Blockchains Enable Non-Interactive MPC

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

[1] Carnegie Mellon University
goyal@cmu.edu, elisawem@cs.cmu.edu, parno@cmu.edu, yifans2@andrew.cmu.edu
[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 communication complexity 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 for an *entire Election Day*. In this work, we rely on blockchains to achieve MPC that *does not require participants to be online at the same time or interact with each other*.

Such non-interactive solutions advance the state of the art of secure multi-party computation, opening up a whole new realm of possible applications. For example, *passive data collection* for privacy preserving collaborative machine learning becomes possible. Federated learning is already used to train machine learning models for the keyboards of mobile devices for the purposes of autocorrect and predictive typing [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? Furthermore, is it possible to design such a protocol under the constraint that only a single message is required from the parties supplying the inputs, and the parties can go offline after submitting this message 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 is not required to interact with other such participants and can go offline after sending only a single message. 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 a 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 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 existing CSaR systems [GKM$^+$20,BGG$^+$20] rely on blockchains, and bulletin boards can be realized using blockchains as well [GG17,CGJ$^+$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*).

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. Only a single message is required from the MPC contributors (the evaluators might be required to produce multiple messages). The adversary is allowed to corrupt as many MPC contributors in π' as is supported by the protocol π.*

In addition to this result, we discuss ways to optimize our construction. To this end, we explain why the *combined communication- and state complexity* 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 (and thus communication complexity) 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 comnined communication and state complexity, we assume multi-key fully homomorphic encryption, probabilistically checkable proofs, collision-resistant hash functions, and IND-CPA secure public key encryption. 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 communication and state complexity depend only on security parameters, 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. Only a single message is required from the MPC contributors (the evaluators might be required to*

produce multiple messages). Furthermore, the communication complexity of this protocol is independent of the function that is being computed using this MPC protocol.

Finally, we achieve an MPC protocol which requires only a single message 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. Only a single message is required from the MPC contributors (the evaluators might be required to produce multiple messages). The adversary is allowed to corrupt as many MPC contributors in π' as is supported by the protocol π.*

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 (*CSaR*s) 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 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] [3], 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

[3] Other instantiations are possible, see Section 2 for details.

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:

Note that 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.

Finally, 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.

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. Only a single message is required from the MPC contributors (the evaluators might be required to produce multiple messages). The adversary is allowed to corrupt as many MPC contributors in π' as is supported by the protocol π.*

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 chose this construction in particular because its communication and state complexity depends only on the security parameters, the number of parties, and the input- and output sizes. In particular, note that the construction's combined communication- and state complexity is independent of the function we are computing.

Our next step is to extend this construction so that it provides security against malicious adversaries. For this, we propose to use the zero-knowledge protocol proposed by Kilian [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

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

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. *Let f be an N-party function. Protocol 6 is an MPC protocol computing f in the plain 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 communication complexity 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 (thus, the majority of the MPC contributors must be honest). While making this change (in addition to a few minor adjustments) would be enough to guarantee GoD in our construction in the setting with only a single evaluator, it is certainly not sufficient when there are multiple evaluators, some of them dishonest. This is due to the following issue: since we must prevent an adversary from executing honest garbled circuits on multiple different inputs, we cannot simply allow each evaluator to execute garbled circuits on the inputs of its choosing. In particular, the CSaR release conditions must ensure that for each wire only a *single* key is revealed. In our first construction this results in the malicious evaluator being able to prevent an honest evaluator from executing the garbled circuits as intended by submitting an invalid first message for any round. Thus, to ensure guaranteed output delivery while maintaining secrecy, we must ensure

that a malicious evaluator posting a wrong message does *not* prevent an honest evaluator from posting a correct message and using it for the key reveal. In particular, we will ensure that only a *correct* (for a definition of "correctness" explained below) message can be used for the wire key reveal.

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

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 letting the CSaR system check every output of the evaluator that appears to be of an invalid form (e.g., missing a signature, having an unexpected length, etc.) and verify 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 incorrect-looking 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 incorrect-looking 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. Only a single message is required from the MPC contributors (the evaluators might be required to produce multiple messages). The adversary is allowed to corrupt as many MPC contributors in π' as is supported by the protocol π.*

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 NIZC 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. Furthermore, in contrast to the work of Benhamouda and Lin, our construction does not require any trusted setup [5] even in the fully malicious model.

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

[5] Other than a PKI.

