SAPIENZA
UNIVERSITÀ DI ROMA

New Perspectives in Multi-Party Computation: Low Round Complexity from New Assumptions, Financial Fairness and Public Verifiability

Ingegneria dell'Informazione, Informatica e Statistica

Dottorato di Ricerca in Informatica – XXXIII Ciclo

Candidate

Daniele Friolo
ID number 1346443

Thesis Advisor

Prof. Daniele Venturi

June 2021

Thesis defended on 08/07/2021
in front of a Board of Examiners composed by:

Prof. Maurizio A. Bonuccelli (chairman)

Department of Computer Science
University of Pisa

Prof. Dario Catalano

Department of Computer Science and Mathematics (DMI)
University of Catania.

Prof. Andrea Marin

Department of Environmental Science, Computer Science and Statistics
"Ca' Foscari" University, Venice, Italy

External Reviewers:

Prof. Bernardo David
IT University of Copenaghen (ITU), Copenaghen, Denmark

Prof. Rafael Dowsley
Department of Software Systems and Cybersecurity
Monash University, Melbourne, Australia

New Perspectives in Multi-Party Computation: Low Round Complexity from New Assumptions, Financial Fairness and Public Verifiability
Ph.D. thesis. Sapienza – University of Rome

This thesis has been typeset by LaTeX and the Sapthesis class.

Author's email: friolo@di.uniroma1.it

A mio padre

Abstract

Research in Multi-Party Computation is constantly evolving over the years. Starting from the very first result by Yao in 1982, to serve new and more practical scenarios, a lot of different protocols with stronger security properties have been introduced and proven for several assumptions.

For some functionalities, properties like public verifiability, fairness and round-optimality can be considered nowadays a minimal set of assumption to consider an MPC protocol practical. Asynchrony, in the sense that different parties should be able to join a protocol at different times, is fundamental for applications like decentralized lotteries, where the protocol execution can last even days. In such case, due to the involvement of monetary payments, parties must also be aware of what happens to their pockets when such protocols are run. In particular, they must be sure that the execution of a certain class of protocols is financially sustainable. We list below our three contributions to the thesis.

We firstly introduce a new theoretical result, showing how to achieve low round MPC from new assumptions. In particular, we show how to construct maliciously secure oblivious transfer (M-OT) from a mild strengthening of key agreement (KA) which we call *strongly uniform* KA (SU-KA), where the latter roughly means that the messages sent by one party are computationally close to uniform, even if the other party is malicious. Our transformation is black-box, almost round preserving (adding only a constant overhead of two rounds), and achieves standard simulation-based security in the plain model.

As we show, 2-round SU-KA can be realized from cryptographic assumptions such as low-noise LPN, high-noise LWE, Subset Sum, DDH, CDH and RSA—all with polynomial hardness—thus yielding a black-box construction of fully-simulatable, round-optimal, M-OT from the same set of assumptions (some of which were not known before).

By invoking a recent result of Benhamouda and Lin (EUROCRYPT 2017), we also obtain (non-black-box) 5-round maliciously secure MPC in the plain model, from the same assumptions.

Our second and third contributions are focused on the concrete application of MPC protocols achieving the aforementioned properties in real-world scenarios. In applications like decentralized lotteries, decentralized payment mechanisms like blockchains relying on smart contracts can be considered a powerful tool to enforce the correct behavior of cheating players with the aid of monetary incentives or punishments. In fact, a weaker version of fairness called *fairness with penalties*, firstly introduced in the lottery protocol of Andrychowicz et al. (S&P '14) and then formally defined by Bentov et al. (CRYPTO'14), can be used to ensure that corrupted players are incentivized to reveal the output to honest players. This can be done successfully through Bitcoin scripts or Ethereum smart contracts.

In our second contribution, we consider executions of smart contracts on forking blockchains (e.g., Ethereum) and study security and delay issues due to forks. As security notion for modeling executions of smart contracts, we focus on MPC. In particular, we consider on-chain MPC executions with the aid of smart contracts. The classical double-spending problem tells us that messages of the MPC protocol

should be confirmed on-chain before playing the next ones, thus slowing down the entire execution.

This contribution consists of two results:

- For the concrete case of fairly tossing multiple coins with penalties, we notice that the lottery protocol of Andrychowicz et al. becomes insecure if players do not wait for the confirmations of several transactions. In addition, we present a smart contract that instead retains security even when all honest players immediately answer to transactions appearing on-chain. We analyze the performance using Ethereum as testbed.

- We design a compiler that takes any "digital and universally composable" MPC protocol (with or without honest majority), and transforms it into another one (for the same task and same setup) which maintains security even if all messages are played on-chain without delays. The special requirements on the starting protocol mean that messages consists only of bits (e.g., no hardware token is sent) and security holds also in the presence of other protocols. We further show that our compiler satisfies fairness with penalties as long as honest players only wait for confirmations once.

By reducing the number of confirmations, our protocols can be significantly faster than natural constructions, maintaining at the same time public verifiability, asynchrony (obtained by making the parties posting messages to the blockchain via smart contracts), and fairness with penalties.

As a third contribution, we survey the state-of-the-art blockchain based penalty protocols (i.e achieving fairness with penalties) and pioneer another type of fairness, *financial fairness*, that is closer to the real-world valuation of financial transactions. Intuitively, a penalty protocol is financially fair if the *net present cost of participation* of honest parties—*i.e.*, the difference between the total value of cash inflows and the total value of cash outflows at the end of the protocol, weighted by the relative discount rate—is the same, even when some parties cheat.

Then, we show that the ladder protocol (CRYPTO'14), and its variants (CCS'15 and CCS'16), fail to achieve financial fairness both in theory and in practice, while the penalty protocols of Kumaresan and Bentov (CCS'14) and Baum, David and Dowsley (FC'20) are financially fair. Moreover, it can be inferred that the fair with penalties extension of the generic compiler presented in our second contribution, based on CCS'14, is financially fair. Hence, our compiler is also financially sustainable.

Contents

Chapter 1

Introduction

In *Multi-Party Computation* a set of mutually distrusted parties want to jointly compute a function without revealing their inputs to each other.

In MPC, a party cannot learn any other information from the other participants besides her input and the final output. MPC can be modeled using the *Real/Ideal* world paradigm. In this paradigm, we assume the existence of an uncorruptible trusted third party communicating with perfectly secure channels with all the protocol participants. When interacting with such trusted party (also called functionality), protocol players handle their inputs to him. Then, the functionality calculates the expected output for each participant and sends it back to them. Instead, in the real world such a trusted party does not exist, and parties can only communicate with each other via unreliable channels. The security definition states that an MPC protocol is secure if and only if an external judge can distinguish between an ideal world and a real execution with a negligible probability.

MPC was firstly proposed by Yao's garbled circuit [Yao82, Yao86], and subsequently [GMW87] and [BGW88], relying on relying on publicly verifiable secret sharing schemes. As then showed by Kilian [Kil88], MPC can be obtained in a blackbox way from a powerful primitive called *Oblivious Transfer* [EGL82].

Recently, MPC is becoming more and more important due to the digitalization of legal procedures like voting and auctions, in which MPC can be applied in a direct manner [AOZZ15, BCD$^+$09]. Moreover, 2-PC/MPC is also a powerful tool that can be used for the most diverse cryptographic applications, like decentralized secure storage, threshold signatures (used nowadays in blockchains to sign transactions securely or to generate anonymous credentials), decentralized computation of MACs, and many others. In any case, the simple fact that MPC can be computable for any polynomially bounded circuit, together with the fact that any functionality can be used as a blackbox sub-functionality in any protocol description, gives to this primitive an immense versatility.

Many MPC properties have been defined in literature: semi-honest vs. dishonest execution, honest majority vs. dishonest majority, security with abort vs. fairness/guaranteed output delivery, synchrony vs. asynchrony. Any of the listed properties can be useful depending on the application scenario.

For our purposes, let us consider decentralized lotteries as an example. Due to the low trust in the authorities, the increasing request for decentralization of

procedures encouraged the creation of large-scale MPC protocols that can be run even by a standard home PC. In the case of lotteries, an MPC protocol must fulfill at least the following properties:

- Asynchrony, in the sense that parties should be able to join the protocol at different times.
- Public verifiability.
- Fairness.
- Round-optimality.
- Financial Sustainability.

A lottery could last even days. A fully synchronized protocol where players must be online all the time is definitely impractical in such a scenario. The protocol transcript should always be publicly accessible and verifiable in the case that some controversy pops up. Round-optimality is also really important: the communication channel can heavily impact the overall protocol execution latency. Moreover, connections of standard end-users can be unreliable. A player could be online to participate for a couple of rounds and then disappear for a while due to connection issues, thus badly impacting the execution time. Fairness guarantees are also crucial: in security with abort an adversary, after learning not being the lottery winner, can abort the execution by denying the honest players to learn the outcome, thus trying to win the lottery in some future execution. In order to give a successful payback to honest parties, or to punish cheating adversaries, decentralized payments systems based on smart contracts can be helpful (as we will see, they can be used even to guarantee a weaker form of fairness based on game-theoretic assumptions, called fairness with penalties). Due to the involvement of monetary payments, investigating whether a protocol execution can be financially sustainable for a participating party is important for such protocols' real-life applicability.

We will start by introducing, in Section 1.1 a theoretical result showing round-optimal OT protocols (that can be compiled to low round MPC) from new assumptions, like Low Noise LPN, High Noise LWE, DQR, DCR (all with polynomial hardness).

As a second result (introduced in Section 1.3), we show how asynchrony, public verifiability, and a weaker version of fairness based on monetary incentives can be achieved by relying on a forking blockchain as a communication channel (a brief introduction with a little discussion on how public verifiability, asynchrony, and fairness can be achieved with the aid of a blockchain is given in Section 1.2). Moreover, we propose a parallel coin tossing protocol and a generic MPC compiler that drastically reduces the number of blockchain rounds needed to complete the execution of such protocols w.r.t a naive execution relying on message confirmation. This result holds when an optimistic setting is assumed (i.e. that the network is in a good state and no player behaves dishonestly). We show that our construction does not compromise any security guarantee.

In Section 1.4 we show how to analyze penalty protocols achieving fairness with penalties known in literature under a financial lens. In particular, we show that

fairness is not only cryptographic, but also financial. Depending on the structure of the protocol or the deposit quantity that must be deposited to participate, a party would be more or less incentivized to join. By relying on our analysis, it can be inferred that our protocols introduced in 1.3 are also financially fair.

1.1 Low Round Complexity from New Assumption: MPC from Round-Optimal OT

Roughly, (1-$out-of$-2) Oblivious Transfer is a functionality taking as inputs two messages m_0, m_1 from a sender and a choice bit b from a receiver. When receiving the sender's two messages, stores them in the local storage. When the choice bit b is sent by the receiver, the functionality answers back to the receiver with the message m_b. An OT protocol, in order to realize such functionality, must guarantee both seller and receiver privacy, meaning that the receiver cannot learn any information about the message m_{1-b} and the sender can not learn which is the message the receiver asked for (i.e. the choice bit) [Rab81, EGL82].

Following previous work, we concentrate on OT protocols that are *"fully simulatable"*, informally meaning that both sender and receiver can be successfully replaced by a simulator knowing no information on the sender's or receiver's inputs. Different flavours of simulation are known, depending on which adversarial model is considered: in our work, we are in the sequential composability framework (cfr Section 2.7.1).

Suprisingly, OT turned out to be sufficient for constructing secure multi-party computation (MPC) for *arbitrary* functionalities [Yao82, Yao86, Kil88, IPS08, IKO+11, BL18, GS18]. Therefore, constructing OT has been an important objective and received much attention.

Nevertheless, previous constructions of fully-simulatable OT suffer from diverse shortcomings (cf. also Section 1.1.2):

1. They require *trusted setup*, or are based on *random oracles* (as, e.g., in [JS07, PVW08]);
2. They have *high round complexity* (as, e.g., in [Hai08]), while the optimal number of rounds would be 4 in the plain model without trusted setup [IKO+11, GMPP16];
3. They are *non-black-box*, in that they are obtained by generically transforming semi-honestly secure OT (SH-OT)—which in turn can be constructed from special types of PKE [GKM+00]—to fully-simulatable OT via (possibly interactive) zero-knowledge proofs (*á la* GMW [GMW91]);
4. They are tailored to *specific hardness assumptions* (as, e.g., in [Lin08, BD18]).

One exception is the work of Ostrovsky, Richelson and Scafuro [ORS15], that provide a black-box construction of 4-round, fully-simulatable OT in the plain model from *certified trapdoor permutations* (TDPs) [BY92, LMRS04, CL18], which in turn can be instantiated from the RSA assumption under some parameter regimes [KKM12, CL18].

This draws our focus to the question:

Can we obtain 4-round, fully-simulatable OT in a black-box way from minimal assumptions, without assuming trusted setup or relying on random oracles?

1.1.1 Our contribution

We give a positive answer to the above question by leveraging a certain type of key agreement (KA) protocols, which intuitively allow two parties to establish a secure channel in the presence of an eavesdropper. The influential work by Impagliazzo and Rudich [IR88] showed a (black-box) separation between secret-key cryptography and public-key cryptography and KA. Ever since, it is common sense that public-key encryption (PKE) requires stronger assumptions than the existence of one-way functions, and thus secure KA is the weakest assumption from which public-key cryptography can be obtained. More recent research efforts have only provided further confidence in this conviction [GMMM18].

More in details, our main contribution is a construction of fully-simulatable OT (a.k.a. *maliciously secure* OT, or M-OT) from a seemingly mild strengthening of KA protocols, which we term *strongly uniform* (SU); our protocol is fully *black-box* and essentially *round-preserving*, adding only a constant overhead of at most two rounds. In particular, we show that:

Theorem 1 (Informal)**.** *For any odd $r \in \mathbb{N}$, with $r > 1$, there is a black-box construction of a $r + 1$-round, fully-simulatable oblivious transfer protocol in the plain model, from any r-round strongly uniform key agreement protocol.*

Since, as we show, 2-round and 3-round SU-KA can be instantiated from several assumptions, including low-noise (ring) LPN, high-noise (ring) LWE, Subset Sum, CDH, DDH, and RSA—all with polynomial hardness—a consequence of our result is that we obtain round-optimal M-OT in the plain model under the same set of assumptions (in a black-box way). In particular, this yields the *first* such protocols from LPN, LWE (with modulus noise ratio $\sqrt{n}$), CDH, and Subset Sum.[1] Note that our LWE parameter setting relates to an approximation factor of $n^{1.5}$ for SIVP in lattices of dimension n [Reg05], which is the weakest LWE assumption known to imply PKE.

In our construction, we use a special kind of *"commit-and-open"* protocols which were implicitly used in previous works [Kil92, ORS15]. As a conceptual contribution, we formalize their security properties, which allows for a more modular presentation and security analysis.

1.1.2 Related Work

Maliciously secure OT. Jarecki and Shamtikov [JS07], and Peikert, Vaikuntanathan, and Waters [PVW08], show how to construct 2-round M-OT in the common reference string model.

[1] We can also base our construction on Factoring when relying on the hardness of CDH over the group of signed quadratic residues [HK09], but this requires a trusted setup of this group which is based on a Blum integer.

A result by Haitner *et al.* [Hai08, HIK+11] gives a black-box construction of M-OT from SH-OT. While being based on weaker assumptions (i.e., plain SH-OT instead of Strongly Uniform SH-OT), assuming the starting OT protocol has round complexity r, the final protocol requires 4 additional rounds for obtaining an intermediate security flavor known as "defensible privacy", plus 4 rounds for cut and choose, plus 2 times the number of rounds required for running coin tossing, plus a final round to conclude the protocol. Assuming coin tossing can be done in 5 rounds [KO04], the total accounts to $r + 19$ rounds, and thus yields 21 rounds by setting $r = 2$.

Lindell [Lin08] gives constructions of M-OT with 7 rounds, under the DDH assumption, the Nth residuosity assumption, and the assumption that homomorphic PKE exists. Camenish, Neven, and shelat [CNS07], and Green and Hohenberger [GH07], construct M-OT protocols, some of which even achieve adaptive security, using computational assumptions over bilinear groups.

There are also several efficient protocols for OT that guarantee only privacy (but not simulatability) in the presence of malicious adversaries, see, e.g. [KO97, NP01, AIR01, Kal05, BD18].

Round-optimal MPC. Katz and Ostrovsky [KO04] proved that 5 rounds are necessary and sufficient for realizing general-purpose two-party protocols, without assuming a simultaneous broadcast channel (where the parties are allowed to send each other messages in the same round). Their result was later extended by Garg *et al.* [GMPP16] who showed that, assuming simultaneous broadcast, 4 rounds are optimal for general-purpose MPC. Together with a result by Ishai *et al.* [IKO+11]—yielding *non-interactive* maliciously secure two-party computation for arbitrary functionalities, in the OT-hybrid model—the latter implies that 4 rounds are optimal for constructing fully-simulatable M-OT in the plain model.

Ciampi *et al.* [COSV17b] construct a special type of 4-round M-OT assuming certified TDPs,[2] and show how to apply it in order to obtain (fully black-box) 4-round two-party computation with simultaneous broadcast. In a companion paper [COSV17a], the same authors further give a 4-round MPC protocol for the specific case of multi-party coin-tossing.

1.2 Blockchain as a Tool to Enforce Security Properties

The rise of blockchains[3] is progressively changing the way transactions are executed over the Internet. Indeed, the traditional client-server paradigm turns out to be insufficient when many parties want to perform a distributed computation, especially in cases where features like public verifiability and automatic punishment are desired. Blockchains through the execution of smart contracts naturally allow many players to perform a joint computation, even when they are not simultaneously online; moreover, they allow to publicly check the actions of all players[4] and enforce

[2] They also claim [COSV17b, Footnote 3] that their OT protocol can be instantiated using PKE with special properties, however no proof of this fact is provided.

[3] We use the terms "blockchain" and "distributed ledger" interchangeably.

[4] We will often use the two terms "party" and "player" as synonyms.

a proper behavior through financial punishments.

1.2.1 Blockchain Overview

The first conceptualization of a Blockchain was given in the famous "Bitcoin: A Peer-to-Peer Electronic Cash System" by Satoshi Nakamoto [Nak19]. Bitcoin, as well as all the other well-known blockchain systems (e.g. Ethereum, Algorand, Ripple, Cardano, etc) is also called *public ledger*. Each message stored in the ledger can be represented in the form of a transaction moving some cryptocurrency (we will often refer to it as *coins*) from a wallet A to a wallet B (or from a set of wallets to another set of wallets). An important feature of these types of transactions is that they can also store additional information like raw data or scripts. The term "blockchain" stems from the fact that the data structure is organized in an ordered chain of blocks, each containing a set of transactions. When a party owning a wallet sends a transaction to the blockchain, he can be sure that, after a fixed period of time, if the transaction was indeed valid, it will eventually appear to the ledger (*Liveness* property). Moreover, it can be sure that once a transaction has been issued to the ledger at some block, if it ends "deep" enough in the chain, it will be included in every honest player's blockchain. It means that if you cut the last k blocks, such transaction cannot be changed unless the honest nodes are less than 51% (*Persistence property).* In Chapter 2 we will formally describe the liveness property [GKL15] and a sub-property of persistence, called *chain consistency* [GG17, PS17], informally stating that a transaction appearing k blocks deep in the blockchain, will be part of the common prefix (obtained by cutting the last k blocks) of all the honest nodes.

Bitcoin Proof-of-Work and the Blockchain Model. In Bitcoin, a set of nodes called "miners" must use their computational power to solve cryptographic puzzles needed to create new blocks. When a miner succeeds in solving the puzzle, a reward, called *coin-base transaction* is assigned to him. This particular type of transaction incentivizes the miners to keep the mining procedure alive [5]. Solving such puzzles is a non-trivial task. To mine a new block, a Bitcoin miner has to provide, together with the block, a *proof-of-work* demonstrating that he used enough computational power during the mining procedure (and therefore spent enough time and money in electricity and computational supplies). In bitcoin, such puzzles can be solved by computing the pre-image of a hash function whose output must be less than a fixed value. As the chain grows, this value decreases, making the proof-of-work more expensive in terms of computation.

Many different blockchain models have been proposed in the literature. The stronger model known until assumes the existence of a global functionality [BGM+18] that can be accessed by any protocol. It is proven secure in a hybrid model, assuming that any entity can query such functionality. Unfortunately, such a model is too much stronger for our purposes. To guarantee the prescribed properties, this global functionality assumes only the common prefix of honest nodes, not the entire

[5] In [GG17, PSS17] such incentive guarantees the *chain growth* property, informally stating that if a transaction has been issued to the network, it will be eventually included in some block. It can be seen as a sub-property of Liveness

subchain owned by each miner. Since updating this functionality with this such a feature increases its complexity and dramatically changes the way the blockchain players can access it, we decided to use a weaker model based on an external but not global functionality. In [GG17, PSS17] they enrich the MPC model with the possibility for any entity (adversary, honest players, simulator) to access in a black-box way to an interface Γ. Such interface offers the players a set of algorithms to communicate with the blockchain that can be queried from any entity to update the local state of the blockchain, extract the list of blocks (i.e., the blockchain) from such state, and broadcast a new transaction into the blockchain network. This model gives us enough flexibility to deal with forks and smoothly prove our result. A formal treatment of this model, that will be used in our protocols in Chapter 4, is given in Section 2.8.

1.2.2 Overcoming Impossibility Results: Fairness

An important property in MPC is the so-called *cryptographic fairness*, which intuitively says that corrupted parties learn the output only if honest parties learn it as well. Unfortunately, without assuming *honest majority* (*e.g.*, for two parties) there are concrete examples of functions for which cryptographic fairness is impossible to achieve [Cle86].

To circumvent this impossibility, several solutions have been proposed: restricted functionalities [GHKL11], partial fairness [GK12, MNS16], gradual release protocols [GMPY11], optimistic models [CC00], and incentivized computation [ALZ13]. A recent trend is to guarantee cryptographic fairness via monetary compensation (a.k.a. *cryptographic fairness with penalties*[6]).

This approach gained momentum as decentralized payment systems (*e.g.*, Bitcoin and Ethereum) offer a convenient way to realize such *penalty protocols* [ADMM14, BK14, KB14, KMB15a, KVV16, KB16, KZZ16, BKM17, DDL18]. The main idea is that each party can publish a transaction containing a time-locked deposit which can be redeemed by honest players in case of malicious aborts during a protocol run. On the other hand, if no abort happens, a deposit owner can redeem the corresponding transaction by showing evidence of having completed the protocol.

1.2.3 Public Verifiability and Asynchrony

Blockchains offer public verifiability of distributed computations, in the sense that in case of dispute everyone can verify what happened and when. Moreover, smart contracts can automatically punish whoever violates some a-priori established rules. Clearly, the above advantages are useful also when players are running a privacy-preserving computation, in the form of a multi-party computation (MPC) protocol.

A popular example of MPC that can benefit from a blockchain is e-voting, since public verifiability is an important property of remote elections and several systems rely on a bulletin board that can be instantiated with a blockchain. Another well known example is the one illustrated by Andrychowicz et al. [ADMM14, ADMM16] mentioned in the previous subsection who, despite the very limited expressive

[6] In Chapter 4 the term "cryptographic" is omitted.

power of Bitcoin transactions, have shown how to use blockchains to obtain fairness through penalties to MPC protocols with dishonest majority.

Note that executing an MPC protocol on-chain[7] allows players not to be online all at the same time. Moreover, differently from protocols running on a TCP/IP WAN where players must know each other's IP address beforehand, with the aid of a ledger any player can join a protocol execution by just reading[8] a transaction containing the required information (e.g., the functionality, the minimum number of parties, or any other identifying information).

1.3 Forking Blockchains and Hasty Players

Forks, finality and double spending. Typical blockchains experience some delays before a transaction can be considered confirmed. Indeed, a large part of the most used blockchains consists of a list of blocks that can temporary fork. In such cases, fork-resolution mechanisms decide which branch is eventually part of the list of blocks and which one is discarded, at the price of cutting off some transactions that for some time have appeared on the blockchain. These finality limitations generate delays and uncertainty, and a significant effort has been made recently to design blockchains with better finality [MMNT19, BG17, PS17, PS18, GHM+17, CPS18].

It is well known that the existence of transactions that appear and then disappear from a blockchain is the source of the (in)famous double-spending attack. The solution to the double spending problem is pretty harsh: the receiver of a payment will have to wait long time (i.e., until the transaction is confirmed and becomes irreversible) before taking future actions. Obviously, this can be problematic when an entire process consists of many sequential transactions and the confirmation time is long.

The double spending problem does not seem to extend to the case where another on-chain transaction is connected to the payment transaction. Indeed, in this case, if as a consequence of a fork the payment transaction disappears, then the connected transaction disappears too. This chaining of transactions related to the same process can be easily implemented through smart contracts.

Insecurity of smart contracts with hasty players. Since transactions are not immediately confirmed in a forking blockchain, the full execution of a smart contract with multiple sequential transactions might take too long. It would thus be natural to speed up the execution of smart contracts by playing messages immediately. Indeed, as mentioned above, by appropriately chaining the transactions of a smart contract, attacks exploiting the cancellation of a transaction like in the double-spending attack are not effective,[9] and therefore playing immediately without waiting confirmations could be a valid option.

[7] Running an MPC protocol with the aid of a blockchain simply means that the players exchange messages using the blockchain.

[8] Blockchain identifiers are usually public pseudonyms not necessarily correlated with the real user identities. This feature offers some privacy compared to IP addresses.

[9] Since we are considering protocols running entirely on-chain, double spending attacks can not be exploited to avoid the payment of some off-chain service.

However, we notice that forks can help an adversary to mount more subtle attacks. For example, let us consider a smart contract executed by two players, Alice and Bob, willing to establish jointly a random string: 1) Alice starts the protocol by sending to the smart contract a commitment to a random string r_1; 2) Bob sends a random string r_2 to the smart contract; 3) Alice then opens the commitment, and if the opening is valid the common string is defined to be $r = r_1 \oplus r_2$. For concreteness, say that Alice is honest and Bob is corrupted, and assume that a fork happens after Alice already sent the commitment. If Bob runs the protocol honestly on the first branch, he gets to see Alice's opening, and thus he can completely bias the output on the other branch by just sending $r_2' = r' \oplus r_1$ to the smart contract, for any value r' of his choice. The above scenario can be a serious threat for integrity of data and even confidentiality in other protocols. This motivating example clearly shows that, unless one has proven some kind of resilience to forks, it is certainly preferable to always wait that transactions are confirmed, at the price of having very slow executions of a smart contract. Such slowness could be unacceptable in some applications.

The above features, and the dilemma about playing immediately risking security or waiting for confirmation making the entire process very slow, motivate our work aiming at obtaining smart contracts for fast/fair/secure/publicly-verifiable MPC protocols on forking blockchains.

We remark that at least some of the aforementioned advantages provided by our constructions do not come already from the use of payment channels. Consider for instance payment channels allowing to run a computation in large part off-chain. The use of similar channels for MPC would require players to be simultaneously online with point-to-point connections, therefore suffering of the issues discussed above.

1.3.1 Our Contributions

Defining on-chain MPC with hasty players. We model the execution of a smart contract through transactions sent by different players as a computation involving multiple parties, and therefore when considering "security" of such computations we naturally refer to secure MPC.

Intuitively, a player is called *non-hasty* if she always waits that the previous messages are confirmed on the blockchain before sending the next one. On the other hand, a *hasty player* sends her next message by just looking at her current view of the blockchain (without pruning blocks).

Apart from these changes, security is defined similarly as in the standard real-ideal world paradigm. Intuitively, in a protocol running with hasty players, block confirmation is not needed. However, if parties wants to keep a natural blockchain feature like public verifiability, the last message exchanged in the protocol must be necessary confirmed. Throughout the paper, when we talk about no confirmation we implicitly assume the last message is confirmed for public verifiability guarantees.

The definition of security in the presence of hasty players has importance in forking blockchains, in which miners can discard non-confirmed blocks, achieving consensus on other blocks. Our definition applies to forking sidechains too.

Fair lottery with penalties and fully hasty players. In Andrychowicz et al. [ADMM14, ADMM16] and in Kumaresan et al. [BK14, KB14] it was shown how to obtain fairness (i.e., the adversary should be discouraged from avoiding that honest players learn the output after he gets it) through penalties. The idea is that a player should deposit some coins of the underlying cryptocurrency and the smart contract should return the coins back only in case the player completes correctly the execution of the protocol defined by the smart contract.

In light of the negative result by Cleve [Cle86] on achieving fairness without honest majority, we will also consider fairness with penalties. Recall that we are planning to do so still admitting that the blockchain could fork and trying to obtain fast executions avoiding as much as possible to wait for confirmations of transactions.

We analyze a variant of the attack described earlier that can be applied to a smart contract based on Andrychowicz *et al.* [ADMM14, ADMM16] protocol for securely realizing multi-party lotteries[10] The main difference with the toy example from above is that in their work each player commits to a random value r_i between 1 and n (where n is the total number of participants to the lottery), and then, after all the commitments have been opened, the winner of the lottery is defined to be the player $w = r_1 + \ldots + r_n \pmod{n} + 1$. An appealing feature of this protocol is that it achieves fairness with penalties: if a malicious player aborts the protocol (e.g., it does not open the commitment before a certain time bound), then a previously deposited amount of coins is automatically transferred to the honest players (i.e., to those that correctly opened the commitment on time).

We note that in the protocol of Andrychowicz et al. it is vital that players are non-hasty and therefore post new transactions only after the previous ones are already confirmed on the blockchain. Indeed, in the presence of hasty players, a malicious party can commit to a value r_i such that $\sum_i r_i \pmod{n} + 1 = i$, assuming that all players already opened the commitments on a minor branch of a fork. As our main contribution, in Section 4.3, we circumvent the limitations of [ADMM14, ADMM16], and present a smart contract that implements the lottery functionality[11] remaining secure even in the presence of hasty players. Fairness with penalties can be added without affecting the efficiency of the protocol. In fact, the smart contract we design is more general, in that it allows the players to establish a common, uniformly random, string (which in turn allows to run a lottery). When referring to our protocol depending on the context we will sometimes say lottery protocol and sometimes parallel coin-tossing protocol.

The main idea in our construction consists of combining unique signatures [Lys02] and random oracles (similarly to constructions of verifiable random functions) as follows: first of all, players compute unique signatures on input the concatenation of the ordered sequence of their public keys. Notice that as long as at least one player is honest, we have a long string that no PPT player could predict when selecting his

[10] Protocols of [ADMM14, ADMM16] is based on Bitcoin, but this makes no difference for our attack.

[11] We specify that our smart contract implements a parallel coin-tossing protocol. In some cases, we say that our smart contract implements a lottery protocol since we are interested in comparing our protocol with the lottery protocol of Andrychowicz et al. We remark that the output of a coin-tossing protocol can be used to compute a lottery winner.

public key. Then this long string is given in input to a random oracle, returning a uniformly distributed string as an output. The simulator will program the random oracle to force in the simulation the same random string obtained in the ideal-world execution.

There is still an attack that can be mounted. Assume that in the presence of a fork the entire protocol is executed in a branch. The adversary could take advantage of the output in one branch to decide to play the same first round or a different first round in the other branch biasing successfully the distribution of the output. To circumvent this problem, we make executions in different branches completely independent by also passing a branch id as input to the unique signature evaluation procedure. As branch id we take the hash of the block containing the last deposit. Therefore, when a protocol is entirely run in a branch, we have that the two branch ids are different and thus there is no point in adaptively choosing the same or a different message in another branch. Indeed, in any case, the outputs in different branches will be completely independent. In order to deal with multiple executions of the real-world protocol in different branches, we will also have a simulator that will play multiple times in the ideal world. Since the output of the protocol is a random string, it can be then used in many applications, not only to run a multi-party lottery. Note that our protocol is around 50% more efficient than the lottery of Andrychowicz et al. Let's say that t is the number of blocks needed for transaction confirmation, then our lottery protocol can be run by using only $t+1$ blocks, whereas Andrychowicz et al. requires $2 \cdot t$ blocks to be completed.

Notice that this result makes no use of finality of transactions on a blockchain except from the one needed for calculating the output. The protocol can be run in the presence of fully hasty players, and is therefore very efficient.

We stress that we consider the adversary as a player that tries to exploit the existence of forks in order to bias the output of the smart contract. We are not modelling the adversary of the smart contract as a player that has control over forks, deciding which branch will eventually be discarded and which one will become permanently part of the blockchain. Obviously, a powerful adversary that has control over the forks can always play the protocol with a different input on each branch to then select the one that produced the output that she likes the most. This is unavoidable when there is little use of finality of transactions. Nevertheless, notice that in many cases this is not a problem. Indeed think of the need of establishing a random string to then use it as first round of a statistically hiding commitment scheme or as common reference string for a non-interactive zero-knowledge proof. In such scenarios the adversary can freely select a random string from any polynomially large set of randomly sampled strings without compromising any security. In other cases like playing bingo, the fact that the adversary can decide the string out of several candidates can be an issue.

General-purpose MPC with hasty players and fairness with penalties. Having motivated the problem of running MPC protocols on forking blockchains, we show a general compiler to obtain smart contracts that implements ideal multi-party functionalities retaining security in the presence of forks and allowing players

to be hasty.[12]

When engaging in the compiled MPC protocol, a party A as soon as reads a message $m_{B,A}^{(r)}$ of round r posted from party B (and directed to A) in the blockchain, decides to immediately post his next message $m_{A,B}^{(r+1)}$ directed to B without waiting that the message $m_{B,A}^{(r)}$ is confirmed. Moreover, even in the case that the A's view changes, and an older message $m_{B,A}^{(r')}$ (where $r' < r$) is present in her view, A answers immediately with the message $m_{A,B}^{(r'+1)}$.

In order to preserve security in the presence of forks, our compiler makes sure that, whenever an execution of the MPC protocol is repeated in multiple branches, each honest player protects herself from attacks exploiting forks by refusing to play again a message of the same execution of the protocol in case the blockchain shows a different prefix in the transcript of the execution. Specifically, if on one branch $\mathcal{B}_2$ there is a player B that changes a message $m_{B,A}^{(r)}$ already played in a different branch $\mathcal{B}_1$ with $m'^{(r)}_{B,A} \neq m_{B,A}^{(r)}$, then each honest player (A in our example) that played her message $m_{A,B}^{(r+1)}$ already in $\mathcal{B}_1$ and is asked to play again on input a different $m'^{(r)}_{B,A}$ in $\mathcal{B}_2$, will abort the execution in $\mathcal{B}_2$. Clearly, this strategy forces a unique execution regardless of forks, and thus security holds even in the presence of fully hasty players. An exhaustive description of our compiler and its extension adding fairness with penalties is given in Section 4.4.

1.3.2 Related Work

Following [ADMM14, ADMM16], other works focus on achieving fairness with penalties for different applications of interest, including lotteries [BK14], decentralized poker [KMB15a, BKM17], and general-purpose computation [BK14, KMS+16, KB16, KVV16, BDD20]. In the more recent work of [BDD20] the authors proposed a fair with penalties MPC protocol with increased efficiency of the off-chain phase. In particular, the line of works by Kumaresan et al. relies on an elegant paradigm working in two phases: 1) during the first phase, players run an MPC protocol to obtain the output in hidden form (e.g., a secret sharing of the output); since the output is hidden, such a protocol can be executed off chain, as malicious aborts do not violate fairness; 2) during the second phase, the output is reconstructed in a fair manner on chain. Unfortunately, the security of this paradigm in the presence of hasty players is difficult to assess, as protocols relying on intermediate ideal functionalities (such as the "claim-or-refund" and "multi-lock" functionality [BK14, KB14], or a smart contract functionality [BDD20]), although implementable using Bitcoin or Ethereum, may be insecure when executed with hasty players. Moreover, known results about designing protocols in a hybrid model allowing to make calls to a functionality are applicable only to the classical setting where multiple executions of the same instance of the protocol due to forks are not possible. Also note that

[12] In this work all our positive results consist of on-chain protocols for secure computation that are stand-alone secure, with security preserved under sequential composition. The reason why we do not try to obtain universal composability is that existing notions of universal composability with a ledger [CGJ19] rely on non-forking ledger functionalities and therefore on non-hasty players.

performing a large part of the computation off chain hinders one of the main advantages of blockchain-aided MPC (i.e., public verifiability of the entire process). Our results, in contrast, consider MPC protocols executed completely on-chain through smart contracts.

A different line of works, shows how to perform MPC in the presence of an abstract transaction ledger [KZZ16, GG17, BMTZ17, SSV19, CGJ19], of which Bitcoin and Ethereum are possible implementations. However, such an idealized ledger does not account for the possibility of forks, thus (implicitly) meaning that the players using it are modeled as non-hasty.

Our main contribution is a protocol to jointly generate a random beacon. It is known that there exist protocols suited for blockchains generating random values. A well known implementation is RANDAO [RT]. The smart contract introduced in RANDAO is similar to a smart contract implementation of the Andrychowicz et al. lottery protocol [ADMM16]. As we show in Section 4.3 even this smart contract is subject to attacks in case some party does not wait for the confirmation of the first phase of the protocol. On the contrary, our lottery protocol described in Section 4.3.1 is secure even if parties do not wait for block confirmations.

1.4 Fairness with Penalties under a Financial Lens

An efficient construction for fairness with penalties proposed by Kumaresan, Vaikuntanathan, and Vasudevan [KVV16] requires a number of rounds/transactions linear in the number of parties, and script complexity independent of the function being computed. Unfortunately, the above efficiency comes at the price of controversial assumptions: the players need to engage in an offline MPC protocol (*i.e.*, without relying on the blockchain) for obtaining a "non-committing" encryption [CFGN96] of the output. However, the latter requires to represent the encryption function itself as a circuit, which is problematic since non-committing encryption with good parameters only exists in the random oracle model [Nie02].

An additional concern that is often not discussed in penalty protocols is that the amount of money that should be put into escrow does not matter, nor how long it should stay there, because all parties would be eventually made whole. While true when the deposit d is a symbol in a crypto paper, things differ when d is a noticeable amount in a bank account.

In fact, empirical studies show that people have a strong preference for immediate payments, and receiving the same amount of money later than others is often not acceptable [BIW15]. Even for the wealthy, there is the opportunity cost of not investing it in better endeavors [LVM16]. For example, in a classical experimental study [BRY89], individuals asked to choose between immediate delivery of money and a deferred payment (for amounts ranging from $40 to $5000) exhibited a discount rate close to the official borrowing rate. These results are consistent across countries (*e.g.*, [BRY89] in the US and [AW97] in Germany). Individuals and companies exhibit varying degree of risk aversion [BIW15, LVM16], but they all agree that money paid or received "now" has a greater value than the same amount received or paid "later" [ALR$^+$01], and that small deposits are always preferable to large deposits. In FinTech, where the base chip d to play is a million US$ [Hat09, MNN$^+$18],

the timings and amounts of deposits can make a huge difference in practice.

1.4.1 Our Contribution

As a useful guide to the extant and future literature we provide three important criteria for characterizing penalty protocols: security models and assumptions, protocol efficiency and *financial fairness*. While the first two criteria are classical ones, the third one is a new property we, for the first time, define and motivate below.

Criteria #1: Security Models and Assumptions Following the principles of modern cryptography, a secure protocol should be accompanied with a formal proof of security in a well defined framework. The standard definitions for MPC (with and without penalties) follow the simulation-based paradigm and are reviewed in Section 2.7.1, along with the main assumptions required for proving security.

Criteria #2: Protocol Efficiency The efficiency of penalty protocols over blockchains is typically measured in terms of: (i) the number of transactions that the parties send to the public ledger (relative to the total transaction fees); (ii) the number of rounds of interaction with the public ledger; and (iii) the script complexity, that intuitively corresponds to the public ledger miners' verification load and the space occupied on the public ledger. To measure the script complexity we usually consider the cumulative script size of all the transactions issued by a user through the overall protocol execution. We elaborate more on these efficiency criteria in Section 5.1.1.

Criteria #3: Financial Fairness In a nutshell, a penalty protocol is financially fair if the difference between the total *discounted* value of cash inflows and the total *discounted* value of cash outflows of honest parties at the end of the protocol is the same (even when some parties cheat). In Section 5.2, we discuss the principle of financial fairness at a level of abstraction that captures a large class of penalty protocols implementing monetary compensation via any kind of currency (with or without smart contracts).

After describing the main state-of-the-art penalty protocols in Section 5.3, we do an exhausting comparison using the above criteria. Our results are summarized in Tab. 1.1. More in details, in Section 5.4, we compare penalty protocols in terms of security model and assumptions (Section 5.4.1) and protocol efficiency (Section 5.4.2). In Section 5.5, we instead study financial fairness both on paper and via empirical simulations on the differences in deposits and net present values as the number of parties increases to a level that would be needed for realistic FinTech applications.

