Gage MPC: Bypassing Residual Function Leakage for Non-Interactive MPC

Ghada Almashaqbeh[1], Fabrice Benhamouda[2], Seungwook Han[3], Daniel Jaroslawicz[3], Tal Malkin[3], Alex Nicita[3], Tal Rabin[4], Abhishek Shah[3], and Eran Tromer[5]

[1] University of Connecticut, ghada.almashaqbeh@uconn.edu
[2] Algorand Foundation, fabrice.benhamouda@gmail.com
[3] Columbia University, {sh3264,dj2468,tm2118,a.nicita,as5258}@columbia.edu
[4] University of Pennsylvania and Algorand Foundation, talr@seas.upenn.edu.
[5] Columbia University and Tel-Aviv University, et2555@columbia.edu

Abstract. Existing models for non-interactive MPC cannot provide full privacy for inputs, because they inherently leak the residual function (i.e., the output of the function on the honest parties' input together with all possible values of the adversarial inputs). For example, in any non-interactive sealed-bid auction, the last bidder can figure out what was the highest previous bid.

We present a new MPC model which avoids this privacy leak. To achieve this, we utilize a blockchain in a novel way, incorporating smart contracts and arbitrary parties that can be incentivized to perform computation ("bounty hunters," akin to miners). Security is maintained under a monetary assumption about the parties: an honest party can temporarily supply a recoverable collateral of value higher than the computational cost an adversary can expend.

We thus construct non-interactive MPC protocols with strong security guarantees (full security, no residual leakage) in the short term. Over time, as the adversary can invest more and more computational resources, the security guarantee decays. Thus, our model, which we call Gage MPC, is suitable for secure computation with limited-time secrecy, such as auctions.

A key ingredient in our protocols is a primitive we call "Gage Time Capsules" (GaTC): a time capsule that allows a party to commit to a value that others are able to reveal but only at a designated computational cost. A GaTC allows a party to commit to a value together with a monetary collateral. If the original party properly opens the GaTC, it can recover the collateral. Otherwise, the collateral is used to incentivize bounty hunters to open the GaTC. This primitive is used to ensure completion of Gage MPC protocols on the desired inputs.

As a requisite tool (of independent interest), we present a generalization of garbled circuit that are more robust: they can tolerate exposure of extra input labels. This is in contrast to Yao's garbled circuits, whose secrecy breaks down if even a single extra label is exposed.

Finally, we present a proof-of-concept implementation of a special case of our construction, yielding an auction functionality over an Ethereum-like blockchain.

1 Introduction

Secure multiparty computation (MPC) is a fundamental area in cryptography, with a rich body of work developed since the first papers in the 80's [12,21,35,52,59]. The setting involves n parties, each holding a private input, who wish to compute a function on their inputs in a manner that reveals only the output, and preserves the privacy of the inputs.

Interaction in MPC. The question of availability and required interaction among the parties in MPC protocols has been extensively studied. Most of the literature on secure computation requires all parties to remain online throughout the computation, engaging in interactive communication with each other. This requirement is problematic in many settings, where the parties are not all available for interaction at the same time (e.g., due to geographic or power constraints), and may not even be a priori aware of all other parties.

Thus, an important goal is to reduce interaction and online coordination in secure computation. Ideally, we envision the following setting for non-interactive MPC (NIMPC). When a party is available it carries out some local computation based on its input and any needed auxiliary information, and then it posts its result to some public bulletin board. Once all the parties are done, an output-producing party combines the information from the public repository and computes the output of the function while maintaining security, i.e., no information on the inputs is leaked beyond the output of the function.[6] However, it is known that this privacy-preserving ideal is impossible: leakage of the *residual function* [39] is inherent. Specifically, an adversary controlling the output-producing party and some of the computing parties can always repeatedly apply the legitimate protocol on any desired inputs provided by the colluding parties, to compute the function on more than a single input (see e.g., [10, 39]).

The works of [10, 39] further prove that, beyond the leakage of the residual function, there also needs to be some additional setup assumption in order to achieve Non-Interactive MPC. The results of [10, 31] work in the semi-honest model and assume some pre-dealt correlated randomness that is given to the parties. The works of [36, 39] rely on the existence of a PKI and assume the availability of an output-computing party that is online at all times and can be viewed as a "coordinator" among the parties. Each party engages in an interactive computation with the output-computing party, but the parties do not need to interact with each other. These papers provide solutions for restricted classes of functions. In [38] a solution is presented for all functions, at the expense of making a much stronger assumption, namely indistinguishability obfuscation, and a PKI. However, when collusions occur, all these results suffer from the unavoidable leakage of the residual function.

MPC and Blockchain. In 2008, Nakamoto proposed Bitcoin [47] as the first decentralized cryptocurrency. The core of Bitcoin is an append-only ledger, called blockchain, maintained by a consensus protocol. Transactions are combined into blocks. For a block to be added to the blockchain, a proof of work is used: parties called miners need to solve a cryptographic puzzle. This process is called mining and successful miners are automatically rewarded using the BTC asset, whose issuance and ownership are managed by the blockchain itself. It soon became clear that the consensus protocol can be extended to not only record transactions, but also store arbitrary state and enforce properties about this state and related transactions, through what became known as smart contracts. This unearthed the potential of combining blockchain technology with MPC, and the interaction between these two designs has evolved over several steps.

Gen I. Utilizing the blockchain to provide an implementation for the broadcast channel required for many MPC protocols (e.g., via `OP_RETURN` data).
Gen II. Incorporating payments into MPC protocols.
Gen III. *This work.* Incorporating smart contracts and the miners as active participants in the MPC.

Gen II started with such results as [7, 14, 43] that introduced monetary compensation and incentives in order to go beyond existing lower bounds in MPC. In particular, they address the fairness impossibility result of [24], which states that in a two-party setting it cannot be avoided that one party learns the output of the protocol and aborts prior to the other party learning the output. These papers present solutions which require each party to commit to a collateral via the payment capabilities of the blockchain. In case one party aborts, their collateral can be used to provide financial compensation for the party that did not receive the output. Thus, the collateral is used to incentivize the party to provide the output, and not abort. Note that while in some scenarios this incentive may be sufficient in order to achieve fairness, in fact the solution itself does not *guarantee* that this happens. It may be the case that despite facing a monetary loss, the party that learns the output might think that it is beneficial to quit the computation and prevent the other party from learning the output. This could happen due to e.g., failure, irrationality, or large external incentives to abort.

Our Work: Gage MPC (Gen III).[7] In this paper, we propose a new model and constructions of non-interactive MPC for any function, without the privacy-violating leakage of the residual function, and with security against semi-honest adversaries.[8] Thus, we circumnavigate the aforementioned impossibility.

Our model, which we call the *Gage MPC model*, assumes a monetary mechanism and a corresponding assumption that enables our solution. There is a party, which we refer to as *party zero*, that sets up the

[6] Due to the new monetary-incentivized model we consider, our work realizes a slightly modified version of NIMPC that we define later.

[7] Gage is an archaic word that means: a valued object deposited as a guarantee of good faith.

[8] We assume a semi-honest adversary only for the first step of the protocol: parties are required to post honestly-generated messages on the blockchain in the first step of the protocol. Later they can arbitrarily misbehave. Furthermore this restriction can be lifted using additional zero-knowledge proofs.

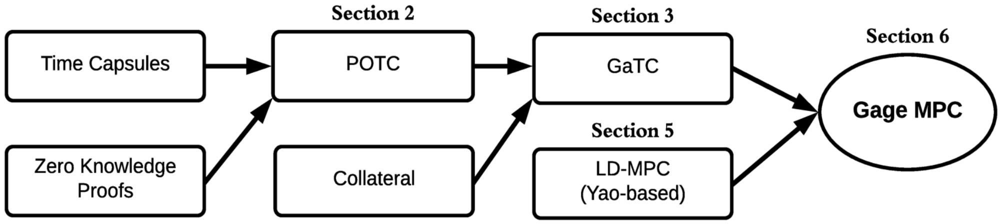

Fig. 1. Gage MPC design.

computations, and puts down some collateral. At a later stage, party zero can come back to provide some final message(s) that allow for the computation of the output. If this last step is completed, party zero can recover the collateral. We call this path of operation the *nominal* opening, which reflects the behavior of an honest party zero. If party zero fails to provide the messages then the collateral is used as payment to *bounty hunters* who will complete the execution via an expensive computation. (These might be, but don't have to be, the same as the blockchain miners.) We call this the *bounty* opening. The bounty opening does not depend on any a priori determined miner (or set of miners) and thus, as long as there exist parties wishing to be paid for computation, the bounty opening will complete successfully. This means that under no circumstances can party zero prevent the computation from being completed and the output from being exposed. This is in contrast to the Gen II MPC solutions [7, 14, 43], where the party also puts down a collateral, but forgoing the collateral enables to prevent the exposure of the output.

This bounty opening path matches the desired NIMPC setup outlined above. Each party posts a message and then the output computing party (in this case, the bounty hunters) will open the commitments and evaluate the output (which is announced on the blockchain). On the other hand, the nominal path (and Gage MPC 2-party protocol presented in Section 6) resembles a different flavor of NIMPC. That is, party zero will come for a second round to open the commitments, and so allow computing the output, to avoid losing the collateral. Nonetheless, party zero is not required to stay online all the time since the parties are posting their messages on the blockchain. Also, the rest of the participants perform a single round of interaction as before.

We rely on the underlying blockchain to ensure a consistent consensus view of these events, as well as delivery of commitment openings messages within bounded time (see below).

The collateral amount, along with the computational difficulty of the bounty opening, are set according to the difficulty that party zero wishes to create for adversaries interested in computing the (residual) function on additional inputs. To make such additional computations very expensive, party zero will set a high computational difficulty, and post a corresponding large collateral to compensate bounty hunters for their potential effort (in the case where party zero later aborts and bounty opening become necessary). This collateral is awarded only for the first opening; an adversary who tries to evaluate the (residual) function on additional inputs would have to perform the expensive computation and pay its cost without being compensated by the collateral.

Crucially, party zero can always recover the collateral via the nominal path, and in that way not lose it. This implies that the collateral can in fact be much higher than the amount of money party zero has, as party zero can for example take a short-term loan.[9] In contrast, for the adversary to break security, the collateral amount is the *actual* cost that needs to be spent for computing the function, which can be prohibitively high. This lends to the introduction of a new type of *monetary assumption*:

> *An honest party can put down a temporary collateral of value much higher than what an adversary can expend on computation.*

In our theorems we will prove that as long as the adversary does not spend (significantly) more than some amount related to the collateral, then it will not learn any additional information.

On Circumventing the Lower Bounds. The aforementioned impossibility, saying the leakage of the residual function is inherent in the standard model, is described for parties that all execute the same protocol.

[9] The main cost for the honest party is the time value of money. E.g., with a 10% APY loan, and a collateral locked for 3 days, the cost of a $1K collateral is less than $1.

Yet, it extends to our setting where there exists a special party, party zero, that speaks first.[10] The necessity of correlated randomness likewise persists.[11]

Our constructions avoid this by creating a setting where to compute the function on any set of inputs requires considerable computational effort. This effort effectively disarms the adversary from being able to compute the residual function, thus eliminating its leakage and circumventing the lower bound. Furthermore, we eliminate setup assumptions such as pre-shared correlated randomness (or PKI), or the need for a dedicated party to be available throughout to ensure the resolution of the computation. Our only requirements are the existence of the blockchain and availability of bounty hunters (i.e., parties willing to perform computation for a reward).

The assumption that party zero makes about the adversary's financial abilities might not hold indefinitely. The model allows for a highly incentivized, wealthy, and patient adversary to attack the computation by outspending the collateral amount, and revealing additional information about the function. Thus, there are two points to note. First, caution should be taken when making assumptions about the finances of the adversary. Second, the confidence we have in the assumption holding in relation to the adversary, due to the nature of the assumption, is weakened over time. This implies that the guarantee that there is no leakage of any additional information about the function (and in particular of the residual function) holds in the short term and decays over time.

Thus, one should consider under what circumstances it makes sense to use this model. A natural setting could be where the computation is of an ephemeral nature, and long after the output of the function has been announced, it is acceptable if additional information about the inputs is eventually revealed. Auction settings (as in our sample application) are often such.

1.1 Overview of our Design

Our design creates a sequence of primitives built on top of one another. It starts with Proof-of-Opening Time Capsules which are incorporated into Gage Time Capsules. Those are combined with Label-Driven MPC (LD-MPC) to finally construct our Gage MPC (Figure 1 shows how these primitives are combined in the Gage MPC construction). In the following sections and appendices we provide a description of our design and the flavor of each one of these constructions.

Blockchain Model. Our model relies on the following *blockchain model* properties. We assume that the blockchain provides an any-to-all broadcast channel, i.e., an ordered list of messages that is consistently visible to all. Any party can post a message that becomes visible to all parties within some bounded time. Furthermore, we assume a liveness property, i.e., messages cannot be blocked or delayed beyond some bounded duration (e.g., in Ethereum, it is assumed that the majority of the mining power is honest, and in particular will not censor transactions that carry adequate transaction fees; the requisite fees and queue size are publicly known). Functionally, we require the blockchain to support smart contracts, i.e., updating its consensus state according to (simple) programs we prescribe.

Gage Time Capsules. We build on a primitive we call *Gage Time Capsule* (GaTC). A GaTC is a commitment mechanism, that contrary to regular commitments, ensures that the committed value is exposed when needed.

We start with a simplified construction of GaTC to give the main ideas that underlie this primitive. GaTC bring time capsules [11, 54] to the blockchain. Recall, that time capsules enable a party to commit to a value in such a manner that another party, if it wishes, can brute-force open the commitment. GaTC combine, via a smart contract on the blockchain, a time capsule and a collateral that acts as an incentive mechanism for opening the time capsule. The GaTC will have two time periods. In the initial grace period, the creator of the GaTC will be able to open the commitment and retrieve the collateral. In the second period, if the collateral has not been retrieved, it will be used to pay bounty hunters who work to open the commitment. This use

[10] Consider the case of two parties P_0 and P_1. In the non-interactive setting, after P_0 speaks, an honest P_1 must be able to compute the output of the function. This implies that a faulty P_1 can in fact compute the function on any input that it wishes, exposing the residual function.

[11] Party zero could send correlated randomness to all the other parties, but only if it knows their identity in advance and has establishing secure channels or PKI with them.

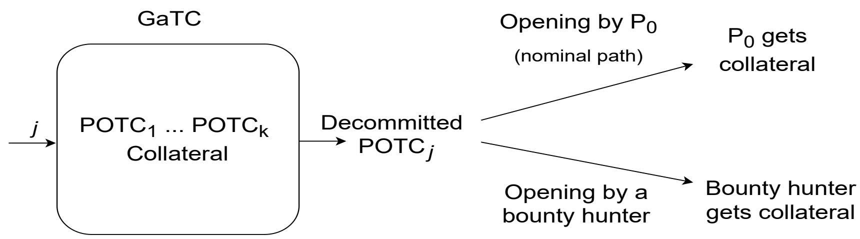

Fig. 2. A GaTC consists of a collection of POTCs with a single associated collateral. The collateral can be transferred to the party that opens the desired POTC, either P_0 through the nominal path or a bounty hunter. The index j indicates which POTC to open.

of the collateral is similar to the notion of incentives for proof of work [29]. Interestingly, this incentive mechanism ensures that given a large enough pool of bounty hunters, it is virtually guaranteed that the GaTC will be opened if and when needed. Note, that these bounty hunters are independent of the parties of the protocol and thus it does not matter which ones participate in the opening. In addition, these bounty hunters do not need to be related to the miners maintaining the blockchain in case of proof-of-work blockchains.

In contrast to designs of time capsules that require a minimum amount of sequential time to be opened, we are interested in time capsules that require some minimum amount of work to be opened. The work can be parallelized, and will be parallelized between the bounty hunters.

However, this naive construction of GaTC from time capsules suffers from the problem that if a bounty hunter finds an opening, other parties may steal it and claim the collateral in place of the bounty hunter who did the work. That is why instead of using plain time capsules, we introduce and use *Proof-of-Opening Time Capsules* (POTC). A POTC adds a decommitment value which requires time to compute. In opening the POTC the bounty hunter proves that it knows this value rather than exposing it in the clear. This proof can be tied to the bounty hunter in such a way that only the bounty hunter who produced the proof will be able to claim the collateral.

Another problem to consider, is the risk of an attacker posting malformed commitments that cannot be opened. This presents a denial of service (DoS) attack against bounty hunters who would waste their computation resources without reward. For simplicity, we assume a semi-honest adversary for the first step (the commitment posting). This assumption can be removed using a generic NIZK proof that P_0 generates to prove the well-formedness of the commitment.

The previous high-level definition of GaTC is actually still insufficient for our purpose: we need a more sophisticated primitive. A GaTC will bundle together a few POTCs and will accept as input an index (see Figure 2). It will only incentivize the opening of the POTC that relates to the given index. No incentives will be given for the opening of the other POTCs inside the GaTC.

Our monetary assumption is utilized in the design of GaTCs. The level of the collateral sets the complexity of opening a POTC. We assume that the complexity is set high enough that the adversary would not be interested in exerting the level of computational power that is required to open the number of additional POTCs within the GaTC so as to violate the security of the general protocol.

In Section 2 we propose an idealized instantiation of POTCs in the random oracle model and the generic group model. We use Fiat-Shamir in order to provide the proofs for the opening. In Section 3 we define our GaTC and provide an instantiation of those. Our GaTC can be used over any blockchain, including proof-of-stake based ones, as long as there are bounty hunters in the world who are interested in receiving the incentives.

Label-Driven MPC (LD-MPC). Our construction is based on a garbled circuit framework, i.e., there is a wire for each input and labels associated with that wire. In order to carry out the computation there is a need to know one label for each wire. The general idea would be that the labels of the wires would be committed to via GaTCs and if needed would be brute-force opened to enable the computation. However, Yao-like schemes leak information about the inputs of the parties even in the case that only one additional label, beyond those required for the computation, is exposed [10]. Given this leakage, basing our solution on Yao will provide only limited results. Thus, we introduce a generalization of garbled circuits, called Label-Driven MPC that is

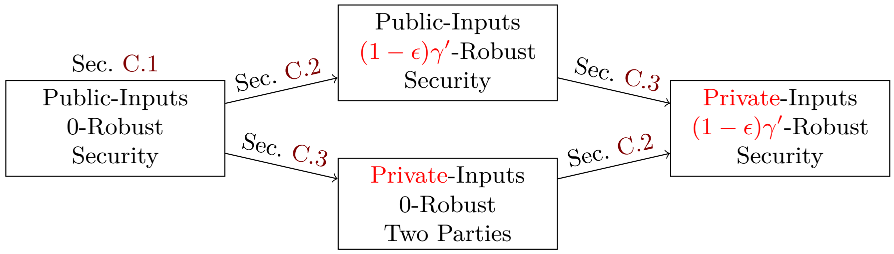

Fig. 3. Gage MPC. The fully private input versions are for two parties only.

robust in the face of exposure of additional labels. This is a powerful generalization that can find applications in other settings.

We obtain more robust garbled circuits by adding one level of indirection. In order to compute the desired Yao garbled circuit C we add a computation of a circuit C' on top of it. The output of the computation of the circuit C' will be the needed label for each input wire for the computation of the circuit C. It would seem that we have not done much, but the circuit C' will be designed in an innovative way by addressing two issues.

The first element in the creation of C' is that we choose an error correcting code that takes words of length γ and expands them to codewords of length $\gamma + \kappa$ (for a code of minimal distance $\kappa + 1$) where κ is the desired robustness level. The circuit C' will have $\gamma + \kappa$ inputs and the computation that it will execute is to test whether the input is a word in the code. If it is, it will output the labels for the circuit C. Observe that the error correcting code in a sense adds a buffer of security. To move from one legal codeword to another there is a need to change κ input wires in C'. Given our general idea that the labels of wires will be committed via GaTCs this expansion with error correcting codes provides the desired security level.

The second issue is that if C' is built as a Yao garbled circuit it may still suffer from the vulnerability of having a single additional wire exposed. However, C' is a simple linear computation, one that only needs to test if the input is in the code. Given that this is a linear computation we can rely on techniques from NIMPC [13] that provide robustness to linear computations.[12] Utilizing these techniques we can offer robustness to C' and thus in return provide robustness to the original computation.

Gage MPC. To achieve our final result of an MPC protocol that circumvents the lower bound of leaking the residual function we define the model of Gage Multiparty Computations and combine our new LD-MPC with our GaTC.

We consider a public function f, with parties $P_0, P_1 \ldots, P_N$ holding inputs $x_0, x_1, \ldots, x_N$ to f. The parties will compute $f(x_0, x_1, \ldots, x_N)$. We note that if we want the function itself to be private, we can set the public function f to be a universal circuit, and input the actual private function as the input x_0 of party zero, P_0.

We will combine different LD-MPC to achieve our solutions: a basic solution (based on Yao), and enhancements utilizing the more robust version of LD-MPC. We provide the following security guarantees.

0-robust security, public inputs. In this setting, the input of party zero, P_0, is the only private one, and other parties' inputs are all public. This solution utilizes a standard garbled circuit that creates two labels for each input wire. The POTCs of the two labels of a wire are combined into a $GaTC_i$ for each wire i by P_0. More specifically, $GaTC_i$ for wire i has $POTC_{i,0}$ for label $\mu_{i,0}$ (wire value $= 0$) and $POTC_{i,1}$ for $\mu_{i,1}$ (wire value $= 1$) in it. It incentivizes to open only one of the labels. For the sake of simplicity, in the introduction, we assume that each party P_i has a single input bit corresponding to the input wire i.