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 – CSaRs

In our work, we rely on what we call *conditional storage and retrieval systems* (*CSaR*s) that allow for a secure storage- and retrieval of secrets. In 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. Note that the system proposed by Benhamouda et al. does not formally explain how the secrets can be stored to- and retrieved from the blockchain given a specific release condition. While this requires further research, it should be possible to take the same approach as is used by the eWEB system. Thus the system by Benhamouda et al. is also a viable candidate for a CSaR instantiation.

Formally, the ideal CSaR functionality is described in Figure 1. 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.

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. 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.

Fig. 1. 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.

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, and digital signatures.

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 [6].

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

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.

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

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 output of these circuits on a the 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 circuit. 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 $(m_1^d \| \cdots \| m_n^d \| \mathit{sigpub}_1^d \| \cdots \| \mathit{sigpub}_n^d \| c_1^d \| \cdots \| c_n^d \| \mathit{sigpr}_1^d \| \cdots \| \mathit{sigpr}_n^d)$. 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.

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

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

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. 2. 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.

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 [7].

Additionally, we note that in eWEB the party storing the secret is required to send multiple messages. In order to ensure that in our MPC protocol a single message from the 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 ob s by subtracting their shares of r from $s + r$. Along the way, the commitments to the sharing of s are made public. We modify it as follows: the user simply publishes the commitments to the sharing of s and sends shares of s (along with the witnesses) to the miners who then verify the correctness of the shares and witnesses.

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 that does not have access to the parties' secrets. No PPT adversary can distinguish between interaction with the simulator and the interaction with the honest parties. 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

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

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 $(id, 1, j, m_j^1, sigpub_j^1)$ on chain.
2. P_j produces Yao's garbled circuits $\{GC_j^i\}$ for each round i based on the circuit C_j^i of the next-message function f^i of the original MPC protocol π. The circuit C_j^i for which P_j does the garbling 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 encryption of the state c_j^{i-1} using $verk_j$. If this check fails, stop the execution and output $\perp$.
   (c) Verify the signature on each public message 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 $f^i(x_j, r_j^i, m^i, s_j^{i-1})$, 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 tables for all of the rounds using a CSaR. The witness needed to release the garbled circuit of round i is a valid proof of publishing of all authoritative messages from round 1 and up to and including round $i - 1$.

Protocol 2 NON-INTERACTIVE MPC – $KeyStoragePhase$

1. Securely store input wire keys 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 are published.
   (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 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 are published.
   (b) i-th bit of the authoritative message of round $d - 1$ is 0 (resp. 1).

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 $(id, 2, m)$ and use the proof of publish as the witness to decrypt the wire keys of the next round. If an authoritative message $(id, 2, m)$ was published on the bulletin board, use it as witness to compute the outputs of the next round if $m = m_1^2 \| \cdots \| m_n^2 \| sigpub_1^2 \| \cdots \| sigpub_n^2 \| c_1^2 \| \cdots \| c_n^2 \| sigpr_1^2 \| \cdots \| sigpr_n^2$. 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 $(id, d, m_1^{d-1} \| \cdots \| m_n^{d-1} \| sigpub_1^{d-1} \| \cdots \| sigpub_n^{d-1} \| c_1^{d-1} \| \cdots \| c_n^{d-1} \| sigpr_1^{d-1} \| \cdots \| sigpr_n^{d-1})$ and uses the proof of publishing as witness to decrypt input keys of the next round. Otherwise, if it is the same message as the one computed by E, E uses the proof of publishing of this message as a witness to decrypt the input keys of the next round. If it is not the same message as the one computed by E, E aborts the execution.
4. E outputs the concatenation of the outputs of the garbled circuits of the last round as the result.

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.

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, $n_{wires,j}^i$ denote the number of input wires of a garbled circuit of the next-message function for round i of party P_j.

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

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

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

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

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

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

Note that the dominant factor in both of the equations is $\sum_{i=2}^{n_{rounds}} \sum_{j=1}^{n} n^i_{wires,j}$. This term is precisely the combined communication- and (encrypted) state complexity of the original MPC protocol 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 eWEB 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 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 [8]. The construction is for deterministic functionalities where all the parties receive the same output, however it can be easily extended using standard techniques to randomized functionalities with individual outputs for different parties [AJLA+12]. For technical details behind the construction and the security proof we refer to Brakerski et al., and Mukherjee and Wichs.

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 [9].

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_i 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.

[8] Their scheme is secure when exactly all but one parties are corrupted. To transform it into a scheme that is secure against any number of corruptions, Brakerski et al. suggest to extend it by a protocol proposed by Mukherjee and Wichs (Section 6.2 in [MW16]) that relies on an 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.