From the perspective of security and efficiency, in Section 5.4, we show that the CCS'16 `Compact Ladder` and `Compact PL` achieve script complexity independent on the size of the output of the function being computed, at the cost of giving up to proving provable security security[13]. To highlight possible efficiency trade-offs,

[13] This is due to the fact that such functionality uses an encryption algorithm that needs to invoke the random oracle which cannot be implemented as a circuit.

Table 1.1. Comparing Penalty Protocols

We compare state-of-the-art penalty protocols under the criteria of security models and assumptions, efficiency, and financial fairness. The first five protocols realize non-reactive functionalities, while the last four are for multistage functionalities. Regarding efficiency, a more detailed discussion with a concrete analysis tailored to Bitcoin implementations is given in Section 5.4.

Criteria	**Description**	`Ladder`	`Multi-Lock`	`Insured MPC`	`Compact Ladder`	`Compact Multi-Lock`	`Locked Ladder`	`Amortized Ladder`	`Planted Ladder`	`Compact PL`
#1 *Security (model)*	● Universal Composability ◐ Sequential Composability ○ Not Provably Secure	●	●	●	○	◐	●	●	●	○
#1 *Security (assumptions)*	● Plain Model ◐ Random Oracle Model ○ Not Provably Secure	●	●	◐	○	●	●	●	●	○
#2 *Efficiency* (rounds)	● Constant ◐ Linear in the num. of parties ○ Quadratic in the num. of parties	◐	●	●	◐	●	○	○	◐	◐
#2 *Efficiency* (transactions)	● Linear in the num. of parties ○ Quadratic in the num. of parties	●	○	●	●	○	○	○	●	●
#2 *Efficiency* (complexity)	● Output independent ○ Output dependent	○	○	○	●	●	○	○	○	●
#3 *Financial Fairness*	● Financially fair ○ Financially unfair	○	●	●	○	o●	○	○	○	○

we propose an illustrative protocol (`Compact Multi-Lock`) achieving the same efficiency of `Compact Ladder` under standard assumptions, but giving up on universal composability [Can00, Can20].

Moving to financial fairness, in Section 5.5, we find that the CRYPTO'14 `Ladder` [BK14] and its variants (`Locked Ladder` [KMB15a], `Amortized Ladder` [KB16], `Planted Ladder` [KVV16]) are not financially fair (in any practical case of interest), whereas the CCS'14 `Multi-Lock` protocol by Kumaresan and Bentov [KB14], as well as the FC'20 `Insured MPC` protocol by Baum *et al.* [BDD20], are financially fair. In particular, our analysis shows that the protocols in [BK14, KMB15a, KB16, KVV16] are only financially viable for the "big guy" with deep pockets beyond the money at stake in the protocol. "Small guys" must rush to be first, or participating would be out of reach. Furthermore, the latter happens even in the practical case of *optimistic computation* for honest parties [KB14]: Playing first or last can yield a gap of several basis points (the units for discount rates of financial institutions).

Another surprising finding is that the CCS'15 `Locked Ladder` is better than its "improved" version, the CCS'16 `Planted Ladder`, in terms of financial fairness. As we will show in Section 5.5, the difference of amount of the total deposit between the first and the last player in in `Ladder` (the amount grows with the player's index in such type of protocols) is "tolerable", meaning that even if some player do not rush to deposit, participating to the protocol can still be worthy. Importantly, these negative results hold regardless of which technology is used in order to implement the penalty protocol (be it simple transactions, or smart contracts).

One may wonder whether financial fairness of the above protocols can be savaged

by playing with the collateral, or by running a financially unfair protocol in a round-robin fashion. Interestingly, in Section 5.6, we show that these approaches are deemed to fail for all practical purposes. The proof uses game-theoretic arguments.

Chapter 2

Useful Tools

2.1 Notation

We use $\lambda \in \mathbb{N}$ to denote the security parameter, sans-serif letters (such as A, $\mathcal{B}$) to denote algorithms, caligraphic letters (such as $\mathcal{X}$, $\mathcal{Y}$) to denote sets, and bold-face letters (such as $\mathbf{v}$, $\mathbf{A}$) to denote vectors and matrices; all vectors are by default row vectors, and $\tau\mathbf{v}$ denotes a column vector. An algorithm is *probabilistic polynomial-time* (PPT) if it is randomized, and its running time can be bounded by a polynomial in its input length. By $y \leftarrow\!\!\$\ \mathsf{A}(x)$, we mean that the value y is assigned to the output of algorithm A upon input x and fresh random coins. We implicitly assume that all algorithms are given the security parameter 1^λ as input.

A function $\nu : \mathbb{N} \to [0,1]$ is negligible in the security parameter (or simply negligible) if it vanishes faster than the inverse of any polynomial in λ, i.e. $\nu(\lambda) \in O(1/p(\lambda))$ for all positive polynomials $p(\lambda)$. We often write $\nu(\lambda) \in \mathsf{negl}(\lambda)$ to denote that $\nu(\lambda)$ is negligible.

For a random variable $\mathbf{X}$, we write $\mathbb{P}[\mathbf{X} = x]$ for the probability that $\mathbf{X}$ takes on a particular value $x \in \mathcal{X}$ (with $\mathcal{X}$ being the set where $\mathbf{X}$ is defined). The statistical distance between two random variables $\mathbf{X}$ and $\mathbf{X}'$ defined over the same set $\mathcal{X}$ is defined as $\mathbb{SD}(\mathbf{X};\mathbf{X}') = \frac{1}{2}\sum_{x\in\mathcal{X}} |\Pr[\mathbf{X} = x] - \Pr[\mathbf{X}' = x]|$. Given two ensembles $\mathbf{X} = \{X_\lambda\}_{\lambda\in\mathbb{N}}$ and $\mathbf{Y} = \{Y_\lambda\}_{\lambda\in\mathbb{N}}$, we write $\mathbf{X} \equiv \mathbf{Y}$ to denote that they are identically distributed, $\mathbf{X} \approx_s \mathbf{Y}$ to denote that they are statistically close (i.e., $\mathbb{SD}(X_\lambda; Y_\lambda) \in \mathsf{negl}(\lambda)$), and $\mathbf{X} \approx_c \mathbf{Y}$ to denote that they are computationally indistinguishable—i.e., for all PPT distinguishers D there exists a negligible function $\nu : \mathbb{N} \to [0,1]$ such that $|\Pr[\mathsf{D}(X_\lambda) = 1] - \Pr[\mathsf{D}(Y_\lambda) = 1]| \leq \nu(\lambda)$.

We call a group efficiently sampleable if and only if there is a PPT sampling procedure Samp for the uniform distribution over the group, and moreover there exists a PPT simulator $\mathsf{SimSamp}$ that given an element of the group, outputs the randomness used by Samp. More precisely, $r \approx_c r'$ where $r' \leftarrow\!\!\$\ \mathsf{SimSamp}(1^\lambda, \mathsf{Samp}(1^\lambda; r))$ and $r \leftarrow\!\!\$\ \{0,1\}^*$.[1]

If $f : (\{0,1\}^*)^n \to (\{0,1\}^*)^n$ is a function, then $f_i(x_1,\ldots,x_n)$ is the i-th element of $f(x_1,\ldots,x_n)$ for $i \in [n]$, and $(x_1,\ldots,x_n) \mapsto (\mathtt{out}_1,\ldots,\mathtt{out}_n)$ is its input-output behavior.

[1] The existence of a simulator is crucial for constructing SUSH-OT from SU-KA; we solely use it for this purpose.

2.2 Non-Interactive Commitment Schemes

A non-interactive commitment scheme is an efficient randomized algorithm Commit taking as input a message $m \in \mathcal{M}$ together with random coins $\omega \in \{0,1\}^\lambda$, and returning a commitment γ. The opening of a commitment γ consists of strings (m, ω) such that $\gamma = \mathsf{Commit}(m;\omega)$; we sometimes write $\delta(m)$ to denote the randomness that is needed to open successfully a value γ, i.e. $\gamma = \mathsf{Commit}(m;\delta(m))$.

As for security, commitment schemes should satisfy two properties called hiding and binding. Intuitively, the first property says that a commitment does not leak any information on the committed message; the second property says that it should be hard to open a given commitment in two different ways. The formal definitions follow.

Definition 1 (Hiding of commitments)**.** A commitment scheme is perfectly (resp., computationally or statistically) hiding, if for all $m_0, m_1 \in \mathcal{M}$ it holds that the ensembles $\{\mathsf{Commit}(m_0;U_\lambda)\}_{\lambda\in\mathbb{N}}$ and $\{\mathsf{Commit}(m_1;U_\lambda)\}_{\lambda\in\mathbb{N}}$ are identically distributed (resp., computationally or statistically close), where U_λ denotes the uniform distribution over $\{0,1\}^\lambda$.

Sometimes we find more useful to define hiding by defining a LR (Left-Right oracle) to capture cases where multiple commitment are used inside the protocol.

Definition 2 (Hiding of commitments (LR Oracle))**.** We say that a non-interactive commitment Commit is *computationally hiding* if for all non-uniform PPT adversaries A the following quantity is negligible

$$\left|\mathbb{P}\left[\mathsf{A}^{\mathsf{LR}(0,\cdot,\cdot)}(1^\lambda)=1\right]-\mathbb{P}\left[\mathsf{A}^{\mathsf{LR}(1,\cdot,\cdot)}(1^\lambda)=1\right]\right|,$$

where the oracle $\mathsf{LR}(b,\cdot,\cdot)$ with hard-wired $b \in \{0,1\}$ takes as input pairs of messages $m_0, m_1 \in \{0,1\}^\ell$, and outputs $\mathsf{Commit}(m_b)$.

Definition 3 (Binding of commitments)**.** A commitment scheme is computationally binding, if for all PPT adversaries A there is a negligible function $\nu : \mathbb{N} \to [0,1]$ such that

$$\Pr\left[\mathsf{Commit}(m;\omega)=\mathsf{Commit}(m';\omega') \wedge m \neq m' :\ ((m,\omega),(m',\omega')) \leftarrow\!\!\$\ \mathsf{A}(1^\lambda)\right] \leq \nu(\lambda).$$

In case the above probability equals zero for all even unbounded adversaries, we say that the commitment scheme is perfectly binding.

2.3 Secret Sharing Schemes

An n-party secret sharing scheme $(\mathsf{Share}, \mathsf{Recon})$ is a pair of poly-time algorithms specified as follows. (i) The randomized algorithm Share takes as input a message $m \in \mathcal{M}$ and outputs n shares $\sigma = (\sigma_1, \ldots, \sigma_n) \in \mathcal{S}_1 \times \cdots \times \mathcal{S}_n$; (ii) The deterministic algorithm Recon takes as input a subset of the shares, say $\sigma_\mathcal{I}$ with $\mathcal{I} \subseteq [n]$, and outputs a value in $\mathcal{M} \cup \{\perp\}$.

Definition 4 (Threshold secret sharing)**.** Let $n \in \mathbb{N}$. For any $t \leq n$, we say that $(\mathsf{Share}, \mathsf{Recon})$ is an (t, n)-threshold secret sharing scheme if it satisfies the following properties.

- **Correctness**: For any message $m \in \mathcal{M}$, and for any $\mathcal{I} \subseteq [n]$ such that $|\mathcal{I}| \geq t$, we have that $\mathsf{Recon}(\mathsf{Share}(m)_{\mathcal{I}}) = m$ with probability one over the randomness of Share.
- **Privacy**: For any pair of messages $m_0, m_1 \in \mathcal{M}$, and for any $\mathcal{U} \subset [n]$ such that $|\mathcal{U}| < t$, we have that

$$\{\mathsf{Share}(1^\lambda, m_0)_{\mathcal{U}}\}_{\lambda \in \mathbb{N}} \approx_c \{\mathsf{Share}(1^\lambda, m_1)_{\mathcal{U}}\}_{\lambda \in \mathbb{N}}.$$

2.4 Secret-Key Encryption

A secret-key encryption (SKE) scheme over key space $\mathcal{K}$ is a pair of polynomial-time algorithms $(\mathsf{Enc}, \mathsf{Dec})$ specified as follows. (i) The randomized algorithm Enc takes as input a key $\mathsf{k} \in \mathcal{K}$ and a message $m \in \mathcal{M}$, and outputs a ciphertext $c \in \mathcal{C}$; (ii) The deterministic algorithm Dec takes as input a key $\mathsf{k} \in \mathcal{K}$ and a ciphertext $c \in \mathcal{C}$, and outputs a value in $\mathcal{M} \cup \{\perp\}$. Correctness says that for every key $\mathsf{k} \in \mathcal{K}$, and every message $m \in \mathcal{M}$, it holds that $\mathsf{Dec}(\mathsf{k}, \mathsf{Enc}(\mathsf{k}, m)) = m$ with probability one over the randomness of Enc.

Definition 5 (Semantic security)**.** We say that $(\mathsf{Enc}, \mathsf{Dec})$ satisfies semantic security if for all pairs of messages $m_0, m_1 \in \mathcal{M}$ it holds that

$$\{\mathsf{Enc}(\mathsf{k}, m_0) : \mathsf{k} \leftarrow\!\!\$\ \mathcal{K}\}_{\lambda \in \mathbb{N}} \approx_c \{\mathsf{Enc}(\mathsf{k}, m_1) : \mathsf{k} \leftarrow\!\!\$\ \mathcal{K}\}_{\lambda \in \mathbb{N}}.$$

2.5 Public-key encryption

A public-key encryption (PKE) scheme is a tuple of polynomial-time algorithms $(\mathsf{Gen}, \mathsf{Enc}, \mathsf{Dec})$ specified as follows. (i) The randomized algorithm Gen takes as input the security parameter, and outputs a pair of keys $(\mathsf{pk}, \mathsf{sk})$; (ii) The randomized algorithm Enc takes as input a public key pk and a message $m \in \mathcal{M}$, and outputs a ciphertext c; (iii) The deterministic algorithm Dec takes as input a secret key sk and a ciphertext c, and outputs a value in $\mathcal{M} \cup \{\perp\}$ (where $\perp$ denotes decryption error). Correctness says that for every key $\lambda \in \mathbb{N}$, every $(\mathsf{pk}, \mathsf{sk})$ in the support of $\mathsf{Gen}(1^\lambda)$, and every message $m \in \mathcal{M}$, it holds that $\mathsf{Dec}(\mathsf{sk}, \mathsf{Enc}(\mathsf{pk}, m)) = m$ with probability one over the randomness of Enc.

Definition 6 (Semantic security)**.** We say that $(\mathsf{Gen}, \mathsf{Enc}, \mathsf{Dec})$ satisfies semantic security if for all PPT attackers $\mathsf{A} := (\mathsf{A}_0, \mathsf{A}_1)$ there exists a negligible function $\nu(\cdot)$ such that:

$$\left| \mathbb{P}\left[b' = b : \begin{array}{c} (\mathsf{pk}, \mathsf{sk}) \leftarrow\!\!\$\ \mathsf{Gen}(1^\lambda); (m_0, m_1, z) \leftarrow\!\!\$\ \mathsf{A}_0(\mathsf{pk}) \\ b \leftarrow\!\!\$\ \{0,1\}; c \leftarrow\!\!\$\ \mathsf{Enc}(\mathsf{pk}, m_b); b' \leftarrow\!\!\$\ \mathsf{A}_1(z, c) \end{array} \right] - \frac{1}{2} \right| \leq \nu(\lambda).$$

2.6 Signature schemes

A signature scheme is a tuple of polynomial-time algorithms $(\mathsf{Gen}, \mathsf{Sign}, \mathsf{Verify})$ specified as follows. (i) The randomized algorithm Gen takes as input the security parameter and outputs a secret key sk together with a public verification key pk; (ii) The deterministic algorithm Sign takes as input the secret key sk and a message $x \in \{0,1\}^*$ and outputs a signature y; (iii) The randomized algorithm Verify takes as an input the verification key pk, a message/signature pair (x, y) and outputs a decision bit.

Correctness says that for all $\lambda \in \mathbb{N}$, for all $(\mathsf{pk}, \mathsf{sk}) \in \mathsf{Gen}(1^\lambda)$, and for all $x \in \{0,1\}^*$ it holds that $\mathsf{Verify}(\mathsf{pk}, x, \mathsf{Sign}(\mathsf{sk}, x)) = 1$ (with probability one over the coin tosses of Verify).

We will need so called *unique* signature schemes, which satisfy two properties known as uniqueness and unforgeability as defined below.

Definition 7 (Uniqueness)**.** For every pk, x, y_0, y_1 with $y_0 \neq y_1$ there exists a negligible function $\nu(\cdot)$ such that the following holds for either $i = 0$ or $i = 1$:

$$\mathbb{P}\left[\mathsf{Verify}(\mathsf{pk}, x, y_i) = 1\right] \leq \nu(\lambda).$$

In words, for every string pk and every x, there exists at most one value y that is a accepting signature of x.

Definition 8 (Unforgeability)**.** For all PPT *valid* attackers A there exists a negligible function $\nu(\cdot)$ such that:

$$\mathbb{P}\left[\mathsf{Sign}(\mathsf{sk}, x) = y : \begin{array}{c}(\mathsf{pk}, \mathsf{sk}) \leftarrow\!\!\$\ \mathsf{Gen}(1^\lambda) \\ (x, y) \leftarrow\!\!\$\ \mathsf{A}^{\mathsf{Sign}(\mathsf{sk},\cdot)}(\mathsf{pk})\end{array}\right] \leq \nu(\lambda),$$

where A is called *valid* if it never queries x to its oracle.

Unique signatures are sometimes also known under the name of verifiable unpredictable functions, and exist based on a variety of assumptions [BR96, MRV99, Lys02, DY05].

2.7 MPC Definitions and Functionalities

In order to define security of an n-party protocol π for computing a function f, we compare an execution of π in the real world with an ideal process where the parties simply send their inputs to an ideal functionality $\mathcal{F}_f$ that evaluates the function on behalf of the players. In the ideal world the ideal functionality, acting as a trusted third party, takes the parties' inputs $(x_i)_{i\in[n]}$ privately, and outputs the value $f_i(x_1, \ldots, x_n)$ to each party $i \in [n]$. In the real world, where parties directly exchange messages between themselves, such trusted third party does not exist. A protocol is said to be secure if the two worlds are (computationally) indistinguishable.

An important feature of simulation-based security is *composability*. Intuitively, this property refers to the guarantee that an MPC protocol securely realizing an ideal functionality, continues to do so even if used as a sub-protocol in a larger

protocol, which greatly simplifies the design and security analysis of MPC protocols. The most basic form of composition is known as *sequential composability*, which roughly corresponds to the assumption that each sub-protocol is run sequentially and in isolation. A much stronger flavor of composition is the so-called *universal composability* (UC) [Can20, Can01, CLOS02], which instead corresponds to the more realistic scenario where many secure protocols are executed together. In particular, in UC, both the real and ideal world are coordinated by an environment $\mathcal{Z}$ that can run multiple interleaved executions of different protocols. We say that π t-securely computes $\mathcal{F}_f$ if for all efficient adversaries A corrupting at most t parties in the real execution, there exists an efficient simulator Sim, such that no efficient environment Z interacting with the adversary in both worlds can tell apart the output of A in the real world from the output of A when its view is simulated by Sim.

In the case of sequential composition, Z is replaced by a distinguisher D handling inputs to the parties and waiting to receive the output and an arbitrary function of A's view at some point. The latter allows the simulator to internally control the adversary, *e.g.* by rewinding it. Since in the UC setting the interaction between Z and A can be arbitrary, eventual rewinds of the adversary from the simulator can be spotted by the environment, and thus input extraction techniques adopted by the simulator cannot be based on rewinding (this is usually called *straight-line simulation*).

2.7.1 Sequential Composability

The Real Execution. In the real world, protocol π is run by a set of parties $\mathsf{P}_1, \ldots, \mathsf{P}_n$ in the presence of a PPT adversary A coordinated by a non-uniform distinguisher D. At the outset, D chooses the inputs $(1^\lambda, x_i)_{i\in[n]}$ for each player P_i, the set of corrupted parties $\mathcal{I} \subset [n]$, auxiliary information $z \in \{0,1\}^*$, and gives $((x_i)_{i\in\mathcal{I}}, z)$ to A. Hence, the protocol π is run with the honest parties following their instructions (using input x_i), and with the attacker A sending all messages of the corrupted players by following any polynomial-time strategy. Finally, A passes an arbitrary function of its view to D, who is also given the output of the honest parties, and returns a bit.

The Ideal Execution. The ideal process for the computation of f involves a set of dummy parties $\mathsf{P}_1, \ldots, \mathsf{P}_n$, an ideal adversary Sim (a.k.a. the simulator), and an ideal functionality $\mathcal{F}_f$. At the outset, D chooses $(1^\lambda, x_i)_{i\in[n]}$, $\mathcal{I}$, z and sends $((x_i)_{i\in[n]}, z)$ to Sim. Hence, each player P_i sends its input x'_i to the ideal functionality—where $x'_i = x_i$ if P_i is honest, and otherwise x'_i is chosen arbitrarily by the simulator—which returns to the parties their respective outputs $f_i(x'_1, \ldots, x'_n)$. Finally, Sim passes an arbitrary function of its view to D, who is also given the output of the honest parties, and returns a bit.

Securely Realizing an Ideal Functionality. Intuitively, an MPC protocol is secure if whatever the attacker can learn in the real world can be emulated by the simulator in the ideal execution. Formally, if we denote by $\mathbf{REAL}_{\pi,\mathsf{A},\mathsf{D}}(\lambda)$ the random variable corresponding to the output of D in the real execution, and by $\mathbf{IDEAL}_{f,\mathsf{Sim},\mathsf{D}}(\lambda)$ the random variable corresponding to the output of D in the

ideal execution, the two probability ensembles have to be computationally indistinguishable.

Definition 9 (Simulation-based security)**.** Let $n \in \mathbb{N}$. Let $\mathcal{F}_f$ be an ideal functionality for $f : (\{0,1\}^*)^n \to (\{0,1\}^*)^n$, and let π be an n-party protocol. We say that π s-securely computes $\mathcal{F}_f$ if for any PPT adversary A there exists a PPT simulator Sim such that all PPT non- uniform distinguishers D corrupting at most s parties, we have

$$\left\{\mathbf{IDEAL}_{f,\mathsf{Sim},\mathsf{D}}(\lambda)\right\}_{\lambda\in\mathbb{N}} \approx_c \left\{\mathbf{REAL}_{\pi,\mathsf{A},\mathsf{D}}(\lambda)\right\}_{\lambda\in\mathbb{N}}.$$

When replacing $\mathbf{IDEAL}_{f,\mathsf{Sim},\mathsf{D}}(\lambda)$ with $\mathbf{IDEAL}_{f_\perp,\mathsf{Sim},\mathsf{D}}(\lambda)$ in Def. 9, we say that π s-securely computes f with aborts.

The Hybrid Model. Let $\mathbf{HYBRID}^g_{\pi,\mathsf{A},\mathsf{D}}(\lambda)$ denote the random variable corresponding to the output of D in the $\mathcal{F}_g$-hybrid model. We say that a protocol π_f for computing f is secure in the $\mathcal{F}_g$-hybrid model if the ensembles $\mathbf{HYBRID}^g_{\pi_f,\mathsf{A},\mathsf{D}}(\lambda)$ and $\mathbf{IDEAL}_{f,\mathsf{Sim},\mathsf{D}}(\lambda)$ are computationally close.

Definition 10 (MPC in the hybrid model)**.** Let $n \in \mathbb{N}$. Let $\mathcal{F}_f$, $\mathcal{F}_g$ be ideal functionalities for $f, g : (\{0,1\}^*)^n \to (\{0,1\}^*)^n$, and let π be an n-party protocol. We say that π s-securely realizes $\mathcal{F}_f$ in the $\mathcal{F}_g$- hybrid model if for all PPT adversaries A there exists a PPT simulator Sim such that for all PPT non-uniform distinguishers D corrupting at most s players, we have

$$\left\{\mathbf{IDEAL}_{f,\mathsf{Sim},\mathsf{D}}(\lambda)\right\}_{\lambda\in\mathbb{N}} \approx_c \left\{\mathbf{HYBRID}^g_{\pi,\mathsf{A},\mathsf{D}}(\lambda)\right\}_{\lambda\in\mathbb{N}}.$$

2.7.2 Universal Composability

The Real Execution In the real world, the protocol π is run in the presence of an adversary A coordinated by a non-uniform environment $\mathsf{Z} = \{\mathsf{Z}_\lambda\}_{\lambda\in\mathbb{N}}$. At the outset, Z chooses the inputs $(1^\lambda, x_i)$ for each player P_i, and gives $\mathcal{I}$, $\{x_i\}_{i\in\mathcal{I}}$ and z to A, where $\mathcal{I} \subset [n]$ represents the set of corrupted players and z is some auxiliary input. For simplicity, we only consider static corruptions (i.e., the environment decides who is corrupt at the beginning of the protocol). The parties then start running π, with the honest players P_i behaving as prescribed in the protocol (using input x_i), and with malicious parties behaving arbitrarily (directed by A). The attacker may delay sending the messages of the corrupted parties in any given round until after the honest parties send their messages in that round; thus, for every r, the round-r messages of the corrupted parties may depend on the round-r messages of the honest parties. Z can interact with A throughout the course of the protocol execution.
Additionally, Z receives the outputs of the honest parties, and must output a bit. We denote by $\mathbf{REAL}_{\pi,\mathsf{A},\mathsf{Z}}(\lambda)$ the random variable corresponding to Z's guess.

The Ideal Execution In the ideal world, a trusted third party evaluates the function f on behalf of a set of dummy players $(\mathsf{P}_i)_{i\in[n]}$. As in the real setting, Z chooses the inputs $(1^\lambda, x_i)$ for each honest player P_i, and gives $\mathcal{I}$, $\{x_i\}_{i\in\mathcal{I}}$ and z to the ideal adversary Sim, corrupting the dummy parties $(\mathsf{P}_i)_{i\in\mathcal{I}}$. Hence, honest

parties send their input $x'_i = x_i$ to the trusted party, whereas the parties controlled by Sim might send an arbitrary input x'_i. The trusted party computes $(y_1, \ldots, y_n) = f(x'_1, \ldots, x'_n)$, and sends y_i to P_i. During the simulation, Sim and A can interact with Z throughout the course of the protocol execution. Additionally, Z receives the outputs of the honest parties and must output a bit. We denote by $\mathbf{IDEAL}_{f,\mathsf{Sim},\mathsf{Z}}(\lambda)$ the random variable corresponding to Z's guess.

Definition 11 (UC-Secure MPC)**.** Let π be an n-party protocol for computing a function $f : (\{0,1\}^*)^n \to (\{0,1\}^*)^n$. We say that π t-securely UC-realizes f in the presence of malicious adversaries if such that for every PPT adversary A there exists a PPT simulator Sim such that for every non-uniform PPT environment Z corrupting at most t parties the following holds:

$$\{\mathbf{REAL}_{\pi,\mathsf{A},\mathsf{Z}}(\lambda)\}_{\lambda\in\mathbb{N}} \approx_c \left\{\mathbf{IDEAL}_{f,\mathsf{Sim},\mathsf{Z}}(\lambda)\right\}_{\lambda\in\mathbb{N}}.$$

When replacing $\mathbf{IDEAL}_{f,\mathsf{Sim},\mathsf{Z}}(\lambda)$ with $\mathbf{IDEAL}_{f_\perp,\mathsf{Sim},\mathsf{Z}}(\lambda)$ we say that π s-securely computes f *with aborts* in the presence of malicious adversaries.

UC Hybrid Model. Let $\mathbf{HYBRID}^g_{\pi,\mathsf{A},\mathsf{Z}}(\lambda)$ denote the random variable corresponding to the output of Z in the $\mathcal{F}_g$-hybrid model. We say that a protocol π_f for computing f is secure in the $\mathcal{F}_g$-hybrid model if the ensembles $\mathbf{HYBRID}^g_{\pi_f,\mathsf{A},\mathsf{Z}}(\lambda)$ and $\mathbf{IDEAL}_{f,\mathsf{Sim},\mathsf{Z}}(\lambda)$ are computationally close.

Definition 12 (UC-Secure MPC in the hybrid model)**.** Let $n \in \mathbb{N}$. Let $\mathcal{F}_f$, $\mathcal{F}_g$ be ideal functionalities for $f, g : (\{0,1\}^*)^n \to (\{0,1\}^*)^n$, and let π be an n-party protocol. We say that π s-securely realizes $\mathcal{F}_f$ in the $\mathcal{F}_g$- hybrid model if for all PPT adversaries A there exists a PPT simulator Sim such that for all PPT non-uniform environments Z corrupting at most t players, we have

$$\left\{\mathbf{IDEAL}_{f,\mathsf{Sim},\mathsf{Z}}(\lambda)\right\}_{\lambda\in\mathbb{N}} \approx_c \left\{\mathbf{HYBRID}^g_{\pi,\mathsf{A},\mathsf{Z}}(\lambda)\right\}_{\lambda\in\mathbb{N}}.$$

2.7.3 Random Oracle Model

In the Random Oracle Model (ROM), all the parties involved in a protocol have oracle access to a truly-random hash function. In particular, when a value v is given as an input from a party to the RO, the latter samples a random answer, stores the pair (v, r), and outputs r to the party. If the RO is queried on the same value v multiple times, the same answer r is output. In the ROM, the simulator needs to further simulate the interaction between the parties and the RO. While doing so, the simulator may program the output of the RO at specific inputs to particularly convenient random-looking values. This powerful feature is known as random-oracle programmability. In the setting of generalized UC, the RO is defined as a global ideal functionality $\mathcal{G}_{\mathsf{RO}}$. In this case, the simulator can only interact with the RO by sending queries to it, which severely limits random-oracle programming. Security proofs in the ROM only guarantee heuristic security. This is because ROs do not exist in the real world, and thus a security proof in the ROM only guarantees that

the protocol remains secure so long as the hash function is close enough to behave as a truly-random function.[2]

While the above may look controversial, security proofs in the ROM are generally considered useful as they essentially guarantee that any security vulnerability can only depend on the hash function.

2.7.4 Security with Penalties

Following [BK14], we extend the MPC framework to the setting of "MPC with coins", where each party is provided with his own wallet and safe[3]. As in [BK14, KMB15a, KVV16], we use $\mathsf{coins}(x)$ to represent a coin of value x, and denote special functionalities dealing with coins with the apex *. If a party owns $\mathsf{coins}(x)$ and deposits (resp. receives) $\mathsf{coins}(d)$ (resp. $\mathsf{coins}(q)$), it will own $\mathsf{coins}(x-d)$ (resp. $\mathsf{coins}(x+q)$).

To define fairness with penalties, we modify the ideal world using the following ideal functionality $\mathcal{F}_f^*$ (see Fig. 2.1 for a formal description):

1. At the outset, $\mathcal{F}_f^*$ receives the inputs and a deposit from each party; the coins deposited by the malicious parties must be enough to compensate all honest players in case of abort.

2. Then, in the output phase, the functionality returns the deposit to the honest parties; if the adversary deposited enough coins, it is given the chance to look at the output, and finally decides whether to continue delivering the output to the honest players, or to abort, in which case all honest players are compensated using the penalty amount deposited during the input phase.

In this setting, we say that a protocol π s-securely computes $\mathcal{F}_f^*$ with penalties (in the hybrid model). We note that this definition captures so-called "fair output delivery" (but not "fair computation"): an adversary can always abort after the inputs are provided for the MPC but before the collateral is deposited, without being punished. However, the latter is tolerated so long as intermediate outputs reveal nothing on the final output. Aborts during the computation can be eliminated by adding business requirements (collaterals first, output later).

The formal definition of secure MPC with penalties (in the hybrid model) is identical to one described in Section 2.7.2, except that $\mathcal{F}_f^{\perp}$ is replaced by the ideal functionality $\mathcal{F}_f^*$ depicted in Fig. 2.1. At the outset, the distinguisher D (or the environment Z) initializes each party's wallet with some number of coins. D (or Z) may read or modify (*i.e.*, add coins to or retrieve coins from) the wallet (but not the safe) of each honest party, whereas in the hybrid (resp. ideal) process, the adversary A (resp. Sim) has complete access to both wallets and safes of corrupt parties.

[2] Even worse, there exist (albeit contrived) cryptoschemes that are secure in the ROM but become always insecure for any possible instantiation of the RO with a real-world hash function [CGH98, CGH04].

[3] To ensure indistinguishaibility between real and ideal world it is crucial that the environment is only allowed to access and modify the wallet of each party, but not the safe. Precise details about the model can be found at [BK14].

At the end of the protocol, honest parties release the coins locked in the protocol to their wallet, the distinguisher is given the distribution of coins and outputs its final output.

Functionality $\mathcal{F}_f^*$

The functionality runs with security parameter 1^λ, minimum penalty amount q, parties $\mathsf{P}_1, \ldots, \mathsf{P}_n$, and adversary Sim that corrupts parties $\{\mathsf{P}_i\}_{i\in\mathcal{I}}$ for some $\mathcal{I} \subseteq [n]$. Let $\mathcal{H} = [n] \setminus \mathcal{I}$ with $h = |\mathcal{H}|$, and d be the amount of the safety deposit.

Input Phase: Wait to receive a message $(\mathtt{input}, sid, ssid, i, x_i, \mathsf{coins}(d))$ from P_i for all $i \in \mathcal{H}$, and message $(\mathtt{input}, \{x_i\}_{i\in\mathcal{I}}, \mathsf{coins}(hq))$ from Sim.

Output Phase: Proceed as follows.

- Send $(\mathtt{return}, sid, ssid, \mathsf{coins}(d))$ to P_i for all $i \in \mathcal{H}$, and let $(y_1, \ldots, y_n) = f(x_1, \ldots, x_n)$.
- Send $(\mathtt{output}, sid, ssid, \{y_i\}_{i\in\mathcal{I}})$ to Sim.
- If Sim returns $(\mathtt{continue}, sid, ssid, \mathcal{H}_{\mathsf{out}})$, then send $(\mathtt{output}, sid, ssid, y_i)$ to P_i for each $i \in \mathcal{H}$, send $(\mathtt{payback}, sid, ssid, \mathsf{coins}((h - |\mathcal{H}_{\mathsf{out}}|)q)$ to Sim, and send $(\mathtt{extrapay}, sid, ssid, \mathsf{coins}(q))$ to P_i for each $i \in \mathcal{H}_{\mathsf{out}}$.
- Else, if Sim returns $(\mathtt{abort}, sid, ssid)$ send $(\mathtt{penalty}, sid, ssid, \mathsf{coins}(q))$ to P_i for each $i \in \mathcal{H}$.

Figure 2.1. The functionality $\mathcal{F}_f^*$ for secure computation with penalties [BK14].

2.7.5 Oblivious Transfer

An interactive protocol Π for the Oblivious Transfer (OT) functionality, features two interactive PPT Turing machines S, R called, respectively, the sender and the receiver. The sender S holds a pair of strings $s_0, s_1 \in \{0,1\}^\lambda$, whereas the receiver R is given a choice bit $b \in \{0,1\}$. At the end of the protocol, which might take several rounds, the receiver learns s_b (and nothing more), whereas the sender learns nothing.

Typically, security of OT is defined using the real/ideal paradigm. Specifically, we compare a real execution of the protocol, where an adversary might corrupt either the sender or the receiver, with an ideal execution where the parties can interact with an ideal functionality. The ideal functionality, which we denote by $\mathcal{F}_{\mathtt{OT}}$, features a trusted party that receives the inputs from both the sender and the receiver, and then sends to the receiver the sender's input corresponding to the receiver's choice bit. We refer the reader to Fig. 2.2 for a formal specification of the $\mathcal{F}_{\mathtt{OT}}$ functionality.

In what follows, we denote by $\mathbf{REAL}_{\Pi,\mathsf{R}^*(z)}(\lambda, s_0, s_1, b)$ (resp., $\mathbf{REAL}_{\Pi,\mathsf{S}^*(z)}(\lambda, s_0, s_1, b)$) the distribution of the output of the malicious receiver (resp., sender) during a real execution of the protocol Π (with s_0, s_1 as inputs of the sender, b as choice bit of the receiver, and z as auxiliary input for the adversary), and by $\mathbf{IDEAL}_{\mathcal{F}_{\mathtt{OT}},\mathsf{Sim}^{\mathsf{R}^*(z)}}(\lambda, s_0, s_1, b)$ (resp., $\mathbf{IDEAL}_{\mathcal{F}_{\mathtt{OT}},\mathsf{Sim}^{\mathsf{S}^*(z)}}(\lambda, s_0, s_1, b)$) the output of the malicious receiver (resp., sender) in an ideal execution where the parties (with

Ideal Functionality $\mathcal{F}_{\mathtt{OT}}$

The functionality runs with Turing machines (S, R) and adversary Sim, and works as follows:

- Upon receiving message $(\mathtt{send}, s_0, s_1, \mathsf{S}, \mathsf{R})$ from S, where $s_0, s_1 \in \{0,1\}^\lambda$, store s_0 and s_1 and answer $\mathtt{send}$ to R and Sim.
- Upon receiving a message $(\mathtt{receive}, b)$ from R, where $b \in \{0,1\}$, send s_b to R and $\mathtt{receive}$ to S and Sim, and halt. If no message $(\mathtt{send}, \cdot)$ was previously sent, do nothing.

Figure 2.2. Oblivious transfer ideal functionality

analogous inputs) interact with $\mathcal{F}_{\mathtt{OT}}$, and where the simulator is given black-box access to the adversary.

Definition 13 (OT with full simulation)**.** Let $\mathcal{F}_{\mathtt{OT}}$ be the functionality from Fig. 2.2. We say that a protocol $\Pi = (\mathsf{S}, \mathsf{R})$ securely computes $\mathcal{F}_{\mathtt{OT}}$ with *full simulation* if the following holds:

(a) For every non-uniform PPT malicious receiver R^*, there exists a non-uniform PPT simulator Sim such that

$$\left\{\mathbf{REAL}_{\Pi,\mathsf{R}^*(z)}(\lambda, s_0, s_1, b)\right\}_{\lambda,s_0,s_1,b,z} \approx_c \left\{\mathbf{IDEAL}_{\mathcal{F}_{\mathtt{OT}},\mathsf{Sim}^{\mathsf{R}^*(z)}}(\lambda, s_0, s_1, b)\right\}_{\lambda,s_0,s_1,b,z}$$

where $\lambda \in \mathbb{N}$, $s_0, s_1 \in \{0,1\}^\lambda$, $b \in \{0,1\}$, and $z \in \{0,1\}^*$.

(b) For every non-uniform PPT malicious sender S^*, there exists a non-uniform PPT simulator Sim such that

$$\left\{\mathbf{REAL}_{\Pi,\mathsf{S}^*(z)}(\lambda, s_0, s_1, b)\right\}_{\lambda,s_0,s_1,b,z} \approx_c \left\{\mathbf{IDEAL}_{\mathcal{F}_{\mathtt{OT}},\mathsf{Sim}^{\mathsf{S}^*(z)}}(\lambda, s_0, s_1, b)\right\}_{\lambda,s_0,s_1,b,z}$$

where $\lambda \in \mathbb{N}$, $s_0, s_1 \in \{0,1\}^\lambda$, $b \in \{0,1\}$, and $z \in \{0,1\}^*$.

Game-based security. One can also consider weaker security definitions for OT, where simulation-based security only holds when either the receiver or the sender is corrupted, whereas when the other party is malicious only game-based security is guaranteed. Below, we give the definition for the case of a corrupted sender, which yields a security notion known as *receiver-sided* simulatability. Intuitively, the latter means that the adversary cannot distinguish whether the honest receiver is playing with choice bit 0 or 1.

Definition 14 (OT with receiver-sided simulation)**.** Let $\mathcal{F}_{\mathtt{OT}}$ be the functionality from Fig. 2.2. We say that a protocol $\Pi = (\mathsf{S}, \mathsf{R})$ securely computes $\mathcal{F}_{\mathtt{OT}}$ with *receiver-sided simulation* if the following holds:

(a) Same as property (a) in Definition 13.