The nominal execution works as follows. Once the parties post their inputs this fixes the index that $GaTC_i$ should open: for input b of party P_i it should open the $POTC_{i,b}$. Party P_0 then comes and opens all

[12] The main result of [13] is the construction of $\kappa = O(1)$-robust NIMPC for polynomial-size circuits. Such NIMPC would provide $\kappa = O(1)$-robust LD-MPC. Unfortunately, this would not yield robustness for larger κ values. Instead, we use an intermediate result of [13] that gives a fully robust NIMPC for linear functions, and combine them with the ideas we described above.

the values and reveals the committed labels, which allows everybody to evaluate the garbled circuit and learn the output $y = f(x_0, \ldots, x_N)$. Party P_0 further retrieves its collateral.

In case P_0 misbehaves and does not reveal the labels, after the initial grace period, bounty hunters are incentivized to open the POTCs of those labels and receive the collateral as payment. Once all the needed POTCs are opened, the output y is computed the same way as in the nominal execution.

As stated, the underlying garbled circuit is not robust to the exposure of additional labels, and thus this computation inherits this lack of robustness. However, as long as the adversary does not exert the computational effort required to open any additional POTC, the resulting protocol achieves classical MPC security: the adversary learns no additional information but $y, x_1, \ldots, x_N$.

If the difficulty of opening a POTC is set high enough and if there are relatively few GaTCs (i.e., few input wires), the total collateral (of all the GaTCs) is not much larger than the difficulty of opening a single GaTC. In that case, it is reasonable to assume that the adversary cannot open any additional POTC and security holds.

However, when there are many input wires, this assumption becomes less likely. The adversary may have enough power to brute force one of the other labels on its own. To prevent this exposure we utilize our more robust LD-MPC.

κ-robust security, public inputs. We present a design that protects against an adversary exerting enough computational effort to open up to κ additional POTCs (i.e., learning κ additional garbled circuit labels). We use the κ-secure LD-MPC. We will want to relate the complexity of the work to the number γ of inputs to the function. If the adversary wants to compute the function on an additional input, we would like the adversary to have to work almost as much as it costs to compute the function on a single input. Thus, we set the parameters as follows. The length of the expanded input is $\gamma' = \gamma + \kappa$ implying that to compute the function on one input requires a number of POTCs (or labels) equals to γ'. We would like the robustness κ to equal $(1 - \epsilon)\gamma'$. This creates a $(1 - \epsilon)\gamma'$-robust Gage MPC with γ' wires, where $\varepsilon > 0$. In other words, this new protocol ensures security as long as the adversary can only exert enough computational effort to open a $1 - \epsilon$ fraction of the POTCs that bounty hunters would have to open in the bounty case. We remark that this is the best that can be achieved. Indeed, if the adversary can open as many POTCs as bounty hunters would have to open if P_0 misbehaves, then the adversary would be able to evaluate the function f on another set of inputs. This would break security (in other words, γ'-opening security is never achievable for a protocol with γ' wires).

We stress that the adversary who puts enough computational effort to open more than κ additional POTCs may learn more than the output of the function on one additional input. After exerting enough computation power, it may even learn the full function. We leave it as an open question whether there exists an efficient solution whose security degrades more gracefully.

0-robust security, private inputs, two parties. We provide a second, orthogonal transformation that provides privacy for all parties' inputs (rather than just P_0). We address this setting only for the case of *two*-party secure computation (and we discuss how this can be extended to the general multiparty case while pointing out potential practicality limitations). Thus, we consider parties P_0, P_1 holding private inputs x_0, x_1 respectively, trying to securely compute $f(x_0, x_1)$.[13] We can address this setting by combining a two-round two-party secure computation evaluating the desired function with a Gage MPC. This is done by transforming the second step of the evaluation of the two-round protocol into a Gage MPC protocol.

κ-robust security, private inputs, two parties. The two transformations described above can be combined, to achieve Gage two-party secure computation with secret inputs and κ-robust security. See Fig. 3.

1.2 Application: Private Decentralized Auctions

An important application, highlighting the power of Gage MPC, is on-chain trading. Auctions allowing parties to place bids for a published offer, and more generally exchanges allowing to place multiple bids and offers (orders) in an order book, are crucial components of economic markets.

[13] As mentioned above, using universal circuits this can also be used when P_0 holds a private function g and P_1 holds an input x and they compute $g(x)$.

The emergence of blockchain technology saw the introduction of many decentralized exchanges (or auctions), which are implemented using a blockchain and often trade assets whose ownership is represented on the blockchain. Goals and motivation in constructing such systems include: eliminating the need for trusted parties to deliver correct execution, privacy, or asset custody (credit risk); reducing costs, fees and onboarding barriers; avoiding censorship; and enabling integration with other blockchain-based systems, such as electronic commerce and decentralized finance instruments. Many such systems have been implemented [1–4, 6, 45, 51, 57], and the total trading volume in decentralized exchanges has recently exceeded US $20B per month [27].

The existing schemes that implement decentralized blockchain-based order-book exchanges follow two paradigms: *open orders books* where the orders are broadcast in plaintext, recorded on the blockchain, and then settled later (by the miners following consensus rules), or some weak notion of *hidden order books* (i.e., *dark pools*) to keep the offers and bids secret. An example of the latter is the commit-and-open paradigm: sending the order in the form of (hiding and biding) commitments to a smart contract, and later publicly opening all these commitments [6]. Another approach is to send orders in secret-shared form to a committee, which uses MPC to reveal just the outcome, while letting unexecuted orders remain hidden [5]. Both approaches have the drawback that they require the continued availability and participation of specific parties (the traders in the former; the MPC committee members in the latter), and if these parties become unavailable, the orders can never be executed. The requisite participation is incentivized by escrows, or "bonds", that are confiscated if those parties fail to operate correctly. They are, however, technically at liberty to abort and just accept the penalty. That is, they operate in the Gen II model.

Crucially, these approaches do not guarantee execution, they are not resilient to auxiliary incentives that may induce participants to abort even if this entails a penalty. For instance, a bidder may realize that the market price has jumped and their already-committed-to offer, if executed, would cause them enormous loss. Or an auction seller, who has committed to a reserve price, may later realize that revealing their low reserve price will harm them in some future transaction. Either of these, or an attacker trying to cause a denial-of-service disruption, may be willing to forego their bonds or bribe MPC parties to lose theirs.

An alternative, *automated market marker*, forgoes order books and implements the trading counterparty as a smart contract. All bids and offers are public, as is the trading algorithm itself (which moreover must be supplied with large liquidity pools).

A recent scheme, which (like us) targets the issue of guaranteed execution without requiring the parties to stay online, exploits time-lock puzzles to realize a sequential blockchain-based auction functionality [28]. It follows the commit-and-open paradigm; parties seal their bids in commitments, provide time-lock puzzles for the opening, then if the parties do not come later to open their commitments other miners will open the puzzles and execute the auction. Their goal is to build an auction-based proof-of-stake (PoS) mining algorithm to discourage hoarding currency.

Our Gage MPC construction enables auctions which have *guaranteed evaluation*, and moreover, are *privacy-preserving*. Unlike [28], which opens all inputs in the clear during the execution phase, our scheme reveals only whether or not the bid matched the offer. The offer price remains secret, and (if using the aforementioned private-input transformation) so does the bid. The privacy level can be calibrated by configuring the amount of computation required to force-open the puzzles, i.e., the bounty-hunting path, which is translated into monetary cost.

Such privacy was unnecessary for the narrow setting of [28], but it is clearly of interest in more general auction applications. Moreover, we provide a general auction functionality that can be used by any user to trade any asset represented on the blockchain.[14] Lastly, note that these auctions are merely a special case, and our construction extend to any (efficiently-computable) two-party functionality, including those that control asset flows in more complex ways. Thus, we solve a special case of the important problem of trustless privacy-preserving smart contracts.

Implementation of POTC and Simple Auctions. As a proof of concept, and to demonstrate the potential practicality of our constructions, we implemented a library that provides a generic interface for our

[14] It is unclear how the mining auctions of [28] would be thus generalized, when their incentives are specific to mining, i.e., the prospects of transaction fees and block rewards. For instance, without incentive for the nominal path, a party can post a bid and then disappear, leaving it to others to expensively open the commitment and execute the auction — with no penalty, thus enabling a DoS attack.

POTC operations, and evaluated its performance overhead for each operation. Next, we used the library to implement an instantiation of Gage MPC for a simple auction functionality, in our basic security setting (public-inputs, no additional opening). That is, a seller posts an offer with a secret reserve price, while buyers' offers are public. Our prototype implementation is fully integrated in Ethereum Virtual Machine (EVM). Our implementation is described in Section 7.

1.3 Related Work

Our work is related to several concepts found in the literature, including time capsules, Non-Interactive MPC, and the use of blockchains to circumvent some impossibility results and/or build new cryptographic primitives. In what follows, we provide a brief description of some of the works in each of these areas, while a more detailed discussion can be found in Appendix A.

The notion of time capsules was first introduced in [11], and is closely related to the notions of timed commitments [17] and time-lock puzzles [54]. These constructions are different from our use of time capsules. In particular, the constructions of [17, 54] need to ensure that the attacker cannot utilize parallelism in order to improve the run-time for the breaking protocol. Yet, we focus on the total amount of computation that the adversary needs to compute in order to break the time capsule, irrespective if this work is done in parallel or not. Thus, most applications that would need to enforce some minimal amount of work, with the fine-grained capability of configuring the amount of information leakage based on the computation power of the adversary, would need our modified time-capsules notion and construction. Other recent works [19, 46] introduced the notion of homomorphic time-lock puzzles, which allows to combine homomorphically time puzzles and then open a single puzzle containing the result of the computation. Homomorphic time-lock puzzles also allow to design decentralized auctions, however the known constructions either require indistinguishability obfuscation [46] or multi-key fully homomorphic encryption [19]. The auction-based mining proposed in [28], similar to our work, brings the notion of time-lock puzzles to the blockchain by having miners force open unopened puzzles. However, as discussed above, its focus is on the time needed for opening rather than the computational effort, and it does not incentivize the nominal path of puzzle opening, thereby leaving the miners vulnerable to a DoS attack.

Another related concept is verifiable delay functions [16, 30], in which a sequential function is evaluated over some input and takes at least t time steps to be completed, while verifying the correctness of the output is much more efficient. Similar to time-lock puzzles, VDFs are about ensuring a given bound on the computation delay is satisfied (and that parallelism is not effective in attacking the scheme) rather than the amount of computation. On the other hand, the notion of pricing functions [29, 48], where the proof-of-work mining is an example of such functions, are similar in spirit to our work. The goal is to utilize moderately hard functions to put a price on some actions (e.g., sending spam emails) in terms of amount of computation. Our work extends this model by involving explicit monetary rewards to enforce computation completion.

On the blockchain model and circumventing impossibility results, as mentioned before, several works targeted the fairness issue in MPC. The works in [7, 14] realize a financial-based notion of fairness in which the party that aborts after learning the output loses its penalty deposit to the honest players. This notion was formalized in [43] under what is called secure MPC with compensation. As opposed to our constructions, the output is not guaranteed: the honest players just get compensated if they do not receive the output. On the other hand, [23] does not rely on financial incentives and achieve full fairness utilizing a public bulletin board (can be instantiated using a blockchain). However, their constructions require either extractable witness encryption which has no known practical implementation (nor even theoretical constructions under standard assumptions) or the use of secure hardware like SGX. A more recent work [33] exploited the consensus protocols in blockchains, particularly that what matters is the honest power majority (whether it is computing power, or stake, etc.) rather than the number of parties to circumvent the $n/3$ lower bound of MPC with malicious parties when no private correlated randomness setup (e.g., a PKI) is used.

A related line of work used the blockchain model as an alternative to strong assumptions. In [37] it is used to avoid the trusted setup needed for non-interactive knowledge (NIZK) proof systems, [22] strengthened this model and allowed the use of global public blockchains.

For implementing new cryptographic primitives or functionalities in the blockchain model, [41] used a public bulletin board to enable a trusted execution environment (TEE) hosted by an untrusted computer, to create a secure state without requiring a persistent internal storage. In [44], a privacy preserving smart contract framework is proposed, which permits implementing MPC protocols on the blockchain. However, the proposed framework requires to trust a manager with the users' inputs. Our scheme does not introduce a privileged entity; it is fully distributed and decentralized. A stronger privacy notion appears in [18], where not only the user's input is protected, but also the executed functionality. Our scheme addresses both data privacy and function privacy (by garbling a universal circuit), but without requiring a privacy-preserving ledger as in [18].

2 Proof-of-Opening Time Capsules

In this section we present proof-of-opening time capsules (POTC), which enhance time capsules [11, 54] to add the feature that a party that opens the time capsule can prove in zero knowledge that it knows the opening (rather than exposing the committed value in the clear). We propose candidate constructions that satisfy our definition. The feature of proving the opening will be critical for our design of the new concept of GaTC and Gage MPC. Though we see the definition and proof of these time capsules as an important contribution of our paper, it is not the main one. We thus focus here on the aspects needed for the subsequent constructions, while (due to space limitation) formal security definitions, constructions, and proofs can be found in Appendix B.

2.1 Definition

We start by defining proof-of-opening time capsules. At a high level, POTC adds the following to the original time capsules [11, 54]. First, it allows the decommitment information to be different from the randomness used in generating the commitment (i.e, the capsule). As such, we introduce an additional algorithm to verify a decommitment. Second, it does not require the opening party to reveal the decommitment. This is needed since the simple method of revealing the decommitment does not work for our purposes; an attacker who observes this value being sent can block or front-run it, and present this opening as their own. By requiring a zero knowledge proof instead to prove opening, our approach disarms such a malicious party and protects the party who did the work to open a capsule.

In the definition below, λ is a standard cryptographic security parameter (in particular, a computation time of 2^λ is not feasible). On the other hand, λ^* is a parameter capturing the hardness of forced-opening. That is, 2^{λ^*} is the time it takes to recover the message and a decommitment from the commitment alone; it should be feasible (polynomial), but hard (λ^* is a measure of how costly this is).

Definition 2.1 (informal). *A* Proof-of-Opening Time Capsule *is defined by five polynomial-time algorithms that make use of a hash function H modeled as a random oracle:*

- *Commitment:* $(c, d) \leftarrow \mathsf{TC.Com}(1^\lambda, 1^{\lambda^*}, \mu)$ *takes as input the security parameter λ, the opening hardness parameter λ^*, a message $\mu \in \{0,1\}^{\text{poly}(\lambda)}$, and outputs a commitment or time capsule c and its associated decommitment d.*
- *Decommitment Verification:* $\mathsf{TC.DVer}(1^\lambda, c, \mu, d)$ *takes as input the security parameter λ, a commitment c, a message μ, a decommitment d and outputs 1 if the decommitment is valid with regards to c and μ; and outputs 0 otherwise.*
- *Forced Opening:* $(\mu, d) \leftarrow \mathsf{TC.ForceOpen}(c)$ *takes as input a commitment c, brute-forces its opening, and outputs the committed message μ and a valid decommitment d.*
- *Opening Proof:* $\pi \leftarrow \mathsf{TC.Prove}(c, \mu, d, \mathsf{tag})$ *generates a proof π, with tag tag, that c commits to μ using the witness d.*
- *Proof Verification:* $\mathsf{TC.PVer}(1^\lambda, c, \mu, \pi, \mathsf{tag})$ *takes as input the security parameter λ, a commitment c, a message μ, a proof π, a tag tag, and outputs 1 if the proof is valid with regards to c, μ, and tag; and outputs 0 otherwise.*

We require the following properties to be satisfied:

Perfect correctness. $\mathsf{TC.DVer}$ *returns* 1 *for honestly generated commitments and decommitments.* $\mathsf{TC.PVer}$ *returns* 1 *for honestly generated commitments and proofs. Furthermore,* $\mathsf{TC.ForceOpen}$ *makes about* 2^{λ^*} *evaluations of H.*

Binding and Soundness. *It is not possible to find a commitment c for which there exists two valid decommitments* d, d' *(resp., two valid proofs) for two different messages* $\mu \neq \mu'$.

Hiding and Simulatability. *The original definition of time capsule stated that it should take about* 2^{λ^*} *evaluations of* H *to learn any information about the committed message* μ *of a commitment* c. *This notion is insufficient for our purpose, as we may use many time capsules, in which case the adversary may be able to open some of them. What we require is that if the adversary makes significantly fewer than* $\kappa \cdot 2^{\lambda^*}$ *evaluations of* H, *it should not be able to learn more than* κ *of the values in the capsules. This property is surprisingly difficult to formalize: our formal definition is simulation-based and uses ideas from trapdoor commitments [32].*

Non-Malleability. *Generating opening proofs for a new tag* tag *(for which no such proof was generated) for* κ *different commitments requires to make around* $\kappa \cdot 2^{\lambda^*}$ *evaluations of* H. *In particular, seeing proofs-of-opening for a commitment* c *and a tag* tag' *does not help in generating a proof-of-opening for the same commitment and a different tag* $\mathsf{tag} \neq \mathsf{tag}'$. *This property is used to ensure that parties cannot "steal" proof-of-opening from another party.*

2.2 Construction

We have the following theorem.

Theorem 2.2. *There exists a POTC in the random oracle and generic group model.*[15]

In this section, we present a simplified construction without proof-of-opening. Adding proof-of-opening could be done generically using simulation-extractable NIZK, but this would be highly inefficient. Instead, in Appendix B.3, we show how to transform the time capsule below to make it more algebraic and uses a Fiat-Shamir proof.

Our time capsule construction is based on the original construction in [11] and references within. In this original construction, to commit to a message μ, the committer first selects a low-entropy seed $s \xleftarrow{R} \{0,1\}^{\lambda^*}$, then hashes this seed and uses this hash as a one-time pad to "encrypt" μ as $c_2 \coloneqq H_2(s) \mathsf{xor} \mu$ (where H_2 is a hash function). It also includes in the commitment a hash of the seed, namely, $c_1 \coloneqq H_1(s)$ where H_1 is another hash function, that is used to verify if a seed is the valid one. The commitment is $c \coloneqq (c_1, c_2)$.[16]

Intuitively, to force open a commitment, we need to enumerate the possible seeds s, hash them, and check which one corresponds to c_1. This requires at most 2^{λ^*} hash evaluations of H_1, and on average $2^{\lambda^*}/2$ hash evaluations of H_1.

Unfortunately, such a time-capsule does not satisfy our definition of security. An adversary making even a single hash evaluation of H_1 and H_2 may stumble upon the correct seed s, with probability $1/2^{\lambda^*}$, which is not negligible. This makes such a commitment construction difficult, if not impossible, to use as a building block of any cryptographic protocol.

We fix the above construction as follows. Instead of using a single seed $s \in \{0,1\}^{\lambda^*}$, we use k seeds $s_1, \ldots, s_k \in \{0,1\}^{\nu_s}$, where $\nu_s = \lambda^* - \lfloor \log_2 k \rfloor$. We commit to each seed individually by computing $c_{1,i} = H_1(i \| s_i)$, and then encrypt the message μ as $c_2 \coloneqq H_2(1 \| s_1) \mathsf{xor} \cdots \mathsf{xor}\, H_2(k \| s_k) \mathsf{xor}\, \mu$. Thus, force opening a commitment takes about $2^{\nu_s} \cdot k \approx 2^{\lambda^*}$ hash evaluations (by trying every s_i).

However, the main difference now is that if the adversary can only make significantly fewer than 2^{λ^*} hash evaluations, the commitment will remain statistically hiding. For example, if the adversary makes at most $k-1$ hash evaluations, the message μ is perfectly hidden since the adversary will be missing one of the one-time pad masks $H_2(i \| s_i)$. This argument can be extended to an adversary making many more than k hash evaluations using a careful probability analysis and tail bounds.

To provide more intuition of why this is true, let us consider two cases which require around the same number of hash evaluations to brute force, which is about 2^{15}:

Case 1. A single 15-bit seed. The probability that the adversary succeeds with q hash evaluations is around $q/2^{15} = q/32768$. This is never negligible.

---

[15] The construction uses both a hash function modeled as a random oracle and a cyclic group modeled as a generic group.

[16] For the sake of simplicity, we do not include any salts in this overview, and assume that a single commitment is made. This is already sufficient to explain the issue we want to highlight.

Case 2. Eight 12-bit seeds. Let us simplify the probability analysis by allowing the adversary to make q hash evaluations per seed, instead of q hash evaluations in total (this only makes the adversary stronger). For each seed, the adversary will hash the correct seed value with probability around $q/2^{12}$, which is not negligible. However, the adversary must guess correctly each seed value, and each of these events are independent, so the adversary's success probability is about $q^8/(2^{12})^8 = q^8/2^{96}$, which is negligible in practice (by which we mean, less than 2^{-80}).

We give some concrete parameters to show that the construction is efficient in Table 1 on page 27. Even for 128 bits of security where the adversary should not learn any committed message ($\kappa = 0$), if the honest parties are assumed to make about 2^{40} hash evaluation to force open a commitment while the adversary is restricted to 2^{30} hash evaluations, then only $k = 15$ seeds are required. For weaker notions of security (where the adversary is allowed to open $\kappa > 0$ messages), the number of used seeds can be further decreased.

3 Gage Time Capsules

3.1 Definition

We proceed to introduce the notion of a *Gage Time Capsule (GaTC)*, which guarantees that one of several posted commitments is opened according to some rule, even if the original committer does not cooperate in the opening.

A GaTC allows a *committer* party to publish several commitments to values, and designate a *controller* party that will choose which commitment will be opened.[17] The committer also posts a collateral. It is guaranteed that once the controller has made the choice of which commitment to open, the commitment will indeed be opened shortly afterwards, one way or another. Either the committer will open the designated commitment within a prescribed time window, and thereby reclaim the posted collateral; or someone (an arbitrary *bounty hunter*) will force-open the commitment, and receive the collateral as profitable compensation for the requisite computational effort.

