Blockchains Enable Non-Interactive MPC

Vipul Goyal[1,2], Elisaweta Masserova[1], Bryan Parno[1], and Yifan Song[1]

[1] Carnegie Mellon University
[2] NTT Research

Abstract. We propose to use blockchains to achieve MPC which does not require the participating parties to be online simultaneously or interact with each other. Parties who contribute inputs but do not wish to receive outputs can go offline after submitting a single message. In addition to our main result, we study combined communication and state complexity in MPC, as it has implications for the efficiency of our main construction. Finally, we provide a variation of our main protocol which additionally provides guaranteed output delivery.

1 Introduction

Secure Multiparty Computation (MPC) [Yao82,GMW87] 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 [Yao82,GMW87,GMPP16]. One important question is the round complexity of MPC schemes. In the semi-honest case, in 1990, Beaver et al. [BMR90] gave the first constant-round MPC protocol for three or more parties. A number of works [KOS03,Pas04,Goy11] aiming to analyze and reduce round complexity followed, both in the semi-honest and fully malicious models. In 2016, Garg et al. [GMPP16] 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 [BHP17,BGJ+18,CCG+20], 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 multiparty 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 [Go17]. 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.

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 2.4 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 note that due to the fact that the existing CSaR system [GKM$^+$20] relies on blockchains, and bulletin boards can be realized using blockchains as well [GG17,CGJ$^+$17,KGM20], relying on bulletin boards in our construction effectively does not add extra assumptions. In the following, 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 π into another scheme π'. The contributors in π' are exactly the participants in π. The evaluators can (but do not have to) be entirely different parties from those who contribute inputs in π.

We are now ready to introduce our first result:

Theorem 1. (Informal) *Any MPC protocol π secure against fully-malicious adversaries can be transformed into another MPC protocol π' 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 π' as it is allowed to corrupt participants in π. 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 6 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 π' 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 [3].

Finally, we achieve an MPC protocol which requires only a single round of participation 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 π that is secure against fully-malicious adversaries and provides guaranteed output delivery can be transformed into another MPC protocol π' 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 π' as it is allowed to corrupt participants in π. The adversary is allowed to corrupt any number of evaluators.*

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 [Yao82,Yao86,BHR12b], a primitive which we dub conditional storage and retrieval systems (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 [Yao82,Yao86].

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 [GG17,CGJ+17,KGM20] and as pointed out by these works can

[3] 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.

be realized both from centralized systems such as the Certificate Transparency project [tra20] and decentralized systems such as proof-of-stake or proof-of-work blockchains.

Finally, we define a primitive which we call conditional storage and retrieval systems (CSaRs). 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.

While systems that provide a similar primitive has been proposed in the past [GKM+20,BGG+20] we provide a clean definition that captures the essence of this functionality. We instantiate the CSaR with eWEB [GKM+20], 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 π into another secure MPC protocol π' 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 π garbles the next-message function for each round of π. 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 π. 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 [BHR12a]). 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 σ 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 π. 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 π, 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 3.

Finally, note that we require the following property from the underlying protocol π: given a transcript of execution of π, the output of π can be publicly computed. As we note in Section 3, this property can be easily achieved by slightly modifying the original protocol π.

We provide all protocol details and outline optimizations in Section 3 and give the formal construction in Protocols 1, 2 and 3. 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 1, 2 and 3 transform any MPC protocol π secure against fully-malicious adversaries into another MPC protocol π' 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 π' as it is allowed to corrupt participants in π. 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 π. 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 [4] 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. [BHP17] 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.

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

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 [Kil92] 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 5 with the formal protocol description in Protocol 6. 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 6 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 6 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 2. (Informal) *There exists an MPC protocol π' 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 π to have the GoD property as well, and thus the majority of the participants in π (recall that these are exactly the contributors in π') must be honest [5]. While making this change (in addition to a

[5] Note that there is no such restriction on the evaluators in π'.

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.