(b) For every non-uniform PPT malicious sender S^* it holds that

$$\left\{ view^{\mathsf{R}}_{\Pi,\mathsf{S}^*(z)}(\lambda, s_0, s_1, 0) \right\}_{\lambda,s_0,s_1,z} \approx_c \left\{ view^{\mathsf{R}}_{\Pi,\mathsf{S}^*(z)}(\lambda, s_0, s_1, 1) \right\}_{\lambda,s_0,s_1,z}$$

where $\lambda \in \mathbb{N}$, $s_0, s_1 \in \{0,1\}^\lambda$, and $z \in \{0,1\}^*$, and where $view^{\mathsf{R}}_{\Pi,\mathsf{S}^*(z)}(\lambda, s_0, s_1, b)$ is the distribution of the view of S^* (with input s_0, s_1 and auxiliary input z) at the end of a real execution of protocol Π with the honest receiver R given b as input.

Receiver-sided simulatability is a useful stepping stone towards achieving full simulatability. In fact, Ostrovsky *et al.* [ORS15] show how to compile any 4-round OT protocol with receiver-sided simulatability to a 4-round OT protocol with full simulatability. This transformation can be easily extended to hold for any r-round protocol, with $r \geq 3$; the main reason is that the transform only relies on an extractable commitment scheme, which requires at least 3 rounds.

Theorem 2 (Adapted from [ORS15])**.** *Assuming $t \geq 3$, there is a black-box transformation from t-round OT with* receiver-sided *simulation to t-round OT with full simulation.*[4]

2.8 A Blockchain Model

Below, we describe verbatim the blockchain model of [GG17] (which in turn builds on [PSS17, GKL15]). A blockchain protocol Γ consists of the following algorithms:

- $\mathsf{UpdateState}(1^\lambda)$: It is a stateful algorithm that take as input a security parameter $\lambda \in \mathbb{N}$, and maintains a local state $\alpha \in \{0,1\}^*$ which essentially consists of the entire blockchain (i.e., the sequence of minted blocks).
- $\mathsf{GetRecords}(1^\lambda, \alpha)$: It takes as input the security parameter and a state $\alpha \in \{0,1\}^*$. It outputs the longest *ordered* sequence of valid blocks (or simply blockchain) $\mathcal{B} = (\beta_1, \beta_2, \ldots)$ contained in the state variable, where each block β in the chain itself contains an *unordered* sequence of records/messages $(m_1, m_2, \ldots)$.
- $\mathsf{Broadcast}(1^\lambda, m)$: It takes as input the security parameter and a message $m \in \{0,1\}^*$, and broadcasts the message over the network to all nodes executing the blockchain protocol. It does not give any output.

The blockchain protocol is also parameterized by a validity predicate V that captures the semantics of any particular blockchain application. The validity predicate takes as input a sequence of blocks $\mathcal{B}$ and outputs a bit, where the value 1 certifies the validity of the blockchain $\mathcal{B}$. Since V is immaterial for our purposes, in what follows we simply omit it.

[4] They also need the existence of one-way functions. Since OT implies OT extension which implies one-way functions [LZ13, LZ18], OT implies one-way functions.

Blockchain execution. Each participant in the protocol runs $\mathsf{UpdateState}$ to keep track of the latest blockchain state. This corresponds to listening on the broadcast network for messages from other nodes. $\mathsf{GetRecords}$ is used to extract an ordered sequence of blocks encoded in the blockchain state variable, which is considered as the common public ledger among all the nodes. Finally, $\mathsf{Broadcast}$ is used by a party when it wants to post a new message on the blockchain; such messages are accepted by the blockchain protocol only if they satisfy the validity predicate given the current state.

The execution of a blockchain protocol $\Gamma = (\mathsf{UpdateState}, \mathsf{GetRecords}, \mathsf{Broadcast})$ is directed by an environment $\mathsf{Z}(1^\lambda)$ which activates the parties as either honest or corrupt, and is also responsible for providing inputs/records to all parties in each round. All the corrupted parties are controlled by the adversary A, which is also responsible for delivery of all network messages. Honest parties start by executing $\mathsf{UpdateState}$ on input 1^λ, with an empty local state $\alpha = \varepsilon$. Then, the protocol execution proceeds in rounds that model times steps, as detailed below.

- In round $r \in \mathbb{N}$, each honest player P_i potentially receives messages from Z, and incoming network messages (delivered by A). It may then perform any computation, broadcast a message to all other players (which will be delivered by the adversary as explained below), and update its local state α_i. It could also attempt to add a new block to its chain i.e., run the mining procedure.
- The attacker A is responsible to deliver all messages sent by parties (honest or corrupted) to all other parties. The adversary cannot modify the content of messages broadcast by honest players, but it may delay or reorder the delivery of a message as long as it eventually delivers all messages within a certain time limit.
- At any point, Z can communicate with A or access $\mathsf{GetRecords}(1^\lambda, \alpha_i)$ where α_i is the local state of player P_i.

With the notation $\mathcal{B} \preceq \mathcal{B}'$, we denote that the blockchain $\mathcal{B}$ is a prefix of $\mathcal{B}'$. We also let $\mathcal{B}^{\lceil k}$ be the chain resulting from pruning the last k blocks in $\mathcal{B}$. Let $\mathbf{EXEC}_{\Gamma,\mathsf{A},\mathsf{Z}}(\lambda)$ be the random variable denoting the joint view of all parties in the execution of protocol Γ with adversary A, and environment Z. Note that this view fully determines the execution.

Blockchain properties. We now define two natural guarantees that are respected by an ideal ledger. The first property, called *consistency*, intuitively states that the view of the blockchain obtained by different players is identical up to pruning a certain number of blocks from the top of the chain. Let $\mathsf{Consistent}^k(\cdot)$ be the predicate that returns 1 iff for all rounds $r \leq \tilde{r}$, and all parties $\mathsf{P}_i, \mathsf{P}_j$ (potentially the same) such that P_i is honest at round r with blockchain $\mathcal{B}$ and player P_j is honest at round $\tilde{r}$ with blockchain $\tilde{\mathcal{B}}$, we have that $\mathcal{B}^{\lceil k} \preceq \tilde{\mathcal{B}}$.

Definition 15 (Chain consistency)**.** A blockchain protocol Γ satisfies $k(\cdot)$-consistency with adversary A and environment Z, if there exists a negligible function $\nu(\cdot)$ such that for every $\bar{k} > k(\lambda)$, the following holds:

$$\mathbb{P}\left[\mathsf{Consistent}^{\bar{k}}(\mathit{view}) = 1 : \ \mathit{view} \leftarrow\!\!\$\ \mathbf{EXEC}_{\Gamma,\mathsf{A},\mathsf{Z}}(\lambda)\right] \geq 1 - \nu(\lambda).$$

We note that previous work considered an even stronger property, called *persistence*, stipulating that if some honest player reports a message m at depth k in its local ledger, then m will be always reported in the same position and equal or more depth by all honest parties. We omit a formal definition, as this property is not required for our purposes.

The second property, called *liveness*, intuitively says that if all honest parties attempt to broadcast a message m, then after w rounds, an honest party will see m at depth k in the ledger. Let $\mathsf{Live}^k(\cdot, w)$ be the predicate that returns 1 iff for any w consecutive rounds $r, \ldots, r+w$ there exists some round $r' \in [r, r+w]$ and index $i \in [n]$ such that: (1) P_i is honest and received a message m at round r, and (2) for every player P_j that is honest at $r+w$ with blockchain $\mathcal{B}$, it holds that $m \in \mathcal{B}^{\lceil k}$.

Definition 16 (Liveness)**.** A blockchain protocol Γ satisfies $(w(\cdot), k(\cdot))$-liveness with adversary A and environment Z, if there exists a negligible function $\nu(\cdot)$ such that for every $\bar{w} \geq w(\lambda)$ the following holds:

$$\mathbb{P}\left[\mathsf{Live}^k(\mathit{view}, \bar{w}) = 1 : \ \mathit{view} \leftarrow\!\!\$\ \mathbf{EXEC}_{\Gamma,\mathsf{A},\mathsf{Z}}(\lambda)\right] \geq 1 - \nu(\lambda).$$

Chapter 3

Round-Optimal OT from LWE and Low-Noise LPN Assumptions

3.1 Technical Overview

We proceed to a high level overview of the techniques behind our main result, starting with the notion of strong uniformity and the abstraction of commit-and-open protocols, and landing with the intuition behind our construction of M-OT (cf. Fig. 3.1).

As an important stepping stone to our main result, in Section 3.3 we introduce the notion of strong uniformity. Recall that a KA protocol allows Alice and Bob to share a key over a public channel, in such a way that the shared key is indistinguishable from uniform to the eyes of a passive eavesdropper. Strong uniformity here demands that, even if Bob is malicious, the messages sent by Alice are computationally close to uniform over an efficiently sampleable group.[1] This flavor of security straightforwardly translates to SH-OT and PKE, yielding so-called SUSH-OT and SU-PKE. In the case of SUSH-OT, it demands that all messages of the receiver have this property (even if the sender is malicious). For SU-PKE, we distinguish two types, which are a strengthening of the types defined by Gertner *et al.* [GKM$^+$00].[2]

- **Type-A PKE:** The distribution of the public key is computationally indistinguishable from uniform. This type of PKE is known to exist under DDH [Gam84] and CDH [GM84] over efficiently sampleable groups,[3] LWE [Reg05],

[1] We call a group efficiently sampleable if we can efficiently sample uniform elements from the group and, given a group element, we can simulate this sampling procedure. In the context of public key encryption a similar property is called oblivious key generation [DN00]. In our construction, we require a stronger property where the public keys are additionally indistinguishable from uniform.

[2] The difference is that the notions in [GKM$^+$00] only ask for oblivious sampleability, rather than our stronger requirement of uniformity over efficiently sampleable groups.

[3] These are groups for which one can directly sample a group element without knowing the discrete logarithm with respect to some generator. The latter requires non black-box access to the group, which is also needed when using ElGamal with messages that are encoded as group elements and not as exponents.

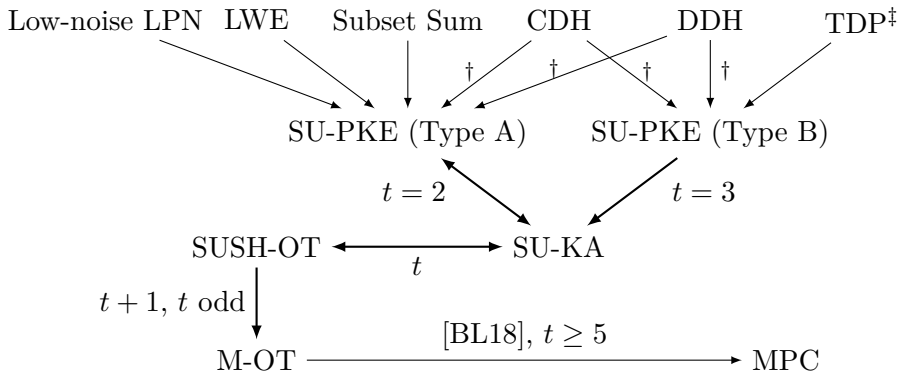


Figure 3.1. Overview over equivalence and implications of the notion of strong uniformity. The value $r \in \mathbb{N}$ denotes the round complexity. † This holds over efficiently sampleable groups. ‡ We need an enhanced certified TDP.

low-noise LPN [Ale03], and Subset Sum [LPS10].

- **Type-B PKE:** The encryption of a uniformly random message w.r.t. a maliciously chosen public key is computationally close to the uniform distribution over the ciphertext space. This type of PKE is harder to obtain, and can be constructed from enhanced certified TDPs, and from CDH and DDH over efficiently sampleable groups. In case of a TDP f, a ciphertext has the form $(f(r), h(r) \oplus m)$, where h is a hardcore predicate for f, and r is a random element from the domain of f. Under CDH or DDH, a ciphertext is defined as g^r and $h(g^{xr}) \cdot m$, $g^{xr} \cdot m$ respectively, where g^r is a uniform group element, and g^x is the public key. Clearly, for a uniform message m, these ciphertexts are uniform even under maliciously chosen public keys.

In Section 3.3, we show that SU Type-A and SU Type-B PKE imply, respectively, 2-round and 3-round SU-KA, whereas 2-round SU-KA implies SU Type-A PKE. Further, we prove that SU-KA is equivalent to SUSH-OT. The latter implies that strong uniformity is a sufficiently strong notion to bypass the black-box separation between OT and KA, in a similar way as Type-A and Type-B PKE bypass the impossibility of constructing OT from PKE [GKM$^+$00].

Commit-and-open protocols. A 1-out-of-2 commit-and-open (C&O) protocol is a 3-round protocol with the following structure: (1) In the first round, the prover, with inputs two messages m_0, m_1 and a choice bit b, sends a string γ (called "commitment") to the verifier; (2) In the second round, the verifier sends a value β to the prover (called "challenge"); (3) In the third round, the prover sends a tuple (δ, m_0, m_1) to the verifier (called "opening"). Security requires two properties. The first property, called existence of a committing branch, demands that a malicious prover must be committed to at least one message already after having sent γ. The second property, called choice bit indistinguishability, asks that a malicious verifier cannot learn the committing branch of an honest prover.

A construction of C&O protocols for single bits is implicit in Kilian [Kil92]. This has been extended to strings by Ostrovsky *et al.* [ORS15]. Both constructions make

black-box use of a statistically binding commitment scheme,[4] and allow a prover to equivocally open one of the messages. In Section 3.2.2 , we revisit the protocol and proof by Ostrovsky *et al.* to show that it indeed satisfies the two security notions sketched above.

M-OT from SUSH-OT: A warm up. In order to explain the main ideas behind our construction of M-OT, we describe below a simplified version of our protocol for the special case of $r = 2$, i.e. when starting with a 2-round SUSH-OT $(\mathsf{S}', \mathsf{R}')$; here, we denote with ρ the message sent by the receiver, and with σ the message sent by the sender, and further observe that for the case of 2 rounds the notion of strong uniformity collapses to standard semi-honest security with the additional property that the distribution of ρ is (computationally close to) uniform to the eyes of an eavesdropper. We then construct a 4-round OT protocol (S, R), as informally described below:

1. $(\mathsf{R} \to \mathsf{S})$: The receiver picks a uniformly random value $m_{1-b} \in \mathcal{M}$, where b is the choice bit, and runs the prover of the C&O protocol upon input m_{1-b}, obtaining a commitment γ that is forwarded to the sender.

2. $(\mathsf{S} \to \mathsf{R})$: The sender samples a challenge β for the C&O protocol, as well as uniformly random elements $r_0, r_1 \in \mathcal{M}$. Hence, it forwards (β, r_0, r_1) to the receiver.

3. $(\mathsf{R} \to \mathsf{S})$: The receiver runs the receiver R' of the underlying 2-round OT protocol with choice bit fixed to 0, obtaining a value ρ_b which is used to define the message $m_b = \rho_b - r_b$ required to complete the execution of the C&O protocol in the non-committing branch b. This results in a tuple (δ, m_0, m_1) that is forwarded to the sender.

4. $(\mathsf{S} \to \mathsf{R})$: The sender verifies that the transcript $T = (\gamma, \beta, (\delta, m_0, m_1))$ is accepting for the underlying C&O protocol. If so, it samples $u_0, u_1 \in \mathcal{M}$ uniformly at random, and runs the sender S' of the underlying 2-round OT protocol twice, with independent random tapes: The first run uses input strings (s_0, u_0) and message $m_0 + r_0$ from the receiver, resulting in a message σ_0, whereas the second run uses input strings (s_1, u_1) and message $m_1 + r_1$ from the receiver, resulting in a message σ_1. Hence, it sends (σ_0, σ_1) to the receiver.

5. Output: The receiver runs the receiver R' of the underlying 2-round OT protocol, upon input message σ_b from the sender, thus obtaining s_b.

Correctness is immediate. In order to prove simulation-based security we proceed in two steps. In the first step, we show the above protocol achieves a weaker security flavor called *receiver-sided simulatability* [NP05, ORS15] which consists of two properties: (1) The existence of a simulator which by interacting with the ideal

[4] Statistically binding commitment schemes are implied by SUSH-OT, since any SUSH-OT implies a (trivial) OT extension from n to $n + 1$ (where n is the security parameter), which in turn implies OWFs [LZ13, LZ18], which imply PRGs [HILL99] and thus statistically binding commitments [Nao91].

OT functionality can fake the view of any efficient adversary corrupting the receiver in a real execution of the protocol (i.e., standard simulation-based security w.r.t. corrupted receivers); (2) Indistinguishability of the protocol transcripts with choice bit of the receiver equal to zero or one, for any efficient adversary corrupting the sender in a real execution of the protocol (i.e., game-based security w.r.t. corrupted senders). In the second step, we rely on a *round-preserving* black-box transformation given in [ORS15], which allows to boost receiver-sided simulatability to fully-fledged malicious security. To show (1), we consider a series of hybrid experiments:

- In the first hybrid, we run the first 3 rounds of the protocol, yielding a partial transcript $\gamma, (\beta, r_0, r_1), (\delta, m_0, m_1)$. Hence, after verifying that $T = (\gamma, \beta, (\delta, m_0, m_1))$ is a valid transcript of the C&O protocol, we rewind the adversary to the end of the first round and continue the execution of the protocol from there using a fresh challenge (β', r'_0, r'_1), except that after the third round we artificially abort if there is no value $\hat{b} \in \{0, 1\}$ such that $m_{\hat{b}} = m'_{\hat{b}}$, where (δ', m'_0, m'_1) is the third message sent by the adversary after the rewinding.

  Notice that an abort means that it is not possible to identify a committing branch for the C&O protocol, which however can only happen with negligible probability; thus this hybrid is computationally close to the original experiment.

- In the second hybrid, we modify the distribution of the value r'_{1-b} (right after the rewinding) to $r''_{1-b} = \rho_{1-b} - m_{1-b}$, where we set $1 - b \stackrel{\text{def}}{=} \hat{b}$ from the previous hybrid, and where ρ_{1-b} is obtained by running the receiver R' of the underlying 2-round OT protocol with choice bit fixed to 1.

  To argue indistinguishability, we exploit the fact that the distribution of m_{1-b} is independent from that of r'_{1-b}, and thus by strong uniformity we can switch $r'_{1-b} + m_{1-b}$ with ρ_{1-b} from the receiver R'.

- In the third hybrid, we use the simulator of the underlying 2-round SH-OT protocol to compute the messages σ_{1-b} sent by the sender. Note that in both the third and the second hybrid the messages $(\rho_{1-b}, \sigma_{1-b})$ are computed by the honest sender, and thus any efficient algorithm telling apart the third and the second hybrid violates semi-honest security of $(\mathsf{S}', \mathsf{R}')$.

In the last hybrid, a protocol transcript is independent of s_{1-b} but still yields a well distributed output for the malicious receiver, which immediately implies a simulator in the ideal world.

To show (2), we first use the strong uniformity property of $(\mathsf{S}', \mathsf{R}')$ to sample m_b uniformly at random at the beginning of the protocol. Notice this implies that the receiver cannot recover the value s_b of the sender anymore. Finally, we use the choice bit indistinguishability of the C&O protocol to argue that the transcripts with $b = 0$ and $b = 1$ are computationally indistinguishable.

M-OT from SUSH-OT: The general case. There are several difficulties when trying to extend the above protocol to the general case where we start with a r-round SUSH-OT. In fact, if we would simply iterate sequentially the above construction,

where one iteration counts for a message from R' to S' and back, the adversary could use different committing branches from one iteration to the other. This creates a problem in the proof, as the simulator would need to be consistent with both choices of possible committing branches from the adversary, which however requires knowing both inputs from the sender.

We resolve this issue by having the receiver sending all commitments γ_i for the C&O protocol in the first round, where each value γ_i is generated including a random message m^i_{1-b} concatenated with the full history $m^{i-1}_{1-b}, \ldots, m^1_{1-b}$. Hence, during each iteration, the receiver opens one commitment as before. As we show, this prevents the adversary from switching committing branch from one iteration to the next one.

We refer the reader to Section 3.4.1 for a formal description of our protocol, and to Section 3.4.2 for a somewhat detailed proof intuition. The full proof appears in Section 3.4.3.

3.1.1 Application to Round-Efficient MPC

Since M-OT implies maliciously secure MPC [BL18, GS18], a direct consequence of Theorem 1 is the following:

Corollary 1. *For any odd $r \in \mathbb{N}$, there is a non-black-box construction of a $\max((r+1), 5)$-round maliciously secure multi-party computation protocol in the plain model, from any r-round strongly uniform key agreement protocol.*

Corollary 1 yields 5-round maliciously secure MPC from any of low-noise LPN, high-noise LWE, Subset Sum, CDH, DDH, and RSA, all with polynomial hardness. Previously to our work, it was known how to get maliciously secure MPC in the plain model, for arbitrary functionalities:

- Using 5 rounds, using interactive ZK proofs and SH-OT [BL18], assuming polynomially-hard LWE with super-polynomial noise ratio and adaptive commitments [BHP17], polynomially-hard DDH [ACJ17], and enhanced certified trapdoor permutations (TDP) [ORS15, BL18];
- Using 4 rounds, assuming sub-exponentially-hard LWE with super-polynomial noise ratio and adaptive commitments [BHP17], polynomially-hard LWE with a SIVP approximation factor of $n^{3.5}$ [BD18], sub-exponentially-hard DDH and one-way permutations [ACJ17], polynomially-hard DDH/QR/DCR [BGJ+18], and either polynomially-hard QR or QR together with any of LWE/DDH/DCR (all with polynomial hardness) [HHPV18].

3.2 Commit-and-Open and ORS Construction

Receiver-sided simulatability (described in Section 2.7.5) is a useful stepping stone towards achieving full simulatability. In fact, Ostrovsky *et al.* [ORS15] show how to compile any 4-round OT protocol with receiver-sided simulatability to a 4-round OT protocol with full simulatability. This transformation can be easily extended to hold for any r-round protocol, with $r \geq 3$; the main reason is that the transform only relies on an extractable commitment scheme, which requires at least 3 rounds.

Theorem 3 (Adapted from [ORS15])**.** *Assuming $t \geq 3$, there is a black-box transformation from t-round OT with* receiver-sided *simulation to t-round OT with full simulation.*[5]

3.2.1 Commit-and-Open Protocols

We envision a 3-round protocol between a prover and a verifier where the prover takes as input two messages $m_0, m_1 \in \mathcal{M}$ and a choice bit $b \in \{0,1\}$. The prover speaks first, and the protocol is public coin, in the sense that the message of the verifier consists of uniformly random bits. Intuitively, we want that whenever the prover manages to convince the verifier, he must be committed to at least one of m_0, m_1 already after having sent the first message.

More formally, a 1-out-of-2 commit-and-open (C&O) protocol is a tuple of efficient interactive Turing machines $\Pi_{\mathsf{c\&o}} \stackrel{\text{def}}{=} (\mathsf{P} = (\mathsf{P}_0, \mathsf{P}_1), \mathsf{V} = (\mathsf{V}_0, \mathsf{V}_1))$ specified as follows. (i) The randomized algorithm P_0 takes m_b and returns a string $\gamma \in \{0,1\}^*$ and auxiliary state information $\alpha \in \{0,1\}^*$; (ii) The randomized algorithm V_0 returns a random string $\beta \leftarrow\!\!\$\ \mathcal{B}$; (iii) The randomized algorithm P_1 takes $(\alpha, \beta, \gamma, m_{1-b})$ and returns a string $\delta \in \{0,1\}^*$; (iv) The deterministic algorithm V_1 takes a transcript $(\gamma, \beta, (\delta, m_0, m_1))$ and outputs a bit.

We write $\langle \mathsf{P}(m_0, m_1, b), \mathsf{V}(1^\lambda) \rangle$ for a run of the protocol upon inputs (m_0, m_1, b) to the prover, and we denote by $T \stackrel{\text{def}}{=} (\gamma, \beta, (\delta, m_0, m_1))$ the random variable corresponding to a transcript of the interaction. Note that the prover does not necessarily need to know m_{1-b} before computing the first message. We say that $\Pi_{\mathsf{c\&o}}$ satisfies completeness if honestly generated transcripts are always accepted by the verifier, i.e. for all $m_0, m_1 \in \mathcal{M}$ and $b \in \{0,1\}$, we have $\Pr[\mathsf{V}_1(T) = 1 : T \leftarrow\!\!\$\ \langle \mathsf{P}(m_0, m_1, b), \mathsf{V}(1^\lambda) \rangle] = 1$, where the probability is over the randomness of $\mathsf{P}_0, \mathsf{V}_0$, and P_1.

Security properties. Roughly, a C&O protocol must satisfy two security requirements. The first requirement is that at the end of the first round, a malicious prover is committed to at least one message. This can be formalized by looking at a mental experiment where we first run the protocol with a malicious prover, yielding a first transcript $T = (\gamma, \beta, (\delta, m_0, m_1))$; hence, we rewind the prover to the point it already sent the first message, and give it a fresh challenge β' which yields a second transcript $T' = (\gamma, \beta', (\delta', m_0', m_1'))$. The security property now states that, as long as the two transcripts T and T' are valid, it shall exist at least one "committing branch" $\hat{b} \in \{0,1\}$ for which $m_{\hat{b}} = m'_{\hat{b}}$. The second requirement says that no malicious verifier can learn any information on the choice bit of the prover. The formal definitions appear below.

Definition 17 (Secure commit-and-open protocol)**.** Let $\Pi_{\mathsf{c\&o}} = (\mathsf{P}_0, \mathsf{P}_1, \mathsf{V}_0, \mathsf{V}_1)$ be a 1-out-of-2 commit-and-open protocol. We say that $\Pi_{\mathsf{c\&o}}$ is secure if, besides completeness, it satisfies the following security properties.

[5] They also need the existence of one-way functions. Since OT implies OT extension which implies one-way functions [LZ13, LZ18], OT implies one-way functions.

- **Existence of Committing Branch:** For every PPT malicious prover $\mathsf{P}^* = (\mathsf{P}_0^*, \mathsf{P}_1^*)$ there exists a negligible function $\nu : \mathbb{N} \to [0,1]$ such that

$$\Pr\left[\begin{array}{c} (\mathsf{V}_1(T)=1) \wedge (\mathsf{V}_1(T')=1) \\ \wedge (m_0 \neq m_0') \wedge (m_1 \neq m_1') \end{array} : \begin{array}{c} (\gamma, \alpha_0) \leftarrow\!\!\$\ \mathsf{P}_0^*(1^\lambda); \\ \beta, \beta' \leftarrow\!\!\$\ \mathsf{V}_0(1^\lambda); \\ (\delta, m_0, m_1) \leftarrow\!\!\$\ \mathsf{P}_1^*(\alpha_0, \beta); \\ (\delta', m_0', m_1') \leftarrow\!\!\$\ \mathsf{P}_1^*(\alpha_0, \beta') \end{array}\right] \leq \nu(\lambda),$$

  with $T = (\gamma, \beta, (\delta, m_0, m_1))$ and $T' = (\gamma, \beta, (\delta, m_0', m_1'))$, and where the probability is taken over the random coin tosses of P^* and V.

- **Choice Bit Indistinguishability:** For all PPT malicious verifiers V^*, and for all messages $m_0, m_1 \in \mathcal{M}$, we have that

$$\left\{\tilde{T} : \tilde{T} \leftarrow\!\!\$\ \langle \mathsf{P}(m_0, m_1, 0), \mathsf{V}^*(1^\lambda)\rangle\right\}_{\lambda\in\mathbb{N}} \approx_c \left\{\tilde{T} : \tilde{T} \leftarrow\!\!\$\ \langle \mathsf{P}(m_0, m_1, 1), \mathsf{V}^*(1^\lambda)\rangle\right\}_{\lambda\in\mathbb{N}}.$$

We show in the next subsection that a protocol by Ostrovsky *et al.* [ORS15] achieves this definition.

3.2.2 The ORS Construction

The ORS string C&O protocol $\Pi_{\mathsf{c\&o}} = (\mathsf{P}_0, \mathsf{P}_1, \mathsf{V}_0, \mathsf{V}_1)$ for string length n is depicted in Fig. 3.2. It relies on a statistically binding commitment scheme $(\mathsf{Commit}, \delta)$ and a linear error detection code G with minimal distance of at least $\frac{1}{2}(n+\kappa)$, which can be instantiated with, e.g., a Reed-Solomon code. In what follows, the code G is a public parameter of the protocol, and we write $G(m)$ to denote an encoding of message m under code G. For simplifying the presentation of the protocol, we use $\psi : \mathbb{Z}_q^q \to \mathbb{Z}_q^{q-1}$ to denote the linear map

$$(\mathbf{x}[0], \ldots, \mathbf{x}[q-1]) \mapsto (\mathbf{x}[1] - \mathbf{x}[0], \ldots, \mathbf{x}[q-1] - \mathbf{x}[0]),$$

where $\mathbf{x}[i]$ is the i-th entry of a vector $\mathbf{x} \in \mathbb{Z}_q^q$.

Remark 1. *In the ORS protocol, the prover does not need to known or fix $\hat{\mathbf{m}}_{1-b}$ till the second round. Nevertheless, during the choice bit indistinguishability experiment, both messages need to be fixed before the first round.*

Lemma 1 (Completeness of the ORS protocol)**.** *For any $\epsilon \in [0,1)$, assuming that the commitment scheme* Commit *is complete with probability at least $1-\epsilon$, then the ORS protocol from Fig. 3.2 is complete with probability at least $(1-\epsilon)^{(q+1)n}$.*

Proof. The verifier opens $(q+1)n$ commitments. By completeness, the openings will be correct with probability $(1-\epsilon)^{(q+1)n}$. In the following, we assume that this is the case. The protocol will succeed if and only if the checks do not fail, i.e. all of the below equations hold:

$$\mathbf{c}_0 + \mathbf{c}_1 = \beta \qquad \hat{\mathbf{m}}_0 = G(m_0) \qquad \hat{\mathbf{m}}_1 = G(m_1)$$
$$\forall k \in \{0,1\}, i \in [n] : \ \psi(\mathbf{M}_{k,i}[\mathbf{c}_k[i], *]) = (-1)^{\mathbf{c}_k[i]}\mathbf{v}_{k,i}.$$

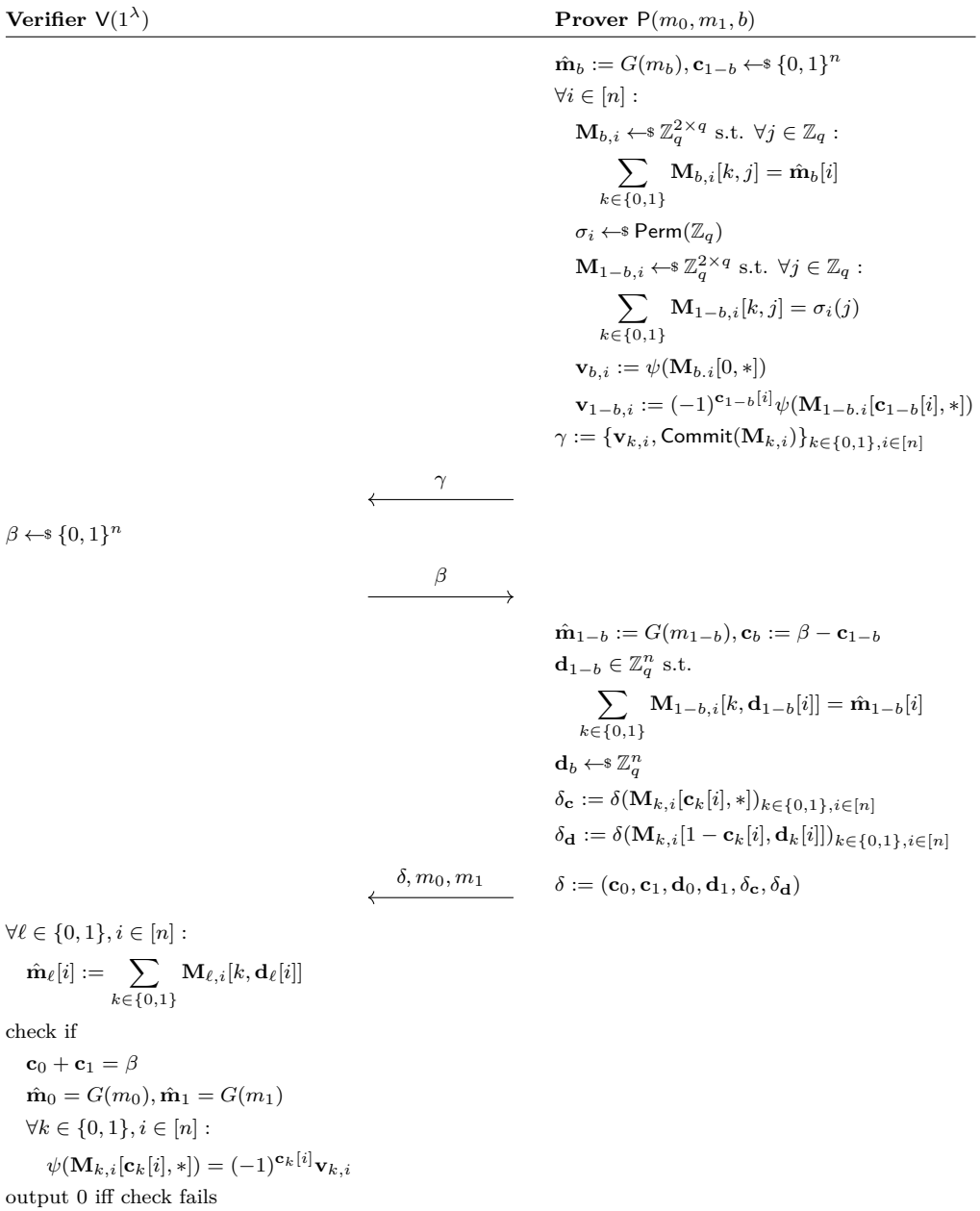


Figure 3.2. The ORS string 1-out-of-2 commit-and-open protocol. $\mathsf{Perm}(\mathbb{Z}_q)$ is the set of permutations over $\mathbb{Z}_q$, and $\delta(m)$ denotes the randomness that is needed to open commitment $\mathsf{Commit}(m)$.

By construction, it is easy to see that $\beta = \mathbf{c}_0 + \mathbf{c}_1$. Next we will show that in a honest execution of the protocol, both $\hat{\mathbf{m}}_0$ and $\hat{\mathbf{m}}_1$ will be codewords w.r.t. code G. The entries of $\hat{\mathbf{m}}_0$ and $\hat{\mathbf{m}}_1$ are computed by the verifier as

$$\hat{\mathbf{m}}_\ell[i] := \sum_{k\in\{0,1\}} \mathbf{M}_{\ell,i}[k, \mathbf{d}_\ell[i]]$$

for $\ell \in \{0,1\}$, $i \in [n]$. For branch $1-b$, the vector $\mathbf{d} \in \mathbb{Z}_q^n$ is chosen by the receiver

such that

$$\sum_{k\in\{0,1\}} \mathbf{M}_{1-b,i}[k, \mathbf{d}_{1-b}[i]] = \hat{\mathbf{m}}'_{1-b}[i]$$

holds for all $i \in [n]$, where $\hat{\mathbf{m}}'_{1-b}[i]$ denotes $\hat{\mathbf{m}}_{1-b}[i]$ on the receiver's side. Further, such a vector $\mathbf{d}_{1-b}$ always exists due to the fact that there are q columns in $\mathbf{M}_{1-b,i}$ and each column sums to a different value in $\mathbb{Z}_q$. For branch b,

$$\sum_{k\in\{0,1\}} \mathbf{M}_{b,i}[k, \mathbf{d}_b[i]] = \hat{\mathbf{m}}'_b[i]$$

holds for any $\mathbf{d}_b \in \mathbb{Z}_q^n$. Therefore the vectors $\hat{\mathbf{m}}_0$ and $\hat{\mathbf{m}}_1$ computed by the verifier are identical to the vectors $\hat{\mathbf{m}}'_0$ and $\hat{\mathbf{m}}'_1$ computed by the prover, which are in particular chosen to be codewords w.r.t. code G for messages m_0 and m_1.

The last part of the checking procedure checks whether the image of ψ for the two rows of $\mathbf{M}$ is indeed consistent with the transmitted value $\mathbf{v}_{k,i}$. More specifically, $\forall k \in \{0,1\}$, $i \in [n]$,

$$\psi(\mathbf{M}_{k,i}[\mathbf{c}_k[i], *]) = (-1)^{\mathbf{c}_k[i]}\mathbf{v}_{k,i}$$

must hold. Again, by construction this is true for all $i \in [n]$ and $k = 1 - b$, simply because $\mathbf{v}_{1-b,i}$ is chosen such that it holds. In case $k = b$ it holds as well, since for each $i \in [n]$ all the columns of $\mathbf{M}_{b,i}$ sum to $\hat{\mathbf{m}}_{b,i}$ or equivalently for all $j \in \mathbb{Z}_q$, $\mathbf{M}_{b,i}[1, j] = \hat{\mathbf{m}}_{b,i} - \mathbf{M}_{b,i}[0, j]$. Due to this fact, for all $c \in \{0,1\}$, $i \in [n]$

$$\begin{aligned}\psi(\mathbf{M}_{b,i}[c, *]) &= (\mathbf{M}_{b,i}[c,1] - \mathbf{M}_{b,i}[c,0], \ldots, \mathbf{M}_{b,i}[c,q-1] - \mathbf{M}_{b,i}[c,0]) \\ &= (-\mathbf{M}_{b,i}[1-c,1] + \mathbf{M}_{b,i}[1-c,0], \ldots, -\mathbf{M}_{b,i}[1-c,q-1] + \mathbf{M}_{b,i}[1-c,0]) \\ &= (-1)\psi(\mathbf{M}_{b,i}[1-c, *])\end{aligned}$$

holds. Further, $\mathbf{v}_{b,i} := \psi(\mathbf{M}_{b.i}[0, *])$ and therefore

$$\psi(\mathbf{M}_{b,i}[\mathbf{c}_b[i], *]) = (-1)^{\mathbf{c}_b[i]}\mathbf{v}_{b,i}$$

holds for any choice of $\mathbf{c}_b \in \{0,1\}^n$. This concludes proving completeness. □

Lemma 2 (Existence of a committing branch for the ORS protocol)**.** *Let $\kappa \in \mathbb{N}$ be a statistical security parameter. Assuming that the commitment scheme* Commit *is statistically binding except with probability at most ϵ, and that code G has minimal distance $\frac{1}{2}(n + \kappa)$, then the ORS protocol from Fig. 3.2 satisfies the property of existence of a committing branch except with probability at most $2\epsilon + 2^{-\kappa}$.*

Proof. We define several hybrids to prove the lemma. In the first hybrid, a malicious prover P^* loses if, for any $i \in [n]$ and any $k \in \{0,1\}$, a partial message $\hat{\mathbf{m}}_k[i]$ differs from $\hat{\mathbf{m}}'_b[i]$ and the opened row of $\mathbf{M}_{k,i}$ differs as well, i.e. $\mathbf{c}_k[i] \neq \mathbf{c}'_k[i]$.

In the second hybrid, the adversary will lose as well if there are more than κ positions $i \in [n]$ for which both messages $\hat{\mathbf{m}}_0[i]$ and $\hat{\mathbf{m}}_1[i]$ differ from the messages $\hat{\mathbf{m}}'_0[i]$ and $\hat{\mathbf{m}}'_1[i]$ of the second run.

Hybrid $\mathbf{H}_0(\lambda)$: This is the original security game, i.e.

$$\begin{aligned}&(\gamma, \alpha_0) \leftarrow\!\!\$\ \mathsf{P}_0^*(1^\lambda);\\ &\beta, \beta' \leftarrow\!\!\$\ \mathsf{V}_0(1^\lambda);\\ &(\delta, m_0, m_1) \leftarrow\!\!\$\ \mathsf{P}_1^*(\alpha_0, \beta);\\ &(\delta', m_0', m_1') \leftarrow\!\!\$\ \mathsf{P}_1^*(\alpha_0, \beta')\end{aligned}$$

and the prover wins iff

$$\begin{aligned}&(\mathsf{V}_1(T) = 1) \wedge (\mathsf{V}_1(T') = 1)\\ \wedge &(m_0 \neq m_0') \wedge (m_1 \neq m_1').\end{aligned}$$

Hybrid $\mathbf{H}_1(\lambda)$: Identical to $\mathbf{H}_0(\lambda)$ except that the prover wins iff

$$\begin{aligned}&(\mathsf{V}_1(T) = 1) \wedge (\mathsf{V}_1(T') = 1)\\ \wedge &(m_0 \neq m_0') \wedge (m_1 \neq m_1')\\ \wedge &\underline{\forall i \in [n], k \in \{0,1\} : (\hat{\mathbf{m}}_k[i] = \hat{\mathbf{m}}_k'[i]) \vee (\mathbf{c}_k[i] = \mathbf{c}_k'[i])}.\end{aligned}$$

Hybrid $\mathbf{H}_2(\lambda)$: Identical to $\mathbf{H}_1(\lambda)$ except the prover wins iff

$$\begin{aligned}&(\mathsf{V}_1(T) = 1) \wedge (\mathsf{V}_1(T') = 1)\\ \wedge &(m_0 \neq m_0') \wedge (m_1 \neq m_1')\\ \wedge &\forall i \in [n], k \in \{0,1\} : (\hat{\mathbf{m}}_k[i] = \hat{\mathbf{m}}_k'[i]) \vee (\mathbf{c}_k[i] = \mathbf{c}_k'[i])\\ \wedge &\underline{|\{i \in [n] \mid \hat{\mathbf{m}}_0[i] \neq \hat{\mathbf{m}}_0'[i] \wedge \hat{\mathbf{m}}_1[i] \neq \hat{\mathbf{m}}_1'[i]\}| < \kappa}.\end{aligned}$$

Claim 1. $\mathbb{SD}\left(\mathbf{H}_0(\lambda); \mathbf{H}_1(\lambda)\right) \leq 2\epsilon$.