Abstractly, the GaTC functionality has the following interface. Given a broadcast channel, a consensus view of public events' ordering, an approximate global clock, and means to programmatically transfer assets (all of these will be realized, below, by an underlying blockchain):

- Creation: The committer party invokes $\mathsf{GaTC.Create}(1^\lambda, 1^{\lambda^*}, \boldsymbol{\mu}, \mathsf{ctrl}, \mathsf{deposit})$ (where λ is the security parameter and λ^* is the opening hardness parameter as defined in POTC) to commit to a vector of messages $\boldsymbol{\mu} = (\mu_0, \ldots, \mu_{L-1})$ where $\mu_i \in \{0,1\}^{\nu_\mu}$, and places an associated *collateral* of a prescribed monetary value, represented by deposit. Initially, this collateral is locked up and inaccessible. The existence of this new GaTC, and L, become public, but not the content of $\boldsymbol{\mu}$. The committer party also designates the party represented by ctrl as the *controller* of this GaTC.
- Request opening: The controller party invokes $\mathsf{GaTC.RequestOpen}(\sigma)$ once, where $\sigma \in \mathbb{Z}_L$, to choose which commitment should be opened. This index σ becomes public. From this event, we measure a prescribed time duration to a deadline T.
- Nominal opening: Subsequently, the committer may invoke GaTC.NominalOpen(), which causes the value μ_σ to be publicly released. If this is the first time that the committer called GaTC.NominalOpen, and moreover either the deadline T has not yet passed or no call to GaTC.ForceOpen has been completed yet, then the collateral is returned to the committer.[18]
- Bounty opening: GaTC.ForceOpen() can be invoked by any party (serving as a bounty hunter). This will cause that party to perform a computationally expensive process, and eventually publish the value μ_σ. If the deadline T has passed, and moreover this is the first invocation of GaTC.NominalOpen or GaTC.ForceOpen that completed, then the the collateral is transferred to the bounty hunter.
- Querying the result: GaTC.GetResult() may be invoked by anyone, after GaTC.RequestOpen has been invoked. This returns μ_σ if GaTC.NominalOpen or GaTC.ForceOpen have already been invoked and completed; otherwise it returns $\perp$.

[17] In our applications, the designated controller will be an existing smart contract, which commits to the procedure by which it will be decided which value to open.

[18] We assume a global clock; in the realization, this will be defined by the length of a blockchain's consensus view, measured in blocks.

Implicit parameters. For brevity, we let the message length be constant, and thus omit it from this description. We also omit specification of the collateral amount and of the time window until the committer's deadline; these need to be calibrated to the properties of the application and the underlying asset and blockchain. In particular, the collateral amount should suffice to compensate and incentivize the computation of $\mathsf{TC.ForceOpen}$ (from the POTC) within $\mathsf{GaTC.ForceOpen}$.

The time window for the deadline should be long enough to allow the committer party to open the time capsules and recover its collateral. It is fixed independent of how long it takes to compute the opening by the bounty hunters. It only depends on how long (in number of blocks) an adversary can censor a transaction (i.e., prevent it from being added to the blockchain), assuring that the committer party can post the opening.

3.2 Realization on a Blockchain

Model. We realize the GaTC functionality using a POTC scheme TC along with a blockchain that supports smart contracts and assets with monetary value. Specifically, we assume an append-only ledger, composed of blocks each of which records an ordered list of transactions; a consensus algorithm that provides all parties with a consistent consensus view of this ledger; and a permissionless censorship-free protocol for appending transactions to this ledger. We assume that the transaction syntax and semantics support smart contracts, i.e., user-defined stateful interactive programs executed by the blockchain's consensus rules (e.g., as implemented in Ethereum). We also assume that transactions can represent ownership and transfers of assets with monetary value, such as a native cryptocurrency (e.g., ETH) or other tokens (e.g., ERC-20 or ERC-721).

We consider time in terms of rounds, where a round is the time needed to append (i.e., mine) a block on the blockchain. Hence, a capsule's grace period, i.e., the time during which P_0 can use the nominal path, is defind in number of rounds. We assume a secure blockchain that satisfies the liveness and persistence security properties [34, 50]. In particular, censorship (blocking publication of a valid transaction for unbounded time) is assumed infeasible due to the liveness property, as discussed in Section 1.1.

Construction. A GaTC coordination smart contract, designated GaTCContract, implements the functionality of storing the state of POTCs, implementing the requisite rules (e.g., checking the proofs of opening), and handling the collateral (i.e., taking custody of it and paying this collateral to the correct party when the conditions are fulfilled). The POTC scheme is used with suitable security parameter λ and opening hardness parameter λ^*, and assets of correspondingly suitable value are used as a collateral and for compensating bounty hunters.

Thus, the GaTC is realized as a "decentralized app," where each of the GaTC's algorithms consists of a portion executed by a party and/or a portion executed by GaTCContract, as follows (in any of the steps below, which corresponds to some message asking to perform a specific operation, if any of the checks fail the contract will simply ignore the calling message).[19]

- The committer invokes $\mathsf{GaTC.Create}(1^\lambda, 1^{\lambda^*}, \boldsymbol{\mu}, \mathsf{ctrl}, \mathsf{deposit})$ where $\boldsymbol{\mu} = (\mu_0, \ldots, \mu_{L-1})$ and $\mathsf{deposit}$ grants control over the assets to be placed as collateral (a spending key for a wallet with adequate balance):
  1. The committer runs $(c_i, d_i) \leftarrow \mathsf{TC.Com}(1^\lambda, 1^{\lambda^*}, \mu_i)$ for $i \in \mathbb{Z}_L$ where λ is the security parameter and λ^* is the opening hardness parameter (see Section 2.1).
  2. The committer calls a method of GaTCContract that, given $\boldsymbol{c} = \{c_i\}_{i \in \mathbb{Z}_L}$, $\mathsf{deposit}$, ctrl and (implicitly) the caller address cmtr:[20]
     (a) initializes GaTCContract's state with the following values: $\boldsymbol{c}$,[21] the controller's address ctrl, the committer's address cmtr, $\sigma = \perp$ and $\mu = \perp$;
     (b) moves ownership of the collateral to GaTCContract using $\mathsf{deposit}$.
  3. The committer remembers $\{(c_i, \mu_i, d_i)\}_{i \in \mathbb{Z}_L}$ for later.

[19] For simplicity, we assume that GaTCContract is created anew for every instance of the GaTC. It can be trivially extended to serve multiple instances.

[20] We identify parties with their account address on the blockchain, and assume they control the assets in this account as well as the ability to send messages that are identified as sent from this account, as in Ethereum.

[21] Since persistent storage in smart contracts is expensive in platforms such as Ethereum, costs can be reduced by storing only the Merkle tree root of the vector $\boldsymbol{c}$, and then including Merkle authentication paths in later calls to the smart contracts. We omit this optimization for simplicity.

- The controller invokes $\mathsf{GaTC.RequestOpen}(\sigma)$, where $\sigma \in \mathbb{Z}_L$:
  1. The controller calls a method of GaTCContract that, given σ and the caller:
     (a) verifies that the caller is ctrl, and that the stored σ is $\perp$;
     (b) stores the new σ;
     (c) stores the deadline T, expressed as a block number and computed by adding a constant to the current block number.
- The committer invokes $\mathsf{GaTC.NominalOpen}()$:
  1. The committer retrieves σ from GaTCContract, and aborts if it is $\perp$.
  2. The committer runs $\pi \leftarrow \mathsf{TC.Prove}(c_\sigma, \mu_\sigma, d_\sigma, \mathsf{cmtr})$ where cmtr is the committer's account address.
  3. The committer calls a method of GaTCContract that, given $\bar{\mu} = \mu_\sigma$ and π:
     (a) verifies that the stored μ is $\perp$, and the stored σ is not $\perp$;
     (b) verifies that $\mathsf{TC.PVer}(1^\lambda, c_\sigma, \bar{\mu}, \pi, \mathsf{cmtr}) = 1$;
     (c) updates the stored μ to $\bar{\mu}$;
     (d) transfers the deposit to cmtr.
- A bounty hunter invokes $\mathsf{GaTC.ForceOpen}()$:
  1. The bounty hunter retrieves σ and c_σ from GaTCContract (and aborts if σ is $\perp$).
  2. The bounty hunter runs $(\bar{\mu}, \bar{d}) \leftarrow \mathsf{TC.ForceOpen}(c_\sigma)$.
  3. The bounty hunter runs $\pi \leftarrow \mathsf{TC.Prove}(c_\sigma, \bar{\mu}, \bar{d}, \mathsf{hntr})$ where hntr is the bounty hunter's account address.
  4. The bounty hunter calls a method of GaTCContract that, given $\bar{\mu}$, π and hntr:
     (a) verifies that the stored μ is $\perp$, and the stored σ is not $\perp$;
     (b) verifies the current time (block number) is later than T;
     (c) verifies that $\mathsf{TC.PVer}(1^\lambda, c_\sigma, \bar{\mu}, \pi, \mathsf{hntr}) = 1$;
     (d) updates the stored μ to $\bar{\mu}$;
     (e) transfers the deposit to hntr.
- $\mathsf{GaTC.GetResult}()$: returns the μ as stored in GaTCContract (possibly $\perp$).

Note that within the smart contract GaTCContract, the only cryptographic operation is calling $\mathsf{TC.PVer}$ to verify proofs; the rest is simple logic, storage and retrieval of (publicly known) state, and asset transfers.

Security. The above construction realizes the definition of GaTC from Section 3.1. For brevity we omit a full formal treatment, but note the following nuances:

By assumption, the underlying blockchain serializes on-chain events, including calls to the smart contract. This avoids *time-of-check-to-time-of-use* vulnerabilities in the tests done by the smart contracts.

There is the concern of *front-running attacks* by bounty hunters (i.e., stealing someone else's opening proofs while they're still in the process of being published to the blockchain, and claiming the collateral for themselves). This is prevented using the tags associated with each proof, which convey the identity of the prover (i.e., the initiator or the bounty hunter who ran $\mathsf{TC.ForceOpen}$). The non-malleability property of the POTC guarantees that the tag associated with a proof cannot be changed by others.

If a commitment opening transaction, sent by the committer or bounty hunter, is blocked for sufficiently long, then the collateral may be collected by another bounty hunter. However, as mentioned under the paragraph "Model" above, prolonged censorship is assumed infeasible due to the liveliness properties of the underlying blockchain.

4 Sample GaTC Usage: Auctions

As a tutorial example of using GaTCs in a higher-level application, we show a simple protocol that implements fully-decentralized semi-private reserve-price auctions over the blockchain, when the set of possible prices is small enough that costs (storage and computation) can be linear in the size of this set.[22] Using Gage Time Capsule, we ensure that the auction is completed even if the seller tries to abort it. Section 6 below extends

[22] While we use auctions for concreteness, this approach naturally extends to evaluating any function with a small truth table, where the function is secret and the input is public. Complexity is linear in the size of the truth table.

this to general MPC (and in particular auctions with a large range of prices), and Section 7 describes our implementation.

One-shot reserve-price auction. Consider an auction for trading digital assets (e.g., selling some Ethereum-based token for some amount of Ether). A seller P_0 announces on the blockchain that she wants to sell a token, but only if the potential buyer pays at least a *secret* reserve price x_0, known to be in a predefined set, $x_0 \in \mathbb{Z}_L = \{0, \ldots, L-1\}$.[23] A buyer P_1 can then come and announce (publicly) a price x_1 on the blockchain. If $x_1 \geq x_0$, the auction succeeds, P_1 should get the token and P_0 obtains x_0 coins in return. Otherwise, the auction completes without moving these assets.

Construction. The application-layer protocol (here: the auction) is implemented as a decentralized app: a combination of a smart contract and local instructions to parties. Concretely, we implement an application-contract AuctionContract which handles the aspect of the auction protocol that rely on consensus state and rules. In turn, AuctionContract makes use of a single GaTC and its associated contract GaTCContract. The role of AuctionContract is to collect application-level inputs from the parties (here: the auction bid); instruct the GaTC which commitments to open accordingly (by calling GaTC.RequestOpen), and then learn the opening (by calling GaTC.GetResult) and act on it (here: by announcing the auction's outcome and potentially transferring the token and the coins if the auction is successful).

Because AuctionContract decides which index the underlying GaTC will be queried on, it serves as the *controller* for that GaTC. Thus, in the realization, the address of the application contract will be passed as ctrl to GaTC.Create, which tells the underlying GaTCContract to respect GaTC.RequestOpen calls from the application contract. This means that at the time when the GaTC is created and the collateral is deposited, it comes with irrevocable instructions for how to determine which commitment to open (which may depend on anything visible to the application contract).

To initiate an auction with a secret reserve price x_0, the seller P_0 does the following:

1. Create an instance of the AuctionContract smart contract (which implements the behavior described below).
2. Let $\boldsymbol{\mu} = (\mu_0, \ldots, \mu_L)$ where $\mu_i = 0$ if $i < x_0$ and $\mu_i = 1$ if $i \geq x_0$. (This is the truth table of the "does the bid reach the reserve price x_0?" function.)
3. Prepare a deposit deposit of a collateral (e.g., a given amount of Ether).
4. Call GaTC.Create($\boldsymbol{\mu}$, AuctionContract, deposit) to create a GaTC for $\boldsymbol{\mu}$, with the AuctionContract instance as the controller that determines which entry of $\boldsymbol{\mu}$ will be opened.
5. Call a method of AuctionContract that tell it to control this instance of GaTCContract.
6. Broadcast the existence of the auction and the address of AuctionContract instance that serves it.

Later, a buyer P_1 wishes to post a bid on this auction, with a public bid price x_1. To do so, it calls a Bid method of AuctionContract, passing x_1. If this is the first such call, then Bid records the account address of P_1 and the bid x_1, and invokes GaTC.RequestOpen(x_1).

Subsequently, the GaTC ensures that value μ_{x_1} will be revealed, whether by having P_0 invoke GaTC.NominalOpen(), or by having some bounty hunter invoke GaTC.ForceOpen(). In either case, and regardless of the auction's outcome, whoever successfully posted the commitment opening will collect the collateral: in the nominal case, P_0 just gets the collateral back; whereas in the bounty case, the bounty hunter receives the collateral as compensation and incentive for their computational work. Given a sufficiently large collateral, this is guaranteed to occur in the presence of rational bounty hunters.

Once the GaTC has been opened, GaTC.GetResult() will output μ_{x_1} when called. Thus the auction outcome can be decided: if $\mu_{x_1} = 1$ the bid has met the reserve price and the trade is settled, meaning that P_0 gets x_1 coins and P_1 gets the token; otherwise the reserve price was not met, and P_0 and P_1 retain their assets. (Automatic settlement can be implemented within AuctionContract, by having it take custody of the token being sold and of the bid, and then either sending them back or exchange one for the other, according to the bid outcome.)

Note that the above protocol assures the decision and settlement, regardless of whether the parties are online and cooperative when the auction outcome is decided and settled.

[23] We assume here that prices are denominated in small integers. More generally, we can support an arbitrary monotonically-increasing list of L real-valued prices, and let x_0 (and x_1 below) serve as *indices* into this list.

The above protocol protects the privacy of the reserve price, though not of the bids. To make the latter private as well, and reveal nothing but the final result, one can use the additional "private inputs" transformation described in Section 1.1.

5 Label-Driven MPC

In this section we present a generalization of garbled circuits which we call Label-Driven MPC (LD-MPC). The purpose of this generalization is to provide constructions of MPC that use secret labels and that are more robust to the exposure of additional labels, beyond the ones required for the computation. Due to space limitation, formal definitions and constructions are provided in Appendix C. Here, we only provide informal definitions.

LD-MPC is defined as follows. Party P_0 holds a secret input x_0 and a public function f to be computed on x_0 and on the inputs $x_1, \ldots, x_N$ held by parties $P_1, \ldots, P_N$. P_0 generates a message m_0 as well as labels $\{\mu_{j,u}\}_{j\in[\gamma],u\in\mathbb{Z}_L}$ for γ wires: $(\mathsf{m}_0, \{\mu_{j,u}\}_{j\in[\gamma],u\in\mathbb{Z}_L}) \leftarrow \mathsf{Msg}_0(1^\lambda, x_0)$. For a Yao-based construction, γ is the number of inputs and $L = 2$ (see Construction C.9 in Appendix C). The other parties P_i can generate messages m_i depending on P_0's message m_0 and their input x_i: $\mathsf{m}_i \leftarrow \mathsf{Msg}_i(\mathsf{m}_0, x_i)$.

Finally, evaluation is done in two steps. First the set of labels that need to be used (represented by a vector $\boldsymbol{\sigma} \in \mathbb{Z}_L^\gamma$) is computed: $\boldsymbol{\sigma} \coloneqq \mathsf{Eval}_1(\mathsf{m}_0, \mathsf{m}_1, \ldots, \mathsf{m}_N)$. For a Yao-based construction, $\boldsymbol{\sigma} = x_1 \| \ldots \| x_N$. Then the output $y = f(x_0, x_1, \ldots, x_N)$ is computed from the labels $\{\mu_{j,\sigma_j}\}_j$ selected by $\boldsymbol{\sigma}$: $y \leftarrow \mathsf{Eval}_2(\mathsf{m}_0, \mathsf{m}_1, \ldots, \mathsf{m}_N, \{\mu_{j,\sigma_j}\}_{j\in[\gamma]})$.

6 Gage MPC

In this section we describe how to achieve our final goal of Gage MPC. We utilize our Gage TC (GaTC) and Label-Driven MPC. Recall that Gage MPC enables a party P_0 with private input x_0, to allow the evaluation of a function $f(x_0, \bullet, \ldots, \bullet)$, on a set of inputs $x_1, \ldots, x_N$ held by parties $P_1, \ldots, P_N$. As described in the Introduction this is achieved as follows: P_0 takes the labels created by the Label-Driven MPC and for each wire, create a GaTC with the labels of this wire. Party P_0 deposits a collateral to create each GaTC. Once the parties $P_1, \ldots, P_N$ reveal their messages $\mathsf{m}_1, \ldots, \mathsf{m}_N$ corresponding to their respective inputs $x_1, \ldots, x_N$, this defines which labels need to be revealed (i.e., which POTC in each GaTC needs to be opened). Each GaTC then gives party P_0 a grace period during which it can open the needed POTC and retrieve its collateral. After the grace period, the collateral is utilized to pay bounty hunters to complete the computation by opening the needed POTCs. In either case, the output $f(x_1, \ldots, x_N)$ can be publicly computed from the opened labels.

The security of the Gage MPC follows from the security of the Label-Driven MPC and the GaTC. Moreover, security holds even under parallel composition, since GaTC already allows for parallel commitments.

We assume that party P_0 also has an address which it will use for retrieving its collateral. Let $\mathit{\Pi} = (\mathsf{Msg}_0, \mathsf{Msg}_1, \ldots, \mathsf{Msg}_N, \mathsf{Eval}_1, \mathsf{Eval}_2)$ be a Label-Driven MPC for the computation of the function f held by P_0. Given the secret input x_0, party P_0 proceeds as follows to create a Gage MPC.

1. Setup carried out by P_0:
   (a) Run $(\mathsf{m}_0, \{\mu_{j,u}\}_{j\in[\gamma],u\in\mathbb{Z}_L}) \leftarrow \mathsf{Msg}_0(1^\lambda, x_0)$. Set $\boldsymbol{\mu}_j = \{\mu_{j,u}\}_{u\in\mathbb{Z}_L}$ for $j \in [\gamma]$.
   (b) Determine level of complexity desired for the opening of the label of a wire. Fix the collateral to the amount related to this level of complexity. Create γ payment transactions $\{\mathsf{deposit}_j\}_{j\in[\gamma]}$ for this collateral amount, one for each wire.
   (c) Instantiate γ times the GaTC functionality as: $\{\mathsf{GaTC}_j\}_{j\in[\gamma]}$ incorporating the collateral from the previous step.
   (d) Create an application smart contract GMPC that will implement the full MPC and will serve as controller for the GaTC:
      – Party P_i can invoke (a single time) $\mathsf{GMPC.Msg}(i, \mathsf{m}_i)$ with message m_i to record the message m_i.
      – If all the messages $\mathsf{m}_1, \ldots, \mathsf{m}_N$ are recorded, GMPC computes $\boldsymbol{\sigma} \coloneqq \mathsf{Eval}_1(\mathsf{m}_0, \ldots, \mathsf{m}_N)$. Then for each $j \in [\gamma]$, invoke $\mathsf{GaTC}_j.\mathsf{RequestOpen}(\sigma_j)$.

– Any party can invoke $\mathsf{GMPC.GetResult}()$ to get the result of the computation. If the result y was already computed before, GMPC outputs y. Otherwise, for each $j \in [\gamma]$, it invokes $\mathsf{GaTC}_j.\mathsf{GetResult}()$ to get μ_{j,σ_j}. If one of μ_{j,σ_j} is $\perp$, output $\perp$. Otherwise, compute and output $y \leftarrow \mathsf{Eval}_2(\mathsf{m}_0, \mathsf{m}_1, \ldots, \mathsf{m}_N, \{\mu_{j,\sigma_j}\}_j)$.

