also able to forge a signature of that honest party. We can thus use this adversary to construct an adversary on the unforgeability of the signature scheme that we use. □

**Hybrid$_5$:** The simulator $S$ behaves the same as in the previous hybrid, except that it aborts if the adversary publishes an authoritative message for some round $i > 1$ that is not the same as expected by the simulator. We denote this by $\mathsf{abort}_3$.

**Lemma 16.** *For the hybrids* ***Hybrid$_5$*** *and* ***Hybrid$_4$*** *holds:* $\epsilon_5 = 0$

*Proof.* Note that at this point the authoritative messages published by the adversary are consistent with $S$'s view of the bulletin board. Since the authoritative message in particular contains pointers to the garbled circuits/wire keys for the previous round which must explain the output claimed by the evaluator, the output claimed by the evaluator must be the same as expected by the simulator. □

**Hybrid$_6$:** The simulator $S$ behaves the same as in the previous hybrid, except that it now uses the garbled circuit simulator $S_{GC}$ instead of honestly constructing the garbled circuit. Specifically, once the authoritative message for round $i-1$ is published (and the deadline $\tau$ has passed), the simulator $S$ honestly computes the output of the garbled circuit (consisting of the public message, the encrypted state, and the signatures) of an honest party $P_j$ in round $i$ using the authoritative message from round $i-1$, as well as honest public and secret keys, signature and verification keys, and (for the garbled circuit of the second round) the honest parties' state from the first round. Whenever a (part of an) authoritative message (needed as part of the input) is shown missing or shown to be invalid, the default message $\perp$ is used instead. Denote the output of an honest party $P_j$'s garbled circuit of round $i$ by $out_i^j$. Then, use the garbled circuit simulator to construct the circuit for round $i$ of $P_j$ as follows: $gc_i^j = S_{GC}(out_i^j)$ and use the result instead of the actual garbled circuit.

**Lemma 17.** *For the hybrids* ***Hybrid$_6$*** *and* ***Hybrid$_5$*** *holds:* $\epsilon_6 \leq negl_6(n)$ *by the security of the garbled circuits construction.*

**Hybrid$_7$:** The simulator behaves the same as in the previous hybrid, except that the simulation of the garbled circuits is done a bit different. Specifically, we change the input we provide to the garbled circuit simulator $S_{GC}$: instead of using an encryption of the state that was published by the adversary in the previous round, we use an encryption of zeroes (the signature is computed using this encryption of zeroes as well).

**Lemma 18.** *For the hybrids* ***Hybrid$_7$*** *and* ***Hybrid$_6$*** *holds:* $\epsilon_7 \leq negl_7(n)$ *by the security of the encryption scheme.*

**Hybrid$_8$:** The simulator behaves the same as in the previous hybrid, except that instead of using the honest parties' inputs, it relies on the simulator $S_{MPC}$ of the original MPC protocol $\pi$ to retrieve the messages used in the construction of the garbled circuits.

**Lemma 19.** *For the hybrids* ***Hybrid$_8$*** *and* ***Hybrid$_7$*** *holds:* $\epsilon_8 \leq negl_8(n)$ *by the security of the original MPC protocol* $\pi$*.*

Note that in the last hybrid, the simulator does not need the honest parties' inputs to simulate the execution, and that all of the simulator's aborts happen only with a negligible probability. Additionally, note that from the proofs above follows that the authoritative messages are consistent with what an honest evaluator would have output (thus in particular, up to some negligible probability, an honest evaluator does not need to abort), and that the messages forwarded to the ideal functionality are consistent with the authoritative messages. Finally, note that it is given that the original protocol has the publicly recoverable output with guaranteed output delivery property. Thus, up to some negligible probability, the output of an honest evaluator is guaranteed to exist and is the same both in the real and in the ideal world. Thus, our protocol securely computes $f$ in the presence of contributors and evaluators with guaranteed output delivery for the evaluators, as required.

## 2.8 Optimizations

Our next goal is to minimize the number of CSaR invocations in our construction. For this, we will focus on our main construction (Protocols 6, 7 and 8), but the optimizations are applicable to our guaranteed output delivery construction (which will be introduced later) as well.

Let $n$ denote the number of parties participating in the original MPC protocol $\pi$, $n_{rounds}$ denote the number of rounds in $\pi$, $n^i_{wires,j}$ denote the number of input wires of a garbled circuit of the next-message function for round $i$ of party $P_j$.

Then, the number of CSaR secret store operations is upper bounded by:

$$N_{store} = n * (n_{rounds} - 1) + \sum_{i=2}^{n_{rounds}} \sum_{j=1}^{n} 2 * n^i_{wires,j}$$

The term $n * (n_{rounds} - 1)$ is due to the fact that each party needs to store a garbled circuit for each round, except for the very first one. The term $\sum_{i=2}^{n_{rounds}} \sum_{i=1}^{n} 2 * n^i_{wires,j}$ is

added because each party also needs to store two wire keys for each input wire of each garbled circuit it publishes.

The number of CSaR secret release operations for each evaluator is upper bounded by:

$$N_{release} = n * (n_{rounds} - 1) + \sum_{i=2}^{n_{rounds}} \sum_{j=1}^{n} n^i_{wires,j}$$

This is because the evaluator needs all of the garbled circuits, as well as a single wire key for each input wire of each garbled circuit, to perform the computation.

Note that the dominant factor in both of the equations is $\sum_{i=2}^{n_{rounds}} \sum_{j=1}^{n} n^i_{wires,j}$. This term is precisely the combined communication and (encrypted) state complexity of the original MPC protocol $\pi$, minus the messages of the first round and plus the signatures on the public messages and the state. Thus, in order to minimize the number of CSaR invocations, we must first and foremost optimize the combined communication and state complexity of the original MPC scheme. We discuss a possible way to do this in the next section.

## 2.9 Optimizing Communication and State Complexity in MPC

Our goal in this section is to design an MPC protocol in the plain model such that its combined communication and state complexity is independent of the function that it is computing. While a number of works have focused on optimizing communication complexity, we are not aware of any construction optimizing both the communication and state complexity.

We achieve it in two steps, starting with a protocol secure against *semi-malicious* adversaries. Semi-malicious security, introduced by Asharov et al [8], intuitively means that the adversary must follow the protocol, but can choose its random coins in an arbitrary way. The adversary is assumed to have a special witness-tape and is required to write a pair of input and randomness $(x, r)$ that explains its behavior. We specifically start with a semi-malicious MPC protocol that has attractive communication and state complexity (i.e., independent of the function being computed). Then, we extend it so that the resulting construction is secure against not only semi-malicious, but also fully malicious adversaries.

### 2.9.1 Step. 1: MPC with semi-malicious security

Our starting point is the solution proposed in the work of Brakerski et al. [29] based on multi-key fully homomorphic encryption (MFHE) that achieves semi-malicious security [9]. The construction is for deterministic functionalities where all the parties receive the same output, however it can be easily extended using standard techniques to randomized functionalities with individual outputs for different parties [8]. For technical details behind the construction and the security proof we refer to Brakerski et al.

We note that while Brakerski et al. do not explicitly explain how to handle circuits of arbitrary depth, the *bootstrapping* approach outlined by Mukherjee and Wichs [91] can be used here. Informally, the bootstrapping is done as follows: each party encrypts their secret key bit-by-bit using their public key and broadcasts the resulting ciphertext. These ciphertexts are used to evaluate the decryption circuit, thus reducing the noise. To do so, the parameters of the MFHE scheme must be set in a way that allows it to handle the evaluation of the decryption circuit. We assume circular security that ensures that it is secure to encrypt a secret key under its corresponding public key and refer to Mukherjee and Wichs [91] for details.

To summarize, the construction in Protocol 12 is an MPC protocol secure against semi-malicious adversaries and can handle functions of arbitrary depth [10].

The communication complexity in Protocol 12 depends only on the security parameters, the number of parties, and input and output sizes [29]. Note that for a party $P_k$ the state that is passed between the rounds in Protocol 12 consists of the following data:

- $\mathsf{params}_k$ (passed from round one to round two and round three)
- $\mathsf{params}$, $(\mathsf{pk}_\mathsf{k}, \mathsf{sk}_\mathsf{k})$, $\{c_{k,j}\}_{j\in[l_{in}]}$, $\{\tilde{c}_{k,j}\}_{j\in[l_{key}]}$ (passed from round two to round three)
- $\{ev_{k,j}\}_{j\in l_{out}}$ (passed from round three to round four)

Note that this data depends only on security parameters, number of parties, and input and output sizes. Thus, the communication and state complexity of the semi-malicious protocol

[9] Their scheme is secure when exactly all but one parties are corrupted. To transform it into a scheme that is secure against any number of corruptions, Brakerski et al. suggest to extend it by a protocol proposed by Mukherjee and Wichs (Section 6.2 in [91]) that relies on a so-called *extended function*. For simplicity, we skip this technical detail in our protocol. We note, however, that the additional communication and state complexity incurred due to the transformation depend only on the security parameter, as well as the parties' input and output sizes.

[10] Again, this construction is secure against exactly $N-1$ corruptions (where $N$ is the total number of parties). When used with the extended function transformation by Mukherjee and Wichs (which we skip here for readability purposes), the construction becomes secure against arbitrary many corruptions.

does not depend on the circuit we are computing.

**Protocol 12** Optimizing MPC

1. Let $P_k$ be the party executing this protocol.
2. Run $\mathsf{params}_k \leftarrow \mathsf{MFHE.DistSetup}(1^\kappa, 1^N, k)$. Broadcast $\mathsf{params}_k$.
3. Set $\mathsf{params} = (\mathsf{params}_1, \ldots, \mathsf{params}_N)$, and do the following:
   - Generate a key-pair $(pk_k, sk_k) \leftarrow \mathsf{MFHE.Keygen}(\mathsf{params}, k)$
   - Let $l_{in}$ denote the length of the party's input. Let $x_k[j]$ denote the $j$-th bit of $P_k$'s input $x_k$. Let $l_{key}$ denote the length of the party's secret key.
   - Encrypt the input bit-by-bit:

     $$\{c_{k,j} \leftarrow \mathsf{MFHE.Encrypt}(pk_k, \mathsf{x}_k[j])_{j\in[l_{in}]}$$

   - Encrypt the secret key bit-by-bit:

     $$\{\tilde{c}_{k,j} \leftarrow \mathsf{MFHE.Encrypt}(pk_k, sk_k[j])_{j\in[l_{key}]}$$

   - Broadcast the public key and the ciphertexts $(pk_k, \{c_{k,j}\}_{j\in[l_{in}]}, \{\tilde{c}_{k,j}\}_{j\in[l_{key}]})$
4. On receiving values $\{pk_i, c_{i,j}\}_{i\in[N]\setminus\{k\},j\in[l_{in}]}$ execute the following steps:
   - Let $f_j$ be the boolean function for $j$-th bit of the output of $f$. Let $l_{out}$ denote the length of the output of $f$.
   - Run the evaluation algorithm to generate the evaluated ciphertext bit-by-bit:

     $$\{c_j \leftarrow \mathsf{MFHE.Eval}(\mathsf{params}, f_j, (c_{1,1}, \ldots, c_{N,l_{in}}))\}_{j\in[l_{out}]},$$

     while performing a bootstrapping (using the previously broadcasted encryptions of the secret keys) whenever needed.
   - Compute the partial decryption for all $j \in [l_{out}]$ :

     $$ev_{k,j} \leftarrow \mathsf{MFHE.PartDec}(sk_k, c_j)$$

   - Broadcasts the values $\{ev_{k,j}\}_{j\in l_{out}}$
5. On receiving all the values $\{ev_{i,j}\}_{i\in[N],j\in[l_{out}]}$ run the final decryption to obtain the $j$-th output bit: $\{y_j \leftarrow \mathsf{MFHE.FinDec}(ev_{1,j}, \ldots, ev_{N,j}, c_j)\}_{j\in[l_{out}]}$. Output $y = y_1 \ldots y_{l_{out}}$.

### 2.9.2 Step. 2: MPC with fully malicious security

In order to protect from fully malicious adversaries, we extend the construction above with the zero-knowledge protocol proposed by Kilian [82]. In the following, we first elaborate on

Kilian's protocol and some changes we need to make to it in order to keep the combined communication and state complexity low. Then, we elaborate on how Kilian's protocol is used in the overall MPC construction.

#### 2.9.2.1 Kilian's zero-knowledge protocol

Kilian's construction [82] relies on probabilistically checkable proofs (PCPs) and allows a party $P$ to prove the correctness of some statement $x$ using a witness $w$ to the prover $V$. We specifically chose Kilian's construction because of its attractive communication and state complexities. Note that we make a minor change to Kilian's construction (Protocol 13) – instead of storing the PCP string that was computed in round two to use it in round four (as is done in the Kilian's original scheme), $P$ recomputes the string (using the same randomness) in round four. Clearly, this changes nothing in terms of correctness and security. However, it allows us to drastically cut the state complexity of Kilian's original construction since the storage of the PCP becomes unnecessary.

**Protocol 13** Optimizing MPC - Kilian's construction

1. Verifier $V$ chooses a collision-resistant hash function $h$ and sends its description to the prover $P$.
2. Prover $P$ uses the PCP prover $P'$ to construct a PCP string $\psi \leftarrow P(x, w)$. Denote by $r_p$ the randomness used by the prover in the generation of $\psi$. $P$ computes the root of the Merkle tree (using the hash function $h$) on $\psi$, and sends the commitment to the Merkle tree root to the verifier $V$.
3. $V$ chooses a randomness $r_v$ and sends it to $P$.
4. $P$ recomputes the PCP string $\psi \leftarrow P(x, w)$ using the randomness $r_p$ and sends PCP answers to the set of queries generated according to the PCP verifier $V'$ (executed on randomness $r_v$) to $V$.
5. $V$ checks the validity of the answers, and accepts if all answers are valid and consistent with the previously received Merkle tree root. Otherwise, $V$ outputs $\perp$.

#### 2.9.2.2 Full construction

The MPC construction secure against fully malicious adversaries is effectively the same as the semi-malicious one, except that additionally the parties commit to their input and randomness in the semi-malicious protocol and prove (using any zero-knowledge argument of knowledge, denoted by $\mathsf{ZKAoK}$ in the following) that they know the opening to the

commitment. Kilian's construction is executed by each party $P_k$ after each of the first three rounds of Protocol 12. In more detail:

We assume that there exists some ordering of parties participating in Protocol 12. Following the approach outlined by Asharov et al. [8], in each round $d$ of Protocol 12 we use Kilian's construction as follows:

For each pair of parties $(P_i, P_j)$, $P_i$ acts as a prover to the verifier $P_j$ in order to prove the statement

$$\mathsf{NextMessage}_d(x_i, r_i, \{m^k\}_{k=1}^d) = m_i^d, com(x_i||r_i, r_i') = c_i$$

Here, NextMessage is the function executed by $P_i$ in this round according to Protocol 12, $x_i$ is the secret input of $P_i$, $r_i$ is the randomness used by $P_i$ in the semi-malicious construction, $\{m_k\}_{k=1}^d$ are (concatenations of) the messages broadcast by all parties participating in Protocol 12 in rounds 1 to $d$, $m_i^d$ is the message broadcast by $P_i$ in round $d$, and $c_i$ is the commitment broadcast by $P_i$ in the first round ($com(x, r)$ denotes a perfectly binding, computationally hiding commitment to value $x$ using randomness $r$). If a check fails, $P_j$ broadcasts $\perp$ and aborts. These proofs are done sequentially (starting a new one only after the previous is fully finished), following the ordering of the (pairs of) parties. If at least one party has broadcasted $\perp$, all parties abort.

### 2.9.3 Properties of the resulting MPC construction

We now discuss the properties of the scheme constructed above. Specifically, we show the following:

**Theorem 5.** *Let $f$ be an $N$-party function. Protocol 14 is an MPC protocol computing $f$ in the plain (authenticated broadcast) model which is secure against fully malicious adversaries corrupting up to $t < N$ parties. Its communication and state complexity depend only on security parameters, number of parties, and input and output sizes. In particular, the complexity is independent of the function $f$.*

**Security.** We outline why this construction is secure. Intuitively, in order to prove security we construct the simulator $S$ as follows: $S$ commits to 0 for each honest party, and uses a zero-knowledge argument of knowledge simulator to prove that it knows the opening to the commitment. Then, $S$ uses an extractor $Ext$ of the argument of knowledge construction to retrieve the input and randomness $x_i$, $r_i$, $r_i'$ of each corrupted party $P_i$'s valid proof. Then, in each round $S$ uses the simulator $S_{sm}$ of the semi-malicious scheme to retrieve the honest parties' messages, while forwarding messages broadcasted by any adversarial

**Protocol 14** Optimizing MPC - handling fully malicious adversaries

1. Let $P_z$ denote the party executing this protocol.
2. Let $\mathsf{NextMessage}_d(\cdot)$ denote the next message function of Protocol 12.
3. Compute and broadcast $c_z = com(x_z||r_z, r'_z)$.
4. Sequentially, for each ordered pair of parties $(P_i, P_j)$:
   (a) If $P_i = P_z$: Act as a prover in a $\mathsf{ZKAoK}$ to prove knowledge of $x_z||r_z$, $r'_z$ such that $c_z = com(x_z||r_z, r'_z)$.
   (b) If $P_j = P_z$: act as verifier in a $\mathsf{ZKAoK}$ to check knowledge of $x_i||r_i$, $r'_i$ such that $c_i = com(x_i||r_i, r'_i)$. If this check fails, broadcast $\perp$.
5. If any party party broadcast $\perp$, abort.
6. For each round $d = 1, \ldots, 3$
   (a) Let $m^d = m_1^{d-1}, \ldots, m_n^{d-1}$.
   (b) Compute $\mathsf{NextMessage}_d(x_z, r_z, \{m^k\}_{k=1}^d) = m_z^d$.
   (c) Broadcast $m_z^d$.
   (d) Sequentially, for each ordered pair of parties $(P_i, P_j)$:
      i. If $P_i = P_z$, $P_z$ acts as a Prover in Protocol 13 and uses the witness $(x_z, r_z, c_z^{d-1}, r'_z)$ to prove that the following holds:

$$\mathsf{NextMessage}_d(x_z, r_z, \{m^k\}_{k=1}^d) = m_z^d, com(x_z||r_z, r'_z) = c_z$$

      ii. If $P_j = P_z$, $P_z$ acts as a Verifier in Protocol 13 to verify that there exist $(x_i, r_i, c_i^{d-1}, r'_i)$ such that the following holds:

$$\mathsf{NextMessage}_d(x_i, r_i, \{m^k\}_{k=1}^d) = m_i^d, com(x_i||r_i, r'_i) = c_i$$

      If this verification check fails, broadcast $\perp$ and abort.
   (e) If any party party broadcast $\perp$, abort.
7. Output $\mathsf{NextMessage}_4(x_z, r_z, \{m^k\}_{k=1}^4, c_z^3) = m_z^d$.

party $P_i$ to $S_{sm}$ (aborting whenever $\mathsf{NextMessage}_d(x_i, r_i, \{m^k\}_{k=1}^d, s_i^{d-1}) \neq m_i^d$ but the proof supplied by the adversary goes through, and writing witnesses $(x_i, r_i)$ extracted by $Ext$ on the witness tape of $P_i$ otherwise). $S$ uses the zero-knowledge simulator $S_{zk}$ of Kilian's protocol to simulate proofs on behalf of the honest parties. $S$ honestly checks the proofs submitted by the adversary, aborting (according to the protocol) whenever a proof is invalid.

**Communication and State Complexity Analysis.**

As we mentioned above, the communication complexity of Protocol 12 depends only on security parameters, number of parties, and input and output sizes. In particular, the

communication and state complexity of the semi-malicious protocol does not depend on the circuit we are computing.

The communication complexity of Kilian's protocol depends on the security parameter as well as the length of the statement. In our case, the statement consists of the messages sent by the parties participating in the semi-malicious MPC protocol in the previous round as well as the message output by the party in the current round. Since the communication complexity of the semi-malicious MPC protocol is independent of the function being computed, the communication complexity of the overall construction is also independent of the function being computed. As for the state complexity, recall that we made a minor change to Kilian's original protocol – instead of storing the PCP, the prover simply recomputes (using the same randomness) it whenever it is needed. Due to this simple modification the PCP string does not contribute to the state complexity. The only other things contributing to the state complexity is the hash function $h$ and the randomness $r_v$, both independent of the function being computed by the MPC [11].

The combined communication and state complexity added due to the broadcasted commitments as well as ZKAoK proofs about these commitments also depends only on security parameters, number of parties, and input and output sizes.

Thus, we have shown that the communication and state complexity of our construction in Protocol 14 is independent of the function the MPC protocol is tasked with computing.

**Integrating communication and state optimized MPC.** As we showed in Section 2.8, the number of CSaR secret store operations in our non-interactive MPC construction (Protocols 6, 7 and 8) is upper bounded by:

$$N_{store} = n * (n_{rounds} - 1) + \sum_{i=2}^{n_{rounds}} \sum_{j=1}^{n} 2 * n^i_{wires,j}$$

The number of CSaR secret release operations for each evaluator is upper bounded by:

$$N_{release} = n * (n_{rounds} - 1) + \sum_{i=2}^{n_{rounds}} \sum_{j=1}^{n} n^i_{wires,j}$$

As we pointed out in Section 2.8, the term $\sum_{i=2}^{n_{rounds}} \sum_{j=1}^{n} n^i_{wires,j}$ is precisely the combined communication and (encrypted) state complexity of the underlying MPC protocol $\pi$, minus the messages of the first round and plus signatures on the public messages and the state.

[11] Additionally, they can be chosen by $V$ independently of any messages from $P$, and thus they can be hardcoded in the garbled circuits and do not add to the state complexity of the non-interactive construction.

Thus, when using Protocol 14 as the underlying protocol $\pi$ in our main non-interactive MPC construction (Protocols 6, 7 and 8), we obtain a construction which number of CSaR store and release operations depends only on the number of rounds in $\pi$, security parameters, number of parties, and input and output sizes. All of these parameters are independent of the function that $\pi$ is tasked with computing.

Thus, we get the following result:

**Corollary 3.** *There exists an MPC protocol $\pi'$ in the blockchain model that has adversarial threshold $t < N$, provides security with abort against fully-malicious adversaries and does not require participants to be online at the same time. Only a single message is required from the MPC contributors (the evaluators might be required to produce multiple messages). Furthermore, the number of calls to CSaR of this protocol is independent of the function that is being computed using this MPC protocol.*

## 2.10 A Note on Statelessness

In prior sections, for simplicity we described our protocols as having a single evaluator $E$. However, we note that in our constructions the evaluators possess no secret information, and in fact, as we explain below, it is sufficient for each evaluator who wishes to contribute to participate only for a single round.

For simplicity, in the following we assume that CSaR-PR is used as an ideal functionality [12]. Consider one round of the underlying MPC protocol in our GoD construction. To execute it, in one round an evaluator (let us denote it by $E_1$), needs to read the bulletin board and send a CSaR request for the garbled keys and circuits for the next round of MPC. Then, the evaluator (denoted by $E_2$), obtains the garbled circuits (along with the corresponding keys) that were published by CSaR-PR in response to $E_1$'s request, executes the circuits and publishes the output on the bulletin board. Note that no state is passed from $E_1$ by $E_2$. In fact, $E_1$ and $E_2$ can be different parties, and it is sufficient that each evaluator sends only a single message.

Similar reasoning applies to our construction with abort, except that because it uses the CSaR system with private release, CSaR returns the requested data to the client $E_1$. Thus, technically, in our construction with abort, each evaluator must participate for two rounds (even though no state needs to be saved between the first and the second round).

[12]The statelessness of eWEB, which we use to instantiate the CSaR in this work, depends on the particular DPSS instantiation.

However, by using CSaR-PR instead of CSaR, just as in our GoD construction, we can reduce the number of participation rounds to a single one.

# Chapter 3

# Towards Scalable YOSO MPC via Packed Secret-Sharing

*It's no use going back to yesterday, because I was a different person then.*

Lewis Carroll, Alice in Wonderland

In the previous chapter we moved one step closer towards obtaining distributed cryptographic protocols as a service by designing a stateless MPC protocol under the CSaR assumption. In the next part of our thesis we continue exploring stateless MPC and remove the assumption of CSaRs. Towards this, we focus on YOSO, short for *You Only Speak Once*, which is a paradigm that is tailored to stateless large-scale environments such as blockchains. While this paradigm has been gaining a lot of traction in the cryptographic community, it is still fairly new, and prior to our work the most asymptotically efficient YOSO MPC protocol had communication complexity per gate that was linear in the size of the MPC committee. By using preprocessing, we can improve the online communication complexity to $O(1)$ broadcast bits per gate under the assumption that the circuit's width is $O(n)$. The total offline communication is $O(n|C|) + O(n^2)$, and the total online communication is $O(|C|)$, where $|C|$ denotes the number of multiplication gates in the circuit $C$ that we aim to compute.

The work presented in this chapter is a joint work with Daniel Escudero and Antigoni Polychroniadou. As the main author, I designed our MPC protocol and proved it secure.

## 3.1 Introduction

In secure multiparty computation security holds even if $t$ out of the $n$ parties are corrupted and collude, and typically, the smaller $t$ is with respect to $n$, the more efficiency and other benefits are gained. However, the smaller $n$ is, the more reasonable it is for an adversary to corrupt a large fraction of the parties. Hence, low ratios such as $t/n < 1/2$, also known as the *honest majority* setting, are a more reasonable assumption when $n$ is somewhat large. Unfortunately, MPC is made of very communication-intensive interactive protocols, and this is particularly critical in a context where a large number of parties are involved. This situation quickly led to adopting the notion of *committees*, already common in distributed computing. Instead of a large pool of parties performing an MPC protocol among each other, a subset of the parties—a *committee*—is sampled at random, and this smaller set is the one in charge of executing the MPC protocol on behalf of the larger set. The intuition is that, by setting parameters properly, if the large set of $N$ parties has $T$ corruptions, then with high probability the smaller set of $n$ parties will have $t$ corruptions, where the ratio $t/n$ is only *slightly larger* than the original $T/N$.

Committee-based MPC is a common approach for handling cases with a large number of parties (see *e.g.* [7]). However, it suffers from an inherent drawback: it is insecure when considering an adversary who can corrupt parties *after* the committee has been sampled, which is a property known as *adaptive security*. For example, if $N = 1000$, we assume the adversary can corrupt $T = 400$ parties, and we sample committees of size $n = 400$, an adaptive adversary can choose to corrupt *all* of the parties of a selected committee once it learns their identities. As a potential solution to this problem, Gentry et al. proposed the YOSO model [55]. It assumes a large pool of parties, who can be used to handle computation tasks. Just as in the traditional committee-based MPC, instead of performing computations using the whole available pool, the computation is done by a number of committees, with the sizes of the committees being significantly smaller than the total number of parties. However, recall that in YOSO, the computation is done via committees of stateless roles, which change from one round to the next, keeping the communication pattern dynamic. If the YOSO role-assignment mechanism is secure, the information on who is selected to execute the roles of the next committee is only known to the parties who have been sampled for that committee. As further every party in every committee only communicates once, assuming they erase state after the execution, corrupting them after they sent the one message is of no use to the adversary, since they don't hold any secrets and are as likely to be chosen to participate in a future committee as any other party. Thus, YOSO MPC protocols can withstand even highly powerful adversaries, for example an

adversary which is performing an adaptive Denial of Service attack. Simultaneously, the committees which are performing the computation are much smaller than the overall pool of physical machines. Hence, this enables MPC protocols with communication complexity that scales sublinearly with the total number of parties.

### 3.1.1 The Efficiency of YOSO MPC.

Since its inception, many works have expanded the limits of the YOSO model, *cf.* [22, 37, 83, 97]. Very interesting results exist, and we survey some of them in detail in Section 3.1.3. However, we note a common trend shared by all YOSO MPC protocols to date: their total communication complexity grows as the size of the committees increases. This is particularly harmful for YOSO, where the committee sizes are considered very large. To illustrate this, consider the work of Gentry et al. [55], which makes use of the role assignment from Benhamouda et al. [21] to sample committees. Benhamouda et al. show that if the global corruption ratio is $f = 0.25$, then committees of size roughly 40k are needed to get honest majority! The associated YOSO MPC protocol given by Gentry et al. [55] has a communication complexity that scales linearly with the size of the committee, leading to extremely poor performance for committees as large as 40K.

### 3.1.2 Our Contributions

In this work we explore YOSO MPC in a context where the committees not only have an honest majority, but there is a *gap* proportional to the committee size between the number of honest parties and the number of corrupt parties. In more detail, let $n$ be the committee size, and let $t$ be the amount of corruptions in the committee. Instead of studying the case $n = 2t + 1$, which is the setting shared by all prior works in YOSO MPC, we initiate the study of YOSO MPC for the setting in which the committee satisfies $t < n(\frac{1}{2} - \epsilon)$, for some constant $\epsilon > 0$. Our motivation to do so is two-fold:

1. As we show in our work, when $t < n(\frac{1}{2} - \epsilon)$ it is possible to design YOSO MPC protocols that scale much better as $n$ grows, in contrast to the case when $\epsilon = 0$. In particular, it is possible to obtain an online phase whose total communication is *independent* of the committee size $n$, allowing for better scalability.
2. For large number of parties—which is the context that YOSO protocols are aimed at—it is not unreasonable to assume that the adversary cannot corrupt not only 49.99% (1/2) of the parties, but a smaller percentage such as, say 45% or 40% $(1/2-\epsilon)$.

In fact, for committee-based protocols such as YOSO, we show that requiring the committees to have a gap $\epsilon > 0$ only increases committee sizes by a marginal amount, while reducing online communication drastically.

We elaborate on these points below.

#### 3.1.2.1 Improved communication of YOSO MPC.

We present a YOSO MPC protocol that is divided in two phases. In an offline phase, a series of *preprocessing committees* are in charge of generating certain correlations that can be consumed by subsequent committees. In this phase, our communication is asymptotically the same as prior works: $O(n|C|)$. Our main benefits appear in the online phase, which is set in motion once the inputs to the computation are known. Here, our total communication is $O(|C|)$, *independently of the committee size* $n$. For instance, this allows deploying YOSO in large scale scenarios, such as the blockchain settings initially devised in the YOSO literature. In more detail, we make use of packed secret-sharing in order to improve prior protocols by a factor of $k \approx n \cdot \epsilon$, at the expense of a slightly larger committee size. Note that the bigger the $\epsilon$, which corresponds to less corruption tolerance, the more the efficiency gains.

#### 3.1.2.2 Role assignment for committees with "gap".

We observe that prior role-assignment works such as the one in the work of Benhamouda et al. [21] focus on choosing the committee size $n$ so that the new corruption threshold $t$ satisfies $t < n/2$, and YOSO MPC protocols such as given by Gentry et al. [55] are designed assuming committees of this size, with $1/2$ as the corruption ratio. In order to obtain committees whose corruption ratio is $1/2 - \epsilon$, we generalize the probability analysis from [21] and show that, by choosing committees of *slightly* larger size, we can achieve a smaller corruption ratios of $1/2 - \epsilon$, for some $\epsilon > 0$. Our results are discussed in Section 3.5. Crucially, we show that the cost of enabling the gap $\epsilon > 0$ is really minimal, and its benefits are substantial. For example, for 5% global corruptions we can already get 28× improvement by moving from committees of size 900 to 1000. For larger corruption ratios such as 20%, we can get 1000× online improvement with respect to the approach from Benhamouda et al. and Gentry et al. by moving from committees of size ≈ 18k to ≈ 20k.
**Remark 1** (Fail-stop tolerance.)**.** *We also highlight an important benefit of considering the ratio* $1/2 - \epsilon$ *instead of* $1/2$*. All current YOSO solutions are designed to tolerate certain amount of active corruptions. However, in large-scale settings, it's essential to safeguard*

*against not only active attacks but also fail-stop parties—honest participants who may inadvertently fail due to various reasons, including external attacks like denial of service, software/hardware errors, or natural events. In current YOSO MPC protocols fail-stop parties are treated the same as active corruptions, which has been shown to be an overkill in the non-YOSO literature [11, 48, 52, 71]. Considering a ratio of* $1/2 - \epsilon$ *not only allows us to gain efficiency, but it also allows us to tolerate unresponsive honest parties. We show in Section 3.4.5 that, if we cut by a factor of two the gains in communication, we can tolerate* $n\epsilon$ *honest parties who may become unresponsive during the protocol execution. This can happen perhaps due to crashes or other issues, which is essential for large scale settings such as YOSO MPC.*

### 3.1.3 Related work

Since the introduction of the YOSO model by Gentry et al. [55], numerous works focused on improving various aspects of YOSO constructions. Most related to our work are those, which study either **(1)** role-assignment protocols, or **(2)** YOSO MPC constructions.

**YOSO Role-Assignment.** The task of assigning physical machines to roles, along with a mechanism of sending private messages to these roles is a core building block in YOSO constructions. The seminal work of Benhamouda et al. [21] introduced the first such mechanism – *receiver-anonymous communication channels* (RACCs). Benhamouda et al.'s solution allows protocol participants to generate short-term keys for the next committee members, with the public portion of the key known to everyone, and the secret portion known only to the corresponding machine. This is done *without* revealing which machines are selected to participate in the next committee. Later, Gentry et al.[56] introduced another RACCs solution. Their protocol supported higher adversarial thresholds, but at the cost of being much more computationally expensive than the solution of Benhamouda et al. More recently, Campanelli et al. [31] proposed an *Encryption to the Future* primitive, a paradigm of sending messages to the anonymous committees in the future. Their solution is based on a special kind of witness encryption. Finally, Cascudo et al. [37] recently proposed another solution, based on publicly verifiable secret sharing (PVSS) towards anonymous committees.

**YOSO MPC.** The first MPC constructions both in the information-theoretic and the computational settings have been proposed by Gentry et al. [55]. The information-theoretic construction of Gentry et al. [55] is based on the famous BGW protocol [20], which, as the authors note, is essentially already a YOSO protocol in the semi-honest setting. To extend

it to the malicious setting, Gentry et al. provide YOSO-style equivalents for IT MACs, IT signatures etc. Unfortunately, the communication complexity of the resulting protocol is prohibitively high. The computationally secure solution is based on the CDN protocol [45]. Intuitively, CDN uses a a linearly homomorphic threshold encryption scheme, where the public key is known to everyone, and the secret key is shared among the committee members. The circuit is then evaluated gate-by-gate, with the addition gates being computed locally. To perform multiplications, parties can decrypt partial results using their shares of the global threshold key. One remaining open problem was the generation of the setup, i.e., the generation of the threshold public key and sharing of the secret key shares to the first committee. This was addressed by Braun et al. [30], who showed how to obtain distributed key generation for a linearly homomorphic threshold encryption scheme in the YOSO setting. Using this primitive allows to easily obtain a CDN-style solution for YOSO MPC. Concurrently to the work of Braun et al., Kolby et al. [83] introduced constant-round YOSO MPC protocols without setup based on garbled circuits and threshold fully homomorphic encryption. Kolby et al. further proposed (a non-constant) CDN-based protocol, which they use to obtain the (constant-round) setup of one of their other constructions.

## 3.2 Technical Overview – Improving Computational YOSO MPC

We now outline our YOSO MPC protocol in the computational setting. While the state-of-the-art protocol boasts an amortized communication complexity of $O(n)$ per circuit gate [55], assuming the circuit's width is $O(n)$, our protocol leverages the widely used offline/online paradigm. In particular, we achieve $O(1)$ online and $O(n)$ total communication complexity per gate, under the assumption that the circuit's width is $O(n)$. In the following, we will use bold font to represent a vector.

### 3.2.1 Starting idea: Building upon an Efficient Non-YOSO Protocol in the Offline/Online Paradigm

In our scheme, we follow the approach of Turbopack [50], which achieves MPC with abort with constant online communication complexity in the traditional (non-YOSO) setting. We now review the main techniques used in this work. At a high level, Turbopack follows the common MPC approach of secret-sharing the clients' secrets, and then evaluating the circuit

gate-by-gate. To achieve its constant amortized communication, Turbopack utilizes *packed* secret sharing, where $k \in O(n)$ secrets are stored using the same sharing. Intuitively, this ensures that the cost of evaluating a set of $k$ multiplication gates is the same as evaluating only a single multiplication gate.

As in the other works which utilize packing, the main challenge in Turbopack is to solve what is known as the *network routing issue*. Specifically, at each circuit layer, the secrets within a packed secret sharing may not be in correct order. For instance, the packed vectors may not be correctly aligned, or sharings intended for a set of multiplication gates are scattered among many different packed output sharings of the previous layer. Naively, re-organizing the secrets at each level by first reconstructing, and then placing them in correct positions of the packed vectors would result in communication complexity of $O(n)$, thus defying the purpose of using packing.

Instead, Turbopack relies on a *circuit-dependent* preprocessing phase to prepare correlated randomness, which later helps with solving the routing issue in an efficient way. During this phase, the circuit is known, but the parties' inputs are not. Turbopack assigns a value $\lambda^\alpha$ to each wire $\alpha$ of the circuit, such that:

- For any output wire of an input gate and any output wire of a multiplication gate, $\lambda^\alpha$ is uniformly random.
- For any addition gate $\gamma$ with input wires $\alpha$ and $\beta$, $\lambda^\gamma = \lambda^\alpha + \lambda^\beta$.

After the preprocessing, for each batch $\boldsymbol{\alpha} = (\alpha_1, \ldots, \alpha_k)$ of $k$ input and multiplication gates, each party has a packed share of $\lambda^{\boldsymbol{\alpha}} = (\lambda^{\alpha_1}, \ldots, \lambda^{\alpha_k})$.

In the online phase, Turbopack uses these sharings to step-by-step compute $\mu^\alpha = v^\alpha - \lambda^\alpha$ in clear for each gate $\alpha$, where $v^\alpha$ is the actual value on this wire. The addition gates can be computed locally, as $\mu^\gamma = \mu^\alpha + \mu^\beta = v^\alpha + v^\beta - \lambda^\alpha - \lambda^\beta$, where $\mu^\alpha$ and $\mu^\beta$ are known, and each party has a share of $\lambda^\alpha$ and $\lambda^\beta$ from the preprocessing. For the multiplication gates, Turbopack adapts the technique of packed Beaver triples [66], a generalization of the famous Beaver triple technique [14], to their solution. Here, the computation again crucially relies on the parties knowing the preprocessed shares of $\lambda^{\boldsymbol{\alpha}}$.

To obtain a solution which is compatible with the YOSO model, we need to solve the following issues:

- In YOSO, the parties who are participating in the online phase of the protocol are *not* the same as the ones preparing the preprocessed values. We must ensure that the parties of the online committee obtain the preprocessed values, e.g., shares of $\boldsymbol{\lambda}^{\boldsymbol{\alpha}}$, in

a way that does not break security, as the adversary can corrupt parties both in the online phase and during the preprocessing. More crucially, in the YOSO setting, the roles of the online committee that are supposed to obtain the preprocessed values *can not be assumed to be known during the preprocessing phase*. Thus, we must find a way to pass the secret values to these roles into the future *without assuming knowledge of their YOSO role keys*, all while ensuring that the communication complexity remains efficient.

- Finally, note that Turbopack achieves security with abort, while our goal is to obtain a solution which achieves guaranteed output delivery. One might think that moving to a computational setting and utilizing non-interactive zero-knowledge proofs ought to be sufficient to obtain GOD. However, in oder to obtain its low communication complexity, Turbopack uses a trick which seems inherently incompatible with GOD. Intuitively, shares of $\mu$ are revealed only to a *single* party. Thus, a single corruption might prevent the protocol from finishing. To ensure that our solution achieves GOD, we must find another way to compute $\mu_\gamma$, while retaining efficient communication complexity.

### 3.2.2 YOSO-ifying Turbopack via CDN

To solve these issues, we carefully combine the techniques from Turbopack with the ideas of the famous CDN protocol [45], and utilize a few additional tricks to ensure that the amortized online communication complexity remains constant per gate.

Recall that CDN relies on a system-wide public key $tpk$ of a linearly homomorphic threshold encryption scheme, where the corresponding secret key $tsk$ is shared among the committee members. In our case, the committees will be changing, hence we will refresh the shares of the $tsk$ after each usage.

Note that ours is not the first work to propose using CDN in the YOSO setting – both the original YOSO work [55], as well as the work by Braun et al. [30] suggest a CDN-based approach. Indeed, CDN is very appealing in the YOSO setting: with private communication between the committees being one of the bottlenecks in YOSO MPC, CDN-based approaches require only a *small* secret state to be maintained, i.e., the parties' shares of the secret key $tsk$. However, the communication complexity in both works remains high due to the following. To compute a multiplication gate with encrypted inputs $x$ and $y$, both works simply generate Beaver triples $(a, b, c)$ encrypted under the $tpk$ on the fly, and let committee members use their shares of $tsk$ to decrypt the ciphertexts $x + a$ and

$y + b$. Note that during the decryption, committee members must not only compute and publish their share of the decrypted value, but also pass the (refreshed) shares of $tsk$ to the next committee. Naively, this results in communication complexity $O(n^2)$ per gate. If committees are responsible for $O(n)$ gates, the communication complexity can be brought down to amortized $O(n)$ by letting committees process $O(n)$ gates in parallel. However, further amortization is not possible, as to decrypt each ciphertext that is encrypted under $tpk$, we ultimately need $O(n)$ committee members to supply their shares.