Proof. There is a difference between the two hybrids if and only if there is an $i \in [n]$ and $k \in \{0,1\}$ such that

$$(\hat{\mathbf{m}}_k[i] \neq \hat{\mathbf{m}}_k'[i]) \wedge (\mathbf{c}_k[i] \neq \mathbf{c}_k'[i]).$$

By the checking procedure of the verifier, we have

$$\psi(\mathbf{M}_{k,i}[\mathbf{c}_k[i], *]) = (-1)^{\mathbf{c}_k[i]}\mathbf{v}_{k,i},$$

which implies the two equalities

$$\begin{aligned}\mathbf{v}[\mathbf{d}_k[i]] &= \mathbf{M}_{k,i}[0, \mathbf{d}_k[i]] - \mathbf{M}_{k,i}[0,0] = -\mathbf{M}_{k,i}[1, \mathbf{d}_k[i]] + \mathbf{M}_{k,i}[1,0],\\ \mathbf{v}[\mathbf{d}_k[i]] &= \mathbf{M}_{k,i}'[0, \mathbf{d}_k'[i]] - \mathbf{M}_{k,i}'[0,0] = -\mathbf{M}_{k,i}'[1, \mathbf{d}_k'[i]] + \mathbf{M}_{k,i}'[1,0].\end{aligned}$$

Further,

$$\hat{\mathbf{m}}_k[i] = \mathbf{M}_{k,i}[0, \mathbf{d}_k[i]] + \mathbf{M}_{k,i}[1, \mathbf{d}_k[i]] = \mathbf{M}_{k,i}[0,0] + \mathbf{M}_{k,i}[1,0]$$

as well as $\hat{\mathbf{m}}_k'[i] = \mathbf{M}_{k,i}'[0,0] + \mathbf{M}_{k,i}'[1,0]$. Since $\hat{\mathbf{m}}_k[i] \neq \hat{\mathbf{m}}_k'[i]$, either $\mathbf{M}_{k,i}[0,0] \neq \mathbf{M}_{k,i}'[0,0]$ or $\mathbf{M}_{k,i}[1,0] \neq \mathbf{M}_{k,i}'[1,0]$ which breaks statistical binding. □

Claim 2. $\mathbb{SD}(\mathbf{H}_1(\lambda); \mathbf{H}_2(\lambda)) \leq 2^{-\kappa}$.

Proof. A malicious prover P^* is successful in $\mathbf{H}_1(\lambda)$ but not in $\mathbf{H}_2(\lambda)$ if for set

$$\mathcal{S} := \{i \in [n] : \ \hat{\mathbf{m}}_0[i] \neq \hat{\mathbf{m}}'_0[i] \wedge \hat{\mathbf{m}}_1[i] \neq \hat{\mathbf{m}}'_1[i]\}$$

the inequality $|\mathcal{S}| \geq \kappa$ holds. To prove the claim, we show this bound on set $\mathcal{S}$.

For any $i \in [n]$ and $k \in \{0,1\}$, either $\hat{\mathbf{m}}_k[i] \neq \hat{\mathbf{m}}'_k[i]$ or $\mathbf{c}_k[i] \neq \mathbf{c}'_k[i]$ holds. Hence, for all elements i in $\mathcal{S}$, we necessarily have $\mathbf{c}_0[i] = \mathbf{c}'_0[i]$ and $\mathbf{c}_1[i] = \mathbf{c}'_1[i]$. This implies that challenge $\beta = \mathbf{c}_0 + \mathbf{c}_1$ is identical with β' on position i. Since β' is uniformly random, this is only the case with probability $1/2$. If it is not the case, the verifier rejects. Since the size of $\mathcal{S}$ has to be at least κ, the probability of this to happen is at most $2^{-|\mathcal{S}|} \leq 2^{-\kappa}$. □

In $\mathbf{H}_2(\lambda)$, the adversary's choice of $\hat{\mathbf{m}}_0$ and $\hat{\mathbf{m}}_1$ will both differ from $\hat{\mathbf{m}}'_0$ and $\hat{\mathbf{m}}'_1$ on at most κ positions. On all other positions, $\hat{\mathbf{m}}_0$ and $\hat{\mathbf{m}}_1$ will be identical to $\hat{\mathbf{m}}'_0$ and $\hat{\mathbf{m}}'_1$. Since there are $n - \kappa$ positions left, at least one of the pairs will be identical on at least $\frac{1}{2}(n - \kappa)$ positions. Let this be $\hat{\mathbf{m}}_b$.

Due to the minimal distance $\frac{1}{2}(n + \kappa)$ of code G, there is a unique codeword that matches these $\frac{1}{2}(n - \kappa)$ positions. Hence, in both runs, a malicious receiver is committed to $\hat{\mathbf{m}}_b = \hat{\mathbf{m}}'_b$, because if $\hat{\mathbf{m}}_b$ or $\hat{\mathbf{m}}'_b$ is not a codeword, the verifier rejects. Thus, $\hat{\mathbf{m}}_b = \hat{\mathbf{m}}'_b$ decodes to a unique message m_b and therefore for all unbounded provers P^* experiment $\mathbf{H}_2(\lambda)$ returns 1 with zero probability, which concludes this proof. □

Lemma 3 (Choice bit indistinguishability of the ORS protocol)**.** *Assuming that the commitment scheme* Commit *satisfies computational hiding, the ORS protocol from Fig. 3.2 satisfies choice bit indistinguishability.*

Proof. To show indistinguishablity, we define a hybrid in which a prover commits to both messages and both branches will follow the same distribution. Let $\mathbf{H}_0(\lambda, b)$ be the experiment defining choice bit indistinguishability, where the adversary V^* acts as a malicious verifier; our goal is to show that for all PPT V^*, we have $\mathbf{H}_0(\lambda, 0) \approx_c \mathbf{H}(\lambda, 1)$. Consider the hybrid experiment $\mathbf{H}(\lambda, b)$ where in the first round the prover takes the following actions:

$$\begin{aligned}
&\hat{\mathbf{m}}_b := G(m_b), \mathbf{c}_{1-b} \leftarrow\!\!\$\ \{0,1\}^n, \underline{\hat{\mathbf{m}}_{1-b} := G(m_{1-b})} \\
&\mathbf{d}_b, \underline{\mathbf{d}_{1-b}} \leftarrow\!\!\$\ \mathbb{Z}_q^n \\
&\forall i \in [n], \underline{\ell \in \{0,1\}} : \\
&\quad \mathbf{M}_{\ell,i} \leftarrow\!\!\$\ \mathbb{Z}_q^{2\times q} \text{ s.t. } \forall j \in \mathbb{Z}_q : \\
&\qquad \sum_{k\in\{0,1\}} \mathbf{M}_{\ell,i}[k,j] = \hat{\mathbf{m}}_b[i] \\
&\quad \mathbf{v}_{\ell,i} := \psi(\mathbf{M}_{\ell.i}[0,*]) \\
&\gamma := \{\mathbf{v}_{k,i}, \mathsf{Commit}(\mathbf{M}_{k,i})\}_{k\in\{0,1\},i\in[n]},
\end{aligned}$$

and moreover during the third round, the prover acts as follows:

$$\begin{aligned}
&\mathbf{c}_b = \beta - \mathbf{c}_{1-b} \\
&\delta_{\mathbf{c}} := \delta(\mathbf{M}_{k,i}[\mathbf{c}_k[i], *])_{k\in\{0,1\},i\in[n]} \\
&\delta_{\mathbf{d}} := \delta(\mathbf{M}_{k,i}[1-\mathbf{c}_k[i], \mathbf{d}_k[i]])_{k\in\{0,1\},i\in[n]}
\end{aligned}$$

Notice that sampling first $\mathbf{c}_{1-b}$ and setting $\mathbf{c}_b = \beta - \mathbf{c}_{1-b}$ has the same distribution as $\mathbf{c}_0, \mathbf{c}_1 \leftarrow\!\!\$\ \{0,1\}^n$ conditioned on $\beta = \mathbf{c}_0 + \mathbf{c}_1$. Therefore both branches have the same distribution.

Claim 3. *For all PPT* V^**, and for all* $b \in \{0,1\}$*, we have that* $\mathbf{H}_0(\lambda, b) \approx_c \mathbf{H}_1(\lambda, b)$*.*

Proof. We will define $n(q-1)$ sub-hybrids. For each $i \in [n]$, there are $q-1$ commitments in branch $b-1$ that are not opened in the third round. We will switch their committed value $\mathbf{M}_{1-\mathbf{c}_i,i}$ step by step from the distribution in $\mathbf{H}_0$ to the distribution in $\mathbf{H}_1$, i.e. from being uniform conditioned on summing to $\sigma_i(j)$ to summing to $\hat{\mathbf{m}}[i]$.

We denote the sub hybrids with $\mathbf{H}_{0,0,0}(\lambda, b)$ to $\mathbf{H}_{0,n,q}(\lambda, b)$, where $\mathbf{H}_{0,0,0}(\lambda, b) \equiv \mathbf{H}_0(\lambda, b)$ and $\mathbf{H}_{0,n,q}(\lambda, b) \equiv \mathbf{H}_1(\lambda, b)$. We switch from $\mathbf{H}_{0,i,j}(\lambda, b)$ to $\mathbf{H}_{0,i,j+1}(\lambda, b)$, and from $\mathbf{H}_{0,i,q}(\lambda, b)$ to $\mathbf{H}_{0,i+1,0}(\lambda, b)$. In the following, we will just show how to transition from $\mathbf{H}_{0,i,j}(\lambda, b)$ to $\mathbf{H}_{0,i,j+1}(\lambda, b)$. The other step is done analogously. Further notice that the the hybrids

$$\mathbf{H}_{0,i,\mathbf{d}_{1-b}[i]-1}(\lambda, b) \quad \text{and} \quad \mathbf{H}_{0,i,\mathbf{d}_{1-b}[i]}(\lambda, b)$$

are already distributed identically. Next, we show that for any $i^* \in [n]$, $j^* \in \mathbb{Z}_q$, and for all PPT V^* and $b \in \{0,1\}$, hybrids $\mathbf{H}_{0,i^*,j^*}(\lambda, b)$ and $\mathbf{H}_{0,i^*,j^*+1}(\lambda, b)$ are computationally close, which finishes the proof of the claim.

Recall that an adversary A against the hiding of the commitment scheme chooses two messages $\tilde{m}_0$ and $\tilde{m}_1$, and receives a commitment $\tilde{\gamma}$ of one of the two messages. We denote this by $\tilde{\gamma} \leftarrow\!\!\$\ \mathcal{O}_{\mathsf{Commit}}(\tilde{m}_0, \tilde{m}_1)$. Attacker A wins if he successfully determines which message has been committed to. In what follows, we mostly ignore branch b since it has the same distribution in both hybrids. In the first round, A

simulates the prover as follows.

$$
\begin{aligned}
&\mathbf{c}_{1-b} \leftarrow\!\!\$\ \{0,1\}^n, \hat{\mathbf{m}}_{1-b} := Gm_{1-b}, \mathbf{d}_{1-b} \leftarrow\!\!\$\ \mathbb{Z}_q^n \\
&\forall (i < i^* \vee (i = i^* \wedge j \leq j^*)), \sigma_i(j) := \hat{\mathbf{m}}_{1-b}[i] \\
&\sigma' \leftarrow\!\!\$\ \mathsf{Perm}(\mathbb{Z}_q) \text{ s.t. } \sigma'(\mathbf{d}_{1-b}[i^*]) = \hat{\mathbf{m}}_{1-b}[i^*] \\
&\forall j > j^*, \sigma_{i^*}(j) := \sigma'(j) \\
&\forall i > i^*, \sigma_i \leftarrow\!\!\$\ \mathsf{Perm}(\mathbb{Z}_q) \text{ s.t. } \sigma_i(\mathbf{d}_{1-b}[i]) = \hat{\mathbf{m}}_{1-b}[i] \\
&\forall i \in [n], \mathbf{M}_{1-b,i} \leftarrow\!\!\$\ \mathbb{Z}_q^{2\times q} \text{ s.t.} \\
&\qquad \sum_{k\in\{0,1\}} \mathbf{M}_{1-b,i}[k,j] = \sigma_i(j) \\
&\forall i < i^*, \mathbf{v}_{1-b,i} := \psi(\mathbf{M}_{1-b,i}[0,*]) \\
&\forall i \geq i^*, \mathbf{v}_{1-b,i} := (-1)^{\mathbf{c}_{1-b}[i]}\psi(\mathbf{M}_{1-b.i}[\mathbf{c}_{1-b}[i],*]) \\
&\forall (i \neq i^* \vee j \neq j^* \vee k \neq \mathbf{c}_{1-b}[i]), \gamma_{i,k,j} \leftarrow\!\!\$\ \mathsf{Commit}(\mathbf{M}_{1-b,i}[k,j]) \\
&\gamma_{i^*,\mathbf{c}_{1-b}[i^*],j^*} \leftarrow\!\!\$\ \mathcal{O}_{\mathsf{Commit}}(\sigma'[j^*] - \mathbf{M}_{1-b,i^*}[\mathbf{c}_b[i^*], j^*], \mathbf{M}_{1-b,i^*}[\mathbf{c}_{1-b}[i^*], j^*]) \\
&\gamma := \{\mathbf{v}_{0,i}, \mathbf{v}_{1,i}, \mathsf{Commit}(\mathbf{M}_{b,i}), (\gamma_{i,k,j})_{k\in\{0,1\},j\in[q]}\}_{i\in[n]}.
\end{aligned}
$$

Since A does not open $\gamma_{i^*,\mathbf{c}_{1-b}[i^*],j^*}$, he can easily simulate the third round:

$$
\begin{aligned}
&\mathbf{c}_b = \beta - \mathbf{c}_{1-b} \\
&\delta_{\mathbf{c}} := \delta(\mathbf{M}_{k,i}[\mathbf{c}_k[i],*])_{k\in\{0,1\},i\in[n]} \\
&\delta_{\mathbf{d}} := \delta(\mathbf{M}_{k,i}[1-\mathbf{c}_k[i], \mathbf{d}_k[i]])_{k\in\{0,1\},i\in[n]}.
\end{aligned}
$$

If the challenger of the commitment security game commits to message $\sigma'(j^*) - \mathbf{M}_{1-b,i^*}[\mathbf{c}_b[i^*], j^*]$, attacker A simulates hybrid $\mathbf{H}_{0,i^*,j^*}(\lambda, b)$, and otherwise if the challenger commits to $\mathbf{M}_{1-b,i^*}[\mathbf{c}_{1-b}[i^*], j^*]$ the attacker simulates hybrid $\mathbf{H}_{0,i^*,j^*+1}(\lambda, b)$. This concludes the proof of this claim.

□

Clearly, the distribution of hybrid $\mathbf{H}_1(\lambda, b)$ is independent of bit b. Therefore, $\mathbf{H}_1(\lambda, 0) \equiv \mathbf{H}_1(\lambda, 1)$. This and the previous claim result in the statement of the lemma. □

3.3 Strongly Uniform PKE, Key Agreement and OT

3.3.1 Strongly Uniform PKE

We start with defining strongly uniform public-key encryption (PKE). Here, we differ between two types of PKE. A Type-A PKE has a public key that is computationally close to uniform, while for a Type-B PKE this is the case for ciphertexts of uniform messages (under malicious public keys).

In general, a PKE scheme Π_{pke} consists of three efficient algorithms ($\mathsf{Gen}, \mathsf{Enc}, \mathsf{Dec}$) specified as follows. (i) The probabilistic algorithm Gen takes as input the security parameter and outputs a pair of keys $(\mathsf{pk}, \mathsf{sk})$; (ii) The probabilistic algorithm Enc takes as input the public key pk and a message $\mu \in \mathcal{M}$, and returns a

ciphertext $c \in \mathcal{C}$; (iii) The deterministic algorithm Dec takes as input the secret key sk and a ciphertext $c \in \mathcal{C}$, and returns a value $\mu \in \mathcal{M} \cup \{\perp\}$. We say that $\Pi_{\mathtt{pke}}$ meets correctness, if for all $\lambda \in \mathbb{N}$, all $(\mathsf{pk}, \mathsf{sk})$ output by $\mathsf{Gen}(1^\lambda)$, and all $\mu \in \mathcal{M}$ the following holds: $\mathbb{P}\left[\mathsf{Dec}(\mathsf{sk}, \mathsf{Enc}(\mathsf{pk}, \mu)) = \mu\right] = 1$.

Definition 18 (Strongly uniform Type-A PKE)**.** A PKE scheme $\Pi_{\mathtt{pke}} = (\mathsf{Gen}, \mathsf{Enc}, \mathsf{Dec})$ is called a strongly uniform Type-A PKE if for any PPT distinguisher $\mathcal{D}$ the following holds:

$$|\Pr[\mathcal{D}(\mathsf{pk}) = 1] - \Pr[\mathcal{D}(u) = 1]| \in \mathsf{negl}(\lambda)\,,$$

where $(\mathsf{pk}, \mathsf{sk}) \leftarrow\!\!\$\ \mathsf{Gen}(1^\lambda)$ and u is uniform over a suitable, efficiently sampleable group.

In case of strongly uniform Type-B PKE, we even ask that a ciphertext of a uniform message is indistinguishable from uniform to a distinguisher that chooses a public key for the encryption procedure in an arbitrary way.

Definition 19 (Strongly uniform Type-B PKE)**.** A PKE scheme $\Pi_{\mathtt{pke}} = (\mathsf{Gen}, \mathsf{Enc}, \mathsf{Dec})$ is called a strongly uniform Type-B PKE if for any PPT distinguisher $\mathcal{D}$ the following holds:

$$|\Pr[\mathcal{D}(c) = 1] - \Pr[\mathcal{D}(u) = 1]| \in \mathsf{negl}(\lambda)\,,$$

where $\mathsf{pk} \in \{0,1\}^*$ is chosen by $\mathcal{D}$, $\mu \leftarrow\!\!\$\ \mathcal{M}$, $c \leftarrow\!\!\$\ \mathsf{Enc}(\mathsf{pk}, \mu)$, and u is uniform over a suitable, efficiently sampleable group.

When using PKE in the following, we also ask for standard security against chosen plaintext attacks, since this is not implied by the notion of strong uniformity in all generality.

3.3.2 Strongly Uniform Key Agreement

Let $\Pi_{\mathtt{ka}} = (\mathsf{P}_1, \mathsf{P}_2)$ be a key agreement (KA) protocol, where P_1 sends messages during r' rounds, which we denote by $\rho^1, \ldots, \rho^{r'}$. The messages from P_2 to P_1 are denoted with $\sigma^1, \ldots, \sigma^{r'+1}$, and are at most $r'+1$. W.l.o.g. we will assume that P_2 sends the last message.

More precisely, algorithms P_1 and P_2 are stateful interactive Turing machines such that for each $i \in [r']$: (i) Algorithm P_1 takes the current state information $\alpha_{\mathsf{P}_1}^{i-1}$ (where $\alpha_{\mathsf{P}_1}^0$ is equal to P_1's input 1^λ) and a message σ^{i-1} from P_2 (with σ^0 empty), and returns ρ^i together with updated state information $\alpha_{\mathsf{P}_1}^i$; (ii) Algorithm P_2 takes the current state information $\alpha_{\mathsf{P}_2}^{i-1}$ (where $\alpha_{\mathsf{P}_2}^0$ is equal to P_2's input 1^λ) and message ρ^i from the receiver, and returns σ^i together with updated state information $\alpha_{\mathsf{P}_2}^i$.

For strong uniformity, we ask that P_1's messages are computationally close to uniform over an efficiently sampleable group $\mathcal{M}$. For simplicity, we assume that this is the same group for all messages. Our results still hold when the messages are uniform in different groups.

Additionally, we ask that given a transcript, one cannot distinguish the key P_1 and P_2 agreed upon from a uniformly random string.

Definition 20 (Strongly uniform secure key agreement)**.** A KA protocol $\Pi_{\mathtt{ka}} = (\mathsf{P}_1, \mathsf{P}_2)$ as defined above is a strongly uniform secure KA if there exists an efficiently samplable group $\mathcal{M}$ such that the messages $(\rho^1, \ldots, \rho^{r'})$ sent by P_1 in a honest execution of the protocol are distributed over $\mathcal{M}$, and moreover the following conditions are met:

(a) **Key Indistinguishability:** For an honest execution of the protocol with agreed key K,

$$\left(\langle \mathsf{P}_1(1^\lambda), \mathsf{P}_2(1^\lambda)\rangle, K\right) \approx_c \left(\langle \mathsf{P}_1(1^\lambda), \mathsf{P}_2(1^\lambda)\rangle, U\right),$$

where U is uniform and independent of the view of P_1 and P_2.

(b) **Uniformity w.r.t. Malicious Interaction:** For all PPT distinguishers $\mathcal{D}$ the following quantity is negligible:

$$\left| \Pr\left[\mathcal{D}(\alpha^{r'}_{\mathcal{D}}, (\rho^i, \sigma^i)_{i\in[r']}) = 1 : \begin{array}{c} \forall i \in [r'], (\alpha^i_{\mathsf{P}_1}, \rho^i) \leftarrow\!\!\$\ \mathsf{P}_1(\alpha^{i-1}_{\mathsf{P}_1}, \sigma^i) \\ \wedge\ (\alpha^i_{\mathcal{D}}, \sigma^i) \leftarrow\!\!\$\ \mathcal{D}(\alpha^{i-1}_{\mathcal{D}}, \rho^{i-1}) \end{array}\right] \right.$$
$$\left. - \Pr\left[\mathcal{D}(\alpha^{r'}_{\mathcal{D}}, (\rho^i, \sigma^i)_{i\in[r']}) = 1 : \begin{array}{c} \forall i \in [r'], \rho^i \leftarrow\!\!\$\ \mathcal{M} \\ \wedge\ (\alpha^i_{\mathcal{D}}, \sigma^i) \leftarrow\!\!\$\ \mathcal{D}(\alpha^{i-1}_{\mathcal{D}}, \rho^{i-1}) \end{array}\right] \right|,$$

where ρ^0 is the empty string, and $\alpha^0_{\mathsf{P}_1} = \alpha^0_{\mathsf{P}_2} = 1^\lambda$.

We show now that the property of strong uniformity is preserved within known construction of KA from Type-A or Type-B PKE, as well as Type-A PKE from KA. Both of the following lemmata are straightforward, and therefore we forego a more formal proof and just sketch them.

Lemma 4. *There exists a 2-round strongly uniform secure KA if and only if there exists a strongly uniform CPA-secure Type-A PKE (constructive).*

Proof. It is a well known fact that 2-round KA implies PKE and vice versa. What we will show is that this construction preserves strong uniformity. In the construction of KA from PKE the receiver sends a public key and receives back an encryption of a uniform key. If the public key is indistinguishable from uniform with all but negligible probability, then all the receivers messages are, and hence the KA is strongly uniform.

In the construction of PKE from KA, one uses the first message of the KA as public key. In a 2-round strongly uniform KA this message is indistinguishable from uniform with all but negligible probability by definition. Hence, the public key is computationally indistinguishable from uniform with all but negligible probability. □

Lemma 5. *If there exists a strongly uniform CPA-secure Type-B PKE, then there exists a 3-round strongly uniform secure KA (constructive).*

Gertner *et al.* [GKM$^+$00] showed a similar lemma, namely that Type-B PKE implies 3-round semi-honestly secure OT. For simplicity, we prefer showing that there is a 3-round strongly uniform secure KA given a strongly uniform CPA-secure Type-B PKE.

Proof. The idea is simple and similar to the proof of Lemma 4. P_1 sends a public key, P_2 sends an encryption of a uniform key. Finally, P_2 decrypts the ciphertext and sends a dummy message. The last message is required by the definition of strongly uniform KA, which asks that P_1's messages are indistinguishable from uniform, where P_2 sends the last message.

In order to achieve strongly uniform KA, even for maliciously chosen public key, the ciphertext needs to be indistinguishable from uniform with all but negligible probability. Type-B PKE has this property, and hence the described protocol is strongly uniform. Security follows trivially, and for identical reasons, as in Lemma 4. □

3.3.3 Strongly Uniform OT

In an OT protocol $\Pi = (\mathsf{S}, \mathsf{R})$ we can w.l.o.g. assume that the sender S always speaks last. We use the same notation as described above for a key agreement protocol. In particular, $\rho^1, \ldots, \rho^{r'}$ are the messages from R to S, and $\sigma^1, \ldots, \sigma^{r'+1}$ the messages from S to R. The initial states are identical with the inputs, i.e. $\alpha^0_{\mathsf{R}} = b \in \{0,1\}$ and $\alpha^0_{\mathsf{S}} = (s_0, s_1) \in \{0,1\}^{2\lambda}$.

Correctness means that for all $b \in \{0,1\}$, and for all $s_0, s_1 \in \{0,1\}^\lambda$, the following probability is overwhelming:

$$\Pr\left[\rho^{r'+1} = s_b : \forall i \in [r'+1], (\alpha^i_{\mathsf{R}}, \rho^i) \leftarrow\!\!\$\ \mathsf{R}(\alpha^{i-1}_{\mathsf{R}}, \sigma^i) \wedge (\alpha^i_{\mathsf{S}}, \sigma^i) \leftarrow\!\!\$\ \mathsf{S}(\alpha^{i-1}_{\mathsf{S}}, \rho^{i-1})\right],$$

where ρ^0 is the empty string, and $\alpha^0_{\mathsf{S}} = (s_0, s_1)$, $\alpha^0_{\mathsf{R}} = b$.

As for security, we require two properties. The first property is equivalent to simulation-based security for honest-but-curious receivers. The second property says that a malicious sender cannot distinguish the case where it is interacting with the honest receiver, from the case where the messages from the receiver are replaced by uniform elements over an efficiently sampleable group $\mathcal{M}$.

Definition 21 (Strongly Uniform semi-honestly secure OT)**.** An OT protocol $\Pi = (\mathsf{S}, \mathsf{R})$ as defined above is a strongly uniform semi-honestly secure OT if there exists an efficiently samplable group $\mathcal{M}$ such that the messages $(\rho^1, \ldots, \rho^{r'})$ sent by R in a honest execution of the protocol are distributed over $\mathcal{M}$, and moreover the following conditions are met:

(a) **Security w.r.t. Semi-Honest Receivers:** There exists a PPT simulator $\mathsf{Sim}_{\mathsf{R}}$ such that for all $b \in \{0,1\}$ and for all $s_0, s_1 \in \{0,1\}^\lambda$ the following holds:

$$\left\{\mathsf{Sim}_{\mathsf{R}}(1^\lambda, b, s_b)\right\}_{\lambda, b, s_b} \approx_c \left\{view^{\mathsf{R}}_{\Pi}(\lambda, s_0, s_1, b)\right\}_{\lambda, s_0, s_1, b},$$

where $view^{\mathsf{R}}_{\Pi}(\lambda, s_0, s_1, b)$ denotes the distribution of the view of the honest receiver at the end of the protocol.

(b) **Uniformity w.r.t. Malicious Senders:** For all PPT distinguishers $\mathcal{D}$, and

for all $b \in \{0,1\}$, the following quantity is negligible:

$$\left| \Pr\left[\mathcal{D}(\alpha_{\mathcal{D}}^{r'}, (\rho^i, \sigma^i)_{i\in[r']}) = 1 : \begin{array}{c} \forall i \in [r'], (\alpha_{\mathsf{R}}^i, \rho^i) \leftarrow\!\!\$\ \mathsf{R}(\alpha_{\mathsf{R}}^{i-1}, \sigma^{i-1}) \\ \wedge\ (\alpha_{\mathcal{D}}^i, \sigma^i) \leftarrow\!\!\$\ \mathcal{D}(\alpha_{\mathcal{D}}^{i-1}, \rho^{i-1}) \end{array}\right] \right.$$
$$\left. - \Pr\left[\mathcal{D}(\alpha_{\mathcal{D}}^{r'}, (\rho^i, \sigma^i)_{i\in[r']}) = 1 : \begin{array}{c} \forall i \in [r'], \rho^i \leftarrow\!\!\$\ \mathcal{M} \\ \wedge\ (\alpha_{\mathcal{D}}^i, \sigma^i) \leftarrow\!\!\$\ \mathcal{D}(\alpha_{\mathcal{D}}^{i-1}, \rho^{i-1}) \end{array}\right] \right|,$$

where ρ^0 is the empty string, and $\alpha_{\mathsf{R}}^0 = b$.

Note that the second property implies game-based security w.r.t. malicious senders (i.e., property (b) of Definition 14). Furthermore, for the special case of $r' = 1$ the above definition collapses to standard semi-honest security, as the only message sent by the malicious sender plays no role in distinguishing the two distributions.

Next, we show a lemma that is not very surprising, namely that strongly uniform secure KA can be constructed from strongly uniform semi-honestly secure OT.

Lemma 6. *If there exists a r-round strongly uniform semi-honestly secure OT, then there exists a r-round strongly uniform secure KA (constructive).*

Proof. We construct a r-round KA $\Pi_{\mathtt{ka}}$ from a r-round OT Π as follows. P_1 and P_2 run Π, where P_1 takes the role of the receiver with choice bit 0. P_2 takes the role of the sender, with inputs equal to a uniform key k, i.e. $s_0 = k$, and a uniformly random string u, i.e. $s_1 = u$.

Given that Π is semi-honestly secure, Gertner *et al.* [GKM$^+$00, Theorem 5] have shown that $\Pi_{\mathtt{ka}}$ is indeed a secure KA. The rough idea is to first switch the receiver's choice bit to 1 by using the game-based security of Π against honest-but-curious senders (which in our case is implied by strong uniformity). Afterwards, we can use the security against an honest-but-curious receiver to argue that an eavesdropper cannot distinguish $s_0 = k$ from random anymore, since even the receiver can only learn s_1 but has no information about s_0. Therefore $\Pi_{\mathtt{ka}}$ is a secure KA. It remains to prove strong uniformity.

Claim 4. *Assuming Π is strongly uniform, so is $\Pi_{\mathtt{ka}}$.*

Proof. Let PPT D' break the strong uniformity of $\Pi_{\mathtt{ka}}$, then we construct a PPT distinguisher D that breaks the strong uniformity of Π as follows. Distinguisher D chooses k and u uniformly, and interacts as a honest sender in Π, where the receiver's messages are either distributed according to the protocol description or uniform. Hence, P_1's messages are either conform with the protocol or uniform. Distinguisher D' receives the view of P_2 generated by D. Now, if D' distinguishes the messages of P_1 being conform with the protocol from uniform, D breaks the strong uniformity of Π. □

□

The next lemma is more surprising, as it implies that strongly uniform secure KA is equivalent to strongly uniform semi-honestly secure OT. Hence, the notion of strong uniformity is sufficiently strong to bypass the black-box separation of KA

and OT by Gertner *et al.* [GKM+00, Corollary 7], which is a consequence of the separation between PKE and OT, and the fact that 2-round KA implies PKE. The above also implies that 2-round secure KA is separated as well from 2-round strongly uniform secure KA.

Lemma 7. *If there exists a r-round strongly uniform secure KA, then there exists a r-round strongly uniform semi-honestly secure OT (constructive).*

Proof. We construct an OT protocol Π using two parallel executions of a KA protocol $\Pi_{\mathtt{ka}}$, which we denote with $\Pi^0_{\mathtt{ka}}$ and $\Pi^1_{\mathtt{ka}}$. The receiver of the OT acts in both executions as P_1. For his choice bit b, he runs $\Pi^b_{\mathtt{ka}}$ according to the protocol description, and in $\Pi^{1-b}_{\mathtt{ka}}$ he samples and sends uniform messages.

In the last round the sender sends $k_0 + s_0$ and $k_1 + s_1$, where for $j \in \{0,1\}$ the key k_j is the exchanged key in $\Pi^j_{\mathtt{ka}}$, and s_0, s_1 are the OT inputs of the sender. Notice that this is a r-round protocol, since the sender can send his masked inputs together with his last messages of the KA protocols.

Claim 5. *Assuming $\Pi_{\mathtt{ka}}$ is strongly uniform, so is Π.*

Proof. Let there be a PPT distinguisher D' that distinguishes the receiver's messages in Π from uniform. We construct a PPT distinguisher D for P_1's messages using D'. Distinguisher D acts in $\Pi_{\mathtt{ka}}$ as P_2, where P_1's messages are either distributed according to the protocol description or uniform. Hence, D picks $b \leftarrow\!\!\$\ \{0,1\}$ and uses the messages sent by D' in Π to interact with P_1 in $\Pi^b_{\mathtt{ka}}$. For $\Pi^{1-b}_{\mathtt{ka}}$, distinguisher D sends uniform messages as in the protocol description. Finally, D outputs the output of D'. Hence, if D' is successful, then so is D. □

Claim 6. *Assuming $\Pi_{\mathtt{ka}}$ is strongly uniform and secure, then Π is secure against honest-but-curious receivers.*

Proof. We use the following hybrids, where a simulator Sim generates a view of the receiver. In the last hybrid, Sim only uses s_b but not s_{1-b} and therefore implements a simulator $\mathsf{Sim}(1^\lambda, b, s_b)$ as required for security against honest-but-curious receivers.

$\mathbf{H}_1(\lambda)$: Sim generates the receivers messages in $\Pi^{1-b}_{\mathtt{ka}}$ as in an actual key agreement $\Pi_{\mathtt{ka}}$, i.e. not uniform as in Π. The receiver's view only contains the messages, not the randomness used in $\Pi_{\mathtt{ka}}$ to generate these messages.

$\mathbf{H}_2(\lambda)$: Sim sends a uniform value u instead of $k_{1-b} + s_{1-b}$.

To prove the claim, we need to show that the receiver's view in the real protocol is indistinguishable from $\mathbf{H}_1(\lambda)$, and that $\mathbf{H}_1(\lambda)$ is indistinguishable from $\mathbf{H}_2(\lambda)$.

Let D' be a PPT distinguisher that distinguishes the recevier's view in the real protocol from $\mathbf{H}_1(\lambda)$ with non-negligible probability. We show that there is a PPT distinguisher D that breaks the strong uniformity of the KA $\Pi_{\mathtt{ka}}$ with the same probability. Distinguisher D runs Π, but replaces the interaction in $\Pi^{1-b}_{\mathtt{ka}}$ on the receiver's side with a challenge instance of $\Pi_{\mathtt{ka}}$ against the strong uniformity. To simulate the view of the receiver correctly, we need to simulate the sampling procedure of the uniform messages in the protocol given only the challenge messages. We can do this by using the simulator $\mathsf{SimSamp}$ of efficiently sampleable groups.

The challenge messages are either uniform, as in the receiver's actual view in Π, or honestly generated, as in $\mathbf{H}_1(\lambda)$. Otherwise, D acts exactly according to the protocol description of Π. If D' distinguishes the two cases, D breaks the uniformity of $\Pi_{\mathtt{ka}}$.

Now let D' be a PPT distinguisher that distinguishes $\mathbf{H}_1(\lambda)$ from $\mathbf{H}_2(\lambda)$ with non-negligible probability. Then we can construct a PPT distinguisher that breaks security, i.e. key indistinguishability, of Π with the same probability. Distinguisher D receives a transcript of $\Pi_{\mathtt{ka}}$ and a challenge z which is either the key k or uniform. Hence, D uses the transcript of $\Pi_{\mathtt{ka}}$ as transcript of $\Pi_{\mathtt{ka}}^{1-b}$, and the challenge z to generate the message $k_{1-b} + s_{1-b}$ as $z + s_{1-b}$. Finally, D generates the remaining parts of the receiver's view honestly. If $z = k$, then D simulates $\mathbf{H}_1(\lambda)$, and if $z = u$, and hence $z + s_{1-b}$ is uniform, D simulates $\mathbf{H}_2(\lambda)$. Hence, whenever D' distinguishes $\mathbf{H}_1(\lambda)$ from $\mathbf{H}_2(\lambda)$, then D distinguishes the actual key from uniform.

□

□

3.4 From Strongly Uniform Semi-Honestly Secure OT to Maliciously Secure OT

Our protocol is described in Section 3.4.1. In Section 3.4.2 we provide a somewhat detailed proof sketch, whereas in Section 3.4.3 we formally show the protocol satisfies receiver-sided simulatability; recall that by using Theorem 3 we immediately get a fully simulatable OT protocol.

3.4.1 Protocol Description

Let $\Pi_{\mathtt{c\&o}} = (\mathsf{P}_0, \mathsf{P}_1, \mathsf{V}_0, \mathsf{V}_1)$ be a 1-out-of-2 C&O protocol and $\Pi' = (\mathsf{S}', \mathsf{R}')$ be a $(2r'+1)$-round OT protocol, where the first message σ^1 might be the empty string. Our OT protocol $\Pi = (\mathsf{S}, \mathsf{R})$ is depicted in Fig. 3.3 on the following page. The protocol consists of $(2r' + 2)$ rounds as informally described below.

1. The receiver samples $m_{1-b,i} \in \mathcal{M}$ for all $i \in [r']$, where b is the choice bit. Then he runs the prover of the C&O protocol upon input $(m_{1-b,j})_{j\in[i]}$ for all $i \in [r']$, obtaining $(\gamma_i)_{i\in[r']}$ which are forwarded to the sender.

2. The sender samples uniform values $u_0, u_1 \leftarrow\!\!\$\ \mathcal{M}$. Then, he runs the underlying $(2r'+1)$-round OT twice with inputs (s_0, u_0) and (s_1, u_1) to generate the first messages σ_0^1 and σ_1^1. Further, the sender samples a challenge β_1 for the C&O protocol, as well as two uniformly random group elements $r_{0,1}$, $r_{1,1}$ from $\mathcal{M}$, and forwards $(\beta_1, r_{0,1}, r_{1,1})$ to the receiver together with the first messages of the OTs (i.e. σ_0^1 and σ_1^1).