(e) Call $\mathsf{GaTC}_j.\mathsf{Create}$ as follows:
$\mathsf{GaTC}_j.\mathsf{Create}(\boldsymbol{\mu}_j, \mathsf{GMPC}, \mathsf{deposit}_j)$

2. P_i, $i \in [N]$, computes $\mathsf{m}_i \leftarrow \mathsf{Msg}_i(\mathsf{m}_0, x_i)$ and invokes $\mathsf{GMPC.Msg}(i, \mathsf{m}_i)$.
3. Nominal path: Given $\boldsymbol{\sigma}$ party P_0 invokes $\mathsf{GaTC}_j.\mathsf{NominalOpen}()$ for all $j \in [\gamma]$ to reveal the labels $\{\mu_{j,\sigma_j}\}$ necessary to finish the computation.
4. Bounty path: After deadline T (from GaTC), any bounty hunter opening a time capsule that must be opened (corresponding to a σ_j) receives the corresponding collateral.
5. Once all openings are available (whether if by the nominal path or the bounty path) anybody can get the result of the computation by invoking $\mathsf{GMPC.GetResult}()$.

7 Implementation of Time Capsule and Gage Auction

We implemented a C++ library for POTC and used it to build a real-world application of Gage MPC, namely, Gage auctions over a blockchain. We used the Ivory Runtime Library [53] for elliptic curve operations over Curve25519 and garble circuit operations. With the chosen parameters, all POTC operations, except the force opening, take less than a few milliseconds on a standard laptop. The real-world application uses the Parity version of the Ethereum Virtual Machine (EVM). We implemented both the simple scheme from Section 4 and the one based on Section C.1. We only implemented the Label-Driven MPC that corresponds to a standard garbled circuit. More robust Label-Driven MPC essentially adds an additional NIMPC. This essentially replaces 2γ 128-bit label by around γ^2/ϵ labels of size $2(\gamma+1)^2 \log \gamma$, where γ is the original number of labels, and we want to achieve $(1-\epsilon)\gamma$-robustness. Overhead computation cost for evaluation is about 2γ sums of γ terms modulo a prime number of size $\log \gamma$. More details can be found in Appendix D.

Acknowledgments. The authors thank Craig Gentry for helping with some probability computations. Tal Malkin was supported in part by a grant from the Columbia-IBM center for Blockchain and Data Transparency, by a gift from LexisNexis Risk Solutions, and by JPMorgan Chase & Co. Tal Rabin was supported by ONR award N00014-19-1-2292. Abhishek Shah was supported by an NSF Graduate Fellowship. Any views or opinions expressed herein are solely those of the authors, and may differ from the views and opinions expressed by JPMorgan Chase & Co. or its affiliates. This material is not a product of the Research Department of J.P. Morgan Securities LLC. This material should not be construed as an individual recommendation for any particular client and is not intended as a recommendation of particular securities, financial instruments or strategies for a particular client. This material does not constitute a solicitation or offer in any jurisdiction.

References

1. Altcoin.io decentralized exchange. https://altcoin.io/
2. Etherdelta decentralized exchange. https://etherdelta.com/
3. Etheropt decentralized exchange (mirror of original software). https://github.com/destenson/etheropt--etheropt.github.io
4. Intrinsically tradable tokens. https://github.com/o0ragman0o/ITT
5. Ren: A privacy preserving virtual machine powering zero-knowledge financial applications. https://renproject.io/litepaper.pdf
6. Solidity by example: Blind auction. https://solidity.readthedocs.io/en/v0.5.3/solidity-by-example.html#id2
7. Andrychowicz, M., Dziembowski, S., Malinowski, D., Mazurek, L.: Secure multiparty computations on bitcoin. In: 2014 IEEE Symposium on Security and Privacy. pp. 443–458. IEEE Computer Society Press (May 2014)
8. Badertscher, C., Maurer, U., Tschudi, D., Zikas, V.: Bitcoin as a transaction ledger: A composable treatment. In: Katz, J., Shacham, H. (eds.) CRYPTO 2017, Part I. LNCS, vol. 10401, pp. 324–356. Springer, Heidelberg (Aug 2017)

9. Baum, C., David, B., Dowsley, R.: Insured mpc: Efficient secure computation with financial penalties. Financial Cryptography and Data Security (FC) (2020)
10. Beimel, A., Gabizon, A., Ishai, Y., Kushilevitz, E., Meldgaard, S., Paskin-Cherniavsky, A.: Non-interactive secure multiparty computation. In: Garay, J.A., Gennaro, R. (eds.) CRYPTO 2014, Part II. LNCS, vol. 8617, pp. 387–404. Springer, Heidelberg (Aug 2014)
11. Bellare, M., Goldwasser, S.: Encapsulated key escrow. Tech. rep., Cambridge, MA, USA (1996)
12. Ben-Or, M., Goldwasser, S., Wigderson, A.: Completeness theorems for non-cryptographic fault-tolerant distributed computation (extended abstract). In: 20th ACM STOC. pp. 1–10. ACM Press (May 1988)
13. Benhamouda, F., Krawczyk, H., Rabin, T.: Robust non-interactive multiparty computation against constant-size collusion. In: Katz, J., Shacham, H. (eds.) CRYPTO 2017, Part I. LNCS, vol. 10401, pp. 391–419. Springer, Heidelberg (Aug 2017)
14. Bentov, I., Kumaresan, R.: How to use bitcoin to design fair protocols. In: Garay, J.A., Gennaro, R. (eds.) CRYPTO 2014, Part II. LNCS, vol. 8617, pp. 421–439. Springer, Heidelberg (Aug 2014)
15. Berenzon, D.: Constant function market makers: Defi's "zero to one"" innovation. https://medium.com/bollinger-investment-group/constant-function-market-makers-defis-zero-to-one-innovation-968f77022159 (2020)
16. Boneh, D., Bonneau, J., Bünz, B., Fisch, B.: Verifiable delay functions. In: Shacham, H., Boldyreva, A. (eds.) CRYPTO 2018, Part I. LNCS, vol. 10991, pp. 757–788. Springer, Heidelberg (Aug 2018)
17. Boneh, D., Naor, M.: Timed commitments. In: Bellare, M. (ed.) CRYPTO 2000. LNCS, vol. 1880, pp. 236–254. Springer, Heidelberg (Aug 2000)
18. Bowe, S., Chiesa, A., Green, M., Miers, I., Mishra, P., Wu, H.: Zexe: Enabling decentralized private computation. Cryptology ePrint Archive, Report 2018/962 (2018), https://eprint.iacr.org/2018/962.pdf
19. Brakerski, Z., Döttling, N., Garg, S., Malavolta, G.: Leveraging linear decryption: Rate-1 fully-homomorphic encryption and time-lock puzzles. In: Hofheinz, D., Rosen, A. (eds.) TCC 2019, Part II. LNCS, vol. 11892, pp. 407–437. Springer, Heidelberg (Dec 2019)
20. Bünz, B., Agrawal, S., Zamani, M., Boneh, D.: Zether: Towards privacy in a smart contract world. IACR Cryptology ePrint Archive 2019, 191 (2019), https://eprint.iacr.org/2019/191
21. Chaum, D., Crépeau, C., Damgård, I.: Multiparty unconditionally secure protocols (extended abstract). In: 20th ACM STOC. pp. 11–19. ACM Press (May 1988)
22. Choudhuri, A.R., Goyal, V., Jain, A.: Founding secure computation on blockchains. In: Ishai, Y., Rijmen, V. (eds.) EUROCRYPT 2019, Part II. LNCS, vol. 11477, pp. 351–380. Springer, Heidelberg (May 2019)
23. Choudhuri, A.R., Green, M., Jain, A., Kaptchuk, G., Miers, I.: Fairness in an unfair world: Fair multiparty computation from public bulletin boards. In: Thuraisingham, B.M., Evans, D., Malkin, T., Xu, D. (eds.) ACM CCS 2017. pp. 719–728. ACM Press (Oct / Nov 2017)
24. Cleve, R.: Limits on the security of coin flips when half the processors are faulty (extended abstract). In: 18th ACM STOC. pp. 364–369. ACM Press (May 1986)
25. David, B., Gazi, P., Kiayias, A., Russell, A.: Ouroboros praos: An adaptively-secure, semi-synchronous proof-of-stake blockchain. In: Nielsen, J.B., Rijmen, V. (eds.) EUROCRYPT 2018, Part II. LNCS, vol. 10821, pp. 66–98. Springer, Heidelberg (Apr / May 2018)
26. De Santis, A., Di Crescenzo, G., Ostrovsky, R., Persiano, G., Sahai, A.: Robust non-interactive zero knowledge. In: Kilian, J. (ed.) CRYPTO 2001. LNCS, vol. 2139, pp. 566–598. Springer, Heidelberg (Aug 2001)
27. DeFiprime.com: Dex tracker - decentralized exchanges trading volume. https://defiprime.com/dex-volume
28. Deuber, D., Döttling, N., Magri, B., Malavolta, G., Thyagarajan, S.A.K.: Minting mechanism for proof of stake blockchains. In: International Conference on Applied Cryptography and Network Security. pp. 315–334. Springer (2020)
29. Dwork, C., Naor, M.: Pricing via processing or combatting junk mail. In: Brickell, E.F. (ed.) CRYPTO'92. LNCS, vol. 740, pp. 139–147. Springer, Heidelberg (Aug 1993)
30. Ephraim, N., Freitag, C., Komargodski, I., Pass, R.: Continuous verifiable delay functions. In: Annual International Conference on the Theory and Applications of Cryptographic Techniques. pp. 125–154. Springer (2020)
31. Feige, U., Kilian, J., Naor, M.: A minimal model for secure computation (extended abstract). In: 26th ACM STOC. pp. 554–563. ACM Press (May 1994)
32. Feige, U., Shamir, A.: Zero knowledge proofs of knowledge in two rounds. In: Brassard, G. (ed.) CRYPTO'89. LNCS, vol. 435, pp. 526–544. Springer, Heidelberg (Aug 1990)
33. Garay, J., Kiayias, A., Ostrovsky, R.M., Panagiotakos, G., Zikas, V.: Resource-restricted cryptography: Revisiting mpc bounds in the proof-of-work era. In: Annual International Conference on the Theory and Applications of Cryptographic Techniques. pp. 129–158. Springer (2020)
34. Garay, J.A., Kiayias, A., Leonardos, N.: The bitcoin backbone protocol: Analysis and applications. In: Oswald, E., Fischlin, M. (eds.) EUROCRYPT 2015, Part II. LNCS, vol. 9057, pp. 281–310. Springer, Heidelberg (Apr 2015)
35. Goldreich, O., Micali, S., Wigderson, A.: How to play any mental game or A completeness theorem for protocols with honest majority. In: Aho, A. (ed.) 19th ACM STOC. pp. 218–229. ACM Press (May 1987)

36. Gordon, S.D., Malkin, T., Rosulek, M., Wee, H.: Multi-party computation of polynomials and branching programs without simultaneous interaction. In: Johansson, T., Nguyen, P.Q. (eds.) EUROCRYPT 2013. LNCS, vol. 7881, pp. 575–591. Springer, Heidelberg (May 2013)
37. Goyal, R., Goyal, V.: Overcoming cryptographic impossibility results using blockchains. In: Kalai, Y., Reyzin, L. (eds.) TCC 2017, Part I. LNCS, vol. 10677, pp. 529–561. Springer, Heidelberg (Nov 2017)
38. Halevi, S., Ishai, Y., Jain, A., Komargodski, I., Sahai, A., Yogev, E.: Non-interactive multiparty computation without correlated randomness. In: Takagi, T., Peyrin, T. (eds.) ASIACRYPT 2017, Part III. LNCS, vol. 10626, pp. 181–211. Springer, Heidelberg (Dec 2017)
39. Halevi, S., Lindell, Y., Pinkas, B.: Secure computation on the web: Computing without simultaneous interaction. In: Rogaway, P. (ed.) CRYPTO 2011. LNCS, vol. 6841, pp. 132–150. Springer, Heidelberg (Aug 2011)
40. Heilman, E., Lipmann, S., Goldberg, S.: The arwen trading protocols. Financial Cryptography and Data Security (FC) (2020)
41. Kaptchuk, G., Green, M., Miers, I.: Giving state to the stateless: Augmenting trustworthy computation with ledgers. In: NDSS 2019. The Internet Society (Feb 2019)
42. Kiayias, A., Russell, A., David, B., Oliynykov, R.: Ouroboros: A provably secure proof-of-stake blockchain protocol. In: Katz, J., Shacham, H. (eds.) CRYPTO 2017, Part I. LNCS, vol. 10401, pp. 357–388. Springer, Heidelberg (Aug 2017)
43. Kiayias, A., Zhou, H.S., Zikas, V.: Fair and robust multi-party computation using a global transaction ledger. In: Fischlin, M., Coron, J.S. (eds.) EUROCRYPT 2016, Part II. LNCS, vol. 9666, pp. 705–734. Springer, Heidelberg (May 2016)
44. Kosba, A.E., Miller, A., Shi, E., Wen, Z., Papamanthou, C.: Hawk: The blockchain model of cryptography and privacy-preserving smart contracts. In: 2016 IEEE Symposium on Security and Privacy. pp. 839–858. IEEE Computer Society Press (May 2016)
45. Labs, A.: Idex: A real-time and high-throughput ethereum smart contract exchange. https://idex.market/
46. Malavolta, G., Thyagarajan, S.A.K.: Homomorphic time-lock puzzles and applications. In: Boldyreva, A., Micciancio, D. (eds.) CRYPTO 2019, Part I. LNCS, vol. 11692, pp. 620–649. Springer, Heidelberg (Aug 2019)
47. Nakamoto, S.: Bitcoin: A peer-to-peer electronic cash system. White Paper, https://bitcoin.org/bitcoin.pdf (2008)
48. Naor, M.: Moderately hard functions: From complexity to spam fighting. In: International Conference on Foundations of Software Technology and Theoretical Computer Science. pp. 434–442. Springer (2003)
49. Noether, S.: Ring signature confidential transactions for monero. IACR Cryptology ePrint Archive 2015, 1098 (2015), http://eprint.iacr.org/2015/1098
50. Pass, R., Seeman, L., shelat, a.: Analysis of the blockchain protocol in asynchronous networks. In: Coron, J., Nielsen, J.B. (eds.) EUROCRYPT 2017, Part II. LNCS, vol. 10211, pp. 643–673. Springer, Heidelberg (Apr / May 2017)
51. Peterson, J., Krug, J.: Augur: a decentralized, open-source platform for prediction markets. arXiv preprint arXiv:1501.01042 (2015)
52. Rabin, T., Ben-Or, M.: Verifiable secret sharing and multiparty protocols with honest majority (extended abstract). In: 21st ACM STOC. pp. 73–85. ACM Press (May 1989)
53. Rindal, P.: The ivory secure computation runtime. https://github.com/ladnir/Ivory-Runtime, [Online; accessed 2019-10-07]
54. Rivest, R.L., Shamir, A., Wagner, D.A.: Time-lock puzzles and timed-release crypto. Tech. rep., Cambridge, MA, USA (1996)
55. Sasson, E.B., Chiesa, A., Garman, C., Green, M., Miers, I., Tromer, E., Virza, M.: Zerocash: Decentralized anonymous payments from bitcoin. In: 2014 IEEE Symposium on Security and Privacy. pp. 459–474. IEEE (2014)
56. Van Saberhagen, N.: Cryptonote v 2.0 (2013)
57. Warren, W., Bandeali, A.: 0x: An open protocol for decentralized exchange on the ethereum blockchain. https://github.com/0xProject/whitepaper/blob/master/0x_white_paper.pdf
58. Williamson, Z.: The Aztec protocol. https://github.com/AztecProtocol/
59. Yao, A.C.C.: Protocols for secure computations (extended abstract). In: 23rd FOCS. pp. 160–164. IEEE Computer Society Press (Nov 1982)
60. Zhang, T., Wang, L.: Republic protocl: A decentralized dark pool exchange providing atomic swaps for ethereum-based assets and bitcoin. https://republicprotocol.github.io/whitepaper/republic-whitepaper.pdf

A Related Work

Following its significant impact on cryptocurrencies, Blockchain technology continues to provide novel templates for building new applications and reshaping distributed computing. This technology also implements an innovative computation model that can be utilized to realize new cryptographic primitives and protocols, as well as circumventing some impossibility results. In this section, we review some of the prior works that exploited this model, in addition to one of the important applications of our constructions, namely, decentralized auctions and exchanges.

A.1 Cryptographic Constructions in the Blockchain Model

The works under this category varies based on which aspects of the blockchain model they exploit. These include the fully decentralized and publicly verifiable payment service, the public bulletin board that blockchains build, and the consensus protocols (and their features) that power this model.

The framework introduced in [14] employs the distributed currency exchange medium of cryptocurrencies to design fair MPC protocols (with dishonest majority). These protocols realize a financial-based notion of fairness in which the party that aborts after learning the output loses its penalty deposit to the honest players. A similar approach has been applied in [7] by building timed commitments that are tied to penalty deposits. The committer loses this deposit if she does not open the commitment within a predefined timeframe. A notion for this financial-based fairness has been formalized in [43] under what is called secure MPC with compensation. The work also extended the previous fairness model to guarantee output delivery or robustness. Another work [9] introduced the notion of Insured MPC and investigated the precise properties of public verifiability needed to achieve this fairness notion, instead of just assuming the use of a public blockchain.

All the previous schemes do not realize complete fairness, i.e., all or none of the parties will obtain the output. Also, they face the challenge of how to accurately estimate the adversary's incentive to set the penalty deposit in a way that economically motivate continuing the protocol until the end. [23] solved this problem by utilizing a public bulletin board (can be instantiated using a blockchain) and witness encryption to achieve complete fairness. The proposed scheme, first, runs an unfair MPC to produce a ciphertext of the output, and then requires a witness to be posted on the bulletin board in order to decrypt, which allows everyone to learn the output. However, the instantiations of this construction require either extractable witness encryption, which has no known practical implementation (nor even theoretical constructions under standard assumptions), or the use of secure hardware like SGX.

A recent work [33] studied the observation that in blockchain-based consensus what matters is the honest power majority (computing power, or stake, etc.) rather than the number of parties. Then they exploit it to circumvent the $n/3$ lower bound of MPC with malicious parties when no private correlated randomness setup (e.g., a PKI) is used. The authors introduced the notion of resource-restricted paradigm and used it along with a fresh CRS to implement MPC that tolerate $t < n/2$ malicious parties.

Other works combined the blockchain model with hardware devices to implement secure computations. For example, [41] used a public bulletin board to enable a trusted execution environment (TEE) hosted by an untrusted computer to create a secure state without requiring a persistent internal storage. This also allows implementing interactive secure computations that depend on external events logged on the bulletin board. Our scheme supports this feature as well; opening time capsules can be tied to any event or message posting on the blockchain. In another work [44], a privacy preserving smart contract framework is proposed, which permits implementing MPC protocols on the blockchain. However, it requires trusting a manager with the users' inputs. Our scheme does not introduce such a special entity, and thus, it is fully decentralized. On the other hand, [18] targeted a stronger privacy notion, in which not only the user's input is protected, but also the executed functionality. The authors developed a decentralized private computation primitive that enables users to implement any computation off-chain and publish on-chain transactions attesting to the state change. Validating these transactions does not reveal anything about off-chain computations, nor require re-executing them. Similarly, our scheme addresses both data privacy and function privacy (as mentioned previously, one can garble a universal circuit in our incentivized MPC to keep the function private). However, our scheme is more general in the sense it works over any ledger, while [18] needs a privacy preserving ledger.

A related line of work used the blockchain model as an alternative to strong assumptions. [37] used it to avoid the trusted setup needed for non-interactive knowledge (NIZK) proof systems, and constructed one time programs from proof-of-stake blockchain without any trusted hardware tokens. Despite being innovative, this work relies on a strong simulator that controls the honest miners, and hence, realizes only local (or private) blockchains. [22] strengthened this model by relying on a weaker simulator that does not have this additional power, which allowed the use of global public blockchains. The authors also exploited the blockchain model to construct a concurrent self-composable MPC based on standard assumptions.

Employing the blockchain in building secure cryptographic primitives requires a formal definition of this model and its security notion. A large body of work has been dedicated to tackle this problem. Among them, [34, 50] provided a property based definition of the blockchain security and proved that Bitcoin consensus protocol satisfies these properties. While [8] introduced a universally composable treatment for Bitcoin's protocol. Other works targeted proof-of-stake consensus by developing both a property-based security notion [42], and a universally composable one [25].

A.2 Blockchain-based Auctions

As mentioned previously, the emergence of blockchain technology saw the introduction of many decentralized exchanges (or auctions), which are implemented using a blockchain and often trade assets whose ownership is represented on the blockchain. Such an application can be built using our constructions of Gage MPC as was introduced earlier.

In general, decentralized exchanges are implemented as "Dapps" (decentralized applications), consisting of a smart contract running on a programmable blockchain such as Ethereum, coupled with a user-facing application that intermediates between the smart contract and the user (and sometimes, back-end services are also interacting with the smart contract). Ideally, the smart contract, whose correct execution is assured by the underlying blockchain platform, tracks and executes orders and bids, while the user-facing application lets users authorize orders, deposits and withdrawals. Examples of such on-chain decentralized exchanges include Intrinsically Tradable Tokens [4], EtherOpt [3], Augur [51], and Altcoin.io [1].

Other exchanges, such as EtherDelta [2], IDEX [45], and 0x [57] use a hybrid architecture where order tracking and matching are done by a centralized trading engine, but settlement is done by a smart contract. This can improve costs, scalability and latency, while still ensuring that settlement does not violate users' orders (e.g., limit prices and expiry). However, the drawback is that it trusts a centralized party to correctly track and execute orders (e.g., matching an offer against the best bid). Arwen [40] follows such a hybrid approach, where the custody service is done on-chain (by creating escrows of the cryptocurrency coins to trade on the blockchain) while the atomic swap is done off-chain on a centralized exchange. The main goal is to utilize the liquidity of these exchanges.