As we will see, in our protocol, by building upon Turbopack, during the online phase **(1)** we will not have to compute Beaver triples, and, more importantly, **(2)** we will perform the expensive threshold decryption procedure only once, instead of for each (batch) of multiplication gates. Instead, we will use the threshold encryption primarily for computations during the offline phase, as well as for passing preprocessed secret values, e.g., shares of $\boldsymbol{\lambda}^{\boldsymbol{\alpha}}$, to online parties in an efficient way.

**Keys For Future: YOSO-compatible Preprocessing Usage.** In more detail, in order to pass preprocessed secret values to the members of the online committees (without knowing their YOSO role keys), we propose the following approach of making traditional preprocessing YOSO-friendly. First, during the setup we generate *keys for future* (KFF) – these are the keys that act as substitutes for the YOSO role keys of the future roles. We generate a KFF public and secret key pair for each member of the later committee. We publish the public keys, and encrypt secret keys under the threshold public key $tpk$ of the linearly homomorphic threshold encryption scheme. Then, whenever a party needs to encrypt certain information to a future committee role whose role key is not known yet, the party encrypts it under the KFF public key of that specific role. Later, once the protocol reaches the phase where the role keys of the corresponding committees are known, one committee can use their shares of $tsk$ to decrypt the ciphertexts which contain the KFF secret keys, and re-encrypt these secret keys under the YOSO role key of the corresponding role. Note that using KFF, as opposed to simply encrypting every secret value that needs to be passed to a future role under the $tpk$ is crucial – each value that is reconstructed via the CDN's decryption procedure requires amortized communication complexity of at least $O(n)$, as $O(n)$ committee members must publish their shares in order to reconstruct this value. Assuming that a role processes $O(n)$ gates, using KFF results in constant online complexity per gate, as opposed to the linear complexity of the naive approach.

We now describe the stages of our protocol – setup, offline, and online phase, in reverse order, and give intuition for the challenges we encounter and our approaches to solving these.

**Notation and Packed Shamir Secret Sharing.** Before diving in, we briefly specify the notation we use in the following, as well as recall the *packed* Shamir secret sharing scheme [53], which is a generalization of the standard Shamir secret sharing [99]. Intuitively, it allows to secret-share a batch of secrets using a single Shamir sharing. For a vector $x \in \mathcal{F}^k$, we use $[\![\boldsymbol{x}]\!]_d$ to denote a degree-$d$ packed Shamir sharing, where $k-1 \leq d \leq n-1$. It requires $d+1$ shares to reconstruct the whole sharing, and any $d-k+1$ shares are independent of the secrets.

Same as the standard Shamir secret sharing, the packed version is linearly homomorphic, i.e., for all $d \geq k-1$ and $\boldsymbol{x}, \boldsymbol{y} \in \mathcal{F}^k$, $[\![\boldsymbol{x}+\boldsymbol{y}]\!]_d = [\![\boldsymbol{x}]\!]_d + [\![\boldsymbol{y}]\!]_d$. Further, we can perform multiplication on the shares as follows: For all $d_1, d_2 \geq k-1$ subject to $d_1 + d_2 < n$, and for all $\boldsymbol{x}, \boldsymbol{y} \in \mathcal{F}^k$, $[\![\boldsymbol{x} * \boldsymbol{y}]\!]_{d_1+d_2} = [\![\boldsymbol{x}]\!]_{d_1} * [\![\boldsymbol{y}]\!]_{d_2}$. Finally, we note that packed Shamir sharing is *multiplication-friendly* [66], in the sense that when $d \leq n-k$, all parties can locally multiply a public vector $\boldsymbol{c} \in \mathcal{F}^k$ with a degree-$d$ packed Shamir sharing $[\![\boldsymbol{x}]\!]_d$ as follows:

1. All parties first locally compute a degree-$(k-1)$ packed Shamir sharing of $\boldsymbol{c}$, denoted by $[\![\boldsymbol{c}]\!]_{k-1}$. Note that for a degree-$(k-1)$ packed Shamir sharing, all shares are determined by the secrets.
2. All parties then locally compute $[\![\boldsymbol{c} * \boldsymbol{x}]\!]_{n-1} = [\![\boldsymbol{c}]\!]_{k-1} * [\![\boldsymbol{x}]\!]_{n-k}$.

In the following, we denote the above process by $[\![\boldsymbol{c} * \boldsymbol{x}]\!]_{n-1} = \boldsymbol{c} * [\![\boldsymbol{x}]\!]_{n-k}$.

### 3.2.3 Online Phase

During the online phase, we follow Turbopack's approach of efficiently computing $\mu^\alpha = v^\alpha - \lambda^\alpha$ in clear for each circuit wire $\alpha$. Recall that here, $v^\alpha$ is the actual value on the wire, and $\lambda^\alpha$ is the value assigned to this wire during the preprocessing. We now describe how to obtain $\mu^\alpha$ for each gate type.

In order to compute $\mu^\alpha$ for the inpute wire $\alpha$ of a circuit, where this input is supplied by some client $C$, we let $C$ learn $\lambda^\alpha$. Then, $C$ can publish $\mu^\alpha = v^\alpha - \lambda^\alpha$. To compute $\mu^\gamma$ for an output wire $\gamma$ of an addition gate with input wires $\alpha$ and $\beta$, anyone can simply add $\mu^\alpha$ and $\mu^\beta$.

Computing multiplication gates is trickier. To obtain a communication-efficient solution, Turbopack adapts the technique of packed Beaver triples [66], which are triples $([\![\boldsymbol{a}]\!]_d, [\![\boldsymbol{b}]\!]_d, [\![\boldsymbol{c}]\!]_d)$, where $\boldsymbol{a}$ and $\boldsymbol{b}$ are random vectors from $\mathcal{F}^k$ and $\boldsymbol{c} = \boldsymbol{a} * \boldsymbol{b}$. Intuitively,

for a group of multiplication gates with input wires $\boldsymbol{\alpha}, \boldsymbol{\beta}$ and output wires $\boldsymbol{\gamma}$, it holds that

$$\begin{aligned}\boldsymbol{\mu}^{\boldsymbol{\gamma}} = \boldsymbol{v}^{\boldsymbol{\alpha}} * \boldsymbol{v}^{\boldsymbol{\beta}} - \boldsymbol{\lambda}^{\boldsymbol{\gamma}} &= (\boldsymbol{\mu}^{\boldsymbol{\alpha}} + \boldsymbol{\lambda}^{\boldsymbol{\alpha}}) * (\boldsymbol{\mu}^{\boldsymbol{\beta}} + \boldsymbol{\lambda}^{\boldsymbol{\beta}}) - \boldsymbol{\lambda}^{\boldsymbol{\gamma}} \\ &= \boldsymbol{\mu}^{\boldsymbol{\alpha}} * \boldsymbol{\mu}^{\boldsymbol{\beta}} + \boldsymbol{\mu}^{\boldsymbol{\alpha}} * \boldsymbol{\lambda}^{\boldsymbol{\beta}} + \boldsymbol{\lambda}^{\boldsymbol{\alpha}} * \boldsymbol{\mu}^{\boldsymbol{\beta}} + \boldsymbol{\lambda}^{\boldsymbol{\alpha}} * \boldsymbol{\lambda}^{\boldsymbol{\beta}} - \boldsymbol{\lambda}^{\boldsymbol{\gamma}} \quad (3.1)\end{aligned}$$

Thus, in order to compute $\boldsymbol{\mu}^{\boldsymbol{\gamma}}$, we must let parties of the current online committee obtain shares of $\boldsymbol{\lambda}^{\boldsymbol{\alpha}}, \boldsymbol{\lambda}^{\boldsymbol{\beta}}$ and shares of $\boldsymbol{\Gamma}^{\boldsymbol{\gamma}} = \boldsymbol{\lambda}^{\boldsymbol{\alpha}} * \boldsymbol{\lambda}^{\boldsymbol{\beta}} - \boldsymbol{\lambda}^{\boldsymbol{\gamma}}$. Looking ahead, these values will be generated during the preprocessing. In order to efficiently pass them to the members of the online committee *without* assuming that the YOSO role keys of the online committees are known during the preprocessing phase, we rely on our keys for future (see 3.2.2 for details). Briefly, the secret shares of $\boldsymbol{\lambda}^{\boldsymbol{\alpha}}, \boldsymbol{\lambda}^{\boldsymbol{\beta}}$, and $\boldsymbol{\Gamma}^{\boldsymbol{\gamma}}$ are encrypted under the keys for future, and the secret key parts of the keys for future are in turn encrypted under the $tpk$. To give parties their secret shares, during the online phase we let the first committee use their shares of $tsk$ to re-encrypt the secret shares of $\boldsymbol{\lambda}^{\boldsymbol{\alpha}}, \boldsymbol{\lambda}^{\boldsymbol{\alpha}}$, and $\boldsymbol{\Gamma}^{\boldsymbol{\gamma}}$ under the (now known) YOSO role keys of the online committee members. Note that after this point we will not require access to $tsk$ anymore. Hence, there is no need to re-share shares of $tsk$, which allows us to save communication.

**Efficiently Computing $\boldsymbol{\mu}_{\boldsymbol{\gamma}}$ with GOD.** One issue remains. Note that in Turbopack $\boldsymbol{\mu}_{\boldsymbol{\gamma}}$ is computed by having each party compute its share of the equation above locally, and send the result to a single party $P_1$. This ensures that the communication complexity remains low – each out of $n$ parties only sends one message to $P_1$, and since $\boldsymbol{\mu}_{\boldsymbol{\gamma}}$ covers to $O(n)$ gates, this ensures that amortized communication complexity per gate is still constant. This unfortunately does not work for us, as our goal is to obtain a solution which has the guaranteed output delivery property. To alleviate this, we first observe that, in all currently known YOSO protocols, P2P messages are sent by posting encryptions to future committees in, for example, a bulletin board. However, what is more interesting is that *broadcast messages* are also disseminated in the same way, which means that, in YOSO MPC, **broadcast has the same cost as P2P communication**! We are able to leverage this observation to improve the communication of YOSO MPC by using less reserved use of the broadcast channel. Additionally, in order to ensure that $\mu_\gamma$ is reconstructed correctly, roles who are contributing their local shares will prove that they did the computation correctly. For this, as in previous works, we will have parties compute and publish a NIZK proof of correctness.

### 3.2.4 Offline Phase

During the offline phase, for each batch of $k$ multiplication gates our goal is to prepare shares of correctly routed packed wire values $\boldsymbol{\lambda}^{\boldsymbol{\alpha}}$, as well as $\boldsymbol{\Gamma}^{\boldsymbol{\gamma}}$. To achieve this, for each multiplication gate, we first prepare a Beaver triple. Then, we let the current committee members jointly generate a random value $\lambda^\alpha$ for each output wire of an input/multiplication gate $\alpha$, encrypted under the global threshold public key $tpk$ (in the following, denote such ciphertext for the wire $\alpha$ by $c^\alpha$). Note that for all values that are published during the offline phase, we let parties attach NIZK proofs explaining that they did everything correctly. We then use only those values for the computation, for which the corresponding proofs verify.

Given ciphertexts $c^\alpha$, we process the circuit gate by gate (first without using packing). For the addition gates, we simply compute the sum of two ciphertexts containing the random wire values of the two input wires to this gate. For each multiplication gate with encrypted inputs $c^\alpha$ and $c^\beta$, and encrypted output wire value $c^\gamma$, we need to compute the encryption of $\Gamma_\gamma = \lambda^\alpha * \lambda^\beta - \lambda^\gamma$. For this, we follow the approach of Gentry et al. [55]. Specifically, to compute a multiplication gate with encrypted inputs $c^\alpha$ and $c^\beta$, we consume one of the Beaver triples $(a, b, c)$, and let committee members use their shares of $tsk$ to decrypt the ciphertexts $c^\alpha + a$ and $c^\beta + b$.

Next, in order to ensure that our online phase is efficient, we pack the random wire values $(\lambda^{\alpha_1}, \ldots, \lambda^{\alpha_k})$ for each group of $k$ circuit input wires $(\alpha_1, \ldots, \alpha_k)$ which are supplied by a single client. Similarly, we pack the random vector $\boldsymbol{\lambda}^{\boldsymbol{\alpha}} = (\lambda^{\alpha_1}, \ldots, \lambda^{\alpha_k})$ of the output wires for each batch of $k$ multiplication gates. This is done as follows: Given ciphertexts $c^{\lambda^{\alpha_1}}, \ldots, c^{\lambda^{\alpha_k}}$, we first generate $t$ *additional* encryptions of random values $c^{r_1}, \ldots, c^{r_t}$. Then, we use these ciphertexts to homomorphically compute encryptions of the evaluation points of the polynomial $f(x)$ of degree $t + k - 1$, which on each $x$-point $-(i-1)$ evaluates to $\lambda_i^\alpha$, $i \in [k]$, and on $i$ evaluates to $r_i$, $i \in [t]$. Then, we use all points (both the values $\lambda^{\alpha_i}$, $i \in [k]$ and extra random values $r_i$, $i \in [1, \ldots, t]$), to locally interpolate using the homomorphic properties of the threshold encryption scheme, and this way obtain *packed* shares $f(1), \ldots, f(n)$ of $\boldsymbol{\lambda}^{\boldsymbol{\alpha}}$ which are encrypted under $tpk$.

Finally, note that currently the packed shares are encrypted under the $tpk$. Note that this is problematic: If a share is encrypted under $tpk$, then during the online phase one committee will have to spend $O(n)$ communication to decrypt a *single packed share*, with each committee member publishing its CDN-style share of the packed share. This would defy the usage of packing and result in communication cost $O(n^2)$ per packed sharing, hence linear amortized cost—no better than prior works. To solve this, we re-encrypt the packed

shares to the KFFs of the roles in the online committee who will use these shares. Doing so allows us to pay the linear cost during the offline phase, and keep the online phase very efficient.

## 3.3 Preliminaries

We now describe our building blocks. We take large parts the following verbatim from Gentry et al. [55].

### 3.3.1 Linearly Homomorphic Key Rerandomizable Threshold Encryption

A linearly homomorphic (over ring $\mathbb{R}$) key rerandomizable threshold encryption scheme $\mathsf{TE}$ has the following algorithms:

- $\mathsf{TKGen}(1^\kappa) \to (tpk, tsk_1, \ldots, tsk_n)$: An algorithm that, given the security parameter $\kappa$, sets up the public key $tpk$ and the shares $tsk_1, \ldots, tsk_n$ of the secret key.
- $\mathsf{TEnc}(tpk, m; r) \to \beta$: An algorithm that, given the public key, a message $m \in \mathbb{R}$ and randomness $r$, outputs an encryption $\beta$ of $m$.
- $\mathsf{TPDec}(tpk, tsk_i, \beta) \to d_i$: An algorithm that, given the public key, a share $tsk_i$ of the secret key and a ciphertext $\beta$, outputs a partial decryption $d_i$.
- $\mathsf{TDec}(tpk, \{d_i\}_{i \in S, |S| > t}) \to m$: An algorithm that, given sufficiently many partial decryptions, returns the decrypted message $m$.
- $\mathsf{TEval}(tpk, \beta_1, \ldots, \beta_k, \lambda_1, \ldots, \lambda_k) \to \beta$: A deterministic algorithm that, given the public key, ciphertexts $\beta_1, \cdots, \beta_k$ corresponding to messages $m_1, \ldots, m_k \in \mathbb{R}^k$ and coefficients $\lambda_1, \ldots, \lambda_k \in \mathbb{R}^k$, outputs a ciphertext $\beta$ that encrypts $\sum_{i=1}^k \lambda_i m_i \in \mathbb{R}$.
- $\mathsf{TKRes}(tpk, tsk_i; r_i) \to (m_{i,1}, \ldots, m_{i,n})$: An algorithm that, given the public key and a share of a secret key, produces $n$ messages to help with the rerandomization of the secret key sharing.
- $\mathsf{TKRec}(tpk, \{m_{j,i}\}_{j \in S, |S| > t}) \to tsk_i$: An algorithm that, given sufficiently many messages for the rerandomization of the secret key sharing, outputs a share of the secret key.
- $\mathsf{SimTPDec}(tpk, \beta, m, \{tsk_i\}_{i \in [n] \setminus S}, \{d_i\}_{i \in S}) \to \{d_i\}_{i \in [n] \setminus S}$: A simulation algorithm that, given a ciphertext, a target message, and partial decryptions belonging to corrupt parties, simulates partial decryptions belonging to honest parties that cause $\mathsf{TDec}$ to

output the desired message.

**Security Properties.** A linearly homomorphic key rerandomizable threshold encryption scheme must satisfy the following correctness properties:

- Decryption on honestly produced ciphertext and keys must return the appropriate message.
- Decryption must remain correct after homomorphic evaluation.
- Decryption must remain correct after a rerandomization of the secret key sharing.

These correctness properties are intuitive, and we do not formalize them here. The important security property of a TE scheme is partial decryption simulatability, described below. Note that it trivially implies chosen plaintext security.

**Definition 12** (Partial Decryption Simulatability)**.** *Informally, a* TE *scheme has partial decryption simulatability if for any honestly produced ciphertext, desired message $m$ and fewer than $t$ partial decryptions, the algorithm* SimTPDec *produces remaining partial decryptions which cause* TDec *to return $m$. More formally, let $\kappa \in N$ be the security parameter, and let* $\mathsf{TE} = (\mathsf{TKGen}, \mathsf{TEnc}, \mathsf{TPDec}, \mathsf{TDec}, \mathsf{TEval}, \mathsf{TKRes}, \mathsf{TKRec}, \mathsf{SimTPDec})$ *be a* TE *scheme. Consider the game between a probabilistic polynomial-time adversary $A$ and a challenger $C$ described in Figure 3.1.*

TE *has partial decryption simulatability if for any sufficiently large security parameter $\kappa$, for any probabilistic polynomial-time adversary $A$, there exists a negligible function negl in the security parameter $\kappa$ such that the probability that $A$ wins the game is less than $\frac{1}{2} + negl(\kappa)$.*

We can instantiate such a linearly homomorphic key rerandomizable threshold encryption scheme by Shamir sharing a Paillier decryption key [46], see Gentry et al. [55] for details.

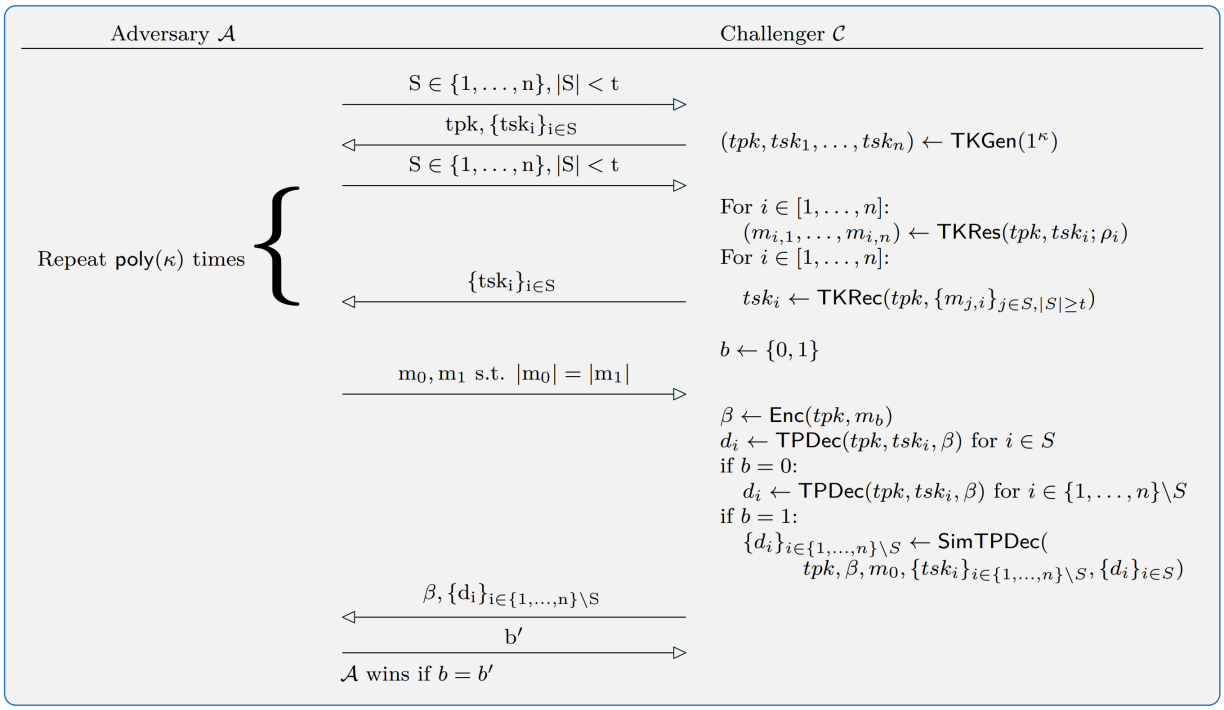
### 3.3.2 Non-Interactive Zero-Knowledge Arguments of Knowledge

A non-interactive zero-knowledge argument of knowledge (NIZKAoK) scheme has the following algorithms, as described by Groth and Maller [70].

$\mathsf{Setup}(1^\kappa, \mathcal{R}) \rightarrow (crs, td)$: An algorithm that, given the security parameter, sets up the global common reference string $crs$ and the trapdoor $td$ for the NIZKAoK system.

$\mathsf{P}(crs, \phi, w) \rightarrow \pi$: An algorithm that, given the common reference string $crs$ for a relation $\mathcal{R}$, a statement $\phi$ and a witness $w$, returns a proof $\pi$ that $(\phi, w) \in \mathcal{R}$. $\mathsf{V}(crs, \phi, \pi) \rightarrow$ **accept/reject**: An algorithm that, given the common reference string $crs$ for a relation

**Figure 3.1:** Security game for the partial decryption simulatability property of the TE




$\mathcal{R}$, a statement $\phi$ and a proof $\pi$, checks whether $\pi$ proves the existence of a witness $w$ such that $(\phi, w) \in \mathcal{R}$.

$\mathsf{SimP}(crs, td, \phi) \rightarrow \pi$: An algorithm that, given the common reference string crs for a relation $R$, the trapdoor $td$ and a statement $\phi$, simulates a proof of the existence of a witness $w$ such that $(\phi, w) \in R$.

**Security properties.** A NIZKAoK scheme must be correct (that is, verification using an honestly produced proof must return accept). The important security properties of a NIZKAoK scheme are zero knowledge, knowledge soundness, and simulation extractability, described below.

**Definition 13** (Zero Knowledge for NIZKAoK)**.** *Informally, a* NIZKAoK *scheme has zero knowledge if a proof does not leak any more information than the truth of the statement.*

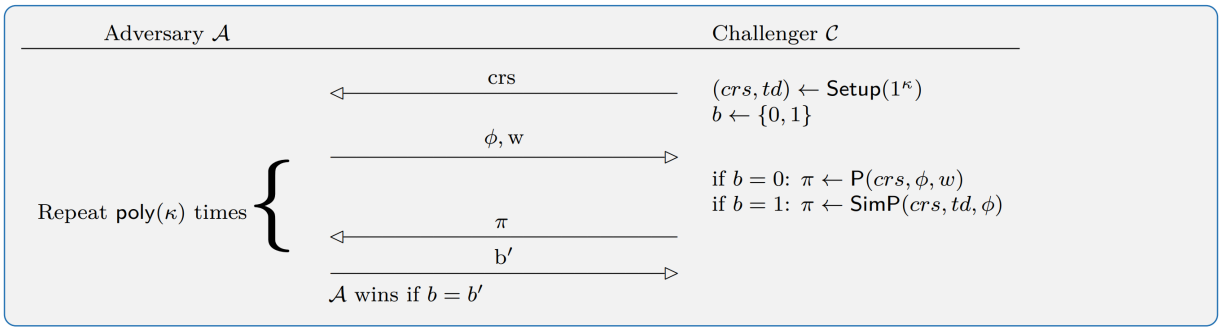
*More formally, let* $\kappa \in \mathbb{N}$ *be the security parameter, and let* $\mathsf{NIZKAoK} = (\mathsf{Setup}, \mathsf{P}, \mathsf{V}, \mathsf{SimP})$ *be a* NIZKAoK *scheme. Consider the game between a probabilistic polynomial-time adversary A and a challenger C described in Figure 3.2.*

NIZKAoK *has zero knowledge if for any sufficiently large security parameter* $\kappa$*, for any probabilistic polynomial-time adversary A, there exists a negligible function negl in the*

*security parameter $\kappa$ such that the probability that $A$ wins the game is less than $\frac{1}{2} + negl(\kappa)$.*

Informally, knowledge soundness is the property that guarantees that it is always possible to extract a valid witness from a proof that verifies. Simulation extractability is a stronger version of knowledge soundness, where it is always possible to extract a valid witness from a proof that verifies even if the adversary has access to a simulation oracle. This is a flavor of non-malleability; an adversary should not even be able to modify a simulated proof in order to forge a proof.

**Figure 3.2:** Security game for the zero knowledge property of the NIZKAoK




**Definition 14** (Simulation Extractability for NIZKAoK)**.** *Informally, a* NIZKAoK *scheme has simulation extractability if it is always possible to extract a valid witness from a proof that verifies.*

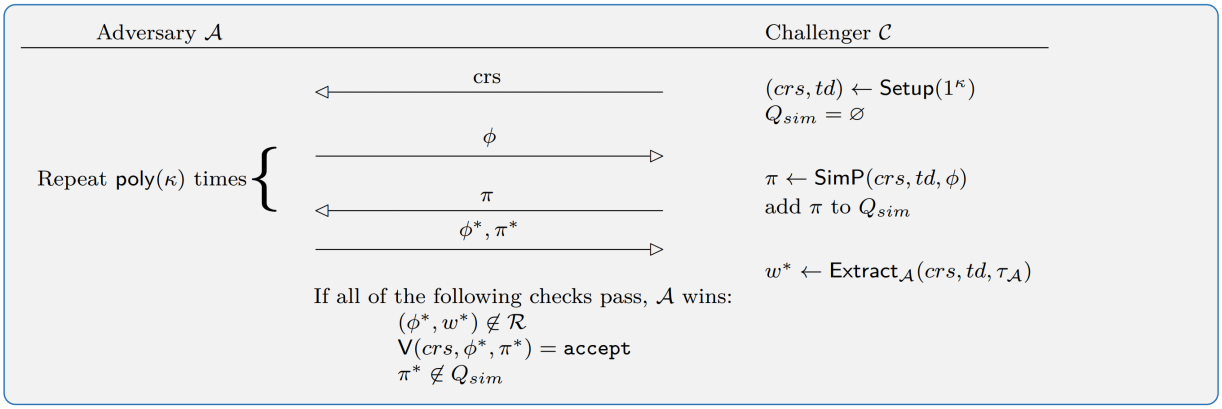
*More formally, let $\kappa \in \mathbb{N}$ be the security parameter, and let* $\mathsf{NIZKAoK} = (\mathsf{Setup}, \mathsf{P}, \mathsf{V}, \mathsf{SimP})$ *be a* NIZKAoK *scheme. Consider the game between a probabilistic polynomial-time adversary $\mathcal{A}$ and a challenger $\mathcal{C}$ described in Figure 3.3, where $\tau_{\mathcal{A}}$ denotes the adversary's inputs and outputs, including its randomness:*

NIZKAoK *has simulation extractability if for any sufficiently large security parameter $\kappa$, for any probabilistic polynomial-time adversary $A$, there exists an extraction algorithm* $\mathsf{Extract}_{\mathcal{A}}$ *and a negligible function negl in the security parameter $\kappa$ such that the probability that $\mathcal{A}$ wins the game is less than $negl(\kappa)$.*

### 3.3.3 Our Model and YOSO Background

In this section we outline our model. Towards this, we first recap the background on YOSO. At its core, it is a variation of the UC framework [32] which separates between physical machines and roles that these machines play in the protocol. It is the job of the roles

**Figure 3.3:** Security game for the simulation extractability property of the NIZKAoK




assignment functionality to map roles to the machines. The YOSO MPC protocols are then described entirely using roles, we refer to such protocols as "abstract YOSO" (vs. "natural YOSO" which includes explicit role assignment) in the following. We now recap further details of the YOSO model and its differences to the standard UC. For a detailed version, refer to the original YOSO work [56].

- To ensure that the roles speak only once, the framework "yoso-ifies" them with a YOSO wrapper. This wrapper ensures that roles are killed immediately after they have spoken. This is modelled by a Spoke token which ideal functionalities send to roles (the time at which the functionality sends the token is determined by the functionality itself). Upon obtaining a Spoke token, the role also passes it onto its sub-routines and its environment. Once a roles is killed, the machine executing it also erases any associated state, which prevents the adversary from obtaining any information from the roles that have already spoken by corrupting the corresponding machines.
- The roles have access to idealised communication functionalities, which in particular allows point-to-point messages between roles.
- Gentry et al. [56] note that if the adversary does not know which roles are assigned to a machine before it is corrupted, the "best" that an adversary can do is corrupt machines at random. They further make the observation that for such random corruptions, the difference between adaptive corruptions and static corruptions is minimal. This allows to design protocols secure against a somewhat restricted adversary in the abstract YOSO model, which nevertheless translate into adaptively secure protocols in the natural YOSO model. To control which corruptions are allowed, the framework

introduces a "corruption controller" (CC). The CC handles the random corruptions roughly as follows: If an adversary wishes to do a random corruption, it asks the environment, which passes the request to the CC. The CC samples the corruption and informs the adversary which role was chosen.

- While the roles which perform the computation are typically corrupt at random, the input and output nodes are assumed to be known machines and are subject to chosen corruptions.

We denote a yoso-ified role $\mathbb{R}$ by $\mathsf{YoS}(\mathbb{R})$, and we denote the protocol obtained by yoso-ifying all roles in protocol $\pi$ by $\mathsf{YoS}(\pi)$. We say that $\pi$ YOSO-securely implements $\mathcal{F}$ and write $\pi \leq^{YOSO} \mathcal{F}$, if $\mathsf{YoS}(\pi)$ UC-securely implements $\mathcal{F}$ against the given class of controlled environments. Recall that $\pi$ UC-securely implements $\mathcal{F}$, if for all PPT adversaries $\mathcal{A}$ there exists a PPT simulator $\mathcal{S}$ such that $\mathsf{Real}_{\pi,\mathcal{A},\mathcal{E}} \approx \mathsf{Ideal}_{\mathcal{F},\mathcal{S},\mathcal{E}}$ for all PPT environments $\mathcal{E}$. Here, $\mathsf{Real}_{\pi,\mathcal{A},\mathcal{E}}$ denotes the random variable representing $\mathcal{E}$'s output in the real world, where the adversary $\mathcal{A}$ is interacting with the honest parties who execute $\pi$; and $\mathsf{Ideal}_{\mathcal{F},\mathcal{S},\mathcal{E}}$ denotes the random variable representing $\mathcal{E}$'s output in the ideal world, where the simulator $\mathcal{A}$ is interacting with the ideal functionality $\mathcal{F}$.

In this work, we consider a synchronous model of execution, i.e., the protocol proceeds in rounds, and parties are aware in which round they are currently in. We consider further consider a rushing adversary, i.e., the adversary can see the messages sent by the honest roles before producing messages of the corrupt roles.

We now recall the ideal broadcast functionality $\mathcal{F}_{BC}$ defined by Gentry et al. [56].

**Ideal Functionality $\mathcal{F}_{BC}$ for Broadcast**

The ideal functionality has the following behavior.

- Initially create a map $y : \mathbb{N} \times \mathsf{Role} \to \mathsf{Msg}_\perp$ with $y(r, \mathbb{R}) = \perp$ for all $r, \mathbb{R}$.
  Below we use $y(r, \cdot)$ to denote the map $y' : \mathsf{Role} \to \mathsf{Msg}_\perp$ with $y'(\mathbb{R}) = y(r, \mathbb{R})$.
- On input $(\mathsf{Send}, \mathbb{R}, x_\mathbb{R} \in \mathsf{Msg})$ in round $r$ proceed as follows:
  1. Update $y(r, \mathbb{R}) = x_\mathbb{R}$. *Store inputs of the round.*
  2. Output $(\mathbb{R}, x_\mathbb{R})$ to $\mathcal{S}$. *Leak messages to the simulator in a rushing fashion.*
  3. If $\mathbb{R}$ is honest then give output $\mathsf{Spoke}$ to $\mathbb{R}$.
- On input $(\mathsf{Read}, \mathbb{R}, r')$ in round $r$ where $r' < r$ output $y(r, \cdot)$ to $\mathbb{R}$.

Finally, we define YOSO MPC. We consider a synchronous model, and recall the ideal functionality $\mathcal{F}^F_{\mathsf{MPC}}$ as defined by Gentry et al. The functionality distinguishes between two stages that the protocol goes through, $\mathsf{GettingInputs}$ and $\mathsf{Evaluated}$. While honest parties

give input in the first round, the protocol can be in the GettingInputs stage for a long time, as corrupted parties might only be committed to their inputs at a later point in time. Once the adversary $S$ decides that it is time to give outputs, it sets the stage to Evaluated. We distinguish between input roles $\mathsf{Role}^{\mathsf{In}}$, output roles $\mathsf{Role}^{\mathsf{Out}}$, and roles $\mathsf{Role}^{\mathsf{Cmp}}$ which perform the computation.

**Ideal Functionality $\mathcal{F}_{\mathsf{MPC}}^{F}$ for MPC**

The functionality is wrapping a function $F(x_1, \ldots, x_n) \to (y_1, \ldots, y_m)$. Roles in $\mathsf{Role}^{\mathsf{In}}$ hold inputs and roles in $\mathsf{Role}^{\mathsf{Out}}$ receive outputs.

- Initially set the stage to be GettingInputs. We set a default input for all roles, which they may overwrite later: for all $\mathbb{R} \in \mathsf{Role}^{\mathsf{In}}$ let $x_{\mathbb{R}} = 0$.
- On input $(\mathsf{Input}, \mathbb{R} \in \mathsf{Role}^{\mathsf{In}}, x \in \mathsf{Msg})$ proceed as follows:
  1. Store $x_{\mathbb{R}} = x$.
  2. If $\mathbb{R} \in \mathsf{Honest}$ then output $(\mathsf{Input}, \mathbb{R}, |x|)$ to $\mathcal{S}$.
  3. If $\mathbb{R} \in \mathsf{Malicious}$ output $(\mathsf{Input}, \mathbb{R}, x)$ to $\mathcal{S}$.

  If $\mathbb{R}$ is honest then output Spoke to $\mathbb{R}$. If $\mathbb{R}$ is honest then consider only the first input, and only if it is given in round 1.
- On Evaluated from $\mathcal{S}$ in a round $r > 1$ and when the stage is GettingInputs, set the stage to be Evaluated and compute $\{y_{\mathbb{R}}\}_{\mathbb{R} \in \mathsf{Role}^{\mathsf{Out}}} = F(\{x_{\mathbb{R}}\})_{\mathbb{R} \in \mathsf{Role}^{\mathsf{In}}}$. Store $y_{\mathbb{R}}$ for all $\mathbb{R} \in \mathsf{Correct}$ and output to $\mathcal{S}$ the value $\{y_{\mathbb{R}}\}_{\mathbb{R} \in \mathsf{Role}^{\mathsf{Out}} \cap \mathsf{Malicious}}$.
- On input $(\mathsf{Read}, \mathbb{R} \in \mathsf{Role}^{\mathsf{Out}})$, if the stage is Evaluated output $y_{\mathbb{R}}$ to $\mathbb{R}$.

The YOSO MPC is then defined as follows:

**Definition 15** (YOSO MPC)**.** *We say that $\pi$ YOSO-securely realizes $F$ against $\tau$ corruption if $\pi \leq^{YOSO} \mathcal{F}_{\mathsf{MPC}}^{F}$ for the set of environments allowed to corrupt any number of roles in $\mathsf{Role}^{\mathsf{In}} \cup \mathsf{Role}^{\mathsf{Out}}$ and a uniformly random fraction $\tau$ of $\mathsf{Role}^{\mathsf{Cmp}}$.*

## 3.4 Improving Computational YOSO MPC

In this section we describe our YOSO MPC protocol in the computational setting. It consists of three phases – setup, offline, and online phase. Importantly, we only assume the knowledge of the YOSO role-assignment public keys for the committee of a given phase starting *only* with the first committee of that phase. In particular, during the offline phase we do not assume knowledge of the public keys of the roles of the future online phase.

In the following, let $\kappa$ denote the security parameter. Let $C_i$ denote the $i$'s online

committee, $C_i^{\mathsf{Off}}$ denote the $i$'s offline committee, and $C_{i,j}$ denote the $j$'s role of the online committee $C_i$ (similarly for $C_{i,j}^{\mathsf{Off}}$). Let $n$ denote the size of the committee. For a set $S$, let $|S|$ denote the size of this set. Let $\mathsf{TE} = (\mathsf{TKGen}, \mathsf{TKEnc}, \mathsf{TPDec}, \mathsf{TDec}, \mathsf{TEval}, \mathsf{TKRes}, \mathsf{TKRec})$ denote a linearly homomorphic key rerandomizable threshold encryption scheme, let $\mathsf{NIZKAoK} = (\mathsf{Setup}, \mathsf{P}, \mathsf{V}, \mathsf{SimP})$ denote a simulation extractable non-interactive zero-knowledge argument of knowledge, and let $\mathsf{PKE} = (\mathsf{PKE.Gen}, \mathsf{PKE.Enc}, \mathsf{PKE.Dec})$ denote an additively homomorphic public key encryption scheme.

For the ease of presentation, we describe each phase of the protocol separately. In the following, we use bold font to represent a vector.

### 3.4.1 Setup

During the setup phase, we instantiate the components that we will need during the offline and online phases of the protocol. In particular, we generate the keys for future (KFF), which will allow us to efficiently pass messages to future roles whose role keys will not be known during the earlier stages of the protocol execution. We further generate the setup for the NIZK proof system that we will be using during the later phases to prove correctness of the protocol execution by specific parties. Finally, we set up the linearly homomorphic key rerandomizable threshold encryption. Note that we assume that in our encryption scheme the public key corresponds to a single decryption key. We describe our assumed setup in Figure 3.4.1 below.

YOSO-Setup

**Setup:**

1. Generate *keys for future*[a]:
   - For each role $C_{l,i}$ of each committee $C_l$ participating in the online phase:
     - Generate $(pk_{C_{l,i}}^{\mathsf{KFF}}, sk_{C_{l,i}}^{\mathsf{KFF}}) \leftarrow \mathsf{PKE.Gen}(1^\kappa)$.
     - Let $c_{C_{l,i}}^{\mathsf{KFF}} \leftarrow \mathsf{TEnc}(tpk, sk_{C_{l,i}}^{\mathsf{KFF}})$.
   - For each input-contributing party $P_i$:
     - Generate $(pk_{P_i}^{\mathsf{KFF}}, sk_{P_i}^{\mathsf{KFF}}) \leftarrow \mathsf{PKE.Gen}(1^\kappa)$.
     - Let $c_{P_i}^{\mathsf{KFF}} \leftarrow \mathsf{TEnc}(tpk, sk_{P_i}^{\mathsf{KFF}})$.
2. Generate the NIZK setup via $crs \leftarrow \mathsf{NIZKAoK.Setup}(1^\kappa)$.
3. Generate the threshold public key and secret keys as $(tpk, tsk_1, \ldots, tsk_n) \leftarrow \mathsf{TKGen}(1^\kappa)$. Publish $tpk$ and give $tsk_i$ to $C_{1,i}^{\mathsf{Off}}$.

[a]These are **not** the keys that will be later generated by the YOSO role-assignment for these

roles.

### 3.4.2 Offline Phase

During the offline phase, our goal is to prepare correlated randomness that will be later consumed during the online phase. Specifically, we generate the following values:

- For each input wire $\alpha$ that is supplied by client $P_i$, we generate a random value $\lambda^\alpha$.
- For each output wire $\alpha$ of a multiplication gate, we generate a random value $\lambda^\alpha$. Then, for each group of $k$ multiplication gates, we prepare a packed sharing of the vector $\boldsymbol{\lambda}^{\boldsymbol{\alpha}} = (\lambda^{\alpha_1}, \ldots, \lambda^{\alpha_k})$.
- For each group of $k$ multiplication gates with input wires $\boldsymbol{\alpha} = (\alpha_1, \ldots, \alpha_k)$ and $\boldsymbol{\beta} = (\beta_1, \ldots, \beta_k)$, and output wires $\boldsymbol{\gamma} = (\gamma_1, \ldots, \gamma_k)$, we compute a packed sharing of the vector $\boldsymbol{\Gamma}^{\boldsymbol{\gamma}} = \boldsymbol{\lambda}^{\boldsymbol{\alpha}} * \boldsymbol{\lambda}^{\boldsymbol{\beta}} - \boldsymbol{\lambda}^{\boldsymbol{\gamma}} = (\lambda^{\alpha_1} \cdot \lambda^{\beta_1} - \lambda^{\gamma_1}, \ldots, \lambda^{\alpha_k} \cdot \lambda^{\beta_k} - \lambda^{\gamma_k})$.