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 π 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 π. 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 π achieves guaranteed output delivery, the protocol we propose achieves guaranteed output delivery as well.

We give full details of the GoD construction in Section 6. The statement about our GoD construction is given below.

Lemma 2. (Informal) *Any MPC protocol π which is secure against fully-malicious adversaries and provides guaranteed output delivery can be transformed into another MPC protocol π' 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 π' as it is allowed to corrupt participants in π. The adversary is allowed to corrupt any number of evaluators.*

1.3 Related Work

Closest to our work is the line of research that studies non-interactive multiparty computation [HIJ+17,FKN94,HLP11], initiated in 1994 by Feige et al. [FKN94], 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. [HLP11] 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. [BGI+14] and Halevi et al. [HIJ+16] subsequently removed the requirement of sequential interaction. Further improving upon these results, the work of Halevi et al. [HIJ+17] removed the requirement of a complex correlated randomness setup that was present in a number of previous works [BGI+14,HIJ+16,GGG+14]. Halevi et al. [HIJ+17] 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) [AMPR14,BGI+17,BJOV18,CDI+19,CJS14], initiated by Ishai et al. [IKO+11]. 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 [BL20] 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. [ABH+21] 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, recently two works ([CGG+21] and [GHK+21]) appeared which are inspired by blockchains and focus on improving the flexibility of the MPC protocols. Choudhuri et al. [CGG+21] proposed the notion of *fluid* MPC which allows parties to dynamically join and leave the computation. Gentry et al. [GHK+21] proposed the YOSO ("You Only Speak Once") model which focuses on stateless parties which can only send a single message. Similar to us, their 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 Preliminaries

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

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 [6], 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.*

[6] Note that we will relax this requirement later, also allowing MPC protocols which use secure point-to-point channels. See Section 3 for details.

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 1. In the real world, parties engage in the real-world MPC protocol π in 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 π is defined as follows:

Definition 2. *A protocol π 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 λ, 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 π 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 π has the property that given the transcript of the protocol execution, the output can be publicly computed (as defined in [JMS20]):

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

In one of our constructions, we consider MPC protocols which provide guaranteed output delivery. In that case the security of protocol π is defined the same way as before, except that the ideal functionality is now $\mathcal{F}_{\text{MPC-GoD}}$, described in Functionality 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 [Yao82,Yao86]. In the following, we provide definitions for the garbling process as well as the security of garbling scheme (taken verbatim from [CCG+20]):

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

Functionality 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.

Functionality 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.

- $(\{\mathtt{lab}^{w,b}\}_{w \in \mathtt{inp}, b \in \{0,1\}}) \leftarrow \mathtt{Gen}(1^\lambda, \mathtt{inp})$: `Gen` *takes the security parameter* 1^λ *and length of input for the circuit as input and outputs a set of input labels* $\{\mathtt{lab}^{w,b}\}_{w \in \mathtt{inp}, b \in \{0,1\}}$.
- $\bar{C} \leftarrow \mathtt{Garble}(C, (\{\mathtt{lab}^{w,b}\}_{w \in \mathtt{inp}, b \in \{0,1\}})$: `Garble` *takes as input a circuit* $C : \{0,1\}^{\mathtt{inp}} \to \{0,1\}^{\mathtt{out}}$ *and a set of input labels* $\{\mathtt{lab}^{w,b}\}_{w \in \mathtt{inp}, b \in \{0,1\}}$ *and outputs the garbled circuit* $\bar{C}$.
- $y \leftarrow \mathtt{Eval}(\bar{C}, \mathtt{lab}^x)$: `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 [LP09].

2.3 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 [KGM20], we assume that parties publish their strings as part of a public chain of values, and abstract the bulletin board syntax as follows:

- $(\mathtt{post}, \sigma) \leftarrow \mathtt{Post}(\mathtt{M})$. 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 $(\mathtt{post}, \sigma)$, where σ 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.
- $\{0,1\} \leftarrow \mathtt{Verify}(\mathtt{post}, \sigma)$. The public verification algorithm takes as input a supposedly published record `post` as well as a proof of publish σ, 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 [GG17,CGJ+17,KGM20]. 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 [tra20], and from the decentralized systems such as proof-of-work or proof-of-stake blockchains.