3. Repeat the following steps for each $i \in [r']$:

   (a) $(\mathsf{R} \to \mathsf{S})$: The receiver runs the receiver R' of the underlying $(2r' + 1)$-round OT protocol with choice bit fixed to 0, and upon input message σ_b^i

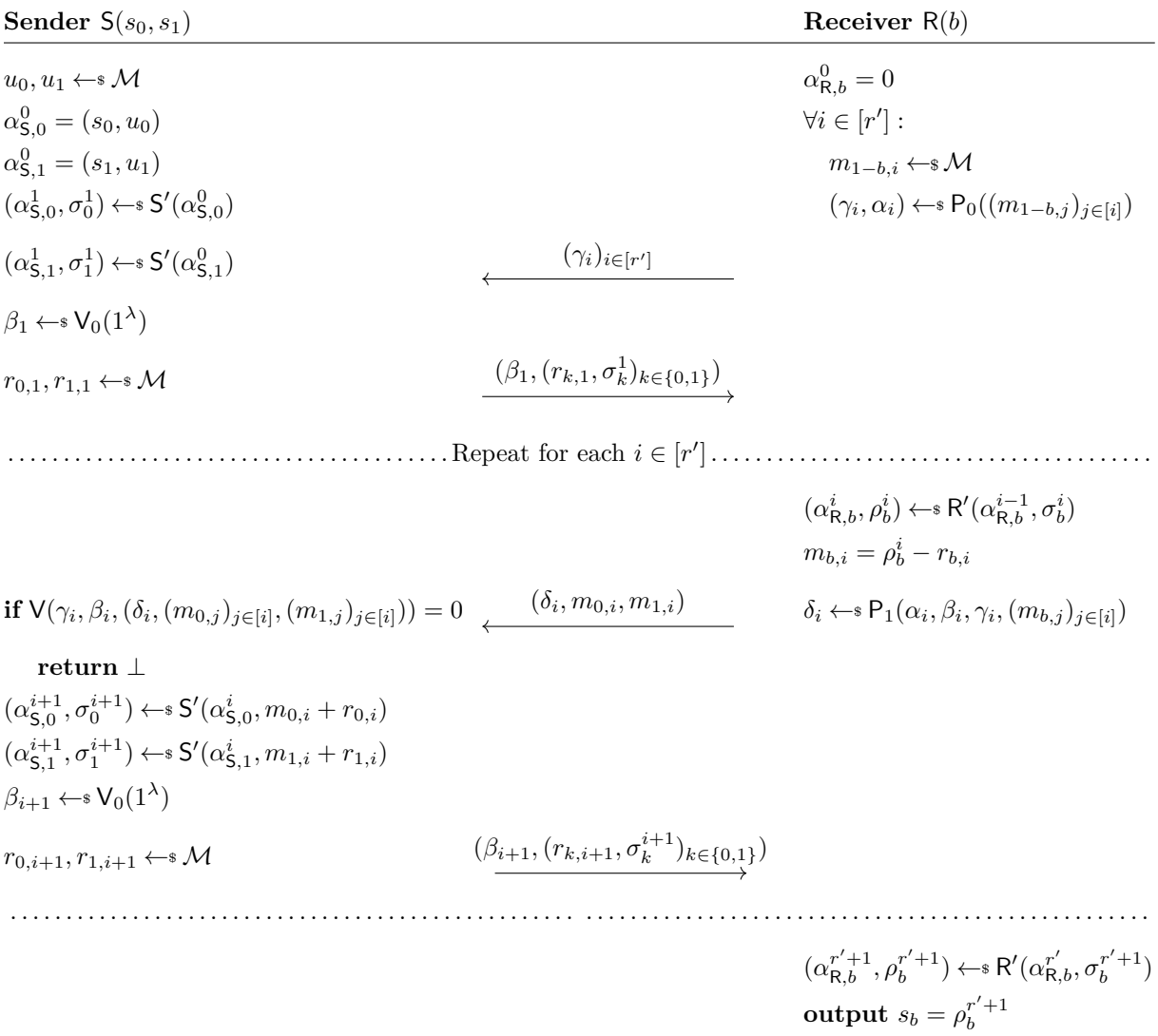


Figure 3.3. $(2r' + 2)$-round OT protocol achieving receiver-sided simulatability from $(2r' + 1)$-round strongly uniform semi-honestly secure OT. Note that the initial state information $\alpha^0_{\mathsf{S},0}, \alpha^0_{\mathsf{S},1}$ and $\alpha^0_{\mathsf{R},b}$ is set to be equal, respectively to the inputs used by the sender and the receiver during the runs of the underlying OT protocol $(\mathsf{S}', \mathsf{R}')$. The values $\beta_{r'+1}, r_{0,r'+1}, r_{1,r'+1}$ are not needed and can be removed, but we avoided to do that in order to keep the protocol description more compact.

from the sender, obtaining a message ρ^i_b which is used to define the message $m_{b,i} = \rho^i_b - r_{b,i}$ required to complete the execution of the C&O protocol in the non-committing branch b. This results in a tuple $(\delta_i, m_{0,i}, m_{1,i})$ that is forwarded to the sender.

(b) $(\mathsf{S} \to \mathsf{R})$: The sender verifies that the transcript $T_i = (\gamma_i, \beta_i, (\delta_i, (m_{0,j})_{j\in[i]}, (m_{1,j})_{j\in[i]}))$ is accepting for the underlying C&O protocol. If so, he continues the two runs of the sender S' for the underlying $(2r'+1)$-round OT protocol. The first run uses state $\alpha^i_{\mathsf{S},0}$ and message $m_{0,i} + r_{0,i}$ from the

receiver resulting in a message σ_0^{i+1} and state $\alpha_{\mathsf{S},0}^{i+1}$, whereas the second run uses state $\alpha_{\mathsf{S},1}^{i}$ and message $m_{1,i}+r_{1,i}$ from the receiver resulting in a message σ_1^{i+1} and state $\alpha_{\mathsf{S},1}^{i+1}$. Finally, the sender samples a challenge β_{i+1} for the C&O protocol, as well as another two uniformly random group elements $r_{0,i+1}$, $r_{1,i+1}$ from $\mathcal{M}$, and forwards $(\sigma_0^{i+1}, \sigma_1^{i+1})$ and β_{i+1}, $r_{0,i+1}$, $r_{1,i+1}$ to the receiver.

4. Output: The receiver runs the receiver R' of the underlying $(2r'+1)$-round OT protocol, upon input the $(r'+1)$-th message $\sigma_b^{r'+1}$ from the sender, thus obtaining an output $\rho_b^{r'+1}$.

Correctness follows by the fact that, when both the sender and the receiver are honest, by correctness of the C&O protocol the transcripts T_i are always accepting, and moreover the messages produced by the sender σ_b^i are computed using message $m_{b,i} + r_{b,i} = \rho_b^i$ from the receiver, so that each pair (ρ_b^i, σ_b^i) corresponds to the i-th interaction of the underlying $(2r'+1)$-round OT protocol with input strings (s_b, u_b) for the sender and choice bit 0 for the receiver, and thus at the end the receiver outputs s_b. As for security, we have:

Theorem 4 (Receiver-sided simulatability of Π)**.** *Assuming that Π' is a $(2r'+1)$-round strongly uniform semi-honestly secure OT protocol, and that $\Pi_{\mathsf{c\&o}}$ is a secure 1-out-of-2 commit-and-open protocol, then the protocol Π from Fig. 3.3 securely realizes $\mathcal{F}_{\mathsf{OT}}$ with receiver-sided simulation.*

3.4.2 Proof Intuition

We give a detailed proof in Section 3.4.3, and here provide some intuition. In order to show receiver-sided simulatability we need to prove two things: (1) The existence of a simulator Sim which by interacting with the ideal functionality $\mathcal{F}_{\mathsf{OT}}$ can fake the view of any efficient adversary corrupting the receiver in a real execution of the protocol; (2) Indistinguishability of the protocol transcripts with choice bit of the receiver equal to zero or one, for any efficient adversary corrupting the sender in a real execution of the protocol.

To show (1), we consider a series of hybrid experiments that naturally lead to the definition of a simulator in the ideal world. In order to facilitate the description of the hybrids, it will be useful to think of the protocol as a sequence of r' iterations, where each iteration consists of 2 rounds, as depicted in Fig. 3.3 on the preceding page.

- In the first hybrid, we run a malicious receiver twice after he has sent his commitments. The purpose of the first run is to learn a malicious receiver's input bit, i.e. on which branch he is not committed. If he is committed on both branches, simulation will be easy since he will not be able to receive any of the sender's inputs. We use the second run to learn the output of a malicious receiver. We describe the two runs now.

  1. The first round of each iteration yields an opening $(\delta_i, m_{0,i}, m_{1,i})$. Hence, after verifying that the opening is valid, we rewind the adversary to the

end of the first round of the i-th iteration to receive another opening $(\delta'_i, m'_{0,i}, m'_{1,i})$.
Now, let $b \in \{0,1\}$ such that $m_{b,i} \neq m'_{b,i}$. By the security of the C&O protocol, there can be at most one such b. If there is no b we continue the first run. Otherwise, if there is such a b, we have learned the equivocal branch and start the second run.

2. We execute the second run according to the protcol with the difference that we now know the equivocal branch, i.e. b, from the very beginning, which will help us later to simulate correctly right from the start. Notice that by the security of the C&O protocol, a malicious receiver cannot change the equivocal branch in the second run. Obviously, he cannot change it during the same iteration since then he would be equivocal on both branches and contradict the security of the C&O protocol. He can also not change the equivocal branch of one of the later rounds $j > i$, since in the j-th commitment δ_j he cannot be committed to both $m_{b,i}$ and $m'_{b,i}$, so he needs to equivocally open δ_j as well. Thus, he needs to be committed on the other branch, i.e. branch $1 - b$.

- The values $m'_{k,i}$ (right after the rewinding) of each iteration of the first run for $k \in \{0,1\}$, and second run for $k = 1 - b$, are identical to $m_{k,i}$. Moreover, $m'_{k,i} \neq m_{k,i}$ holds only for the second run for branch $k = b$. Therefore, in the second hybrid, we can change the distribution of $r'_{k,i}$ to $r'_{k,i} = \rho^i_k - m_{k,i}$ for $k \in \{0,1\}$, and both runs except branch $k = b$ during the second run. The value ρ^i_k is obtained by running the simulator for the receiver of the underlying strongly uniform semi-honest OT protocol with choice bit 1 and input u_k. We can use the messages generated by this simulator on the sender's side as well.

  We will use the strong uniformity of the OT to argue that a malicious receiver cannot distinguish $r'_{k,i} = \rho^i_k - m_{k,i}$ from uniform. By the semi-honest security, the messages generated by the simulator are indistinguishable from the actual semi-honest OT. At the same time this simulator is independent of the sender's inputs s_0 and s_1. Note that in this hybrid, we only need to known s_b for the second run after having learned b.

In the last hybrid, a protocol transcript is independent of s_{1-b} but still yields a well distributed output for the malicious receiver, which directly yields a simulator in the ideal world.

To show (2), we first use the strong uniformity of the underlying OT protocol to sample $m_{b,i}$ uniformly at random at the beginning of the protocol. Notice that this implies that the receiver cannot recover the value s_b of the sender anymore. Further, we need the strong uniformity property here, since the receiver is interacting with a malicious sender who could influence the distribution of $m_{b,i}$ sent by the receiver. Once both messages, $m_{0,i}$ and $m_{1,i}$ for all iterations are known before the start of the protocol, we can challenge the choice bit indistinguishability of the C&O protocol. As a consequence, we can argue that the transcripts with $b = 0$ and $b = 1$ are computationally indistinguishable, which implies game-based security against a malicious sender.

3.4.3 Security Analysis

Simulatability Against a Malicious Receiver

We need to prove that for all non-uniform PPT malicious receivers R^*, there exists a PPT simulator Sim such that

$$\left\{\mathbf{REAL}_{\Pi,\mathsf{R}^*(z)}(\lambda, s_0, s_1, b)\right\}_{\lambda,s_0,s_1,b,z} \approx_c \left\{\mathbf{IDEAL}_{\mathcal{F}_{\mathrm{OT}},\mathsf{Sim}^{\mathsf{R}^*(z)}}(\lambda, s_0, s_1, b)\right\}_{\lambda,s_0,s_1,b,z}$$

where $\lambda \in \mathbb{N}$, $s_0, s_1 \in \{0,1\}^*$, $b \in \{0,1\}$, and $z \in \{0,1\}^*$.

To this end, we introduce several hybrid experiments naturally leading to the definition of an efficient simulator in the ideal world. Let $\mathbf{H}_0(\lambda)$ be the real world experiment with a malicious receiver R^*. (All experiments are further parameterized by the inputs s_0, s_1 for the sender, but we omit to explicitly write this for simplicity.)

First hybrid. Hybrid $\mathbf{H}_1(\lambda)$ proceeds as follows.

1. The sender picks $u_0, u_1 \leftarrow_\$ \mathcal{M}$ and lets $\tilde{\alpha}^0_{\mathsf{S},0} = \alpha^0_{\mathsf{S},0} = (s_0, u_0)$, $\tilde{\alpha}^0_{\mathsf{S},1} = \alpha^0_{\mathsf{S},1} = (s_1, u_1)$, and $b, b', b'' = \perp$.
2. R^* forwards $(\gamma_i)_{i\in[r']}$, to which the sender replies with $(\beta_1, r_{0,1}, r_{1,1}, \tilde{\sigma}^1_0, \tilde{\sigma}^1_1)$, where $(\tilde{\alpha}^1_{\mathsf{S},0}, \tilde{\sigma}^1_0) \leftarrow_\$ \mathsf{S}'(1^\lambda, \tilde{\alpha}^0_{\mathsf{S},0})$, $(\tilde{\alpha}^1_{\mathsf{S},1}, \tilde{\sigma}^1_1) \leftarrow_\$ \mathsf{S}'(1^\lambda, \tilde{\alpha}^0_{\mathsf{S},1})$.
3. Repeat the steps below, for each $i \in [r']$:
   (a) R^* sends a tuple $(\delta_i, m_{0,i}, m_{1,i})$. Let $T_i = (\gamma_i, \beta_i, (\delta_i, (m_{0,j})_{j\in[i]}, (m_{1,j})_{j\in[i]}))$. Hence:
      i. If $\mathsf{V}_1(T_i) = 0$, restart the experiment with fresh randomness for R^*. Since the protocol is correct with non-negligible probability, it will only take polynomial time to find a run where R^* never gets restarted within this step in any iteration.
      ii. Rewind R^* at the beginning of the current iteration, and send a freshly sampled tuple $(\beta'_i, r'_{0,i}, r'_{1,i})$ with the same distribution as before.
   (b) R^* replies with $(\delta'_i, m'_{0,i}, m'_{1,i})$. Let $T'_i = (\gamma_i, \beta'_i, (\delta'_i, (m'_{0,j})_{j\in[i]}, (m'_{1,j})_{j\in[i]}))$. Hence:
      i. If $\mathsf{V}_1(T'_i) = 0$, we restart R^* as in step 3(a)i (again this can be done in polynomial time). If $\mathsf{V}_1(T'_i) = 1$ and on both branches $(m'_{0,j})_{j\in[i]} \neq (m_{0,j})_{j\in[i]}$ *and* $(m'_{1,j})_{j\in[i]} \neq (m_{1,j})_{j\in[i]}$, the sender aborts.
      ii. Attempt to define b' as the binary value for which $(m'_{b',j})_{j\in[i]} \neq (m_{b',j})_{j\in[i]}$, but $(m'_{1-b',j})_{j\in[i]} = (m_{1-b',j})_{j\in[i]}$. If such value is found, halt and go directly to step 4 after setting $b \stackrel{\text{def}}{=} b'$.
   (c) The sender computes $(\tilde{\alpha}^{i+1}_{\mathsf{S},0}, \tilde{\sigma}^{i+1}_0) \leftarrow_\$ \mathsf{S}'(\tilde{\alpha}^i_{\mathsf{S},0}, m'_{0,i}+r'_{0,i})$, $(\tilde{\alpha}^{i+1}_{\mathsf{S},1}, \tilde{\sigma}^{i+1}_1) \leftarrow_\$ \mathsf{S}'(\tilde{\alpha}^i_{\mathsf{S},1}, m'_{1,i}+r'_{1,i})$, samples $(\beta_{i+1}, r_{0,i+1}, r_{1,i+1})$ as in the original protocol, and forwards $(\tilde{\sigma}^{i+1}_0, \tilde{\sigma}^{i+1}_1, \beta_{i+1}, r_{0,i+1}, r_{1,i+1})$ to R^*.
4. Rewind R^* to step 2, and re-start running the experiment from there with the following differences applied to each iteration $i \in [r']$:

(a) Denote by $(\beta''_i, r''_{0,i}, r''_{1,i})$ the new challenges sent to R^* in step 3(a)ii, and with $(\delta''_i, m''_{0,i}, m''_{1,i})$ the corresponding answer computed by R^* in step 3b. Also let $T''_i = (\gamma_i, \beta''_i, (\delta''_i, (m''_{0,j})_{j\in[i]}, (m''_{1,j})_{j\in[i]}))$.

(b) If either $\mathsf{V}_1(T''_i) = 0$, or $\mathsf{V}_1(T''_i) = 1$ and on both branches $(m''_{0,j})_{j\in[i]} \neq (m_{0,j})_{j\in[i]}$ *and* $(m''_{1,j})_{j\in[i]} \neq (m_{1,j})_{j\in[i]}$, the sender aborts.

(c) Attempt to define b'' as the binary value for which $(m_{b'',j})_{j\in[i]} \neq (m''_{b'',j})_{j\in[i]}$, but $(m_{1-b'',j})_{j\in[i]} = (m''_{1-b'',j})_{j\in[i]}$. If such value is found, but $b'' \neq b$ the sender aborts.

(d) The sender aborts if $b'' \neq \perp$, but $(m''_{b'',j})_{j\in[i]} = (m_{b'',j})_{j\in[i]}$.

(e) The sender computes $(\alpha^{i+1}_{\mathsf{S},0}, \sigma^{i+1}_0) \leftarrow\!\!\$\ \mathsf{S}'(\alpha^i_{\mathsf{S},0}, m'_{0,i}+r'_{0,i})$, $(\alpha^{i+1}_{\mathsf{S},1}, \sigma^{i+1}_1) \leftarrow\!\!\$\ \mathsf{S}'(\alpha^i_{\mathsf{S},1}, m'_{1,i}+r'_{1,i})$ samples $(\beta_{i+1}, r_{0,i+1}, r_{1,i+1})$ as in the original protocol, and forwards $(\sigma^{i+1}_0, \sigma^{i+1}_1, \beta_{i+1}, r_{0,i+1}, r_{1,i+1})$ to R^*.

5. Experiment output: The output of R^*.

Lemma 8. $\{\mathbf{H}_0(\lambda)\}_{\lambda\in\mathbb{N}} \approx_c \{\mathbf{H}_1(\lambda)\}_{\lambda\in\mathbb{N}}$.

Proof. First notice that all the restarts and rewindings of R^* do not change R^*'s output distribution, they only decrease the probability of a protocol abort at the cost of a polynomial increase in the running time.

For $i \in [r']$, consider the following events defined over the probability space of $\mathbf{H}_1(\lambda)$.

Event $W^i_{1,1}$: The event becomes true if the sender aborts during step 3(b)i, i.e. the values $(\delta_i, m_{0,i}, m_{1,i})$ and $(\delta'_i, m'_{0,i}, m'_{1,i})$ output by R^* are such that there is no $\hat{b} \in \{0,1\}$ for which $(m_{\hat{b},j})_{j\in[i]} = (m'_{\hat{b},j})_{j\in[i]}$, and furthermore both transcripts T_i and T'_i are valid transcripts for the underlying commit-and-prove protocol.

Event $W^i_{1,2}$: The event becomes true if the sender aborts during step 4b, i.e. the values $(\delta_i, m_{0,i}, m_{1,i})$ and $(\delta''_i, m''_{0,i}, m''_{1,i})$ output by R^* are such that there is no $\hat{b} \in \{0,1\}$ for which $(m_{\hat{b},j})_{j\in[i]} = (m''_{\hat{b},j})_{j\in[i]}$, and furthermore both transcripts T_i and T''_i are valid transcripts for the underlying commit-and-prove protocol.

Event $W^i_{1,3}$: The event becomes true if the sender aborts during step 4c, i.e., the non-committing branches b' and b'' are different for the two runs of the adversary (after rewinding).

Event $W^i_{1,4}$: The event becomes true if the sender aborts during step 4d, i.e., the value b'' was set in some previous iteration $k < i$, meaning that $(m_{b'',j})_{j\in[k]} \neq (m''_{b'',j})_{j\in[k]}$, but during the i-th iteration the same branch becomes again committing, meaning that $(m_{b'',j})_{j\in[i]} = (m''_{b'',j})_{j\in[i]}$.

Define $W^i_1 \stackrel{\text{def}}{=} W^i_{1,1} \vee W^i_{1,2} \vee W^i_{1,3} \vee W^i_{1,4}$. For all PPT distinguishers $\mathcal{D}$, by a union bound, we can write

$$\Delta_{\mathcal{D}}(\mathbf{H}_0(\lambda); \mathbf{H}_1(\lambda)) \leq \Pr[\exists i \in [r'] : \ W^i_1] \leq \sum_{i=1}^{r'} \sum_{j=1}^{4} \Pr[W^i_{1,j}],$$

and thus it suffices to prove that each of the events happens with negligible probability for all $i \in [r']$. We show this fact below, which concludes the proof of the lemma.

Claim 7. *For all PPT* R^**, and for all* $i \in [r']$*, we have that* $\Pr[W^i_{1,1}] \in \mathsf{negl}(\lambda)$.

Proof. The proof is down to the property of existence of a committing branch for the commit-and-prove protocol. By contradiction, assume that there is a pair $s_0, s_1 \in \{0,1\}^\lambda$, some $i \in [r']$, a non-uniform PPT adversary R^*, and an auxiliary input $z \in \{0,1\}^*$, such that $\mathsf{R}^*(z)$ provokes event $W^i_{1,1}$ in an execution of $\mathbf{H}_1(\lambda)$ with non-negligible probability. We build a non-uniform PPT adversary P^* that, given $i \in [r']$, attacks the security of $\Pi_{\mathsf{c\&o}}$ as follows:[6]

1. Run $\mathsf{R}^*(z)$, and after receiving $(\gamma_i)_{i\in[r']}$, forward γ_i to the challenger, thus obtaining a challenge β.
2. Emulate a run of experiment $\mathbf{H}_1(\lambda)$ with R^*, except that the value β_i is defined by embedding the value β received from the challenger.
3. Upon receiving $(\delta_i, m_{0,i}, m_{1,i})$ from R^*, check that $T_i = (\gamma_i, \beta_i, (\delta_i, (m_{0,j})_{j\in[i]}, (m_{1,j})_{j\in[i]}))$ is a valid transcript; if so, forward $(\delta_i, (m_{0,j})_{j\in[i]}, (m_{1,j})_{j\in[i]})$ to the challenger.
4. Upon receiving a fresh challenge β' for the commit-and-prove protocol from the challenger, rewind R^* as described in $\mathbf{H}_1(\lambda)$, except that the value β'_i is defined by embedding the value β' received from the challenger.
5. Upon receiving $(\gamma'_i, m'_{0,i}, m'_{1,i})$ from R^*, check that $T'_i = (\gamma_i, \beta'_i, (\delta'_i, (m'_{0,j})_{j\in[i]}, (m'_{1,j})_{j\in[i]}))$ is a valid transcript; if so, forward $(\gamma'_i, (m'_{0,j})_{j\in[i]}, (m'_{1,j})_{j\in[i]})$ to the challenger.
6. Complete the remaining steps of the protocol with R^*, as described in $\mathbf{H}_1(\lambda)$.

Notice that the above simulation is perfect; this is because the values (β, β') that the reduction embeds during the i-th iteration have exactly the same distribution as in an execution of experiment $\mathbf{H}_1(\lambda)$, whereas all other iterations are perfectly distributed as in $\mathbf{H}_1(\lambda)$. It follows that adversary R^* will provoke event $W^i_{1,1}$ with non-negligible probability, which means that both the transcripts T_i and T'_i are accepting, and moreover $(m_{0,j})_{j\in[i]} \neq (m'_{0,j})_{j\in[i]}$ and $(m_{1,j})_{j\in[i]} \neq (m'_{1,j})_{j\in[i]}$. Thus, P^* wins with non-negligible probability, which concludes the proof of the claim.

□

Claim 8. *For all PPT* R^**, and for all* $i \in [r']$*, we have that* $\Pr[W^i_{1,2}] \in \mathsf{negl}(\lambda)$.

Proof. The proof is similar to the one of the previous claim, and therefore omitted. The only difference is that the challenge β' is now embedded by the reduction in β''_i, and also the tuple $(\gamma''_i, (m''_{0,j})_{j\in[i]}, (m''_{1,j})_{j\in[i]})$ is sent to the challenger after the rewinding.

□

[6] We can also make the reduction uniform, at the cost of losing a polynomial factor in the computational distance between the two hybrids (which is needed to guess the index i for which event $W^i_{1,1}$ is provoked).

Claim 9. *For all PPT* R^**, and for all* $i \in [r']$*, we have that* $\Pr[W^i_{1,3}] \in \mathsf{negl}(\lambda)$.

Proof. Without loss of generality, assume that $b' = 0$ and $b'' = 1$. Notice that event $W^i_{1,3}$ means that both transcripts T_i and T''_i are accepting for the commit-and-prove protocol, and additionally $(m_{0,j})_{j\in[i]} \neq (m'_{0,j})_{j\in[i]}$, whereas $(m_{1,j})_{j\in[i]} \neq (m''_{0,j})_{j\in[i]}$. The latter contradicts the property of existence of a committing branch for the commit-and-prove protocol. The formal reduction is similar to the one given above, and is therefore omitted.

□

Claim 10. *For all PPT* R^**, and for all* $i \in [r']$*, we have that* $\Pr[W^i_{1,4}] \in \mathsf{negl}(\lambda)$.

Proof. Notice that event $W^i_{1,4}$ means that, for some iteration $k < i$, both transcripts $T_k = (\gamma_k, \beta_k, (\delta_k, (m_{0,j})_{j\in[k]}, (m_{1,j})_{j\in[k]}))$ and $T''_k = (\gamma_k, \beta''_k, (\delta''_k, (m''_{0,j})_{j\in[k]}, (m''_{1,j})_{j\in[k]}))$ are accepting for the commit-and-prove protocol, and additionally there exists a value $b \in \{0, 1\}$ such that branch b is non-committing, which means $(m_{1-b,j})_{j\in[k]} = (m''_{1-b,j})_{j\in[k]}$. However, during the i-th iteration, both transcripts T_i and T''_i are accepting for the commit-and-prove protocol, but branch b becomes committing again. The latter implies that there exist accepting transcripts T_k and T''_k for which both $(m_{1-b,j})_{j\in[i]} = (m''_{1-b,j})_{j\in[k]}$ and $(m_{b,j})_{j\in[k]} = (m''_{b,j})_{j\in[k]}$, which contradicts the property of existence of a committing branch for the commit-and-prove protocol. The formal reduction is similar to the one given above, and is therefore omitted.

□

□

Second hybrid. Hybrid $\mathbf{H}_2(\lambda)$ proceeds identically to $\mathbf{H}_1(\lambda)$, except for the following differences.

1. In step 1, the sender additionally sets $\tilde{\alpha}^0_{\mathsf{R}',0} = 1$.
2. The distribution of the values $r'_{0,i}$ computed during step 3(a)ii is changed by evaluating $(\tilde{\alpha}^i_{\mathsf{R}',0}, \tilde{\rho}^i_0) \leftarrow\!\!\$\ \mathsf{R}'(\tilde{\alpha}^{i-1}_{\mathsf{R}',0}, \tilde{\sigma}^i_0)$, and by letting $r'_{0,i} = \tilde{\rho}^i_0 - m_{0,i}$.

Notice that the latter change is applied only to the first run of R^* (i.e., up to the point where the value b' is set). This means that the distribution of the values $(r''_{0,i})_{i\in[r']}$ is not modified.

Lemma 9. $\{\mathbf{H}_1(\lambda)\}_{\lambda\in\mathbb{N}} \approx_c \{\mathbf{H}_2(\lambda)\}_{\lambda\in\mathbb{N}}$.

Proof. Let W^i_2 be the same event as W^i_1, but over the probability space of $\mathbf{H}_2(\lambda)$. For all PPT distinguishers $\mathcal{D}$, we can write:

$$\Delta_{\mathcal{D}}(\mathbf{H}_1(\lambda); \mathbf{H}_2(\lambda)) \leq \Delta_{\mathcal{D}}(\mathbf{H}_1(\lambda); \mathbf{H}_2(\lambda) | \forall i \in [r'] : \neg W^i_2) + \Pr[\exists i \in [r'] : W_2].$$

An argument similar to that used in the proof of Lemma 8 shows that $\Pr[\exists i \in [r'] : W^i_2]$ is negligible, hence it suffices to prove that $\Delta_{\mathcal{D}}(\mathbf{H}_1(\lambda); \mathbf{H}_2(\lambda) | \forall i \in [r'] : \neg W^i_2)$ is also negligible. Note that the only difference between the two experiments comes from the distribution of the messages $(r'_{0,j})_{j\in[i^*]}$, with $i^* \leq r'$ being the

index corresponding to the round (if any) where the bit b' is set during a run of the protocol: In experiment $\mathbf{H}_1(\lambda)$ these values are uniformly random, whereas in experiment $\mathbf{H}_2(\lambda)$ they are set to $\tilde{\rho}_0^j - m_{0,j}$, where $\tilde{\rho}_0^j$ is generated by a fresh run of the receiver for Π' with choice bit fixed to one.

By contradiction, assume that there exists a pair of values $s_0, s_1 \in \{0,1\}^\lambda$, and a non-uniform PPT distinguisher $\mathcal{D}$, such that $\mathcal{D}$ can tell apart $\mathbf{H}_1(\lambda)$ and $\mathbf{H}_2(\lambda)$ with non-negligible probability. We use $\mathcal{D}$ to construct a PPT distinguisher $\hat{\mathcal{D}}$ attacking the uniformity property (cf. property (b) in Definition 21) of protocol Π'. Actually, for this particular step of the proof we only need a weaker property where the distinguisher $\hat{\mathcal{D}}$ is honest but curious. The reduction works as follows:

1. Forward $\hat{b} = 1$, $\hat{s}_0 = s_0$, and uniform $\hat{s}_1 = u_0$ to the challenger.
2. Receive a challenge $((\hat{\rho}^i, \hat{\sigma}^i)_{i\in[r']}, \hat{\sigma}^{r'+1})$ from the challenger.
3. Run experiment $\mathbf{H}_2(\lambda)$ with $\mathcal{D}$, except that the changes below are applied to each iteration of the first run of the distinguisher:
   (a) During step 3(a)ii, the value $r'_{0,i}$ is set to be $r'_{0,i} = \hat{\rho}^i - m'_{0,i}$, whereas $r'_{1,i}$ is chosen uniformly at random in $\mathcal{M}$.
   (b) During step 3c, the value $\tilde{\sigma}_0^{i+1}$ is defined by embedding the value $\hat{\sigma}^{i+1}$ from the challenge.
4. Output the same as $\mathcal{D}$(output of R^*).

By inspection, depending on each pair $(\hat{\rho}^i, \hat{\sigma}^i)$ being distributed either as in a honest execution of protocol Π' between $\mathsf{S}'(s_0, u_0)$ and $\mathsf{R}'(1)$, or as in an interaction between $\mathsf{S}'(s_0, u_0)$ and using uniformly random group elements for the messages of the receiver, the distribution generated by the reduction is identical either to that of $\mathbf{H}_1(\lambda)$ or to that of $\mathbf{H}_2(\lambda)$. The latter in particular holds since we are conditioning on the event W_2^i not happening for all $i \in [r']$, which means that in $\mathbf{H}_2(\lambda)$ the values $\tilde{\sigma}_0^{i+1}$ are computed by running the honest sender $\mathsf{S}'(s_0, u_0)$ upon input $r'_{0,i} + m_{0,i} = (\tilde{\rho}_0^i - m_{0,i}) + m_{0,i} = \tilde{\rho}_0^i$.

It follows that $\hat{\mathcal{D}}$ makes a perfect simulation, and thus it retains the same distinguishing advantage as that of $\mathcal{D}$, which concludes the proof of the lemma. □

Third hybrid. Hybrid $\mathbf{H}_3(\lambda)$ proceeds identically to $\mathbf{H}_2(\lambda)$, except for the following differences.

1. In step 1, the sender additionally sets $\tilde{\alpha}^0_{\mathsf{Sim}',0} = (1, u_0)$ and defines $\tilde{\sigma}_0^1$ as $(\tilde{\alpha}^1_{\mathsf{Sim}',0}, \tilde{\sigma}_0^1) \leftarrow\!\!\$\ \mathsf{Sim}'_{\mathsf{R}'}(1^\lambda, \tilde{\alpha}^0_{\mathsf{Sim}',0})$.
2. The distribution of the values $\tilde{\rho}_0^i$ defined during step 3(a)ii, and of the values $\tilde{\sigma}_0^i$ defined during step 3c is changed by evaluating $(\tilde{\alpha}^{i+1}_{\mathsf{Sim}',0}, \tilde{\rho}_0^i, \tilde{\sigma}_0^{i+1}) \leftarrow\!\!\$\ \mathsf{Sim}'_{\mathsf{R}'}(\tilde{\alpha}^1_{\mathsf{Sim}',0})$.

Notice that the latter change is applied only to the first run of R^* (i.e., up to the point where the value b' is set). This means that the distribution of the values $(\rho_0^i, \sigma_0^i)_{i\in[r']}$ is not modified.

Lemma 10. $\mathbf{H}_2(\lambda) \approx_c \mathbf{H}_3(\lambda)$.

Proof. Let W_3^i be the same event as W_2^i, but over the probability space of $\mathbf{H}_3(\lambda)$. For all PPT distinguishers $\mathcal{D}$, we can write:

$$\Delta_{\mathcal{D}}(\mathbf{H}_2(\lambda); \mathbf{H}_3(\lambda)) \leq \Delta_{\mathcal{D}}(\mathbf{H}_2(\lambda); \mathbf{H}_3(\lambda) | \forall i \in [r'] : \neg\, W_3^i) + \Pr[\exists i \in [r'] : W_3].$$

An argument similar to that used in the proof of Lemma 8 shows that $\Pr[\exists i \in [r'] : W_3^i]$ is negligible, hence it suffices to prove that $\Delta_{\mathcal{D}}(\mathbf{H}_2(\lambda); \mathbf{H}_3(\lambda) | \forall i \in [r'] : \neg\, W_3^i)$ is also negligible. Note that the only difference between the two experiments comes from the distribution of the values $\tilde{\sigma}_0^{r'+1}, (\tilde{\rho}_0^j, \tilde{\sigma}_0^j)_{j \in [i^*]}$, with $i^* \leq r'$ being the index corresponding to the round (if any) where the bit b' is set during a run of the protocol: In experiment $\mathbf{H}_2(\lambda)$ these values are generated through a honest execution of protocol Π' between receiver R' with choice bit fixed to 1 and sender S' with inputs (s_0, u_0), whereas in experiment $\mathbf{H}_3(\lambda)$ they are generated by running the simulator $\mathsf{Sim}'_{\mathsf{R}'}$.

The proof is down to the security of the underlying $(2r'+1)$-round OT protocol $\Pi' = (\mathsf{S}', \mathsf{R}')$ w.r.t. semi-honest receivers (cf. property (a) of Definition 21). By contradiction, assume that there exists a pair of inputs $s_0, s_1 \in \{0,1\}^\lambda$, and a non-uniform PPT distinguisher $\mathcal{D}$, such that $\mathcal{D}$ can tell apart $\mathbf{H}_2(\lambda)$ and $\mathbf{H}_3(\lambda)$ with non-negligible probability. We construct a PPT distinguisher $\hat{\mathcal{D}}$ that given (s_0, s_1) attacks semi-honest security of Π' as follows:

1. Forward $\hat{b} = 1$, $\hat{s}_0 = s_0$, and $\hat{s}_1 = s_1$ to the challenger.

2. Receive a challenge $(\hat{\rho}^i, \hat{\sigma}^i)_{i \in [r']}, \hat{\sigma}^{r'+1}$ from the challenger.

3. Run experiment $\mathbf{H}_3(\lambda)$ with $\mathcal{D}$, except that the changes below are applied to each iteration of the first run of the distinguisher:

   (a) During step 3(a)ii, the value $r'_{0,i}$ is set to be $r'_{0,i} = \hat{\rho}^i - m'_{0,i}$, whereas $r'_{1,i}$ is chosen uniformly at random in $\mathcal{M}$.

   (b) During step 3c, the value $\tilde{\sigma}_0^{i+1}$ is defined by embedding the value $\hat{\sigma}^{i+1}$ from the challenge.

4. Output the same as $\mathcal{D}$(output of R^*).

By inspection, depending on each pair $(\hat{\rho}^i, \hat{\sigma}^i)$ being distributed either as in a honest execution of protocol Π' between $\mathsf{S}'(s_0, u_0)$ and $\mathsf{R}'(1)$, or as computed by the simulator $\mathsf{Sim}'_{\mathsf{R}'}$ with inputs $(1^\lambda, 1, u_0)$, the distribution generated by the reduction is identical either to that of $\mathbf{H}_2(\lambda)$ or to that of $\mathbf{H}_3(\lambda)$. The latter in particular holds since we are conditioning on the event W_3^i not happening for all $i \in [r']$, which means that in $\mathbf{H}_2(\lambda)$ the values $\tilde{\sigma}_0^i$ are computed by running the honest sender $\mathsf{S}'(s_0, u_0)$ upon input $r'_{0,i} + m_{0,i} = (\tilde{\rho}_0^i - m_{0,i}) + m_{0,i} = \tilde{\rho}_0^i$.

It follows that $\hat{\mathcal{D}}$ makes a perfect simulation, and thus it retains the same distinguishing advantage as that of $\mathcal{D}$, which concludes the proof of the lemma. □

Fourth hybrid. Hybrid $\mathbf{H}_4(\lambda)$ proceeds identically to $\mathbf{H}_3(\lambda)$, except for the following differences.

1. In step 1, the sender additionally sets $\tilde{\alpha}^0_{\mathsf{R}',1} = 1$.
2. The distribution of the values $r'_{1,i}$ computed during step 3(a)ii is changed by evaluating $(\tilde{\alpha}^i_{\mathsf{R}',1}, \tilde{\rho}^i_1) \leftarrow\!\!\$\ \mathsf{R}'(\tilde{\alpha}^{i-1}_{\mathsf{R}',1}, \tilde{\sigma}^i_1)$, and by letting $r'_{1,i} = \tilde{\rho}^i_1 - m_{1,i}$.

Notice that the latter change is applied only to the first run of R^* (i.e., up to the point where the value b' is set). This means that the distribution of the values $(r''_{1,i})_{i\in[r']}$ is not modified. The proof of the lemma below is identical to the proof of Lemma 9, and is therefore omitted.

Lemma 11. $\{\mathbf{H}_3(\lambda)\}_{\lambda\in\mathbb{N}} \approx_c \{\mathbf{H}_4(\lambda)\}_{\lambda\in\mathbb{N}}$.

Fifth hybrid. Hybrid $\mathbf{H}_5(\lambda)$ proceeds identically to $\mathbf{H}_4(\lambda)$, except for the following differences.

1. In step 1, the sender additionally sets $\tilde{\alpha}^0_{\mathsf{Sim}',1} = (1, u_1)$ and defines $\tilde{\sigma}^1_1$ as $(\tilde{\alpha}^1_{\mathsf{Sim}',1}, \tilde{\sigma}^1_1) \leftarrow\!\!\$\ \mathsf{Sim}'_{\mathsf{R}'}(1^\lambda, \tilde{\alpha}^0_{\mathsf{Sim}',1})$
2. The distribution of the values $\tilde{\rho}^i_1$ defined during step 3(a)ii, and of the values $\tilde{\sigma}^i_1$ defined during step 3c is changed by evaluating $(\tilde{\alpha}^{i+1}_{\mathsf{Sim}',1}, \tilde{\rho}^i_1, \tilde{\sigma}^{i+1}_1) \leftarrow\!\!\$\ \mathsf{Sim}'_{\mathsf{R}'}(\tilde{\alpha}^i_{\mathsf{Sim}',1})$.

Notice that the latter change is applied only to the first run of R^* (i.e., up to the point where the value b' is set). This means that the distribution of the values $(\rho^i_1, \sigma^i_1)_{i\in[r']}$ is not modified. The proof of the lemma below is identical to that of Lemma 10, and is therefore omitted.

Lemma 12. $\{\mathbf{H}_4(\lambda)\}_{\lambda\in\mathbb{N}} \approx_c \{\mathbf{H}_5(\lambda)\}_{\lambda\in\mathbb{N}}$.

Sixth hybrid. Hybrid $\mathbf{H}_6(\lambda)$ proceeds identically to $\mathbf{H}_5(\lambda)$, except for the following differences.

1. In step 4, the sender additionally sets $\alpha^0_{\mathsf{R}',1-b} = 1$. If $b = \perp$, set both $\alpha^0_{\mathsf{R}',0} = \alpha^0_{\mathsf{R}',1} = 1$.
2. The distribution of the values $r''_{1-b,i}$ computed during step 4a is changed by evaluating $(\alpha^i_{\mathsf{R}',1-b}, \rho^i_{1-b}) \leftarrow\!\!\$\ \mathsf{R}'(\alpha^{i-1}_{\mathsf{R}',1-b}, \sigma^i_{1-b})$, and by letting $r'_{1-b,i} = \tilde{\rho}^i_{1-b} - m_{1-b,i}$. If $b = \perp$, such a change is applied on both branches.

The proof of the lemma below is identical to the proof of Lemma 9, and is therefore omitted.

Lemma 13. $\{\mathbf{H}_5(\lambda)\}_{\lambda\in\mathbb{N}} \approx_c \{\mathbf{H}_6(\lambda)\}_{\lambda\in\mathbb{N}}$.

Seventh hybrid. Hybrid $\mathbf{H}_7(\lambda)$ proceeds identically to $\mathbf{H}_6(\lambda)$, except for the following differences.