These exchanges use *open orders books* where the orders are broadcast in plaintext and are visible to any interested party. They do not provide privacy for users' inputs.

Hidden Order Book auctions, where bids are revealed only at the auction's end, can be realized by using a commit-and-open paradigm that combines smart contracts and cryptographic commitments [6]. In the bidding phase, bidders send commitments to their bids, along with adequate deposits, to the smart contract. In the opening phase, the bidders open their commitments; the smart contract computes and settles the highest bid, and returns the deposits of bids that were opened and valid but not the highest. This approach has the drawback that bidders' participation in the reveal phrase is required for correct completion of the auction with respect to the committed bids. Parties' participation is *incentivized* by confiscation of their deposit if they fail to open their commitments. However, they are at liberty to do so, and may have greater *auxiliary incentives* that would make it rational to defect from the protocol.

This problem can be addressed by forced-open auctions, where our schemes allow building that as presented earlier. Another work, namely, the time-lock puzzle-based sequential auctions in [28] achieve a similar goal by exploiting the miners to solve time lock puzzles containing the openings of all unopened commitments. However, the goal is to build an auction-based proof-of-stake (PoS) mining algorithm to discourage hoarding currency rather than a generalized auction functionality. Furthermore, the proposed scheme suffer from a potential DoS attack against miners, where participants in the auction are not strongly incentivised to open their commitments. This leaves the miners with the job of doing that without any penalty on the participants' side.

Another approach is taken by Ren[24], which follows the aforementioned hybrid architecture, but realizes the trusted centralized trading engine using an interactive MPC done by a committee. Orders are submitted

[24] The Ren (née Republic Protocol) whitepaper [5, 60] provides only an informal and perhaps aspirational overview; we have been unable to find any specification or implementation of the cryptographic protocol alluded to.

to the committee by secret-sharing them across the members. The committee is assumed to have sufficiently many honest parties, and members' honest behavior is incentivized by bond deposits that would be confiscated if misbehavior is detected. Compared to the commit-and-open approach, Ren's approach has the advantage that orders can be kept secret, unless and until they are matched and settled. However this approach, too, would not be resilient to auxiliary incentives that exceed the value of the bonds. Moreover, it relies on *interaction with specific parties*; if (sufficiently many of) these refuse to cooperate, the orders can never be opened and executed.

Beyond hiding the order book, it may be desirable to hide the *amounts* and *parties* of *settled* trades. That is not possible on public blockchains that log everything in the clear. There exist privacy-preserving blockchain protocols [20,49,55,56,58] and deployments (e.g., Monero and Zcash). By operating the auction or exchange functionality on such a blockchain, and with suitable integration, it appears feasible to hide *all* trading and settlement information from *all* parties; each party would know only the orders they've placed and the assets they have sent/received. Detailed study of this integration lies outside the scope of the current work.

Automated Market Maker is an alternative model for decentralized exchanges, which does not use an order book at all. Instead, a smart contract holds pools of the traded assets, and offers an interface by which traders can deposit one asset and withdraw another, at an algorithmically-determined exchange rate. This includes the constant-product market maker algorithm (used by, e.g., Uniswap and Bancor), and variants (as used by, e.g., Curve); see [15] for a survey. Essentially, this makes one of the trading counterparties (the market maker) ultimately non-private: not merely its orders are public, but even its trading algorithm is published as a smart contract.

B Proof-of-Opening Time Capsules

In this section we present proof-of-opening time capsules (POTC), which enhance time capsules [11,54] to add the feature that a party that opens the time capsule can prove in zero knowledge that it knows the opening (rather than exposing the committed value in the clear). We propose candidate constructions that satisfy our definition. The feature of proving the opening will be critical for our design of the new concept of Gage Time Capsules (GaTC) and Gage MPC.

This appendix expands on the brief discussion of POTC in Section 2.

B.1 Time Capsules

We recall the notion of time capsules from [11]. In contrast to [11], we modify the definition slightly for our setting, to allow for the decommitment information to be different from the randomness used to generate the commitment. Because of that, we introduce an additional algorithm used to verify the decommitment.

In the definition below, λ is a standard cryptographic security parameter (in particular, 2^λ is not feasible). On the other hand, λ^* is a parameter capturing the hardness of brute force-opening: 2^{λ^*} is the time it takes to obtain a decommitment from the commitment alone; it should be feasible (polynomial), but hard (λ^* is a measure of how costly this is).

Definition B.1. *A* time capsule *is defined by three polynomial-time algorithms* $(\mathsf{TC.Com}, \mathsf{TC.DVer}, \mathsf{TC.ForceOpen})$ *with the following syntax:*

- *Commitment:* $(c, d) \leftarrow \mathsf{TC.Com}(1^\lambda, 1^{\lambda^*}, \mu)$ *takes as input the security parameter λ, the opening hardness parameter λ^*, a message $\mu \in \{0,1\}^{\mathrm{poly}(\lambda)}$, and outputs a commitment c and its associated decommitment d.*
- *Decommitment Verification:* $\mathsf{TC.DVer}(1^\lambda, c, \mu, d)$ *takes as input the security parameter λ, a commitment c, a message μ, a decommitment d and outputs* 1 *if the decommitment is valid with regards to c and μ; and outputs* 0 *otherwise.*
- *Forced Opening:* $(\mu, d) \leftarrow \mathsf{TC.ForceOpen}(c)$ *takes as input a commitment c, brute-forces its opening, and outputs the committed message μ and a valid decommitment d.*

We require the following properties to be satisfied:

Perfect correctness. *There exists a negligible function* negl*, such that for all parameters* $\lambda, \lambda^* \in \mathbb{N}$*, all messages* $\mu \in \{0,1\}^{\mathrm{poly}(\lambda)}$*:*

$$\Pr\left[\mathsf{TC.DVer}(1^\lambda, c, \mu, d) = 1 \;\middle|\; (c,d) \leftarrow \mathsf{TC.Com}(1^\lambda, 1^{\lambda^*}, \mu)\right] = 1 \;\;,$$

and

$$\Pr\left[\mathsf{TC.DVer}(1^\lambda, c, \mu', d') = 1 \;\middle|\; \begin{array}{l}(c,d) \leftarrow \mathsf{TC.Com}(1^\lambda, 1^{\lambda^*}, \mu)\\ (\mu', d') \leftarrow \mathsf{TC.ForceOpen}(c)\end{array}\right] \geq 1 - \mathrm{negl}(\lambda) \;\;.$$

Furthermore, TC.ForceOpen *should make at most about* 2^{λ^*} *operations.*[25] *We assume that* 2^{λ^*} *is polynomial in* λ *for all the asymptotic security analyses.*

Binding. *For all PPT adversaries* $\mathcal{A}$*, there exists a negligible function* negl*, such that for all security parameters* $\lambda \in \mathbb{N}$*:*

$$\Pr\left[\begin{array}{l}\mu \neq \mu' \wedge\\ \mathsf{TC.DVer}(1^\lambda, c, \mu, d) = 1 \wedge\\ \mathsf{TC.DVer}(1^\lambda, c, \mu', d') = 1\end{array} \;\middle|\; (c, \mu, d, \mu', d') \leftarrow \mathcal{A}(1^\lambda)\right] \leq \mathrm{negl}(\lambda) \;\;.$$

Remark B.2. The binding property holds independently from λ^**: the adversary can select any* λ^* *of its choice.*

Hiding. *The essence of the time capsules is dealing with the time and/or computation required to achieve the forced opening. We start with the intuition, and give the formal definition of our actual requirement below (see Definition B.3). Intuitively, the forced opening algorithm should take about* 2^{λ^*} *operations to open the capsule. However, if significantly fewer operations are computed by the adversary, the capsule should remain completely hiding. This property must hold even when there are* $\kappa + 1$ *capsules: if the adversary spends significantly fewer than* $\kappa \cdot 2^{\lambda^*}$ *operations, it should not be able to learn more than* κ *of the values in the capsules.*

The hiding property above is informal. If we only considered adversaries making significantly fewer than 2^{λ^*} operations, we could require that the classical hiding property holds: the commitment of a message μ_0 is indistinguishable from a commitment of a message μ_1. However, because we want to consider stronger adversaries that may be able to force open several commitments but not all of them, we need to introduce a different property. It is unclear how to extend classical hiding to this setting. Instead, we consider a much stronger property: *simulatability*.

To define simulatability, we assume that our commitments are in the random oracle model and that the operations that we count are queries to the random oracle H. Concretely, we bound by some number q, the number of queries, made by the adversary, to H. We require the existence of a simulator able to simulate commitments (without knowing the messages), later open them to any message, and simulate the random oracle H. Since the adversary can make enough random oracle queries to force open a certain number κ of commitments, to properly simulate the random oracle, the simulator may need to learn what messages were committed to in these commitments (but not the other commitments). Hence the simulator has access to an oracle Open'' outputting the committed messages of any commitment made by the adversary. However, the simulator is restricted to make at most κ queries to this oracle Open''.

Simulatability is an extension of the security property of trapdoor commitment [32], which states that there exists a simulator able to simulate commitments and later open them to any message.

In the formal definition of simulatability, we introduce yet another oracle $O^\star$. This oracle allows to model the operation in the generic group model or a second random oracle H^*. We do not restrict the number of queries to $O^\star$, thus in particular to H^*. This is in contrast to H, which the adversary can only access q times. Introducing $O^\star$ is without loss of generality: if the time capsule implementation does not require the generic model nor an extra random oracle, then this oracle $O^\star$ can be replaced by a trivial oracle doing nothing.

Definition B.3 (Simulatability). *A time-capsule* (TC.Com, TC.DVer, TC.ForceOpen) *(that has access to a random oracle* H *and another oracle* $O^\star$*) is* (q, κ, ϵ)*-simulatable if there exists a PPT simulator* TC.Sim

[25] Later we formally define operations as calls to a hash function modeled as a random oracle.

Real World Oracles:

Setup: Empty tables T for the commitments, and H for the random oracle H.

$\underline{\mathsf{Com}(\mu)}$:
$(c, d) \xleftarrow{R} \mathsf{TC.Com}(1^\lambda, 1^{\lambda^*}, \mu)$
$\mathsf{T}[c] := (\mu, d)$
Return c

$\underline{\mathsf{Open}(c)}$:
Return $\mathsf{T}[c]$

$\underline{H(x)}$:
If $\mathsf{H}[x] = \perp$:
$\quad \mathsf{H}[x] \leftarrow \{0,1\}^{\nu_H}$
$\quad$(assume the output of H has ν_H bits)
Return $\mathsf{H}[x]$

$O^\star(x)$ is defined by the scheme. It can for example correspond to the operation of the generic group model.

Ideal World Oracles:

Setup: Empty table T' for the commitments.

$\underline{\mathsf{Com}'(\mu)}$:
$c \xleftarrow{R} \mathsf{TC.Sim}(\mathtt{com}, 1^\lambda, 1^{\lambda^*})$
$\mathsf{T}'[c] := \mu$
Return c

$\underline{\mathsf{Open}'(c)}$:
$\mu := \mathsf{T}'[c]$
Return $\mathsf{TC.Sim}(\mathtt{open}, 1^\lambda, c, \mu)$

$\underline{H'(x)}$:
Return $\mathsf{TC.Sim}^{\mathsf{Open}''}(\mathtt{H}, 1^\lambda, x)$
$\quad$ where $\mathsf{Open}''(c)$ returns $\mu = \mathsf{T}'[c]$
$\quad$ ($\mathsf{TC.Sim}$ is allowed to make at most κ queries to Open'')

$\underline{O'^\star(x)}$:
Return $\mathsf{TC.Sim}(\mathtt{O}^*, 1^\lambda, x)$
the first argument of $\mathsf{TC.Sim}$ is the letter O with the exponent *

Fig. 4. Oracles for Simulatability Security Notion (Definition B.3)

(making at most κ queries to an oracle Open'' defined below), such that for any PPT adversary $\mathcal{A}$ making at most q queries to the random oracle H, for any auxiliary input $z \in \{0,1\}^{\mathrm{poly}(\lambda)}$:

$$|\Pr[\mathcal{A}^{\mathsf{Com},\mathsf{Open},H,O^\star}(z) = 1] - \Pr[\mathcal{A}^{\mathsf{Com}',\mathsf{Open}',H',O'^\star}(z) = 1]| \leq \epsilon \ .$$

where the oracles $\mathsf{Com}, \mathsf{Open}, H, \mathsf{Com}', \mathsf{Open}', H', O'^\star, \mathsf{Open}''$ are defined in Fig. 4. A time-capsule is (q,κ)-simulatable if it is (q,κ,ϵ)-simulatable for ϵ that is a negligible function of the security parameter λ.

Remark B.4. Formally, we should write $\mathsf{TC.Com}^{H,O^\star}$, $\mathsf{TC.DVer}^{H,O^\star}$, $\mathsf{TC.ForceOpen}^{H,O^\star}$ but for the sake of simplicity we omit the oracles $H, O^\star$ when they are clear from the context.

In the sequel, all time capsules are assumed to be simulatable.

Remark B.5. We work in the semi-honest model and thus implicitly assume that the commitment is honestly generated. In cases where an adversary may generate an incorrect commitment that the forced opening algorithm cannot open at all, there would be a need to add a generic NIZK to the commitment proving that it was well formed.

B.2 Proof-of-Opening Time Capsules (POTC)

We now add the functionality to enable a party to prove that they know the value committed to in the time capsule. The simplest method available to a party to show that it it knows the opening of a commitment is to announce the opening (μ, d). However, this simple method will not work for our purposes, because an attacker who observes this value being sent can block or front-run it, and present this opening as their own.

One way to solve this issue is to design schemes for which it is possible to prove the opening of a commitment to a message μ without exposing the decommitment d, thus disarming the malicious party. Intuitively, assuming that finding a decommitment of a commitment c is hard even if the committed message μ is known, we could use generic simulation-sound extractable NIZK proofs [26] to achieve this goal. Assuming that each party has some unique account address associated to it, we will bind this address to the proof, e.g via a label or tag. Indeed, simulation-sound extractable proofs allow a simulator to extract the witness (the decommitment d in our case) of proofs generated by the adversary, even if simulated proofs (containing no witness in particular) are provided to the adversary. This means that a malicious party cannot steal a proof from an honest party (that we simulate and hence does not contain any witness), as otherwise, we could extract from this malicious party a valid decommitment, which is hard (require to perform approximately 2^{λ^*} operations).

More formally, we introduce the notion of Proof-of-Opening Time Capsules (POTC)

Definition B.6. *A* proof-of-opening Time Capsule *is defined by five algorithms* $\mathsf{TC} = (\mathsf{TC.Com}, \mathsf{TC.DVer}, \mathsf{TC.ForceOpen}, \mathsf{TC.Prove}, \mathsf{TC.PVer})$ *with the following syntax:*

- *Time Capsule:* $(\mathsf{TC.Com}, \mathsf{TC.DVer}, \mathsf{TC.ForceOpen})$ *is a (simulatable) time capsule. It is assumed to use a hash function H modeled as a random oracle.*
- *Opening Proof:* $\pi \leftarrow \mathsf{TC.Prove}(c, \mu, d, \mathsf{tag})$ *generates a proof π with tag* tag *that c commits to μ using the witness d.*
- *Proof Verification:* $\mathsf{TC.PVer}(1^\lambda, c, \mu, \pi, \mathsf{tag})$ *takes as input the security parameter λ, a commitment c, a message μ, a proof π, a tag* tag*, and outputs 1 if the proof is valid with regards to c, μ, and* tag*; and outputs 0 otherwise.*

We require the following properties to be satisfied:

Perfect Correctness, Binding, and Simulatability. *These properties should hold for the time capsule* $(\mathsf{TC.Com}, \mathsf{TC.DVerTC.ForceOpen})$*. The properties are defined exactly as in Definition B.1, except we augment the ideal world oracles with lists to support non-malleability with respect to the same* $\mathsf{TC.Sim}$ *(see the non-malleability definition below, and Fig. 5).*

Soundness. *For all PPT adversaries $\mathcal{A}$, there exists a negligible function* negl*, such that for all security parameters $\lambda \in \mathbb{N}$:*

$$\Pr\left[\begin{array}{l}\mu \neq \mu' \wedge \\ \mathsf{TC.PVer}(1^\lambda, c, \mu, \pi, \mathsf{tag}) = 1 \wedge \\ \mathsf{TC.PVer}(1^\lambda, c, \mu', \pi', \mathsf{tag}') = 1\end{array} \middle| (c, \mu, \pi, \mathsf{tag}, \mu', \pi', \mathsf{tag}') \leftarrow \mathcal{A}(1^\lambda)\right] \leq \mathrm{negl}(\lambda) \ .$$

Non-Malleability. TC *is (q, κ, ϵ)-non-malleable if there exists a PPT simulator* $\mathsf{TC.Sim}$ *(making at most κ queries to an oracle* Open'' *defined below), such that for any PPT adversary $\mathcal{A}$ making at most q queries to the random oracle H, for any auxiliary input $z \in \{0,1\}^{\mathrm{poly}(\lambda)}$:*

$$\Pr\left[\begin{array}{l}\mathsf{TC.PVer}(1^\lambda, c, \mu, \pi, \mathsf{tag}) \wedge \\ (c, \mu) \notin L_{\mathsf{Open}} \wedge \\ (c, \mu, \mathsf{tag}) \notin L_{\mathsf{Prove}}\end{array} \middle| (c, \mu, \pi, \mathsf{tag}) \leftarrow \mathcal{A}^{\mathsf{Com}', \mathsf{Open}', \mathsf{Prove}', H', O'^{\star}}(z)\right] \leq \epsilon \ .$$

where the oracles $\mathsf{Com}', \mathsf{Open}', \mathsf{Prove}', H', O'^{\star}, \mathsf{Open}''$ *and the lists L_{Open} and L_{Prove} are defined in Fig. 5.* TC *is (q, κ)-strongly-simulatable if it is (q, κ, ϵ)-strongly-simulatable for ϵ a negligible function of the security parameter λ.*

Essentially, non-malleability ensures that if an adversary can only make q queries to the random oracle, then it can generate proofs with a fresh tag only for the commitments c it explicitly asked to open (via Open') or at most κ other commitments (that were implicitly opened via Open'' through random oracle queries to H). Intuitively, this means that to generate a proof for a fresh tag tag, the adversary must take enough time. It cannot "steal" the proof π generated by someone else with tag tag' and transform it into a proof with tag tag.

B.3 Constructions

Time Capsule Construction. We start by presenting the construction (without proof of opening) that was presented in [11] and references within. To commit to a message μ, the committer first selects a low-entropy seed $s \xleftarrow{R} \{0,1\}^{\lambda^*}$, then hashes this seed and uses this hash as a one-time pad to "encrypt" μ: $c_2 := H_2(s) \,\mathsf{xor}\, \mu$

Setup: Empty tables T' for the commitments. Empty lists L_{Open} and L_{Prove}. Com', H', Open' are defined as in Fig. 4 with the following changes: Open' and Open'' add (c, μ) to the list L_{Open}.	$\underline{\mathsf{Prove}'(c, \mathsf{tag})\text{:}}$ $\mu := \mathsf{T}'[c]$ $(\mu, d) \leftarrow \mathsf{TC.Sim}(\mathsf{open}, 1^\lambda, c, \mu)$ Add (c, μ, tag) to L_{Prove} Return $\mathsf{TC.Prove}(c, \mu, d, \mathsf{tag})$

Fig. 5. Oracles for Non-Malleability of Time Capsule with Opening Proofs (Definition B.6)

(where H_2 is a hash function). It also includes in the commitment a hash of the seed to be able to know when a seed is the valid one: $c_1 \coloneqq H_1(s)$ (where H_1 is another hash function). The commitment is $c \coloneqq (c_1, c_2)$.[26]

Intuitively, to force open a commitment, you need to enumerate the possible seeds s, hash them, and check which one corresponds to c_1. This requires at most 2^{λ^*} hash evaluations of H_1, and on average $2^{\lambda^*}/2$ hash evaluations of H_1.

Unfortunately, such a time-capsule does not satisfy our definition of security: an adversary making even a single hash evaluation of H_1 and H_2 may stumble upon the correct seed s, with probability $1/2^{\lambda^*}$, which is not negligible. This makes such a commitment difficult, if not impossible, to use as a building block of our MPC constructions.

A Better Time Capsule. We fix the above construction to improve the security. Instead of using a single seed $s \in \{0,1\}^{\lambda^*}$, we use k seeds $s_1, \dots, s_k \in \{0,1\}^{\nu_s}$, where $\nu_s = \lambda^* - \lfloor \log_2 k \rfloor$. We commit to each seed individually: $c_{1,i} = H_1(i \| s_i)$ and encrypt the message μ as $c_2 \coloneqq H_2(1\|s_1) \mathsf{~xor~} \cdots \mathsf{~xor~} H_2(k\|s_k) \mathsf{~xor~} \mu$. In the worst case, force-open a commitment takes $2^{\nu_s} \cdot k \approx 2^{\lambda^*}$ (by enumerating every s_i).

However, the main difference now is that if the adversary can only make significantly fewer than 2^{λ^*} hash evaluations, the commitment will remain statistically hiding. For example, if the adversary makes at most $k-1$ hash evaluations, the message μ is perfectly hidden, as the adversary will be missing one of the one-time pad mask $H_2(i\|s_i)$. This argument can be extended to adversary making many more than k hash evaluations using a careful probability analysis and tail bounds.

Formally, we have the following construction.