2.4 CSaRs

In our work, we rely on what we call conditional storage and retrieval systems (CSaRs) that allow for a secure storage and retrieval of secrets. In more detail, the user who stores the secret with a CSaR specifies a release condition, and the secret is released if and only if this condition is satisfied. While such systems could be realised via a trusted third party, they can also be realised 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. Recently, multiple independent works appeared that use blockchains to provide such functionality [GKM+20,BGG+20]. We provide a clean definition of the core functionality that these works aim to provide (without fixating on blockchains) and outline why the eWEB system [GKM+20] satisfies this definition.

Formally, the ideal CSaR functionality is described in Figure 3. The security of a CSaR system is then defined 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.

2.5 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 4. Note

Fig. 3. 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 τ 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.

that the difference to the standard ideal functionality for MPC with abort (described in Figure 1) is that we distinguish between contributors and evaluators.

In the real world, parties execute the protocol π 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 π is defined as follows:

Definition 5. *A protocol π 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 λ, it holds that*

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

where $z \in \{0,1\}^*$ is the auxiliary input, $\mathtt{IDEAL}_{f,S(z),I}$ denotes the output of the interaction of the adversary $S(z)$ (who corrupts parties in I) with the ideal functionality $\mathcal{F}_{eval\text{-}MPC}$ (this output consists of the output of the adversary $S(z)$ as well as the outputs of the honest parties), and $\mathtt{REAL}_{\pi,A(z),I}$ denotes the output of the interaction between the adversary $A(z)$ who corrupts parties in I and the honest parties in the protocol π (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 π is defined the same way as before, except that the ideal functionality is now $\mathcal{F}_{\text{eval-MPC-GoD}}$, described in Functionality 5.

Functionality 4. $\mathcal{F}_{\text{eval-MPC}}$

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 5. $\mathcal{F}_{\text{eval-MPC-GoD}}$

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.

2.6 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. [BHP17], which in turn utilizes multi-key fully homomorphic encryption scheme with distributed setup. In the following, we for-

mally define this primitive (in large parts taken verbatim from Brakerski et al. [BHP17]).

Definition 6 (Multi-key homomorphic encryption scheme). *A multi-key homomorphic encryption scheme with distributed setup consists of five procedures,* $\mathsf{MFHE} = (\mathsf{MFHE.DistSetup}, \mathsf{MFHE.Keygen}, \mathsf{MFHE.Encrypt}, \mathsf{MFHE.Decrypt}, \mathsf{MFHE.Eval})$:

- Setup $\mathsf{params}_i \leftarrow \mathsf{MFHE.DistSetup}(1^\kappa, 1^N, i)$: On input the security parameter κ and number of users N, outputs the system parameters for the i-th player 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 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 κ) 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 skiiI , 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 params_i.

They consider a rushing adversary that chooses N and $i \in [N]$, then it sees params_i and produces 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.

Simulatability 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).

3 Our Non-Interactive MPC Construction

We now present our first construction - given an MPC protocol π, we use Yao's garbled circuits as well as a CSaR to transform it into an MPC protocol π' that does not require parties to be online at the same time and only requires a single message from the contributors in π. The contributors in π 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 π. 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. Such bulletin boards can be implemented in practice via a blockchain. Finally,we assume IND-CCA secure public key encryption.

For the ease of presentation, we assume the following about the MPC protocol π: (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 [7].

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.

[7] 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.

3.1 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 π.