1. In step 4, the sender additionally sets $\alpha^0_{\mathsf{Sim}',1-b} = (1, u_{1-b})$ and defines σ^1_{1-b} as $(\alpha^1_{\mathsf{Sim}',1-b}, \sigma^1_{1-b}) \leftarrow\!\!\$\ \mathsf{Sim}'_{\mathsf{R}'}(1^\lambda, \alpha^0_{\mathsf{Sim}',1-b})$. If $b = \perp$, set both $\alpha^0_{\mathsf{Sim}',0} = (1, u_0)$, $\alpha^0_{\mathsf{Sim}',1} = (1, u_1)$ and generate σ^1_0, σ^1_1 as $(\alpha^1_{\mathsf{Sim}',0}, \sigma^1_0) \leftarrow\!\!\$\ \mathsf{Sim}'_{\mathsf{R}'}(1^\lambda, \alpha^0_{\mathsf{Sim}',0})$, $(\alpha^1_{\mathsf{Sim}',1}, \sigma^1_1) \leftarrow\!\!\$\ \mathsf{Sim}'_{\mathsf{R}'}(1^\lambda, \alpha^0_{\mathsf{Sim}',1})$

2. The distribution of the values ρ^i_{1-b} defined during step 4a, and of the values σ^{i+1}_{1-b} defined during step 4e is changed by evaluating $(\alpha^{i+1}_{\mathsf{Sim}',1-b}, \rho^i_{1-b}, \sigma^{i+1}_{1-b}) \leftarrow\!\!\$\ \mathsf{Sim}'_{\mathsf{R}'}(\alpha^i_{\mathsf{Sim}',1-b})$. If $b = \perp$, such changes are applied on both branches.

The proof of the lemma below is identical to that of Lemma 10, and is therefore omitted.

Lemma 14. $\{\mathbf{H}_6(\lambda)\}_{\lambda\in\mathbb{N}} \approx_c \{\mathbf{H}_7(\lambda)\}_{\lambda\in\mathbb{N}}$.

Simulator. We are now ready to describe the simulator Sim, interacting with the ideal functionality $\mathcal{F}_{\mathsf{OT}}$. The simulator works as follows:

1. Pick $u_0, u_1 \leftarrow\!\!\$\ \mathcal{M}$, and let $\tilde{\alpha}^0_{\mathsf{Sim}',0} = (1, u_0)$, $\tilde{\alpha}^0_{\mathsf{Sim}',1} = (1, u_1)$, $(\tilde{\alpha}^1_{\mathsf{Sim}',0}, \tilde{\sigma}^1_0) \leftarrow\!\!\$\ \mathsf{Sim}'_{\mathsf{R}'}(1^\lambda, \tilde{\alpha}^0_{\mathsf{Sim}',0})$, $(\tilde{\alpha}^1_{\mathsf{Sim}',1}, \tilde{\sigma}^1_1) \leftarrow\!\!\$\ \mathsf{Sim}'_{\mathsf{R}'}(1^\lambda, \tilde{\alpha}^0_{\mathsf{Sim}',1})$, and $b, b', b'' = \perp$.

2. Upon receiving $(\gamma_i)_{i\in[r']}$ from R^*, sample $\beta_1 \leftarrow\!\!\$\ \mathsf{V}_0(1^\lambda)$, $r_{0,1}, r_{1,1} \leftarrow\!\!\$\ \mathcal{M}$, and send $(\beta_1, r_{0,1}, r_{1,1}, \tilde{\sigma}^1_0, \tilde{\sigma}^1_1)$ to R^*.

3. Repeat the steps below, for each $i \in [r']$:

   (a) Upon receiving a tuple $(\delta_i, m_{0,i}, m_{1,i})$ from R^*, let $T_i = (\gamma_i, \beta_i, (\delta_i, (m_{0,j})_{j\in[i]}, (m_{1,j})_{j\in[i]}))$. Hence:
      i. If $\mathsf{V}_1(T_i) = 0$, restart R^*.
      ii. Rewind R^* at the beginning of the current iteration, and send a tuple $(\beta'_i, r'_{0,i}, r'_{1,i})$ where $\beta'_i \leftarrow\!\!\$\ \mathsf{V}_0(1^\lambda)$, $r'_{0,i} = \tilde{\rho}^i_0 - m_{0,i}$ and $r'_{1,i} = \tilde{\rho}^i_1 - m_{1,i}$, for $(\tilde{\alpha}^{i+1}_{\mathsf{Sim}',0}, \tilde{\rho}^i_0, \tilde{\sigma}^{i+1}_0) \leftarrow\!\!\$\ \mathsf{Sim}'_{\mathsf{R}'}(\tilde{\alpha}^i_{\mathsf{Sim}',0})$ and for $(\tilde{\alpha}^{i+1}_{\mathsf{Sim}',1}, \tilde{\rho}^i_1, \tilde{\sigma}^{i+1}_1) \leftarrow\!\!\$\ \mathsf{Sim}'_{\mathsf{R}'}(\tilde{\alpha}^i_{\mathsf{Sim}',1})$.

   (b) Upon receiving a tuple $(\delta'_i, m'_{0,i}, m'_{1,i})$ from R^*, let $T'_i = (\gamma_i, \beta'_i, (\delta'_i, (m'_{0,j})_{j\in[i]}, (m'_{1,j})_{j\in[i]}))$. Hence:
      i. If $\mathsf{V}_1(T'_i) = 0$, restart R^*. If $\mathsf{V}_1(T'_i) = 1$ and on both branches $(m'_{0,j})_{j\in[i]} \neq (m_{0,j})_{j\in[i]}$ *and* $(m'_{1,j})_{j\in[i]} \neq (m_{1,j})_{j\in[i]}$, abort.
      ii. Attempt to define b' as the binary value for which $(m'_{b',j})_{j\in[i]} \neq (m_{b',j})_{j\in[i]}$, but $(m'_{1-b',j})_{j\in[i]} = (m_{1-b',j})_{j\in[i]}$. If such value is found, halt and go directly to step 4 after setting $b \stackrel{\text{def}}{=} b'$.

   (c) Forward $(\tilde{\sigma}^{i+1}_0, \tilde{\sigma}^{i+1}_1, \beta_{i+1}, r_{0,i+1}, r_{1,i+1})$ to R^*, where $\beta_{i+1} \leftarrow\!\!\$\ \mathsf{V}_0(1^\lambda)$, and $r_{0,i+1}, r_{1,i+1} \leftarrow\!\!\$\ \mathcal{M}$.

4. Query $\mathcal{F}_{\mathsf{OT}}$ upon input b, obtaining a value $s_b \in \{0,1\}^\lambda$.[7] Let $\alpha^0_{\mathsf{Sim}',1-b} = (1, u_{1-b})$, $\alpha^0_{\mathsf{S}',b} = (s_b, u_b)$ and define σ^1_0, σ^1_1 as $(\alpha^1_{\mathsf{Sim}',1-b}, \sigma^1_{1-b}) \leftarrow\!\!\$\ \mathsf{Sim}'_{\mathsf{R}'}(1^\lambda, \alpha^0_{\mathsf{Sim}',1-b})$,$(\alpha^1_{\mathsf{S}',b}, \sigma^1_b) \leftarrow\!\!\$\ \mathsf{S}'(1^\lambda, \alpha^0_{\mathsf{S}',b})$. Rewind R^* to step 2, sample $\beta_1 \leftarrow\!\!\$\ \mathsf{V}_0(1^\lambda)$, $r_{0,1}, r_{1,1} \leftarrow\!\!\$\ \mathcal{M}$, and send $(\beta_1, r_{0,1}, r_{1,1}, \sigma^1_0, \sigma^1_1)$ to R^*.

5. Repeat the steps below, for each $i \in [r']$:

   (a) Upon receiving a tuple $(\delta_i, m_{0,i}, m_{1,i})$ from R^*, let $T_i = (\gamma_i, \beta_i, (\delta_i, (m_{0,j})_{j\in[i]}, (m_{1,j})_{j\in[i]}))$. Hence:
      i. If $\mathsf{V}_1(T_i) = 0$, restart R^*.
      ii. Rewind R^* at the beginning of the current iteration, and send a tuple $(\beta''_i, r''_{0,i}, r''_{1,i})$ where $\beta''_i \leftarrow\!\!\$\ \mathsf{V}_0(1^\lambda)$, $r''_{1-b,i} = \rho^i_{1-b} - m_{1-b,i}$ and $r''_{b,i} \leftarrow\!\!\$\ \mathcal{M}$, for $(\alpha^{i+1}_{\mathsf{Sim}',1-b}, \rho^i_{1-b}, \sigma^{i+1}_{1-b}) \leftarrow\!\!\$\ \mathsf{Sim}'_{\mathsf{R}'}(\alpha^i_{\mathsf{Sim}',1-b})$.

   (b) Upon receiving a tuple $(\delta''_i, m''_{0,i}, m''_{1,i})$ from R^*, let $T''_i = (\gamma_i, \beta''_i, (\delta''_i, (m''_{0,j})_{j\in[i]}, (m''_{1,j})_{j\in[i]}))$. Hence:
      i. If either $\mathsf{V}_1(T''_i) = 0$, or $\mathsf{V}_1(T''_i) = 1$ and on both branches $(m''_{0,j})_{j\in[i]} \neq (m_{0,j})_{j\in[i]}$ *and* $(m''_{1,j})_{j\in[i]} \neq (m_{1,j})_{j\in[i]}$, abort.
      ii. Attempt to define b'' as the binary value for which $(m''_{b'',j})_{j\in[i]} \neq (m_{b'',j})_{j\in[i]}$, but $(m''_{1-b'',j})_{j\in[i]} = (m_{1-b'',j})_{j\in[i]}$. If such value is found, but $b'' \neq b$, abort.
      iii. If $b'' \neq \perp$, but $(m''_{b'',j})_{j\in[i]} = (m_{b'',j})_{j\in[i]}$.

   (c) Forward $(\sigma^{i+1}_0, \sigma^{i+1}_1, \beta_{i+1}, r_{0,i+1}, r_{1,i+1})$ to R^*, where $\beta_{i+1} \leftarrow\!\!\$\ \mathsf{V}_0(1^\lambda)$, and $r_{0,i+1}, r_{1,i+1} \leftarrow\!\!\$\ \mathcal{M}$, and further $(\alpha^{i+1}_{\mathsf{S}',b}, \sigma^{i+1}_b) \leftarrow\!\!\$\ \mathsf{S}'(\alpha^i_{\mathsf{S}',b}, m''_{b,i} + r''_{b,i})$, while σ^{i+1}_{1-b} was obtained in step 5(a)ii above.

6. Return the output of R^*.

The distribution of $\mathbf{H}_7(\lambda)$ is identical to that of the ideal experiment $\mathbf{IDEAL}_{\mathcal{F}_{\mathsf{OT}},\mathsf{Sim}^{\mathsf{R}^*(z)}}(\lambda, s_0, s_1, b)$ for the above defined simulator. This concludes the proof of property (a) in the definition of receiver-sided simulatability.

Indistinguishability Against a Malicious Sender

We need to show that given the view of a malicious sender it is hard to distinguish whether he has interacted with a receiver using choice bit $b = 0$ or $b = 1$. More precisely, for every non-uniform PPT malicious sender S^* it holds that

$$\left\{ view^{\mathsf{R}}_{\Pi,\mathsf{S}^*(z)}(\lambda, s_0, s_1, 0) \right\}_{\lambda,s_0,s_1,z} \approx_c \left\{ view^{\mathsf{R}}_{\Pi,\mathsf{S}^*(z)}(\lambda, s_0, s_1, 1) \right\}_{\lambda,s_0,s_1,z}$$

where $\lambda \in \mathbb{N}$, $s_0, s_1 \in \{0,1\}^\lambda$, and $z \in \{0,1\}^*$, and where $view^{\mathsf{R}}_{\Pi,\mathsf{S}^*(z)}(\lambda, s_0, s_1, b)$ is the distribution of the view of S^* (with input s_0, s_1 and auxiliary input z) at the end of a real execution of Π with the honest receiver R (with input b).

[7] In case $b = \perp$, it is not necessary to query the ideal functionality. In fact, the latter means that in all iterations of the first run with the adversary, both branches for the C&O protocol are committing, and so they will be in the second run. Thus, the simulator can simply use the simulation strategy for the committing branch, which is independent of the sender's input, on both branches.

Let $\mathbf{H}_0(\lambda, b) \equiv view^{\mathsf{R}}_{\Pi,\mathsf{S}^*(z)}(\lambda, s_0, s_1, b)$. To show the above, we define the following hybrid $\mathbf{H}(\lambda, b)$.

1. The receiver picks for all $i \in [r']$ $m_{1-b,i} \leftarrow\!\!\$\ \mathcal{M}$. Then, he computes $(\gamma_i, \alpha_i) \leftarrow\!\!\$\ \mathsf{P}_0((m_{1-b,j})_{j\in[i]})$ and sends $(\gamma_i)_{i\in[r']}$.

2. Repeat for each $i \in [r']$: Upon receiving $(\sigma_0^i, \sigma_1^i, \beta_i, r_{0,i}, r_{1,i})$ the receiver picks $\rho_b^i \leftarrow\!\!\$\ \mathcal{M}$, sets $m_{b,i} = \rho_b^i - r_{b,i}$, computes $\delta_i \leftarrow\!\!\$\ \mathsf{P}_1(\alpha_i, \beta_i, \gamma_i, (m_{b,j})_{j\in[i]})$, and sends $(\delta_i, m_{0,i}, m_{1,i})$.

3. The experiment outputs the view of malicious sender S^*.

Notice that the output distribution of $\mathbf{H}_1(\lambda, b)$ does not change when we sample $m_{b,i} \leftarrow\!\!\$\ \mathcal{M}$ during the first step and define $\rho_b^i = m_{b,i} + r_{b,i}$ in the second step.

Lemma 15. *For all $b \in \{0,1\}$, we have that $\{\mathbf{H}_0(\lambda, b)\}_{\lambda\in\mathbb{N}} \approx_c \{\mathbf{H}_1(\lambda, b)\}_{\lambda\in\mathbb{N}}$.*

Proof. By contradiction, assume that there exists a PPT distinguisher $\mathcal{D}$, a bit $b \in \{0,1\}$, and a polynomial $p(\lambda) \in \mathsf{poly}(\lambda)$ such that for infinitely many values of $\lambda \in \mathbb{N}$:

$$|\Pr[\mathcal{D}(\mathbf{H}_0(\lambda, b) = 1] - \Pr[\mathcal{D}(\mathbf{H}_1(\lambda, b)) = 1]| \geq 1/p(\lambda).$$

We will construct a PPT distinguisher $\mathcal{D}'$ such that

$$\left|\mathbb{P}\left[\mathcal{D}'(\alpha_{\mathcal{D}}^{r'}, (\rho^i, \sigma^i)_{i\in[r']}) = 1 : \begin{array}{c}\forall i \in [r'], (\alpha_{\mathsf{R}}^i, \rho^i) \leftarrow\!\!\$\ \mathsf{R}(\alpha_{\mathsf{R}}^{i-1}, \sigma^i) \\ \wedge\ (\alpha_{\mathcal{D}}^i, \sigma^i) \leftarrow\!\!\$\ \mathcal{D}(\alpha_{\mathcal{D}}^{i-1}, \rho^i)\end{array}\right]\right.$$
$$\left. - \mathbb{P}\left[\mathcal{D}'(\alpha_{\mathcal{D}}^{r'}, (\rho^i, \sigma^i)_{i\in[r']}) = 1 : \begin{array}{c}\forall i \in [r'], \rho^i \leftarrow\!\!\$\ \mathcal{M} \\ \wedge\ (\alpha_{\mathcal{D}}^i, \sigma^i) \leftarrow\!\!\$\ \mathcal{D}(\alpha_{\mathcal{D}}^{i-1}, \rho^i)\end{array}\right]\right| \geq 1/p(\lambda).$$

We define $\mathcal{D}'$ as follows. Distinguisher $\mathcal{D}'$ invokes $\mathcal{D}$ and acts as in the actual protocol, except for the way he samples the values ρ^i which are obtained from the challenger after forwarding each of the values σ^i sent by the malicious sender. Finally, $\mathcal{D}'$ outputs the same as $\mathcal{D}$. It is easy to see that when ρ^i is generated by R, then D' simulates $\mathbf{H}_0(\lambda, b)$, and when ρ^i is picked uniformly at random he generates $\mathbf{H}_1(\lambda, b)$. Hence, $\mathcal{D}'$ has the same distinguishing advantage as that of $\mathcal{D}$. This finishes the proof. □

In $\mathbf{H}_1(\lambda, b)$ we can sample both messages $m_{1,i}$, $m_{0,i}$ for all $i \in [n]$ of the C&O protocol in the very beginning. Therefore we can use the choice bit indistinguishability to argue that the receivers choice bit is hidden. This fact is formalized in the lemma below.

Lemma 16. $\{\mathbf{H}_1(\lambda, 0)\}_{\lambda\in\mathbb{N}} \approx_c \{\mathbf{H}_1(\lambda, 1)\}_{\lambda\in\mathbb{N}}$.

Proof. The proof is by a standard hybrid argument. For each $j \in [0, r']$, let $\mathbf{H}_{1,j}(\lambda, b)$ be the hybrid experiment that is identical to $\mathbf{H}_1(\lambda, b)$ except that after sampling $(m_{0,i}, m_{1,i})_{i\in[r']}$ uniformly from $\mathcal{M}$, the receiver defines all commitments $(\gamma_i)_{i\leq j}$ by running the prover P of the underlying C&O protocol upon input $(m_{1-b,i})_{i\leq j}$, whereas the commitments $(\gamma_i)_{i>j}$ are defined by running the prover P upon input $(m_{b,i})_{i\leq j}$. Observe that $\mathbf{H}_{1,0}(\lambda, b) \equiv \mathbf{H}_1(\lambda, 1-b)$ and $\mathbf{H}_{1,r'}(\lambda, b) \equiv$

$\mathbf{H}_1(\lambda, b)$; hence, it suffices to show that $\mathbf{H}_{1,j}(\lambda, b) \approx_c \mathbf{H}_{1,j+1}(\lambda, b)$ holds for all $b \in \{0,1\}$ and for all $j \in [0, r']$.

By contradiction, assume that there exists a PPT distinguisher $\mathcal{D}$, a value $b \in \{0,1\}$, an index $j \in [0, r']$, and a polynomial $p(\lambda) \in \mathsf{poly}(\lambda)$, such that for infinitely many values of $\lambda \in \mathbb{N}$:

$$|\mathbb{P}\left[\mathcal{D}(\mathbf{H}_{1,j}(\lambda, b)) = 1\right] - \mathbb{P}\left[\mathcal{D}(\mathbf{H}_{1,j+1}(\lambda, b)) = 1\right]| \geq 1/p(\lambda).$$

We will construct a PPT distinguisher $\mathcal{D}'$ and a PPT malicious verifier V^* such that

$$\begin{aligned}\Big|\mathbb{P}\Big[\mathcal{D}'(\langle \mathsf{P}((m_{b,i})_{i\leq j+1}, (m_{1-b,i})_{i\leq j+1}, 0), \mathsf{V}^*(1^\lambda)\rangle) = 1\Big] \\ - \mathbb{P}\Big[\mathcal{D}'(\langle \mathsf{P}((m_{b,i})_{i\leq j+1}, (m_{1-b,i})_{i\leq j+1}, 1), \mathsf{V}^*(1^\lambda)\rangle) = 1\Big]\Big| \geq 1/p(\lambda),\end{aligned}$$

where for all $i \in [r']$, the values $m_{0,i}, m_{1,i}$ are uniformly sampled by $\mathcal{D}'$ from $\mathcal{M}$. Verifier V^* invokes $\mathcal{D}$ and emulates faithfully a run of $\mathbf{H}_{1,j}(\lambda, b)$ except that it embeds the commitment received from the challenger in the value γ_{j+1} which is part of the first message sent to $\mathcal{D}$, and similarly, after receiving $(\sigma_0^{j+1}, \sigma_1^{j+1}, \beta_{j+1}, r_{0,j+1}, r_{1,j+1})$ from $\mathcal{D}$, it forwards β_{j+1} to the challenger, obtaining a value δ_{j+1} that is used together with $(m_{b,i})_{i\leq j+1}$ and $(m_{1-b,i})_{i\leq j+1}$ in order to terminate the execution of the experiment. In the end, $\mathcal{D}'$ outputs the output of $\mathcal{D}$.

Clearly, when the challenger uses committing branch zero, the reduction perfectly simulates $\mathbf{H}_{1,j}(\lambda, b)$, and when the challenger uses committing branch 1, the reduction perfectly simulates $\mathbf{H}_{1,j+1}(\lambda, b)$. Since $r' \in \mathsf{poly}(\lambda)$, the statement follows. □

Chapter 4

Fair and Publicly Verifiable MPC on Forkable Blockchains

4.1 Threat Model

Our n-party parallel coin-tossing (PCT) protocol in the presence of hasty players is secure w.r.t. dishonest majority, meaning that it can tolerate up to $n-1$ corrupted players. We assume that the blockchain adversary is computationally bounded, and when there is a fork in the blockchain, we pragmatically assume that the adversary has negligible impact on deciding which branch will be confirmed. Our generic compiler can be secure in the presence of hasty players w.r.t. dishonest majority when the protocol to be compiled is secure w.r.t. dishonest majority. We point out that if one would like to consider a very strong adversary with even 49% of the computational power of the network, then clearly our assumption does not hold. However, we stress that with such an adversary even the 6-block rule in Bitcoin does not make much sense. To guarantee that a delicate transaction (i.e., the coinbase transaction) is confirmed with a strong enough adversary, up to 144 blocks are necessary in Bitcoin [BW], meaning 1 day to communicate even a single protocol message. Therefore if one would like to consider such strong adversaries even a protocol requiring one confirmation might be impractical.

We will also consider adversaries mounting DoS attacks through aborts. In our context the adversary can mount this attack by causing an abort to the protocol by e.g. not playing anymore and, in our generic compiler, also by sending different messages on different branches, making honest players abort the execution. Such adversaries have the only purpose of penalizing honest players that will therefore waste time and transaction fees and perhaps restarting the protocol from scratch.

4.2 Running MPC on Forking Blockchains

In this section, we formalize different ways how to run an MPC protocol with the aid of a blockchain. In Section 4.2.1 we specify what it means to run an MPC protocol on the blockchain both in the presence of hasty and non-hasty players. The security definition appears in Section 4.2.2.

4.2.1 Blockchain-Aided MPC

Next, we define what it means to run an n-party protocol π for securely computing some function $f : (\{0,1\}^*)^n \to (\{0,1\}^*)^n$ over a blockchain protocol Γ.

Intuitively, running π on Γ simply means that the players write the protocol's messages on the blockchain instead of using point-to-point connections. However, since the blockchain may fork, the protocol's participants have to choose how to manage possibly unconfirmed blocks that are part of the current chain. Looking ahead, this choice will have impact both on the efficiency and on the security of the protocol execution. In particular, we distinguish between *hasty* and *non-hasty* players as formalized below.

Non-hasty execution. Roughly speaking, a player is said to be *non-hasty* if it always decides its next message by looking at the transcript of the protocol that is obtained by pruning the last k blocks of the blockchain, where k is the parameter for the consistency property of the underlying blockchain.

Definition 22 (Non-hasty player)**.** Let $\Gamma = (\mathsf{UpdateState}, \mathsf{GetRecords}, \mathsf{Broadcast})$ be a blockchain protocol with k-consistency. A player P_i is said to be *non-hasty* if it behaves as follows:

- Initialize $\tau_i^{(0)} := \varepsilon$, $\alpha_i := \varepsilon$ and $r_i := 0$.
- Run the following loop:
  - Update the state α_i by running $\mathsf{UpdateState}(1^\lambda)$, and retrieve $\mathcal{B}_i \leftarrow\!\!\$\ \mathsf{GetRecords}(\alpha_i)$ until the partial transcript $\tau^{(r_i)}$ is contained in $\mathcal{B}_i^{\lceil k}$.
  - If the protocol is over (i.e., the transcript $\tau^{(r_i)}$ is sufficient for determining the output), output the value y_i as a function of $\tau_i^{(r_i)}$ and terminate.
  - Else, compute the next protocol message $m_i^{(r_i+1)}$, invoke $\mathsf{Broadcast}(m_i^{(r_i+1)})$, and set $r_i := r_i + 1$.

Hasty execution. On the other hand, a player is *hasty* if it decides and broadcasts its next message by looking at the latest version of the blockchain (i.e., without pruning blocks). Since the consistency property does not hold for the last k blocks, hasty players may retrieve different protocol's transcripts as the protocol proceeds. In particular, it may happen that at a given time step party P_i reads from the blockchain a partial transcript $\tau^{(\tilde{r})}$, whereas at a later time step the same player reads $\tau^{(\tilde{r}')}$ for some $\tilde{r}' < \tilde{r}$. This is due to the fact that some of the messages contained in $\tau^{(\tilde{r})}$ may end up in unconfirmed blocks, and thus be discarded.

Definition 23 (Hasty execution)**.** Let $\Gamma = (\mathsf{UpdateState}, \mathsf{GetRecords}, \mathsf{Broadcast})$ be a blockchain protocol with k-consistency. A player P_i is said to be *hasty* if it behaves as follows:

- Initialize $\tau_i^{(0)} := \varepsilon$ and $\alpha_i := \varepsilon$.
- Run the following loop:

- Update the state α_i by running $\mathsf{UpdateState}(1^\lambda)$, and let $\mathcal{B}_i \leftarrow\!\!\$\ \mathsf{GetRecords}(\alpha_i)$.
- Let $\tilde{r} \geq 0$ be the maximum value such that the partial transcript $\tau^{(\tilde{r})} \in \mathcal{B}_i$.
- If the protocol is over (i.e., the transcript $\tau^{(\tilde{r})}$ is sufficient for determining the output), output the value y_i as a function of $\tau^{(\tilde{r})}$ and terminate.
- Else, compute the next protocol message $m_i^{(\tilde{r}+1)}$ and invoke $\mathsf{Broadcast}(m_i^{(\tilde{r}+1)})$.

More generally, we call φ-hasty a player that is non-hasty until a partial transcript $\tau^{(\varphi)}$ is at least k blocks deep in the blockchain, and afterwards it starts being hasty. We sometimes call φ the *finality parameter*. Note that a 0-hasty player is identical to a hasty player, whereas an ∞-hasty player is identical to a non-hasty player. We call (χ, φ)-hasty a player that is hasty for the first χ rounds, and then behaves like a φ-hasty player.

4.2.2 Security in the Presence of Hasty Players

We can now define security of MPC protocols running on the blockchain. As in the standard setting, the definition compares a protocol execution in the real world with one in the ideal setting where a trusted party is made available. The main difference with the standard definition is that the attacker A is given black-box access to the algorithms in Γ, which it can use arbitrarily. The simulator is not allowed to control the blockchain (i.e. it must simulate the view of the adversary while invoking the algorithms in Γ on behalf of the honest players).

The real model: This is the execution of π on Γ, where the honest players are φ-hasty. As usual, the adversary A is coordinated by a non-uniform distinguisher D. At the outset, D chooses the inputs $(1^\lambda, x_i)$ for each player P_i, and gives $\mathcal{I}$, $\{x_i\}_{i\in\mathcal{I}}$ and z to A, where $\mathcal{I} \subseteq [n]$ represents the set of corrupted players and z is some auxiliary input. The parties then start running π on Γ, with the honest players P_i being φ-hasty and behaving as prescribed in π (using input x_i), and with malicious parties behaving arbitrarily (directed by A). At some point, A gives to D an arbitrary function of its view; note that the latter includes the view generated via $\mathbf{EXEC}_{\Gamma,\mathsf{A},\mathsf{D}}(\lambda)$ in the blockchain protocol. Finally, D receives the outputs of the honest parties and must output a bit. We denote by $\mathbf{REAL}^{\Gamma,\varphi}_{\pi,\mathsf{A},\mathsf{D}}(\lambda)$ the random variable corresponding to D's guess.

The ideal model: This is identical to the ideal model for standard MPC, with the only difference that the simulator Sim is also responsible for simulating the attacker's view corresponding to the interaction of the honest players with the blockchain. The latter is achieved using the algorithms of the underlying blockchain protocol Γ. We denote by $\mathbf{IDEAL}^{\Gamma}_{f,\mathsf{Sim},\mathsf{D}}(\lambda)$ and $\mathbf{IDEAL}^{\Gamma}_{f_\perp,\mathsf{Sim},\mathsf{D}}(\lambda)$ the random variable corresponding to D's guess in the ideal world, where the latter is for the case of security with aborts.

Definition 24 (Secure MPC in the presence of hasty players)**.** Let π be an n-party protocol run over a blockchain protocol Γ. We say that π t-securely computes f in the presence of φ-hasty players and malicious adversaries if for every PPT

adversary A there exists a PPT simulator Sim such that for every non-uniform PPT distinguisher D corrupting at most t parties the following holds:

$$\left\{\mathbf{REAL}^{\Gamma,\varphi}_{\pi,\mathsf{A},\mathsf{D}}(\lambda)\right\}_{\lambda\in\mathbb{N}} \approx_c \left\{\mathbf{IDEAL}^{\Gamma}_{f,\mathsf{Sim},\mathsf{D}}(\lambda)\right\}_{\lambda\in\mathbb{N}}.$$

When replacing $\mathbf{IDEAL}^{\Gamma}_{f,\mathsf{Sim},\mathsf{D}}(\lambda)$ with $\mathbf{IDEAL}^{\Gamma}_{f_\perp,\mathsf{Sim},\mathsf{D}}(\lambda)$ we say that π t-securely computes f *with aborts* in the presence of φ-hasty players and malicious adversaries.

Remark 2 (On $\varphi = \infty$)**.** *One may think that every protocol π that t-securely computes f (with or without aborts) in the presence of malicious adversaries, must t-securely compute f (with or without aborts) in the presence of ∞-hasty (i.e., non-hasty) players and malicious adversaries.*

Remark 3 (On $\varphi = 0$)**.** *Note that when the players are fully hasty (i.e., $\varphi = 0$), the adversary's view in the real world may include multiple executions of the original protocol π (upon the same inputs chosen by the distinguisher). This view may not be possible to simulate in the ideal world, where the simulator can invoke the ideal functionality f only once.*

For this reason, whenever $\varphi = 0$, we implicitly assume that the simulator is allowed to query the ideal functionality f multiple times. Note that this yields a meaningful security guarantee only for certain functionalities f, similarly to the setting of resettably secure computation [GS09].

Remark 4 (On the power of the adversary)**.** *We stress that we assume that the adversary of the MPC protocol has no impact on the execution of the consensus protocol of the underlying blockchain. Note that if we would instead assume that the adversary of the MPC protocol also creates new branches and/or contributes in deciding which branch of a fork is eventually confirmed on the blockchain then he can have an unfair advantage. Indeed the adversary can start more branches when he does not like the output computed in a branch, and/or can decide which output among the various outputs appearing in different branches should be confirmed on the blockchain. Obviously the above unfair advantages are unavoidable and our protocol is still secure by introducing the unavoidable real-world attack into the ideal world, similarly to the classical fairness issue resolved through aborts in the ideal world.*

Remark 5 (On public verifiability)**.** *We notice that any on-chain MPC protocol with hasty players admits the case where a honest player complete her execution computing an output that does not necessarily correspond to the transcript that others later on will see on the blockchain. In other words, the local output computed by players could not match the publicly verifiable execution that remains visible on the blockchain. The reason why public verifiability could fail is that an execution of the protocol could be entirely contained in a branch of a fork that will not become permanent in the blockchain. The above issue is intrinsic in all protocols played on-chain in the presence of forks and hasty players. An obvious solution for a honest player consists of waiting that the last message of the protocol is confirmed on the blockchain and only after that the computation ends returning the computed output.*

Random oracle model. Our result in Section 4.3.1 are secure in the random oracle model (ROM). The definition remains the same, except that each player in the real world has now access to a truly random hash function Hash chosen at the beginning of the experiment. The simulator of the ideal world can program the random oracle.

4.3 Parallel Coin Tossing

A coin-tossing protocol allows a set of players to agree on a uniformly random string, and has many important applications (e.g., it allows to easily implement a decentralized lottery). Our protocol leverages standard techniques to achieve fairness with penalties, but does not require finality (thus allowing players to be fully hasty). We start summarizing the protocol of [ADMM16] below and we show that their protocol becomes completely insecure in the presence of hasty players. This naturally leads to our new protocol, which we describe and analyze in Section 4.3.1.

The protocol of Andrychowicz et al. Recall that in the Bitcoin ledger, each account is associated to a pair of keys $(\mathsf{pk}, \mathsf{sk})$, where pk is the verification key of a signature scheme—representing the address of an account—while sk is the corresponding secret key used to sign (the body of) the transactions. Each block on the ledger contains a list of transactions, and new blocks are issued by an entity called *miner*. The blockchain is maintained via a consensus mechanism based on proof of work; users willing to add a transaction to the ledger forward it to the miners, which will try to include it in the next minted block.

In the description below, we say that a transaction is *valid* if it is computed correctly (i.e., the signature is valid, the coins have not been spent already, and so on), and that it is *confirmed* if it appears in the common-prefix of all the miners (i.e., it is at least k-blocks deep in the ledger). Each transaction Tx includes:

- A set of input transactions $\mathsf{Tx}_1, \mathsf{Tx}_2, \cdots$ from which the coins needed for the actual transaction Tx are taken;
- A set of input scripts containing the input for the output scripts of $\mathsf{Tx}_1, \mathsf{Tx}_2, \cdots$;
- An output script defining in which condition Tx can be claimed;
- The number of coins taken from the redeemed transactions;
- A time lock t specifying when Tx becomes valid (i.e., a time-locked transaction won't be accepted by the miners before time t has passed).

The construction by [ADMM14, ADMM16] relies on a primitive called *time-locked commitment*. Let n denote the number of parties. Each party P_j creates $n-1$ $\mathsf{Commit}^j_{i \neq j}$ transactions containing a commitment to its lottery value. In particular, the output script of such a transaction ensures that it can be claimed either by P_j via an Open^j_i transaction exhibiting a valid opening for the commitment, or by another transaction that is signed by both P_j and P_i. Before posting these transactions on

the ledger, P_j creates a time-locked transaction $\mathsf{PayDeposit}_i^j$ redeeming Commit_i^j, sends it off-chain to each $\mathsf{P}_{i\neq j}$, and finally posts all the Commit_i^j transactions on the ledger. In case P_j does not open the commitment before time τ, then each recipient of a $\mathsf{PayDeposit}_i^j$ transaction can sign it and post it on the ledger; since time τ has passed, the miners will now accept the transaction as a valid transaction redeeming Commit_i^j. More in details:

Deposit phase: Each player P_j computes a commitment $y_j = \mathsf{Hash}(x_j||\delta_j)$, where δ_j is some randomness, sends off-chain the $\mathsf{PayDeposit}_i^j$ transactions (with time-lock τ) to each $\mathsf{P}_{i\neq j}$, and posts the Commit_i^j transactions.

Betting phase: P_j bets one coin in the form of a transaction $\mathsf{PutMoney}_j$ (redeeming a previous transaction held by P_j, and with P_j's signature as output script). All the players agree and sign off-chain a $\mathsf{Compute}$ transaction taking as input all the $(\mathsf{PutMoney}_j)_{j\in[n]}$ transactions, and then the last player that receives the $\mathsf{Compute}$ transaction posts it on the ledger. In order to claim this transaction, a player $\mathsf{P}_{w'}$ must exhibit the openings of the commitments of all participants: The script checks that the openings are valid, computes the index of the winner w (as a function of the values $x_1, \ldots, x_n$), and checks that $w' = w$ (i.e., the only participant that can claim the $\mathsf{Compute}$ transaction is the winner of the lottery).

Compensation phase: After time τ, in case some player P_j did not send all of its $\{\mathsf{Open}_i^j\}_{i\in[n],i\neq j}$ transactions, all the other players $\mathsf{P}_{i\neq j}$ can post the $\mathsf{PayDeposit}_i^j$ transaction, thus obtaining a compensation.

A simple attack in the presence of hasty players. The main idea behind our attack is that, in the presence of hasty players, the protocol's messages can end-up answering messages appeared on (still) unconfirmed blocks. By looking at different branches of a fork, an attacker can try to change an old (in the sense that even an answer to it has already been published on-chain) unconfirmed transaction by re-posting it, with the hope that it will end-up on a different branch and become part of the common prefix. This essentially corresponds to a reset attack on the protocol.

The construction described above relies on the (implicit) assumption that the players are non-hasty. In particular, each player P_j should wait to post its $\mathsf{PutMoney}_i^j$ transaction only after all the Commit_i^j transactions are confirmed on the ledger, in such a way that all players are aligned on the same branch (and so the miners have the $\{\mathsf{Commit}_i^j\}_{i\in[n],j\neq i}$ transactions in their common prefix). In the case of hasty players, when a fork occurs, an attacker can take advantage of the openings of the other parties played in a faster branch in order to bias the result of the lottery on a slower branch. If eventually the slower branch remains permanently in the blockchain, then clearly the attack is successful.

For concreteness, let us focus on Blum's coin tossing, in which the winner is defined to be $w = x_1 + \ldots + x_n \bmod n + 1$. Consider the following scenario:

- The (hasty) players $\mathsf{P}_1, \ldots, \mathsf{P}_n$ run a full instance of the protocol; note that this requires at least 3 blocks.

- The attacker P_n hopes to see a fork containing all the $\{\mathsf{Commit}^i_j\}$ transactions of the other $n-1$ players.
- Since the attacker P_n now knows the openings $x_1, \ldots, x_{n-1}$, it can post a new set of $\{\mathsf{Commit}'^i_n\}_{i\in[n],i\neq n}$ transactions containing a commitment to a value x'_n such that $x_1 + \ldots + x_{n-1} + x'_n \mod n+1 = n$.

In case the new set of transactions ends up on a different branch which is finally included in the common prefix, P_n wins the lottery. In the next section, we propose a new protocol that does not suffer from this problem.

4.3.1 Our PCT Protocol

We now present a parallel coin-tossing (PCT) protocol on blockchain that is secure in the presence of hasty players. The main challenge that we face is that the protocol must prevent an adversary from choosing adaptively her contribution to the coin tossing in a branch of a fork, after possibly seeing the contributions of the other players in different branches.

We tackle this problem by requiring that each honest party computes his contribution by evaluating a unique signature upon input the public keys of all players. Notice that if the adversary A sees some signatures in a branch, and changes her public key in another branch, then A cannot predict the signatures of the honest players on this other branch by the unforgeability property of the signature scheme, and thus A will not manage to bias the final output. Hence, we hash the concatenation of all the signatures in order to determine the final output. Assuming that the hash function is modelled as a random oracle, we would like to argue that the output of the protocol looks uniform.

However, the following subtlety arises. Assume without loss of generality that only P_n is corrupt and that the protocol proceeds until the end on a given branch of the blockchain. Denote by pk_n the public key chosen by the attacker. Further, assume that A notices another branch where all honest players have already sent their public keys. Now, the adversary can either: (i) publish a different public key pk'_n, or (ii) publish the same public key pk_n as in the other branch. In case A "likes" the outcome of the protocol on the first branch, she will choose option (ii) and thus can bias the protocol output.

To avoid the above attack, we identify each branch with a string *bid* that is uniquely associated to it, and include *bid* as part of the message to sign. Intuitively, this solves the previous problem as, even if all the public keys stay the same on two different branches, the value *bid* will change thus ensuring that the protocol output will also be different (and uniformly random). We proceed with a more detailed description of our protocol (see also Fig. 4.1).[1]

- One of the players chooses a random value *sid* that represents the identifier of the current protocol execution, and publishes *sid* on the blockchain.

[1]Note that our protocol can be run on generic blockchains. In Section 4.5 , we provide an implementation using Ethereum smart contracts, but the protocol can also be implemented in Bitcoin using the opcode OP_RETURN in case players do not need to get fairness with penalties.

- Each player P_i willing to participate generates the public and private keys for the unique signature $(\mathsf{pk}_i, \mathsf{sk}_i) \leftarrow\!\!\$\ \mathsf{Gen}(1^\lambda)$, and publishes pk_i on the blockchain.
- Each player P_i lets $y_i = \mathsf{Sign}(\mathsf{sk}_i, \mathsf{pk}_1||\cdots||\mathsf{pk}_n||sid||bid)$, where *bid* is the hash of the blockchain[2] up to the block that contains the last public key, and publishes y_i on the blockchain.
- Each player P_i checks that $\mathsf{Verify}(\mathsf{pk}_j, x, y_j) = 1$ for all $j \neq i$, where $x = \mathsf{pk}_1||\cdots||\mathsf{pk}_n||sid||bid$, and outputs $\mathsf{Hash}(y_1||\cdots||y_n)$.