We then prepare the following data for the online phase:

- For each input wire $\alpha$ that is supplied by client $P_i$, we prepare a ciphertext $c^\alpha$, which encrypts $\lambda^\alpha$ under the KFF of $P_i$.
- For each group of $k$ multiplication gates with output wires $\boldsymbol{\gamma} = (\gamma_1, \ldots, \gamma_k)$, we prepare an encryption of the $i$-th packed share of $\boldsymbol{\lambda}^{\boldsymbol{\alpha}}$ under the public key of the $i$-th committee member of the online phase who will need this value.
- Similarly, for each group of $k$ multiplication gates with output wires $\boldsymbol{\gamma} = (\gamma_1, \ldots, \gamma_k)$, we prepare an encryption of the $i$-th packed share of $\boldsymbol{\Gamma}^{\boldsymbol{\gamma}}$ under the public key of the $i$-th committee member of the online phase who will need this value.

At a high level, the offline phase works as follows: First, for each multiplication gate, we prepare a Beaver triple. Then, we let each committee member generate a random value $\lambda_i^\alpha$ for each output wire of an input/multiplication gate, and encrypt it under the global threshold public key $tpk$, while attaching a NIZK proof that the result is a valid ciphertext. For each wire $\alpha$, we then use the homomorphic properties of the encryption scheme $\mathsf{TE}$ to add contributions of all committee members. This ensures that each wire value $\lambda^\alpha$ is truly random.[1] Given encryptions of these values, we then process the circuit gate by gate. In the following, let $c^\alpha$ denote the ciphertext containing the value $\lambda^\alpha$ for each wire $\alpha$. Note

[1] Although it is common in honest majority settings to apply *randomness extraction* to achieve better communication [47], in our case such techniques do not turn out to yield any benefits, given that, in all currently known YOSO protocols, the cost of broadcast is the same as peer-to-peer.

that for all values that are published in the following description, we ask the parties to always attach NIZK proofs stating that they did everything correctly. The parties then use for the computation only the values for which the corresponding proofs verify.

For the addition gates, we obtain the encryption of the output wire value by simply adding the ciphertexts containing the encryptions of the input values. For each multiplication gate with encrypted input wire values $c^\alpha, c^\beta$, and encrypted output wire value $c^\gamma$, we need to compute the encryption of $\Gamma^\gamma = \lambda^\alpha \cdot \lambda^\beta - \lambda^\gamma$. To do this, we use an encrypted Beaver triple $(c^x, c^y, c^z)$ to compute $c^\epsilon = c^\alpha + c^x$ and $c^\delta = c^\beta + c^y$. Then, we let the current committee use its shares of $tsk$ to decrypt $c^\epsilon$ and $c^\delta$. Denote the result by $\epsilon$ and $\delta$. Then, we use the homomorphic properties of the encryption scheme to obtain $c^\beta \cdot \epsilon - c^x \cdot \delta + c^z - c^\gamma$, which is precisely the encryption of $\lambda^\beta \cdot (\lambda^\alpha + \lambda^x) - \lambda^x \cdot (\lambda^\beta + \lambda^y) + \lambda^z - \lambda^\gamma = \lambda^\alpha \cdot \lambda^\beta - \lambda^\gamma$.

Finally, in order to ensure that our online phase remains efficient, we pack the random wire values $(\lambda^{\alpha_1}, \ldots, \lambda^{\alpha_k})$ for each group of $k$ input wires $(\alpha_1, \ldots, \alpha_k)$ which belong to a single client. Similarly, we pack the random vector $\boldsymbol{\lambda}^{\boldsymbol{\alpha}} = (\lambda^{\alpha_1}, \ldots, \lambda^{\alpha_k})$ of the output wires for each group of $k$ multiplication gates. For this we use the fact that we have encryptions $c^{\lambda^{\alpha_1}}, \ldots, c^{\lambda^{\alpha_k}}$: by getting $t$ extra encryptions of random values $c^{r_1}, \ldots, c^{r_t}$ (which can be obtained by letting each committee member prepare $t$ random values $r_{i,j}$, and encrypt each of these values under the global threshold key $tpk$, and then adding up the contributions), we can use these ciphertexts to homomorphically compute encryptions of the evaluation points of the polynomial $f(x)$ of degree $t + k - 1$, which on each $x$-point $-(i-1)$ evaluates to $\lambda_i^\alpha$, $i \in [k]$, and on $i$ evaluates to $r_i$, $i \in [t]$. Then, we can use all points (both the values $\lambda^{\alpha_i}$, $i \in [k]$ and extra random values $r_i$, $i \in [1, \ldots, t]$), to locally interpolate using the homomorphic properties of the threshold encryption scheme, and this way obtain *packed* shares $f(1), \ldots, f(n)$ of $\boldsymbol{\lambda}^{\boldsymbol{\alpha}}$ which are encrypted under $tpk$.

As the last step, we must re-encrypt previously computed packed shares (which are currently encrypted under the $tpk$) to the KFFs of the roles in the online committee who will use these shares. Note that this step is crucial: If we simply leave the packed shares encrypted under $tpk$, then during the online phase one committee will have to spend $O(n)$ communication to decrypt a *single packed share*, with each committee member publishing its CDN-style share of the packed share. This would defy the usage of packing and result in communication cost $O(n^2)$ per packed sharing, hence linear amortized cost—no better than prior works. Re-encrypting the share towards the KFF of an online committee member who will be using this value allows us to pay the linear cost during the offline phase, and keep the online phase very efficient.

For the ease of presentation, we first separately present a few helper functions, which we

will use throughout our protocol. We start with $\mathsf{Re\text{-}encrypt}$. This function takes as input a ciphertext $c$ which was encrypted using the threshold key $tpk$, and enables the current committee $C_l$ to re-encrypt the underlying plaintext using the given key $pk$. For this, the members of the current committee first recover their shares of $tsk$. See Figure 1 for details.

Note that the relation $R$ that we prove is the following:

$$R = \left\{ \begin{array}{l|l} \phi = (pk, tpk, \{pk_j\}_{j\in[n]}, & sk \text{ corresponds to } pk, \\ \{c_j^{new}\}_{j\in[n]}, \{c_j^{old}\}_{j\in S}, c^{new}, c^{old}) & s_j^{old} \leftarrow \mathsf{PKE.Dec}(sk, c_j^{old}) \text{ for } j \in S, \\ w = (sk, tsk, r_c, r, \{r_j\}_{j\in[n]}, d, & tsk \leftarrow \mathsf{TKRec}(tpk, \{s_j^{old}\}_{j\in S}), \\ \{s_j^{new}\}_{j\in[n]}, \{s_j^{old}\}_{j\in S}) & \{s_j^{new}\}_{j\in[n]} \leftarrow \mathsf{TRes}(tpk, tsk; r), \\ & d \leftarrow \mathsf{TPDec}(tpk, tsk), \\ & c^{new} \leftarrow \mathsf{PKE.Enc}(pk, d; r_c), \\ & \{c_j^{new}\}_{j\in[n]} \leftarrow \mathsf{PKE.Enc}(pk_j, s_j^{new}; r_j) \\ & \text{for } j \in [n] \end{array} \right\}$$

**Protocol 1:** $\Pi_{\mathsf{YOSO\text{-}Re\text{-}encrypt}}$

Let $C_m$ denote the committee that published encryptions of subshares of $tsk$ that were intended for $C_{l,j}$. Let $\pi_{C_{m,j}}$ denote the corresponding proof by party $C_{m,j}$ that everything was done correctly (details below). Finally, let $c^{\mathsf{TK}}_{C_l,i,j}$ denote the encryption of the $i$-th subshare of the share of $tsk$ that is intended for $C_{l,j}$.

$\mathsf{Re\text{-}encrypt}_{C_l}(pk, c)$ **:** Each role $C_{l,i}$ of committee $C_l$ does the following:

1. Reconstruct its key share:
   - Let $S$ denote a set of parties in $C_m$ whose proof $\pi_{C_{m,j}}$ verifies.
   - Decrypt encryptions of the subshares of $tsk$ intended for $C_{l,i}$: $s_{j,i}^{old} \leftarrow \mathsf{PKE.Dec}(sk_{C_{l,i}}, c^{\mathsf{TK}}_{C_l,j,i})$ for each $j \in S$.
   - Reconstruct its key share as $tsk_i \leftarrow \mathsf{TRec}(tpk, \{s_{j,i}^{old}\}_{j\in S})$.
2. Compute the partial decryption as $d_i \leftarrow \mathsf{TPDec}(tpk, tsk_i)$.
3. Sample randomness $r_{enc}$ and encrypt the partial decryption under the given public key $c_i \leftarrow \mathsf{PKE.Enc}(pk, d_i; r_{enc})$.
4. Re-share its key share to the committee $C_k$ that will need it next:
   - Sample randomness $r$ and compute $(s_{i,1}^{new}, \ldots, s_{i,n}^{new}) \leftarrow \mathsf{TRes}(tpk, tsk_i; r)$.
   - Sample randomness $r_j$ and compute $c^{\mathsf{TK}}_{C_k,i,j} \leftarrow \mathsf{PKE.Enc}(pk_{C_{k,j}}, s_{i,j}; r_j)$ for each $j \in [n]$.

- Compute a NIZK proof $\pi_{C_{l,i}}$ that everything was done correctly:

$$\pi_{C_{l,i}} \leftarrow \mathsf{NIZKAoK.P}\left(\begin{array}{l} crs, \\ \phi = (pk_{C_{l,i}}, tpk, \{pk_{C_{k,j}}\}_{j\in[n]}, \{c^{\mathsf{TK}}_{C_k,i,j}\}_{j\in[n]}, \\ \{c^{\mathsf{TK}}_{C_l,j,i}\}_{j\in S}, c_i, c), \\ w = (sk_{C_{l,i}}, tsk_i, r_{enc}, r, \{r_j\}_{j\in[n]}, d, \\ \{s^{new}_{i,j}\}_{j\in[n]}, \{s^{old}_{j,i}\}_{j\in S}) \end{array}\right)$$

5. Broadcast $(c^{\mathsf{TK}}_{C_k,i,1}, \ldots, c^{\mathsf{TK}}_{i,n})$ along with $\pi_{C_l,i}$.
6. Broadcast $c_i$.

We further use the function Decrypt, which is essentially the same as Re-encrypt, except that instead of re-encrypting the obtained share under a new public key, we simply broadcast it in clear. See Figure 3 for details, where the NIZK relation that we prove is the following:

$$R = \left\{ \begin{array}{l|l} \phi = (pk, tpk, \{pk_j\}_{j\in[n]}, & sk \text{ corresponds to } pk, \\ \{c^{new}_j\}_{j\in[n]}, \{c^{old}_j\}_{j\in S}, c^{old}) & s^{old}_j \leftarrow \mathsf{PKE.Dec}(sk, c^{old}_j) \text{ for } j \in S, \\ w = (sk, tsk, r_c, r, \{r_j\}_{j\in[n]}, d, & tsk \leftarrow \mathsf{TKRec}(tpk, \{s^{old}_j\}_{j\in S}), \\ \{s^{new}_j\}_{j\in[n]}, \{s^{old}_j\}_{j\in S}) & \{s^{new}_j\}_{j\in[n]} \leftarrow \mathsf{TRes}(tpk, tsk; r), \\ & d \leftarrow \mathsf{TPDec}(tpk, tsk), \\ & \{c^{new}_j\}_{j\in[n]} \leftarrow \mathsf{PKE.Enc}(pk_j, s^{new}_j; r_j) \\ & \text{for } j \in [n] \end{array} \right\}$$

**Protocol 2:** $\Pi_{\mathsf{YOSO\text{-}Decrypt}}$

Let $C_m$ denote the committee that published encryptions of subshares of $tsk$ that were intended for $C_{l,j}$. Let $\pi_{C_{m,j}}$ denote the corresponding proof by party $C_{m,j}$ that everything was done correctly (details below). Finally, let $c^{\mathsf{TK}}_{C_l,i,j}$ denote the encryption of the $i$-th subshare of the share of $tsk$ that is intended for $C_{l,j}$.

$\mathsf{Decrypt}_{C_l}(c)$ **:** Each role $C_{l,i}$ of committee $C_l$ does the following:

1. Reconstruct its key share:
   - Let $S$ denote a set of parties in $C_m$ whose proof $\pi_{C_{m,j}}$ verifies.
   - Decrypt encryptions of the subshares of $tsk$ intended for $C_{l,i}$: $s^{old}_{j,i} \leftarrow \mathsf{PKE.Dec}(sk_{C_{l,i}}, c^{\mathsf{TK}}_{C_l,j,i})$ for each $j \in S$.
   - Reconstruct its key share as $tsk_i \leftarrow \mathsf{TRec}(tpk, \{s^{old}_{j,i}\}_{j\in S})$.
2. Compute the partial decryption as $d_i \leftarrow \mathsf{TPDec}(tpk, tsk_i)$.
3. Re-share its key share to the committee $C_k$ that will need it next:
   - Sample randomness $r$ and compute $(s^{new}_{i,1}, \ldots, s^{new}_{i,n}) \leftarrow \mathsf{TRes}(tpk, tsk_i; r)$.
   - Sample randomness $r_j$ and compute $c^{\mathsf{TK}}_{C_k,i,j} \leftarrow \mathsf{PKE.Enc}(pk_{C_{k,j}}, s_{i,j}; r_j)$ for each

$j \in [n]$.

- Compute a NIZK proof $\pi_{C_{l,i}}$ that everything was done correctly:

$$\pi_{C_{l,i}} \leftarrow \mathsf{NIZKAoK.P}\left(\begin{array}{l} crs, \\ \phi = (pk_{C_{l,i}}, tpk, \{pk_{C_{k,j}}\}_{j\in[n]}, \{c^{\mathsf{TK}}_{C_k,i,j}\}_{j\in[n]}, \\ \{c^{\mathsf{TK}}_{C_l,j,i}\}_{j\in S}, c), \\ w = (sk_{C_{l,i}}, tsk_i, r_{enc}, r, \{r_j\}_{j\in[n]}, d, \\ \{s^{new}_{i,j}\}_{j\in[n]}, \{s^{old}_{j,i}\}_{j\in S}) \end{array}\right)$$

4. Broadcast $(c^{\mathsf{TK}}_{C_k,i,1}, \ldots, c^{\mathsf{TK}}_{i,n})$ along with $\pi_{C_l,i}$.
5. Broadcast $d_i$.

Finally, we use the following function to have two committees $C_1$ and $C_2$ prepare a Beaver triples. The relation $R$ that the members of the committee $C_2$ have to prove is the following:

$$R = \left\{ \begin{array}{l|l} \phi = (tpk, c^a, c^b_i, c^c_i) & c^b_i \leftarrow \mathsf{TEnc}(tpk, b_i; r_i), \\ w = (b, r) & c^c_i \leftarrow \mathsf{TEval}(tpk, c^a, b_i) \end{array} \right\}$$

**Protocol 3:** $\Pi_{\mathsf{YOSO\text{-}Beaver\text{-}Triples}}$

Let $C_1$ and $C_2$ denote two committees that are participating in the generation of the Beaver triple.

$\mathsf{Beaver\text{-}Triple}_{\boldsymbol{C_1},\boldsymbol{C_2}}$ **:**

- Each role $C_{1,i}$ of committee $C_1$ does the following:
  - Generate a random value $a_i$.
  - Generate randomness $r_i$, and encrypt $a_i$ via $c^a_i \leftarrow \mathsf{TEnc}(tpk, a_{i,j}; r_i)$.
  - Broadcast $c^a_i$ along with a NIZK proof $\pi_{C_1,i}$ that the ciphertext was computed correctly.
- Let $S$ denote a set of roles in $C_1$ whose proof $\pi_{C_1,i}$ verifies. Let $|S|$ denote the size of $S$.
- Everyone can now compute $c^a \leftarrow \mathsf{TEval}(tpk, \{c^a_i\}_{i\in S}, (1)^{|S|})$.
- Each role $C_{2,i}$ of committee $C_2$ does the following:
  - Generate a random value $b_i$.
  - Generate randomness $r_i$, and encrypt $b_i$ via $c^b_i \leftarrow \mathsf{TEnc}(tpk, b_i; r_i)$.
  - Compute $c^c_i \leftarrow \mathsf{TEval}(tpk, c^a, b_i)$

- Compute a NIZK proof $\pi_{C_2,i}$ that everything was done correctly:

$$\pi_{C_{l,i}} \leftarrow \mathsf{NIZKAoK.P}\left(\begin{array}{l} crs, \\ \phi = (tpk, c^a, c_i^b, c_i^c), \\ w = (b_i, r_i) \end{array}\right)$$

- Broadcast $b_i, c_i$ along with the proof $\pi_{C_2,i}$.

- Let $S'$ denote a set of roles in $C_2$ whose proof $\pi_{C_2,i}$ verifies. Let $|S'|$ denote the size of $S'$.
- Everyone can now compute $c^a \leftarrow \mathsf{TEval}(tpk, \{c_i^b\}_{i\in S'}, (1)^{|S'|})$ and $c^c \leftarrow \mathsf{TEval}(tpk, \{c_i^c\}_{i\in S'}, (1)^{|S'|})$
- The resulting triple is $(c^a, c^b, c^c)$.

We now give the full description of the offline phase in Figure 4.

**Protocol 4:** $\Pi_{\mathsf{YOSO\text{-}Offline}}$

**Offline phase:**

Let $C$ denote the circuit.

Step 1: Prepare Beaver Triples Let the first two committees $C_1^{\mathsf{Off}}$ and $C_2^{\mathsf{Off}}$ prepare a Beaver triple for each multiplication gate of the circuit via $\mathsf{Beaver\text{-}Triple}_{C_1,C_2}$.

Step 2: Prepare random wire values

For each wire $\alpha$ of $C$ that is an output wire of an input/multiplication gate:

1. Each $C_{3,i}^{\mathsf{Off}}$ does the following:
   - Generate a random value $\lambda_i^\alpha$.
   - Encrypt $\lambda_i^\alpha$ via $c_i^\alpha \leftarrow \mathsf{TEnc}(tpk, \lambda_i^\alpha)$.
   - Broadcast $c_i^\alpha$ along with a NIZK proof $\pi_{C_{3,i}^{\mathsf{Off}}}$ that the ciphertext was computed correctly.
2. Let $S$ denote the set of roles $C_{3,i}^{\mathsf{Off}}$ whose proofs verified correctly.
3. Everyone computes $c^\alpha \leftarrow \mathsf{TEval}(tpk, \{c_i^\alpha\}_{i\in S}, (1)^{|S|})$.

Step 3: Compute dependent wire values

For each *addition gate* with inputs wires $\alpha$ and $\beta$, and output wire $\gamma$, set (from lowest depth gates to highest):

- $c^\gamma \leftarrow \mathsf{TEval}(tpk, (c^\alpha, c^\beta), (1, 1))$.

For each *multiplication gate* with inputs wires $\alpha$ and $\beta$, and output wire $\gamma$, compute the encryption $c^{\Gamma^\gamma}$ of the value $\lambda^\alpha * \lambda^\beta - \lambda^\gamma$ on wire $\gamma$. For this, let $(c^x, c^y, c^z)$ denote an unused Beaver triple (from the ones prepared in Step 1). Then, compute the following for groups of $k$ multiplication gates in parallel:

- Everyone computes $c^\epsilon \leftarrow \mathsf{TEval}(tpk, (c^\alpha, c^x), (1, 1))$ and

$c^{\delta} \leftarrow \mathsf{TEval}(tpk, (c^{\beta}, c^{y}), (1, 1))$.

- Current committee $C_l^{\mathsf{Off}}$ decrypts $\epsilon = \mathsf{Decrypt}_{C_l^{\mathsf{Off}}}(c^{\epsilon})$ and $\delta = \mathsf{Decrypt}_{C_l^{\mathsf{Prep}}}(c^{\delta})$.
- Everyone computes $c^{\Gamma^{\gamma}} \leftarrow \mathsf{TEval}(tpk, (c^{\beta}, c^{x}, c^{z}, c^{\gamma}), (\epsilon, -\delta, 1, -1))$.

Step 4: Pack values for multiplication gates

For each group of $k$ multiplication gates, let $(c^{\alpha_1}, \ldots, c^{\alpha_k})$ denote the ciphertexts containing the values on the corresponding output wires. Each role $C_{l,i}^{\mathsf{Off}}$ of the current committee $C_l^{\mathsf{Off}}$ does the following:

- Generate $t$ random values $(r_i^{k+1}, \ldots, r_i^{t+k})$.
- Let $c_i^{\mathsf{Help},\alpha_j} \leftarrow \mathsf{TEnc}(tpk, r_i^j)$ for $j \in [k+1, \ldots, t+k]$
- Broadcast $\{c_i^{\mathsf{Help},\alpha_j}\}_{j \in [k+1,\ldots,t+k]}$ along with a NIZK proof $\pi_{C_{l,i}^{\mathsf{Off}}}$ that the ciphertext was computed correctly.

Now, everyone does the following:

- Let $S$ denote the set of roles $C_{l,i}^{\mathsf{Off}}$ whose proofs verified correctly.
- Compute $c^{\mathsf{Help},\alpha_j} \leftarrow \mathsf{TEval}(tpk, \{c_i^{\mathsf{Help},\alpha_j}\}_{i \in S}, (1)^{|S|})$ for each $j \in [k+1, t+k]$.
- For each $i \in [n]$, compute the share of the $i$-th committee role as:

$$c_i^{\boldsymbol{\alpha}} \leftarrow \mathsf{TEval}(tpk, (c^{\alpha_1}, \ldots, c^{\alpha_k}, c^{\mathsf{Help},\alpha_{k+1}}, \ldots, c^{\mathsf{Help},\alpha_{t+k}}), (l_1(i), \ldots, l_{t+k}(i))),$$

  where $l_j(\cdot)$ denotes the $j$-th Lagrange basis polynomial.

Similarly, for each group of $k$ multiplication gates, pack the ciphertexts containing $c^{\Gamma^{\gamma}}$.

Step 5: Prepare ciphertexts to the future, input gates

For each input gate $\alpha$ that belongs to the $i$-th client, let $c^{\alpha}$ denote the ciphertext containing the value on the corresponding output wire. Current committee $C_l^{\mathsf{Off}}$ computes the following for all $i \in [n]$ in parallel:

- $c_{P_i}^{\alpha} \leftarrow \mathsf{Re\text{-}encrypt}_{C_l^{\mathsf{Off}}}(pk_{P_i}^{\mathsf{KFF}}, c^{\alpha})$

Step 6: Prepare ciphertexts to the future, multiplication gates

For each group of $k$ multiplication gates, let $c_i^{\boldsymbol{\alpha}}$ denote the encryption of the $i$-th (packed) share of the left input wires, $c_i^{\boldsymbol{\beta}}$ the encryption of the $i$-th (packed) share of the right input wires, and $c_i^{\boldsymbol{\Gamma}^{\gamma}}$ denote the encryption of $i$-th (packed) share of the values $\lambda_{\boldsymbol{\alpha}} * \lambda_{\boldsymbol{\beta}} - \lambda_{\boldsymbol{\gamma}}$ for the vector of corresponding output wires $\boldsymbol{\gamma}$. Let $C_m$ denote the committee that will be evaluating gates $\gamma$ in the online phase. Current committee $C_l^{\mathsf{off}}$ computes the following for all $i \in [n]$ in parallel:

- $c_{C_{m,i}}^{\boldsymbol{\alpha}} \leftarrow \mathsf{Re\text{-}encrypt}_{C_l^{\mathsf{off}}}(pk_{C_{m,i}}^{\mathsf{KFF}}, c_i^{\boldsymbol{\alpha}})$
- $c_{C_{m,i}}^{\boldsymbol{\beta}} \leftarrow \mathsf{Re\text{-}encrypt}_{C_l^{\mathsf{off}}}(pk_{C_{m,i}}^{\mathsf{KFF}}, c_i^{\boldsymbol{\beta}})$
- $c_{C_{m,i}}^{\boldsymbol{\Gamma}^{\gamma}} \leftarrow \mathsf{Re\text{-}encrypt}_{C_l^{\mathsf{Off}}}(pk_{C_{m,i}}^{\mathsf{KFF}}, c_i^{\boldsymbol{\Gamma}^{\gamma}})$

**Communication analysis.**

In Step 1, we need $O(n)$ communication to prepare a Beaver triple [2]. We require $O(n)$ communication to prepare each random wire value in Step 2. Next, Step 3 requires $\frac{O(n^2)}{k} = O(n)$ communication per gate. In Step 4, the communication cost is again $\frac{O(n^2)}{k} = O(n)$ per gate. Re-encrypting values in Steps 5 and 6 each requires $O(n)$ communication per re-encrypted value, i.e., per wire, plus $O(n^2)$ one-time cost. In total, the communication complexity of our offline phase is $O(n|C|)$.

### 3.4.3 Online Phase

During the online phase, our goal is to compute values $\mu_\alpha = v_\alpha - \lambda_\alpha$ for each wire of the circuit, where $v_\alpha$ is the actual wire value, and $\lambda_\alpha$ the random value prepared for this wire during the offline phase. To do this, we must first let the roles of the online committees obtain the random values prepared for these roles during the preprocessing. Note that during the offline phase, we were encrypting the secret random values under the KFFs of the corresponding roles' of the online phase. Hence, it is sufficient to let the online roles learn the secret key portion of their corresponding KFF. For this, we have parties of the first online committee use their shares of $tsk$ to re-encrypt the secret portions of the KFFs towards the *role keys of the YOSO role-generation*, which can now safely assumed to be known.

For the input gates, given input $v_\alpha$ for the input wire $\alpha$, each client can simply use its secret KFF to decrypt the value $\lambda_\alpha$ and broadcast $v_\alpha - \lambda_\alpha$.

For the addition gates, given input wires $\alpha$ and $\beta$, by invariant we know both $\mu_\alpha$ and $\mu_\beta$. Hence, everyone can locally compute $\mu_\alpha + \mu_\beta$.

For the multiplication gates, given input wire vectors $\boldsymbol{\alpha}$ and $\boldsymbol{\beta}$, and output wire vector $\boldsymbol{\gamma}$, the $i$-th member of the current committee can use the packed shares of $\lambda_i^{\boldsymbol{\Gamma}^\gamma}$ which it has from the offline phase, along with the publicly reconstructed $\boldsymbol{\mu_\alpha}$ and $\boldsymbol{\mu_\beta}$, to obtain $\mu_i^\gamma = \mu_i^{\boldsymbol{\alpha}} * \mu_i^{\boldsymbol{\beta}} + \mu_i^{\boldsymbol{\alpha}} * \lambda_i^{\boldsymbol{\beta}} + \mu_i^{\boldsymbol{\beta}} * \lambda_i^{\boldsymbol{\alpha}} + \lambda_i^{\boldsymbol{\Gamma}^\gamma}$.

Finally, for each output wire $\alpha$, we let the last committee use its shares of $tsk$ to re-ecrypt the preprocessed value $\lambda^\alpha$ towards the public key of the client $P_i$, who is supposed to get the output of this wire. Then, $P_i$ can use the publicly available $\mu^\alpha$ to reconstruct $v^\alpha$ as $\mu^\alpha + \lambda^\alpha$.

[2]Note that our offline phase is bottlenecked by a step which requires linear communication, hence for simplicity we skip potential optimizations that could amortize the cost of non-bottleneck steps to a constant one.

**Protocol 5:** $\Pi_{\mathsf{YOSO-Online}}$

Let $pk_{P_i}$ denote public key of the client $P_i$, and $pk_{C_{l,i}}$ denote YOSO role-assignment key of the committee member $C_{l,i}$.

**Online phase:**

Future key distribution

The first online committee $C_1$ computes the following for all clients and all future committee roles $C_l[i]$ in parallel:

- $c_{P_i}^{\mathsf{KFF'}} \leftarrow \mathsf{Re\text{-}encrypt}_{C_1}(pk_{P_i}, c_{P_i}^{\mathsf{KFF}})$
- $c_{C_{l,i}}^{\mathsf{KFF'}} \leftarrow \mathsf{Re\text{-}encrypt}_{C_1}(pk_{C_{l,i}}, c_{C_{l,i}}^{\mathsf{KFF}})$

Input

For each input gate that belong client $P_i$, let $c_{P_i}^{\alpha}$ denote the ciphertext containing the prepared random value on the corresponding output wires. Each client $P_i$ uses its secret role key $sk_{P_i}$ from the YOSO role-assignment to compute the following:

- Let $sk_{P_i}^{\mathsf{KFF}} \leftarrow \mathsf{Dec}(sk_{P_i}, c_{P_i}^{\mathsf{KFF'}})$
- Let $\lambda^{\alpha} \leftarrow \mathsf{Dec}(sk_{P_i}^{\mathsf{KFF}}, c_{P_i}^{\alpha})$
- Compute and publish $\mu^{\alpha} = v^{\alpha} - \lambda^{\alpha}$

Addition

For two addition gates with input wires $\alpha$ and $\beta$, everyone can locally compute $\mu^{\alpha} + \mu^{\beta}$.

Multiplication

Let $C_l$ denote the current committee. For each group of multiplication gates with input wires $\boldsymbol{\alpha}, \boldsymbol{\beta}$ and output wires $\boldsymbol{\gamma}$, let $c_{C_{l,i}}^{\boldsymbol{\alpha}}, c_{C_{l,i}}^{\boldsymbol{\beta}}, c_{C_{l,i}}^{\boldsymbol{\Gamma^{\gamma}}}$ denote the ciphertexts containing the share prepared for the $i$-th role of $C_l$. Let $\boldsymbol{\mu_{\alpha}}$ and $\boldsymbol{\mu_{\beta}}$ denote the publicly reconstructed values on wires $\boldsymbol{\alpha}$ and $\boldsymbol{\beta}$. Let $\mu_i^{\boldsymbol{\alpha}}$ and $\mu_i^{\boldsymbol{\beta}}$ denote the $i$-th share of a degree-$(k-1)$ packed sharing of each of this vectors. Each member of the current committee uses its secret role key $sk_{C_{l,i}}$ from the YOSO role-assignment to compute the following:

- Let $sk_{C_{l,i}}^{\mathsf{KFF}} \leftarrow \mathsf{Dec}(sk_{C_{l,i}}, c_{C_{l,i}}^{\mathsf{KFF'}})$
- Let $\lambda_i^{\boldsymbol{\alpha}} \leftarrow \mathsf{Dec}(sk_{C_{l,i}}^{\mathsf{KFF}}, c_{C_{l,i}}^{\boldsymbol{\alpha}})$
- Let $\lambda_i^{\boldsymbol{\beta}} \leftarrow \mathsf{Dec}(sk_{C_{l,i}}^{\mathsf{KFF}}, c_{C_{l,i}}^{\boldsymbol{\beta}})$
- Let $\lambda_i^{\boldsymbol{\Gamma^{\gamma}}} \leftarrow \mathsf{Dec}(sk_{C_{l,i}}^{\mathsf{KFF}}, c_{C_{l,i}}^{\boldsymbol{\Gamma^{\gamma}}})$
- Compute and publish $\mu_i^{\boldsymbol{\gamma}} = \mu_i^{\boldsymbol{\alpha}} * \mu_i^{\boldsymbol{\beta}} + \mu_i^{\boldsymbol{\alpha}} * \lambda_i^{\boldsymbol{\beta}} + \mu_i^{\boldsymbol{\beta}} * \lambda_i^{\boldsymbol{\alpha}} + \lambda_i^{\boldsymbol{\Gamma^{\gamma}}}$ along with the proof of correctness.

In order to reconstruct $\boldsymbol{\mu^{\gamma}}$, anyone can use $t + 2(k-1) + 1$ shares $\boldsymbol{\mu}_i^{\boldsymbol{\gamma}}$ that verified correctly.

Output

The last committee $C_l$ uses its shares of $tsk$ to re-encrypt (in parallel) each value $c^{\alpha}$ towards the client $P_i$ who is obtaining the output of wire $\alpha$.

- $c_{P_i}^{\alpha} \leftarrow \mathsf{Re\text{-}encrypt*}_{C_1}(pk_{P_i}, c^{\alpha})$

Above, Re-encrypt* is the same as Re-encrypt, except that we do not distribute shares of

$tsk$ anymore. This reduces the overall communication per output wire to $O(n)$.
To obtain the output for a wire $\alpha$, client $P_i$ uses $sk_{P_i}$ to decrypt the ciphertext $c^{\alpha}_{P_i}$ containing the corresponding random wire value. Then, the client computes the output using the public value $\mu_\alpha$:

- Let $\lambda^\alpha \leftarrow \mathsf{Dec}(sk^{\mathsf{KFF}}_{P_i}, c^{\alpha}_{P_i})$.
- Compute $v^\alpha = \mu^\alpha + \lambda^\alpha$.

**Communication analysis.** The future key distribution step results in $O(n)$ communication per role, as each committee member must publish its share of the value that is being re-encrypted, plus a one-time cost of $O(n^2)$ for the re-distribution of the shares of $tsk$. Further, the input step requires a one-time cost that is linear in the number of inputs. Addition gates do not require any communication, and a batch of $O(n)$ multiplication gates requires each committee member to publish its share of $\boldsymbol{\mu}^\gamma$ along with a proof of correctness, and thus incurs $O(n)$ cost per batch. Finally, the output layer requires $O(n)$ communication per output wire. Assuming that each role processes $O(n)$ values, this gives us $O(1)$ amortized communication per gate.

This construction allows us to obtain the following result:

**Theorem 6.** *Assuming a secure broadcast and role-assignment, for any n-party function $f$, protocol $\Pi = (\mathsf{YOSO\text{-}Setup}, \Pi_{\mathsf{YOSO\text{-}Offline}}, \Pi_{\mathsf{YOSO-Online}})$, YOSO-securely implements the ideal functionality $\mathcal{F}_M PC(f)$ for a corruption threshold $t < \frac{n}{2} \cdot (1-\epsilon)$. The offline communication complexity is $O(n)$ elements per gate, the online communication complexity is $O(1)$ elements per gate.*

### 3.4.4 Security Analysis

In the following, we describe the hybrid games which lead to a simulator description for the protocol given above.

**Game 0:** The simulator honestly follows the protocol, except that during the Setup phase the simulator additionally stores the NIZKAoK trapdoor $\mathsf{td}$.

**Game 1:** The simulator behaves as before, except that it uses the NIZKAoK simulator $\mathsf{SimP}$ to simulate the proofs of the honest roles'. Note that this game is indistinguishable from the previous one by the zero knowledge property of NIZKAoK system that we use.

**Game 2:** The simulator behaves as before, except that it now extracts a witness from each proof that was given by a corrupt party. The simulator aborts if it fails to extract a

valid witness for a proof that verifies correctly. Note that this game is indistinguishable from the previous one by the simulation extractability property of the NIZKAoK.

Note that now, the simulator in particular always knows the shares of $tsk$ (even for corrupt parties), along with the plaintexts inside the ciphertexts distributed during the offline phase.

**Game 3:** Let $M$ denote the set of currently corrupt roles. The simulator behaves as before, except that during the Re-encrypt* step of computing the output, for each output wire $\alpha$ that belongs to a malicious client, the simulator does the following:

- Let $c^\alpha$ denote the ciphertext that contains the value $\lambda^\alpha$ encrypted under $tpk$.
- The simulator computes $\lambda^\alpha$.
- The simulator computes the decryption shares of the corrupt roles as $d_i \leftarrow \mathsf{TPDec}(tpk, tsk_i, c^\alpha)$.
- Compute the decryption shares of the honest roles as $\{d_i\}_{i\in[n]\setminus M} \leftarrow \mathsf{SimTPDec}(tpk, c^\alpha, \lambda^\alpha, \{tsk_i\}_{i\in[n]\setminus M}, \{d_i\}_M)$

Note that the indistinguishability to the previous game holds by the partial decryption simulatability property of the threshold encryption scheme.

**Game 4:** Let $M$ denote the set of the currently corrupt roles. The simulator behaves as before, except that for each output wire $\alpha$ that belongs to a malicious client, the simulator does the following:

- Let $c^\alpha$ denote the ciphertext that contains the value $\lambda^\alpha$ encrypted under $tpk$.
- The simulator computes the values $v^\alpha$ of the corrupt parties using its knowledge of $\lambda^\alpha$, sends these values to the ideal functionality $\mathcal{F}_{\mathsf{MPC}}F$, obtains $v^\alpha$ from it, and computes $\lambda^\alpha = \mu^\alpha - v^\alpha$.
- The simulator computes the decryption shares of the corrupt roles as $d_i \leftarrow \mathsf{TPDec}(tpk, tsk_i, c^\alpha)$.
- Compute the decryption shares of the honest roles as $\{d_i\}_{i\in[n]\setminus M} \leftarrow \mathsf{SimTPDec}(tpk, c^\alpha, \lambda^\alpha, \{tsk_i\}_{i\in[n]\setminus M}, \{d_i\}_M)$

Note that the indistinguishability to the previous game holds by the partial decryption simulatability property of the threshold encryption scheme.

**Game 5:** Let $M$ denote the set of the currently corrupt roles. The simulator behaves as before, except that for each input wire $\alpha$ that belongs to an honest client, the simulator does the following:

- Let $c^\alpha$ denote the ciphertext that contains the value $\lambda^\alpha$ encrypted under $tpk$.
- The simulator generates a fresh random value $\hat{\lambda}^\alpha$.
- The simulator computes the decryption shares of the corrupt roles as $d_i \leftarrow \mathsf{TPDec}(tpk, tsk_i, c^\alpha)$.

- Compute the decryption shares of the honest roles as
  $\{d_i\}_{i\in[n]\setminus M} \leftarrow \mathsf{SimTPDec}(tpk, c^\alpha, \hat{\lambda}^\alpha, \{tsk_i\}_{i\in[n]\setminus M}, \{d_i\}_M)$

Note that the indistinguishability to the previous game holds as $\hat{\lambda}^\alpha$ and $\lambda^\alpha$ are both uniformly random, and the threshold encryption scheme has the partial decryption simulatability property.

**Game 6:** The simulator behaves as before, except that when an honest client needs to publish $\mu^\alpha$, the simulator simply samples a random value $\mu^\alpha$, and publishes it instead. Note that $\mu^\alpha = v^\alpha - \lambda^\alpha$, and from the previous game we know that $\lambda^\alpha$ is uniformly random. Hence, $\mu^\alpha$ is uniformly random and thus the distribution of $\mu^\alpha$ remains the same as before.

Note that in our last game, the simulator no longer needs the honest input roles' inputs. This finishes our proof.

### 3.4.5 Bonus: Supporting Fail-Stop Parties

We observe that, in our protocol, the online phase requires at least $t + 2(k-1) + 1$ partial decriptions to be posted. There are $n - t$ honest parties, so we need to ensure that $n - t \geq t + 2(k-1) + 1$ for GOD, or $n > 2t + 2(k-1)$. Assuming $t < n(\frac{1}{2} - \epsilon)$, this is equivalent to $n \geq n(1-2\epsilon) + 2(k-1)$, or $k - 1 \leq n\epsilon$ This allows us to get a saving factor in the online phase of $k \approx n\epsilon$.

However, note we can do the following. Write $\epsilon = 2\epsilon'$, and set $k = n\epsilon' + 1$. We have that $t + 2(k-1) + 1 < n(\frac{1}{2} - \epsilon) + 2(n\epsilon') + 1 = n/2 + 1$. On the other hand, the amount of honest parties is $n - t > n - n(\frac{1}{2} - \epsilon) = n(\frac{1}{2} + \epsilon)$, which is at least $n\epsilon$ more than the required amount of parties for reconstruction. We conclude the following: by reducing the packing parameter from $\approx n\epsilon$ to $\approx n\epsilon/2$, the protocol is able to proceed, even if $n \cdot \epsilon$ *honest parties* do not participate. We believe this property is crucial for YOSO MPC protocols, which are intended to be deployed in settings with large number of parties and hence are prone to non-malicious failures such as slowdowns, hardware/software errors, and others.

## 3.5 Role Assignment: Committee Size Analysis

In previous sections we have described a YOSO MPC protocol with high efficiency, assuming that the committee sizes $n$ are such that the amount of corrupted parties in each committee is upper bounded by $t < n(\frac{1}{2} - \epsilon)$. Ensuring this corruption threshold is in charge of the

role assignment layer, which is generally a separate task to that of YOSO MPC. Different works have proposed different role assignment layers for YOSO [21], but generally, these approaches focus on obtaining committees with 1/2 corruption ratios. In this section we make use of the analysis from [21] in order to obtain the stronger honest majority bound $t < n(\frac{1}{2} - \epsilon)$. Briefly recall that Benhamouda *et al.*'s role assignment works as follows: to select members of the MPC committee, a so-called *nominating committee* is first formed by self-selection via a process known as *cryptographic sortition*. Each nominator then chooses a member of the MPC committee, and generates short-term public/secret key pair $(\mathsf{epk}, \mathsf{esk})$. The nominator encrypts $\mathsf{esk}$ under the chosen party's long-term key to obtain a ciphertext $c$, and publishes the pair $(\mathsf{epk}, c)$. Finally, each party can attempt to decrypt the ciphertexts $c$ to check whether they were selected to participant in the MPC committee or not. We refer the reader to [21] for details.