Step 1. Preparing Garbled Circuits and Keys. Each party P_j that wishes to participate (contribute inputs) in π starts by garbling the slightly modified next-message functions of each round of π. 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 π. Once all required information is stored, an evaluator E can execute the original MPC protocol π. It is not required that E is one of the parties participating in the protocol π 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 π 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 protocol that assumes point-to-point channels can be easily converted to a protocol in a broadcast model. A generic transformation is outlined in the eWEB paper (Protocols 1 and 2 in [GKM+20]), it requires using a protocol to "package" a message that must be sent and another protocol to "unpack" a message received by a party. Intuitively, these protocols rely on authenticated communication channels (which can be realized via signatures). The packaging is done via appending the id of the sender to the message and IND-CCA encrypting the resulting string. The unpacking is done via decrypting and verifying that the party id specified in the message corresponds to the id of the party who sent this message via the authenticated communication channel.

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 6. For security purposes, it is important that we do *not* perform any circuit optimizations based on the value of w.

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

Fig. 6. 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.

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 π 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 [8].

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 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

[8] 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.

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 π 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 π has the publicly recoverable output property and Eval denotes the algorithm used to retrieve the output from the transcript.

The full construction is given in Protocols 1 and 2 (preparation of the garbled circuits and keys), as well as Protocol 3 (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 π. 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 π. We give the formal proof in Section A.

4 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 1, 2 and 3), 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 π, n_{rounds} denote the number of rounds in π, $n^i_{wires,j}$ 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:

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

Protocol 1 NON-INTERACTIVE MPC – *CircuitPreparationPhase*

1. P_j computes the output (m_j^1, s_j^1) of the first round of π. 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 $\mathtt{M}_\mathtt{j}^\mathtt{1} = (id, 1, j, m_j^1, sigpub_j^1)$ 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 π:

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

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

   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 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 $i - 1$.

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 $\sum_{i=2}^{n_{rounds}} \sum_{i=1}^{n} 2 * n_{wires,j}^i$ is added because each party also needs to store two wire keys for each input wire of each garbled circuit it publishes.

Protocol 2 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).

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

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

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 $\sum_{i=2}^{n_{rounds}} \sum_{j=1}^{n} n_{wires,j}^{i}$. This term is precisely the combined communication and (encrypted) state complexity of the original MPC protocol π, 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.

5 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 [AJLA+12], 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

Protocol 3 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 π. 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 σ^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 $\mathtt{M}^{\mathtt{d}} = (id, d, m_1^{d-1} \| \cdots \| m_n^{d-1} \| sigpub_1^{d-1} \| \cdots \| sigpub_n^{d-1} \| c_1^{d-1} \| \cdots \| c_n^{d-1} \| sigpr_1^{d-1} \| \cdots \| sigpr_n^{d-1})$:

   $$(\mathtt{post}^{\mathtt{d}}, \sigma^d) \leftarrow \mathtt{Post}(\mathtt{M}^{\mathtt{d}})$$

   and uses the proof of publishing σ^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 τ denote the resulting transcript of execution of π. E outputs $\mathsf{Eval}(\tau)$ as the result.

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.

5.1 Step. 1: MPC with semi-malicious security

Our starting point is the solution proposed in the work of Brakerski et al. [BHP17] 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

[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

standard techniques to randomized functionalities with individual outputs for different parties [AJLA+12]. 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 [MW16] 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 [MW16] for details.

To summarize, the construction in Protocol 4 is an MPC protocol secure against semi-malicious adversaries and can handle functions of arbitrary depth [10].

The communication complexity in Protocol 4 depends only on the security parameters, the number of parties, and input and output sizes [BHP17]. Note that for a party P_k the state that is passed between the rounds in Protocol 4 consists of the following data:

- 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 does not depend on the circuit we are computing.

5.2 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 [Kil92]. 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.

6.2 in [MW16]) 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.

Protocol 4 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 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 $y = y_1 \ldots y_{l_{out}}$.

Kilian's zero-knowledge protocol Kilian's construction [Kil92] 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 5) – 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.