We stress that thanks to the value *bid*, the protocol execution becomes branch dependent. In particular, the chances of success of a corrupted P_j to bias the output are not affected by the potential use of different public keys in branches of a fork corresponding to a protocol run with a given *sid*.

Parallel Coin Tossing Protocol π^*_{pct}

Let $(\mathsf{Gen}, \mathsf{Sign}, \mathsf{Verify})$ be a signature scheme with message space $\mathcal{M} = \{0,1\}^*$, and $\mathsf{Hash} : \{0,1\}^* \to \{0,1\}^\lambda$ be a hash function.

- P_1 picks $sid \leftarrow\!\!\$\ \{0,1\}^\lambda$, and runs $\mathsf{Broadcast}(sid)$.
- For each $i \in [n]$, P_i generates $(\mathsf{pk}_i, \mathsf{sk}_i) \leftarrow\!\!\$\ \mathsf{Gen}(1^\lambda)$ and runs $\mathsf{Broadcast}(\mathsf{pk}_i)$.
- For each $i \in [n]$, P_i executes $\mathcal{B}_i \leftarrow\!\!\$\ \mathsf{GetRecords}(1^\lambda, \mathsf{UpdateState}(1^\lambda))$ until all public keys $\mathsf{pk}_1, \ldots, \mathsf{pk}_n$ are contained in $\mathcal{B}_i$, and then defines $bid := \mathsf{Hash}(\mathcal{B}_i)$.
- For each $i \in [n]$, P_i computes $y_i = \mathsf{Sign}(\mathsf{sk}_i, x)$, where $x = \mathsf{pk}_1||\ldots||\mathsf{pk}_n||sid||bid$ and runs $\mathsf{Broadcast}(y_i)$.
- For each $i, j \in [n]$, with $i \neq j$, P_i checks that $\mathsf{Verify}(\mathsf{pk}_i, x, y_i) = 1$, and, if so, it outputs $\mathsf{Hash}(y_1||\cdots||y_n)$ and else it aborts.

Figure 4.1. Our new protocol for parallel coin tossing.

Security analysis. Let f^{pct} be the n-party functionality that picks a uniformly random string ω and sends it to all the n parties. The theorem below establishes the security of our coin-tossing protocol in the (programmable) random oracle (RO) model. We note that the security of the original protocol by Andrychowicz et al. [ADMM14, ADMM16] also relies on the RO heuristic, as do all currently known analysis of blockchain protocols [GKL15, PSS17].

Theorem 5. *If* $(\mathsf{Gen}, \mathsf{Sign}, \mathsf{Verify})$ *is a unique signature scheme, the protocol of Fig. 4.1 securely implements the functionality* f^{pct} *in the presence of hasty players and malicious adversaries with aborts, in the programmable RO model.*

We need to show that for every PPT adversary A, there exists a PPT simulator Sim such that no non-uniform PPT distinguisher D can tell apart the experiments

[2]For efficiency the hash can be more simply applied to the block containing pk_n. Nevertheless, for the sake of simplicity of the protocol description and of the security analysis we will stick with hashing the entire blockchain.

$\mathbf{REAL}^{\Gamma,0}_{\pi,\mathsf{A},\mathsf{D}}(\lambda)$ and $\mathbf{IDEAL}^{\Gamma}_{f_{\perp}^{\mathsf{pct}},\mathsf{Sim},\mathsf{D}}(\lambda)$. In particular, the simulator needs to simulate the interaction of the honest players with the blockchain protocol Γ as it happens in the real experiment.

Recall that the ideal coin-tossing functionality does not take any input, and returns a random string ω to all parties. Intuitively, the simulator Sim will just emulate a real execution of the protocol by reading the attacker's messages from the blockchain. Furthermore, after querying the ideal functionality, Sim will program the random oracle upon input ω. At the same time, Sim needs to simulate the answer to A's random oracle queries q, which is done as follows:

- If A already queried the random oracle upon q, obtaining answer $\tilde{\omega}$, then Sim returns the same value $\tilde{\omega}$;
- Else, if $\mathsf{pk}_1, \ldots, \mathsf{pk}_n$ are not all available on the blockchain, Sim answers the query q with a fresh random value $\tilde{\omega}$;
- Otherwise, Sim parses q as $\tilde{y}_1 || \ldots || \tilde{y}_n$, and checks that $\mathsf{Verify}(\mathsf{pk}_i, x, \tilde{y}_i) = 1$ for all $i \in [n]$, where $x = \mathsf{pk}_1 || \ldots || \mathsf{pk}_n || sid || bid$ and bid is derived by hashing the blockchain up to the block containing the last public key. If the check passes, Sim programs $\mathsf{Hash}(q)$ to the value ω obtained by $f_{\perp}^{\mathsf{pct}}$, else it answers with a fresh random value $\tilde{\omega}$.

The above strategy does not consider the fact that the real protocol may be run multiple times on different branches. However, since each execution has associated a different $bid' \neq bid$, the simulator can simply query again the functionality in order to obtain a new random value ω', and program the random oracle on ω' in the execution corresponding to bid'. Roughly speaking, the attacker A can potentially take advantage from the following three actions: (i) A can refuse to publish her signed message; (ii) A can try to change the public keys of corrupted parties over different branches; (iii) A can try to choose the signature that produces the best possible output. Action (i) is equivalent to aborting the protocol, and can be easily handled by the simulator, since we achieve security with aborts. Action (ii) is tackled by the use effect of the of bid: regardless of A using the same or another public key, the outcome of the protocol in different branches are always independent. Finally, as for action (iii), note that this attack is prevented by the uniqueness property of the signature scheme that ensures that for every (possibly malicious) public key pk and every input x, there exists at most one valid signature y that is not rejected by the verification algorithm.

Proof. We begin by describing the simulator. Upon input the set of corrupted parties $\mathcal{I}$, and auxiliary input z, the simulator Sim proceeds as follows:[3]

1. Initialize an empty array $\mathcal{L}$. Sample a random value $sid \leftarrow\!\!\$\ \{0,1\}^{\lambda}$, and run $\mathsf{Broadcast}(sid)$.
2. Generate $(\mathsf{pk}_i, \mathsf{sk}_i) \leftarrow\!\!\$\ \mathsf{Gen}(1^{\lambda})$ and run $\mathsf{Broadcast}(\mathsf{pk}_i)$ for each $i \notin \mathcal{I}$.

[3]For simplicity, we assume that the player P_1 initiating the protocol is honest; if not, it is easy to adapt the simulation by having Sim using the value sid written by A on the blockchain.

3. Keep running $\mathsf{GetRecords}(\mathsf{UpdateState}(1^\lambda))$ until all public keys $\mathsf{pk}_1, \ldots, \mathsf{pk}_n$ are published on the blockchain; when that happens define *bid* to be the hash of the blockchain up to the block containing the last public key.

4. If $\mathcal{L}$ does not already contain the value *bid*, query the ideal functionality $f_\perp^{\mathsf{pct}}$, obtaining a random value ω, and store (bid, ω) in $\mathcal{L}$. Then complete the simulation for branch *bid* as follows:

   (a) For each $i \notin \mathcal{I}$, let $y_i = \mathsf{Sign}(\mathsf{sk}_i, x)$ where $x = \mathsf{pk}_1||\cdots||\mathsf{pk}_n||sid||bid$. Then run $\mathsf{Broadcast}(y_i)$.

   (b) Keep running $\mathsf{GetRecords}(\mathsf{UpdateState}(1^\lambda))$ until all values $y_1, \ldots, y_n$ are published on the blockchain; when that happens check that $\mathsf{Verify}(\mathsf{pk}_i, x, y_i) = 1$ for all $i \in [n]$. If any of these checks fails, send `abort` to $f_\perp^{\mathsf{pct}}$, simulate A aborting, and terminate. Else, in case $\mathsf{Hash}(y_1||\cdots||y_n)$ was already set to $\omega' \neq \omega$, abort the simulation and terminate. In this last case, we say that the simulator fails to program the random oracle.

5. Upon input a random oracle query q from A, answer as follows:

   (a) If there exists a pair $(bid, \omega) \in \mathcal{L}$ for which it is possible to parse $q := y_1||\cdots||y_n$ so that $\mathsf{Verify}(\mathsf{pk}_i, x, y_i) = 1$ for all $i \in [n]$—where $x = pk_1||\cdots||\mathsf{pk}_n||sid||bid$ for the values $sid, (\mathsf{pk}_1, \ldots, \mathsf{pk}_n)$ that appear in the simulation of branch *bid*—program $\mathsf{Hash}(q) := \omega$ and answer query q with ω. If ω was already given as output for a different query then we say that the simulator fails creating a collision when programming the random oracle.

   (b) Else, return a random value (maintaining consistency among repeated queries).

Notice that Sim simulates perfectly the messages written on the blockchain by the honest players in a real execution of π^*_{pct}, including their interaction with the blockchain protocol Γ. Moreover, in each branch *bid*, the simulator perfectly emulates an abort of the protocol due to the fact that A sends invalid signatures. Hence, the only difference between the real and ideal experiment is that in the latter, for each branch *bid*, the simulator forces the protocol output to be a fresh random value received from the ideal coin-tossing functionality. Consider the following events:

Event $\mathbf{BAD}_1$: The event becomes true in case the simulator fails to program the random oracle in step 4b of the simulation.

Event $\mathbf{BAD}_2$: The event becomes true in case, the simulator fails creating a collision as described in step 5a. This is possible when for a given branch *bid*, there exists an index $i \in \mathcal{I}$ such that the attacker produces two outputs y_i and y_i' such that $y_i \neq y_i'$ and $\mathsf{Verify}(\mathsf{pk}_i, x, y_i) = \mathsf{Verify}(\mathsf{pk}_i, x, y_i') = 1$, and queries the random oracle first using y_i' therefore obtaining ω, and then querying y_i therefore obtain again ω.

Event $\mathbf{BAD}_3$: The event becomes true in case there exist two branches with different public keys, but for which the value *bid* is the same.

Let $\mathbf{BAD} = \mathbf{BAD}_1 \cup \mathbf{BAD}_2 \cup \mathbf{BAD}_3$. We claim that conditioning on $\overline{\mathbf{BAD}}$, the experiments $\mathbf{REAL}^{\Gamma,0}_{\pi,\mathsf{A},\mathsf{D}}(\lambda)$ and $\mathbf{IDEAL}^{\Gamma}_{f^{\mathsf{pct}}_{\perp},\mathsf{Sim},\mathsf{D}}(\lambda)$ are identical. This is because conditioning on $\mathbf{BAD}_1$ not happening, the attacker never queries the random oracle on $\mathsf{pk}_1||\cdots||\mathsf{pk}_n||sid||bid$ before the protocol execution on branch bid terminates, and thus the final output ω on that branch is uniformly distributed from the point of view of A (as it happens to be in the ideal world). Furthermore, conditioning on $\mathbf{BAD}_2$ and $\mathbf{BAD}_3$ not happening, the simulator correctly assigns a single random value ω to each branch identified by bid.

Next, we show that each of the above events happens at most with negligible probability. By a standard argument, this concludes the proof as the computational distance between $\mathbf{REAL}^{\Gamma,0}_{\pi,\mathsf{A},\mathsf{D}}(\lambda)$ and $\mathbf{IDEAL}^{\Gamma}_{f^{\mathsf{pct}}_{\perp},\mathsf{Sim},\mathsf{D}}(\lambda)$ is at most equal to the probability of event $\mathbf{BAD}$.

Lemma 17. *For all PPT* A*, there exists a negligible function* $\nu_1(\cdot)$ *such that* $\mathbb{P}[\mathbf{BAD}_1] \leq \nu_1(\lambda)$.

Proof. Notice that event $\mathbf{BAD}_1$ happens if and only if there exists a protocol execution corresponding to a branch bid for which the attacker A queries the random oracle upon input $y_1||\cdots||y_n$ before these values appear on the blockchain. Intuitively, this requires that A forges a signature for one of the public keys corresponding to one of the honest players, and thus $\mathbb{P}[\mathbf{BAD}_1]$ must be negligible. The reduction is straightforward: Given a PPT attacker provoking event $\mathbf{BAD}_1$ with non-negligible probability, we can construct a PPT attacker A' as follows. Initially, A' tries to guess the index $i \notin \mathcal{I}$ and the branch index $j \in \mathsf{poly}(\lambda)(\lambda)$ corresponding to the protocol execution in which A will provoke event $\mathbf{BAD}_1$. Hence, A' simulates the execution of protocol π^*_{pct} with A as done in the real experiment, except that on the j-th branch it sets pk_i to be the public key pk^* received from the challenger.

Finally, A' waits that A makes a random oracle query $y_1||\cdots||y_n$ such that $\mathsf{Verify}(\mathsf{pk}_i, x, y_i) = 1$, where $x = \mathsf{pk}_1||\cdots||\mathsf{pk}_n||sid||bid$; if the latter does not happen, A' aborts, else it forwards (x, y_i) to the challenger. The proof follows by observing that, with non-negligible probability, A' does not abort, and thus it breaks unforgeability with non-negligible probability. □

Lemma 18. *For all PPT* A*, there exists a negligible function* $\nu_2(\cdot)$ *such that* $\mathbb{P}[\mathbf{BAD}_2] \leq \nu_2(\lambda)$.

Proof. Notice that event $\mathbf{BAD}_2$ directly contradicts uniqueness of the signature scheme $(\mathsf{Gen}, \mathsf{Sign}, \mathsf{Verify})$. Hence, $\mathbb{P}[\mathbf{BAD}_2]$ must be negligible. The reduction is straightforward: Given a PPT attacker provoking event $\mathbf{BAD}_2$ with non-negligible probability, we can construct a PPT attacker A' as follows. Initially, A' tries to guess the index $i \in \mathcal{I}$ and the branch index $j \in \mathsf{poly}(\lambda)(\lambda)$ corresponding to the protocol execution in which A will provoke event $\mathbf{BAD}_2$. Hence, A' simulates the execution of protocol π^*_{pct} with A as done in the real experiment.

Finally, A' waits that A publishes on the j-th branch the two values y_i, y'_i which make the event $\mathbf{BAD}_2$ become true; if the latter does not happen, A' aborts, else it forwards $(\mathsf{pk}_i, y_i, y'_i)$ to the challenger where the public key pk_i is the public key corresponding to the i-th player on the j-th branch. The proof follows by

observing that, with non-negligible probability, A' does not abort, and thus it breaks uniqueness with non-negligible probability. □

Lemma 19. *For all PPT* A*, there exists a negligible function* $\nu_3(\cdot)$ *such that* $\mathbb{P}[\mathbf{BAD}_3] \leq \nu_3(\lambda)$.

Proof. Notice that event $\mathbf{BAD}_3$ directly contradicts collision resistance of the hash function Hash. Hence, $\mathbb{P}[\mathbf{BAD}_3]$ must be negligible. The reduction is straightforward: Given a PPT attacker provoking event $\mathbf{BAD}_3$ with non-negligible probability, we can construct a PPT attacker A' that simply emulates a protocol execution with A as in the real experiment. The values *bid*, as well as the answers to A's random oracle queries, are obtained by querying the target random oracle. Hence, whenever A' finds a fork with two different branches $\mathcal{B}$ and $\mathcal{B}'$ such that $\mathsf{Hash}(\mathcal{B}) = \mathsf{Hash}(\mathcal{B}')$, it outputs $(\mathcal{B}, \mathcal{B}')$ and stops. Since A provokes event $\mathbf{BAD}_3$ with non-negligible probability, A' wins with the same probability. This concludes the proof. □

Putting the above lemmas together, by a union bound, there exists a negligible function $\nu(\cdot)$ such that $\mathbb{P}[\mathbf{BAD}] \leq \nu(\lambda)$, as desired. □

Fairness with penalties. We now discuss how to augment the protocol π^*_{pct} in order to achieve fairness with penalties. First of all, each party should publish also a deposit along with her public key on the blockchain. The deposit can be redeemed by showing a valid signature on the value $x = \mathsf{pk}_1||\cdots||\mathsf{pk}_n||sid||bid$.

Assume that P_n is corrupted. The adversary can wait that the honest parties publish their value $y_1, \ldots, y_{n-1}$ on a given branch, and thus locally compute the output $\mathsf{Hash}(y_1||\cdots||y_n)$, where y_n is P_n's signatures on x corresponding to public key pk_n. Now, if P_n does not like the output it can either: (i) publish y_n in any case , or (ii) decide not to publish y_n. In case (i), P_n plays honestly, takes back his deposit and every player obtains the output. In case (ii), P_n aborts the protocol, but loses her deposit.

Note that the penalty mechanism for our protocol is too sophisticated for the scripting language used in Bitcoin. Instead in Ethereum we can design a smart contract to define the PCT protocol, having fairness with penalties and without penalizing the efficiency.

In Section 4.5 we give details about how the smart contract works.

We call $\tilde{\pi}^*_{\mathsf{pct}}$ the fair (with penalties) version of protocol π^*_{pct}. The informal description of the smart contract used in $\tilde{\pi}^*_{\mathsf{pct}}$ is given in Fig. 4.2 and the protocol is described below:

(i) *Setup phase:* At the beginning, one of the players creates the smart contract. When the contract is posted on the blockchain, the constructor automatically generates a unique session identifier *sid*.

(ii) *Deposit phase:* For each $i \in [n]$, P_i can decide to participate to the PCT protocol by triggering the function d to send a safety deposit and his public key pk_i of an unique signature scheme. After `time1` blocks have passed, if $(\mathsf{pk}_1, \ldots, \mathsf{pk}_n)$ are collected by the smart contract, it computes *bid* as $\mathsf{Hash}(\mathcal{B})$, where $\mathcal{B}$ is the blockchain that contains $(\mathsf{pk}_1, \ldots, \mathsf{pk}_n)$. The deposit phase ends and parties can start to redeem their deposit.

(iii) *Claim phase:* For each $i \in [n]$, P_i can claim his deposit back by triggering the function `claim` of the smart contract and sending a value y_i such that $\mathsf{Verify}(\mathsf{pk}_i, x, y_i) = 1$, where $x = \mathsf{pk}_1||\cdots||\mathsf{pk}_n||sid||bid$, and pk_i is the public key of P_i. After `time2` blocks have passed, the claim phase ends and the smart contract computes and publishes the output as $\mathsf{Hash}(y_1||\cdots||y_n)$.

The **Parallel Coin Tossing Smart Contract** runs with players $\mathsf{P}_1, \ldots, \mathsf{P}_n$ and consists of two main functions d and `claim` and two fixed timestamps `time1`,`time2` and a session id *sid*.

Deposit Phase: In round t_1, when $d(\mathsf{pk}_i)$ together with d coins is triggered from a party P_i, store (i, pk_i). Then, if $(\mathsf{pk}_1, \ldots, \mathsf{pk}_n)$ are stored, compute and store $bid := \mathsf{Hash}(\mathcal{B})$ and proceed to the Claim Phase. Otherwise, for all i, if the message (i, pk_i) has been stored, send back d coins to P_i and terminate.

Claim Phase: In round t_2, when `claim`(i, y_i) is triggered from P_i, check if $\mathsf{Verify}(\mathsf{pk}_i, x, y_i) = 1$, where $x = \mathsf{pk}_1||\cdots||\mathsf{pk}_n||sid||bid$. If the check is correct send d coins back to P_i.

Compute Phase: If, after `time2`, all the y_i are correctly claimed, compute and publish $\mathsf{Hash}(y_1, \ldots, y_n)$.

Figure 4.2. Smart contract for parallel coin tossing.

Theorem 6. *If* $(\mathsf{Gen}, \mathsf{Sign}, \mathsf{Verify})$ *is a unique signature scheme,* $\tilde{\pi}^*_{\mathsf{pct}}$ *described in Fig. 4.2 securely realizes* f^{pct} *and satisfies fairness with penalties in the presence of hasty players and malicious adversaries, in the programmable RO model.*

Proof Sketch. The privacy of the protocol is proven by following the same outline of Theorem. 5. We have to prove that $\tilde{\pi}^*_{\mathsf{pct}}$ is fair (with penalties).

There are four possible scenarios that can happen and in each of them either all parties learn the output or the adversary A loses her deposit. An output *out* of $\tilde{\pi}^*_{\mathsf{pct}}$ is considered valid if it is confirmed on the blockchain. The four scenarios are described below.

Scenario 1: A does not exploit branches to play different public keys on different execution of $\tilde{\pi}^*_{\mathsf{pct}}$. Fairness follows from the smart contract execution (i.e., if A does not provide the signature of x, A will lose her deposit).

Scenario 2: There is the following time-line: there is a fork with two branches b_1 and b_2 and the Setup Phase is published before the fork, but the Deposit Phase is executed after the fork. A aborts (i.e., A does not provide the signature of x) the execution of $\tilde{\pi}^*_{\mathsf{pct}}$ in b_1 and exploits b_2 in the following way: in b_2 a corrupted player P_i double spends (for any kind of transaction) the coins

deposited in b_1. In this case, either b_1 gets confirmed, thus, boiling down to Scenario 1, or b_2 get confirmed and all the transactions sent to the smart contract of $\tilde{\pi}^*_{\mathsf{pct}}$ in branch b_1 are not valid b_2 since the deposit of P_i is previously spent in another transaction. It guarantees fairness since in b_1 the adversary is punished, and in b_2 there's no execution, and so no valid output

Scenario 3: This scenario follows the same time-line of Scenario 2. A aborts (i.e., A does not provide the signature of x) the execution of $\tilde{\pi}^*_{\mathsf{pct}}$ in b_1 and exploits b_2 in the following way: a corrupted player P_i publishes a different public key pk_i in b_2 (wlog., we analyze the case with two executions but it can be extended to multiple executions). In this case, the execution of $\tilde{\pi}^*_{\mathsf{pct}}$ is restarted from the beginning of the Deposit Phase in b_2 and the output that A learned on b_1 is not valid anymore. If b_1 gets confirmed it boils down to Scenario 1, otherwise the honest players learn the valid output computed in b_2. It guarantees fairness because in b_1 the adversary is punished, while in b_2 all parties will learn the output.

Scenario 4: There is the following time-line: there is a fork with two branches b_1 and b_2 and the Deposit Phase is published before the fork, but the Claim Phase is executed after the fork. A corrupted party P_i sends $y_i = \mathsf{Sign}(\mathsf{sk}_i, x)$ in the Claim Phase in b_1 and disliking the output *out* of the execution of $\tilde{\pi}^*_{\mathsf{pct}}$ in b_1, P_i exploits b_2 to send $y'_i \neq y_i$. We note that if y'_i is a valid signature for x under secret key sk_i we can create an adversary A' breaking the uniqueness of the signature scheme. It means that y'_i cannot be a valid signature. If b_1 gets confirmed P_i is not penalized and every player will learn *out*, otherwise if b_2 gets confirmed P_i is penalized and no party will learn the output.

□

A remark on DoS attacks. Note that in $\tilde{\pi}^*_{\mathsf{pct}}$ there is no need to fix the identity of the participants "before" the execution of the protocol. We can consider the case in which a party P_i participates to a protocol execution only after she triggers the d function giving as an input her public key. Our PCT protocol can be executed even if only two parties decide to participate and thus n does not need to be fixed beforehand. Moreover, `time1` is independent of n and of the number of blocks to wait for considering a transaction confirmed. Registered parties are not incentivized to abort the protocol (i.e., by not triggering the `claim` function) due to financial compensation (parties must send a collateral deposit together with the first message). This makes DoS attacks in which the attacker aborts the protocol multiple times (making honest parties waste time and money) financially inconvenient.

4.3.2 Experimental Evaluation

We also provide some experiments to show noticeable improvements of our PCT protocol with respect to the lottery protocol of Andrychowicz et al. in terms of number of blocks needed for completion of the protocol.

We evaluate the efficiency of π^*_{pct} compared with the protocol from [ADMM14, ADMM16]. To evaluate the efficiency in the best case we consider the following assumptions:

- Transactions in the last k blocks are considered not confirmed yet.
- All parties send the message at round i of the protocol as soon as they read all messages from round $i-1$ on $\mathcal{B}_i^{\lceil k}$, where k is 0 in case of hasty executions.
- Whenever a player broadcasts a transaction, it appears in the next block.
- All messages in a round of the MPC protocol fit in a single block.

In case of non-hasty executions if we have a ρ-round MPC protocol π running on the blockchain, the number of blocks needed to complete the execution with the previous assumption is $\rho \cdot k$.

Analysis. We now give a comparison between our coin-tossing protocol and the one of Andrychowicz et al. In particular we will interpret the execution time as the number of blocks needed to complete the steps of each protocol. To allow for a fair comparison between our coin-tossing protocol and the one presented in [ADMM14, ADMM16], we implemented both protocols in Solidity using Ethereum smart contracts.[4] See Fig. 4.6 and Fig. 4.7 for the code of the smart contracts.

For both protocols, in the deposit phase, a timeout $\bar{t}$ must be provided by the contract creator, so that players have enough time to send their deposits together with the corresponding additional information required by the protocol. In the ideal conditions described above, the timeout can be of just one block. The same argument applies to the opening phase of [ADMM14, ADMM16]. A comparison is described below:

- *Lottery:* Due to the expressiveness of smart contracts, our implementation of [ADMM14, ADMM16] requires one step less than the original implementation using in bitcoin. Specifically, we can embed the betting phase in the commit/deposit phase, by just requiring that the players deposit 1 more coin. Since in their setting block confirmation is required at each step, the overall execution takes exactly $3 \cdot k$ blocks (including one round for posting the smart contract).
- *Our PCT:* As proven in Section 4.3.1, our PCT protocol can be executed in fully-hasty mode. The entire execution consists of $2 + k$ blocks (including 1 block for posting the contract and confirmation of the output). In the worst case, where all messages will appear to the state of the honest player after k blocks for each step, the overall execution takes $3 \cdot k$, as much as [ADMM16].

GAS consumption. As it can be seen in Fig. 4.3, PCT is more expensive in terms of GAS consumption than Lottery. It is well motivated by the fact that Lottery uses only hash function to compute the commitments and no other expensive computations. Our PCT protocol needs also unique signatures. Our GAS calculation for unique signature is based on the BLS signatures implementation provided for testing in [Fic], but improved implementations could potentially lower the GAS consumption. It can be seen anyway as an affordable cost to pay to achieve efficiency still maintaining the same security guarantees.

[4] Notice that the average time for a new block to appear is around 15 seconds[Eth].

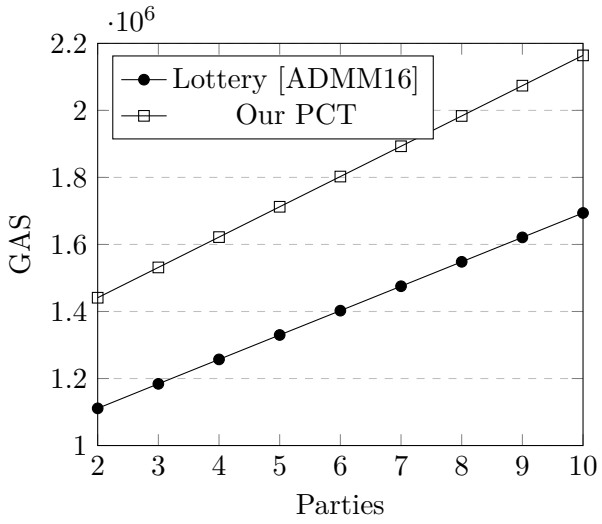


Figure 4.3. GAS consumption comparison between our smart contract implementation of PCT (Section 4.3.1) protocol and the Lottery of Andrychowicz et al. [ADMM16] (Section 4.3).

4.4 Our Generic Compiler

Our compiler starts from the observation that a stand-alone MPC protocol could be insecure when executed on a blockchain. To be concrete, a rewinding simulator of the MPC protocol can not be used to prove the security of the on-chain MPC protocol, since rewinding would have the unclear meaning of rewinding the blockchain. Moreover, we do not want to give control of the blockchain to the simulator (i.e., no control of the majority of the stake, of the computational power, and so on) since our result aims at being generic w.r.t. the type of blockchain used. Essentially, the simulator is going to incarnate just the honest players of the MPC protocol during the simulation. In order to perform a simulation in the presence of a concurrently played blockchain protocol, (i.e., rewinding is not possible and the blockchain is generic and therefore not controlled by the simulator), we therefore require the initial protocol π received in input by the compiler to be universally composable secure. This guarantees the existence of a straight-line simulator and allows us to avoid simulators that "control" the blockchain[5], therefore allowing the applicability of our results to generic blockchains. Additionally, we require π to have only "digital" communication since players when running the protocol on-chain must produce messages that consists of bits only. Therefore an exchange of hardware (e.g., PUFs) in π can not be accepted.

Notice that the original protocol might require private and authenticated channels. Since the entire traffic of our protocol will be redirected to the blockchain, we will use public-key encryption and digital signatures. The first message of each player in the compiled protocol will consist of a pair of public keys, one to receive

[5]Typically a simulator that controls the blockchain requires some specific assumptions on the blockchain like in [GG17] where only some restricted proof-of-stake blockchains were compatible with the simulation.

encrypted messages and one to allow others to verify signatures of messages. Intuitively our transformation proceeds as follows. Our starting point is any MPC protocol π UC-securely computing an n-party functionality $f : (\{0,1\}^*)^n \to (\{0,1\}^*)^n$ in the presence of malicious players (with aborts). Hence, the honest players fix the random tape for running π and simply execute protocol π by broadcasting their messages on the blockchain. Furthermore, each honest player P_i keeps track of the longest protocol transcript α_i generated so far and, in the presence of a fork, aborts the execution in case the view on a given branch is not consistent with α_i. This intuitively ensures that the underlying protocol π is run only once, even in the presence of forks.

Since the initial protocol π may require private channels between the players, we need to augment the above transformation in such a way that subsets of honest parties can exchange messages in a confidential and authenticated manner. Let $m_{i,j}^{(r)}$ be the message that P_i sends to P_j at the generic round r. The latter is achieved by having P_i encrypting $m_{i,j}^{(r)}$ using the public encryption key ek_j of P_j, and then signing the resulting ciphertext $c_{i,j}^{(r)}$ with its own private signing key sk_i, which is the standard way of building a secure channel. We refer the reader to the next sub-section for a formal description.

On fairness through penalties. To obtain a fair with penalties protocol π_{fair}, we use the following technique, borrowed from [KB14, BK14]. Let's consider a protocol π' running with parties $\mathsf{P}_1, \ldots, \mathsf{P}_n$ for a functionality f' that, given the output $y \leftarrow f(x_1, \ldots, x_n)$, where x_i is the input of player P_i, secret shares y into $(\sigma_1, \ldots, \sigma_n)$ (for a full threshold sharing scheme), generates a set of commitments $C = (\gamma_1, \ldots, \gamma_n)$ such that γ_i is the commitment of σ_i. Each player P_i obtains as an output the pair (C, σ_i)[6]. The fair protocol with penalties in the presence of hasty players can be obtained as follows: (i) We compile π' with our generic compiler, obtaining π'_{bc}. (ii) In our protocol π_{fair}, parties $\mathsf{P}_1, \ldots, \mathsf{P}_n$ first engage in π'_{bc}. After π'_{bc} ends, each P_i obtains the output (C, σ_i). Now, each P_i has a limited time t_1 to send his tuple C to a smart contract together with a payment of some deposit. (iii) If everyone sent the correct tuple C, each player P_i has another time shift t_2 to send their share σ_i of γ_i to receive back their deposit. Else, if after t_2, $(\sigma_1, \ldots, \sigma_n)$ are posted to the smart contract, each P_i can reconstruct the output by using all collected shares. Else, players that have not opened their shares within t_2, will be penalized since their coins will remain frozen forever into the smart contract.
We prove that fairness can be achieved if honest parties playing π_{fair} wait for confirmation only of step (ii). The reason for requiring the confirmation of phase (ii) is that otherwise the adversary can try to generate an abort during the execution of π'_{bc} after learning the output of the entire protocol π_{fair} on a different branch. Now, let's say that t is the time needed for transaction confirmation in the blockchain, and r the number of rounds of π'_{bc}, π_{fair} requires around $r + 2t$ blocks to complete the on-chain execution (including output confirmation). To maintain security of Kumaresan et al. protocols by blindly posting messages on-chain, the overall execution requires around $r \cdot t$ blocks to be successfully terminated. More details are

[6] P_i implicitly receives also a decommitment information of γ_i.

provided in Section 4.4.3.

Remark on DoS attacks. Note that in our construction deposits can be made at the end of step (ii) since adversaries trying to violate fairness can be spotted only during step (iii). Therefore an adversary can freely abort the execution before step (iii). Intuitively, by taking as input a protocol achieving identifiable abort [IOZ14] that is publicly verifiable[7], a player cheating in any point of the protocol execution can be successfully spotted and punished. This can be done by making a player posting a smart contract that will act as an external judge that exploits public verifiability of the underlying protocol. Unfortunately, protocols compiled with our construction would lose the identifiable abort property. This is due to the fact that the adversary can make honest players aborting by running two correct executions of the underlying protocol on two branches but using different messages. In such case, the two executions would be both considered valid in both branches by the smart contract mentioned above.

We provide the formal description of our compiler in the following section, while in Section 4.4.2 we analyze its security. Finally, in Section 4.4.3, we formally discuss how to extend our generic transformation in order to achieve fairness with penalties, as long as the players start being hasty after the confirmation of the first round.

4.4.1 Compiler Description

Intuitively our transformation proceeds as follows. Our starting point is any MPC protocol π UC-securely computing an n-party functionality $f : (\{0,1\}^*)^n \to (\{0,1\}^*)^n$ in the presence of malicious players (with aborts). Hence, the honest players fix the random tape for running π and simply execute protocol π by broadcasting their messages on the blockchain. Furthermore, each honest player P_i keeps track of the longest protocol transcript α_i generated so far and, in the presence of a fork, aborts the execution in case the view on a given branch is not consistent with α_i. This intuitively ensures that the underlying protocol π is run only once, even in the presence of forks.

Since the initial protocol π may require private channels between the players, we need to augment the above transformation in such a way that subsets of honest parties can exchange messages in a confidential and authenticated manner. Let $m_{i,j}^{(r)}$ be the message that P_i sends to P_j at the generic round r. The latter is achieved by having P_i encrypting $m_{i,j}^{(r)}$ using the public encryption key ek_j of P_j, and then signing the resulting ciphertext $c_{i,j}^{(r)}$ with its own private signing key sk_i, which is the standard way of building a secure channel. We refer the reader to Fig. 4.4 for a formal description. Note that wlog. we assume that for each round of the underlying protocol π every P_i sends a single message to each $\mathsf{P}_{j\neq i}$ over a private and authenticated channel. Moreover, P_i picks a sufficiently long random tape ω_i that is then used to run π over Γ. Observe that ω_i includes both the randomness required to compute the messages in π and the random coins used to encrypt them. In particular, in the presence of forks, an honest P_i that does not abort broadcasts on the blockchain exactly the same ciphertexts on multiple branches.

[7] An efficient construction can be found at [BOSS20].

Generic Compiler π^*

Let π be an n-party ρ-round protocol, and $\Gamma = (\mathsf{UpdateState}, \mathsf{GetRecords}, \mathsf{Broadcast})$ be a blockchain protocol. Further, let $(\mathsf{Gen}, \mathsf{Enc}, \mathsf{Dec})$ be a PKE scheme and $(\mathsf{Gen}', \mathsf{Sign}, \mathsf{Verify})$ be a signature scheme, both with domain $\{0,1\}^*$. The protocol π^* proceeds as follows:

- For $i \in [n]$, each player P_i initializes $\alpha_i := \varepsilon$, samples $(ek_i, dk_i) \leftarrow\!\!\$\ \mathsf{Gen}(1^\lambda)$, $(\mathsf{vk}_i, \mathsf{sk}_i) \leftarrow\!\!\$\ \mathsf{Gen}'(1^\lambda)$, and $\omega_i \leftarrow\!\!\$\ \{0,1\}^*$, and invokes $\mathsf{Broadcast}(ek_i||\mathsf{vk}_i||i)$.
- For $i \in [n]$, each player P_i keeps running $\alpha_i \leftarrow\!\!\$\ \mathsf{UpdateState}(1^\lambda)$ and $\mathcal{B}_i \leftarrow\!\!\$\ \mathsf{GetRecords}(\alpha_i)$ until all the messages $(ek_j, \mathsf{vk}_j)_{j\in[n]} \in \mathcal{B}_i$.
- For $i \in [n]$, each player P_i sets $\tau^{(0)} := (ek_j, \mathsf{vk}_j)_{j\in[n]}$ and $\alpha_i := \tau^{(0)}$, and then runs the following loop:
  1. Update the state α_i by running $\mathsf{UpdateState}(1^\lambda)$, and let $\mathcal{B}_i \leftarrow\!\!\$\ \mathsf{GetRecords}(\alpha_i)$.
  2. Let $\tilde{r} \geq 0$ be the maximum value such that the partial transcript $\tau^{(\tilde{r})} \in \mathcal{B}_i$. Then:
     - If the ciphertexts in $\tau^{(\tilde{r})}$ are not consistent with those in α_i, output $\perp$ and terminate.
     - Else if $\tilde{r} = \rho$, output the value y_i as a function of $\tau^{(\rho)}$ and terminate.
     - Else, go to the next step and if α_i is a prefix of $\tau^{(\tilde{r})}$ let $\alpha_i := \tau^{(\tilde{r})}$.
  3. For each $j \in [n]$, with $j \neq i$, and for each $r \leq \tilde{r}$, decrypt the ciphertexts $c_{j,i}^{(r)}$ and use the corresponding values $m_{j,i}^{(r)}$ to compute the messages $m_{i,j}^{(\tilde{r}+1)}$ to be sent at round $\tilde{r}+1$ (using the corresponding portion of the random tape ω_i).
  4. Finally, let $c_{i,j}^{(\tilde{r}+1)} \leftarrow\!\!\$\ \mathsf{Enc}(ek_j, m_{i,j}^{(\tilde{r}+1)})$ (using again random coins coming from ω_i) and $\sigma_{i,j}^{(\tilde{r}+1)} = \mathsf{Sign}(\mathsf{sk}_i, c_{i,j}^{(\tilde{r}+1)})$, and invoke $\mathsf{Broadcast}((c_{i,j}^{(\tilde{r}+1)}||\sigma_{i,j}^{(\tilde{r}+1)})_{j\in[n]\setminus\{i\}})$.

Figure 4.4. Generic compiler for obtaining blockchain-aided MPC with hasty players.

4.4.2 Security Analysis

The theorem below establishes the security of our generic compiler.

Theorem 7. *Let* $(\mathsf{Gen}, \mathsf{Enc}, \mathsf{Dec})$ *be a semantically secure PKE scheme, and* $(\mathsf{Gen}', \mathsf{Sign}, \mathsf{Verify})$ *be a (deterministic) unforgeable signature scheme. Furthermore, let* π *be an* n*-party* ρ*-round protocol that* t*-securely UC-realizes a functionality* f *with aborts in the presence of malicious adversaries. Then, the protocol* π^* *of Fig. 4.4* t*-securely computes* f *with aborts in the presence of hasty players and malicious adversaries.*

UC security is needed due to the fact that the attacker in the real world may interact with the blockchain by posting messages and reading its state. As shown in [CGJ19], such blockchain-active adversaries render standard simulation techniques (e.g., black-box rewinding) moot. Note also that Remark 3 does not hold for our protocol. If the adversary tries to furnish two different inputs in two different branches it can be spotted by some honest player, leading to an abort. Therefore only one possible input can be given to the functionality.

We need to show that for every PPT adversary A^*, there exists a PPT simulator Sim^* such that no non-uniform PPT distinguisher D^* can tell apart the experiments $\mathbf{REAL}^{\Gamma,0}_{\pi,\mathsf{A}^*,\mathsf{D}^*}(\lambda)$ and $\mathbf{IDEAL}^{\Gamma}_{f_\perp,\mathsf{Sim}^*,\mathsf{D}^*}(\lambda)$. In particular, the simulator Sim^* needs to simulate the interaction of the honest players with the blockchain protocol Γ as it happens in the real experiment. Intuitively, Sim^* relies on the simulator Sim guaranteed by the underlying protocol π as follows. At the beginning, Sim^* samples the public/secret keys for encryption/signatures for the honest players. Then, Sim^* runs A^* reading its messages from the emulated execution of the blockchain protocol Γ, and simulates its view as follows: (i) The round-r messages $m^{(r)}_{j,i}$ sent by the honest players P_j to the malicious players P_i are obtained from the simulator Sim; (ii) The round-r messages $m^{(r)}_{j,j'}$ that are exchanged by the honest players $\mathsf{P}_j, \mathsf{P}_{j'}$ are replaced with the all-zero string. Of course, Sim^* does additional bookkeeping in order to simulate a real execution of the protocol using the blockchain; in particular, Sim^* needs to check that the attacker plays consistently on different branches of a fork, and simulate an abort whenever the latter does not happen. Moreover, when Sim extracts the inputs for the malicious parties, the simulator Sim^* forwards the same inputs to the trusted party, obtains the outputs for the malicious parties, and sends it to Sim. Finally, Sim^* completes the simulation consistently with the choice of Sim of aborting or not.