Construction B.7. Let $H_1 \colon \{0,1\}^* \to \{0,1\}^{2\lambda}$, $H_{\mathsf{K}} \colon \{0,1\}^* \to \{0,1\}^{\lambda}$, and $H_2 \colon \{0,1\}^* \to \{0,1\}^{\nu_\mu}$, be hash functions, where ν_μ is the length of the messages to committed.[27] In the security proof, we model all these hash functions using a single random oracle H, the input being prefixed by the name of the hash function and the output being truncated to the correct length. This construction does not use the extra oracle $O^\star$.

The scheme is defined as follows:

- $(c, d) \leftarrow \mathsf{TC.Com}(1^\lambda, 1^{\lambda^*}, \mu)$ generates low-entropy seeds $s_1, \dots, s_k \xleftarrow{R} \{0,1\}^{\nu_s}$ and a full-entropy salt $\mathsf{slt} \xleftarrow{R} \{0,1\}^{\lambda}$, where $\nu_s \coloneqq \lambda^* - \lfloor \log_2 k \rfloor$. It defines $d \coloneqq (s_1, \dots, s_k)$ and:

$$c \coloneqq (\lambda^*, \mathsf{slt}, \{c_{1,i}\}_{i \in [k]}, c_2)$$

$$\text{where } \begin{cases} K \coloneqq H_{\mathsf{K}}(1\|\mathsf{slt}\|s_1) \mathsf{~xor~} \cdots \mathsf{~xor~} H_{\mathsf{K}}(k\|\mathsf{slt}\|s_k), \\ c_{1,i} \coloneqq H_1(i\|\mathsf{slt}\|s_i),\ c_2 \coloneqq H_2(\mathsf{slt}\|K) \mathsf{~xor~} \mu \end{cases} \tag{1}$$

- $b = \mathsf{TC.DVer}(c, \mu, d)$ outputs $b = 1$ if and only if Eq. (1) is satisfied.
- $(\mu, d) \leftarrow \mathsf{TC.ForceOpen}(c)$ for each $i \in [k]$, enumerates $s_i \in \{0,1\}^{\nu_s}$ until finding s_i such that $c_{1,i} = H_1(i\|\mathsf{slt}\|s_i)$ is satisfied. When it happens, outputs $d = (s_1, \dots, s_k)$ and $\mu = H_2(\mathsf{slt}\|K) \mathsf{~xor~} c_2$, where $K = H_{\mathsf{K}}(1\|\mathsf{slt}\|s_1) \mathsf{~xor~} \cdots \mathsf{~xor~} H_{\mathsf{K}}(k\|\mathsf{slt}\|s_k)$.

We note that force-opening takes in the worst case $k2^{\nu_s} \approx 2^{\lambda^*}$ evaluations of the function H_1 (and k evaluation of the function H_{K}, as well as one evaluation of the function H_2). If not all $s_1, \dots, s_k$ are found,

[26] For the sake of simplicity, we do not include any salts in this overview, and assume that a single commitment is made. This is already sufficient to explain the issue we want to highlight.

[27] H_2 is used to allow to extend the output of one-time pad to the length of the message. In the overview above, H_2 essentially played the role of H_2 and H_{K}.

then the security provided by the random oracle model ensures that the message μ remains completely hidden. We also remark that finding a decommitment is hard, even if the committed message μ is known.

Theorem B.8. *Construction B.7 is a correct, binding, and (q, κ, ϵ) simulatable for:*

$$\epsilon \leq \frac{q^{k(\kappa+1)}}{2^{k(\kappa+1)\nu_s} \cdot (k(\kappa+1))!} + \mathrm{negl}(\lambda) \leq \frac{1}{\sqrt{2\pi k(\kappa+1)}} \left(\frac{qe}{(\kappa+1)2^{\lambda^*}} \right)^{k(\kappa+1)} + \mathrm{negl}(\lambda) \ ,$$

where e is the base of the natural logarithm In particular, if $q < \alpha(\kappa+1)2^{\lambda^*}$ for a constant $\alpha < 1/e$, then $\epsilon \leq \mathrm{negl}(k) + \mathrm{negl}(\lambda)$.

We give some concrete parameters in Table 1. We remark that even for 128 bits of security and even $\kappa = 0$ (the strongest notion of security), if the honest parties are assumed to make about 2^{40} hash evaluation to force open a commitment while the adversary is restricted to $2^{30} \cdot (\kappa+1)$ hash evaluations, then $k = 15$ is sufficient. And if we allow for $\kappa \geq 14$, then a single seed can even be used.

Table 1. Concrete parameters for Construction B.7

λ^*	q	k	κ	$-\log_2 \epsilon$
40	$2^{30} \cdot (\kappa+1)$	9	≥ 0	79.9
40	$2^{30} \cdot (\kappa+1)$	1	≥ 8	79.9
40	$2^{38} \cdot (\kappa+1)$	8	≥ 16	80
40	$2^{30} \cdot (\kappa+1)$	15	≥ 0	131
40	$2^{30} \cdot (\kappa+1)$	1	≥ 14	131

Proof. Correctness is straightforward.

Binding comes from the fact that H is collision resistant (as the output length is 2λ and it is modelled as a random oralce).

Simulatability. We then construct the simulator $\mathsf{TC.Sim}$ as follows: $\mathsf{TC.Sim}$ uses a table of counters $\mathsf{CNT}[\mathsf{slt}, i]$ counting the number of queries of the form $H_1(\mathsf{slt}\|i\|\bullet)$ and $H_{\mathtt{K}}(\mathsf{slt}\|i\|\bullet)$, where $\bullet \in \{0,1\}^{\lambda^*}$. It also maintains a table COM mapping a salt to some parameters, as well as tables H_1, H_2, and $\mathsf{H}_{\mathtt{K}}$, which are the tables for the random oracles H_1, H_2, and $H_{\mathtt{K}}$. When an entry does not exist in a table, its value is $\perp$.
$\mathsf{TC.Sim}(\mathtt{com}, 1^\lambda, 1^{\lambda^*}, \mu)$:

1. Generate $\mathsf{slt} \xleftarrow{R} \{0,1\}^\lambda$. If $\mathsf{COM}[\mathsf{slt}] \neq \perp$ or there exists entries of the form $\mathsf{slt}\|\star$ in H_1, H_2, or $\mathsf{H}_{\mathtt{K}}$, abort.
2. Generate $c_{1,1}, \dots, c_{1,k} \xleftarrow{R} \{0,1\}^{2\lambda}$, $K_1, \dots, K_k \xleftarrow{R} \{0,1\}^\lambda$, and $c_2 \xleftarrow{R} \{0,1\}^{\nu_\mu}$. Set $c := (\mathsf{slt}, \{c_{1,i}\}_{i\in[k]}, c_2)$. Define $K = K_1 \,\mathsf{xor} \cdots \mathsf{xor}\, K_k$ and $s_i := \perp$ for $i \in [k]$.
3. Generates $q_1^*, \dots, q_k^* \leftarrow [2^{\lambda^*}]$, and store $\mathsf{COM}[\mathsf{slt}] := (c, K, \{s_i, K_i, q_i^*\}_{i\in[k]}, \perp)$.
4. Return c.

$\mathsf{TC.Sim}(\mathtt{open}, 1^\lambda, c, \mu)$:

1. Find slt such that $\mathsf{COM}[\mathsf{slt}] = (c, K, \{s_i, K_i, q_i^*\}_{i\in[k]}, d)$, for some K, s_i, K_i, q_i^*, d.
2. If $d \neq \perp$, this means that the commitment was already open, return d.
3. Set $H^\dagger[\mathsf{slt}\|K] := \mu \,\mathsf{xor}\, c_2$.
4. For all i such that $s_i = \perp$, generate $s_i \leftarrow \{0,1\}^{\nu_s}$ and set $\mathsf{H}_1[\mathsf{slt}\|i\|s_i] := c_{1,i}$ and $\mathsf{H}_{\mathtt{K}}[\mathsf{slt}\|i\|s_i] := K_i$.
5. Set $d := (s_1, \dots, s_k)$.
6. Return d.

$\mathsf{TC.Sim}(\mathtt{H}, x)$:

– If this is a query of the form $H_1(\mathsf{slt}\|i\|s)$ (resp., $H_{\mathtt{K}}(\mathsf{slt}\|i\|s)$):

1. If $\mathsf{COM}[\mathsf{slt}] =\perp$, then return $\mathsf{H}_1[\mathsf{slt}\|i\|s]$ (resp., $\mathsf{H}_\mathsf{K}[\mathsf{slt}\|i\|s]$) —if the value is $\perp$, set it to a random value first.
2. Let $\mathsf{COM}[\mathsf{slt}] = (c, K, \{s_i, K_i, q_i^*\}_{i\in[k]}, d)$.
3. If $d \neq\perp$, then return $\mathsf{H}_1[\mathsf{slt}\|i\|s]$ (resp., $\mathsf{H}_\mathsf{K}[\mathsf{slt}\|i\|s]$) —if the value is $\perp$, set it to a random value first.
4. If there were $q_i^* - 1$ bitstrings s' for which $H_1(\mathsf{slt}\|i\|\star)$ or $H_\mathsf{K}(\mathsf{slt}\|i\|\star)$ was queried before, then set $s_i := s$, $\mathsf{H}_1[\mathsf{slt}\|i\|s_i] := c_{1,i}$, and $\mathsf{H}_\mathsf{K}[\mathsf{slt}\|i\|s_i] := K_i$.
5. Otherwise set $\mathsf{H}_1[\mathsf{slt}\|i\|s]$ (resp., $\mathsf{H}_\mathsf{K}[\mathsf{slt}\|i\|s]$) to a random value if not already set.
6. Return $\mathsf{H}_1[\mathsf{slt}\|i\|s]$ (resp., $\mathsf{H}_\mathsf{K}[\mathsf{slt}\|i\|s]$).

– If this is a query of the form $H_2(\mathsf{slt}\|K')$:
  1. If $\mathsf{COM}[\mathsf{slt}] =\perp$, then return $\mathsf{H}_2[\mathsf{slt}\|K']$ —if the value is $\perp$, set it to a random value first.
  2. Let $\mathsf{COM}[\mathsf{slt}] = (c, K, \{s_i, K_i, q_i^*\}_{i\in[k]}, d)$.
  3. If $K' \neq K$, then return $\mathsf{H}_2[\mathsf{slt}\|K']$ —if the value is $\perp$, set it to a random value first.
  4. If $K' = K$ and $d \neq\perp$, then return $\mathsf{H}_2[\mathsf{slt}\|K']$.
  5. If $K' = K$ and $d =\perp$, then:
     - If $s_i =\perp$ for some i, abort.
     - Otherwise, query $\mathsf{Open}''(c)$ and get the message μ. Set and return $\mathsf{H}_2[\mathsf{slt}\|K] := c_2 \mathbin{\mathsf{xor}} \mu$.

It remains to prove that the ideal world with the above simulator is indistinguishable from the real world and to prove that the simulator is making at most κ queries to Open''. For the latter point, let ϵ' be the probability that Open' is called at least $\kappa + 1$ times. We remark that Open'' can only be called for a commitment c with salt slt, when the adversary made queries to $H_1(\mathsf{slt}\|i\|s)$ or $H'(\mathsf{slt}\|i\|s)$ for at least q_i^* values s, for each $i \in [k]$. As the values q_i^* are uniformly random in $[2^{\nu_s}]$ and unknown to the adversary before it makes at least q_i^* queries as specified above, the worst case is when the adversary manage to query $H_1(\mathsf{slt}\|i\|s)$ for $i \in [k]$ and $\kappa+1$ different slt (corresponding to $\kappa+1$ different commitments. Let $n := k(\kappa+1)$. Let $\{X_i\}_{i\in[n]}$ be n independent random variables uniform in the *integer* interval $[2^{\nu_s}]$. We have:

$$\epsilon' \le \Pr[\sum_{i=1}^{n} X_i \le q] \ .$$

We conclude using the following lemma and the fact that $2^{\nu_s} \cdot k \ge 2^{\lambda^*}$.

Lemma B.9. *Let $X_1, \dots, X_n$ be n independent variables uniform in the integer interval $[N]$. For any positive integer q, we have:*

$$\Pr[\sum_{i=1}^{k} X_i \le q] \le \frac{q^n}{N^n \cdot n!} \le \frac{(qe)^n}{\sqrt{2\pi n}\cdot (Nn)^n} \ ,$$

where e is the base of the natural logarithm.

Proof. Let us count the number M of tuples $(x_1, \dots, x_n) \in \mathbb{N}_{>0}^n$ such that $\sum_{i=1}^n x_i \le q$. This is equivalent to counting the number of possibilities to throw n balls in q bins: x_1 is the index of the first bin with a ball, and x_i is the distance between the $(i-1)$-th bin with a ball and the i-th bin with a ball for $i \in \{2, \dots, n\}$. Hence we have: $M = \binom{q}{n}$.

Furthermore, we also have:

$$\Pr[\sum_{i=1}^{n} X_i \le q] \le M/N^n \le \binom{q}{n}/N^n \ ,$$

as there are N^n tuples $(x_1, \dots, x_n) \in [N]^n$.

We conclude using the fact that $\binom{q}{n} \le q^n/n!$ and the Stirling inequality $n! \ge \sqrt{2\pi n}(n/e)^n$. □

This conclude the proof of Theorem B.8. □

Proof of Opening Time Capsules. Using generic simulation-sound extractable NIZK proofs proving valid decommitment would work for incentivization in this construction but would be much too expensive. To provide better performance we change our commitment construction to be more algebraic and amenable to efficient Fiat-Shamir zero-knowledge proofs.

Construction B.10. We assume that the security parameter defines a cyclic group $\mathbb{G}$ of order p and generator g. Let h be another random generator of $\mathbb{G}$ (which can be generated by evaluating a hash function modeled as a random oracle on a fixed string). Let $f\colon \{0,1\}^\lambda \to \mathbb{G}$ be an injective function. Let $H_1\colon \{0,1\}^* \to \{0,1\}^{2\lambda}$, $H_{\mathtt{K}}\colon \{0,1\}^* \to \{0,1\}^\lambda$, $H_{\mathtt{r}}\colon \{0,1\}^* \to \mathbb{Z}_p$, $H_2\colon \{0,1\}^* \to \{0,1\}^{\nu_\mu}$, and $H^*\colon \{0,1\}^* \to \mathbb{Z}_p$ be hash functions, where ν_μ is the length of the messages to committed.

In the security proof, we model all these hash functions except H^* using a single random oracle H, the input being prefixed by the name of the hash function and the output being truncated to the correct length. For simulatability and non-malleability, the adversary is restricted to make q queries to these functions. On the other hand, $\mathbb{G}$ is modeled as a generic group and H^* as a random oracle, both modeled by the oracle $O^\star$ in Definition B.6. The adversary is allowed to make a polynomial number of queries to these functions (and is not constrained by making only q queries).

The scheme works as follows.

– $(c,d) \leftarrow \mathsf{TC.Com}(1^\lambda, 1^{\lambda^*}, \mu)$ generates low-entropy seeds $s_1,\ldots,s_k \xleftarrow{R} \{0,1\}^{\nu_s}$ and a full-entropy salt $\mathsf{slt} \xleftarrow{R} \{0,1\}^\lambda$, where $\nu_s \coloneqq \lambda^* - \lfloor \log_2 k \rfloor$. It computes:

$$c \coloneqq (\lambda^*, \mathsf{slt}, \{c_{1,i}\}_{i\in[k]}, c_2, c_3, c_4)$$
$$\text{where} \begin{cases} c_{1,i} \coloneqq H_1(\mathsf{slt}\|s_i), \\ K \coloneqq H_{\mathtt{K}}(1\|\mathsf{slt}\|s_1) \mathbin{\mathsf{xor}} \cdots \mathbin{\mathsf{xor}} H_{\mathtt{K}}(k\|\mathsf{slt}\|s_k), \\ c_2 \coloneqq H_2(\mathsf{slt}\|K) \mathbin{\mathsf{xor}} \mu, \\ r \coloneqq H_{\mathtt{r}}(1\|\mathsf{slt}\|s_1) + \cdots + H_{\mathtt{r}}(k\|\mathsf{slt}\|s_k) \in \mathbb{Z}_p, \\ c_3 \coloneqq g^r,\ c_4 \coloneqq h^r \cdot f(K) \end{cases} \tag{2}$$
$$d \coloneqq (K, r)\ .$$

– $\mathsf{TC.DVer}(c,\mu,d)$ checks that

$$c_2 = H_2(\mathsf{slt}\|K) \mathbin{\mathsf{xor}} \mu \quad \wedge \quad c_3 = g^r \quad \wedge \quad c_4 \coloneqq h^r \cdot f(K)\ .$$

– $(\mu, d) \leftarrow \mathsf{TC.ForceOpen}(c)$ for each $i \in [k]$, enumerates $s_i \in \{0,1\}^{\nu_s}$ until finding s_i such that $c_{1,i} = H_1(i\|\mathsf{slt}\|s_i)$ is satisfied. When it happens, outputs $d = (K, r)$ and $\mu = H_2(\mathsf{slt}\|K) \mathbin{\mathsf{xor}} c_2$, where $K = H_{\mathtt{K}}(1\|\mathsf{slt}\|s_1) \mathbin{\mathsf{xor}} \cdots \mathbin{\mathsf{xor}} H_{\mathtt{K}}(k\|\mathsf{slt}\|s_k)$ and $r = H_{\mathtt{r}}(1\|\mathsf{slt}\|s_1) + \cdots + H_{\mathtt{r}}(k\|\mathsf{slt}\|s_k)$.

– $\pi \leftarrow \mathsf{TC.Prove}(c,\mu,d,\mathsf{tag})$ outputs a proof $\pi \coloneqq (K, z, \mathsf{ch})$ where (z, ch) is a Fiat-Shamir proof of knowledge of $r \in \mathbb{Z}_p$ (under the tag tag) such that:

$$c_3 = g^r \quad \wedge \quad c_4 \coloneqq h^r \cdot f(K)\ .$$

Concretely, $\mathsf{TC.Prove}$ generates $t \leftarrow \mathbb{Z}_p$, sets $u_3 \coloneqq g^t$, $u_4 \coloneqq h^t$, computes $\mathsf{ch} \coloneqq H^*(c\|\mathsf{tag}\|u_3\|u_4)$, and $z \coloneqq t - \mathsf{ch} r \in \mathbb{Z}_p$.

– $\mathsf{TC.PVer}(1^\lambda, c, \mu, \pi, \mathsf{tag})$ verifies the proof $\pi = (K, z, e)$ by checking that:

$$u_3 = g^z c_3^{\mathsf{ch}} \quad \wedge \quad u_4 \coloneqq h^z (c_4/f(K))^{\mathsf{ch}} \quad \wedge\ \mathsf{ch} = H^*(c\|\mathsf{tag}\|u_3\|u_4) \quad \wedge \quad c_2 = H_2(\mathsf{slt}\|K) \mathbin{\mathsf{xor}} \mu\ .$$

Remark B.11. $\mathsf{TC.DVer}$ cannot check that $c_{1,i} = H_1(\mathsf{slt}\|s_i)$ as it does not receive the values s_i. The values $c_{1,i}$ are only used when force opening the commitment. The c_3, c_4 part, on the other hand, is used to allow for efficient Fiat-Shamir zero-knowledge proofs.

Theorem B.12. *Construction B.10 is a correct, binding, (q,κ,ϵ)-simulatable, sound, (q,κ,ϵ)-non-malleable POTC in the random oracle model and the generic group model for:*

$$\epsilon \le \frac{q^{k(\kappa+1)}}{2^{k(\kappa+1)\nu_s} \cdot (k(\kappa+1))!} + \operatorname{negl}(\lambda) \le \frac{1}{\sqrt{2\pi k(\kappa+1)}} \left(\frac{qe}{(\kappa+1)2^{\lambda^*}} \right)^{k(\kappa+1)} + \operatorname{negl}(\lambda)\ ,$$

where e is the base of the natural logarithm In particular, if $q < \alpha(\kappa+1)2^{\lambda^*}$ for a constant $\alpha < 1/e$, then $\epsilon \le \operatorname{negl}(k) + \operatorname{negl}(\lambda)$.