Full construction The MPC construction secure against fully malicious adversaries is effectively the same as the semi-malicious one, except that additionally

Protocol 5 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 ψ. P computes the root of the Merkle tree (using the hash function h) on ψ, 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 $\perp$.

the parties commit to their input and randomness in the semi-malicious protocol and prove (using any zero-knowledge argument of knowledge, denoted by 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 4. In more detail:

We assume that there exists some ordering of parties participating in Protocol 4. Following the approach outlined by Asharov et al. [AJLA+12], in each round d of Protocol 4 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, $\mathsf{NextMessage}$ is the function executed by P_i in this round according to Protocol 4, 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 4 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.

5.3 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 6 is an MPC protocol computing f in the plain (authenticated broadcast) model which is secure against fully*

Protocol 6 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 4.
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 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 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 5 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 5 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 $\mathsf{NextMessage}_4(x_z, r_z, \{m^k\}_{k=1}^4, c_z^3) = m_z^d$.

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 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 4 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 6 is independent of the function the MPC protocol is tasked with computing.

Integrating communication and state optimized MPC As we showed in Section 4, the number of CSaR secret store operations in our non-interactive MPC construction (Protocols 1, 2 and 3) 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:

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

[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.

As we pointed out in Section 4, the term $\sum_{i=2}^{n_{rounds}} \sum_{j=1}^{n} n_{wires,j}^{i}$ is precisely the combined communication and (encrypted) state complexity of the underlying MPC protocol π, minus the messages of the first round and plus signatures on the public messages and the state. Thus, when using Protocol 6 as the underlying protocol π in our main non-interactive MPC construction (Protocols 1, 2 and 3), we obtain a construction which number of CSaR store and release operations depends only on the number of rounds in π, security parameters, number of parties, and input and output sizes. All of these parameters are independent of the function that π is tasked with computing.

Thus, we get the following result:

Corollary 3. *There exists an MPC protocol π' 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.*

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 π 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 τ.
- 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 τ 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 7), 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 $(m_1^d \parallel \cdots \parallel m_n^d \parallel c_1^d \parallel \cdots \parallel c_n^d \parallel sig_1^d \parallel \cdots \parallel sig_n^d \parallel \mathcal{P})$, 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 [12]. The following exceptions are allowed:

[12] 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.

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 τ 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 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 τ 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 τ 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 τ 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 π 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 π has guaranteed output delivery, each honest participant in π is guaranteed to be able to correctly compute the honest output. Given

honest majority among the participants (which we assume in order for π 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 π has the publicly recoverable output property with guaranteed otput delivery and Eval denotes the algorithm used to retrieve the output from the transcript.

The full construction is given in Protocols 7 and 8 (preparation of the garbled circuits and keys), as well as Protocol 9 (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 §B.

Protocol 7 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 π:

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

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

   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 $m^i = m_1^{i-1} \| \cdots \| m_n^{i-1}$.
   (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 τ 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 τ.

Protocol 8 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 τ 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 τ 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).

7 Acknowledgments

Bryan Parno, A and Elisaweta Masserova were supported by a fellowship from the Alfred P. Sloan Foundation, a gift from Bosch, NSF Grant No. 1801369, and by the CONIX Research Center, one of six centers in JUMP, a Semiconductor Research Corporation (SRC) program sponsored by DARPA. Vipul Goyal and Yifan Song were supported by the NSF award 1916939, DARPA SIEVE program, a Cylab Presidential Fellowship, a gift from Ripple, a DoE NETL award, a JP Morgan Faculty Fellowship, a PNC center for financial services innovation award, and a Cylab seed funding award.

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

1. Wait until either deadline τ 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 π. 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 τ' denote the resulting transcript of execution of π. E outputs $\mathsf{Eval}(\tau')$ as the result.

References

Go17. 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.

ABH+21. 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. Cryptology ePrint Archive, Report 2021/256, 2021. `https://eprint.iacr.org/2021/256`.