For this section, we stick to the notation from [21, Section 3.2], so that we can easily reuse their analysis. Let $N$ be the total number of parties, among which committees will be sampled, and let us assume that the adversary corrupts $f \cdot N$ out of these $N$ parties. Cryptographic sortition is a *probabilistic* process that samples committees by including each party in the committee with some probability $C/N$.[3] Let $c$ be the random variable denoting the size of the committee, and let $\phi$ be the random variable denoting the number of corrupted parties in the selected committee. Our goal is to determine a threshold $t$ so that, with high probability, $\phi < t$ and $t \leq c \cdot (\frac{1}{2} - \epsilon)$. Note that $t$ in [21] has a different connotation than in our previous sections: here, $t - 1$ corresponds an upper bound (with high probability) for the number of corruptions in a committee. This is roughly equivalent to what we refer to as "$t$" in prior sections.

Let $k_1$, $k_2$ and $k_3$ be three security parameters for the analysis, as follows.

1. The adversary can try to win the cryptographic sortition at most $2^{k_1}$ times
2. We want to ensure that $\phi < t$ with probability $\geq 1 - 2^{k_2}$
3. We want to ensure that $t \leq c \cdot (\frac{1}{2} - \epsilon)$ with probability $\geq 1 - 2^{k_3}$.

In [21], the threshold $t$ is written as $t = B_1 + B_2 + 1$, where $B_1 = fC(1 + \epsilon_1)$ and $B_2 = f(1 - f)C(1 + \epsilon_2)$, for some $\epsilon_1, \epsilon_2 > 0$. In [21] it is shown that, if

$$C > \max\left\{ \frac{(k_1 + k_2 + 1)(2 + \epsilon_1)\ln 2}{f\epsilon_1^2},\ \frac{(k_2 + 1)(2 + \epsilon_2)\ln 2}{f(1 - f)\epsilon_2^2} \right\}, \tag{3.2}$$

then Item 2 from above holds, that is, the number of corruptions $\phi$ among the elected committee is at most $t$, except with probability $2^{-k_2}$.

[3] Note that, even though the *expected size* of the committee is $C$, its concrete size is variable.

Regarding Item 3, we note that $t < c \cdot (\frac{1}{2} - \epsilon)$ is equivalent to the number of honest parties $c - t$ being greater than $\frac{1/2+\epsilon}{1/2-\epsilon} \cdot t$. In [21] this is shown to hold with probability $\geq 1 - 2^{k_3}$ if, for some $\epsilon_3 > 0$:

$$C > \max\left\{\frac{2k_3 \ln 2}{(\epsilon_3(1-f))^2},\ \frac{(\frac{1}{2}+\epsilon)\cdot(fC(1+\epsilon_1)+f(1-f)C(1+\epsilon_2))}{(\frac{1}{2}-\epsilon)(1-f)^2(1-\epsilon_3)}\right\} \tag{3.3}$$

Interestingly, this is a generalization of [21], with their set of constraints being achieved by setting $\epsilon = 0$. We see then that the only difference with respect to their work is the factor $\frac{1/2+\epsilon}{1/2-\epsilon}$ in Eq. (3.3). That is, the cost to pay to enable the gap $\epsilon > 0$ is reflected in this term.

In what follows, let us fix as in [21] $k_1 = 64$, $k_2 = k_3 = 128$. Set $\epsilon_1$ and $\epsilon_2$ as small as possible so that Eq. (3.2) holds. Using numerical methods we find this is

$$\epsilon_1 > \frac{1}{2}\sqrt{\frac{1544Cf\ln(2) + 37249\ln^2(2)}{C^2f^2}} + \frac{193\ln(2)}{2Cf} \tag{3.4}$$

and

$$\epsilon_2 > \frac{1}{2}\sqrt{\frac{-1032Cf^2\ln(2) + 1032Cf\ln(2) + 16641\ln^2(2)}{C^2(f^2-f)^2}} - \frac{129\ln(2)}{2C(f^2-f)}. \tag{3.5}$$

Note these values—which determine $t$—only depend on $C$ and $f$. Now, let $\epsilon_3$ to be the smallest value that satisfies (3.3), we have:

$$\sqrt{\frac{256\ln 2}{C(1-f)^2}} < \epsilon_3 < 1 - \left(\frac{\frac{1}{2}+\epsilon}{\frac{1}{2}-\epsilon}\right)\cdot\frac{(fC(1+\epsilon_1)+f(1-f)C(1+\epsilon_2))}{(1-f)^2C} \tag{3.6}$$

Let us denote $\delta = \frac{\frac{1}{2}+\epsilon}{\frac{1}{2}-\epsilon}$, which satisfies $1 \leq \delta$ for $0 < \epsilon < 1/2$. Note that the most efficient choice is to take $\epsilon_1$ and $\epsilon_2$ as small as possible, according to Eqs. (3.4) and (3.5). Then, set $\epsilon_3$ as small as possible according to Eq. (3.6). At this point, $\delta > 1$ must satisfy the right inequality of Eq. (3.6). The work of [21] took $\delta = 1$, for which $\epsilon = 0$. Here, we note that we can take some $\delta > 1$, which corresponds to $\epsilon > 0$, as long as the inequality still holds.

In Table 3.1 we present a selection of parameters obtained with the reasoning above. We choose the sortition parameter $C$ in $\{1000, 5000, 10000, 20000, 40000\}$ (which is the *expected* size of the committee), and the global corruption ratio $f$ in $\{0.05, 0.10, 0.15, 0.20, 0.25\}$. Then, we compute $\epsilon_1, \epsilon_2, \epsilon_3$ as described above. This determines $t$, where $t-1$ is (w.h.p.) an upper bound the number of corruptions in the committee, and along with it $c = t/(1/2 - \epsilon)$

is determined, which is (w.h.p.) a lower bound on the size of the committee. Note that the lower bound that is guaranteed (w.h.p.) in [21] is $c' = 2t$. However, our analysis shows that the committee size is actually larger than that, and that gap can be used to gain a $k = n \cdot \epsilon$ improvement factor. Overall, we see that by increasing the committee size from $c'$ to $c$, which as $f$ grows this becomes a less pronounced jump, we can actually reduce communication by a factor of $k$, which can be as big as three orders of magnitude! For instance, setting $C = 20000$, for 20% global corruptions, we can take a committee of size $\approx$ 20k instead of $\approx$ 18k from [21] and get an improvement factor in terms of online communication complexity of $> 1000\times$. For smaller $f$ this factor improves even more, at the expense of having a bigger difference in committee sizes. This disparity can be reduced by settling for a smaller packing factor.

## 3.6 Summary

In this work we have shown that YOSO MPC can benefit greatly from transitioning from a setting where $t < n/2$, to $t < n(1/2 - \epsilon)$. In large-scale scenarios such as YOSO it is reasonable to assume the adversary corrupts strictly less than a minority, and such "gap" can be exploited in order to get concrete benefits in terms of online communication, as well as fail-stop tolerance. Our work shows how packed secret-sharing can be used to materialize such communication savings. Furthermore, we have shown that requiring such gap does not substantially affect the size of elected committees.

Our work serves as a starting point to fully unleash the practicality of YOSO MPC. Relevant follow-up works include the following:

- Instantiating our framework with class groups-based solutions such as [30, 36], which remove the trusted setup. We did not follow this approach in our work since (1) it is simpler in terms of exposition to consider the original presentation from [55], and (2) class group-based solutions have a $n!$ overhead in terms of communication due to the use of integer secret-sharing, which may not be suitable for large-party settings.
- We adapted the role assignment from [21], showing that demanding for a gap does not affect committee sizes drastically (specially given the benefits). It remains to be seen what is the impact of this condition in more recent role assignment protocols such as [31].
- As a feasibility result, it is interesting to explore what the impact of the "gap" is in the context of the *information-theoretic security*, where no computational assumptions are used at the protocol level.

- Our preprocessing unfortunately does not benefit from the packing parameter $k$. This is an inherent limitation of Turbopack[50], and we find it highly relevant to remove such limitation.

| $C$ | $f$ | $t$ | $c$ | $c'$ | $\epsilon$ | $k$ |
|---|---|---|---|---|---|---|
| | 0.05 | 446 | 949 | 893 | 0.03 | 28 |
| | 0.1 | $\perp$ | $\perp$ | $\perp$ | $\perp$ | $\perp$ |
| 1000 | 0.15 | $\perp$ | $\perp$ | $\perp$ | $\perp$ | $\perp$ |
| | 0.2 | $\perp$ | $\perp$ | $\perp$ | $\perp$ | $\perp$ |
| | 0.25 | $\perp$ | $\perp$ | $\perp$ | $\perp$ | $\perp$ |
| | 0.05 | 1078 | 4699 | 2157 | 0.27 | 1271 |
| | 0.1 | 1721 | 4925 | 3444 | 0.15 | 741 |
| 5000 | 0.15 | 2293 | 5106 | 4588 | 0.05 | 259 |
| | 0.2 | $\perp$ | $\perp$ | $\perp$ | $\perp$ | $\perp$ |
| | 0.25 | $\perp$ | $\perp$ | $\perp$ | $\perp$ | $\perp$ |
| | 0.05 | 1754 | 9518 | 3509 | 0.32 | 3004 |
| | 0.1 | 2937 | 9841 | 5876 | 0.20 | 1982 |
| 10000 | 0.15 | 4004 | 10098 | 8009 | 0.10 | 1045 |
| | 0.2 | 4983 | 10319 | 9968 | 0.02 | 175 |
| | 0.25 | $\perp$ | $\perp$ | $\perp$ | $\perp$ | $\perp$ |
| | 0.05 | 2998 | 19264 | 5998 | 0.34 | 6633 |
| | 0.1 | 5216 | 19723 | 10433 | 0.24 | 4645 |
| 20000 | 0.15 | 7237 | 20088 | 14476 | 0.14 | 2806 |
| | 0.2 | 9107 | 20401 | 18215 | 0.05 | 1093 |
| | 0.25 | $\perp$ | $\perp$ | $\perp$ | $\perp$ | $\perp$ |
| | 0.05 | 5331 | 38907 | 10664 | 0.36 | 14121 |
| | 0.1 | 9552 | 39558 | 19106 | 0.26 | 10226 |
| 40000 | 0.15 | 13437 | 40074 | 26875 | 0.16 | 6600 |
| | 0.2 | 17047 | 40517 | 34096 | 0.08 | 3211 |
| | 0.25 | 20408 | 40911 | 40818 | 0.01 | 47 |

**Table 3.1: Sample parameters.** $C$ is the parameter for the cryptographic sortition, that is, each party from the global pool is chosen with prob. $C/N$. $f$ is the global corruption ratio, that is, there are $f \cdot N$ corrupt parties among the global $N$ parties. $t$ is the threshold that upper-bounds the number of corruptions (plus one) in the committee (w.h.p.). $c = t/(\frac{1}{2} - \epsilon)$ is the lower bound on the committee size (w.h.p.), and $c' = 2t$ is the lower bound on the committee size if one takes $\epsilon = 0$ (w.h.p.). $\epsilon$ is the gap. $k$ is the packing factor. $\perp$ means that the given ratio $f$ is impossible for the given value of $C$.

# Chapter 4

# Improved YOSO Randomness Generation with Worst-Case Corruptions

> *Another way to be prepared is to think negatively.*
> *Yes, I'm a great optimist. but, when trying to make a decision, I often think of the worst case scenario. I call it "the eaten by wolves factor." If I do something, what's the most terrible thing that could happen? Would I be eaten by wolves?*
> *One thing that makes it possible to be an optimist, is if you have a contingency plan for when all hell breaks loose.*
>
> Randy Pausch, The Last Lecture

We conclude our quest of improving the state of the art in stateless MPC by focusing on a specialized multi-party functionality, the task of distributed randomness generation. We design protocols for this functionality in the recent model of YOSO with *worst-case corruptions* [92], which is tailored to the problem of randomness generation. Prior work in this model provided feasibility results in the information-theoretic setting [92]. Unfortunately, the corresponding constructions were inefficient, and in particular, required exponential communication complexity. In our work, we switch to the computational setting and provide efficient protocols which also have good resiliency. We further complement our results by lower bounds in terms of round-complexity in the setting without setup and give a protocol which matches these lower bounds (in contrast to our efficient protocols, this one requires exponential communication complexity).

The work presented in this chapter is based on two joint works with Chen-Da Liu-Zhang, João Ribeiro, Pratik Soni, and Sri AravindaKrishnan Thyagarajan. The first of these works has been published at FC 2024 [88]. As one of the main authors, I contributed to the design and security analysis of most of our protocols and lower bounds.

## 4.1 Introduction

Public randomness is a fundamental component of numerous financial and security protocols [81, 96]. Randomness usage is ubiquitous: From establishing fairness in the green card lottery, to assessing risk via Monte Carlo simulations, to generating the public parameters for the cryptographic protocols [13, 86]. In the past, public randomness was typically obtained via trusted third parties. However, with the emergence of blockchains and web3, there has been an increased effort to decentralize economic activities, and as a consequence, to decentralize public randomness generation as well [34, 35, 40, 69, 101].

A protocol for such distributed public randomness allows multiple mutually distrusting parties, each with their own source of randomness, to generate and agree on a public random value. However, designing a secure protocol which provides such a functionality is a notoriously hard task. Indeed, the cryptographic community put significant effort into designing distributed randomness generation protocols [34, 35, 40, 69, 101], as well as improving functionalities such as verifiable delay functions [27] and time-lock puzzles [102], which oftentimes serve as building blocks in such protocols.

Traditionally, those protocols consider *static* adversaries, where security is guaranteed as long as the adversary decides which parties to corrupt prior to the start of the execution. However, such an assumption seems unjustified, especially for protocols that run over long periods of time. A far more realistic setting would allow the adversary to corrupt parties *dynamically* during the course of the execution. This gave rise to a line of *adaptively-secure* protocols that are built out of ephemeral one-time roles (e.g., [24, 39, 90, 94]), mostly focused on agreement primitives. In the context of general multi-party computation, such protocols have been recently proposed in the YOSO line of work.

Typically, the protocols designed in the YOSO setting rely on the fact that the adversary's best option is to corrupt machines *at random*. However, this assumption is viable only if role-assignment mechanism is truly secure. This makes role-assignment protocols hard to design, and the currently known constructions compromise either in terms of efficiency [55] or in terms of the supported corruption threshold [21]. In order to reduce trust in role-

assignment, Nielsen, Ribeiro, and Obremski (NRO in the following) recently introduced a model for YOSO with *worst-case corruptions* [92], which we dub YOSO$^{\mathsf{WCC}}$. In this model, prior to the start of the protocol, the adversary can choose any up to $t$ roles to corrupt overall *across* all participating parties. The YOSO$^{\mathsf{WCC}}$ model is tailored to the randomness generation setting, and the authors introduce two information-theoretic protocols which are secure given worst-case corruption of roles. Unfortunately, these protocols incur exponential communication- and computation complexities, which motivates us to ask the following question:

*Can we design efficient distributed randomness generation protocols in the model of YOSO with worst-case corruptions?*

As it is trivially possible to adapt known stateful randomness generation protocols to the YOSO$^{\mathsf{WCC}}$ setting at the cost of having a very low adversarial threshold (see Section 4.2 for details), we further refine the question as follows:

*Can we design efficient distributed randomness generation protocols in the model of YOSO with worst-case corruptions while optimizing the required number of roles?*

Finally, given the increasing significance of (variants) of the YOSO model for secure distributed computing, it is imperative to understand the weakest cryptographic assumptions necessary to generate useful yet stateless randomness in the distributed setting. This motivates us to ask the following question:

*What are the minimal assumptions under which efficient (poly-time) computationally secure randomness generation is feasible in the YOSO$^{\mathsf{WCC}}$ model?*

### 4.1.1 Our Contributions

In the remainder of this chapter we answer the questions above. As in NRO, we distinguish between two different adversarial models, the *sending-leaks* and *execution-leaks* models. Intuitively, in the execution-leaks model the adversary only obtains messages addressed to corrupted parties upon their execution. In the stronger sending-leaks model, the adversary obtains the messages addressed to corrupted parties immediately upon the sender sending the message. We design two randomness generation protocols in the sending-leaks model, along with an optimized version for the execution-leaks model, and prove these protocols secure. Our protocols are in the computational setting, meaning that the adversary we consider is computationally bounded.

In our first construction, we build upon a non-interactive *publicly verifiable secret sharing* (PVSS) protocol [100], which allows a dealer to share a secret in a single round among a set of parties (a subset of which can be corrupt) in a way that lets anyone verify that the dealer behaved correctly. Our PVSS-based randomness generation protocol requires $3t + 2$ roles, and has communication complexity that grows quadratically in the number of parties.

While the above construction requires setup and somewhat heavy cryptographic machinery in the form of simulation-extractable non-interactive zero-knowledge proofs, in our second result we build a protocol which requires $5t + 3$ roles in the execution-leaks model and is based only on one-way functions. With the same assumption, the protocol can be adapted to the stronger sending-leaks model while requiring $n = 6t + 3$ parties.

Next, we show that we can reduce the number of parties to $n = 4t + 2$ in the execution-leaks model assuming the slightly stronger cryptographic primitive of non-interactive perfectly binding commitments. Non-interactive commitments can be instantiated from a variety of concrete assumptions including factoring [25, 58, 105], more recently from LWE and LPN [62], and even from the general assumption of injective one-way functions. While black-box separations between general one-way functions and non-interactive commitments are known [89], non-interactive commitments are fundamental and one of the weakest complexity-theoretic cryptographic assumptions. In the sending-leaks model, we can adapt the above protocol with $n = 5t + 2$ parties, while requiring the same computational assumptions.

In the setting without setup, we further reduce the number of parties to $3t + 1$ in the execution-leaks model (and $n = 4t$ in the sending-leaks model) at the expense of exponential computational and communication complexities in the corruption threshold $t$. Therefore, the resulting protocol is only efficient when the number of parties $n$ satisfies $n = O(\log \lambda)$, where $\lambda$ is the security parameter. This is relevant as it also makes sense to consider settings where the adversary's running time grows much faster than the number of parties. In particular, the protocol is efficient when the number of parties is constant.

We complement the above by studying lower bounds for computationally secure $\text{YOSO}^{\mathsf{WCC}}$ protocols without setup. Previous work [92] proved an impossibility result for information-theoretic $\text{YOSO}^{\mathsf{WCC}}$ randomness generation with $n = 4$ parties and $t = 1$ corruptions, which is tight and directly extends to $t > 1$ corruptions and $n = 4t$ parties in the sending-leaks model.[1] First, we observe that the same approach gives an impossibility

[1]The work [92] claims that the information-theoretic $(t = 1, n = 4)$ impossibility result also extends directly to $t > 1$ and $n = 4t$ in the weaker execution-leaks model, but it is not clear whether this holds. We discuss this in more detail in Section 4.9.

result for computationally secure YOSO$^{\mathsf{WCC}}$ randomness generation with $n = 3$ parties and $t = 1$ corruptions, which is tight and easily extends to $t > 1$ corruptions and $n = 3t$ parties in the sending-leaks model.

More interestingly, we investigate whether it is also possible to extend the $(t = 1, n = 3)$ impossibility in the computational setting to $t > 1$ and $n = 3t$ *in the execution-leaks model*. In this endeavor, we take the first step with a novel approach and show the impossibility of $(t = 2, n = 6)$-computationally secure YOSO$^{\mathsf{WCC}}$ randomness generation in the execution-leaks model. We leave extending this result to $t > 2$ as an interesting open problem. Coupled with our feasibility results, this result tells us that $n = 4$ parties are necessary and sufficient for computationally secure YOSO$^{\mathsf{WCC}}$ randomness generation against $t = 1$ corruptions. It also tells us that $n = 7$ parties are necessary and sufficient against $t = 2$ corruptions in the execution-leaks model, while $n = 7$ parties are necessary and $n = 8$ parties are sufficient in the sending-leaks model.

In the following, we first briefly outline our model, and then provide an overview of the main techniques and ideas used in our work.

### 4.1.2 Our Model and Security Goal

We now briefly outline the YOSO$^{\mathsf{WCC}}$ model we work in, following the communication model description of NRO [92]. We distinguish between stateless "roles" and physical machines which may run for a long time and retain state. Note that in the following we use the terms "role" and "party" interchangeably. We consider $n$ parties $P_1, \ldots, P_n$, which are executed one after the other. We assume that each party has its own internal source of randomness. We consider a computationally bounded adversary which is allowed to corrupt any $t$ out of $n$ parties before the protocol starts. Upon its execution, $P_i$ can publicly broadcast a value $x_i$ and send secret values $s_{i,j}$ to each "future" party $P_j$, i.e., any $P_j$ such that $j > i$. We consider the following two adversarial network settings:

- In the *sending-leaks* model an adversary obtains a message $s_{i,j}$ sent by an honest $P_i$ to a corrupt $P_j$ as soon as $P_i$ sent it. We call the corresponding adversary the *sending-leaks* adversary.
- In the *execution-leaks* model an adversary obtains a message $s_{i,j}$ sent by an honest $P_i$ to a corrupt $P_j$ only once $P_j$ is activated. We call the corresponding adversary the *execution-leaks* adversary.

Our goal is the following: After the execution of all parties is complete, anyone (not just

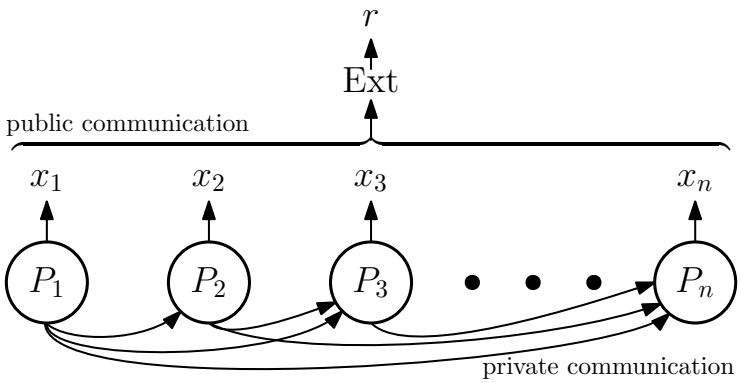



**Figure 4.1:** Communication model from [92, Figure 1]. Parties $P_i$ speak one after the other, send secrets to future parties $P_j$ for $j > i$, and publish public values, which are available to all parties.

physical machines which acted as roles $P_1, \ldots, P_n$) can obtain unbiased public randomness by applying a publicly known and deterministic extraction function to the values $(x_1, \ldots, x_n)$. See Figure 4.1 for a visual representation of this process.

More formally, let $\lambda$ denote a security parameter. Consider an interaction of an adversary $A$ with the honest parties in the randomness generation protocol and let $\mathsf{OUT}(A)$ denote the coin output of this protocol with adversary $A$. Let $L(\lambda)$ denote the length of this output. Let $D$ be a distinguisher. Consider the following experiment (for protocols which assume trusted setup, this setup is generated by the challenger):

1. $b \xleftarrow{\$} \{0,1\}$.
2. $r \xleftarrow{\$} \{0,1\}^{L(\lambda)}$.
3. If $b = 0$, set $\mathsf{coin} \leftarrow \mathsf{OUT}(A)$. Otherwise, set $\mathsf{coin} \leftarrow r$.
4. $b' \leftarrow D(\mathsf{coin})$.

Then, we have the following formal security definition.

**Definition 16** (Computationally secure YOSO$^{\mathsf{WCC}}$ randomness generation)**.** *A YOSO$^{\mathsf{WCC}}$ randomness generation protocol with $n$ parties is* $(t, n)$-computationally secure in the sending-leaks (resp. execution-leaks) model *if for all PPT sending-leaks (resp. execution-leaks) adversaries $A$ that corrupt $t$ out of $n$ parties and all PPT distinguishers $D$ in the above security game it holds that*

$$\left|\Pr[b = b'] - \frac{1}{2}\right| \leq \mathsf{negl}.$$

## 4.2 Our Techniques

First, note that, as pointed out by NRO, any stateful $r$-round multiparty computation protocol which is secure against $t$ out of $n$ corruptions can be ported to the YOSO$^{\mathsf{WCC}}$ setting as follows: Use $r$ roles $P_{i,r}$ to implement the behavior of each participant $P_i$ of the stateful protocol over $r$ rounds. Role $P_{i,k}$ mimics the behavior of $P_i$ in round $k$ of the stateful protocol, with the caveat that it additionally sends its state to the future role $P_{i,k+1}$. Unfortunately, this approach is costly in terms of the required number of roles: It requires $n \cdot r$ roles, while tolerating only $t$ corrupted parties.

To address this issue, we design randomness generation protocols which are tailored to the YOSO$^{\mathsf{WCC}}$ setting. For simplicity, say we wish to generate only a single random bit $r \in \{0, 1\}$.

### 4.2.1 First Attempts at YOSO$^{\mathsf{WCC}}$ Protocols

Our first idea is the following: As each party has its own source of randomness, we could set $n = t + 1$ and simply XOR all values $r_i$, where $r_i$ is the random bit generated by $P_i$, i.e., set $r = \bigoplus_{i \in [t+1]} r_i$. As at least one party out of $t + 1$ is honest, the XOR should result in an unbiased bit. However, we need to be careful – we must not let a corrupt party see the values of the honest parties before supplying its own $r_i$. Thus, intuitively, we have to make each party *commit* to the randomness it wishes to contribute prior to revealing the values of other parties. This approach requires a party to speak two times: Once when committing to a value, and once when opening it. This can be naively achieved by using two roles to implement $P_i$, and having the first role privately send its state to its counterpart.

Perhaps surprisingly, this approach still does not achieve what we want: As corrupt parties can refuse to open the committed values, in our protocol we must specify how to proceed in such a case. This allows for *conditional abort* attacks. Consider the following. When dealing with verifiably malicious behavior, we can either choose to ignore each such party $P_i$, thereby making their contribution equal to $r_i = 0$ (first case in the following), or set $r_i = 1$ (second case). In both cases, a corrupt $P_{t+1}$ can bias the outcome of the final XOR by committing to $r_{t+1} = 1$ in the first case and $r_{t+1} = 0$ in the second case, and then adaptively deciding whether to open the value or not during the execution of its second role, thereby setting the result $r$ to the value of its choice. As the second role of $P_{t+1}$ is the last party speaking, all values supplied by the honest parties are known upon its execution.

### 4.2.2 Utilizing PVSS

We address the issue above by ensuring that the coin output is fixed *prior* to the reveal phase. We begin by considering a setting with trusted setup. In this case, we can rely on a $(t, n)$-*publicly verifiable secret sharing* (PVSS) protocol. Using such a protocol, a dealer can secret share its secret among $n$ parties in a way that any $t + 1$ parties can reconstruct the secret, but any $t$ (potentially corrupted) parties have no information about the secret. Moreover, public verifiability ensures that anyone (even non-recipients) can verify that the dealer sharing has been performed correctly; i.e., there exists a *unique* secret which can be later reconstructed by any set of $t + 1$ recipient parties.

Intuitively, this fixes the secret at the end of the commit/sharing phase, and if the adversary corrupts at most $t$ parties, it does not learn any information about the secret. If we ensure that the secret reconstruction starts only after the sharing phase of *all* secrets is complete, the adversary can no longer bias the outcome. However, there is one caveat: As anyone must be able to verify that the sharing was done correctly, the dealer cannot send the shares to the parties via *private* communication. Instead, the dealer publishes encryptions of the shares of the parties with respect to their corresponding public keys. In a scenario such as ours, where we run not only one, but *multiple* PVSS protocols, publicly revealing encryptions of the shares makes PVSS susceptible to malleability attacks. To prevent such attacks from adversarial dealers, we make use of a PVSS protocol with appropriate non-malleability properties. Such properties can be achieved, for example, via simulation-extractable non-interactive zero-knowledge proofs [68].

If we use a $(t, 2t + 1)$-PVSS protocol, the above protocol requires only $3t + 2$ roles in total: $t + 1$ dealers and $2t + 1$ parties who obtain the secret shares. This protocol allows us to achieve the following result:

**Theorem 7** (informal)**.** *Assuming public key encryption and simulation-extr-actable NIZKs, there exists a computationally secure randomness generation protocol with* $3t + 2$ *roles in the sending-leaks model, where* $t$ *is the number of corruptions.*

We give a formal description of our PVSS-based construction in Section 4.4.

While the protocol above enjoys good efficiency properties and requires only a small number of parties, it relies on somewhat heavy cryptographic assumptions and a trusted setup. We now address these limitations.

### 4.2.3 An Efficient YOSO$^{\mathsf{WCC}}$ Protocol based on Digital Signatures

Recall that to circumvent the "conditional abort" issue, we need to ensure that the coin output is fixed *prior* to the reveal phase. Toward this end, we turn to Verifiable Secret Sharing (VSS). Roughly speaking, a $(t, n)$-VSS protocol allows threshold secret sharing of a secret to $n$ parties such that (1) a secret shared by an honest dealer is always reconstructed correctly by any set of $t + 1$ parties, (2) prior to reconstruction phase, no information is leaked about the secret to any set of $t$ parties, and (3) receivers can verify that the dealer behaved correctly, i.e., there exists a unique secret corresponding to the sharing phase, and it can be correctly reconstructed.

Intuitively, this fixes the secret at the end of the commit (*sharing*) phase, and the adversary corrupting at most $t$ parties learns no information about the secret. If we ensure that the coin reconstruction starts only after the sharing phase of *all* secrets is complete, the adversary can no longer bias the outcome.

Our first goal is to design a YOSO$^{\mathsf{WCC}}$ VSS protocol with as few roles and as minimal assumptions as possible. We will then see how to use this VSS protocol to build a full-fledged YOSO$^{\mathsf{WCC}}$ randomness generation protocol. In fact, we show that our randomness generation protocol can be based on a weaker version of VSS, which we call *split-dealer VSS*.

**Unoptimized Stateful VSS.** Our starting point is the elegant *stateful* VSS protocol by Hirt and Zikas [76] that is based on the BGW VSS protocol [19] and the work of Cramer, Damgård, Dziembowski, Hirt, and Rabin [44]. The protocol requires $n = 2t + 1$ parties who hold secret shares and rely on a standard signature scheme, private communication channels, and access to a broadcast channel. This protocol consists of two phases each with several rounds as described below.

Sharing Phase:

(1) *Share round:* Dealer $D$ with secret $s$ selects a uniform bi-variate polynomial $f(x, y)$ of degree at most $t$ in each variable, such that $f(0, 0) = s$. Let $s_{i,j} = f(i, j)$. $D$ privately sends shares $\{s_{k,i}\}_{k\in[n]}$ and $\{s_{i,k}\}_{k\in[n]}$, along with signatures on these values to each party $P_i$. $P_i$ denotes these values as $\{s^{(i)}_{k,i}\}_{k\in[n]}$ and $\{s^{(i)}_{i,k}\}_{k\in[n]}$.

(2) *Share check round:* Each party $P_i$ checks whether the values they received are *$t$-consistent*, i.e., fit onto a polynomial of degree at most $t$, and contain valid signatures from $D$. If not, $P_i$ broadcasts a complaint.

(3) *Dealer response round:* Dealer $D$ addresses the complaint of each party $P_i$ by broadcasting the correct values for $P_i$, along it's signatures. If these values are not $t$-consistent or the signatures are invalid, $D$ is deemed corrupt and the execution halts. Otherwise, $P_i$ adopts the new values as the messages it received in the sharing round.

(4) *Subshare exchange round:* Party $P_i$ sends the value $s_{i,j}^{(i)}$ and both *its own, and the dealer's* signature on it privately to $P_j$.

(5) *First subshare check round:* Party $P_i$ checks if they received a message along with valid signatures from every other party. If a message from $P_j$ is missing or does not contain valid signatures, $P_i$ broadcasts a complaint.

(6) *Resolve complaints round:* Party $P_i$ checks if there is a complaint by any $P_j$ about $P_i$. If yes, $P_i$ broadcasts $s_{i,j}^{(i)}$ along with $D$'s and its own signature. If $P_i$ is silent, or any of the signatures are invalid, $P_i$ is deemed corrupt, and everyone sets signatures of $P_i$ to $\perp$. Otherwise, $P_j$ adopts the message broadcast by $P_i$ as the message it received during the subshare exchange.

(7) *Second subshare check round:* Party $P_i$ checks if it received any value $s_{j,i}^{(j)}$ during the subshare exchange or resolve complaints round which is inconsistent with its view. If yes, $P_i$ broadcasts $s_{j,i}^{(j)}$, $s_{j,i}^{(i)}$, along with $D$'s signature on both values. If the two values are different, and have valid signatures, $D$ is deemed corrupt and the execution halts.

Reconstruction Phase: In this phase each party $P_i$ broadcasts $\{s_{k,i}^{(i)}\}_{k\in[n]}$, along with the signature for each $s_{j,i}^{(i)}$ that $P_i$ received from $P_j$. Each party checks if the values broadcast by every $P_i$ are $t$-consistent, and all signatures are valid. If not, $P_i$ is disqualified. The values of all non-disqualified parties are interpolated to compute $f(0,0)$.

The protocol's correctness relies on honest parties only sending or broadcasting consistent, correctly signed values, with their shares being sufficient to compute the secret. Privacy is maintained as any $t$ shares reveal no information about the secret, and the adversary learns nothing additional from honest parties during the sharing phase. For verifiability, if the dealer is not disqualified, a sufficient number of honest parties ($n - t = t + 1$) possess consistent shares that define a unique secret. Even a malicious dealer cannot prevent the secret's reconstruction, as no honest party would sign inconsistent shares.

**Reducing Round Complexity.** The scheme above is not well-suited for a direct transformation into the YOSO$^{\mathsf{WCC}}$ setting. In fact, a naive transformation of the above protocol to the YOSO$^{\mathsf{WCC}}$ model would require $n = 6(2t + 1) + 2 = 12t + 8$ parties. The round complexity of the above VSS is the first clear bottleneck in our approach. Therefore, we

aim to reduce the number of rounds, and thus reduce the number of parties needed in the $\text{YOSO}^{\mathsf{WCC}}$ model.

**Merging Subshare Checks.** Observe that if round 5 and round 7 subshare checks could be merged, followed by a one-shot "resolve complaints" (round 6), this would reduce the round complexity by one round, resulting in a reduction of $2t + 1$ parties in the $\text{YOSO}^{\mathsf{WCC}}$ setting.

The intuition behind having the "resolve complaint" phase between the two subshare checks is the following: If party $P_i$ complains in (5), party $P_j$ resolves the complaint *publicly* in (6) while including its own and the dealer's signatures. Everyone can verify whether $P_j$'s response is valid. Even if so, $P_i$ might still be unhappy, as the dealer could have given inconsistent shares to $P_i$ and $P_j$. In this case, $P_i$ can complain again, this time including its own and the dealer's signature on the corresponding value. Since all the complaints and the resolving messages are public, anyone can conclude if the dealer is to be blamed or not. We make the following crucial observation: $P_i$ *never changes* its own share based on the resolving message from $P_j$ in round (6). Additionally, $P_i$ can verify whether the value it received from $P_j$ in the subshare exchange (4) is consistent with its own shares received from the dealer *directly after (4)*.

These observations allow us to change the protocol as follows, while retaining its security. If the share that $P_i$ was supposed to receive from $P_j$ during the subshare exchange (4) is missing, contains invalid signatures, *or is inconsistent with its own share*, $P_i$ complains *and includes its own and the dealer's signatures*. Then, $P_j$ is forced to publicly respond. As before, if $P_j$'s response is missing or contains invalid signatures, $P_j$ is discarded. If $P_j$'s response is inconsistent with $P_i$'s, but the signatures are valid, everyone can conclude that the dealer misbehaved. This has the same effect as before – either the parties agree on their shares, or either a malicious party $P_i$ or the malicious dealer is disqualified.

**Merging Share Check and Subshare Exchange.** Next, we seek to merge the share check (2) and the subshare exchange (4) rounds, which would result in another reduction of $2t + 1$ parties.

Currently, the dealer ensures in (3) that either all parties are *happy* with their shares, or the dealer can be deemed corrupt for *publicly* providing inconsistent shares. Thus after (3), all parties *must* have complete sharings signed by the dealer, and if there are complaints in later rounds, we can definitively assign blame to either the dealer or a party. Observe that if a party complains in (2), *everyone* knows that the party is unhappy, and other happy parties can still crosscheck their values. However, we must ensure that if the dealer resolves

complaints *after* the subshare checks, the new shares are *still* consistent with the shares of the happy ones. Subtle modifications allow merging (2) and (4) and delaying the dealer response until the sharing phase's end.

Happy parties proceed with the subshare exchange, while unhappy party $P_i$ skips this phase and broadcasts a complaint. $P_i$ also skips the subshare checks and the resolve complaints round. If happy party $P_j$ complains about a missing message from $P_i$, $P_j$ includes $s_{j,i}^{(j)}$ and $s_{i,j}^{(j)}$, along with its own and the dealer's signature. After the resolve complaint phase, the dealer addresses $P_i$'s complaint. Everyone verifies that the values posted by the dealer are consistent, in particular with the *non-complaining parties*, like $P_j$. As $P_j$ broadcast $s_{j,i}^{(j)}, s_{i,j}^{(j)}$ along with valid signatures from the dealer, anyone can see if the new share distributed by the dealer is consistent, and the dealer can be discarded if this is not the case.

**Removing "Resolve Complaints".** Currently, the "resolve complaints" round is followed by the dealer response. We might hope that the dealer can resolve the complaints *on behalf* of every party $P_j$, eliminating the need for the resolve complaints round. However, this modification requires some care: A malicious dealer and a malicious party $P_i$ can provide a share inconsistent with an honest happy party $P_j$. Imagine a malicious $P_i$ complaining about an honest party $P_j$, while including a valid signature on $s_{j,i}^{(j)}, s_{i,j}^{(j)}$ from a malicious dealer. Previously, $P_j$ would have responded publicly by providing the dealer's signature on its own share, thus unmasking the malicious dealer. However, now the dealer can simply confirm $P_i$'s share, and thus the share held by $P_j$ becomes inconsistent.

We rectify this issue by using $3t + 1$ share receivers (instead of $2t + 1$), and skipping the "resolve complaints" round, *without having the dealer resolve parties' complaints about each other*. Every point on which $P_i$ did not get a valid signature from $P_j$ is now essentially lost for reconstruction. $P_i$ still checks whether the points from $P_j$ contain valid signatures of the dealer on an inconsistent share, and blames the dealer if so. $P_i$ also broadcasts a complaint for a missing message from $P_j$, along with the dealer's signature on its corresponding share to ensure that any share that the dealer will broadcast for an unhappy $P_j$ remains consistent with $P_i$'s share. In the reconstruction phase, we consider any polynomial of $P_i$ to be valid if it has $2t + 1$ points $s_{j,i}^{(i)}$, such that (1) each point is either correctly signed by $P_j$, or broadcast by the dealer if $P_j$ complained during the share check, and (2) all these points lie on a polynomial of degree at most $t$. Intuitively, requiring $2t + 1$ valid points ensures that any party's share is consistent with at least $2t + 1 - t = t + 1$ honest parties. Since any $t + 1$ honest shares fix the secret, any valid share is thus consistent with the secret. Simultaneously, given $3t + 1$ share receivers, any party can provide at least $2t + 1$ such

points, as at most $t$ of the points can be unavailable due to malicious parties. While this adds $t$ receivers, we can cut the resolve complaints round and save $t+1$ parties with further optimisations discussed below.

**Obtaining YOSO$^{\mathsf{WCC}}$ VSS.** In the scheme we arrived at, the sharing phase consists of the following rounds: *sharing, share check with subshare exchange, subshare checks*, and *dealer response*. Consider a "linearized" version of this scheme in the YOSO$^{\mathsf{WCC}}$ setting, where parties are executed one after the other: It requires one party $D_1$ to be the dealer in share round, and another party $D_2$ to execute the role of the dealer in dealer response (this will be the resolver). It further requires a set of $3t+1$ parties $\mathcal{P}^1$ to execute share check with subshare exchange, another set $\mathcal{P}^2$ of size $3t+1$ to execute the subshare check, and a set $\mathcal{P}'$ of $3t+1$ parties to perform the reconstruction. The scheme works as follows: First, the dealer $D_1$ distributes shares of secrets to parties in $\mathcal{P}^1$, and additionally sends its entire state to the future dealer $D_2$. Then, one after the other, each party $P_i^1 \in \mathcal{P}^1$ performs the share check, and either sends its subshares to the parties in $\mathcal{P}^2$, or complains about the dealer. Additionally, $P_i^1$ sends its state to its future counterpart $P_i^2 \in \mathcal{P}_2$. Each $P_i^2 \in \mathcal{P}^2$ performs the subshare checks, complains if necessary, and sends its state to the future counterpart $P_i'$. Finally, parties in $P_i'$ output the data from which $s$ can be computed according to the reconstruction phase.