C Label-Driven MPC

Formally, we define Label-Driven MPC as follows:[28]

Definition C.1. *A Label-Driven MPC for a functionality* $f\colon (\{0,1\}^*)^{N+1} \to \{0,1\}^*$ *is defined by* $N+3$ *algorithms* $(\mathsf{Msg}_0, \mathsf{Msg}_1, \ldots, \mathsf{Msg}_N, \mathsf{Eval}_1, \mathsf{Eval}_2)$ *with the following syntax:*

- $\underline{P_0\text{-Message:}}$ $(\mathsf{m}_0, \{\mu_{j,u}\}_{j\in[\gamma],u\in\mathbb{Z}_L}) \leftarrow \mathsf{Msg}_0(1^\lambda, x_0)$ *takes as input the security parameter* λ*, the input* $x_0 \in \{0,1\}^{\mathrm{poly}(\lambda)}$ *of* P_0*, and outputs a message* m_0 *and a set of labels* $\{\mu_{j,u}\}_{j\in[\gamma],u\in\mathbb{Z}_L}$*, where* γ *and* L *are two parameters polynomial in* λ*.*
- $\underline{P_i\text{-Message } (i \in [N])\text{:}}$ $\mathsf{m}_i \leftarrow \mathsf{Msg}_i(\mathsf{m}_0, x_i)$ *takes as input a message* m_0 *from* P_0*, the input* $x_i \in \{0,1\}^{\mathrm{poly}(\lambda)}$ *of* P_i*, and outputs a message* m_i*.*
- *First Step of Evaluation:* $\boldsymbol{\sigma} \coloneqq \mathsf{Eval}_1(\mathsf{m}_0, \mathsf{m}_1, \ldots, \mathsf{m}_N)$ *takes as input the messages* $\mathsf{m}_0, \mathsf{m}_1, \ldots, \mathsf{m}_N$ *from all the parties and deterministically outputs a vector* $\boldsymbol{\sigma} \in \mathbb{Z}_L^\gamma$*.*
- *Second Step of Evaluation:* $y \leftarrow \mathsf{Eval}_2(\mathsf{m}_0, \mathsf{m}_1, \ldots, \mathsf{m}_N, \{\mu_{j,\sigma_j}\}_{j\in[\gamma]})$ *takes as input the messages* $\mathsf{m}_0, \mathsf{m}_1, \ldots, \mathsf{m}_N$ *as well as the labels* μ_{j,σ_j} *and outputs the value* y*.*

We require a Label-Driven MPC to satisfy the following properties:

Correctness. *There exists a negligible function* negl*, such that for all security parameters* $\lambda \in \mathbb{N}$*, all inputs* $x_0, \ldots, x_N \in \{0,1\}^{\mathrm{poly}(\lambda)}$ *the following probability is* $\geq 1 - \mathrm{negl}(\lambda)$*:*

$$\Pr\left[y = f(\{x_i\}_i) \;\middle|\; \begin{array}{l} (\mathsf{m}_0, \{\mu_{j,u}\}_{j,u}) \leftarrow \mathsf{Msg}_0(1^\lambda, x_0), \\ \forall i \in [N],\ \mathsf{m}_i \leftarrow \mathsf{Msg}_i(\mathsf{m}_0, x_i), \\ \boldsymbol{\sigma} \coloneqq \mathsf{Eval}_1(\mathsf{m}_0, \mathsf{m}_1, \ldots, \mathsf{m}_N), \\ y \leftarrow \mathsf{Eval}_2(\{\mathsf{m}_i\}_i, \{\mu_{j,\sigma_j}\}_j) \end{array} \right]$$

To define security, we first define the view of the Label-Driven MPC protocol that takes as input the set of corrupted parties, $\overline{P}$, as well as the inputs x_i and randomness ρ_i of all parties, as follows:

$$\mathsf{View}\Big(\overline{P}, \{x_i, \rho_i\}_{i\in[0,N]}\Big) \coloneqq (\{x_i, \rho_i\}_{i\in\overline{P}}, \{\mathsf{m}_i\}_{i\in[0,N]}, M) \ ,$$

where $(\mathsf{m}_0, \{\mu_{j,u}\}_{j,u}) \leftarrow \mathsf{Msg}_0(1^\lambda, x_0; \rho_0)$, and for all $i \in [N]$, $\mathsf{m}_i \leftarrow \mathsf{Msg}_i(\mathsf{m}_0, x_i)$, and:

- if $0 \in \overline{P}$ (P_0 is corrupted), $M = \{\mu_{j,u}\}_{j,u}$ are the labels produced by Msg_0.
- if $0 \notin \overline{P}$ (P_0 is not corrupted), $M = \{\mu_{j,\sigma_j}\}_j$ are the labels needed by Eval_2, where $\boldsymbol{\sigma} \coloneqq \mathsf{Eval}_1(\mathsf{m}_0, \mathsf{m}_1, \ldots, \mathsf{m}_N)$.

In the following definition we consider the setting of MPC with private inputs where the adversary is allowed to learn at most κ additional labels $\mu_{j,u}$. Note that the definition for the case where P_0 is corrupted does not include κ, this is due to the fact that P_0 knows already all the labels $\mu_{j,u}$.

Definition C.2 (κ-Robust Security, Private-Inputs). *A Label-Driven MPC is* κ*-robust secure with private-inputs if it satisfies the following properties:*

Security with Corrupted P_0**.** *There exists a stateful PPT simulator* $\mathcal{S}$*, such that for all PPT adversaries* $\mathcal{A}$*, there exists a negligible function* negl*, such that for all security parameters* λ*, sets of corrupted parties* $\overline{P} \subseteq [0,N]$*,* $0 \in \overline{P}$*, inputs* $x_0, x_1, \ldots, x_N$*, random tapes* $\{\rho_i\}_{i\in\overline{P}}$*, auxiliary input* $z \in \{0,1\}^{\mathrm{poly}(\lambda)}$*:*

$$|\Pr[\mathcal{A}^{\mathsf{Reveal}}(z, \mathsf{View}(\overline{P}, \{x_i, \rho_i\}_{i\in[0,N]})) = 1] - \Pr[\mathcal{A}^{\mathcal{S}(\texttt{reveal},\bullet)}(z, \mathcal{S}(\texttt{view}, \overline{P}, \{x_i, \rho_i\}_{i\in\overline{P}}, y)) = 1]| \leq \mathrm{negl}(\lambda) \ ,$$

where $y = f(x_0, x_1, \ldots, x_N)$ *and the adversary* $\mathcal{A}$ *is allowed access to an oracle* Reveal *that takes as input a pair* (j,u) *and outputs* $\mu_{j,u}$*.*

[28] Note that a LD-MPC is a tool like garbled circuits rather than a full-blown protocol, and as such, a property-based definition is more suitable. Composability of protocols using LD-MPC can be argued from these properties.

Security with Honest P_0. *This is defined similarly to security with corrupted P_0 except that $0 \notin \overline{P}$ and the adversary is restricted to make at most κ queries to* Reveal*.*

κ-robust security, public-inputs is defined similarly except that $\mathcal{S}$ is given as input all the inputs $x_1, \dots, x_N$ except for the input of P_0. When P_0 is corrupted, this security notion is trivial as all the inputs are known by the simulator. But when P_0 is not corrupted, the security notion ensures the privacy of x_0.

Our constructions provide the following theorems. First the basic construction which is a Yao garbled circuit (that can be constructed assuming one-way functions):

Theorem C.3. *Let f be a function that can be computed in polynomial time. Assuming the existence of one-way functions, there exists a correct and 0-robust secure public-inputs Label-Driven MPC for f.*

We further present an improved construction using error-correcting codes and NIMPC techniques and provide the following.

Theorem C.4. *Let f be a function that can be computed in polynomial time. Let $\varepsilon > 0$. Assuming the existence of one-way functions, there exists a correct and $(1-\epsilon)\gamma$-robust secure public-inputs Label-Driven MPC for f, for some integer γ (which is also the length of the vector $\boldsymbol{\sigma}$).*

This is a very strong theorem as it basically turns the efforts of the adversary into a 0-1 situation. Either the adversary exerts the full effort to complete the computation of the function on a different set of inputs or it learns nothing.

Before providing the detailed general transformation, we recall the formal definitions of two building blocks: error correcting code and NIMPC.

Building Block: Linear Error Correcting Code. We make use of linear error correcting codes.

Definition C.5. *Let $\mathbb{F}_L$ be a finite field.*[29] *A $[n,k,\delta]_L$* linear code *is a linear subspace C with dimension k of $\mathbb{F}_L^n$, such that the Hamming Distance of any two distinct codewords $\boldsymbol{x}, \boldsymbol{x}' \in C$ is at least δ (called the minimum distance of C). A* generating matrix *G of C is a $n \times k$-matrix whose rows generates the subspace C.*

For any $k \le n \le L$, we recall that there exist a $[n,k,(n-k+1)]_L$ linear code: the *Reed-Solomon* code.

Building Block: NIMPC. Let $Q_1, \dots, Q_\nu$ be ν parties that hold inputs $\chi_1, \dots, \chi_\nu$ respectively. (These parties are different from the parties $P_0, P_1, \dots, P_N$ of the incentivized MPC we are constructing.) We assume that for all $i \in [\nu]$, $\chi_i \in \mathbb{F}_L$, where L is a prime number of size logarithmic in the security parameter.

An NIMPC protocol for a public function g works as follow. A trusted party or setup with some private input $\chi_0 \in \{0,1\}^*$ generate correlated randomness $(\rho_0, \boldsymbol{\rho}) \leftarrow \mathsf{NIMPC.Setup}_g(1^\lambda, \chi_0)$. Each party P_i receives $\rho_i = \{\rho_{i,u}\}_{u \in \mathbb{F}_L}$ and send ρ_{i,χ_i} to a stateless evaluator who can compute the output:

$$g(\chi_0, \chi_1, \dots, \chi_\nu) = \mathsf{NIMPC.Eval}_g(\rho_0, \mu_1, \dots, \mu_\nu).$$

We define the *residual function* $g|_{\bar{T}, \boldsymbol{\chi}_{\bar{T}}}$ to be the function g with the inputs corresponding to positions $\bar{T} = \{0, \dots, \nu\} \setminus T$ fixed to $\{\chi_i\}_{i \in \bar{T}}$. Full robustness ensures that if the evaluator colludes with any subset T of parties, it only learns this residual function.

We now formally define adaptively fully robust NIMPC. The classical definition of NIMPC in [10] is not sufficient for our purposes as our Label-Driven MPC allows for adaptive corruption.

Definition C.6. *A non-interactive multiparty computation (NIMPC) for a function $g\colon \{0,1\}^* \times \mathbb{F}_L^\nu \to \{0,1\}^*$ is defined by 2 algorithms* $(\mathsf{NIMPC.Setup}, \mathsf{NIMPC.Eval})$ *with the following syntax:*

– *Setup: $(\rho_0, \boldsymbol{\rho}) \leftarrow \mathsf{NIMPC.Setup}_g(1^\lambda, \chi_0)$ takes as input the security parameter λ, a secret input χ_0, and outputs a bitstring ρ_0 and a vector $\boldsymbol{\rho} = \{\rho_{i,u}\}_{i \in [\nu], u \in \mathbb{F}_L}$ of νL string of* $\mathrm{poly}(\lambda)$ *bits. Each party P_i receives $\rho_i \coloneqq \{\rho_{i,u}\}_{u \in \mathbb{F}_L}$.*

[29] We use the unusual notation L for the order of the field as the usual notation q denotes the number of queries to the random oracle.

– *Evaluation:* $y \leftarrow \mathsf{NIMPC.Eval}_g(\rho_0, \mu_1, \ldots, \mu_\nu)$ *takes a input the value* ρ_0 *and the messages* $\mu_1, \ldots, \mu_\nu$ *from all the parties, and outputs the value* y.

(where the index g is omitted when clear from context) satisfying the following properties:

Perfect Correctness. *For all security parameters* $\lambda \in \mathbb{N}$, *all inputs* χ_0 *and* $\boldsymbol{\chi} = \{\chi_i\}_{i\in[\nu]}$*:*

$$\Pr\left[g(\chi_0, \boldsymbol{\chi}) \;=\; \mathsf{NIMPC.Eval}(\rho_{1,\chi_1}, \ldots, \mu_{\nu,\chi_\nu}) \;\mid\; \{\rho_{i,u}\}_{i\in[\nu],u\in\mathbb{F}_L} \;\leftarrow\; \mathsf{NIMPC.Setup}(1^\lambda, \chi_0)\right] \;=\; 1 \;.$$

Adaptive Full Robustness. *There exists a stateful PPT simulator* $\mathcal{S}$ *such that, for all adversaries* $\mathcal{A}$ *(even non polynomial-time ones), there exists a negligible function* negl, *such that for all security parameters* $\lambda \in \mathbb{N}$, *all inputs* χ_0 *and* $\boldsymbol{\chi} = \{\chi_i\}_{i\in[\nu]}$, *all auxiliary inputs* $z \in \{0,1\}^{\mathrm{poly}(\lambda)}$*:*

$$\left|\Pr[\mathcal{A}^{\mathsf{Corrupt}}(z, \rho_0, \{\rho_{i,\chi_i}\}_{i\in[\nu]}) \mid \boldsymbol{\rho} \leftarrow \mathsf{NIMPC.Setup}_g(1^\lambda, \chi_0)] - \Pr[\mathcal{A}^{\mathsf{Corrupt}'}(z, \mathcal{S}(\mathtt{msg}, 1^\lambda, g(\boldsymbol{\chi}))]\right| \leq \mathrm{negl}(\lambda) \;,$$

where the oracles Corrupt, Corrupt′ *are defined in Fig. 6.*

Real World Oracles:	Ideal World Oracles:
Setup: Empty list T of corrupted parties.	Setup: Empty list T of corrupted parties.
$\mathsf{Corrupt}(i)$:	$\mathsf{Corrupt}'(i)$:
Add i to T	Add i to T
Return $\{\rho_{i,u}\}_u$	Return $\mathcal{S}^{g\vert_{\bar{T}},\boldsymbol{\chi}_{\bar{T}}}(\mathtt{corrupt}, i)$

Fig. 6. Oracles for Adaptive Full Robustness of NIMPC (Definition C.6)

Using ideas from [13], we prove the following theorem that is used in our transformation below.

Theorem C.7. *Let* G *be the generating matrix of a* $[\nu, k, \delta]_L$ *linear code* C. *Let* $j \in [k]$ *and* $u \in \mathbb{F}_L$. *There exists an NIMPC for the following function:*

$$g_{G,j,u}(\chi_0, \chi_1, \ldots, \chi_\nu) = \begin{cases} (\chi_0, \chi_1, \ldots, \chi_\nu) & \text{if } \exists \boldsymbol{\sigma} \in \mathbb{F}_L^k,\ \boldsymbol{\chi}^\intercal = \boldsymbol{\sigma}^\intercal \cdot G \ \wedge\ \sigma_j = u \;, \\ (\bot, \chi_1, \ldots, \chi_\nu) & \text{otherwise,} \end{cases} \tag{3}$$

where $\boldsymbol{\chi} = (\chi_1, \ldots, \chi_\nu)$. *Furthermore, the protocol can be constructed from the parameters* G, j, u *in time polynomial in* ν, k, δ, L.[30]

The function in Eq. (3) essentially outputs a secret value χ_0 if χ is a valid codeword of a vector $\boldsymbol{\sigma}$, where σ_j is some value u. In our transformation, χ_0 will be the label $\mu_{j,u}$. We will use νL such NIMPC, one for each j, u.

Remark C.8. The inclusion of $\chi_1, \ldots, \chi_\nu$ in the output weakens the security requirement: the only actual hidden value is χ_0 is the following condition is not satisfied: $\exists \boldsymbol{\sigma} \in \mathbb{F}_L^k,\ \boldsymbol{\chi}^\intercal = \boldsymbol{\sigma}^\intercal \cdot G \ \wedge\ \sigma_j = u$. Otherwise, everything is revealed.

Proof Sketch of Theorem C.7. The high-level idea is to start from the NIMPC for the following function from [13] (where such an NIMPC is called an "outputting-message NIMPC"):

$$g_{M,\boldsymbol{v}}(\chi_0, \chi_1, \ldots, \chi_\nu) = \begin{cases} \chi_0 & \text{if } \boldsymbol{v} = M \cdot \boldsymbol{\chi}, \text{ where } \boldsymbol{\chi} = (\chi_1, \ldots, \chi_\nu) \;, \\ \bot & \text{otherwise,} \end{cases}$$

[30] Based on equation 3, our work provides fully robust (i.e., robust against any number of corruptions) protocols for the specific functionality we need.

where $M \in \mathbb{F}_L^{k\times\nu}$ be a public matrix and $\boldsymbol{v} \in \mathbb{F}_L^k$ be a public vector.[31]

The security property we need is slightly different from the one in [13]. On the one hand, our security requirements are weaker: we allow $\chi_1, \dots, \chi_\nu$ to be revealed in any case, and we allow $\boldsymbol{v}$ to be public (while it is secret in [13]). On the other hand, we need full robustness to hold in presence of adaptive corruptions, while only static corruptions are handled in [13]. We can prove that the construction works in this modified setting. Intuitively, since security holds perfectly (as it is proven via linear algebra), no adversarial input, even if chosen adaptively, will break it.

We can then conclude using the following remark. Let G be the generating matrix of a $[\nu, k, \delta]_L$ linear code C. Let $j \in [k]$ and $u \in \mathbb{F}_L$. We remark that the set of vectors $\boldsymbol{\chi} \in \mathbb{F}_L^\nu$ such that $\boldsymbol{\chi}^\intercal = \boldsymbol{\sigma}^\intercal \cdot G$ where $\sigma_j = u$ is an affine subspace of $\mathbb{F}_L^\nu$ and hence there exists a matrix M and a vector $\boldsymbol{v}$ such that:

$$\exists \boldsymbol{\sigma} \in \mathbb{F}_L,\ \boldsymbol{\chi}^\intercal = \boldsymbol{\sigma}^\intercal \cdot G \ \wedge\ \sigma_j = u \quad \Leftrightarrow \quad \boldsymbol{v} = M\boldsymbol{\chi}\ .$$

This concludes the sketch of the proof. □

We now proceed to present our designs for the LD-MPC.

C.1 0-Robust Secure, Public Inputs

The Yao garbled circuit design directly provides a 0-robust secure, public-inputs Label-Driven MPC. The fact that it satisfies the definition is derived directly from the security proofs of Yao.

Construction C.9. Concretely, the construction is as follows:

- $(\mathsf{m}_0, \{\mu_{j,u}\}_{j\in[\gamma],u\in\mathbb{Z}_L}) \leftarrow \mathsf{Msg}_0(1^\lambda, x_0)$, where $L = 2$, garbles the circuit corresponding to the function f. The message m_0 consists of the labels corresponding to the input x_0 and of the labels of the garbled circuit itself. The values $\mu_{j,u}$ are the input labels corresponding to the inputs $x_1, \dots, x_N$.
- $\mathsf{m}_i \leftarrow \mathsf{Msg}_i(\mathsf{m}_0, x_i)$ outputs $\mathsf{m}_i = x_i$.
- $\boldsymbol{\sigma} \leftarrow \mathsf{Eval}_1(\mathsf{m}_0, \dots, \mathsf{m}_N)$ return $\boldsymbol{\sigma} = x_1 \| x_2 \| \dots \| x_N$.
- $y \leftarrow \mathsf{Eval}_2(\mathsf{m}_0, \dots, \mathsf{m}_N, \{\mu_{j,\sigma_j}\}_j)$ evaluates the garbled circuit contained in m_0 using the labels for x_0 in m_0 and the labels $\{\mu_{j,\sigma_j}\}_j$ for $x_1, \dots, x_N$.

We recall that a garbled circuit scheme is correct if and only if the evaluation on the correct label yields the correct result with overwhelming probability; and it is secure if there exists a simulator able to generate the garbled circuit and the labels corresponding to the input, knowing only the output of the function. We thus have the following immediate theorem, which implies Theorem C.3.

Theorem C.10. *Construction C.9 is correct and 0-robust secure if and only if the underlying garbled circuit scheme is correct and secure.*

C.2 From 0-Robust Secure to $(1-\epsilon)\gamma'$-robust Secure

In this section, we show how to transform (public- or private-inputs) Label-Driven MPC which is 0-robust secure into one which is $(1-\epsilon)\gamma'$-secure. What this implies is that if the number of labels required to compute the function on a single set of input is γ' then to compute the function on an additional set of inputs would require to open more than $(1-\varepsilon)\gamma'$ additional labels. This is a great improvement as it increases the effort that the adversary needs to exert considerably in order to learn information about the function beyond what it is "legally" entitled to learn. Intuitively, when we design the Gage MPC this means that the adversary needs to be able to force-open more than $\kappa = (1-\epsilon)\gamma'$ time capsules, which is almost the number γ' of time capsules that bounty hunters need to open if P_0 does not follow the nominal path.

We start with a high-level overview. Let us focus on the public-inputs construction from Appendix C.1 for $N + 1 = 2$ parties. P_0 garbles the function $f(x_0, \cdot)$. P_1 outputs the message $\mathsf{m}_1 = x_1$ (its input). Evaluation consists of opening the commitments of the labels corresponding to the bits of $x_1 = \mathsf{m}_1$, which allows to evaluate the garbled circuit on x_1.

[31] The integer k in this definition (and in general, in this full section) has no relation with the number k of seeds in the construction of time capsules in Appendix B.3.

We proved this construction is 0-robust secure. But it is not even 1-robust secure for the following reason. If the adversary learns one more label (i.e., by force-opening one POTC), it now has two labels for the same wire of the garbled circuit. It is known that garbled circuits do not provide any security guarantees in this case.

The idea of our transformation is to encode the input x_1 using an error correcting code and to enforce that the only way for the adversary to learn anything is to learn labels corresponding to valid codewords. If the distance of the code is δ, this ensures that even if the adversary learns $\delta - 1$ more labels, it will learn nothing more.

To realize this idea, instead of using directly the labels of the garbled circuits, we use new labels (aka, correlated randomness) of a non-interactive multiparty computation (NIMPC) which outputs the garbled circuit labels, if the inputs are valid codewords (and nothing otherwise). The security properties of the NIMPC ensures that the only way for the adversary to learn any extra label of the inner garbled circuit is for the adversary to get labels from the NIMPC corresponding to another valid codeword. As discussed before, if the distance of the code is δ, this requires the adversary to learn δ more labels to be able to learn any extra label of the inner garbled circuit. Furthermore, for linear error correcting codes, such an NIMPC protocol can be constructed using techniques in [13].

The transformation can be generalized to any Label-Driven MPC and any number of parties. Instead of encoding x_1, the vector $\boldsymbol{\sigma}$ is encoded using the linear error correcting code.

Concretely, our construction proceeds as follows.