AJLA+12. Gilad Asharov, Abhishek Jain, Adriana López-Alt, Eran Tromer, Vinod Vaikuntanathan, and Daniel Wichs. Multiparty computation with low communication, computation and interaction via threshold fhe. In Annual International Conference on the Theory and Applications of Cryptographic Techniques, pages 483–501. Springer, 2012.

AMPR14. 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, pages 387–404. Springer, 2014.

BGG+20. 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, pages 260–290. Springer, 2020.

BGI+14. Amos Beimel, Ariel Gabizon, Yuval Ishai, Eyal Kushilevitz, Sigurd Meldgaard, and Anat Paskin-Cherniavsky. Non-interactive secure multiparty computation. In Annual Cryptology Conference, pages 387–404. Springer, 2014.

BGI+17. 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, pages 275–303. Springer, 2017.

BGJ+18. 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, pages 459–487. Springer, 2018.

BHP17. Zvika Brakerski, Shai Halevi, and Antigoni Polychroniadou. Four round secure computation without setup. In Theory of Cryptography Conference, pages 645–677. Springer, 2017.

BHR12a. 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, pages 134–153. Springer, 2012.

BHR12b. Mihir Bellare, Viet Tung Hoang, and Phillip Rogaway. Foundations of garbled circuits. In Proceedings of the 2012 ACM conference on Computer and communications security, pages 784–796, 2012.

BJOV18. 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, pages 118–138. Springer, 2018.

BL20. Fabrice Benhamouda and Huijia Lin. Mr nisc: Multiparty reusable non-interactive secure computation. In Theory of Cryptography Conference, pages 349–378. Springer, 2020.

BMR90. Donald Beaver, Silvio Micali, and Phillip Rogaway. The round complexity of secure protocols. In Proceedings of the twenty-second annual ACM symposium on Theory of computing, pages 503–513, 1990.

CCG+20. 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, pages 291–319. Springer, 2020.

CDI+19. 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, pages 462–488. Springer, 2019.

CGG+21. 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, pages 94–123. Springer, 2021.

CGJ+17. 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 Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security, pages 719–728, 2017.

CJS14. Ran Canetti, Abhishek Jain, and Alessandra Scafuro. Practical UC security with a global random oracle. In Proceedings of the 2014 ACM SIGSAC Conference on Computer and Communications Security, pages 597–608, 2014.

FKN94. Uri Feige, Joe Killian, and Moni Naor. A minimal model for secure computation. In Proceedings of the twenty-sixth annual ACM symposium on Theory of computing, pages 554–563, 1994.

GG17. Rishab Goyal and Vipul Goyal. Overcoming cryptographic impossibility results using blockchains. In Theory of Cryptography Conference, pages 529–561. Springer, 2017.

GGG+14. 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, pages 578–602. Springer, 2014.

GHK+21. Craig Gentry, Shai Halevi, Hugo Krawczyk, Bernardo Magri, Jesper Buus Nielsen, Tal Rabin, and Sophia Yakoubov. Yoso: You only speak once. In Annual International Cryptology Conference, pages 64–93. Springer, 2021.

GKM+20. Vipul Goyal, Abhiram Kothapalli, Elisaweta Masserova, Bryan Parno, and Yifan Song. Storing and retrieving secrets on a blockchain. Cryptology ePrint Archive, Report 2020/504, 2020. https://eprint.iacr.org/2020/504.

GMPP16. 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, pages 448–476. Springer, 2016.

GMW87. Oded Goldreich, Silvio Micali, and Avi Wigderson. How to play any mental game. In Proceedings of the Nineteenth ACM Symp. on Theory of Computing, STOC, pages 218–229. ACM, 1987.

Goy11. Vipul Goyal. Constant round non-malleable protocols using one way functions. In Proceedings of the forty-third annual ACM symposium on Theory of computing, pages 695–704, 2011.

HIJ+16. Shai Halevi, Yuval Ishai, Abhishek Jain, Eyal Kushilevitz, and Tal Rabin. Secure multiparty computation with general interaction patterns. In Proceedings of the 2016 ACM Conference on Innovations in Theoretical Computer Science, pages 157–168, 2016.