[9] 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 $y = y_1 \ldots y_{l_{out}}$.

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.

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.

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 $\perp$.

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

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

Here, 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^d is the randomness used by P_i in round d, $\{m_k\}_{k=1}^d$ are (concatenations of) the messages broadcast by all parties participating in Protocol 4 in rounds 1 to d, and m_i^d is the message broadcast by P_i in round d. 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.

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. For each round $d = 1, \ldots, 3$
   (a) Let $m^d = m_1^{d-1}, \ldots, m_n^{d-1}$.
   (b) Compute $\mathsf{NextMessage}_d(x_z, r_z, \{m^k\}_{k=1}^d, c_z^{d-1}) = (m_z^d, c_z^d)$.
   (c) Broadcast m_z^d.
   (d) 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}) to prove that the following holds:

$$\mathsf{NextMessage}_d(x_z, r_z, \{m^k\}_{k=1}^d, c_z^{d-1}) = m_z^d.$$

      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}) such that the following holds:

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

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

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 model which is secure against fully malicious adversaries corrupting up to $t < N$ parties. Its communication and state complexity depend only on security parameters, number of parties, and input and output sized. In particular, the complexity is independent of the function f.*

Correctness Correctness of the overall construction follows directly from the completeness of Kilian's scheme [Kil92] as well as the correctness of the protocol of Brakerski et al. [BHP17].

Security We outline why this construction is secure. Intuitively, in order to prove security we construct the simulator S as follows: 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 whenever a proof is invalid. Note that for the correctly chosen PCP, Kilian's construction is extractable, and thus there exists an extractor Ext. S uses Ext to retrieve the witness (x, r) used by the adversary in each valid proof. Finally, S uses the simulator S_{sm} of the semi-malicious scheme (writing witnesses (x, r) extracted by Ext on the adversary's witness tape) to simulate the execution of the underlying semi-malicious construction.

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 [10].

Thus, we have shown that the communication- and state complexity of our MPC construction that is secure against fully malicious adversaries with arbitrary number of corruptions 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 main 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:

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

As we pointed out in Section 4, the term $\sum_{i=2}^{n_{rounds}} \sum_{j=1}^{n} n^i_{wires,j}$ is precisely the combined communication- and (encrypted) state complexity of the underlying MPC protocol 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

[10] 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.

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.

Apart from the CSaR store- and release requests the only other data that is contributing to the communication complexity of the overall construction is the data that is being posted on the bulletin board:

- messages (as well as signatures) on these of the first round – MPC contributors are tasked with posting these on the bulletin board.
- outputs of the garbled circuits – evaluators are tasked with posting these on the bulletin board.

The outputs of the garbled circuits consist of the messages exchanged by the parties in π, the signatures on these messages, the encrypted state information, and the signatures on the encrypted state. Thus, the size of this data depends only on the combined communication- and (encrypted) state complexity of the underlying MPC protocol π. When using Protocol 6 as the underlying protocol π, the size of this data is independent of the function which is being computed.

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 communication complexity 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).
- Whenever a message posted by the evaluator is of an invalid length or missing a valid signature, the miners use the garbled circuits and wire keys of the current round (that were previously published on the bullet board) to check whether the message posted 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 parties to prove the correctness of the computation. The relation that the miners then check 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 published values of the CSaR participants 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 miners release 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), 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 the evaluator 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 the evaluator's public key.

The definition of the *authoritative* message for this construction is a bit different to account for the fact that the signatures 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 $(m_1^d \| \cdots \| m_n^d \| c_1^d \| \cdots \| c_n^d \| sig_1^d \| \cdots \| sig_n^d)$.

– each m_j^d has correct length, and each sig_j^d is a valid signature of P_k on the tuple (id, d, j, m_j^d, c_j^d), with the following exceptions allowed:
  1. if a required message from some party P_j is missing, the evaluator must prove that P_j failed to post some of the messages needed for the computation 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.
  2. if the signature of some party P_j is invalid, or m_j^d (or c_j^d) has invalid length, the evaluator must prove that the output of the garbled circuit posted by P_j in the previous round is indeed what the evaluator claims this output to be. In this case, wire keys for the default value $\perp$ are released as wire keys corresponding to that messages.
– This is the first published message that satisfies the requirements above.

Same 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.
– This is the first published message that satisfies the requirements above.

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 [11].

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.

[11] This time we additionally prove that the guaranteed output delivery property holds for our construction.

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.

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.

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.