We observe the following: during the sharing check with subshare exchange, when party $P_j^1$ is executed, it could have already obtained the shares of all parties $P_i^1$, for $i < j$, and perform the subshare checks, as those parties have *already* executed the subshare exchange. If we were to require the checks to be performed only by such *ordered* pairs of parties, in the YOSO$^{\mathsf{WCC}}$ model we could remove the $3t+1$ parties $\mathcal{P}^2$ which are currently used to execute the subshare check round of the stateful construction. Note that in the stateful version, we already do not require parties $P_i$ to resolve the complaints about them, as we were able to *remove the resolve complaints round* by requiring $3t+1$ share receivers and $2t+1$ valid points in the reconstruction phase instead. Thus, we simply must ensure that if honest parties notice an inconsistency in the prior version of the protocol, this inconsistency still becomes public, even if the check is done only for pairs $(P_i, P_j)$, where $i < j$. Note that given two honest parties $P_i$, $P_j$, it is sufficient if only one of them checks their shares for consistency and complains on behalf of the pair if necessary. For this, we let the complaint include signatures on the respective values from *both* $P_i$ and $P_j$, as well as the dealer's signatures on these values. This way, in the YOSO$^{\mathsf{WCC}}$ version of the construction, we can slash additional $3t+1$ parties $\mathcal{P}^2$ by merging the sharing check with subshare exchange, and subshare checks phases. In the following, we have only the set $\mathcal{P}$, instead of two separate

sets $\mathcal{P}^1$ and $\mathcal{P}^2$.

**Optimizing Reconstruction in the Sending-Leaks Model.** In the sending-leaks model we optimize our construction as follows. Having a one-to-one correspondence between the $3t+1$ parties $P$ and parties $P'$ seems wasteful, as intuitively $2t+1$ reconstructors ought to be sufficient. Consider the following optimization: Instead of having each $P_i \in \mathcal{P}$ send the state only to its counterpart $P'_i \in \mathcal{P}'$, party $P_i$ *secret shares* it to parties $\mathcal{P}'$, $|\mathcal{P}'| = 2t+1$, using a standard $(t, 2t+1)$-Shamir secret sharing, while including its own signature on the share. The privacy of the honest shares is still preserved for the duration of the sharing phase by the privacy of the secret sharing scheme. The $2t+1$ reconstructors now broadcast the signed shares of each $P_i$'s state. As only the shares correctly signed by $P_i$ will be used for reconstruction of $P_i$'s state, no adversary can modify the reconstructed state of an honest $P_i$. Further, $t+1$ correctly signed shares are available for any honest $P_i$ as there are at least $t+1$ honest reconstructors. Now, we only require $2t+1$ parties in the reconstruction phase. This finalizes our scheme in the sending-leaks model. A full formal description is given in Protocol 17.

**Optimizing Reconstruction in the Execution-Leaks Model.** In the execu-tion-leaks model we can further reduce the number of reconstructors compared to the sending-leaks model. Consider the following modification: Instead of having each $P_i \in \mathcal{P}$ send the state only to its counterpart $P'_i \in \mathcal{P}'$, $P_i$ sends it (signed) to *every* party in $\mathcal{P}'$. As in the execution-leaks model the adversary obtains the values only when an adversarial party is being executed, the privacy of the honest shares is preserved for the duration of the sharing phase. Now, we only require $t+1$ parties in the reconstruction phase to ensure there is at least 1 honest reconstructor in $\mathcal{P}'$. These parties gather the information sent to them by the parties $P_i$, and reveal all shares that are verified, along with all available signatures. This finalizes our scheme in the execution-leaks model. We give the full construction in Protocol 16, and provide a formal security proof in Section 4.6.1.

**Putting It Together: Efficient YOSO$^{\mathsf{WCC}}$ Randomness Generation.** We now compile several instances of our YOSO$^{\mathsf{WCC}}$ SD-VSS protocol introduced above into the randomness generation protocol we aimed for. Concretely, we take $t+1$ instances of our YOSO$^{\mathsf{WCC}}$ SD-VSS so that we have $t+1$ dealers and the final coin is computed through some deterministic function of the non-misbehaving dealers' secrets. This compilation step has to be done carefully to minimize the role-overhead for our randomness generation protocol without compromising security.

As a first step, consider the following construction: $t+1$ dealers in $\mathcal{D}$ share their secrets

via YOSO$^{\mathsf{WCC}}$ SD-VSS to the *same* set of share receivers $\mathcal{P}$. Each $P_i \in \mathcal{P}$ verifies its shares, distributes subshares, submits complaints and sends its state to the future as specified by YOSO$^{\mathsf{WCC}}$ SD-VSS above. Then, for each dealer $P_i \in \mathcal{D}$ its counterpart $P'_i \in \mathcal{D}'$, dubbed *resolver*, addresses the complaints of the receivers. The reconstructors publish data according to the YOSO$^{\mathsf{WCC}}$ SD-VSS. Given this information, anyone can compute the value shared by each dealer, and output for instance the XOR of each valid value that was reconstructed. This requires $6t + 4$ parties in total in the execution-leaks model, and $7t + 4$ parties in the sending-leaks model.

**Role-Stacking.** We introduce *role-stacking* to reduce the number of parties needed. Notice that when dealer $P_i \in \mathcal{D}$ is active, each dealer $P_j$ for $j < i$ has already distributed their share. Thus, $P_i$ can act as a dealer for its secret, a *share receiver* of all secrets from prior dealers $P_j$ for $j < i$, and a receiver for the sharing of its own secret. This allows us to "stack" multiple instances of SD-VSS, so each party performs multiple roles during execution. Role-stacking merges (1) dealers and share recipients, and (2) share recipients and resolvers.

This approach maintains security, as at most $t$ dealers can be corrupted, ensuring at least one honest value is used in the output. For each SD-VSS, at most $t$ recipients are corrupted, preserving the security properties of the stacked construction. However, care is required when using role-stacking in general: For example, security immediately breaks down if we stack roles from the sharing phase with roles from the reconstruction phase. Using role-stacking in combination with YOSO$^{\mathsf{WCC}}$ SD-VSS, we obtain a YOSO$^{\mathsf{WCC}}$ randomness generation with role complexity of $n = 5t+3$ in the execution-leaks model, and $n = 6t+3$ in the sending-leaks model with the minimal assumption of the existence of digital signatures (or one-way functions).

### 4.2.4 An Efficient YOSO$^{\mathsf{WCC}}$ Protocol based on Non-Interactive Commitments

In our next step, we improve the number of required roles by providing efficient YOSO$^{\mathsf{WCC}}$ protocols under the assumption of non-interactive commitments. Our construction can be seen as a variant of the protocol from [88] that uses ElGamal commitments, but generalized to allow for any non-interactive commitment scheme (even if it is not homomorphic). The protocol from [88] is a custom version of a sequence $t + 1$ instantiations of Pedersen's VSS protocol [95], where each dealer shares a random value. More precisely, each instantiation has the following roles:

1. Party $D$, who acts as the dealer distributing the secrets (publishing commitments to the coefficients of $t$-degree polynomial and bilaterally sending to each receiver a share evaluation), and sends its state to its counterpart $D'$.
2. $2t+1$ receivers $R_i$, who receive and verify the secret shares, complain about the shares if applicable, and otherwise send these to each party $R'_i$.
3. Party $D'$ who obtains a state from $D$ and uses it to publish the shares of the receivers that complained. If $D'$ cannot resolve a complaint, this instance is aborted.
4. $t+1$ receivers $R'_i$ who receive all the shares from each party $R_i$, as well as set their shares to the ones broadcast by $D'$ (if the counterpart $R_i$ complained), and publicly reveal all these shares.

The idea is that after the first three steps, the dealer has committed to a random value, which will be reconstructed in Step 4. Note that before Step 4, if both $D$ and $D'$ are honest, no information about the committed random value is revealed. Therefore, to generate a random coin, one can use a standard linearization of $t+1$ instances of the above protocol (where the first three steps of each instance are executed, and subsequently all committed random values are reconstructed). The final coin is then the sum of the $t+1$ random values. This works because since there are $t+1$ dealers, at least one of them is honest; moreover, before the reconstruction phase starts (Step 4 of each instance), the adversary submits secrets without knowing the honest secrets, and every instance that succeeded, is guaranteed to be reconstructed.

The above construction requires the commitment to be homomorphic, since the receivers need to compute commitments to the point evaluations from the commitments to the polynomial coefficients. We observe that one can instead let the dealer $D$ publish $2t+1$ commitments to the points themselves, rather than the $t+1$ coefficients, and send its state to $D'$. However, the difference is that now a cheating dealer could commit to a polynomial that is not of degree $t$.

To solve this, one can instead let the dealer commit to all the points of a bivariate degree-$t$ polynomial $F(x,y)$, and send to each receiver $R_i$ the openings corresponding to the $i$-th projection (horizontal and vertical), i.e. openings to the commitments of the points $\{F(i,j)\}_{j\in[2t+1]}$ and $\{F(j,i)\}_{j\in[2t+1]}$. Each receiver can now directly check the openings against the published commitments, and also check that the two projections are of degree $t$. The party $D'$ will publish the openings to the points of any $R_i$ that complained.

After resolving the complaints, observe that the projections corresponding to any two honest receivers $R_i$ and $R_j$ are consistent among each other and have degree $t$, and therefore the committed bivariate polynomial $F$ has degree $t$. Moreover, the state of each $R_i$ is sent

to all receivers $R'_i$, and therefore any honest receiver $R'_i$ can provide enough information to reveal the whole polynomial.

At a high level, the protocol described above uses a total of $n = 5t + 4$ parties: a group $\mathcal{D}$ of size $t + 1$, group $\mathcal{R}$ of size $2t + 1$, group $\mathcal{D}'$ of size $t + 1$, and group $\mathcal{R}'$ of size $t + 1$. The parties execute the following roles:

- Each $D_i \in \mathcal{D}$ acts as the dealer $D$ in the $i$-th linearization.
- Each $R_i \in \mathcal{R}$ executes the role of the $i$-th receiver $R_i$ in *each* of the $t + 1$ linearizations.
- Each $D'_i \in \mathcal{D}'$ acts as the dealer $D'$ in the $i$-th linearization.
- Each $R'_i \in \mathcal{R}'$ executes the role of the $i$-th receiver $R'_i$ in *each* of the $t + 1$ linearizations.

However, one can slightly improve the number of roles to $4t + 4$ with the following modification. Instead of letting $t + 1$ dealers, each of whom shares secrets among the *same set* $\mathcal{R}$ of $2t + 1$ parties, we let each dealer share secrets among the *next* $2t + 1$ parties. More details can be found in Section 4.7.

### 4.2.5 A YOSO$^{\mathsf{WCC}}$ Protocol for $n = 3t + 1$ Parties

We now discuss how we can further reduce the number of parties to $n = 3t+1$ in the execution-leaks model, at the expense of exponential computational and communication complexities in the corruption threshold $t$. The protocol follows the familiar high-level structure where we split the set of parties into *dealers* and *receivers*. In previous instantiations of this structure, we had each dealer perform one sharing. Instead, here we have each dealer perform a number of sharings potentially exponential in $t$.

Let $\ell$ be the number of dealers (to be set later). Each sharing is represented by a nonzero vector $v \in \{0, 1\}^\ell$. The dealer corresponds to $\min \mathsf{Supp}(v)$, i.e., the position of the leftmost 1 in $v$. Our building block is a non-interactive commitment scheme $\mathsf{COM}$ (that is perfectly binding and computationally hiding). We focus on generating a random bit here. A sharing proceeds as follows:

1. The dealer samples a random bit $r_v \leftarrow\!\!\$\ \{0, 1\}$ and broadcasts a commitment $\mathsf{com}$ to $r_v$. It also sends $r_v$ to all parties $P_i$ such that $i \in \mathsf{Supp}(v)$ and to all receivers via private messages.
2. Each party $P_i$ such that $i \in \mathsf{Supp}(v)$ forwards the openings it receives from earlier parties in $\mathsf{Supp}(v)$ to all receivers. If it receives an inconsistent opening from the earlier party, then it broadcasts $(\texttt{Complain}, v)$.
3. Receivers broadcast everything that they receive through private messages.

Overall, the $m$ total sharings are represented by rows of a binary "$t$-sharing matrix" $M \in \{0,1\}^{m \times \ell}$. Given this matrix, the full protocol works as follows:

1. The $\ell$ dealers $P_1, \ldots, P_\ell$ perform sharings according to the $m$ rows of $M$.
2. Party $P_{\ell+1}$ outputs a random bit $r^\star \leftarrow\!\!\$\ \{0,1\}$.
3. The $t+1$ receivers $P_{\ell+2}, \ldots, P_{\ell+t+1}$ broadcast everything they receive from the dealers $P_1, \ldots, P_\ell$.

To reconstruct the coin from the public broadcasts, we use the information published by the receivers to open the commitments published by the dealers, and then XOR the random bits $r_v$ for all rows $v$ of $M$ with the special random bit $r^\star$. If a complaint was broadcast during the sharing of some bit $r_v$, then we ignore that sharing (i.e., replace it by 0 in the XOR).

We identify two properties of the $t$-sharing matrix $M$ that are sufficient to yield a secure protocol against $t$ corruptions:

1. Every row of $M$ has Hamming weight $t$ (i.e., $|\mathsf{Supp}(v)| = t$ for all rows $v$);
2. For any $t-1$ columns $M_{\cdot j_1}, M_{\cdot j_2}, \ldots, M_{\cdot j_{t-1}}$ of $M$, there exists an index $i$ such that $M_{ij_1} = M_{ij_2} = \cdots = M_{ij_{t-1}} = 0$. In other words, the coordinate-wise union of any $t-1$ columns is not the all-1s vector.

The second property is reminiscent of the notion of *$t$-disjunct matrices*, which are useful in the design of non-adaptive group testing schemes (see [72, Chapter 22] for a discussion of this topic).

**Intuition behind the security proof.** We give some intuition on why the properties above are sufficient to establish security. First, note that if the adversary corrupts $t$ dealers, then they cannot bias the coin because they do not know the random bit published by $P_{\ell+1}$ (who is honest), and, furthermore, all the receivers are also honest. Therefore, we may assume that the adversary corrupts at most $t-1$ dealers, and possibly some other non-dealer parties.

By Property 1 above, the assumption that only at most $t-1$ dealers are dishonest means that every sharing contains at least one honest party. This forces the dishonest parties in that particular sharing to commit to *some* (possibly non-random) value during the sharing procedure, as otherwise the honest party would identify an inconsistency and complain. If $P_{\ell+1}$ is honest, then the output of the protocol will be unbiased, because this party is only executed after the sharing phase has concluded and so its published bit $r^\star$ is uniformly random and independent of the bits generated in the sharing phase. On the

other hand, if $P_{\ell+1}$ is dishonest then we invoke Property 2, which ensures that there is a sharing in which all participating parties are honest. Intuitively, the random bit produced in this fully honest sharing is hidden (by the security of the commitment scheme) from the adversary until the receivers are executed, at which point other shared values have already been committed to and party $P_{\ell+1}$ has already been executed.

**Constructing the $t$-sharing matrix.** It remains to give a construction of a $t$-sharing matrix $M \in \{0,1\}^{m \times \ell}$ satisfying the properties laid out above. A simple option that works, and which we use, is to consider $\ell = 2t-1$ columns and $m = \binom{2t-1}{t}$ rows, one per weight-$t$ vector of length $2t-1$. Instantiating the framework above with this matrix yields a protocol with

$$n = \ell + 1 + (t+1) = (2t-1) + 1 + (t+1) = 3t+1$$

parties. It is natural to wonder whether one can obtain sharing matrices with fewer rows (meaning reduced computational complexity), or with a better tradeoff between number of rows (the number of sharings) and number of columns (the number of dealers) in a $t$-sharing matrix.

We use a simple argument to show that any $t$-sharing matrix $M \in \{0,1\}^{m \times \ell}$ must satisfy $m \geq \binom{\ell}{t-1} / \binom{\ell-t}{t-1}$. In particular, this means that when $\ell = 2t-1$, which is the minimum number of columns in a $t$-sharing matrix, then it is not possible to do better than our simple construction above. We can hope to obtain some complexity *versus* number of parties tradeoff by increasing the number of columns and decreasing the number of rows, but this lower bound also shows that the number of sharings remains exponential in $t$ unless we significantly increase the number of dealers.

Finally, we note that we can easily extend our execution-leaks protocol to work in the sending-leaks model by adding $t-1$ additional parties. More details can be found in Section 4.8.

## 4.3 Preliminaries

### 4.3.1 Notation

The sampling of a value $x$ according to a distribution $X$ is denoted by $x \leftarrow\!\!\$\, X$. If $S$ is a set, we also write $x \leftarrow\!\!\$\, S$ when $x$ is sampled uniformly at random from $S$. The support of the distribution $X$ is denoted by $\mathsf{Supp}(X)$.

We denote by $\lambda \in \mathbb{N}$ the security parameter and by $x \leftarrow \mathcal{A}(\mathsf{in}; r)$ the output of the algorithm $\mathcal{A}$ on input $\mathsf{in}$ using $r \leftarrow\!\!\$\ \{0,1\}^*$ as its randomness. We often omit this randomness and only mention it explicitly when required. We consider *probabilistic polynomial time* (PPT) machines as efficient algorithms.

For integers $m, n \in \mathbb{N}$ we write $[n] = \{1, 2, \ldots, n\}$ and $[m, n] = \{m, m+1, \ldots, n\}$. A function $\mathsf{negl}[] : \mathbb{N} \to \mathbb{R}$ is negligible if it is asymptotically smaller than any inverse-polynomial function, namely, for every constant $c > 0$ there exists an integer $N_c$ such that $\mathsf{negl} \leq \lambda^{-c}$ for all $\lambda > N_c$.

### 4.3.2 Digital Signatures

A *digital signature scheme* $\mathsf{DS}$ is a tuple of algorithms:

- A *key generation algorithm* $\mathsf{KGen}(1^\lambda)$ that takes the security parameter $\lambda$ and outputs the verification/signing key pair $(\mathsf{vk}, \mathsf{sk})$.
- A *signing algorithm* $\mathsf{Sign}(\mathsf{sk}, m)$ which outputs a signature $\sigma$ on input the secret key $\mathsf{sk}$ and the message $m$.
- A *verification algorithm* $\mathsf{Vf}(\mathsf{vk}, m, \sigma)$ with binary output. We say that $\sigma$ is a *valid* signature of $m$ under the verification key $\mathsf{vk}$ if $\mathsf{Vf}(\mathsf{vk}, m, \sigma) = 1$, and *invalid* otherwise.

We require standard unforgeability of the digital signature scheme.

**Definition 17** (Unforgeability)**.** *A digital signature scheme* $\mathsf{DS}$ *is* unforgeable *if for any PPT adversary* $\mathcal{A}$ *there exists a negligible function* $\mathsf{negl}$ *such that the probability of winning the following game is upper bounded by* $\mathsf{negl}$*, where* $\lambda$ *is a security parameter:*

1. *The challenger runs* $\mathsf{KGen}(1^\lambda)$ *and obtains a verification/secret key pair* $(\mathsf{vk}, \mathsf{sk})$.
2. *The adversary* $\mathcal{A}$ *can adaptively choose messages* $m$ *and query the challenger to learn a corresponding signature* $\mathsf{Sign}(\mathsf{sk}, m)$*. Let* $m_1, \ldots, m_q$ *be the messages queried by* $\mathcal{A}$*, for some integer* $q$*.*
3. *The adversary* $\mathcal{A}$ *chooses a fresh message* $m' \notin \{m_1, \ldots, m_q\}$ *and wins the game if they output a valid signature* $\sigma$ *of* $m'$ *under the verification key* $\mathsf{vk}$*.*

Signature schemes have been constructed from a wide range of assumptions starting from one-way functions [85], to more structured algebraic assumptions like the discrete logarithm problem [49, 98], pairing-based problems [28, 103], and the Shortest Integer Solution problem (SIS) [57].

### 4.3.3 Non-Interactive Commitments

A *non-interactive commitment scheme* $\mathsf{COM}$ is a tuple of algorithms:

- A *commitment generation algorithm* $\mathsf{Commit}(m; r)$ that takes as input a message $m \in \{0,1\}^{\ell_m(\lambda)}$ (for some message space $\mathcal{M}$), and some randomness $r \in \{0,1\}^{\ell_r(\lambda)}$, and returns a commitment $\mathsf{com} \in \{0,1\}^{\ell_c(\lambda)}$. Here $\ell_m, \ell_r, \ell_c$ are some polynomials in $\lambda$, the security parameter.
- The *opening* of the commitment $\mathsf{com}$. In our case, this is simply the message $m$ and the randomness $r$.

We will require two standard properties of non-interactive commitments: perfect binding and computational hiding.

**Definition 18** (Perfectly binding commitment)**.** *A non-interactive commitment scheme* $\mathsf{COM}$ *is* perfectly binding *if for all* $m_0, m_1 \in \{0,1\}^{\ell_m(\lambda)}$ *such that* $m_0 \neq m_1$ *it holds that*

$$\{\mathsf{Commit}(m_0; r_0)\}_{r_0 \in \{0,1\}^{\ell_r(\lambda)}} \bigcap \{\mathsf{Commit}(m_1; r_1)\}_{r_1 \in \{0,1\}^{\ell_r(\lambda)}} = \varnothing.$$

**Definition 19** (Computationally hiding commitment)**.** *A non-interactive commitment scheme* $\mathsf{COM}$ *is* computationally hiding *if for every polynomially bounded function* $\alpha(\cdot)$ *and every PPT adversary* $\mathcal{A}$ *there exists a negligible function* $\mathsf{negl}[]$ *such that the probability of winning the following game is upper bounded by* $1/2 + \mathsf{negl}$*, where* $\lambda$ *is the security parameter:*

1. *The adversary* $\mathcal{A}(1^\lambda)$ *samples distinct messages* $m_0, m_1 \in \{0,1\}^{\alpha(\lambda)}$ *and sends them to the challenger;*
2. *The challenger samples a bit* $b \leftarrow\!\!\$\ \{0,1\}$*, computes a commitment* $\mathsf{com} = \mathsf{Commit}(m_b; r)$ *to* $m_b$*, and sends* $\mathsf{com}$ *to* $\mathcal{A}$*.*
3. *The adversary* $\mathcal{A}$ *outputs a bit* $b'$ *and wins if and only if* $b' = b$*.*

As mentioned before, non-interactive commitments can be instantiated from a variety of concrete assumptions including factoring [25, 58, 105], more recently from LWE and LPN [62], and even from the general assumption of injective one-way functions. While black-box separations between general one-way functions and non-interactive commitments are known [89], non-interactive commitments are fundamental and one of the weakest complexity-theoretic cryptographic assumptions.

## 4.4 PVSS-based YOSO$^{\mathsf{WCC}}$ Randomness Generation

We introduce a randomness generation scheme which relies on publicly verifiable secret sharing (PVSS). Before going into our protocol, we briefly explain what a PVSS is.

### 4.4.1 Publicly Verifiable Secret Sharing

Recall the definition of Publicly Verifiable Secret Sharing (PVSS) from [34]. In PVSS, a dealer $D$ shares a secret to a set of $n$ parties $\mathcal{P} = \{P_1, \cdots, P_n\}$. A $(t, n)$-PVSS protocol ensures that a secret is split in a way that allows $t + 1$ parties to reconstruct a secret, but at the same time, knowing $t$ shares does not reveal any information about the secret. Any external verifier $V$ is able to check that $D$ acts honestly. More formally, a PVSS protocol consists of the algorithms $(\mathsf{Setup}, \mathsf{Dist}, \mathsf{Verif}, \mathsf{Reconstr\text{-}Dec}, \mathsf{Reconstr\text{-}Pool})$, where $\mathsf{Setup} = (\mathsf{Setup}_\pi, \mathsf{Setup}_{\mathsf{PKI}})$, and which denote the following:

- **Setup:** Consists of $(\mathsf{Setup}_\pi, \mathsf{Setup}_{\mathsf{PKI}})$, which take security parameter $\lambda$ as input. In $\mathsf{Setup}_\pi$, the parameters of the proof system are generated in a trusted fashion. Using $\mathsf{Setup}_{\mathsf{PKI}}$, every party generates a public key $pk_i$ and withholds the corresponding secret key $sk_i$.
- **Distribution:** The dealer creates shares $s_1, \cdots, s_n$ for the secret $s$, encrypts share $s_i$ with the key $pk_i$ for $i = \{1, \cdots, n\}$ and publishes these encryptions $\hat{s}_i$, together with a proof $\text{PROOF}_D$ that these are indeed encryptions of a valid sharing of some secret.
- **Verification:** In this phase, any external $V$ (not necessarily being a participant in the protocol) can verify non-interactively, given all the public information until this point, that the values $\hat{s}_i$ are encryptions of a valid sharing of some secret.
- **Reconstruction:** This phase is divided in two.
  *Decryption of the shares:* This phase can be carried out by any set $Q$ of $t + 1$ or more parties. Every party $P_i$ in $Q$ decrypts the share $s_i$ from the ciphertext $\hat{s}_i$ by using its secret key $sk_i$, and publishes $s_i$ together with a (non-interactive) zero-knowledge proof $\text{PROOF}_i$ that this value is indeed a correct decryption of $\hat{s}_i$.
  *Share pooling:* Any external verifier $V$ (not necessarily being a participant in the protocol) can now execute this phase. $V$ first checks whether the proofs $\text{PROOF}_i$ are correct. If the check passes for less than $t + 1$ parties in $Q$ then $V$ aborts; otherwise $V$ applies a reconstruction procedure to the set $s_i$ of shares corresponding to parties $P_i$ that passed the checks.

A PVSS protocol $(\mathsf{Setup}, \mathsf{Dist}, \mathsf{Verif}, \mathsf{Reconstr\text{-}Dec}, \mathsf{Reconstr\text{-}Pool})$ must provide three

security guarantees: Correctness, Verifiability and IND1-Secrecy. These properties are defined below:

- **Correctness:** If the dealer and all players in $Q$ are honest, then all checks in the verification and reconstruction phases pass, and the secret can be reconstructed from the information published by the players in $Q$ during reconstruction.
- **Verifiability:** If the check in the Verification phase passes, then with high probability the values $\hat{s}_i$ are encryptions of a valid sharing of some secret. Furthermore, if the check in the Reconstruction phase passes, then the values $s_i$ are indeed the shares of the secret distributed by $D$.
- **IND1-Secrecy:** Prior to the reconstruction phase, the public information together with the secret keys $sk_i$ of any set of at most $t$ players gives no information about the secret.

We first formally specify the IND-1 secrecy of the scheme:

**Definition 20.** *We say that the PVSS is* IND1-secret *if for any PPT adversary A corrupting at most t parties, A has negligible advantage in the following game played against a challenger C.*

1. *C runs the Setup phase of the PVSS and sends all public information to A. Moreover, it creates secret and public keys for all uncorrupted parties, and sends the corresponding public keys to A.*
2. *A creates secret keys for the corrupted parties and sends the corresponding public keys to C.*
3. *C chooses values $x_0$ and $x_1$ at random in the space of secrets. Furthermore it chooses $b \leftarrow \{0,1\}$ uniformly at random. It runs the Distribution phase with $x_0$ as secret. It sends A all public information generated in that phase, together with $x_b$.*
4. *A outputs a guess $b' \in \{0,1\}$.*

*The advantage of A is defined as $|\Pr[b = b'] - \frac{1}{2}|$.*

In addition to the above IND-1 secrecy, as well as correctness and verifiability, which have been defined previously, we require our PVSS to be *non-malleable*.

**A Note on Non-Malleability.** To obtain the non-malleability guarantee required by our construction we informally require the compatibility with the (unbounded) computational zero-knowledge property and the simulation-extractability property of the underlying proof system. In more detail, consider the proof system $(G, P, V)$ for a relation $R$, where

- $\sigma \leftarrow G(1^\lambda)$: given a security parameter $\lambda$, the key generation algorithm produces a crs $\sigma$.

- $\pi \leftarrow V(\sigma, x, w)$: the prover algorithm takes as input a crs $\sigma$, a statement $x$, and a witness, and produces a proof $\pi$.
- $b \leftarrow G(\sigma, x, \pi)$: the verifier algorithm takes as input a crs $\sigma$, a statement $x$ produces a crs $\sigma$.

We consider non-interactive proofs, and require that in addition to the standard completeness and soundness guarantees, the proof system has the following properties:

**Definition 21** ((unbounded) computational zero-knowledge)**.** *A non-interactive proof system $(G, P, V)$ is zero-knowledge for a relation $R$, if there exists a PPT simulator consisting of a tuple of PPT algorithms $S = (S_1, S_2)$, such that for all PPT adversaries $A$ holds that*

$$Pr|[\sigma \leftarrow G(1^\lambda) : A^{P(\sigma,\cdot,\cdot)}(\sigma) = 1] - Pr[(\sigma, \tau) \leftarrow S_1(1^\lambda) : A^{S(\sigma,\tau,\cdot,\cdot)}(\sigma) = 1]| < \mathsf{negl},$$

*where $S(\sigma, \tau, x, w) = S_2(\sigma, \tau, x)$ if $(x, w) \in R$.*

We call non-interactive zero-knowledge proof systems *NIZKs*. We additionally require the following:

**Definition 22** (Simulation Extractability [68])**.** *We call a NIZK $(G, P, V)$ simulation-extractable if there exists a tuple of PPT algorithms $(SE_1, E)$, such that $SE_1$ output a triple $(\sigma, \tau, \zeta)$, which is identical to the output of $S_1$ when restricted to the first two parts, and for all PPT adversaries $A$ holds that*

$$\Pr\left[\begin{array}{ccc} (\sigma,\tau,\zeta) \leftarrow SE_1(1^\lambda) & & (x,\pi) \notin Q \\ (x,\pi) \leftarrow A^{S_2(\sigma,\tau,\cdot)}(\sigma,\zeta) & : & (x,w) \notin R \\ w \leftarrow E(\sigma,\zeta,x,\pi) & & V(\sigma,x,\pi) = 1 \end{array}\right] < \mathsf{negl},$$

*where $Q$ is the list of $A$'s simulation queries and responses.*

Additionally, we require that the proof system is "decoupled" from the encryption scheme used in the PVSS, in the sense that the keys and the proof crs are generated independently of each other, and the distribution algorithm can be split into two steps, first of which produces the ciphertexts $\hat{s}_i$, and the second of which produces a NIZK proof given these ciphertexts.

Note that such PVSS scheme can be trivially built from a public-key encryption scheme and a simulation-extractable NIZK as follows. First, the dealer splits its secret using a $(t, n)$ Shamir secret sharing, and encrypts each share using the public keys of the share receivers. Then, the dealer generates a NIZK proof confirming that it knows shares underlying the ciphertexts, and these lie on a polynomial of degree at most $t$. Anyone can verify the correctness of the dealer's sharing using the verifier for the NIZK proof. In the reconstruction phase, every share recipient decrypts its share, and generates a proof that the decrypted

value indeed corresponds to the ciphertext published by the dealer. Given $t+1$ honest share recipients, the correctness of the scheme follows from the correctness of the encryption scheme, completeness and soundness of the NIZK, and correctness of Shamir's secret sharing. Privacy follows from the security of the encryption scheme, zero-knowledge of the NIZK, and security of Shamir's secret sharing. Verifiability follows from the simulation-extractability of the NIZK, correctness of the encryption, and and the fact that any $t+1$ shares fix the secret.

### 4.4.2 Our PVSS-Based Randomness Generation Protocol

Our protocol is in the sending-leaks model (thus also secure in the execution-leaks model). We describe the scheme and outline the security proof.

The high-level idea of the scheme is the following: given $n = 3t+2$ parties, split them into two groups $\mathcal{P}$ and $\mathcal{P}'$ of size $t+1$ and $2t+1$, respectively. We dub the parties from the first group *dealers*, denoted by $P_1, P_2, \cdots, P_{t+1}$, and the parties from the second group *decryptors*, denoted by $P'_1, P'_2, \cdots, P'_{2t+1}$. Let $(\mathsf{Setup} := (\mathsf{Setup}_\pi, \mathsf{Setup}_{\mathsf{PKI}}), \mathsf{Dist}, \mathsf{Verif}, \mathsf{RDec}, \mathsf{RPool})$ denote a $(t, 2t+1)$-PVSS protocol. The protocol starts with a "sharing" phase, where every $P_i$ is executed one after another and acts as a PVSS dealer distributing its secret to the decryptors in $\mathcal{P}'$. Then, decryptors $P'_i \in \mathcal{P}'$ are executed one after another, and each decryptor $P'_i$ executes the share decryption part of the PVSS reconstruction phase for each dealer $P_i$. Finally, any party $C$ can execute the share pooling phase of the PVSS reconstruction phase in order to obtain the secret shared by each dealer. We give the full scheme in Protocol 15.

For security, we need our PVSS to be non-malleable, which can be naively achieved by using simulation-extractable NIZKs [68] as PVSS proofs. Intuitively, a strawman PVSS scheme which provides the required non-malleability works as follows: Share the secret using a $(t, n)$ secret sharing scheme (e.g, Shamir's secret sharing [99]), encrypt each share using a public key of the corresponding share receiver, and append a simulation-extractable NIZK proof confirming that the dealer knows the shares underlying the ciphertexts, and these shares correspond to the $(t, n)$ secret sharing. The reconstruction works by having each receiver decrypt its share, and publish a proof confirming that it knows a secret key such that the decryption of the corresponding ciphertext results in the stated value. The communication complexity is $O(n^2|c| + n|p|)$, where $|c|$ is the length of a single ciphertext, and $|p|$ of a proof.

**Theorem 8.** *Assuming public key encryption and simulation-extractable NIZKs, there*

**Protocol 15** Randomness Beacon from PVSS in the Sending-Leaks Model.

**Setup:** PVSS $\mathsf{Setup}_\pi$ algorithm is executed in a trusted fashion to obtain the common reference string $\mathsf{crs}$. Public key of every party in the protocol is generated according to $\mathsf{Setup}_{\mathsf{PKI}}$.

**Sharing phase:** Each party $P_i$, $i \in [t+1]$ does the following:

1. $P_i$ samples $x_i$ from $\{0,1\}$ uniformly at random.
2. $P_i$ uses PVSS algorithm $\mathsf{Dist}$ as the dealer to distribute shares of $x_i$ to the parties $P'_1, \cdots, P'_{2t+1}$:

$$(\{\hat{s}_j^{(i)}\}_{j\in[2t+1]}, \text{PROOF}_D^{(i)}) \leftarrow \mathsf{Dist}(x_i, \{pk_{P'_j}\}_{j\in[2t+1]}, \mathsf{crs}).$$

3. $P_i$ publishes $(\{\hat{s}_j^{(i)}\}_{j\in[2t+1]}, \text{PROOF}_D^{(i)})$.

**Reconstruction phase:** Each party $P'_j$, $j \in [2t+1]$ does the following:

1. For each $P_i$, $P'_j$ uses $\mathsf{Verif}(\{\hat{s}_j^{(i)}\}_{j\in[2t+1]}, \text{PROOF}_D^{(i)}, \mathsf{crs})$ to verify that $P_i$ dealt a valid secret. For each $P_i$ who passed the check, $P'_j$ verifies that the proof $\text{PROOF}_D^{(i)}$ and every encryption $\hat{s}_m^{(i)}$ distributed by $P_i$ is not the same as one distributed by any dealer $P_k$, where $k < i$. Denote $P_i$ as *valid* if so.
2. For each valid $P_i$, $P'_j$ uses the PVSS algorithm $\mathsf{RDec}(\hat{s}_j^{(i)}, sk'_{P'_j}, \mathsf{crs})$ to obtain $(s_j^{(i)}, \text{PROOF}_j^{(i)})$, and publishes this pair.

Any party $C$ can use the PVSS algorithm $\mathsf{RPool}$ on information published by the parties $P'_1, \cdots, P'_{2t+1}$ to obtain $x_i$. Output $\bigoplus_{i\in I} x_i$, where $I$ denotes an index set of dealers for which $C$ obtained the secret using $\mathsf{RPool}$.

*exists a YOSO$^{\mathsf{WCC}}$ $(t, 3t+2)$-computationally secure randomness generation protocol in the sending-leaks model.*

## 4.5 PVSS-based YOSO$^{\mathsf{WCC}}$ Randomness Generation: Security Proof

We prove the theorem via a hybrid argument. In the following, let $\lambda$ denote the security parameter.

**Hybrid $\mathbf{H_0}$** : This hybrid corresponds to the real world experiment as defined in Definition 16 with the bit $b$ fixed to 0. Specifically, the challenger interacts with a PPT adversary $A$ that corrupts a set $M$ of parties, where $|M| = t$, and interacts with a set $H$, $|H| = 2t+2$, of honest parties to obtain the coin output of the Protocol 15, and forwards

this output to a PPT distinguisher $D$.

1. $\mathsf{crs} \leftarrow \mathsf{Setup}_\pi(1^\lambda)$.
2. For $P_i' \in H$, let $(pk_i', sk_i') \leftarrow \mathsf{Setup}_{\mathsf{PKI}}(1^\lambda)$.
3. For $P_i' \in M$, let $pk_i' \leftarrow A(\{pk_j'\}_{P_j' \in H})$.
4. For $P_i \in H$:
   (a) Sample $x_i \leftarrow \{0,1\}$ uniformly at random.
   (b) Publish
   $$(\{\hat{s}_j^{(i)}\}_{j\in[2t+1]}, \text{PROOF}_D^{(i)}) \leftarrow \mathsf{Dist}(x_i, \{pk_j'\}_{j\in[2t+1]}, \mathsf{crs}).$$
5. For $P_i' \in H$:
   (a) For each $P_i$ such that
   $$\mathsf{Verif}(\{\hat{s}_j^{(i)}\}_{j\in[2t+1]}, \text{PROOF}_D^{(i)}, \mathsf{crs}) = 1,$$
   check whether $\text{PROOF}_D^{(i)}$ and every encryption $\hat{s}_m^{(i)}$ distributed by $P_i$ is not the same as one distributed by any dealer $P_k$, where $k < i$. Denote $P_i$ as *valid* if so.
   (b) For each valid $P_i$ publish
   $$(s_j^{(i)}, \text{PROOF}_j^{(i)}) \leftarrow \mathsf{RDec}(\hat{s}_j^{(i)}, sk_i', \mathsf{crs}).$$
6. For each $i \in [t+1]$ let
   $$\mathsf{out}_i \leftarrow \mathsf{RPool}((s_j^{(i)}, \text{PROOF}_j^{(i)}, \mathsf{crs})_{j\in[2t+1]}).$$
7. $\mathsf{out} \leftarrow \bigoplus_{i\in I} \mathsf{out}_i$, where $I$ denotes the index set such that for every $i \in I$ holds $\mathsf{out}_i \neq \perp$.
8. $b' \leftarrow D(\mathsf{out})$.

Here, $M$ denotes the set of parties controlled by $A$, and $H$ is the set of honest parties.

**Hybrid $\mathbf{H_1}$** : This hybrid is the same as before, except that instead of computing proofs $\text{PROOF}_D$ and $\text{PROOF}_i$ honestly, the proofs are generated using a simulator.

The game becomes the following (changes from the previous hybrid in red):

1. $(\mathsf{crs}, \tau) \leftarrow \mathsf{SimSetup}_\pi(1^\lambda)$.
2. For $P_i' \in H$, let $(pk_i', sk_i') \leftarrow \mathsf{Setup}_{\mathsf{PKI}}(1^\lambda)$.
3. For $P_i' \in M$, let $pk_i' \leftarrow A(\{pk_j'\}_{P_j' \in H})$.
4. For $P_i \in H$:
   (a) Sample $x_i \leftarrow \{0,1\}$ uniformly at random.
   (b) Publish
   $$(\{\hat{s}_j^{(i)}\}_{j\in[2t+1]}, \text{PROOF}_D^{(i)}) \leftarrow \mathsf{SimDist}(x_i, \{pk_j'\}_{j\in[2t+1]}, \mathsf{crs}, \tau).$$

5. For $P_i' \in H$:
   (a) For each $P_i$ such that $\mathsf{Verif}(\{\hat{s}_j^{(i)}\}_{j\in[2t+1]}, \text{PROOF}_D^{(i)}, \mathsf{crs}) = 1$, check whether $\text{PROOF}_D^{(i)}$ and every encryption $\hat{s}_m^{(i)}$ distributed by $P_i$ is not the same as one distributed by any dealer $P_k$, where $k < i$. Denote $P_i$ as *valid* if so.
   (b) For each valid $P_i$ publish

$$(s_j^{(i)}, \text{PROOF}_j^{(i)}) \leftarrow \mathsf{SimRDec}(\hat{s}_j^{(i)}, sk_i', \mathsf{crs}, \tau).$$

6. For each $i \in [t+1]$ let

$$\mathsf{out}_i \leftarrow \mathsf{RPool}((s_j^{(i)}, \text{PROOF}_j^{(i)}, \mathsf{crs})_{j\in[2t+1]}).$$

7. $\mathsf{out} \leftarrow \bigoplus_{i\in I} \mathsf{out}_i$, where $I$ denotes the index set such that for every $i \in I$ holds $\mathsf{out}_i \neq \perp$.
8. $b' \leftarrow D(\mathsf{out})$.