Construction C.11. Let $f\colon (\{0,1\}^*)^{N+1} \to \{0,1\}^*$ be a functionality. Let $\mathsf{MPC} = (\mathsf{Msg}_0, \ldots, \mathsf{Msg}_N, \mathsf{Eval}_1, \mathsf{Eval}_2)$ be a 0-opening secure Label-Driven MPC for f. For the sake of simplicity, we assume that L is a prime power and $L \geq \gamma$. This may slightly increase the number of labels output by Msg_0, but not the number of labels that need to be revealed to compute the output. (In particular, that does not increase the number of labels that need to be force-open in the resulting Gage MPC when P_0 does not follow the nominal path.) Let $\mathsf{NIMPC}_{g_{G,j,u}} = (\mathsf{NIMPC.Setup}_{g_{G,j,u}}, \mathsf{NIMPC.Eval}_{g_{G,j,u}})$ be NIMPC schemes for the functions $g_{G,j,u}$ defined in Eq. (3) (where $k = \gamma$).

Let $\epsilon > 0$. We want the resulting Label-Driven MPC to be $(1-\epsilon)\gamma'$-opening secure, for some well chosen γ' (where $\boldsymbol{\sigma} \in \mathbb{F}_L^{\gamma'}$). For that purpose, we set $\gamma' = \lceil \gamma/\epsilon \rceil$ and consider the $[\gamma', \gamma, \delta]_L$ Reed-Solomon code C with generating matrix G. Its distance is $\delta = \gamma' - \gamma \geq (1-\epsilon)\gamma'$.

We then construct the following Label-Driven MPC scheme $\mathsf{MPC}' = (\mathsf{Msg}'_0, \ldots, \mathsf{Msg}'_N, \mathsf{Eval}'_1, \mathsf{Eval}'_2)$ for f:

– $(\mathsf{m}'_0, \{\mu'_{j,u}\}_{j,u}) \leftarrow \mathsf{Msg}'_0(1^\lambda, x_0)$ generates $(\mathsf{m}_0, \{\mu_{j,u}\}_{j,u}) \leftarrow \mathsf{Msg}_0(1^\lambda, x_0)$ and then computes: for each $j \in [N]$ and $u \in \mathbb{F}_L$, it generates:

$$(\rho_0^{j,u}, \boldsymbol{\rho}^{j,u}) \leftarrow \mathsf{NIMPC.Setup}_{g_{G,j,u}}(1^\lambda, \mu_{j,u}) \ .$$

It then defines and outputs:

$$\begin{aligned} \mathsf{m}'_0 &\coloneqq (\mathsf{m}_0, \{\rho_0^{j,u}\}_{j,u}) \ , \\ \mu'_{j',u'} &\coloneqq \{\rho_{j',u'}^{j,u}\}_{j,u} \qquad \text{for all } j' \in [\gamma'],\ u' \in \mathbb{F}_L \ . \end{aligned}$$

– $\mathsf{m}'_i \leftarrow \mathsf{Msg}'_i(\mathsf{m}'_0, x_i)$ behaves exactly as Msg_i for $i \in [N]$.
– $\boldsymbol{\sigma}' \coloneqq \mathsf{Eval}'_1(\mathsf{m}'_0, \mathsf{m}'_1, \ldots, \mathsf{m}'_N)$ computes $\boldsymbol{\sigma} \coloneqq \mathsf{Eval}_1(\mathsf{m}_0, \mathsf{m}'_1, \ldots, \mathsf{m}'_N)$ and outputs $\boldsymbol{\sigma}' \coloneqq \boldsymbol{\sigma}^\intercal \cdot G$.
– $y \leftarrow \mathsf{Eval}'_2(\mathsf{m}'_0, \ldots, \mathsf{m}'_N, \{\mu'_{j',\sigma_{j'}}\}_{j'})$ first computes for all $j \in [\gamma]$:

$$\mu_{j,\sigma_j} = \mathsf{NIMPC.Eval}_{g_{G,j,\sigma_j}}(\rho_0^{j,\sigma_j}, \rho_{1,\sigma'_1}^{j,\sigma_j}, \ldots, \rho_{\nu,\sigma'_\nu}^{j,\sigma_j}) \ .$$

Finally it outputs $y \leftarrow \mathsf{Eval}_2(\mathsf{m}_0, \ldots, \mathsf{m}_N, \{\mu_{j,\sigma_j}\}_j)$, where $\mathsf{m}_i \coloneqq \mathsf{m}'_i$ for $i \in [N]$.

We have the following theorem, which implies Theorem C.4 when combined with Theorem C.3:

Theorem C.12. *Let* MPC *be a* 0*-robust public-inputs (resp., private-inputs) secure Label-Driven MPC (with* $\gamma = |\boldsymbol{\sigma}|$ *wires) for a function* f*. Then for any* $\epsilon > 0$*, there exists a* $(1-\epsilon)\gamma'$*-robust public-inputs (resp., private-inputs) secure Label-Driven MPC for* f *with* γ' *wires (where* $\gamma' = O(\gamma/\epsilon)$*).*

We state the following security theorem, which implies Theorem C.12:

Theorem C.13. *If* MPC *is a* 0*-opening public-inputs (resp., private-inputs) secure Label-Driven MPC for* f *and if the NIMPC schemes* $\mathsf{NIMPC}_{g_{G,j,u}}$ *are adaptively fully robust, then the scheme* MPC' *constructed above is a* $(1-\epsilon)\gamma'$*-opening public-inputs (resp., private-inputs) secure Label-Driven MPC for* f*.*

Proof Sketch. The main case to prove is security when P_0 is honest and the adversary force opens $(1-\epsilon)\gamma'$ additional commitments. In that case, adaptive full robustness of the NIMPC schemes ensure that the adversary does not learn anything more than when the adversary did not force open any additional commitment. This is because the residual function for $g_{G,j,u}$ always outputs $\bot$ if $u \neq \sigma_j$, and outputs $\bot$ or μ_{j,σ_j} otherwise, since the distance of the code is $\delta \geq (1-\epsilon)\gamma'$. ☐

C.3 From Public-Inputs to Private-Inputs Security

We show how to transform a public-inputs Label-Driven MPC into a private-inputs Label-Driven MPC, for the special case that there are two parties P_0 and P_1 (namely $N = 1$). The transformation makes use of any 2-round 2-party secure computation (2PC) with solitary output protocol, namely where only P_0 receives an output. As for Label-Driven MPC, we only consider the semi-honest setting.

Definition C.14. *A 2-Round 2-Party Secure Computation Protocol with solitary output (*2PC *for short, in this paper) for a functionality* $f\colon (\{0,1\}^*)^2 \to \{0,1\}^*$ *is defined by three algorithms* $(\mathsf{2PC.Msg}_0, \mathsf{2PC.Msg}_1, \mathsf{2PC.Out}_0)$ *with the following syntax:*

- *P_0-Message:* $(\tilde{\mathsf{m}}_0, \tilde{\mathsf{st}}_0) \leftarrow \mathsf{Msg}_0(1^\lambda, x_0)$ *takes as input the security parameter* λ*, the input* $x_0 \in \{0,1\}^{\mathrm{poly}(\lambda)}$ *of* P_0*, and output a message* $\tilde{\mathsf{m}}_0$ *and a state* $\tilde{\mathsf{st}}_0$*.*
- *P_1-Message:* $\tilde{\mathsf{m}}_1 \leftarrow \mathsf{2PC.Msg}_1(\tilde{\mathsf{m}}_0, x_1)$ *takes as input a message* $\tilde{\mathsf{m}}_0$ *from* P_0*, the input* $x_1 \in \{0,1\}^{\mathrm{poly}(\lambda)}$ *of* P_1*, and output a message* $\tilde{\mathsf{m}}_1$*.*
- *P_0-Output:* $y \leftarrow \mathsf{2PC.Out}(\tilde{\mathsf{st}}_0, \tilde{\mathsf{m}}_1)$ *takes a input the message* m_1 *from* P_1 *and the state* $\tilde{\mathsf{st}}_0$ *of* P_0*, and outputs the value* y*.*

satisfying the following properties:

Perfect Correctness. *For all security parameters* $\lambda \in \mathbb{N}$*, all inputs* $x_0, x_1 \in \{0,1\}^{\mathrm{poly}(\lambda)}$*:*

$$\Pr\left[f(x_0, x_1) = \mathsf{2PC.Out}(\tilde{\mathsf{st}}_0, \tilde{\mathsf{m}}_1) \;\middle|\; \begin{array}{l} (\tilde{\mathsf{m}}_0, \tilde{\mathsf{st}}_0) \leftarrow \mathsf{2PC.Msg}_0(1^\lambda, x_0), \\ \tilde{\mathsf{m}}_1 \leftarrow \mathsf{2PC.Msg}_1(\tilde{\mathsf{m}}_0, x_1) \end{array} \right] = 1 \enspace .$$

Security against Corrupted P_i **(for** $i \in \{0,1\}$**).** *There exists a simulator* $\mathcal{S}$*, such that for all PPT adversaries* $\mathcal{A}$*, there exists a negligible function* negl*, such that for all security parameters* λ*, inputs* x_0, x_1*, random tape* r_i*, auxiliary inputs* $z \in \{0,1\}^{\mathrm{poly}(\lambda)}$*:*

$$|\Pr[\mathcal{A}(z, \mathsf{View}(\{i\}, (x_0, r_0, x_1, r_1))) = 1] - \Pr[\mathcal{A}(z, \mathcal{S}(\{i\}, (x_i, r_i)) = 1]| \leq \mathrm{negl}(\lambda)0| \enspace ,$$

where:

$$\mathsf{View}(\{i\}, (x_0, r_0, x_1, r_1)) \coloneqq (x_0, x_1, r_i, \tilde{\mathsf{m}}_0, \tilde{\mathsf{m}}_1, y) \enspace ,$$

where $(\tilde{\mathsf{m}}_0, \tilde{\mathsf{st}}_0) \leftarrow \mathsf{2PC.Msg}_0(1^\lambda, x_0)$*,* $\tilde{\mathsf{m}}_1 \leftarrow \mathsf{Msg}_1(\tilde{\mathsf{m}}_0, x_1)$ *and* $y = f(x_0, x_1)$ *if* $i = 0$ *(i.e.,* P_0 *is corrupted).*

Transformation for $N+1=2$ parties. Let us now present the transformation from public-inputs to private-inputs for $N+1=2$ parties.

Construction C.15. Let $f\colon (\{0,1\}^*)^2 \to \{0,1\}^*$ be a functionality. Let $\mathsf{2PC} = (\mathsf{2PC.Msg}_0, \mathsf{2PC.Msg}_1, \mathsf{2PC.Out}_0)$ be a 2PC for f. Let g be the following function: $g(\tilde{\mathsf{st}}_0, \tilde{\mathsf{m}}_1) = \mathsf{2PC.Out}(\tilde{\mathsf{st}}_0, \tilde{\mathsf{m}}_1)$. Let $\mathsf{MPC} = (\mathsf{Msg}_0, \mathsf{Msg}_1, \mathsf{Eval})$ be a κ-opening public-inputs secure Label-Driven MPC for g.

We construct the following Label-Driven MPC protocol $\mathsf{MPC}' = (\mathsf{Msg}'_0, \mathsf{Msg}'_1, \mathsf{Eval}'_1, \mathsf{Eval}'_2)$ for f:

– $(\mathsf{m}'_0, \{\mu'_{j,u}\}_{j,u}) \leftarrow \mathsf{Msg}'_0(1^\lambda, x_0)$ generates $(\tilde{\mathsf{m}}_0, \tilde{\mathsf{st}}_0) \leftarrow \mathsf{2PC.Msg}_0(1^\lambda, x_0)$, computes $(\mathsf{m}_0, \{\mu_{j,u}\}_{j,u}) \leftarrow \mathsf{Msg}_0^{\mathsf{TC.Com}}(1^\lambda, \tilde{\mathsf{st}}_0)$, and outputs:

$$\mathsf{m}'_0 \coloneqq (\tilde{\mathsf{m}}_0, \mathsf{m}_0) \quad \text{and} \quad \{\mu'_{j,u}\}_{j,u} \coloneqq \{\mu_{j,u}\}_{j,u} \ .$$

– $\mathsf{m}'_1 \leftarrow \mathsf{Msg}'_i(\mathsf{m}'_0, x_1)$ generates $\tilde{\mathsf{m}}_1 \leftarrow \mathsf{2PC.Msg}_1(\tilde{\mathsf{m}}_0, x_1)$, compute $\mathsf{m}_1 \leftarrow \mathsf{Msg}_1(\mathsf{m}_0, \tilde{\mathsf{m}}_1)$, and outputs: $\mathsf{m}'_1 \coloneqq \mathsf{m}_1$.
– $\boldsymbol{\sigma} \leftarrow \mathsf{Eval}'_1(\mathsf{m}'_0, \mathsf{m}'_1)$ computes $\boldsymbol{\sigma} \coloneqq \mathsf{Eval}_1(\mathsf{m}_0, \mathsf{m}_1)$.
– $y \leftarrow \mathsf{Eval}'_2(\mathsf{m}'_0, \mathsf{m}'_1, \{\mu_{j,\sigma_j}\}_j)$ computes $y \coloneqq \mathsf{Eval}_2(\mathsf{m}_0, \mathsf{m}_1, \{\mu_{j,\sigma_j}\})$.

We have the following theorem, which implies Theorem C.17.

Theorem C.16. *If* 2PC *is a secure 2-round 2-party secure computation scheme for f with solitary output, and if* MPC *is a κ-robust public-inputs secure Label-Driven MPC for g, then the scheme* MPC′ *constructed above is a κ-robust private-inputs secure Label-Driven MPC for f.*

Proof sketch. Correctness follows from correctness of 2PC and MPC: using notations in the construction, the value output by Eval'_2 (on honestly generated inputs) is:

$$g(\tilde{\mathsf{st}}_0, \tilde{\mathsf{m}}_1) = \mathsf{2PC.Out}(\tilde{\mathsf{st}}_0, \tilde{\mathsf{m}}_1) = f(x_0, x_1) \ .$$

Private-Inputs Security when P_0 is corrupted. P_1 is simulated as follows: generate $\tilde{\mathsf{m}}_1$ using the simulator of P_1 for 2PC, and output $\mathsf{m}'_1 \coloneqq \mathsf{m}_1 \leftarrow \mathsf{Msg}_1(\mathsf{m}_0, \tilde{\mathsf{m}}_1)$.

Private-Inputs Security when P_1 is corrupted. P_0 is simulated as follows: generate $\tilde{\mathsf{m}}_0$ using the simulator of P_0 for 2PC, then generate m_0 using the simulator of P_0 for MPC, and output $\mathsf{m}'_0 \coloneqq (\tilde{\mathsf{m}}_0, \mathsf{m}_0)$. □

The above theorem implies Theorem C.17.

Theorem C.17 (informal). *In the setting with $N + 1 = 2$ parties, if there exists a secure 2-round 2-party secure computation scheme* 2PC *with solitary output for f and if there exists a κ-robust public-inputs secure Label-Driven MPC for a specific function g (corresponding to the "output" function of* 2PC*), then there exists a κ-robust private-inputs secure Label-Driven MPC for f.*

Remark C.18 (On the restriction to two parties). When there are more than two parties, private-input κ-robust security cannot be achieved (except for very limited functionalities). The adversary can indeed corrupt P_0 and P_1, honestly generate messages m_0, m_1, (for some inputs x_0 and x_1), get the messages m_2, m_3, $\dots$, m_N from parties $P_2, \dots, P_N$ (on inputs $x_2, \dots, x_N$). Then it can evaluate the output of the function f on inputs $(x_0, x'_1, x_2, \dots, x_N)$ for any x'_1 (and not just on x_1), because the adversary know all the labels $\{\mu_{i,u}\}_{i,u}$.

We remark that this impossibility relies on the fact that parties $P_1, \dots, P_N$ are not allowed to produce any labels ($\{\mu_{j,u}\}_{j,u}$) from which the adversary only see part of them ($\{\mu_{j,\sigma_j}\}_j$). When compiling Label-Driven MPC into Gage MPC, this restriction will translate to the fact that only P_0 is allowed to make time capsule commitments. If we remove this restriction, we should be able to construct private-inputs Gage MPC using 2-round MPC protocols and similar ideas as the ones used below for $N + 1 = 2$ parties. Such a construction however would likely be quite inefficient.

D Time Capsule Evaluation and Implementation of Gage Auctions

In order to understand how the proposed schemes will perform in practice, in this section we show a proof-of-concept implementation of our constructions. We begin with describing a library we developed to support the POTC interface followed by a demonstration of the library's overhead in terms of computation and bandwidth costs. We then explain how we use POTC and LD-MPC to build a real-world application of Gage MPC, namely, Gage auctions over a blockchain.

D.1 Time Capsule Library Implementation and Evaluation

We build a C++ library implementing the POTC construction found in Appendix B.10. We use SHA-256 for hashing, and the Ivory Runtime Library [53] for elliptic curve operations over Curve25519. We remark that our library provides a generic POTC interface, which could be of an independent interest.

For evaluating the computation and bandwidth overhead of the POTC construction, we conducted microbenchmarks on a server with a 2.6 GHz Intel Core i7 processor and 16GB of RAM running macOS Mojave. We use a message size of 8 bytes to model message types (e.g., labels) encountered in our LD-MPC construction in C.1. We also use $k = 8$ seeds and $\lambda^* = 25$ (See 2.2) unless otherwise noted. We chose these values for the parameters as they provide a reasonable trade-off between the security and difficulty of opening capsules. Each of the tested operations was called 100 times.

Computation cost. Table 2 shows the processing time needed to perform each operation within our construction. As shown, all operations (other than TC.ForceOpen, which has a configurable difficulty, and hence, computation cost) can be computed at a fast rate. In particular, our modest server can generate 500 capsules/sec, verify the opening of 10,000 capsules/sec, generate correctness proofs for 250 capsules/sec, and verify 250 proofs/sec.

Table 2. Computation cost of POTC.

Operation	**Average Time (sec)**
TC.Com	$0.002 \pm 1 \times 10^{-4}$
TC.DVer	$0.0001 \pm 2 \times 10^{-6}$
TC.ForceOpen	1004.6 ± 159
TC.Prove	0.004 ± 10^{-4}
TC.PVer	0.004 ± 10^{-4}

For TC.ForceOpen, we show how the computation cost (or difficulty) to open a capsule varies with λ^*. The results are found in Figure 7. As expected, larger λ^* means more computation is needed to force open the capsule (this is represented by the longer time since we do not exploit any parallelism here). This parameter can be configured in a way that achieves the desired level of hardness.

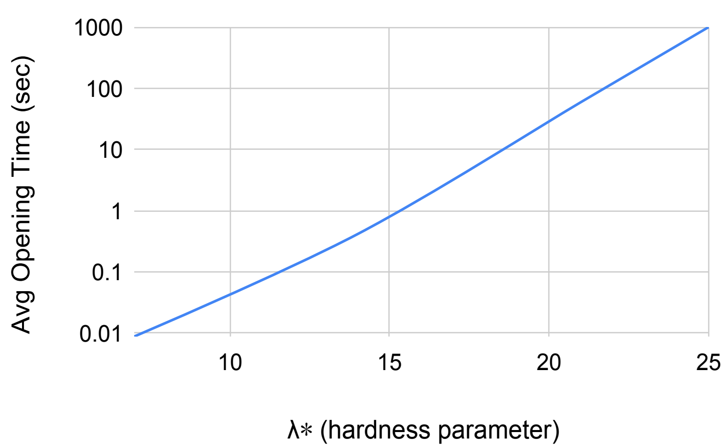

Fig. 7. Impact of capsule difficulty (i.e., λ^*) on the average opening time. Note the y-axis uses a logarithmic scale to capture the exponential relationship.

Finally, we show the bandwidth overhead of our POTC construction from Appendix B.10 in Table 3, which also represents the storage cost. For a message size of 8 bytes, the commitment size is 346 bytes, and the proof size is 96 bytes. While the size of the commitment increases linearly with the message size and number of seeds, the size of the proof is independent with respect to these parameters. These experiments illustrate that the runtime and communication overheads of these operations are minimal relative to the network and compute costs of a blockchain environment.

Table 3. Bandwidth cost of POTC. The size of a commitment and proof construction can be computed from the sizes of these elements as detailed in Section C.1.

Element	Size (bytes)
$c_{1,i}$	32
c_2	8
c_3	33
c_4	33
K	32
ch	32
z	32

D.2 Using Time Capsules for Gage Auctions over the Blockchain

In this section, we describe our proof-of-concept implementation of one application of Gage MPC, namely, auctions. We build a prototype by using the Parity version of the Ethereum Virtual Machine (EVM). We implement both the simple scheme from Section 4 and the one based on Section C.1 (we focus on the latter in the following description). We use the Ivory library [53] to implement our LD-MPC construction found in Section C.1, which we modify to integrate our label-based time capsules in the garbling process that sellers use to prepare their offers that will be posted on the Ethereum blockchain.

Towards this end, and to allow posting offers and the monetary incentives of opening the time capsules, we developed a smart contract for the auction functionality. The smart contract state is initialized to the encoding of the garbled circuit with its time capsules by the seller. Our smart contract interface then allows buyers to post public bids that will specify which time capsules in the underlying GaTC should be opened. It also implement routines that can be invoked to submit proofs of the time capsule opening, which the contract verifies with ($\mathsf{TC.PVer}$) using a new EVM precompile instruction. Upon verifying the opening proof, the smart contract pays the opening party the collateral, which the seller locked as a deposit when creating the contract.

After obtaining the opening of all the required times capsules, the smart contract can evaluate the auction by evaluating the offer garbled circuit using the opened labels. In the case of a match, the smart contract closes the auction, recording the outcome so that settlement can be performed off-chain. We remark that this implementation does not modify the underlying blockchain architecture or consensus protocol since all added functions are at the smart contract level apart from the new precompile that we built. Hence, the same framework can be imported to any other blockchain that supports smart contracts and precompiles.

Our smart contract prototype does not yet handle the nominal path. However, we plan to fully implement this feature when we open-source our library and smart contract prototype for the community soon.