HIJ+17. 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, pages 181–211. Springer, 2017.

HLP11. Shai Halevi, Yehuda Lindell, and Benny Pinkas. Secure computation on the web: Computing without simultaneous interaction. In Annual Cryptology Conference, pages 132–150. Springer, 2011.

IKO+11. 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, pages 406–425. Springer, 2011.

JMS20. Aayush Jain, Nathan Manohar, and Amit Sahai. Combiners for functional encryption, unconditionally. In Annual International Conference on the Theory and Applications of Cryptographic Techniques, pages 141–168. Springer, 2020.

KGM20. Gabriel Kaptchuk, Matthew Green, and Ian Miers. Giving state to the stateless: Augmenting trustworthy computation with ledgers. In Network and Distributed Systems Seminar, volume 1, 2020.

Kil92. Joe Kilian. A note on efficient zero-knowledge proofs and arguments. In Proceedings of the twenty-fourth annual ACM symposium on Theory of computing, pages 723–732, 1992.

KOS03. 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, pages 578–595. Springer, 2003.

LP09. Yehuda Lindell and Benny Pinkas. A proof of security of yao's protocol for two-party computation. Journal of cryptology, 22(2):161–188, 2009.

MW16. 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, pages 735–763. Springer, 2016.

Pas04. Rafael Pass. Bounded-concurrent secure multi-party computation with a dishonest majority. In Proceedings of the thirty-sixth annual ACM symposium on Theory of computing, pages 232–241, 2004.

tra20. Certificate transparency, 2020. `https://www.certificate-transparency.org/`.

Yao82. Andrew C Yao. Protocols for secure computations. In 23rd annual symposium on foundations of computer science (sfcs 1982), pages 160–164. IEEE, 1982.

Yao86. Andrew C Yao. How to generate and exchange secrets. In 27th Annual Symposium on Foundations of Computer Science (sfcs 1986), pages 162–167. IEEE, 1986.

A Security Proof - Main Construction

Formally, we show that our construction supports the MPC functionality $\mathcal{F}_{\text{eval-MPC}}$ described in Functionality 4.

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 secure protocol π. 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 $\mathsf{Ideal}_{\text{CSaR}}$. 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 $\mathsf{Ideal}_{\text{CSaR}}$ 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 $\mathsf{Ideal}_{\text{CSaR}}$) (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 3, 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 ϵ_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 1, 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.

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 1 and 2 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 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 1 or of Protocol 2 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.

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.

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* ***Hybrid$_5$*** *and* ***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 **Hybrid$_4$** and **Hybrid$_5$** as well) are indistinguishable.

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 abort_2.

Lemma 8. *For the hybrids* ***Hybrid$_6$*** *and* ***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* ***Hybrid$_7$*** *and* ***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 **Hybrid$_9$**, the simulator S computes the garbled circuit directly as $S_{GC}(\perp)$.

Lemma 10. *For the hybrids* ***Hybrid$_8$*** *and* ***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 1. Thus, the input we feed into the garbled circuit simulator does not change.

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 π 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 **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* ***Hybrid$_9$*** *and* ***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 π. Thus, the indistinguishability of the hybrids holds by the security of the original MPC protocol π.

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

B 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 π. 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 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 $(id, 1, j, m_j^1)$, 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 τ 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 τ 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 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 9, 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 ϵ_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):

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

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 **Hybrid$_1$** and **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 **Hybrid$_2$** and **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 abort_1.

Lemma 14. *For the hybrids **Hybrid$_3$** and **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 abort_2.

Lemma 15. *For the hybrids* ***Hybrid$_4$*** *and* ***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 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 τ 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 π 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* π*.*

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.

C 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 7.

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.

Fig. 7. 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 τ 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.

D Instantiating CSaRs

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 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 σ. 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 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.

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

$\mathbf{Hybrid_1}$: 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.