**Lemma 20.** *Assuming that the proof system used in the PVSS scheme has the zero-knowledge property, the outputs of experiments* $\mathbf{H_0}$ *and* $\mathbf{H_1}$ *are computationally indistinguishable.*

*Proof.* This is a series of hybrids in which each honest proof is replaced one by one. Given a PPT adversary $A$ and a distinguisher $D$ who is able to distinguish between the two hybrids given the output of the challenger's interaction with $A$, we construct an adversary $B$ on the zero-knowledge property of the underlying PVSS scheme as follows. $B$ obtains the setup information for the proof system used in PVSS from its challenger $C$, and forwards this information to the adversary $A$. Then, $B$ follows the game as specified by the previous hybrid (using the challenger to obtain simulated proofs if required by the previous hybrid), except that when $B$ must produce the proof $\text{PROOF}_D$ ($\text{PROOF}_i$), $B$ uses the simulated proof which it obtains from $C$. $B$ forwards the protocol output of its interaction with $A$ to $D$. If $D$ outputs "Hybrid 0", $B$ outputs "Real prover", otherwise "Simulator". As the only difference between the two hybrids is the way that $\text{PROOF}_D$ ($\text{PROOF}_i$) is being generated, $B$'s advantage is the same as $D$'s. Thus, if the advantage of the pair $A$ and $D$ is non-negligible, $B$'s advantage is non-negligible as well. □

**Hybrid $\mathbf{H_2}$** : This hybrid is the same as before, except that the challenger uses the extractor $\mathsf{Ext}$ to extract the corresponding secret from each $\text{PROOF}_D$ that verifies correctly. The challenger aborts if the extraction fails.

1. $(\mathsf{crs}, \tau, \zeta) \leftarrow \mathsf{SimExtSetup}_\pi(1^\lambda)$.
2. For $P_i' \in H$, let $(pk_i', sk_i') \leftarrow \mathsf{Setup}_{\mathsf{PKI}}(1^\lambda)$.
3. For $P_i' \in M$, let $pk_i' \leftarrow A(\{pk_j'\}_{P_j' \in H})$.
4. For $P_i \in H$:
   (a) Sample $x_i \leftarrow \{0,1\}$ uniformly at random.
   (b) Publish
   $$(\{\hat{s}_j^{(i)}\}_{j\in[2t+1]}, \text{PROOF}_D^{(i)}) \leftarrow \mathsf{SimDist}(x_i, \{pk_j'\}_{j\in[2t+1]}, \mathsf{crs}, \tau).$$
5. For $P_i' \in H$:
   (a) For each $P_i$ such that
   $$\mathsf{Verif}(\{\hat{s}_j^{(i)}\}_{j\in[2t+1]}, \text{PROOF}_D^{(i)}, \mathsf{crs}) = 1,$$
   check whether $\text{PROOF}_D^{(i)}$ and every encryption $\hat{s}_m^{(i)}$ distributed by $P_i$ is not the same as one distributed by any dealer $P_k$, where $k < i$. Denote $P_i$ as *valid* if so.
   (b) For each valid $P_i \in M$ let $w \leftarrow \mathsf{Ext}(\text{PROOF}_D^{(i)}, \mathsf{crs}, \zeta)$. If
   $$(crs, \{\hat{s}_j^{(i)}\}_{j\in[2t+1]}, \text{PROOF}_D^{(i)}, w) \notin R,$$
   then abort.
   (c) For each valid $P_i$ publish
   $$(s_j^{(i)}, \text{PROOF}_j^{(i)}) \leftarrow \mathsf{SimRDec}(\hat{s}_j^{(i)}, sk_i', \mathsf{crs}, \tau).$$
6. For each $i \in [t+1]$ let
   $$\mathsf{out}_i \leftarrow \mathsf{RPool}((s_j^{(i)}, \{pk_i'\}_{i\in[2t+1]}, \text{PROOF}_j^{(i)}, \mathsf{crs})_{j\in[2t+1]}).$$
7. $\mathsf{out} \leftarrow \bigoplus_{i\in I} \mathsf{out}_i$, where $I$ denotes the index set such that for every $i \in I$ holds $\mathsf{out}_i \neq \perp$.
8. $b' \leftarrow D(\mathsf{out})$.

**Lemma 21.** *Assuming that the proof system used in the PVSS scheme has the simulation extractability property, the outputs of experiments* $\mathbf{H_1}$ *and* $\mathbf{H_2}$ *are computationally indistinguishable.*

*Proof.* This is a series of hybrids in which each adversarial proof which verifies correctly is handled one by one. In the $i$-th such hybrid step, given a PPT adversary $A$ and a distinguisher $D$ who is able to distinguish between the two hybrids given the output of the challenger's interaction with $A$, we construct an adversary $B$ on the simulation extractability

of the underlying PVSS scheme as follows. $B$ obtains the setup information for the proof system used in PVSS from its challenger $C$, and forwards this information to the adversary $A$. Then, $B$ follows the game as specified by the previous hybrid, except that it uses $C$ to obtain simulated proofs that $B$ is required to generate according to the protocol. When the adversary $A$ publishes the $i$-th adversarial proof, $B$ forwards this proof to its challenger $C$, and uses the extractor on this proof. If the extraction succeeds, $B$ forwards the protocol output of its interaction with $A$ to $D$, otherwise $B$ aborts. Note that the two hybrids are exactly the same when the proof extraction succeeds. Thus, we get that

$$\begin{aligned}
&\Pr[(A, D) \text{ wins}] \\
&= \Pr[(A, D) \text{ wins} | \text{Extr. succeeds}] \cdot \Pr[\text{Extr. succeeds}] \\
&+ \Pr[(A, D) \text{ wins} | \text{Extr. fails}] \cdot \Pr[\text{Extr. fails}] \\
&= \left(\frac{1}{2} + \mathsf{negl}\right)(1 - \Pr[\text{Extr. fails}]) \\
&+ \Pr[(A, D) \text{ wins} | \text{Extr. fails}] \cdot \Pr[\text{Extr. fails}] \\
&\leq \frac{1}{2} + \mathsf{negl} + \Pr[\text{Extr. fails}] \left(1 - \frac{1}{2} - \mathsf{negl}\right).
\end{aligned}$$

□

Therefore, we have that

$$\Pr[\text{Extr. fails}] \geq \frac{\Pr[A \text{ wins}] - \frac{1}{2} - \mathsf{negl}}{\frac{1}{2} - \mathsf{negl}}.$$

Note that $B$ wins whenever the extraction fails. Thus, if $A$ wins with some non-negligible advantage, $B$ wins with non-negligible probability as well.

**Hybrid $\mathbf{H_3}$** : This hybrid is the same as before, except that the protocol outcome computation is modified as follows: For the dealers *controlled by the adversary* which pass the check of the PVSS verification phase, instead of using the secrets obtained for these dealers during the reconstruction phase, the challenger uses the secrets that were extracted using the extractor $\mathsf{Ext}$.

1. $(\mathsf{crs}, \tau, \zeta) \leftarrow \mathsf{SimSetup}_\pi(1^\lambda)$.
2. For $P_i' \in H$, let $(pk_i', sk_i') \leftarrow \mathsf{Setup}_{\mathsf{PKI}}(1^\lambda)$.
3. For $P_i' \in M$, let $pk_i' \leftarrow A(\{pk_j'\}_{P_j' \in H})$.
4. For $P_i \in H$:
   (a) Sample $x_i \leftarrow \{0, 1\}$ uniformly at random.

(b) Publish

$$(\{\hat{s}_j^{(i)}\}_{j\in[2t+1]}, \text{PROOF}_D^{(i)}) \leftarrow \mathsf{SimDist}(x_i, \{pk'_j\}_{j\in[2t+1]}, \mathsf{crs}, \tau).$$

5. For $P'_i \in H$:
   (a) For each $P_i$ such that $\mathsf{Verif}(\{\hat{s}_j^{(i)}\}_{j\in[2t+1]}, \text{PROOF}_D^{(i)}, \mathsf{crs}) = 1$, check whether $\text{PROOF}_D^{(i)}$ and every encryption $\hat{s}_m^{(i)}$ distributed by $P_i$ is not the same as one distributed by any dealer $P_k$, where $k < i$. Denote $P_i$ as *valid* if so.
   (b) For each valid $P_i \in M$ let $w_i \leftarrow \mathsf{Ext}(\text{PROOF}_D^{(i)}, \mathsf{crs}, \tau, \zeta)$. If

$$(crs, \{\hat{s}_j^{(i)}\}_{j\in[2t+1]}, \text{PROOF}_D^{(i)}, w) \notin R,$$

   then abort.
   (c) For each valid $P_i$ publish

$$(s_j^{(i)}, \text{PROOF}_j^{(i)}) \leftarrow \mathsf{SimRDec}(\hat{s}_j^{(i)}, sk'_i, \mathsf{crs}, \tau)$$

6. For each $i \in [t+1]$ let

$$\mathsf{out}_i \leftarrow \mathsf{RPool}((s_j^{(i)}, \{pk'_i\}_{i\in[2t+1]}, \text{PROOF}_j^{(i)}, \mathsf{crs})_{j\in[2t+1]}).$$

7. $\mathsf{out} \leftarrow \bigoplus_{i\in I\cap H} \mathsf{out}_i \oplus \tilde{x}_{i\in\tilde{I}}$, where $I$ denotes the index set such that for every $i \in I$ holds $\mathsf{out}_i \neq \perp$, $\tilde{I} \subseteq M$ denotes the index set such that for each $i \in \tilde{I}$ holds $P'_i$ is valid, and $\tilde{x}_i$ is the secret corresponding to the witness $w_i$.
8. $b' \leftarrow D(\mathsf{out})$.

**Lemma 22.** *Assuming that the PVSS scheme is verifiable, the outputs of experiments* $\mathbf{H_2}$ *and* $\mathbf{H_3}$ *are computationally indistinguishable.*

*Proof.* This is a series of hybrids, where we change the contribution of each malicious dealer one-by-one. By the verifiability property of the PVSS scheme, if the verifications checks passes, then the sharing phase determines a unique secret, and this secret will be reconstructed by the end of the reconstruction phase. As the extractor $\mathsf{Ext}$ extracted a valid secret $s^*$, and the secret determined by the sharing phase is *unique* and is guaranteed to be reconstructed by the end of the protocol, $s^*$ is exactly the secret that the parties would have reconstructed for this dealer by the end of the reconstruction phase. □

**Hybrid $\mathbf{H_4}$** : This hybrid is the same as before, except that the protocol outcome computation is modified as follows: For the *honest* dealers, instead of using the secrets obtained for these dealers during the reconstruction phase, the challenger uses the secrets that these dealers shared during the sharing phase:

1. $(\mathsf{crs}, \tau, \zeta) \leftarrow \mathsf{SimSetup}_\pi(1^\lambda)$.
2. For $P'_i \in H$, let $(pk'_i, sk'_i) \leftarrow \mathsf{Setup}_{\mathsf{PKI}}(1^\lambda)$.
3. For $P'_i \in M$, let $pk'_i \leftarrow A(\{pk'_j\}_{P'_j \in H})$.
4. For $P_i \in H$:
   (a) Sample $x_i \leftarrow \{0,1\}$ uniformly at random.
   (b) Publish

$$(\{\hat{s}_j^{(i)}\}_{j\in[2t+1]}, \text{PROOF}_D^{(i)}) \leftarrow \mathsf{SimDist}(x_i, \{pk'_j\}_{j\in[2t+1]}, \mathsf{crs}, \tau).$$

5. For $P'_i \in H$:
   (a) For each $P_i$ such that

$$\mathsf{Verif}(\{\hat{s}_j^{(i)}\}_{j\in[2t+1]}, \text{PROOF}_D^{(i)}, \mathsf{crs}) = 1,$$

   check whether $\text{PROOF}_D^{(i)}$ and every encryption $\hat{s}_m^{(i)}$ distributed by $P_i$ is not the same as one distributed by any dealer $P_k$, where $k < i$. Denote $P_i$ as *valid* if so.
   (b) For each valid $P_i \in M$ let $w_i \leftarrow \mathsf{Ext}(\text{PROOF}_D^{(i)}, \mathsf{crs}, \tau, \zeta)$. If

$$(crs, \{\hat{s}_j^{(i)}\}_{j\in[2t+1]}, \text{PROOF}_D^{(i)}, w) \notin R,$$

   then abort.
   (c) For each valid $P_i$ publish

$$(s_j^{(i)}, \text{PROOF}_j^{(i)}) \leftarrow \mathsf{SimRDec}(\hat{s}_j^{(i)}, sk'_i, \mathsf{crs}, \tau).$$

6. For each $i \in [t+1]$ let

$$\mathsf{out}_i \leftarrow \mathsf{RPool}((s_j^{(i)}, \{pk'_i\}_{i\in[2t+1]}, \text{PROOF}_j^{(i)}, \mathsf{crs})_{j\in[2t+1]}).$$

7. $\mathsf{out} \leftarrow \bigoplus_{i\in H} x_i \oplus \tilde{x}_{i\in\tilde{I}}$, where $\tilde{I} \subseteq M$ denotes the index set such that for each $i \in \tilde{I}$ holds $P'_i$ is valid, and $\tilde{x}_i$ is the secret corresponding to the witness $w_i$.
8. $b' \leftarrow D(\mathsf{out})$.

**Lemma 23.** *Assuming that the PVSS scheme is correct, the outputs of experiments* $\mathbf{H_3}$ *and* $\mathbf{H_4}$ *are indistinguishable.*

*Proof.* This is a series of hybrids, where we change the contribution of each honest dealer one-by-one. By the correctness property of the PVSS scheme all verifications checks in the protocol pass (for the secret distributed by this honest dealer) and the reconstructed secret is the same as the honest dealer shared during the sharing phase. □

**Hybrid $\mathbf{H_5}$** : This hybrid is the same as before, except that the challenger stops its interaction with $A$ after the sharing phase.

1. $(\mathsf{crs}, \tau, \zeta) \leftarrow \mathsf{SimSetup}_\pi(1^\lambda)$.
2. For $P'_i \in H$, let $(pk'_i, sk'_i) \leftarrow \mathsf{Setup}_{\mathsf{PKI}}(1^\lambda)$.
3. For $P'_i \in M$, let $pk'_i \leftarrow A(\{pk'_j\}_{P'_j \in H})$.
4. For $P_i \in H$:
   (a) Sample $x_i \leftarrow \{0,1\}$ uniformly at random.
   (b) Publish

$$(\{\hat{s}_j^{(i)}\}_{j\in[2t+1]}, \text{PROOF}_D^{(i)}) \leftarrow \mathsf{SimDist}(x_i, \{pk'_j\}_{j\in[2t+1]}, \mathsf{crs}, \tau).$$

5. For $P'_i \in H$:
   (a) For each $P_i$ such that $\mathsf{Verif}(\{\hat{s}_j^{(i)}\}_{j\in[2t+1]}, \text{PROOF}_D^{(i)}, \mathsf{crs}) = 1$, check whether $\text{PROOF}_D^{(i)}$ and every encryption $\hat{s}_m^{(i)}$ distributed by $P_i$ is not the same as one distributed by any dealer $P_k$, where $k < i$. Denote $P_i$ as *valid* if so.
   (b) For each valid $P_i \in M$ let $w_i \leftarrow \mathsf{Ext}(\text{PROOF}_D^{(i)}, \mathsf{crs}, \tau, \zeta)$. If

$$(crs, \{\hat{s}_j^{(i)}\}_{j\in[2t+1]}, \text{PROOF}_D^{(i)}, w) \notin R,$$

   then abort.
   (c) For each valid $P_i$ publish

$$(s_j^{(i)}, \text{PROOF}_j^{(i)}) \leftarrow \mathsf{SimRDec}(\hat{s}_j^{(i)}, sk'_i, \mathsf{crs}, \tau).$$

6. For each $i \in [t+1]$ let

$$\mathsf{out}_i \leftarrow \mathsf{RPool}((s_j^{(i)}, \{pk'_i\}_{i\in[2t+1]}, \text{PROOF}_j^{(i)}, \mathsf{crs})_{j\in[2t+1]}).$$

7. $\mathsf{out} \leftarrow \bigoplus_{i\in I\cap H} x_i \oplus \tilde{x}_{i\in\tilde{I}}$, where $\tilde{I} \subseteq M$ denotes the index set such that for each $i \in \tilde{I}$ holds $P'_i$ is valid, and $\tilde{x}_i$ is the secret corresponding to the witness $w_i$.
8. $b' \leftarrow D(\mathsf{out})$.

**Lemma 24.** *The outputs of experiments* $\mathbf{H_4}$ *and* $\mathbf{H_5}$ *are indistinguishable.*

*Proof.* Note that in the previous hybrid the protocol output, which is exactly the input of the distinguisher $D$, was *already* fixed and could be computed by the challenger by the end of the sharing phase. Thus, nothing changed. □

**Hybrid $\mathbf{H_6}$** : This hybrid is the same as before, except that in the beginning of the sharing phase each honest dealer $P_i$ now chooses a value $x'_i$ uniformly at random. Each encryption of a share sent by an honest dealer to a party $P_j$ is now changed to an encryption of a corresponding share of $x'_i$.

1. $(\mathsf{crs}, \tau, \zeta) \leftarrow \mathsf{SimSetup}_\pi(1^\lambda)$.
2. For $P'_i \in H$, let $(pk'_i, sk'_i) \leftarrow \mathsf{Setup}_{\mathsf{PKI}}(1^\lambda)$.
3. For $P'_i \in M$, let $pk'_i \leftarrow A(\{pk'_j\}_{P'_j \in H})$.
4. For $P_i \in H$:
   (a) Sample $x_i \leftarrow \{0,1\}$ uniformly at random.
   (b) Sample $x'_i \leftarrow \{0,1\}$ uniformly at random.
   (c) Publish
   $$(\{\hat{s}_j^{(i)}\}_{j\in[2t+1]}, \text{PROOF}_D^{(i)}) \leftarrow \mathsf{SimDist}(x'_i, \{pk'_j\}_{j\in[2t+1]}, \mathsf{crs}, \tau).$$
5. For $P'_i \in H$:
   (a) For each $P_i$ such that
   $$\mathsf{Verif}(\{\hat{s}_j^{(i)}\}_{j\in[2t+1]}, \text{PROOF}_D^{(i)}, \mathsf{crs}) = 1,$$
   check whether $\text{PROOF}_D^{(i)}$ and every encryption $\hat{s}_m^{(i)}$ distributed by $P_i$ is not the same as one distributed by any dealer $P_k$, where $k < i$. Denote $P_i$ as *valid* if so.
   (b) For each valid $P_i \in M$ let $w_i \leftarrow \mathsf{Ext}(\text{PROOF}_D^{(i)}, \mathsf{crs}, \tau, \zeta)$. If
   $$(crs, \{\hat{s}_j^{(i)}\}_{j\in[2t+1]}, \text{PROOF}_D^{(i)}, w) \notin R,$$
   then abort.
6. $\mathsf{out} \leftarrow \bigoplus_{i\in I\cap H} x_i \oplus \tilde{x}_{i\in\tilde{I}}$, where $I' \subseteq M$ denotes the index set such that for each $i \in \tilde{I}$ holds $P'_i$ is valid, and $\tilde{x}_i$ is the secret corresponding to the witness $w_i$.
7. $b' \leftarrow D(\mathsf{out})$.

**Lemma 25.** *Assuming that the PVSS scheme satisfies the IND1-secrecy property, the outputs of experiments* $\mathbf{H_6}$ *and* $\mathbf{H_7}$ *are computationally indistinguishable.*

*Proof.* This is a series of hybrids, where we change sets encryptions sent by each honest dealer $D_i$ one-by-one. Given a PPT adversary $A$ and a distinguisher $D$ who is able to distinguish between the two hybrids given the output of the challenger's interaction with $A$, we construct an adversary $B$ on the IND1-secrecy property of the underlying PVSS scheme as follows. Upon obtaining the PVSS setup information as well as the public keys of the honest parties from its challenger $C$, $B$ generates new PVSS setup, and forwards

this setup information to the adversary $A$, along with the public keys of the honest parties generated by $C$. Then, $B$ forwards the public keys supplied by $A$ to $C$. $B$ follows the game as specified by the previous hybrid, except when it needs to act as the dealer $D_i$. Then, upon obtaining the ciphertexts, $\text{PROOF}_D$, and $x_b$ from its challenger, $B$ generates a new $\text{PROOF}'_D$ using the simulator for the setup generated by $B$, and forwards the ciphertexts along with the new $\text{PROOF}'_D$ to the adversary $A$. Then, $B$ continues to follow the game as specified by the previous hybrid. When computing the output of the protocol, $B$ uses $x_b$ as the contribution from the honest dealer. $B$ outputs exactly what $D$ outputs. Note that if $x_b$ was $x_0$, the game played corresponds exactly to the game in the previous hybrid. Otherwise, the game played corresponds exactly to the game specified in the new hybrid. Thus, if the advantage of the pair $A$ and $D$ is non-negligible, $B$'s advantage is non-negligible as well. □

Note that in the last hybrid the information about each honest secret $x_i$ the adversary receives during the sharing phase of the protocol is completely independent of the honest secrets values $x_i$. This corresponds to the security game outlined in Definition 16 with the bit $b$ fixed to 1.

## 4.6 Protocols from One-Way Functions

We introduce a new building block called *Split-Dealer Verifiable Secret Sharing in YOSO*$^{\mathsf{WCC}}$, which can be constructed using any signature scheme. Later, we will formally describe how this building block can be used to construct an efficient randomness generation protocol with $n = 5t + 3$ roles for $t \in O(\text{poly})$.

### 4.6.1 Split-Dealer Verifiable Secret Sharing in YOSO$^{\mathsf{WCC}}$

We now describe our split-dealer verifiable secret sharing (SD-VSS) protocol, consisting of $n$ roles, divided into four categories: a dealer with an initial secret input $s$, a set of receivers, a resolver, and a set of reconstructors. The protocol satisfies the usual security guarantees for correctness, privacy, and verifiability defined below, with the caveat that *correctness and privacy are required only when both the dealer and resolver are honest.* Nevertheless, we will show in Section 4.6.2 that this notion will be enough to obtain randomness generation.

**SD-VSS in the execution-leaks model** We will first consider the execution-leaks model as the technical crux is easier to understand in this model. Informally, the scheme works as

follows: The dealer $P_1$ generates a symmetric bivariate polynomial $f(x, y)$ of degree at most $t$ in each variable uniformly at random, subject to the constraint that $P_1$'s secret is encoded at point $(0, 0)$. Then, $P_1$ sends a projection $f(i, y)$ to each *receiver* $P_i$ for $i = \{2, \cdots, 3t+2\}$. Here, the projection is a set of points $f(i, j)$ for $j \in [3t + 1]$, each signed by the dealer. Additionally, $P_1$ sends the bivariate polynomial to the party $P_{3t+3}$, which we call the *resolver*. Each $P_i$ verifies the projection it received, and if it is not valid, broadcasts a complaint about the dealer. Otherwise, $P_i$ sends a point $f(i, j)$, signed both by $P_i$ and the dealer, to party $P_j$. Additionally, $P_i$ verifies all points $f(k, i)$ it received from parties $P_k$, for $k < i$. If any of these points contain a dealer's signature on a message which is inconsistent with the corresponding share that $P_i$ received from the dealer, $P_i$ broadcasts these points along with dealer's signatures on them. If any point is missing or contains invalid signatures, $P_i$ broadcasts a complaint about the corresponding party $P_k$, and includes $f(i, k)$, as well as its own, and the dealer's signatures on this value. Finally, $P_i$ sends all shares it received, along with their signatures, to each of the parties $P_{3t+1}, \cdots, P_{5t+4}$, dubbed *reconstructors*. Now, the resolver responds to the complaint of each party $P_i$ about the dealer by broadcasting this party's projection. Then, each reconstructor verifies the shares it obtained from each receiver $P_i$. The share is valid if it contains at least $2t + 1$ points $f(i, j)$ which are correctly signed by $P_i$, $P_j$, and the dealer (or contained in the resolver's message). The shares of all valid receivers are then used to obtain $f(0, 0)$. See Protocol 16 for the full scheme. Here, we call $P_1$ the dealer, $P_2, \ldots, P_{3t+2}$ the receivers, $P_{3t+3}$ the resolver and $P_{3t+4}, \ldots, P_{4t+4}$ the reconstructors. We let $s$ be the secret input of the dealer $P_1$. We let $\mathsf{DS} = (\mathsf{KGen}, \mathsf{Sign}, \mathsf{Vf})$ denote a digital signatures scheme. The communication complexity of this construction is $O(n^3)$ (excluding any polynomial factors in the security parameter). We show that the protocol in the execution-leaks model satisfies the following security guarantees.

**Lemma 26** (Correctness)**.** *If the dealer and the resolver are honest, the sharing phase in Protocol 16 does not fail and the reconstructed secret after executing both the sharing and consequently the reconstruction phase is $s$.*

*Proof.* First, we show that the sharing phase does not fail. This is because all points signed by the dealer lie on the bivariate polynomial $F$, and any polynomial broadcasted by the dealer or point signed by the dealer is consistent with $F$. That is, any message `Complain` with a point and a dealer's signature that is broadcasted by a receiver lies on $F$, and any polynomial broadcasted by the resolver as a response to a message `ComplainPoly` lies on $F$ as well.

Moreover, consider each $i$-th honest receiver, for $i \in [1, 3t + 1]$. The receiver correctly received a signed degree-$t$ projection, so he did not broadcast any `ComplainPoly` message.

**Protocol 16** Split-Dealer Verifiable Secret Sharing, Execution-Leaks.

Each $P_i$, $i \in [1, 4t+4]$, does the following:

**Sharing phase:**

- If $P_i$ is the dealer (i.e., $i = 1$):
  1. Sample a symmetric polynomial $F(x, y)$ such that $F(0, 0) = s$.
  2. Sample $\left(\mathsf{vk}^D, \mathsf{sk}^D\right) \leftarrow \mathsf{DS.KGen}(1^\lambda)$. Publicly broadcast the public key $\mathsf{vk}^D$. Also privately send the polynomial $F$ to the resolver (party $P_{3t+3}$).
  3. Let $f_j := F(j, y)$, $j \in [3t+1]$, be the $j$-th vertical projection of $F$. Privately send to the $j$-th receiver all points of $f_j$ signed, i.e. send to party $P_{j+1}$ the pairs $\{f_j(y), \sigma^D_{j,y}\}_{y \in [3t+1]}$, where $\sigma^D_{j,y} = \mathsf{DS.Sign}_{\mathsf{sk}^D}(f_j(y))$.
- If $P_i$ is the $j$-th receiver (i.e. $i = j+1$ and $j \in [3t+1]$):
  1. Sample $\left(\mathsf{vk}^j, \mathsf{sk}^j\right) \leftarrow \mathsf{DS.KGen}(1^\lambda)$ and publicly broadcast the public key $\mathsf{vk}^j$.
  2. For the messages received from the dealer ($P_1$), $\{f_j(y), \sigma^D_{j,y}\}_{y \in [3t+1]}$:
     (a) Check that the signatures are correct according to the broadcasted public key $\mathsf{vk}^D$, i.e. $\mathsf{DS.Vf}_{\mathsf{vk}^D}(f_j(y), \sigma^D_{j,y}) = 1$ for each point.
     (b) Check that the points $\{f_j(y)\}_{y \in [3t+1]}$ lie on a degree-$t$ polynomial.
     (c) If any check fails, publicly broadcast a message `ComplainPoly`.
     (d) Otherwise, for each receiver $j < k < 3t+2$, send $(f_j(k), \sigma^D_{j,k}, \sigma^j_k)$ to the $k$-th receiver, where $\sigma^j_k = \mathsf{DS.Sign}_{\mathsf{sk}^j}(f_j(k))$.
  3. For the message $(f_k(j), \sigma^D_{k,j}, \sigma^k_j)$ received from the $k$-th receiver (i.e., party $P_{k+1}$) with $k < j$: check the signature validity, i.e. $\mathsf{DS.Vf}_{\mathsf{sk}^D}(f_k(j)) = \mathsf{DS.Vf}_{\mathsf{sk}^k}(f_k(j)) = 1$.
     (a) If the received signatures are incorrect (or no message was received) and no `ComplainPoly` was broadcast at step $2b$, broadcast a complaint message with the received point from the dealer and its signature: $(\texttt{Complain}, k, f_j(k), \sigma^D_{j,k})$.
     (b) If the received signatures are correct, no `ComplainPoly` is being broadcasted at step $2b$, do the following. If $f_k(j) \neq f_j(k)$, broadcast a complaint $(\texttt{Complain}, k, f_k(j), \sigma^D_{k,j}, f_j(k), \sigma^D_{j,k})$ and the sharing phase is failed. Otherwise, send $(f_j(k), \sigma^D_{j,k}, \sigma^k_j, \sigma^j_k)$ privately to every reconstructor.
     (c) If the received signatures are correct but `ComplainPoly` is being broadcasted at step $2b$, broadcast a complaint message with the received point from the $P_{k+1}$ and both signatures: $(\texttt{Complain}, k, f_k(j), \sigma^D_{k,j}, \sigma^k_j)$.
- If $P_i$ is the resolver: For each message `ComplainPoly` from the $j$-th receiver, publicly broadcast the projection polynomial $f_j$.

The sharing is considered to have failed if the polynomial output by the resolver does not have degree-$t$ or it is inconsistent with any point broadcasted by a receiver that is correctly signed by the dealer at steps $3a$ or $3c$, or other projection polynomial broadcasted by the resolver at this step.

**(Protocol 16) Reconstruction phase:**

- If party $P_i$ is a reconstructor and the sharing phase did not fail, do the following. Consider the triply-signed points $\{f_j(k), \sigma^D_{j,k}, \sigma^k_j, \sigma^j_k\}_k$ by the $j$-th receiver. If the $j$-th receiver did not broadcast `ComplainPoly`, broadcast all the received the triply-signed points.
- **Secret Reconstruction:** The secret $s$ can be reconstructed by any party that takes any valid projections for $t+1$ receivers. A projection for the $j$-th receiver is considered valid if 1) it was broadcasted by the resolver as a result of a `ComplainPoly` message, or 2) a reconstructor broadcasted triply-signed points $\{f_j(k), \sigma^D_{j,k}, \sigma^k_j, \sigma^j_k\}_k$by the $j$-th receiver, form a degree-$t$ polynomial and there are at least $2t+1$ points which are all either correctly triply signed, or were contained in one of the `ComplainPoly` messages.

Moreover, for at least $2t+1$ points, the receiver sent this point triply signed by the dealer, the $j$-th receiver and his own signature to all the reconstructors at step $3b$.

Now we show that the reconstructed secret is $s$. In the reconstruction phase, the $t+1$ reconstructors broadcast all the received points and signatures that have been received, for each receiver. Since there is at least one honest reconstructor, this reconstructor will broadcast points corresponding to the $t+1$ consistent valid projections (with the polynomial $F$) from honest receivers, uniquely determine the secret $s$. Moreover, any projection broadcasted by the resolver as a result of a `ComplainPoly` also lies on $F$. Finally, note that the adversary cannot contribute any projection that is not consistent with $F$ because the projection must be signed by each of the honest receivers. □ □

**Lemma 27** (Privacy)**.** *If the dealer and the resolver are honest, no information about the secret $s$ is revealed before the reconstruction phase is started in Protocol 16.*

*Proof.* The view of the adversary contains only the projections of $F$ with indices that belong to corrupted receivers. Since there are at most $t$ corrupted receivers, this information is statistically independent of the secret $s$.

Note that any broadcasted message of the form `ComplainPoly` comes from a corrupted receiver and therefore the resolver broadcasts a projection that was already known to the adversary. Similarly, any broadcasted message `Complain` containing a point $F(i,j)$ at step $3a$ or $3c$,[2] is only broadcasted when either the $i$-th or the $j$-th receiver is corrupted (and

[2]Note that step $3b$ does not reveal any information, since we consider the execution-leaks model and messages in this step are sent to the reconstructors, which are executed only in the reconstruction phase.

therefore the point was already known by the adversary). □ □

**Lemma 28** (Verifiability)**.** *At the end of the sharing phase in Protocol 16, if it succeeded, there is only one secret $s'$ that will be reconstructed upon executing the reconstruction phase.*

*Proof.* Let $i, j \in [1, 3t+1]$ with $i < j$. We first show that if the sharing phase succeeds, the projections corresponding to every pair of $i$-th and $j$-th honest receivers (taken into account in the Secret Reconstruction phase) are consistent. We divide four cases.

1. Both projections are broadcasted by the resolver as a result of corresponding `ComplainPoly` messages. In this case, since the sharing succeeds, the protocol prescribes that both polynomials need to be consistent.
2. No party broadcasted a `ComplainPoly` message and therefore no projection is broadcasted by the resolver. In this case, both honest receivers received correctly signed degree-$t$ projections from the dealer. Moreover, the $i$-th receiver sent the point $f_i(j)$ signed by the dealer and his own signature to the $j$-th receiver. Since the sharing phase succeeds, we have that both projections are consistent, i.e. $f_i(j) = f_j(i)$, and therefore the point $f_j(i)$ and the signatures from the dealer, the $i$-th receiver and the $j$-th receiver are sent to every reconstructor at step $3b$. Moreover, at least one (honest) reconstructor will broadcast this information.
3. Only the $i$-th receiver broadcasts a `ComplainPoly`, and as a result the resolver broadcasts the projection $f_i$. In this case, the $i$-th receiver did not send any message to the $j$-th receiver at step $2d$, and therefore the $j$-th receiver broadcasts a message `Complain` with his own point $f_j(i)$. Since the sharing phase succeeds, it must be case that $f_i(j) = f_j(i)$. (Note that $2t+1$ triply signed points of $f_j$ are sent to each of the reconstructors.)
4. Only the $j$-th receiver broadcasts a `ComplainPoly`, and as a result the resolver broadcasts the projection $f_j$. In this case, the $i$-th receiver sent the point $f_i(j)$ signed by the dealer and his own signature to the $j$-th receiver. The $j$-th receiver receives this point with correct signatures and broadcasts a message `Complain` with the point $f_i(j)$ signed by the dealer and the $i$-th receiver. Since the sharing phase succeeds, this point is consistent with the broadcasted polynomial $f_j$. (Note that $2t+1$ triply signed points of $f_i$ are sent to each of the reconstructors.)

The projections corresponding to honest receivers are sufficient to uniquely define a bivariate polynomial $F'$, which in turn defines a value $s'$.

Furthermore, note that any valid projection for a corrupted receiver is also consistent with $F'$. Either the projection was broadcasted as a result of `ComplainPoly` (in which case it is consistent with the honest receiver's projections), or it is signed by at least $2t+1$

receivers ($t+1$ are honest, and therefore these uniquely define a consistent projection with $F'$). □ □

#### 4.6.1.1 SD-VSS in the sending-leaks model.

In the sending-leaks model, the protocol is almost the same, except that we require a simple adaptation, i.e., we now require $2t+1$ reconstructors. Each receiver $P_i$ now shares its state to the reconstructors, while signing each share. The reconstructors output all shares along with the signature of the corresponding $P_i$. The correctly signed shares are used to reconstruct the state of $P_i$, which in turn is used to compute the output as in the execution-leaks case. The full protocol description is given in Protocol 17. Same as in the execution-leaks case, the communication complexity is $O(n^3)$ (excluding any polynomial factors in the security parameter).

The resulting protocol can be proven to achieve the same lemma statements.

The following lemmas formally argue that our VSS protocol in the sending-leaks model satisfies *correctness*, *privacy*, and *verifiability*.

**Lemma 29** (Correctness)**.** *If the dealer and the resolver are honest, the sharing phase in Protocol 17 does not fail and the reconstructed secret after executing both the sharing and consequently the reconstruction phase is $s$.*

*Proof.* First, we show that the sharing phase does not fail. This is because all points signed by the dealer lie on the bivariate polynomial $F$, and any polynomial broadcasted by the dealer or point signed by the dealer is consistent with $F$. That is, any message `Complain` with a point and a dealer's signature that is broadcasted by a receiver lies on $F$, and any polynomial broadcasted by the resolver as a response to a message `ComplainPoly` lies on $F$ as well.

Moreover, consider each $i$-th honest receiver, for $i \in [1, 3t+1]$. The receiver correctly received a signed degree-$t$ projection, so he did not broadcast any `ComplainPoly` message. Moreover, the receiver sends a signed share of at least $2t+1$ points, each point triply signed by the dealer, the $j$-th receiver and himself, to each reconstructors at step $3b$.

Now we show that the reconstructed secret is $s$. In the reconstruction phase, the $t+1$ reconstructors broadcast all the correctly signed shares they received from each receiver. For each receiver, only the messages signed by this receiver are used to reconstruct the values it submitted. As there are at least $t+1$ honest reconstructors, each share of each honest receiver will be reconstructed correctly. The shares of $t+1$ honest receivers (where

some of them might have broadcast the `ComplainPoly`) uniquely determine the secret $s$. Finally, note that the adversary cannot contribute any projection that is not consistent with $F$ because the projection must be signed by the dealer. □ □

**Lemma 30** (Privacy)**.** *If the dealer and the resolver are honest, no information about the secret $s$ is revealed before the reconstruction phase is started in Protocol 17.*

*Proof.* The view of the adversary contains only the projections of $F$ with indices that belong to corrupted receivers. Since there are at most $t$ corrupted receivers, this information is statistically independent of the secret $s$.

Note that any broadcasted message of the form `ComplainPoly` comes from a corrupted receiver and therefore the resolver broadcasts a projection that was already known to the adversary. Similarly, any broadcasted message `Complain` containing a point $F(i, j)$ at step $3a$ or $3c$, is only broadcasted when either the $i$-th or the $j$-th receiver is corrupted (and therefore the point was already known by the adversary). Finally, step $3b$ does not reveal any information, as we are using a $(t + 1, 2t + 1)$ secret sharing scheme, and $t + 1$ reconstructors are honest. □ □

**Lemma 31** (Verifiability)**.** *At the end of the sharing phase in Protocol 17, if it succeeded, there is only one secret $s'$ that will be reconstructed upon executing the reconstruction phase.*

*Proof.* Let $i < j$, $i, j \in [1, 3t + 1]$. We first show that if the sharing phase succeeds, the projections corresponding to every pair of $i$-th and $j$-th honest receivers (taken into account in the Secret Reconstruction phase) are consistent. We divide four cases.

1. Both projections are broadcasted by the resolver as a result of corresponding `ComplainPoly` messages. In this case, since the sharing succeeds, the protocol prescribes that both polynomials need to be consistent.
2. No party broadcasted a `ComplainPoly` message and therefore no projection is broadcasted by the resolver. In this case, both honest receivers received correctly signed degree-$t$ projections from the dealer. Moreover, the $i$-th receiver sent the point $f_i(j)$ signed by the dealer and his own signature to the $j$-th receiver. Since the sharing phase succeeds, we have that both projections are consistent, i.e. $f_i(j) = f_j(i)$, and therefore the point $f_j(i)$ and the signatures from the dealer, the $i$-th receiver and the $j$-th receiver are secret-shared to the reconstructors at step $3b$, and each such share is signed. The $t + 1$ honest reconstructors are sufficient to recover this information, while no adversarial reconstructor can suggest a wrong share for $P_j$, as the share must be signed.

3. Only the $i$-th receiver broadcasts a `ComplainPoly`, and as a result the resolver broadcasts the projection $f_i$. In this case, the $i$-th receiver did not send any message to the $j$-th receiver at step $2d$, and therefore the $j$-th receiver broadcasts a message `Complain` with his own point $f_j(i)$. Since the sharing phase succeeds, it must be case that $f_i(j) = f_j(i)$. (Note that $2t + 1$ triply signed points of $f_j$ are secret-shared to the reconstructors.)
4. Only the $j$-th receiver broadcasts a `ComplainPoly`, and as a result the resolver broadcasts the projection $f_j$. In this case, the $i$-th receiver sent the point $f_i(j)$ signed by the dealer and his own signature to the $j$-th receiver. The $j$-th receiver receives this point with correct signatures and broadcasts a message `Complain` with the point $f_i(j)$ signed by the dealer and the $i$-th receiver. Since the sharing phase succeeds, this point is consistent with the broadcasted polynomial $f_j$. (Note that $2t + 1$ triply signed points of $f_i$ are secret-shared to the reconstructors.)

The projections corresponding to honest receivers are sufficient to uniquely define a bivariate polynomial $F'$, which in turn defines a value $s'$.

Furthermore, note that any valid projection for a corrupted receiver is also consistent with $F'$. Either the projection was broadcasted as a result of `ComplainPoly` (in which case it is consistent with the honest receiver's projections), or it is signed by at least $2t + 1$ receivers ($t + 1$ are honest, and therefore these uniquely define a consistent projection with $F'$). □ □

**Protocol 17** Split-Dealer Verifiable Secret Sharing, Sending-Leaks.

Each $P_i$, for $i \in [1, 5t+4]$ does the following: **Sharing phase:**

- If $P_i$ is the dealer (i.e., $i = 1$):
  1. Sample a symmetric polynomial $F(x, y)$ such that $F(0, 0) = s$.
  2. Sample $\left(\mathsf{vk}^D, \mathsf{sk}^D\right) \leftarrow \mathsf{DS.KGen}(1^\lambda)$. Publicly broadcast the public key $\mathsf{vk}^D$. Also privately send the polynomial $F$ to the resolver (party $P_{3t+3}$).
  3. Let $f_j := F(j, y)$, $j \in [3t+1]$, be the $j$-th vertical projection of $F$. Privately send to the $j$-th receiver all points of $f_j$ signed, i.e. send to party $P_{j+1}$ the pairs $\{f_j(y), \sigma^D_{j,y}\}_{y \in [3t+1]}$, where $\sigma^D_{j,y} = \mathsf{DS.Sign}_{\mathsf{sk}^D}(f_j(y))$.
- If $P_i$ is the $j$-th receiver (i.e. $i = j+1$ and $j \in [3t+1]$):
  1. Sample $\left(\mathsf{vk}^j, \mathsf{sk}^j\right) \leftarrow \mathsf{DS.KGen}(1^\lambda)$ and publicly broadcast the public key $\mathsf{vk}^j$.
  2. For the messages received from the dealer (party $P_1$), $\{f_j(y), \sigma^D_{j,y}\}_{y \in [3t+1]}$, do the following:
     (a) Check that the signatures are correct according to the broadcasted public key $\mathsf{vk}^D$, i.e. $\mathsf{DS.Vf}_{\mathsf{vk}^D}(f_j(y), \sigma^D_{j,y}) = 1$ for each point.
     (b) Check that the points $\{f_j(y)\}_{y \in [3t+1]}$ lie on a degree-$t$ polynomial.
     (c) If any check fails, publicly broadcast a message `ComplainPoly`.
     (d) Otherwise, for each receiver $j < k < 3t+2$, send $(f_j(k), \sigma^D_{j,k}, \sigma^j_k)$ to the $k$-th receiver, where $\sigma^j_k = \mathsf{DS.Sign}_{\mathsf{sk}^j}(f_j(k))$.
  3. For the message $(f_k(j), \sigma^D_{k,j}, \sigma^k_j)$ received from the $k$-th receiver (party $P_{k+1}$) with $k < j$: check that the signatures are valid, i.e. $\mathsf{DS.Vf}_{\mathsf{sk}^D}(f_k(j)) = \mathsf{DS.Vf}_{\mathsf{sk}^k}(f_k(j)) = 1$.
     (a) If the received signatures are incorrect (or no message was received) and no message `ComplainPoly` is being broadcasted at step $2b$, broadcast a complaint message $(\texttt{Complain}, k, f_j(k), \sigma^D_{j,k})$.
     (b) If the received signatures are correct, no `ComplainPoly` is being broadcasted at step $2b$, do the following. If the points are not consistent $f_k(j) \neq f_j(k)$ broadcast a complaint with both points signed by the dealer: $(\texttt{Complain}, k, f_k(j), \sigma^D_{k,j}, f_j(k), \sigma^D_{j,k})$ and the sharing phase is failed. Otherwise, if the points are consistent, let $s_{j,k}$ denote the point with the signatures from the dealer, the $k$-th receiver and the $j$-th receiver: $(f_j(k) = s_{j,k}, \sigma^D_{j,k}, \sigma^k_j, \sigma^j_k)$. Share $s_{j,k}$ using $(t+1, 2t+1)$ Shamir's secret sharing, let $s_{j,k}(m)$ denote the $m$-th share. Send the share $s_{j,k}(m)$, along with its own signature on it $\mathsf{DS.Sign}_{\mathsf{sk}^j}(s_{j,k}(m))$ to the reconstructor $P_{3t+3+m}$, for $m \in [2t+1]$.
     (c) If the received signatures are correct but `ComplainPoly` is being broadcasted at step $2b$, broadcast a complaint $(\texttt{Complain}, k, f_k(j), \sigma^D_{k,j}, \sigma^k_j)$.
- If $P_i$ is the resolver: For each message `ComplainPoly` from the $j$-th receiver, publicly broadcast the projection polynomial $f_j$.

The sharing is considered to have failed if the polynomial output by the resolver does not have degree-$t$ or it is inconsistent with any point broadcasted by a receiver that is correctly signed by the dealer at steps $3a$ or $3c$, or other projection polynomial broadcasted by the resolver at this step.

**(Protocol 17) Reconstruction phase:**

- If party $P_i$ is a reconstructor and the sharing phase did not fail, do the following. Consider the shares $s_{j,k}(i-(3t+3))$ and signatures on these shares $\mathsf{DS.Sign}_{\mathsf{sk}^j}(s_{j,k}(i-(3t+3)))$ by the $j$-th receiver. If the $j$-th receiver did not broadcast `ComplainPoly`, broadcast these shares and signatures.
- **Secret Reconstruction:** The secret $s$ can be reconstructed by any party $C$. First, for each receiver $j$ who did not broadcast `ComplainPoly`, each point $k \in [3t+1]$, $C$ reconstructs the value $s_{j,k}$ using any $t+1$ shares of this value with correct signatures from $P_{j+1}$. Then, $C$ reconstructs the secret $s$ using valid projections for $t+1$ receivers. A projection for the $j$-th receiver is considered valid if 1) it was broadcasted by the resolver as a result of a `ComplainPoly` message, or 2) a reconstructor broadcasted triply-signed points $\{f_j(k), \sigma^D_{j,k}, \sigma^k_j, \sigma^j_k\}_k$by the $j$-th receiver, form a degree-$t$ polynomial and there are at least $2t+1$ points which are all either correctly triply signed, or were contained in one of the `ComplainPoly` messages.

### 4.6.2 YOSO$^{\mathsf{WCC}}$ SD-VSS-based Randomness Generation with $n = 5t+3$

We will now show how to build a randomness generation protocol in the YOSO$^{\mathsf{WCC}}$ model with our SD-VSS as the building block. The basic idea is to execute $t+1$ independent instances of our SD-VSS protocol, and the final random coin is set as the XOR of all the reconstructed secret coins of each SD-VSS instance that did not fail. We have $t+1$ instances to ensure that there is at least one honest dealer, and thus we get a random string as output in the end.

We employ *role-stacking* to amortize the number of roles needed for our randomness generation scheme. Instead of naively repeating the SD-VSS protocol sequentially one after the other, we pipeline the protocol execution in a careful way. This means that some parties may have to perform the role of a dealer in some $i$-th SD-VSS instance, a receiver in some $k$-th SD-VSS instance, and so on. We give our randomness generation protocol in the

execution-leaks model in Protocol 18. The communication complexity of this construction is $O(n^4)$ (excluding any polynomial factors in the security parameter).

**Protocol 18** Randomness Generation from YOSO$^{\mathsf{WCC}}$ SD-VSS in the Execution-Leaks Model.

We have the $t+1$ parties (or roles) as dealers, parties $P_i$ for $i \in [2, 4t+2]$ are referred to as *receivers*, parties $P_i$ for $i \in [3t+3, 4t+3]$ are referred to as *resolvers* and the final set of parties $P_i$ for $i \in [4t+4, 5t+4]$ are called *reconstructors*.

For $i \in [t+1]$, in the $i$-th instance of the execution-leaks SD-VSS:

- For $k = i$, party $P_k$ plays the role of the dealer.
- For $k \in [i+1, 3t+i+1]$, party $P_k$ plays the role of the $(k-i)$-th receiver.
- For $k = 3t+2+i$, party $P_k$ plays the role of the resolver.
- For $k \in [4t+4, 5t+4]$, party $P_k$ plays the role of a reconstructor.

**Protocol 19** Randomness Generation from YOSO$^{\mathsf{WCC}}$ SD-VSS in the Sending-Leaks Model.

We have the $t+1$ parties (or roles) as dealers, parties $P_i$ for $i \in [2, 4t+2]$ are referred to as *receivers*, parties $P_i$ for $i \in [3t+3, 4t+3]$ are referred to as *resolvers* and the final set of parties $P_i$ for $i \in [4t+4, 6t+4]$ are called *reconstructors*.

For $i \in [t+1]$, in the $i$-th instance of the sending-leaks VSS:

- For $k = i$, party $P_k$ plays the role of the dealer.
- For $k \in [i+1, 3t+i+1]$, party $P_k$ plays the role of the $(k-i)$-th receiver.
- For $k = 3t+2+i$, party $P_k$ plays the role of the resolver.
- For $k \in [4t+4, 6t+4]$, party $P_k$ plays the role of a reconstructor.

The protocol for the sending-leaks is the same as the execution-leaks protocol except that we have parties $P_i$ for $i \in [4t+4, 6t+4]$ acting as reconstructors and we make use of the sending-leaks SD-VSS as our building block. For completeness, we describe our sending-leaks protocol in Protocol 19, which is deferred to the appendix due to space constraints. The communication complexity of the sending-leaks construction is also $O(n^4)$ (excluding any polynomial factors in the security parameter).

**Further Role Reduction.** We can reduce the number of roles even further from $5t+4$ to $5t+3$ in the case of execution-leaks and from $6t+4$ to $6t+3$ in the case of sending-leaks. To do this, we make a simple observation that the $i$-th dealer can be their own recipient in the $i$-th SD-VSS instance. This is further optimising the role-stacking mechanism above and the receivers are now parties $P_k$ for $k \in [4t+1]$ instead of $4t+2$. Thus we are able to shave-off one more role from our randomness generation protocols. For ease of understanding, we present our formal protocols without including this optimization.

**Theorem 9.** *Assuming the existence of digital signatures, there exists a* $(t, n = 5t + 3)$*-computationally secure YOSO*$^{\mathsf{WCC}}$ *randomness generation protocol in the execution-leaks model.*

*Proof.* The role-stacking mechanism executes $t + 1$ instances of verifiable secret sharing. In order to see that the output coin is uniform distributed, we make following observations.

First, there is at least one instance of verifiable secret sharing that has the respective dealer and resolver parties honest. This is simply because there are $t + 1$ SD-VSS instances, with at most $t$ parties being corrupted, and each party in the protocol can execute at most one role as a dealer or as a resolver (but not both).

Second, by privacy of the SD-VSS that has an honest dealer and resolver (see Lemma 27), no information about his shared value is revealed before any of the reconstructor parties are executed. Moreover, by correctness of SD-VSS (see Lemma 26), the sharing phase of this SD-VSS instance is successful (and will be publicly reconstructed by the reconstructor parties).

Third, by the verifiability property of SD-VSS (see Lemma 28), for each other instance, either the sharing phase failed, or there is a fixed value that will be reconstructed by the reconstructor parties. Importantly, this value is fixed at the end of the sharing phase, independently of any value that has been distributed in an instance of SD-VSS that has an honest dealer and resolver (since in such an instance the adversary obtains no information, in a statistical sense, about the shared value).

Given that the output-coin is computed as the sum of the values that are publicly reconstructed, and one of the values is distributed in an instance of SD-VSS with honest dealer and resolver, the output is uniformly random. □

**Theorem 10.** *Assuming the existence of digital signatures, there exists an efficient* $(t, n = 6t + 3)$*-computationally secure YOSO*$^{\mathsf{WCC}}$ *randomness generation protocol in the sending-leaks model.*

*Proof.* The proof for correctness and verifiability is similar to the case we saw for execution-leaks model. The only change is in arguing privacy. Notice that no $t$ reconstructors can recover the secret of a honest dealer as the information is secret shared among the parties of $\mathcal{P}'$ using a $(t + 1)$-out of-$(2t + 1)$ secret sharing. Therefore, in the worst-case, the adversary only has access to $t$ shares and therefore a honest secret is information-theoretically hidden from its view. □

## 4.7 Protocols from Non-Interactive Commitments

Our construction consists of a sequence of $t+1$ instantiations of a split-dealer VSS protocol, which is a modification of the protocol presented in [88]. To recall, each instance contains the following roles:

1. Party $D$, who acts as the dealer distributing the secrets (publishing commitments to the points of a bivariate degree-$t$ polynomial and bilaterally sending to each receiver the openings to the horizontal and vertical projections), and sends its state to its counterpart $D'$.
2. $2t+1$ receivers $R_i$, who receive and verify the projections (they are of degree-$t$ and consistent with the commitments); complain about the received values if applicable, and otherwise send these to each party $R'_i$.
3. Party $D'$ who obtains a state from $D$ and uses it to reveal the projections of the receivers that complained. If $D'$ cannot resolve a complaint, this instance is aborted ($D'$ is deemed corrupt).
4. $t+1$ receivers $R'_i$ who receive all the shares from each party $R_i$, as well as set their shares to the ones broadcast by $D'$ (if the counterpart $R_i$ complained), and publicly reveal all these shares.

To compose the instances, we organize the roles are as follows:

- For $1 \leq i \leq t+1$, party $P_i$ executes the role of the dealer $D$ in $i$-th instance. If additionally $i > 1$, $P_i$ also executes the role $R_{i-j}$ in $j$-th instance, where $j < i$.
- For $t+2 \leq i \leq 3t+2$, party $P_i$ executes the role $R_{i-j}$ in $j$-th instance, where $j < i$. If additionally $i > 2t+2$, $P_i$ also executes the role of the dealer $D'$ in the $i-2t-2$-th instance.
- For $i = 3t+3$, party $P_i$ executes the role $D'$ in the $(t+1)$-st instance.
- For $3t+4 \leq i \leq 4t+4$, $P_i$ executes the role $R'_{i-3t-3}$ for each instance.

We formally describe the protocol below.

**Theorem 11.** *Assuming non-interactive perfectly binding commitments, there is an efficient $(t, n = 4t+4)$-computationally secure YOSO$^{\mathsf{WCC}}$ randomness generation protocol in the execution-leaks model.*

*Proof.* Note that since the adversary corrupts up to $t$ parties and there are $t+1$ instances, there exists an instance where both $D_i$ and $D'_i$ are honest.

First, observe that for this particular instance, we prove that before role $3t+4$, the

**Protocol 20** Ex. Leaks Coin Tossing from any Non-Interactive Commitment

**Sharing phase:**

Each $D_i$, $i \in [t+1]$ does the following:

1. $D_i$ chooses a random bivariate degree-$t$ (on both variables) polynomial $F^i(x, y)$.
2. $D_i$ commits to each point of $F^i(x, y)$ using the non-interactive commitment scheme, leading to commitments $\mathsf{com}_{x,y} \leftarrow \mathsf{Commit}(F^i(x, y); r_{x,y})$ for random coins $r_{x,y}$, and publicly broadcasts the commitments $\mathsf{com}_{x,y}$ for each $x, y \in [1, 2t+1]$.
3. $D_i$ sends the opening information of the $x$-th horizontal and vertical projections points: $\{(F^i(x, y), r_{x,y})\}_{y\in[2t+1]}$ and $\{(F(y, x), r_{y,x})\}_{y\in[2t+1]}$ to each $P_x$, $x \in [2t+1]$; and sends all opening information to $D'_i$.

Each $R_i$, $i \in [2t+1]$ does the following:

1. For each dealer $D_j$, $R_i$ checks whether the received openings against the published commitments, and also that the received projections are of degree-$t$.
2. $R_i$ sends its private state to every $R'_j$.

Each $D'_i$, $i \in [t+1]$ does the following:

1. $D'_i$ broadcasts the openings of all points corresponding to parties who complained about $D_i$. If any broadcasted opening by $D'_i$ is not consistent with the corresponding commitment, or any openings corresponding to a projection do not form a degree-$t$ polynomial, $D'_i$ is deemed corrupt.

**Reconstruction phase:**

Each $R'_i$, $i \in [t+1]$ does the following:

1. If $R_i$ complained about $D_j$, and $D'_j$ was not deemed corrupt, $R'_i$ sets the points of the $i$-th projections to the values broadcasted by $D'_j$.
2. $R'_i$ outputs all points obtained for non-corrupt dealers.

Client $C$ does the following to compute the coin:

1. For each $D'_i$ who was not deemed corrupt, $C$ uses any $t+1$ projections that pass the verification check against the corresponding published commitments to reconstruct a bivariate polynomial $F^i$. Let the value $s_i = F^i(0, 0)$.
2. Let $H$ denote the index set of dealers $D'_i$ which were not deemed corrupt. $C$ outputs $\bigoplus_{i\in H} s_i$.

corresponding bivariate polynomial $F^i$ is unknown. This is because at most $t$ roles $R_j$ belong to corrupted parties, and therefore only up to $t$ projections of the bivariate polynomial $F^i$ are initially known to the adversary. Further note that honest receivers $R_j$ do not complain about $D_i$, since an honest $D_i$ always commits to a degree-$t$ bivariate polynomial $F^i$, and the points that are sent private are consistent with the commitments. Therefore, any point that is publicly opened by an honest $D'_i$ belongs to a projection that is known to the adversary. Further note that at the end of the protocol, the client will reconstruct $F^i(0,0)$. This is because there is at least one honest recipient $R'_j$ that holds openings corresponding to $t+1$ projections and that are consistent with the published commitments.

Second, observe that at the start of role $3t+4$, any instance $j$ where either $D_j$ or $D'_j$ were corrupted, and where the instance did not abort ($D'_j$ was not publicly deemed corrupt), has a fixed bivariate polynomial $F^j$ and the client will reconstruct $F^j(0,0)$. This is because the honest receivers hold consistent projections of degree-$t$ with the published commitments. Moreover, by the first point, the value $F^j(0,0)$ was fixed independently of the honest instance $F^i$, due to the hiding property of the commitments.

From the two points above, since the final coin is computed as the sum of instances that were successful (where the dealer was not deemed corrupt), and in at least one instance the random value was chosen by an honest dealer, the coin has negligible bias. □ □

#### 4.7.0.1 Sending-Leaks Variant.

For the sending-leaks model, we similarly implement the behavior of each dealer using two roles – one responsible for the sharing of a secret, and one responsible for addressing the complaints. However, we not only have $2t+1$ parties $R_i$, but also $2t+1$ parties $R'_j$. Each $R_i$ follows the procedure of round two, and if its shares verify, but it additionally sends its shares to *only* $R'_i$. Finally, each $R'_i$ publishes the shares (it got from $R_i$) which verified correctly. Note that there are $t+1$ pairs $(R_i, R'_i)$ that are both honest, which will publish projections that are consistent with the published commitments and therefore will be enough to reconstruct the bivariate polynomials.

**Theorem 12.** *Assuming non-interactive perfectly binding commitments, there is an efficient* $(t, n = 5t+4)$*-computationally secure YOSO*$^{\mathsf{WCC}}$ *randomness generation protocol in the sending-leaks model.*

## 4.8 Improved Protocols for Small Number of Parties from Non-Interactive Commitments

We now present our efficient randomness generation protocol for $n = 3t + 1$ parties using non-interactive commitments in the execution-leaks model. We make use of a combinatorial object that we formally define below.

### 4.8.1 $t$-Sharing Matrices

**Definition 23** ($t$-sharing matrix)**.** *We say that a matrix $M \in \{0,1\}^{m \times \ell}$ is a* $t$-sharing matrix *if the following two properties hold:*

- *Every row of $M$ has Hamming weight $t$;*
- *For any set $S \subseteq [\ell]$ of size $t-1$ there exists $i \in [m]$ such that $M_{ij} = 0$ for all $j \in S$.*

We note that $t$-sharing matrices are related to (but weaker than) constant-weight $t$-disjunct matrices.

Our next lemma exhibits a lower bound on the number of rows of any $t$-sharing matrix $M$.

**Lemma 32.** *If $M \in \{0,1\}^{m \times \ell}$ is a $t$-sharing matrix, then we must have $m \geq \frac{\binom{\ell}{t-1}}{\binom{\ell-t}{t-1}}$.*

*Proof.* We say that a vector $v \in \{0,1\}^{\ell}$ *evades* a set $S \subseteq [\ell]$ if $v_S = 0$. Every such $v$ of weight $t$ evades exactly $\binom{\ell-t}{t-1}$ sets $S \subseteq [\ell]$ of size $t-1$. This means that at most $m \cdot \binom{\ell-t}{t-1}$ such sets are evaded by at least one of the $m$ rows of $M$. On the other hand, for $M$ to be $t$-sharing it must be the case that every set $S \subseteq [\ell]$ of size $t-1$ (of which there $\binom{\ell}{t-1}$ choices) is evaded by some row of $M$, and so we must have

$$m \cdot \binom{\ell - t}{t-1} \geq \binom{\ell}{t-1}.$$

This yields the desired lower bound on $m$. □ □

We use the following simple construction of a $t$-sharing matrix in our general protocol for $n = 3t + 1$ parties.

**Lemma 33.** *There exists a $t$-sharing matrix $M \in \{0,1\}^{m \times \ell}$ with $\ell = 2t - 1$ columns and $m = \binom{2t-1}{t}$ rows. Moreover, this matrix can be constructed in time polynomial in $\ell$ and $m$.*

*Proof.* Consider the matrix $M \in \{0,1\}^{\binom{2t-1}{t} \times (2t-1)}$ where the rows of $M$ correspond to all $(2t-1)$-bit vectors of weight exactly $t$. It suffices to check that for all subsets of $t-1$ columns of $M$ there exists an index $i$ on which they are all 0.

Fix any subset $S \subseteq [2t-1]$ of size $t-1$. Since $[\ell] \setminus S$ has size $t$, there is a row of $M$ whose support lies outside $S$. Therefore, the columns of $M$ indexed by $S$ are all 0 on this row. □ □

By lemma 32, we get that the number of rows in the construction of lemma 33 cannot be improved if the number of rows is kept as is.

### 4.8.2 Our Protocol

With the aid of a $t$-sharing matrix $M$ as defined above, we present our randomness generation protocol in the YOSO$^{\mathsf{WCC}}$ execution-leaks model, as detailed in Protocol 21.

**Theorem 13.** *If* $\mathsf{COM}$ *is a perfectly binding and computationally hiding non-interactive commitment scheme, then Protocol 21 is a* $(t, n = 3t+1)$*-computationally secure YOSO*$^{\mathsf{WCC}}$ *randomness generation protocol in the execution-leaks model. Furthermore, its computational and communication complexities are polynomial in* $t$ *and the security parameter* $\lambda$*.*

*Proof.* Our proof proceeds on a case-by-case basis. This way we can capture all the adversarial strategies in an easy-to-understand way. In all the cases, the adversary corrupts up to $t$ parties in $\{P_1, \ldots, P_{3t+1}\}$.

We state the following lemma that will be useful in our analysis.

**Lemma 34.** *If the adversary corrupts some party* $P_{i^*}$ *where* $i^* \in [2t-1]$*, such that for some* $j \in [m]$*,* $M_{jk} = 0$ *for* $k \in [i^*-1]$ *and* $M_{ji^*} = 1$*, then either the adversary's commitment* $\mathsf{com}_j$ *receives a* $(\texttt{Complain}, j)$ *or a valid opening is broadcast by some party* $P_h$ *for* $h \in [2t+1, 3t+1]$*.*

*lemma 34.* Consider index $i^* \in [2t-1]$, such that party $P_{i^*}$ is corrupt and any index $j \in [m]$ such that $M_{jk} = 0$ for all $k \in [i^*-1]$ and $M_{ji^*} = 1$. The adversary broadcasts commitment $\mathsf{com}_j$, and the openings to any party it wishes to. However, note that there exists an honest party $P_k$ for $k \in [i^*, 2t-1]$ and $M_{jk} = 1$, which if it did not receive the valid opening, it would broadcast a message $(\texttt{Complain}, j)$. On the other hand, if the adversary sends valid opening of the commitment $\mathsf{com}_j$ to party $P_k$, then party $P_k$ would send the opening to all recipients $\{P_{2t+1}, \ldots, P_{3t+1}\}$. Note that there is at least 1 honest

**Protocol 21** Randomness Generation using $n = 3t + 1$ roles in the Execution-Leaks Model.

We have a $t$-sharing matrix $M \in \{0,1\}^{m \times \ell}$ according to lemma 33, where $\ell = 2t - 1$ and $m = \binom{2t-1}{t}$.

1. For $i \in [2t-1]$:
   (a) For $j \in [m]$, party $P_i$ does the following:
      i. If $\forall k \in [i-1], M_{jk} = 0$ and $M_{ji} = 1$,
         A. Choose value $s_j \leftarrow\!\!\$\ \{0,1\}^{\ell_m(\lambda)}$ and generate $\mathsf{com}_j \leftarrow \mathsf{Commit}(s_j; r_j)$ for random coins $r_j \in \{0,1\}^{\ell_r(\lambda)}$.
         B. Broadcast $\mathsf{com}_j$ and send the opening $(s_j, r_j)$ to all parties $P_k$ where $k \in [i+1, \ell]$ and $M_{jk} = 1$, and to all parties $P_k$ where $k \in [2t+1, 3t+1]$.
      ii. Else if $M_{ji} = 1$,
         A. Receive $(s_j, r_j)$ from party $P_k$ for some $k \in [i-1]$ and $M_{jk} = 1$.
         B. Broadcast $(\texttt{Complain}, j)$ if nothing was received or if $\mathsf{Commit}(s_j; r_j) \neq \mathsf{com}_j$. Else, send $(s_j, r_j)$ to all parties $P_k$ where $k \in [2t+1, 3t+1]$.
2. Party $P_{2t}$ samples $s^* \leftarrow\!\!\$\ \{0,1\}^{\ell_m(\lambda)}$ and broadcasts $s^*$.
3. For $i \in [2t+1, 3t+1]$, party $P_i$ does the following:
   (a) For any $j \in [m]$, if no message $(\texttt{Complain}, j)$ was seen, then receive all messages $(s_j, r_j)$ such that $\mathsf{Commit}(s_j; r_j) = \mathsf{com}_j$, and output $(s_j, r_j)$.

Let $C \subseteq [m]$ be the set of indices such that for any $j \in C$, no message $(\texttt{Complain}, j)$ was seen and $(s_j, r_j)$ was broadcast such that $\mathsf{Commit}(s_j; r_j) = \mathsf{com}_j$. Let the final randomness be set as $s = \bigoplus_{j \in C} s_j \oplus s^*$.

party $P_h \in \{P_{2t+1}, \ldots, P_{3t+1}\}$ that receives the opening of $\mathsf{com}_j$ from $P_k$ and outputs that during its execution. This ensures that the opening of the adversarial commitment will be broadcast by an honest party $P_h$. □ □

*Case 1 - Adversary corrupts t roles in* $\{P_1, \ldots, P_{2t-1}\}$: In this case, by the definition of $t$-sharing matrix $M$, the adversary can learn the openings of all the honest parties' commitments. Moreover, the commitments generated by the adversary, the openings, and the recipients in $\{P_{2t+1}, \ldots, P_{3t+1}\}$ that it chooses to reveal the openings to, are all completed before the execution of the honest $P_{2t}$. Therefore, we can say that the adversary's choices are made independent of the random string that honest party $P_{2t}$ will output. Therefore, the final values output by $\{P_{2t+1}, \ldots, P_{3t+1}\}$ irrespective of the adversarial choices, will result in a random string that is uniformly distributed in the coin space.

*Case 2 - Adversary corrupts t roles except party* $P_{2t}$: Let the adversary corrupt $t_1 \leq (t-1)$ parties in $\{P_1, \ldots, P_{2t-1}\}$ and $t - t_1$ among $\{P_{2t+1}, \ldots, P_{3t+1}\}$. Consider index $i^* \in [2t-1]$, such that party $P_{i^*}$ is corrupt and any index $j \in [m]$ such that $M_{jk} = 0$ for all $k \in [i^* - 1]$ and $M_{ji^*} = 1$. The adversary broadcasts commitment $\mathsf{com}_j$. We know by lemma 34, that the commitment either receives a message $(\texttt{Complain}, j)$ from an honest party $P_k$ for $k \in [i^*, 2t-1]$, and the commitment and its opening will be ignored during the computation of the final random coin of the protocol. This is even before party $P_{2t}$ outputs its random value, similar to the previous case. Alternatively, some honest party $P_{k'} \in \{P_{2t+1}, \ldots, P_{3t+1}\}$ outputs the correct opening of $\mathsf{com}_j$ during its execution.

Now observe that similar to the previous case, the adversary is committed to its value, and choice of recipients independent of the execution of the honest party $P_{2t}$ that will output a uniformly distributed random string. By the above argument, there exists an honest party $P_{k'} \in \{P_{2t+1}, \ldots, P_{3t+1}\}$ that will open all the adversarial commitments correctly if they have not received `Complain` messages already. Therefore, the final random coin is generated independent of the adversary's choices and is hence uniformly distributed in the coin space.

*Case 3 - Adversary corrupts t roles including* $P_{2t}$: Let the adversary corrupt $t_1 \leq (t-1)$ parties in $\{P_1, \ldots, P_{2t-1}\}$, party $P_{2t}$, and $(t-1)-t_1$ among $\{P_{2t+1}, \ldots, P_{3t+1}\}$. By lemma 34, we know that adversary's commitments will either receive `Complain` message or be opened with a valid opening by some honest party in $\{P_{2t+1}, \ldots, P_{3t+1}\}$. The only analysis left is to show that the adversary cannot bias its behaviour by learning information from honest party's commitments. To see this, we reduce the security to the hiding property of the commitment scheme $\mathsf{COM}$.

The proof follows a standard hybrid argument, where the hybrids are simulated executions of the protocol. More precisely, the simulator simulates the operations of the honest parties, while receiving all the broadcast messages and those that were sent privately by the adversary.

$\underline{\mathsf{Hybrid}_0}$: This hybrid is an execution of the protocol where the honest parties behave as per the protocol specification, and the final random coin is set as the xor of all valid opened values. This is the real-world execution of the protocol.

$\underline{\mathsf{Hybrid}_1}$: This hybrid execution is the same as $\mathsf{Hybrid}_0$, except that the simulator exits the execution of the protocol after party $P_{2t}$ and sets the final random coin. This is done by looking at the random coins being committed to by the honest parties and the openings shares by the adversary. Importantly, the parties $\{P_{2t+1}, \ldots, P_{3t+1}\}$ need not be executed as the final coin is already determined. Note the random coins generated in both hybrids are identically distributed given that the commitments are perfectly binding and following from lemma 34.

$\underline{\mathsf{Hybrid}_2}$: The hybrid execution is the same as $\mathsf{Hybrid}_1$, except for the following. The simulator does not broadcast $(\texttt{Complain}, \cdot)$ on behalf of honest parties for the commitments broadcast by honest parties. In other words, the honest parties do not complain about other honest parties' commitments. The execution in both hybrids is identical as honest parties behave according to the protocol when they commit to values and open them and will receive no complaints from other honest parties.

$\underline{\mathsf{Hybrid}_3}$: This hybrid execution is the same as $\mathsf{Hybrid}_2$, except for the following change. Consider an honest party $P_i$ for $i \in [2t-1]$, such that for some $j \in [m]$, all $M_{jk} = 0$ for all $k \in [i-1]$, and $M_{ji} = 1$. Moreover, we also have that for all $k' \in [i, 2t-1]$, where $M_{jk'} = 1$, party $P_{k'}$ is honest. The existence of such an index $j \in [m]$ is guaranteed by the definition of the matrix $M$ (see lemma 33). The simulator sets the commitment $\mathsf{com}_j$ broadcast by party $P_i$ to be commitments to 0. The final coin is computed right after the execution of $P_{2t}$ like before, but choosing random value $s_j$ as opening for party $P_i$'s commitment $\mathsf{com}_j$. Clearly, the opening is not correct, but note that the opening is not sent to the adversary given that for all $k' \in [i, 2t-1]$, where $M_{jk} = 1$, party $P_{k'}$ is honest.

We now argue that the random coin generated in $\mathsf{Hybrid}_3$ is computationally indistinguishable from a uniformly random string. To see this, we consider a distinguisher $\mathcal{D}$ against the hiding property of $\mathsf{COM}$. The distinguisher chooses a random value $s_j$ and sends $(s_j, 0)$ as the two challenge messages to its challenger. It receives a commitment $\mathsf{com}_j^*$. The distinguisher proceeds with the simulation as in $\mathsf{Hybrid}_2$, except that the $j$-th

commitment broadcast by party $P_i$ is set to be $\mathsf{com}_j^*$. The final random coin denoted by $s$ is computed as in $\mathsf{Hybrid}_2$ using $s_j$ as the supposed opening of the commitment $\mathsf{com}_j^*$. Let $\mathcal{D}'$ be the distinguisher that distinguishes between the random coin generated by $\mathsf{Hybrid}_2$ and $\mathsf{Hybrid}_3$. Distinguisher $\mathcal{D}$ returns the random coin $s$ to $\mathcal{D}'$ and returns whatever bit $b$ that $\mathcal{D}'$ outputs. Notice that if the challenger sets $\mathsf{com}_j^*$ to be the commitment to $s_j$, then the random coin $s$ is set according to $\mathsf{Hybrid}_2$. On the other hand, if $\mathsf{com}_j^*$ was set to be the commitment to 0, then the random coin $s$ is set according to $\mathsf{Hybrid}_3$. Therefore, distinguisher $\mathcal{D}$ is able to win the hiding game with the same non-negligible advantage as that of $\mathcal{D}'$. Given the computationally hiding property of $\mathsf{COM}$, we arrive at a contradiction and can conclude that $\mathsf{Hybrid}_3$ is computationally indistinguishable from $\mathsf{Hybrid}_2$.

We can see that in $\mathsf{Hybrid}_3$ the commitment $\mathsf{com}_j^*$ has no information about $s_j$, and the adversary has committed to its values, commitments, and choice of recipients independent of $s_j$. Therefore, the final coin $s$ is computationally indistinguishable from a uniform random string. □

□

#### 4.8.2.1 Our Protocol in the Sending-leaks Model.

Our sending-leaks model protocol is similar to the above protocol except that we have $2t+1$ receivers instead of $t+1$. More formally, in Protocol 22, we present our randomness generation protocol in the YOSO$^{\mathsf{WCC}}$ sending-leaks model. The security is stated formally in the theorem below.

**Theorem 14.** *If* $\mathsf{COM}$ *is a perfectly binding and computationally hiding non-interactive commitment scheme, then Protocol 22 is a* $(t, n = 4t)$*-computationally secure YOSO*$^{\mathsf{WCC}}$ *randomness generation protocol in the sending-leaks model. Furthermore, its computational and communication complexities are polynomial in* $t$ *and the security parameter* $\lambda$*.*

*Proof.* The proof of the theorem follows the same strategy as the proof for theorem 13. For completeness, we describe it below. We make use of the following adapted lemma.

**Lemma 35.** *If the adversary corrupts some party* $P_{i^*}$ *where* $i^* \in [2t-1]$*, such that for some* $j \in [m]$*,* $M_{jk} = 0$ *for* $k \in [i^*-1]$ *and* $M_{ji^*} = 1$*, then either the adversary's commitment* $\mathsf{com}_j$ *receives a* $(\texttt{Complain}, j)$ *or a valid opening is reconstructed after all parties* $P_h$ *for* $h \in [2t, 4t]$ *have completed their execution.*

*lemma 35.* Consider index $i^* \in [2t-1]$, such that party $P_{i^*}$ is corrupt and any index $j \in [m]$

such that $M_{jk} = 0$ for all $k \in [i^* - 1]$ and $M_{ji^*} = 1$. The adversary broadcasts commitment $\mathsf{com}_j$, and the openings to any party it wishes to. However, note that there exists an honest party $P_k$ for $k \in [i^*, 2t-1]$ and $M_{jk} = 1$, which if it did not receive the valid opening, it would broadcast a message $(\texttt{Complain}, j)$. On the other hand, if the adversary sends a valid opening of the commitment $\mathsf{com}_j$ to party $P_k$, then party $P_k$ would send $t+1$-out of $2t+1$ sharing of the opening to all recipients $\{P_{2t}, \ldots, P_{4t}\}$. That is party $P_h$ for $h \in [2t, 4t]$ receives the $(h-(2t-1))$-th share of the opening. Note that there are at least $t+1$ honest parties in the set $\{P_{2t}, \ldots, P_{4t}\}$ that receive the share of the opening of $\mathsf{com}_j$ from $P_k$ and output that during their execution. This ensures that the opening of the adversarial commitment will be reconstructed after $P_{4t}$ has completed its execution. □ □

*Case 1 - Adversary corrupts t roles in* $\{P_1, \ldots, P_{2t-1}\}$: The case is the same as in the execution leaks. The final values output by $\{P_{2t}, \ldots, P_{4t}\}$ irrespective of the adversarial choices, will result in a random string that is uniformly distributed in the coin space.

*Case 2 - Adversary corrupts t roles except party* $P_{2t}$: Let the adversary corrupt $t_1 \leq (t-1)$ parties in $\{P_1, \ldots, P_{2t-1}\}$ and $t - t_1$ among $\{P_{2t+1}, \ldots, P_{4t}\}$. This case is similar to the execution-leaks model except that we invoke lemma 35 to show that adversarial commitments are either complained on, or successfully opened after $P_{4t}$ completes its execution. This is determined even before party $P_{2t}$ outputs its random value, similar to the previous case. Alternatively, some honest party $P_{k'} \in \{P_{2t+1}, \ldots, P_{3t+1}\}$ outputs the correct opening of $\mathsf{com}_j$ during its execution. Therefore, the final random coin is generated independent of the adversary's choices and is hence uniformly distributed in the coin space.

*Case 3 - Adversary corrupts t roles including* $P_{2t}$: Let the adversary corrupt $t_1 \leq (t-1)$ parties in $\{P_1, \ldots, P_{2t-1}\}$, party $P_{2t}$ and $(t-1)-t_1$ among $\{P_{2t+1}, \ldots, P_{4t}\}$. By lemma 35, we know that adversary's commitments will either receive `Complain` message or be opened with a valid opening after the execution of $P_{4t}$. The only analysis left is to show that the adversary cannot bias its behaviour by learning information from honest party's commitments. To see this, we reduce the security to the hiding property of the commitment scheme $\mathsf{COM}$.

The proof follows a similar hybrid argument, where the hybrids are simulated executions of the protocol. More precisely, the simulator simulates the operations of the honest parties, while receiving all the broadcast messages and those that were sent privately by the adversary.

$\underline{\mathsf{Hybrid}_0}$: This hybrid is an execution of the protocol where the honest parties behave as per the protocol specification, and the final random coin is set as the xor of all valid opened

values. This is the real-world execution of the protocol.

$\underline{\mathsf{Hybrid}_1}$: This hybrid execution is the same as $\mathsf{Hybrid}_0$, except that the simulator exits the execution of the protocol after party $P_{2t}$ and sets the final random coin. This is done by looking at the random coins being committed to by the honest parties and the openings shared by the adversary. Importantly, the parties $\{P_{2t+1}, \ldots, P_{4t}\}$ need not be executed as the final coin is already determined. Note the random coins generated in both hybrids are identically distributed given that the commitments are perfectly binding and following from lemma 35.

$\underline{\mathsf{Hybrid}_2}$: The hybrid execution is he same as $\mathsf{Hybrid}_1$, except for the following. The simulator does not broadcast $(\texttt{Complain}, \cdot)$ on behalf of honest parties for the commitments broadcast by honest parties. In other words, the honest parties do not complain about other honest parties' commitments. The execution in both hybrids is identical as honest parties behave according to the protocol when they commit to values and open them and will receive no complaints from other honest parties.

$\underline{\mathsf{Hybrid}_3}$: This hybrid execution is the same as $\mathsf{Hybrid}_2$, except for the following change. Consider an honest party $P_i$ for $i \in [2t-1]$, such that for some $j \in [m]$, all $M_{jk} = 0$ for all $k \in [i-1]$, and $M_{ji} = 1$. Moreover, we also have that for all $k' \in [i, 2t-1]$, where $M_{jk} = 1$, party $P_{k'}$ is honest. The existence of such an index $j \in [m]$ is guaranteed by the definition of the matrix $M$ (see lemma 33). The simulator sets the commitment $\mathsf{com}_j$ broadcast by party $P_i$ to be a commitment to 0. The final coin is computed right after the execution of $P_{2t}$ like before, but choosing random value $s_j$ as opening for party $P_i$'s commitment $\mathsf{com}_j$. The opening is not correct, but note that the adversary can corrupt at most $t$ parties in $\{P_{2t}, \ldots, P_{4t}\}$ and therefore the opening information $s_j$ is information-theoretically hidden from the adversary.

The argument to show that the random coin generated in $\mathsf{Hybrid}_3$ is computationally indistinguishable from the one in $\mathsf{Hybrid}_2$ is the same as in theorem 13. That is, we reduce the advantage of a distinguishing attacker to the advantage of a distinguished against the hiding property of $\mathsf{COM}$. We can see that in $\mathsf{Hybrid}_3$ the commitment $\mathsf{com}_j^*$ has no information about $s_j$, and the adversary has committed to its values, commitments, and choice of recipients independent of $s_j$. Therefore, the final coin $s$ is computationally indistinguishable from a uniform random string. □

□

## 4.9 Lower Bounds for YOSO$^{\mathsf{WCC}}$ Protocols without Setup

In this section we discuss lower bounds on the number of parties of computationally secure YOSO$^{\mathsf{WCC}}$ randomness generation protocols as a function of the corruption threshold. Such lower bounds were obtained for information-theoretic YOSO$^{\mathsf{WCC}}$ in [92]. More precisely, they showed an impossibility result for $t = 1$ corruptions and $n = 4$ parties, which generalizes directly to an impossibility result for $t$ corruptions and $n = 4t$ parties in the sending-leaks model (and hence to an $n \geq 4t + 1$ lower bound for protocols in this model). However, contrary to what they claim, their impossibility result does not directly extend to $t > 1$ corruptions in the execution-leaks model.

We begin by noting a computational analog of their impossibility result for $t = 1$ corruptions (a case where the execution-leaks and sending-leaks models coincide). The proof follows that of [92] very closely, with some added observations about the efficiency of certain potential attacks. We present it for completeness.

**Theorem 15.** *There is no* $(t = 1, n = 3)$*-computationally secure YOSO*$^{\mathsf{WCC}}$ *randomness generation protocol with bias* $\varepsilon < 0.01$ *in either the execution-leaks or sending-leaks model.*

As mentioned before, one take-away of this theorem combined with our prior feasibility result is that $n = 4$ parties are necessary and sufficient for computationally secure YOSO$^{\mathsf{WCC}}$ randomness generation against $t = 1$ corruptions. In the information-theoretic setting, $n = 5$ parties are known to be necessary and sufficient [92].

*Proof.* Assume for the sake of contradiction that there exists a protocol which satisfies the conditions stated above. Let $f(x_1, x_2, x_3)$ denote the coin output of the protocol. Now, consider party $P_3$. As we are in the plain model without setup, $P_3$ is able to efficiently compute $x_3$ in its head using the messages sent by the honest $P_1$ and $P_2$ and thus compute $f(x_1, x_2, x_3)$. Therefore, $P_3$ should not be able to change the outcome of the coin too much. More formally, consider sampling honest $x_1, x_2, x_3$. Then, it must hold that

$$\Pr[f(x_1, x_2, \bot) = 1 - b | f(x_1, x_2, x_3) = b] \leq \frac{4\varepsilon}{1 + 2\varepsilon}$$

for $b \in \{0, 1\}$, where the probability is taken over the honest sampling of $x_1$, $x_2$, and $x_3$. To see this, consider $b = 0$ without loss of generality. If this inequality does not hold, an adversary who corrupts $P_3$ can efficiently sample $x_3$ honestly, and, if the result is 0, the adversary outputs $\bot$ for $P_3$. In this efficient attack, the final coin is 1 with probability at

least

$$\left(\frac{1}{2}-\varepsilon\right)+\left(\frac{1}{2}+\varepsilon\right)\cdot\Pr[f(x_1,x_2,\perp)=1|f(x_1,x_2,x_3)=0]>1/2+\varepsilon, \tag{4.1}$$

which contradicts the security of the protocol. From eq. (4.1), it follows that

$$\Pr[f(x_1,x_2,\perp)\neq f(x_1,x_2,x_3)]\leq 2\cdot\left(\frac{1}{2}+\varepsilon\right)\cdot\frac{4\varepsilon}{1+2\varepsilon}=4\varepsilon. \tag{4.2}$$

Due to eq. (4.2), not only does $P_3$ have little control over changing the outcome of the coin, but also $P_2$. This is because $P_2$ can now efficiently compute the output of the coin by setting $P_3$'s message to $\perp$. Consider the following: Suppose that an honest $P_1$ outputs a public $x_1$ and sends private messages $s_{1,2}, s_{1,3}$. Given the messages $x_1$ and $s_{1,2}$, sample two values $x_2^1$, $x_2^2$, along with the corresponding private messages $s_{2,3}^1$ and $s_{2,3}^2$ (each in an honest way). Sample $x_3^1$ (resp. $x_3^2$) using $x_1, x_2^1, s_{1,3}, s_{2,3}^1$ (resp. $x_1, x_2^2, s_{1,3}, s_{2,3}^2$) in an honest way. Combining eq. (4.2) with a union bound over the two simulated runs shows that

$$\Pr[f(x_1,x_2^1,x_3^1)=f(x_1,x_2^1,\perp),f(x_1,x_2^2,x_3^2)=f(x_1,x_2^2,\perp)]\geq 1-8\varepsilon.$$

Furthermore, it must hold that

$$\Pr[f(x_1,x_2^1,x_3^1)=f(x_1,x_2^1,\perp),f(x_1,x_2^1,x_3^1)=f(x_1,x_2^2,\perp), f(x_1,x_2^1,\perp)\neq f(x_1,x_2^2,\perp)]\leq\varepsilon. \tag{4.3}$$

Otherwise, an adversary corrupting $P_2$ would be able to efficiently bias by sampling two messages $x_2^1, x_2^2$, each in an honest way, computing $f(x_1,x_2^1,\perp)$ and $f(x_1,x_2^2,\perp)$, and picking the one that corresponds to final coin 1. By combining eqs. (4.2) and (4.3) with a union bound, we conclude that

$$\Pr[f(x_1,x_2^1,x_3^1)=f(x_1,x_2^1,\perp),f(x_1,x_2^1,x_3^1)=f(x_1,x_2^2,\perp), f(x_1,x_2^1,\perp)=f(x_1,x_2^2,\perp)]\geq 1-9\varepsilon.$$

This means that party $P_1$ fully determines the output of the protocol with probability at least $1-9\varepsilon$. In this case, $P_1$ also can also efficiently compute the value of the coin. Thus, an adversary corrupting $P_1$ can conduct the following efficient attack: Sample two sets of messages $x_1^1, s_{1,2}^1, s_{1,3}^1$ and $x_1^2, s_{1,2}^2, s_{1,3}^2$, compute the coin by emulating $P_2$ and setting $P_3$'s message to $\perp$. If either of the coins is 1, output the corresponding set $x_1^i, s_{1,2}^i, s_{1,3}^i$, where $i\in\{1,2\}$. As these are two independent runs of the protocol and we assume (for the sake of contradiction) that the protocol has bias at most $\varepsilon$, with probability at least $\frac{1}{2}-\varepsilon^2$ we get that the two runs result in a different coin. Thus, $P_1$'s attack succeeds with probability at least $(\frac{1}{2}-\varepsilon^2)-2\cdot 9\varepsilon>0.01\geq\varepsilon$, which leads to a contradiction. □ □

Next, we show how to extend theorem 15 to $t > 1$ corruptions and $n = 3t$ parties in the sending-leaks model. This implies that $(t, n)$-computationally secure YOSO$^{\mathsf{WCC}}$ protocols in the sending-leaks model require $n \geq 3t + 1$ parties, for any $t \geq 1$. The proof is simple and intuitive. We obtain an impossibility by arguing that a $(t, n = 3t)$-computationally secure YOSO$^{\mathsf{WCC}}$ protocol in the sending-leaks model can be translated into a protocol for $t = 1$ corruptions and $n = 3$ parties by having each party emulate a block of $t$ consecutive parties in the original protocol.

**Theorem 16.** *There is no $(t, n = 3t)$-computationally secure YOSO$^{\mathsf{WCC}}$ randomness generation protocol for $t > 1$ corruptions with bias $\varepsilon < 0.01$ in the sending-leaks model.*

*Proof.* For the sake of contradiction, say that such a protocol $\Pi$ exists. We will show that given such a protocol, it is possible to obtain a YOSO$^{\mathsf{WCC}}$ protocol $\Pi'$ for $n' = 3$ parties and $t' = 1$ corruptions, which would contradict Theorem 15.

We start by splitting the $n$ parties into three groups $P^1 := \{P_1, \ldots, P_t\}$, $P^2 := \{P_{t+1}, \ldots, P_{2t}\}$, and $P^3 := \{P_{2t+1}, \ldots, P_{3t}\}$. Now, in the protocol $\Pi'$ for $n' = 3$ parties $P'_1, P'_2, P'_3$, we have the first party $P'_1$ do the following: locally execute the protocol $\Pi$ for each of the parties $P_1$ up to $P_t$ one after the other, publish a concatenation of the corresponding public messages $x_1, \ldots, x_t$, and forward private messages that are to be received by parties in $P^2$ to the party $P'_2$ (similarly, forward private messages that are to be received by parties in $P^3$ to the party $P'_3$). Party $P'_2$ then similarly executes the protocol $\Pi$ for each of the parties $P_{t+1}$ up to $P_{2t}$, while publishing the concatenation of the messages $x_{t+1}, \ldots, x_{2t}$, and forwarding private messages designated for parties in $P^3$ to $P'_3$. Finally, party $P'_3$ executes the protocol $\Pi$ for each of the parties $P_{2t+1}$ up to $P_{3t}$, while publishing the concatenation of the messages $x_{2t+1}, \ldots, x_{3t}$. Note that if protocol $\Pi$ is secure, protocol $\Pi'$ is secure as well: Corrupting a party in $\Pi'$ corresponds to corrupting a block of parties in $\Pi$ (crucially, the adversary obtains *all* private messages designated for a corrupt party in $\Pi'$ before starting to execute this party, as in the sending-leaks model the adversary obtains the messages designated to the corrupt parties by the time they are sent). As each block is of size $t$ and $\Pi$ is secure for $t$ corruptions, we get that $\Pi'$ is a secure protocol for $n' = 3$ parties and $t' = 1$ corruptions, thus contradicting Theorem 15. □

□

It is not clear whether theorem 16 applies in the execution-leaks model. Note that in order for, say, party $P'_2$ to emulate the group of $t$ parties $P^2 = \{P_{t+1}, \ldots, P_{2t}\}$ in the proof above, $P'_2$ needs to know the private messages sent from the block $P^1$ to, say, party $P_{2t}$ *already when executing party* $P_{t+1}$. In the execution-leaks model, messages sent to $P_{2t}$ will

only be revealed to the adversary when this party is executed.

Given this, it is natural to attempt to understand whether there are significant differences with respect to feasibility between the execution-leaks and sending-leaks models. To conclude this section, we present a new approach that allows us to show an improved lower bound for $t = 2$ corruptions in the execution-leaks model which matches the lower bound in the sending-leaks model.

**Theorem 17.** *There is no YOSO$^{\mathsf{WCC}}$ protocol for $n = 6$ parties and $t = 2$ corruptions with negligible bias in the execution-leaks model.*

Combining this result with our prior feasibility result allows us to conclude that $n = 7$ parties are necessary and sufficient for computationally secure YOSO$^{\mathsf{WCC}}$ randomness generation against $t = 2$ corruptions in the execution-leaks model. We leave it as an interesting open problem to generalize this result to arbitrary $t > 2$.

*Proof.* Towards a contradiction, suppose that there exists such a protocol. Let $f$ be the deterministic polynomial-time function which computes the coin based on the public broadcasts from the protocol. In other words, if $X_1, X_2, \ldots, X_6$ are published by parties $P_1, P_2, \ldots, P_6$, respectively, then the coin value is $f(X_1, X_2, \ldots, X_6)$. From this assumption, we know that

$$|\Pr[f(X_1, \ldots, X_6) = 0] - \Pr[f(X_1, \ldots, X_6) = 1]| \leq \mathsf{negl}$$

for any PPT execution-leaks adversary that corrupts at most $t = 2$ parties.

We begin by showing that we may assume that if the public values $X_1, \ldots, X_4$ of parties $P_1, \ldots, P_4$, respectively, are sampled honestly according to the protocol, then $P_5$ and $P_6$ both publish $\perp$. We corrupt $P_5$ and $P_6$ and analyze two PPT attacks. First, consider an attack where $P_5$ behaves honestly (publishing $X_5$) and $P_6$ generates its potential public value $X_6$ honestly. If $f(X_1, \ldots, X_6) = 1$ (which $P_6$ can compute), then $P_6$ outputs $X_6$. Otherwise, $P_6$ outputs $\perp$. Since the protocol is secure, we conclude that

$$f(X_1, \ldots, X_5, X_6) = f(X_1, \ldots, X_5, \perp) \tag{4.4}$$

except with negligible probability. Next, consider an attack where $P_5$ first generates its potential public value $X_5$ honestly. From eq. (4.4), we conclude that $P_5$ can efficiently predict the value the coin will take if he decides to publish $X_5$, except with negligible probability, by computing $f(X_1, \ldots, X_5, \perp)$ locally. If $f(X_1, \ldots, X_5, \perp) = 1$, then $P_5$ publishes $X_5$ and $P_6$ publishes $\perp$. Otherwise, $P_5$ and $P_6$ both publish $\perp$. Again by the security of the protocol against $t = 2$ corruptions, we must have

$$f(X_1, \ldots, X_4, \perp, \perp) = f(X_1, \ldots, X_4, X_5, X_6) \tag{4.5}$$

except with negligible probability.

We now corrupt $P_1$ and $P_4$ with the aim of arguing that the private message $s_{1,4}$ from $P_1$ to $P_4$ can be taken to be $s_{1,4} = \perp$, so long as $P_1$ samples its public value and the private messages to $P_2$ and $P_3$ honestly. Consider the attack where $P_1$ behaves honestly except that it sends $s_{1,5} = s_{1,6} = \perp$ and $P_4$ behaves as follows: First, $P_4$ samples $X_4$ honestly and $X'_4$ honestly *but assuming that the private message* $s_{1,4} = \perp$. From eq. (4.5), if $P_4$ publishes $X_4$ then the value of the coin will be $f(X_1, \ldots, X_4, \perp, \perp)$, which $P_4$ can efficiently compute locally, except with negligible probability. If $f(X_1, \ldots, X_4, \perp, \perp) = 1$, then $P_4$ publishes $X_4$. Otherwise, $P_4$ publishes $X'_4$. By the security of the protocol, we must have

$$f(X_1, \ldots, X_4, X_5, X_6) = f(X_1, X_2, X_3, X'_4, X'_5, X'_6) \tag{4.6}$$

except with negligible probability, where $X'_5$ and $X'_6$ are sampled honestly based on $X'_4$ (and on private messages $s_{1,5} = s_{1,6} = \perp$).

Next, corrupt $P_2$ and $P_3$ and consider the following attack. Party $P_2$ behaves honestly, except that it also forwards the private messages $s_{2,4}, s_{2,5}, s_{2,6}$, which it sent to $P_4, P_5, P_6$, respectively, to $P_3$. By eq. (4.6), $P_3$ can efficiently predict the final coin if he publishes $X_3$ except with negligible probability because (1) $P_3$ can efficiently sample $P_4$'s public value $X'_4$ and private messages $s_{4,5}$ and $s_{4,6}$ based on $X_1, X_2, X_3$ and the private messages $s_{1,4} = \perp$ and $s_{2,4}$, and (2) $P_3$ can efficiently sample $X'_5$ and $X'_6$ based on $X_1, X_2, X_3, X'_4$ and the private messages $s_{1,5} = s_{1,6} = \perp$, $s_{2,5}, s_{2,6}$, and $s_{4,5}, s_{4,6}$. Party $P_3$ runs two independent honest samplings of $X_3$ and the messages $s_{3,i}$ for $i > 3$ – denote them by $X_3^j, (s_{3,i}^j)_{i>3}$ for $j \in \{1, 2\}$. If $P_3$ predicts coin value 1 when publishing $X_3^1$ and sending $s_{3,i}^1$ for $i > 3$, it publishes this value and sends these messages to the corresponding later parties. Otherwise, $P_3$ publishes $X_3^2$ and sends $s_{3,i}^2$ for $i > 3$. The security of the protocol then implies that

$$f(X_1, X_2, X_3^1, X_4^1, X_5^1, X_6^1) = f(X_1, X_2, X_3^2, X_4^2, X_5^2, X_6^2) \tag{4.7}$$

except with negligible probability, where $X_4^j, X_5^j, X_6^j$ are generated honestly conditioned on $X_3^j$ for $j \in \{1, 2\}$.

Finally, we corrupt $P_1$ and $P_2$. The attack proceeds as follows: Party $P_1$ samples potential public values $X_1^j, X_2^j$ and potential private messages $(s_{1,i}^j)_{i>1}$ and $(s_{2,i}^j)_{i>2}$, for $j \in \{1, 2\}$. By eq. (4.7), $P_1$ can efficiently predict the value of the coin in the $j$-th run of the protocol based solely on the samples above, assuming that $P_2$ indeed publishes $X_2^j$ and sends private messages $(s_{2,i}^j)_{i>2}$. Therefore, $P_1$ can check whether the $j$-th run leads to coin value 1, publish $X_1^j$, send private messages $(s_{1,i}^j)_{i>1}$, and additionally tell $P_2$ to publish $X_2^j$ and send private messages $(s_{2,i}^j)_{i>2}$. Since the two protocol runs are honest and

independent, the correctness of the protocol ensures that there is a run leading to coin value 1 with probability at least $3/4 - \mathsf{negl}$, and, as we argued above, $P_1$ can predict which run has this property.

□

**Protocol 22** Randomness Generation using $n = 4t$ roles in the Sending-Leaks Model.

We have a $t$-sharing matrix $M \in \{0,1\}^{m \times \ell}$ according to lemma 33, where $\ell = 2t-1$ and $m = \binom{2t-1}{t}$.

1. For $i \in [2t-1]$:
   (a) For $j \in [m]$, party $P_i$ does the following:
      i. If $\forall k \in [i-1], M_{jk} = 0$ and $M_{ji} = 1$,
         A. Choose value $s_j \leftarrow\!\!\$\ \{0,1\}^{\ell_m(\lambda)}$ and generate $\mathsf{com}_j \leftarrow \mathsf{Commit}(s_j; r_j)$ for random coins $r_j \in \{0,1\}^{\ell_r(\lambda)}$.
         B. Broadcast $\mathsf{com}_j$ and send the opening $(s_j, r_j)$ to all parties $P_k$ where $k \in [i+1, \ell]$ and $M_{jk} = 1$.
         C. It generates a $t+1$-out of-$2t+1$ sharing of $(s_j, r_j)$ and send the $k$-th share privately to party $P_k$ where $k \in [2t, 4t]$.
      ii. Else if $M_{ji} = 1$,
         A. Receive $(s_j, r_j)$ from party $P_k$ for some $k \in [i-1]$ and $M_{jk} = 1$.
         B. Broadcast $(\texttt{Complain}, j)$ if nothing was received or if $\mathsf{Commit}(s_j; r_j) \neq \mathsf{com}_j$.
         C. Else, generate a $t+1$-out of-$2t+1$ sharing of $(s_j, r_j)$ and send the $k$-th share privately to party $P_k$ where $k \in [2t, 4t]$.
2. Party $P_{2t}$ samples $s^* \leftarrow\!\!\$\ \{0,1\}^{\ell_m(\lambda)}$ and broadcasts $s^*$.
3. For $i \in [2t, 4t]$, party $P_i$ does the following:
   (a) For any $j \in [m]$, if no message $(\texttt{Complain}, j)$ was seen, then receive shares from different parties for the opening of $\mathsf{com}_j$ and output the shares.

Let $C \subseteq [m]$ be the set of indices such that for any $j \in C$, no message $(\texttt{Complain}, j)$ was seen and there are $t+1$ shares output by the parties $P_{2t}, \ldots, P_{4t}$ that reconstruct the opening $(s_j, r_j)$ such that $\mathsf{Commit}(s_j; r_j) = \mathsf{com}_j$. Let the final randomness be set as $s = \bigoplus_{j \in C} s_j \oplus s^*$.

# Chapter 5

# Conclusion

We are used to relying on digital services in our day-to-day life. However, when we entrust sensitive data to a centralized service, we often give up control over this data. Even if the service itself acts faithfully, it may become the target of an attack, in which case your private data might get leaked anyway. Can't we do better?

Indeed, we can. By distributing trust among multiple parties, we can eliminate single points of failure, reducing the risk of compromise. This thesis makes progress towards obtaining distributed cryptography as a service, laying the foundation for more secure and resilient systems. We started by observing that protocols which are offered as a service ought to be run for long periods of time. To improve the flexibility of the solution and not require long-term commitments from the contributing parties, we aimed to design protocols in a way which makes contributing parties easily replaceable. We further observed that protocols which don't require parties to hold on to local private state easily satisfy this party-replaceability requirement. We designed such protocols for MPC, one under the CSaR assumption, and another one in the YOSO model. We further designed special-purpose stateless MPC solutions for the randomness-generation functionality.

This is an exciting space, and numerous problems remain open:

- Have we arrived at the best possible model for stateless MPC or is there potential for an even better framework? Currently, in addition to YOSO and YOSO$^{\mathsf{WCC}}$, the cryptographic community is exploring various alternative models, such as Fluid MPC [42] and Scales [4]. Each of these models involves distinct trade-offs, and arguable none yet provides a clearly practical and comprehensive solution.
- Can we obtain asynchronous stateless protocols? As of now, all known solutions

in models such as Fluid and YOSO are in a synchronous setting. This means that security guarantees might be lost if network delays end up higher than anticipated by the protocol. Obtaining asynchronous protocols is thus of utmost importance to ensure security in often unpredictable environments such as Internet.

- Another key question is whether it is possible to obtain a comprehensive formal study of stateless protocols. Typically, protocol development in a model begins with foundational components, such as basic secure messaging and broadcast, and gradually builds towards complex constructions like MPC. However, stateless models such as YOSO and Fluid started with MPC, and the low-level primitives in these models are still missing.

# Bibliography


[1] Certificate transparency, 2020. `https://www.certificate-transparency.org/`. 15, 32

[2] Google AI Blog. Brendan McMahan and Daniel Ramage. Federated Learning: Collaborative Machine Learning without Centralized Training Data. `https://ai.googleblog.com/2017/04/federated-learning-collaborative.html`, 2017. 12

[3] Behzad Abdolmaleki, Sebastian Ramacher, and Daniel Slamanig. Lift-and-shift: Obtaining simulation extractable subversion and updatable snarks generically. In *ACM SIGSAC Conference on Computer and Communications Security*, 2020. 38

[4] Anasuya Acharya, Carmit Hazay, Vladimir Kolesnikov, and Manoj Prabhakaran. SCALES - MPC with small clients and larger ephemeral servers. In *Theory of Cryptography*, 2022. 183

[5] Arash Afshar, Payman Mohassel, Benny Pinkas, and Ben Riva. Non-interactive secure computation based on cut-and-choose. In *Annual International Conference on the Theory and Applications of Cryptographic Techniques*, 2014. 23

[6] Ghada Almashaqbeh, Fabrice Benhamouda, Seungwook Han, Daniel Jaroslawicz, Tal Malkin, Alex Nicita, Tal Rabin, Abhishek Shah, and Eran Tromer. Gage MPC: bypassing residual function leakage for non-interactive MPC. *Proceedings on Privacy Enhancing Technologies*, 2021. 24

[7] Bar Alon, Moni Naor, Eran Omri, and Uri Stemmer. MPC for tech giants (GMPC): enabling Gulliver and the Lilliputians to cooperate amicably. In *Annual International Cryptology Conference*, 2024. 82

[8] 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 *Annual International Conference on the Theory and Applications of Cryptographic Techniques*, 2012. 71, 72, 75

[9] Saikrishna Badrinarayanan, Sanjam Garg, Yuval Ishai, Amit Sahai, and Akshay

Wadia. Two-message witness indistinguishability and secure computation in the plain model from new assumptions. In *International Conference on the Theory and Application of Cryptology and Information Security*, 2017. 23

[10] Saikrishna Badrinarayanan, Vipul Goyal, Abhishek Jain, Yael Tauman Kalai, Dakshita Khurana, and Amit Sahai. Promise zero knowledge and its applications to round optimal MPC. In *Annual International Cryptology Conference*, 2018. 12

[11] Saikrishna Badrinarayanan, Aayush Jain, Nathan Manohar, and Amit Sahai. Secure MPC: laziness leads to GOD. In *International Conference on the Theory and Application of Cryptology and Information Security*, 2020. 85

[12] Saikrishna Badrinarayanan, Abhishek Jain, Rafail Ostrovsky, and Ivan Visconti. Non-interactive secure computation from one-way functions. In *International Conference on the Theory and Application of Cryptology and Information Security*, 2018. 23

[13] Thomas Baignères, Cécile Delerablée, Matthieu Finiasz, Louis Goubin, Tancrède Lepoint, and Matthieu Rivain. Trap me if you can – million dollar curve. *IACR Cryptol. ePrint Arch.*, 2015. 120

[14] Donald Beaver. Efficient multiparty protocols using circuit randomization. In *Annual International Cryptology Conference*, 1991. 87

[15] Donald Beaver, Silvio Micali, and Phillip Rogaway. The round complexity of secure protocols. In *Annual ACM Symposium on Theory of Computing*, 1990. 12

[16] Amos Beimel, Ariel Gabizon, Yuval Ishai, Eyal Kushilevitz, Sigurd Meldgaard, and Anat Paskin-Cherniavsky. Non-interactive secure multiparty computation. In *Annual Cryptology Conference*, 2014. 23

[17] Mihir Bellare, Viet Tung Hoang, and Phillip Rogaway. Adaptively secure garbling with applications to one-time programs and secure outsourcing. In *International Conference on the Theory and Application of Cryptology and Information Security*, 2012. 17

[18] Mihir Bellare, Viet Tung Hoang, and Phillip Rogaway. Foundations of garbled circuits. In *ACM Conference on Computer and Communications Security*, 2012. 15

[19] M. Ben-Or, S. Goldwasser, and A. Wigderson. Completeness theorems for noncryptographic fault-tolerant distributed computations. In *Annual Symposium on the Theory of Computing*, 1988. 127

[20] Michael Ben-Or, Shafi Goldwasser, and Avi Wigderson. Completeness theorems for non-cryptographic fault-tolerant distributed computation. In *ACM Symposium on Theory of Computing*, 1988. 85

[21] 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 Conference*, 2020. 3, 37, 83, 84, 85, 113, 114, 115, 120

[22] Fabrice Benhamouda, Shai Halevi, Hugo Krawczyk, Alex Miao, and Tal Rabin. Threshold cryptography as a service (in the multiserver and YOSO models). In *ACM SIGSAC Conference on Computer and Communications Security*, 2022. 83

[23] Fabrice Benhamouda and Huijia Lin. Mr NISC: multiparty reusable non-interactive secure computation. In *Theory of Cryptography Conference*, 2020. 24

[24] Erica Blum, Jonathan Katz, Chen-Da Liu-Zhang, and Julian Loss. Asynchronous byzantine agreement with subquadratic communication. In *Theory of Cryptography*, 2020. 120

[25] Manuel Blum. Coin flipping by telephone a protocol for solving impossible problems. *SIGACT News*, 1983. 122, 139

[26] Manuel Blum, Paul Feldman, and Silvio Micali. Non-interactive zero-knowledge and its applications (extended abstract). In Janos Simon, editor, *Annual ACM Symposium on Theory of Computing*, 1988. 38

[27] Dan Boneh, Joseph Bonneau, Benedikt Bünz, and Ben Fisch. Verifiable delay functions. In *Annual International Cryptology Conference*, 2018. 120

[28] Dan Boneh, Ben Lynn, and Hovav Shacham. Short signatures from the Weil pairing. In *International Conference on the Theory and Application of Cryptology and Information Security*, 2001. 138

[29] Zvika Brakerski, Shai Halevi, and Antigoni Polychroniadou. Four round secure computation without setup. In *Theory of Cryptography Conference*, 2017. 12, 20, 33, 72

[30] Lennart Braun, Ivan Damgård, and Claudio Orlandi. Secure multiparty computation from threshold encryption based on class groups. In *Annual International Cryptology Conference*, 2023. 86, 88, 115

[31] Matteo Campanelli, Bernardo David, Hamidreza Khoshakhlagh, Anders Konring, and Jesper Buus Nielsen. Encryption to the future - A paradigm for sending secret messages to future (anonymous) committees. In *International Conference on the Theory and Application of Cryptology and Information Security*, 2022. 85, 115

[32] Ran Canetti. Universally composable security: A new paradigm for cryptographic protocols. In *Annual Symposium on Foundations of Computer Science*, 2001. 96

[33] Ran Canetti, Abhishek Jain, and Alessandra Scafuro. Practical UC security with

a global random oracle. In *ACM Conference on Computer and Communications Security*, 2014. 23

[34] Ignacio Cascudo and Bernardo David. SCRAPE: Scalable Randomness Attested by Public Entities. In *International Conference on Applied Cryptography and Network Security*, 2017. 120, 140

[35] Ignacio Cascudo and Bernardo David. ALBATROSS: Publicly AttestabLe BATched Randomness based On Secret Sharing. In *International Conference on the Theory and Application of Cryptology and Information Security*, 2020. 120

[36] Ignacio Cascudo and Bernardo David. Publicly verifiable secret sharing over class groups and applications to DKG and YOSO. In *Annual International Conference on the Theory and Applications of Cryptographic Techniques*, 2024. 115

[37] Ignacio Cascudo, Bernardo David, Lydia Garms, and Anders Konring. YOLO YOSO: fast and simple encryption and secret sharing in the YOSO model. In *International Conference on the Theory and Application of Cryptology and Information Security*, 2022. 83, 85

[38] Melissa Chase, Yevgeniy Dodis, Yuval Ishai, Daniel Kraschewski, Tianren Liu, Rafail Ostrovsky, and Vinod Vaikuntanathan. Reusable non-interactive secure computation. In *Annual International Cryptology Conference*, 2019. 23

[39] Jing Chen and Silvio Micali. Algorand: A secure and efficient distributed ledger. *Theoretical Computer Science*, 2019. 120

[40] Kevin Choi, Aathira Manoj, and Joseph Bonneau. SoK: Distributed randomness beacons. In *IEEE Symposium on Security and Privacy*, 2023. 120

[41] Arka Rai Choudhuri, Michele Ciampi, Vipul Goyal, Abhishek Jain, and Rafail Ostrovsky. Round optimal secure multiparty computation from minimal assumptions. In *Theory of Cryptography Conference*, 2020. 12, 27

[42] Arka Rai Choudhuri, Aarushi Goel, Matthew Green, Abhishek Jain, and Gabriel Kaptchuk. Fluid MPC: Secure multiparty computation with dynamic participants. In *Annual International Cryptology Conference*, 2021. 24, 183

[43] Arka Rai Choudhuri, Matthew Green, Abhishek Jain, Gabriel Kaptchuk, and Ian Miers. Fairness in an unfair world: Fair multiparty computation from public bulletin boards. In *ACM SIGSAC Conference on Computer and Communications Security*, 2017. 4, 13, 15, 32

[44] Ronald Cramer, Ivan Damgård, Stefan Dziembowski, Martin Hirt, and Tal Rabin. Efficient multiparty computations secure against an adaptive adversary. In *Annual*

*International Conference on the Theory and Application of Cryptographic Techniques*, pages 311–326, 1999. 127

[45] Ronald Cramer, Ivan Damgård, and Jesper Buus Nielsen. Multiparty computation from threshold homomorphic encryption. In *International Conference on the Theory and Application of Cryptographic Techniques*, 2001. 7, 86, 88

[46] Ivan Damgård, Mads Jurik, and Jesper Buus Nielsen. A generalization of Paillier's public-key system with applications to electronic voting. *International Journal of Information Security*, 2010. 94

[47] Ivan Damgård and Jesper Buus Nielsen. Scalable and unconditionally secure multiparty computation. In *Annual International Cryptology Conference*, 2007. 101

[48] Ivan Damgård, Daniel Escudero, and Antigoni Polychroniadou. Phoenix: Secure Computation in an Unstable Network with Dropouts and Comebacks. In *Conference on Information-Theoretic Cryptography*, 2023. 85

[49] Taher ElGamal. A public key cryptosystem and a signature scheme based on discrete logarithms. *IEEE Transactions on Information Theory*, 1985. 138

[50] Daniel Escudero, Vipul Goyal, Antigoni Polychroniadou, and Yifan Song. Turbopack: Honest majority MPC with constant online communication. In *ACM SIGSAC Conference on Computer and Communications Security*, 2022. 6, 86, 116

[51] Uri Feige, Joe Killian, and Moni Naor. A minimal model for secure computation. In *Annual ACM Symposium on Theory of Computing*, 1994. 23

[52] Matthias Fitzi, Martin Hirt, and Ueli M. Maurer. Trading correctness for privacy in unconditional multi-party computation (extended abstract). In *Annual International Cryptology Conference*, 1998. 85

[53] Matthew K. Franklin and Moti Yung. Communication complexity of secure computation (extended abstract). In *Annual ACM Symposium on Theory of Computing*, 1992. 90

[54] Sanjam Garg, Pratyay Mukherjee, Omkant Pandey, and Antigoni Polychroniadou. The exact round complexity of secure computation. In *Annual International Conference on the Theory and Applications of Cryptographic Techniques*, 2016. 12

[55] 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. In *Annual International Cryptology Conference*, 2021. 2, 5, 24, 82, 83, 84, 85, 86, 88, 92, 93, 94, 115, 120

[56] Craig Gentry, Shai Halevi, Bernardo Magri, Jesper Buus Nielsen, and Sophia Yak-

oubov. Random-index PIR and applications. In *Theory of Cryptography*, 2021. 3, 85, 97, 98

[57] Craig Gentry, Chris Peikert, and Vinod Vaikuntanathan. Trapdoors for hard lattices and new cryptographic constructions. In *Annual ACM Symposium on Theory of Computing*, 2008. 138

[58] O. Goldreich and L. A. Levin. A hard-core predicate for all one-way functions. In *Annual ACM Symposium on Theory of Computing*, 1989. 122, 139

[59] Oded Goldreich, Silvio Micali, and Avi Wigderson. How to play any mental game. In *Annual ACM Symposium on Theory of Computing*, 1987. 11, 12

[60] Shafi Goldwasser, S Dov Gordon, Vipul Goyal, Abhishek Jain, Jonathan Katz, Feng-Hao Liu, Amit Sahai, Elaine Shi, and Hong-Sheng Zhou. Multi-input functional encryption. In *Annual International Conference on the Theory and Applications of Cryptographic Techniques*, 2014. 23

[61] Rishab Goyal and Vipul Goyal. Overcoming cryptographic impossibility results using blockchains. In *Theory of Cryptography Conference*, 2017. 4, 13, 15, 32, 37

[62] Rishab Goyal, Susan Hohenberger, Venkata Koppula, and Brent Waters. A generic approach to constructing and proving verifiable random functions. In *Theory of Cryptography*, 2017. 122, 139

[63] Vipul Goyal. Constant round non-malleable protocols using one way functions. In *Annual ACM Symposium on Theory of Computing*, 2011. 12

[64] Vipul Goyal, Abhiram Kothapalli, Elisaweta Masserova, Bryan Parno, and Yifan Song. Storing and retrieving secrets on a blockchain. In *International Conference on Practice and Theory of Public-Key Cryptography*, 2022. 11

[65] Vipul Goyal, Elisaweta Masserova, Bryan Parno, and Yifan Song. Blockchains enable non-interactive MPC. In *Theory of Cryptography*, 2021. 11

[66] Vipul Goyal, Antigoni Polychroniadou, and Yifan Song. Sharing transformation and dishonest majority MPC with packed secret sharing. In *Annual International Cryptology Conference*, 2022. 87, 90

[67] Jens Groth. Simulation-sound nizk proofs for a practical language and constant size group signatures. In *Conference on the Theory and Application of Cryptology and Information Security*, 2006. 30, 31

[68] Jens Groth. Simulation-sound NIZK proofs for a practical language and constant size group signatures. In *International Conference on the Theory and Application of Cryptology and Information Security*, 2006. 126, 142, 143

[69] Jens Groth. Non-interactive distributed key generation and key resharing. *IACR Cryptol. ePrint Arch.*, 2021. 120

[70] Jens Groth and Mary Maller. Snarky signatures: Minimal signatures of knowledge from simulation-extractable snarks. In *Annual International Cryptology Conference*, 2017. 94

[71] Yue Guo, Rafael Pass, and Elaine Shi. Synchronous, with a chance of partition tolerance. In *Annual International Cryptology Conference*, 2019. 85

[72] Venkatesan Guruswami, Atri Rudra, and Madhu Sudan. *Essential Coding Theory*. 2023. Draft available at `https://cse.buffalo.edu/faculty/atri/courses/coding-theory/book`. 136

[73] Shai Halevi, Yuval Ishai, Abhishek Jain, Ilan Komargodski, Amit Sahai, and Eylon Yogev. Non-interactive multiparty computation without correlated randomness. In *International Conference on the Theory and Application of Cryptology and Information Security*, 2017. 23

[74] Shai Halevi, Yuval Ishai, Abhishek Jain, Eyal Kushilevitz, and Tal Rabin. Secure multiparty computation with general interaction patterns. In *ACM Conference on Innovations in Theoretical Computer Science*, 2016. 23

[75] Shai Halevi, Yehuda Lindell, and Benny Pinkas. Secure computation on the web: Computing without simultaneous interaction. In *Annual Cryptology Conference*, 2011. 23

[76] Martin Hirt and Vassilis Zikas. Adaptively secure broadcast. In *Annual International Conference on the Theory and Applications of Cryptographic Techniques*, 2010. 127

[77] Yuval Ishai, Eyal Kushilevitz, Rafail Ostrovsky, Manoj Prabhakaran, and Amit Sahai. Efficient non-interactive secure computation. In *Annual International Conference on the Theory and Applications of Cryptographic Techniques*, 2011. 23

[78] Aayush Jain, Nathan Manohar, and Amit Sahai. Combiners for functional encryption, unconditionally. In *Annual International Conference on the Theory and Applications of Cryptographic Techniques*, 2020. 26

[79] Gabriel Kaptchuk. Giving state to the stateless: Augmenting trustworthy computation with ledgers. In *Network and Distributed Systems Seminar*, 2020. 4, 13, 15, 32

[80] Jonathan Katz, Rafail Ostrovsky, and Adam Smith. Round efficiency of multi-party computation with a dishonest majority. In *International Conference on the Theory and Applications of Cryptographic Techniques*, 2003. 12

[81] John Kelsey, Luís T. A. N. Brandão, Rene Peralta, and Harold Booth. A reference

for randomness beacons: Format and protocol version 2. Technical report, National Institute of Standards and Technology, 2019. 120

[82] Joe Kilian. A note on efficient zero-knowledge proofs and arguments. In *ACM Symposium on Theory of Computing*, 1992. 20, 73, 74

[83] Sebastian Kolby, Divya Ravi, and Sophia Yakoubov. Constant-round YOSO MPC without setup. *IACR Cryptol. ePrint Arch.*, 2022. 83, 86

[84] Ahmed Kosba, Zhichao Zhao, Andrew Miller, Yi Qian, H Chan, Charalampos Papamanthou, Rafael Pass, Shelat Abhi, and Elaine Shi. CØCØ: a framework for building composable zero-knowledge proofs. *IACR ePrint, 2015/1093*, 2015. 38

[85] Leslie Lamport. Constructing digital signatures from a one way function. Technical report, 1979. 138

[86] Arjen K. Lenstra and Benjamin Wesolowski. A random zoo: sloth, unicorn, and trx. *IACR Cryptol. ePrint Arch.*, 2015. 120

[87] Yehuda Lindell and Benny Pinkas. A proof of security of Yao's protocol for two-party computation. *Journal of Cryptology*, 2009. 28

[88] Chen-Da Liu-Zhang, Elisaweta Masserova, João Ribeiro, Pratik Soni, and Sri AravindaKrishnan Thyagarajan. Improved yoso randomness generation with worst-case corruptions. In *Financial Cryptography and Data Security 2024*, 2024. 120, 133, 165

[89] Mohammad Mahmoody and Rafael Pass. The curious case of non-interactive commitments – on the power of black-box vs. non-black-box use of primitives. In *Annual Cryptology Conference*, 2012. 122, 139

[90] Silvio Micali. Very Simple and Efficient Byzantine Agreement. In *Innovations in Theoretical Computer Science Conference*, 2017. 120

[91] Pratyay Mukherjee and Daniel Wichs. Two round multiparty computation via multi-key FHE. In *Annual International Conference on the Theory and Applications of Cryptographic Techniques*, 2016. 72

[92] Jesper Buus Nielsen, João Ribeiro, and Maciej Obremski. Public randomness extraction with ephemeral roles and worst-case corruptions. In *Annual International Cryptology Conference*, 2022. xiii, 3, 7, 119, 121, 122, 123, 124, 176

[93] Rafael Pass. Bounded-concurrent secure multi-party computation with a dishonest majority. In *Annual ACM Symposium on Theory of Computing*, 2004. 12

[94] Rafael Pass and Elaine Shi. The sleepy model of consensus. In *International Conference on the Theory and Applications of Cryptology and Information Security*, 2017. 120

[95] Torben P. Pedersen. Non-interactive and information-theoretic secure verifiable secret sharing. In *Annual International Cryptology Conference*, 1991. 133

[96] Michael O Rabin. Transaction protection by beacons. *Journal of Computer and System Sciences*, 27(2):256–267, 1983. 120

[97] Tal Rabin. New multiparty computational model: From Nakamoto to YOSO. In *ACM Asia Conference on Computer and Communications Security*, 2022. 83

[98] Claus-Peter Schnorr. Efficient signature generation by smart cards. *Journal of Cryptology*, 1991. 138

[99] Adi Shamir. How to share a secret. *Communications of the ACM*, 1979. 90, 143

[100] Markus Stadler. Publicly verifiable secret sharing. In *International Conference on the Theory and Application of Cryptographic Techniques*, 1996. 122

[101] Ewa Syta, Philipp Jovanovic, Eleftherios Kokoris Kogias, Nicolas Gailly, Linus Gasser, Ismail Khoffi, Michael J. Fischer, and Bryan Ford. Scalable bias-resistant distributed randomness. In *IEEE Symposium on Security and Privacy*, 2017. 120

[102] Sri Aravinda Krishnan Thyagarajan, Guilhem Castagnos, Fabian Laguillaumie, and Giulio Malavolta. Efficient CCA timed commitments in class groups. In *ACM SIGSAC Conference on Computer and Communications Security*, 2021. 120

[103] Brent Waters. Efficient identity-based encryption without random oracles. In *Annual International Conference on the Theory and Applications of Cryptographic Techniques*, 2005. 138

[104] Andrew C Yao. Protocols for secure computations. In *Annual Symposium on Foundations of Computer Science*, 1982. 11, 12, 15, 27

[105] Andrew C. Yao. Theory and application of trapdoor functions. In *Annual Symposium on Foundations of Computer Science*, 1982. 122, 139

[106] Andrew C Yao. How to generate and exchange secrets. In *Annual Symposium on Foundations of Computer Science*, 1986. 15, 27