Very roughly, the security of the PKE scheme and of the signature scheme imply that the view of the attacker is identical to that in a real execution of protocol π, so that security of π^* follows by that of π.

Proof. We begin by describing the simulator Sim^*. Let Sim be the PPT simulator guaranteed by the malicious security of π. Upon input the set of corrupted parties $\mathcal{I}$, inputs $(x_i)_{i\in\mathcal{I}}$, and auxiliary input z, the simulator Sim^* proceeds as follows:

1. Initialize Sim upon input $(\mathcal{I}, (x_i)_{i\in\mathcal{I}}, z)$, with uniformly chosen random tape $\omega_{\mathsf{sim}} \leftarrow\!\!\$\ \{0,1\}^*$.
2. For each $j \notin \mathcal{I}$, sample $(ek_j, dk_j) \leftarrow\!\!\$\ \mathsf{Gen}(1^\lambda)$, $(\mathsf{vk}_j, \mathsf{sk}_j) \leftarrow\!\!\$\ \mathsf{Gen}'(1^\lambda)$, $\omega_j \leftarrow\!\!\$\ \{0,1\}^*$, and invoke $\mathsf{Broadcast}(ek_j||\mathsf{vk}_j||j)$.

3. For each $j \notin \mathcal{I}$, keep running $\alpha_j \leftarrow\!\!\$\ \mathsf{UpdateState}(1^\lambda)$ and $\mathcal{B}_j \leftarrow\!\!\$\ \mathsf{GetRecords}(\alpha_j)$ until all the messages $(ek_i, \mathsf{vk}_i)_{i\in[n]} \in \mathcal{B}_j$. Set $\tau_j^{(0)} := (ek_i, \mathsf{vk}_i)_{i\in[n]}$ and $\alpha_j := \tau^{(0)}$.

4. For each $j \notin \mathcal{I}$, emulate the behavior of party P_j as follows:

   (a) Update the state α_j by running $\mathsf{UpdateState}(1^\lambda)$, and let $\mathcal{B}_j \leftarrow\!\!\$\ \mathsf{GetRecords}(\alpha_j)$.

   (b) Let $\tilde{r} \geq 0$ be the maximum value such that the partial transcript $\tau^{(\tilde{r})} \in \mathcal{B}_j$. Then:
   - If the ciphertexts in $\tau^{(\tilde{r})}$ are not consistent with those in α_j, send `abort` to the trusted party, simulate A^* aborting in the real protocol, and terminate.
   - Else, go to the next step and if α_j is a prefix of $\tau^{(\tilde{r})}$ let $\alpha_j := \tau^{(\tilde{r})}$.

   (c) Extract from $\tau_j^{(\tilde{r})}$ the ciphertexts $(c_{i,j}^{(\tilde{r})})_{i\in\mathcal{I}}$ and the signatures $(\sigma_{i,j}^{(\tilde{r})})_{i\in\mathcal{I}}$ that A^* (on behalf of each corrupted player P_i) forwards to P_j. If there exists $i \in \mathcal{I}$ such that $\mathsf{Verify}(\mathsf{vk}_i, \sigma_{i,j}^{(\tilde{r})}) = 0$, send `abort` to the trusted party, simulate A^* aborting in the real protocol, and terminate. Else, for each $r \leq \tilde{r}$, decrypt the ciphertexts $c_{i,j}^{(r)}$ using the decryption key dk_j, and pass the corresponding messages $((m_{i,j}^{(1)})_{i\in\mathcal{I},j\in\mathcal{H}}, \ldots, (m_{i,j}^{(\tilde{r})})_{i\in\mathcal{I},j\in\mathcal{H}})$ to Sim. Hence:
   - Upon receiving `abort` from Sim, send `abort` to the trusted party, simulate A^* aborting in the real protocol, and terminate.
   - Upon receiving $(x_i)_{i\in\mathcal{I}}$ from Sim, send $(x_i)_{i\in\mathcal{I}}$ to the trusted party, obtain the outputs $(y_i)_{i\in\mathcal{I}}$, and forward $(y_i)_{i\in\mathcal{I}}$ to Sim. In case Sim replies with $(\texttt{continue}, \mathcal{H}')$, send $(\texttt{continue}, \mathcal{H}')$ to the trusted party and terminate.
   - Upon receiving a set of messages $(m_{j,i}^{(\tilde{r}+1)})_{j\in\mathcal{H},i\in\mathcal{I}}$—corresponding to the simulated messages that each honest player P_j sends to the corrupted party P_i— for each $j \in \mathcal{H}$ and $i \in \mathcal{I}$ compute $c_{j,i}^{(\tilde{r}+1)} \leftarrow\!\!\$\ \mathsf{Enc}(ek_i,$ $, m_{j,i}^{(\tilde{r}+1)})$ (using coins from ω_j) and $\sigma_{j,i}^{(\tilde{r}+1)} = \mathsf{Sign}(\mathsf{sk}_j, c_{j,i}^{(\tilde{r}+1)})$. Then, for each $j, j' \in \mathcal{H}$, let $c_{j,j'}^{(\tilde{r}+1)} \leftarrow\!\!\$\ \mathsf{Enc}(ek_{j'}, 0^{|m_{j,j'}^{(\tilde{r}+1)}|})$ (using coins from ω_j) and $\sigma_{j,j'}^{(\tilde{r}+1)} \leftarrow\!\!\$\ \mathsf{Sign}(\mathsf{sk}_j, c_{j,j'}^{(\tilde{r}+1)})$, and finally invoke $\mathsf{Broadcast}((c_{j,i}^{(\tilde{r}+1)}||$ $\sigma_{j,i}^{(\tilde{r}+1)})_{i\in[n]\setminus\{j\}})$.

To conclude the proof, we consider a sequence of hybrid experiments (ending with the real experiment) and argue that each pair of hybrids is computationally close thanks to the properties of the underlying cryptographic primitives.

Hybrid $\mathbf{H}_3(\lambda)$: This experiment is identical to $\mathbf{IDEAL}^{\Gamma}_{f_\perp,\mathsf{Sim}^*,\mathsf{D}^*}(\lambda)$.

Hybrid $\mathbf{H}_2(\lambda)$: Identical to $\mathbf{H}_3(\lambda)$ except that we replace the ciphertexts $(c_{j,j'}^{(r)})_{j\in\mathcal{H},j'\in\mathcal{H}\setminus\{j\}}$ that each honest party P_j sends to the other honest players $\mathsf{P}_{j'}$ with an encryption of the real messages $(m_{j,j'}^{(r)})_{j\in\mathcal{H},j'\in\mathcal{H}\setminus\{j\}}$ that the same parties would send in a real execution of π. Note that the other ciphertexts $(c_{j,i}^{(r)})_{j\in\mathcal{H},i\in\mathcal{I}}$

are still emulated using the simulator, and the output of the experiment is determined by the trusted party.

The inputs for the honest parties are chosen to be the values $(x_i)_{i\in\mathcal{H}}$ chosen by the distinguisher D^* at the beginning of the experiment, and the random tape of each player is chosen uniformly once and for all as in the real world.

Hybrid $\mathbf{H}_1(\lambda)$: Identical to $\mathbf{H}_2(\lambda)$ except that we artificially abort if A^* modifies one of the ciphertexts $(c^{(r)}_{j,i})_{j\in\mathcal{H},i\in[n]\setminus\{j\}}$ corresponding to the messages that each honest player sends in a given round. Note that these ciphertexts correspond to both the real messages $(m^{(r)}_{j,j'})_{j\in\mathcal{H},j'\in\mathcal{H}\setminus\{j\}}$ and the simulated messages $(m^{(r)}_{j,i})_{j\in\mathcal{H},i\in\mathcal{I}}$.

Hybrid $\mathbf{H}_0(\lambda)$: This experiment is identical to $\mathbf{REAL}^{\Gamma,0}_{\pi^*,\mathsf{A}^*,\mathsf{D}^*}(\lambda)$.

Lemma 20. $\{\mathbf{H}_3(\lambda)\}_{\lambda\in\mathbb{N}} \approx_c \{\mathbf{H}_2(\lambda)\}_{\lambda\in\mathbb{N}}$.

Proof. We reduce to semantic security of $(\mathsf{Gen},\mathsf{Enc},\mathsf{Dec})$. Let $h = |\mathcal{H}|$. For $k \in [0,h]$, consider the hybrid experiment $\mathbf{H}_{3,k}(\lambda)$ in which the distribution of the ciphertexts $(c^{(r+1)}_{j,j'})_{j\in\mathcal{H},j'\in\mathcal{H}\setminus\{j\}}$ is modified as in $\mathbf{H}_2(\lambda)$ only for the first h honest parties. Clearly, $\{\mathbf{H}_{3,0}(\lambda)\}_{\lambda\in\mathbb{N}} \equiv \{\mathbf{H}_3(\lambda)\}_{\lambda\in\mathbb{N}}$ and $\{\mathbf{H}_{3,h}(\lambda)\}_{\lambda\in\mathbb{N}} \equiv \{\mathbf{H}_2(\lambda)\}_{\lambda\in\mathbb{N}}$.

Next, we prove that for every $k \in [0,h]$ it holds that $\{\mathbf{H}_{3,k}(\lambda)\}_{\lambda\in\mathbb{N}} \approx_c \{\mathbf{H}_{3,k+1}(\lambda)\}_{\lambda\in\mathbb{N}}$ which concludes the proof of the lemma. By contradiction, assume that there exists an index $k \in [0,h]$, and a pair of PPT algorithms $(\mathsf{D}^*,\mathsf{A}^*)$ such that D^* can distinguish the two experiments $\mathbf{H}_{3,k}(\lambda)$ and $\mathbf{H}_{3,k+1}(\lambda)$ with non-negligible probability. We construct a PPT attacker B breaking semantic security of $(\mathsf{Gen},\mathsf{Enc},\mathsf{Dec})$ as follows:

- Receive the target public encryption key ek^* from the challenger.
- Run D^*, receiving the set of corrupted parties $\mathcal{I}$, the inputs $(x_i)_{i\in[n]}$, and the auxiliary input z, then pass $(\mathcal{I},(x_i)_{i\in\mathcal{I}},z)$ to A^*.
- Interact with A^* as described in the ideal experiment, except that:
  - The public encryption key for player P_{k+1} is set to be the target public key ek^*.
  - For each $j \le k$, when it comes to simulating the ciphertexts $(c^{(r)}_{j,j'})_{j'\in\mathcal{H}\setminus\{j\}}$, use the real messages $(m^{(r)}_{j,j'})_{j'\in\mathcal{H}\setminus\{j\}}$, encrypt them using the public encryption key $ek_{j'}$ of $\mathsf{P}_{j'}$, and sign the ciphertexts with the secret key sk_j (which is known to the reduction).
  - When it comes to simulating the ciphertexts $(c^{(r)}_{k+1,j'})_{j'\in\mathcal{H}\setminus\{j\}}$, forward the pair of plaintexts $(m^{(r)}_{k+1,j'}, 0^{|m^{(r)}_{k+1,j'}|})_{j'\in\mathcal{H}\setminus\{k+1\}}$ to the left-or-right encryption oracle and sign the corresponding ciphertexts using the secret signing key sk_{k+1} of P_{k+1} (which is known to the reduction).
  - For each $j > k+1$, when it comes to simulating the ciphertexts $(c^{(r)}_{j,j'})_{j'\in\mathcal{H}\setminus\{j\}}$, use the dummy messages $(0^{|m^{(r)}_{j,j'}|})_{j'\in\mathcal{H}\setminus\{j\}}$, encrypt them using the public

encryption key $ek_{j'}$ of $\mathsf{P}_{j'}$, and sign the ciphertexts with the secret key sk_j (which is known to the reduction).

- Finally, run D^* upon the final output generated by A^*, and return whatever D^* outputs.

Note that the reduction can indeed simulate the interaction with the blockchain protocol Γ as in the ideal experiment, and moreover it can generate the real messages $(m^r_{j,j'})_{j\in\mathcal{H},j'\in\mathcal{H}\setminus\{j\}}$ as it knows the parties' inputs $(x_i)_{i\in[n]}$. By inspection, the simulation performed by B is perfect in the sense that when the challenger encrypts the messages $m^{(r)}_{k+1,j'}$ the view of $(\mathsf{D}^*,\mathsf{A}^*)$ is identical to that in $\mathbf{H}_{3,k+1}(\lambda)$. Similarly, when the challenger encrypts the dummy messages $0^{|m^{(r)}_{k+1,j'}|}$ the view of $(\mathsf{D}^*,\mathsf{A}^*)$ is identical to that in $\mathbf{H}_{3,k}(\lambda)$. Hence, B breaks semantic security of $(\mathsf{Gen},\mathsf{Enc},\mathsf{Dec})$ with non-negligible probability, concluding the proof. □

Lemma 21. $\{\mathbf{H}_2(\lambda)\}_{\lambda\in\mathbb{N}} \approx_c \{\mathbf{H}_1(\lambda)\}_{\lambda\in\mathbb{N}}$.

Proof. Let **BAD** be the event that an artificial abort happens in $\mathbf{H}_1(\lambda)$. Note that this means that, for some $j \in \mathcal{H}$, the attacker A^* replaces one of the ciphertexts $c^{(r)}_{j,i}$ that P_j would send to P_i in the real protocol with a different ciphertext $\tilde{c}^{(r)}_{j,i}$, in such a way that the corresponding signature $\tilde{\sigma}^{(r)}_{j,i}$ is still accepting. Clearly, the experiments $\mathbf{H}_2(\lambda)$ and $\mathbf{H}_1(\lambda)$ are identical conditioning on **BAD** not happening, and does it suffices to show that $\mathbb{P}[\mathbf{BAD}]$ is negligible.

Given a PPT distinguisher D^* and a PPT attacker A^* such that A^* provokes event **BAD** in a run of $\mathbf{H}_2(\lambda)$ with non-negligible probability, we can construct a PPT attacker B breaking security of the signature scheme $(\mathsf{Gen}',\mathsf{Sign},\mathsf{Verify})$. The reduction works as follows:

- Receive the target public verification key vk^* from the challenger.
- Choose a random j^* as a guess for the index corresponding to the honest party for which A^* provokes the bad event.
- Run D^*, receiving the set of corrupted parties $\mathcal{I}$, the inputs $(x_i)_{i\in[n]}$, and the auxiliary input z, then pass $(\mathcal{I},(x_i)_{i\in\mathcal{I}},z)$ to A^*.
- Interact with A^* as described in $\mathbf{H}_2(\lambda)$, except that:
  - The public verification key for player P_{j^*} is set to be the target public key vk^*.
  - When it comes to simulating the round-r messages from party P_{j^*}, generate the ciphertexts $(c^{(r)}_{j^*,i})_{i\in[n]\setminus\{j^*\}}$ as done in $\mathbf{H}_2(\lambda)$, and then forward each of $c^{(r)}_{j^*,i}$ to the challenger, obtaining the corresponding signature $\sigma^{(r)}_{j^*,i}$ that is needed in order to complete the simulation.
  - Keep updating the local state of P_{j^*} until an index $i \in [n]\setminus\{j^*\}$ is found such that the partial transcript α_{j^*} contains a pair $(\tilde{c}^{(r)}_{j^*,i},\tilde{\sigma}^{(r)}_{j^*,i})$ such that $\mathsf{Verify}(\mathsf{vk}_{j^*},\tilde{c}^{(r)}_{j^*,i},\tilde{\sigma}^{(r)}_{j^*,i}) = 1$ and $\tilde{c}^{(r)}_{j^*,i}$ is different from the original ciphertext $c^{(r)}_{j^*,i}$ previously sent on behalf of P_{j^*}.

- If no such pair is found, abort the simulation. Else, return $(\tilde{c}^{(r)}_{j^*,i}, \tilde{\sigma}^{(r)}_{j^*,i})$.

Note that the simulation performed by B is perfect, in that the view of $(\mathsf{D}^*, \mathsf{A}^*)$ is identical to that in a run of $\mathbf{H}_2(\lambda)$. Moreover, conditioning on B guessing the index j^* correctly, the reduction is successful in breaking security of the signature scheme exactly with probability at least $\mathbb{P}[\mathbf{BAD}]$, which is non-negligible. Since the former event also happens with non-negligible probability, this concludes the proof. □

Lemma 22. $\{\mathbf{H}_1(\lambda)\}_{\lambda\in\mathbb{N}} \approx_c \{\mathbf{H}_0(\lambda)\}_{\lambda\in\mathbb{N}}$.

Proof. The proof is by reduction to UC-security of the underlying protocol π. By contradiction, assume that there exists a PPT adversary A^* and a non-uniform PPT distinguisher D^* that can distinguish between $\mathbf{H}_1(\lambda)$ and $\mathbf{H}_0(\lambda)$ with non-negligible probability. Consider the following PPT attacker A, initialized with a set of corrupted parties $\mathcal{I}$, inputs $(x_i)_{i\in\mathcal{I}}$ for the malicious players, and auxiliary input $z = (z^*, (ek_i, dk_i)_{i\in[n]}, (\mathsf{vk}_i, \mathsf{sk}_i)_{i\in[n]})$ which will be specified later:

- Pass $(\mathcal{I}, (x_i)_{i\in\mathcal{I}}, z^*)$ to A^*.
- For each $i \in \mathcal{I}$, upon receiving the round-r messages $(m^{(r)}_{j,i})_{j\in\mathcal{H}}$ from the honest players to the malicious players, let $c^{(r)}_{j,i} \leftarrow\!\!\$\ \mathsf{Enc}(ek_i, m^{(r)}_{j,i})$ and $\sigma^{(r)}_{j,i} = \mathsf{Sign}(\mathsf{sk}_j, c^{(r)}_{j,i})$, and emulate broadcasting $(c^{(r)}_{j,i}, \sigma^{(r)}_{j,i})_{j\in\mathcal{H},i\in\mathcal{I}}$ via the blockchain protocol.
- For each $j \in \mathcal{H}$, upon receiving the round-r messages $(m^{(r)}_{i,j})_{i\in\mathcal{I}}$ that A^* wants to send to the honest parties, let $c^{(r)}_{i,j} \leftarrow\!\!\$\ \mathsf{Enc}(ek_j, m^{(r)}_{i,j})$ and $\sigma^{(r)}_{i,j} = \mathsf{Sign}(\mathsf{sk}_i, c^{(r)}_{i,j})$, and emulate broadcasting $(c^{(r)}_{i,j}, \sigma^{(r)}_{i,j})_{i\in\mathcal{I},j\in\mathcal{H}}$ via the blockchain protocol.
- For each $j \in \mathcal{H}$, compute the messages $(m^{(r)}_{j,j'})_{j'\in\mathcal{H}\setminus\{j\}}$ exchanged between honest parties as done in $\mathbf{H}_0$ (which is the same in $\mathbf{H}_1(\lambda)$), let $c^{(r)}_{j,j'} \leftarrow\!\!\$\ \mathsf{Enc}(ek_{j'}, m^{(r)}_{j,j'})$ and $\sigma^{(r)}_{j,j'} = \mathsf{Sign}(\mathsf{sk}_j, c^{(r)}_{j,j'})$, and emulate broadcasting $(c^{(r)}_{j,j'}, \sigma^{(r)}_{j,j'})_{j\in\mathcal{H},j'\in\mathcal{H}\setminus\{j\}}$ via the blockchain protocol.
- In case a fork appears during the simulation of the underlying blockchain protocol, replicate the messages from the honest players as done in the other branches (using exactly the same randomness). On the other hand, if the messages from A^* differ from those sent on the simulation of a previous branch, simulate A^* aborting and terminate.
- Output whatever A^* outputs.

Additionally, let Z be the following PPT distinguisher:

- Run D^*, receiving the set of corrupted parties $\mathcal{I}$, the inputs $(x_i)_{i\in[n]}$, and the auxiliary input z^*, then sample (ek_i, dk_i) and $(\mathsf{vk}_i, \mathsf{sk}_i)$ for all $i \in [n]$, and pass $(\mathcal{I}, (x_i)_{i\in\mathcal{I}}, z)$ to the above defined attacker A, where $z = (z^*, (ek_i, dk_i)_{i\in[n]}, (\mathsf{vk}_i, \mathsf{sk}_i)_{i\in[n]})$.

- Upon receiving the final output from A, pass it to D^* and output whatever D^* outputs.

By inspection, in case the attacker A is playing in a real execution of protocol π, the view of D^* is identical to that in an execution of $\mathbf{H}_0(\lambda)$ with A^* controlling the malicious parties. Similarly, in case the view of A is emulated using the simulator Sim (corrupting the dummy parties controlled by A) of protocol π, the view of D^* is identical to that in an execution of $\mathbf{H}_1(\lambda)$ with A^* controlling the malicious parties. It follows that Z can distinguish between $\mathbf{REAL}_{\pi,\mathsf{A},\mathsf{Z}}(\lambda)$ and $\mathbf{IDEAL}_{f_\perp,\mathsf{Sim},\mathsf{Z}}(\lambda)$ with non-negligible probability, a contradiction. □

The theorem now follows directly by combining the above lemmas. □

4.4.3 Extending to Fairness with Penalties

In their work, Andrychowicz et al. [ADMM14, ADMM16] proposed a different notion of fairness for MPC protocols that run on blockchain: fairness with penalties. This notion states that if an adversary in an MPC protocol decides to abort the execution of the protocol it will be financially penalized. To obtain the penalization in the lottery protocol, Andrychowicz et al. added a deposit step in the protocol.

W discuss here how to obtain fairness with penalties following in part the outline of [KB14, BK14].

Let's now assume the existence of an (n, n)-secret sharing scheme $(\mathsf{Share}, \mathsf{Recon})$, non-interactive commitment schemes, and consider a functionality f' that first calculates $y \leftarrow f(x_1, \ldots, x_n)$, where each party P_i holds x_i, and then calculates the shares of the output $(\sigma_1, \ldots, \sigma_n) \leftarrow\!\!\$\ \mathsf{Share}(y)$, the commitments $C = (\gamma_1, \ldots, \gamma_n)$, where $\gamma_i \leftarrow\!\!\$\ \mathsf{Commit}(\sigma_i)$, and outputs (C, σ_i) to each player P_i. Let's call π' the protocol realizing f', we can apply our generic compiler to π' to obtain a protocol π'_{bc} that can be run in the blockchain. Our protocol π_{fair}, running with players $\mathsf{P}_1 \ldots, \mathsf{P}_n$ works as follows

(i) *Protocol execution*: All the players engage in the protocol π'_{bc}. A party P_i aborts the execution if π'_{bc} aborts. Otherwise, obtains (C, σ_i) in the last round.

(ii) *Smart contract*: P_1 publishes the smart contract depicted in Fig. 4.5.

(iii) *Commitment phase*: For each $i \in [n]$, P_i triggers $d(C_i)$ together with d coins, where d is a fixed deposit. If some player does not publish his commitments with the deposit or there is a disagreement on the commitments within `time1` (i.e., a player P_j sends $C_j \neq C_i$ for some $P_{i \neq j}$, or deposits a value $d_i < d$, P_i abort the execution. Recall that abort in this phase is still fine, since no information about the output y is released. Otherwise, if `time1` has passed, go to the Opening Phase.

(iv) *Opening phase*: For each $i \in [n]$, P_i opens his commitment by sending `openCom`(i, σ_i), thus receiving back his d coins, wait that all the openings are published in the smart contract (within `time2`) and calculates $y \leftarrow \mathsf{Recon}(\sigma_1, \ldots, \sigma_n)$. If, after `time2`, some share is missing, P_i aborts the execution.

During the last phase, if some player did not open the commitment or sent an incorrect value, the smart contract will penalize him by freezing his deposit. Thus, the adversary is not incentivized to send an incorrect share.

This attempt to add fairness with penalties, however, introduces an attack. Given an n-party protocol $\pi_{f'}^{\Gamma}$ obtained by the compiler described in Section **??** applied to $\pi_{f'}$, with the addition of the smart contract, commit and opening phases described above, we have the following scenario:

- For all $i \in [n]$, party P_i runs π_{fair} obtaining (C, σ_i).
- For all $i \in [n]$, party P_i triggers $d(C)$ together with d coins to the smart contract.
- For all $i \in [n-1]$, party P_i opens his commitment by triggering $\texttt{openCom}(\sigma_i)$.
- Wlog., we say that P_n is an adversary. P_n compute the output y. If P_n does not like y in the current branch, P_n can try to exploit a fork happening during the execution of π_{fair} to change the in a different branch to obtain a new couple (C', σ'_n).
- The honest parties $\mathsf{P}_1, \ldots, \mathsf{P}_{n-1}$ notice that there is a message published by P_n that differs from the value previously received always by P_n. Since the transcript obtained from the blockchain differs from the transcript stored in their local state, they will abort.

The protocol described is not fair, since we can construct a counterexample that prove that the unfair party P_n can obtain the output without being penalized.

$\mathsf{P}_1, \ldots, \mathsf{P}_n$ will play $\pi_{f'}^{\Gamma}$ to obtain (C, σ_i). As the smart contract is published, P_n will trigger $d(C)$ together with d coins. At this point, P_n waits that all P_i, with $i \in [n]$ publish the opening σ_i.

When P_n sees $\sigma_1, \ldots, \sigma_{n-1}$ on the blockchain, computes the output. If P_n dislikes the output, then he tries to exploit a branch created before the end of the execution of π_{fair} to change messages in that branch to obtain an advantage. Since P_n publishes different messages on different branches of the blockchain, there exist some party P_i, with $i \in [n]$ that will notice it, causing an abort in the protocol.

Let's call b_1 the branch where P_n learned the output and b_2 the branch exploited to change the execution of π_{fair}. We have two cases:

- If b_1 is the branch that will be confirmed on the blockchain, P_n will be penalized.
- If b_2 is the branch that will be confirmed on the blockchain, P_n will cause an abort in the protocol before that the commitment phase starts. In this case he does not get penalized for learning the output.

With this counterexample we show that the proposed solution is not enough to obtain fairness with penalties, since P_n has the possibility to learn the output without incurring in any punishment. It is possible to obtain fairness with penalties with our general compiler, and waiting that the commitment phase is confirmed on

The **General Compiler Smart Contract** runs with players $\mathsf{P}_1, \ldots, \mathsf{P}_n$ and consists of two main functions d and `openCom` and two fixed timestamps `time1`,`time2`.

Commitment Phase: In round t_1, when $d(C_i)$ together with d coins is triggered from a party P_i, store (i, C_i). Then, if $\forall i, j : C_i = C_j$ proceed to the Opening Phase. Otherwise, for all i, if the message (i, C_i) has been stored, send message d coins to P_i and terminate.

Opening Phase: In round t_2, when `openCom`(i, σ_i) is triggered from P_i, check if $\mathsf{Commit}(\sigma_i) = \gamma_i$, where γ_i is obtained from parsing $C_i = (\gamma_1, \ldots, \gamma_n)$ (recall that all the C_i are the same), and send d coins back to P_i.

Figure 4.5. General compiler smart contract.

the ledger. Since in this case the commitment phase is finalized, an adversary A cannot learn y unless she decides to lose the d coins deposited in the commitment phase. Since the commitment phase is confirmed on the ledger, A cannot find a fork to exploit the execution of the protocol on another branch. Yet, A can cause an abort in the protocol, but if it happens before the commitment phase she will not learn the output y. If the abort happens after the commitment phase, A will learn the output but will be penalized.

Theorem 8. *Let's assume the existence of non-interactive commitment schemes and (n, n)-secret sharing schemes. Let π'_{bc} be an n-party ρ-round protocol realizing f' in the presence of hasty players. Then, the protocol π_{fair} described above securely realizes f satisfying fairness with penalties in the presence of $(\rho, 1)$-hasty players.*

Proof Sketch. We can claim security of the compiled protocol π'_{bc} obtained by applying the general compiler to π', by referring to the same proof of Theorem. 7. Now, we argue that the overall protocol π_{fair} achieves fairness with penalties. As mentioned before, aborts during the execution of π'_{bc} are acceptable, since the adversary cannot learn any information about the output. After the committing phase, that is finalized, the adversary could try to exploit different branches to send different openings of his commitments. We have the following time-line: The execution started in a branch b_1 and a forks happens right after the committing phase, generating a branch b_2. Wlog. of generality we can extend this argument to multiple parallel executions in different branches. We have the following scenarios:

Scenario 1: A corrupted player abort in both branches. Since the commitments are finalized, fairness with penalties follows in a straightforward manner, since he did not open his commitments in each branch, and so also in the confirmed one.

Scenario 2: A corrupted player opens his share in b_1 and aborts the execution in b_2 (after the commitment phase). If b_1 gets confirmed, the honest parties will

learn the output. If b_2 gets confirmed, it automatically boils down to Scenario 1.

Scenario 3: A corrupted player P_i opens his share in b_1 and tries to open on a different share in b_2. Since the commitment is always confirmed, the adversary cannot try to change his commitment by exploiting forks. If A is able to open the commitment by providing two different shares, then we can define an adversary $\mathsf{A}_{\mathsf{com}}$ breaking the binding property of the underlying commitment scheme with non-negligible probability. That means that at least in one of the two branches P_i gets penalized, and if he provides the correct opening in one of the branches and it gets confirmed, honest players will learn the output.

□

4.4.4 Efficiency Analysys

Given an n-party ρ-round UC-secure MPC protocol π, we evaluate the efficiency of a protocol π^* obtained compiling π with the compiler described in Fig. 4.4. Our compiler adds only one round to the execution of the protocol, in which the parties publish their encryption keys of the underlying encryption scheme and signature keys of the signature scheme. We analyze the number of block needed to end π^* in case of standard MPC with aborts and in case of fairness with penalties. As we noted in Section 4.4.3, to obtain fairness with penalties in π^*, the players of the protocol have to wait that the first round (the deposit) became final on the blockchain before continuing to run the protocol with a hasty execution.

Let us consider the case of MPC with aborts first. Since in this setting the protocol can be run a full-hasty mode, if the underlying protocol π has ρ rounds, then π^* will have $\rho+1$ rounds and the number of blocks needed to end the computation is $\rho+1$.

In case of fairness with penalties, the execution time will be $\rho+w$ blocks, where $w \geq k$ is the liveness parameter. Since we assume that in the ideal conditions all the players broadcast the deposit message at the same time, their deposit will be posted and confirmed after at most w blocks.

4.5 Smart Contracts

We describe below our smart contract implementations in details.

Lottery protocol by [ADMM16]. The smart contract execution is described as follows:

- *Setup phase:* A player publishes the smart contract in Fig. 4.7 on page 95, indicating in the constructor the addresses of the players' wallet, and the committing phase timeout `time1` and opening phase timeout `time2`.
- *Committing phase:* The players trigger the `commit` function upon input the commitment to some value and a deposit of `minDep` $= n(n-1)+1$, where $n(n-1)$ coins are used for the penalty mechanism and 1 coin is used to put

money for the lottery. After `time1` blocks, the n commitments are collected and the committing phase ends.

- *Opening phase:* All the participants open their commitment by triggering the `openCom` function, and the winner can then claim his bet by triggering the `claimWinner` function of the smart contract (if all the parties have opened).

- *Compensation phase:* If, after `time2` blocks, some player did not open his commitment, the function `payDeposit` can be triggered, so that all player that hasn't open before `time2` will be penalized and the players who had opened receive their deposit back together with a fraction of the adversaries' deposit.

Our coin-tossing protocol. The description of the smart contract execution of π^*_{pct} on Fig. 4.6 works as follows:

- *Setup phase:* At the beginning, one of the players creates the smart contract specifying a deposit amount `minDep` and timeout `time1`. When the contract is posted on the blockchain, the constructor automatically generates a unique session identifier *sid* by triggering `generateSid`.

- *Deposit phase:* For each $i \in [n]$, P_i can decide to participate to the PCT protocol by triggering the function `depositFunc` to send a safety deposit and his public key pk_i for an unique signature scheme. After that $(\mathsf{pk}_1, \ldots, \mathsf{pk}_n)$ are collected by the smart contract and `time1` blocks are passed, the deposit phase ends parties can start to redeem their deposit. [8]

- *Claim phase:* During this phase, each player P_i can claim his deposit back by triggering the function `claim` of the smart contract and sending a value y_i such that $\mathsf{Verify}(\mathsf{pk}_i, x, y_i) = 1$, where $x = \mathsf{pk}_1 || \cdots || \mathsf{pk}_n || sid || bid$, pk_i is the public key of P_i. For the signature verification, we can use a unique signature scheme with fast verification like BLS Signatures [BLS01], invoked with `BLSVerify` in the code, or RSA-FDH [BR96].[9]

[8] Note that when some party P_i sends his deposit to the smart contract, the variable *bid* is updated with the hash of the last block of the contract state (uniquely identifying the branch in which the smart contract state is updated). This implies that *bid* is fixed after the last player sends his public key.

[9] We do not explicitly implement this signature, but solidity implementations are available for testing [Fic].

```
pragma solidity ^0.4.0;

contract ParallelCoinTossing {
  struct Player {
    bool isPlaying;
    bool hasClaimed;
    string pk;
    uint d; //Player's deposit
    uint c; //Player's claim
    }
  address[] playersAddr;
  mapping(address => Player) players;
  uint sid, time1, time2;
  bytes32 bid;

  //flags
  bool claimPhase = false; //true if the claimPhase starts

  //common message to be signed
  uint x;

  constructor(uint _time1, uint _time2) public  {
    sid = generateSid(); //session id
    time1 = _time1; //first timelock
    time2 = _time2; //second timelock
  }
  function deposit(string pubKey) public payable {
    require (msg.sender.balance >= minDep && msg.value >= minDep && players[msg.sender].d == 0 && now < time1
        );
    playersAddr.push(msg.sender); //add the public key of the current sender
    Player p = players[msg.sender];
    p.isPlaying = true;
    p.pk = pubKey;
    p.d = msg.value;  //msg.value is the deposit value of the player
    bid = block.blockhash(now); //Every time he receives a public key, it updates the blockhash, so that the
        correct bid is the blockchain state of the last public key deposited.
    }
  function claim(uint y) public {
    require (claimPhase && now < time2 && players[msg.sender].isPlaying && !players[msg.sender].hasClaimed &&
        BLSVerify(players[msg.sender].pk,x,y));
    Player p = players[msg.sender];
    p.c = y;
    p.hasClaimed = true;
    msg.sender.transfer(p.d);
}

  //automatic check functions run after a certain time
  function checkDeposit() public {
    require (!claimPhase && now >= time1);
    uint n = playersAddr.length;
    x = sha3(player[playersAddr[0]].pk||...||players[playersAddr[n-1]].pk||sid||bid);
    claimPhase = true;
    }
}
```

Figure 4.6. Pseudocode implementation of our smart contract for realizing parallel coin tossing. For simplicity, we omit an explicit definition of the concatenation function in the computation of `x`.

```
pragma solidity >=0.4.21 <0.6.0;

contract FairLottery {
  struct Player {
    address addr;
    bool hasCommitted, hasOpened, isPlaying;
    uint balance, index;
    bytes32 com;
    int opn;
    }
  uint public n, time1, time2;
  address[] addresses;
  mapping (address => Player) players;

constructor(address[] _addresses, uint _time1, uint _time2) public { //creates a new instance of the lottery
      for a set of prescribed players
  addresses = _addresses;
  for (uint i = 0; i < addresses.length; i++) {
    Player p = players[addresses[i]];
    p.isPlaying = true;
    p.index = i;
    }
  n = addresses.length;
  time1 = _time1;
  time2 = _time2;
  }
  function commit(bytes32 _com) public payable { //sha3 value commit (n*(n-1) coins + 1 coin for bet)
    require(msg.value >= (n*(n-1)+1)) &&   players[msg.sender].isPlaying && !players[msg.sender].hasCommitted
        );
    Player p = players[msg.sender];
    p.com = _com;
    p.hasCommitted = true;
    p.balance = msg.value;
  }
  function openCom(int openVar) public  { //opening of the commitment
    require(players[msg.sender].hasCommitted && now > time1 && now < time2 && !players[msg.sender].hasOpened
        && sha3(openVar) == players[msg.sender].com);
    Player p = players[msg.sender];
    p.hasOpened = true;
    msg.sender.transfer(n*(n-1)); //pays the sender back
  }
  function payDeposit() public { //compensation function
    require(players[msg.sender].isPlaying && now >= time2);
    uint index = players[msg.sender].index;
      for (uint i = 0; i < n; i++)
        if (i != index && !players[addresses[i]].hasOpened) msg.sender.transfer(players[msg.sender].balance/n
            ); //the player msg.sender get is compensation of n coins
  }
  function claimWinner(uint[] secrets) public { //function triggered by the winner
    require (secrets.length == n && checkWinner(secrets,msg.sender));
    msg.sender.transfer(n);  //redeem the won coins
}
//private local functions
  function checkWinner(uint[] secrets, address _sender) private returns (bool) {
    int sum = 0;
      for (uint i = 0; i < secrets.length; i++) {
        if (sha3(secrets[i] != players[addresses[i]].com) return false;
        sum += secrets[i];
      }
    if ((sum%n != players[_sender].index))
      return false;
    return true;
    }
  }
```

Figure 4.7. Pseudocode implementation of the lottery protocol by Andrychowicz et al. [ADMM16], when using smart contracts.

Chapter 5

Financial Fairness in Blockchain-Aided MPC

5.1 Security and efficiency

To compare protocols in terms of security, we refer to the security definitions given in 2.7.

5.1.1 On-chain and Off-chain Efficiency

The efficiency of a penalty protocol can be broken down into two parts: off-chain and on-chain efficiency. The former refers to traditional MPC efficiency in terms of: the number of communication rounds, the required bandwidth, and the computational complexity; the latter refers to efficiency in terms of the interaction between the blockchain and the miners in terms of: the number of transactions, the number of round executed on-chain, and the script complexity.

On-chain efficiency in a penalty protocol is much more important compared to off-chain efficiency, as:

- The number of transactions determine the transaction fee that a penalty protocol incurs;
- The number of rounds executed on-chain determine how long the protocol runs, as a round executed on-chain requires for a transaction to be confirmed which corresponds to, *e.g.*, 6 blocks (*i.e.,* 1 hour) in Bitcoin;
- The script complexity needs to be multiplied with the number of miners in the network, which could be more than 100K.[1]
- off-chain complexity is not dependent of blockchain's block generation rate and transaction throughput

Transaction, round and script complexity can asymptotically depend on the security parameter λ, the number of players n, the size of the output of the function $m = |f|$, and the number of stages (for multistage protocols).

[1] As of Dec 2020, the Bitcoin mining pool called Slushpool (`https://slushpool.com/stats/?c=btc`) has 116157 active miners.

5.2 Financial Fairness

5.2.1 Economics Principles

To capture financial fairness, economists introduced the concept of *net present value* and *discount rate*. The former tells us how much an amount of money received (or paid) later (at time r) is discounted w.r.t. the same amount of money received (or paid) now (at time $r = 0$). The difference in value between two adjacent instances is captured by the discount rate.[2]

The Cost of Participation. Let $\eta_i(r)$ be the function representing the net present value at the beginning of the protocol (*i.e.*, at time 0) of a unit coin that is transacted at a later round[3] (*i.e.*, at round r), according to the i-th party's own discount rate. Let $d_{i,r}$ be the coins put into escrow by player i during round r, and let $q_{i,r}$ be the coins that the same player receives at round r (possibly including compensating penalties extracted from misbehaving parties). Given a sequence of deposits $d_{i,r}$ and refunds $q_{i,r}$ made by P_i at rounds $r \in [0, \tau]$ of a protocol running up to time τ, the *net present cost of participation* for P_i is then

$$\chi_i(\tau) := -\sum\nolimits_{r\in[0,\tau]}(-d_{i,r} + q_{i,r}) \cdot \eta_i(r).t \tag{5.1}$$

ADDED MINUS SIGN at the beginning of Eq. 1 to make the cost POSITIVE.
The intuition behind the net present value calculated using Eq. (5.1) is that money received at a later round t' is *less valued* than the money received at the current round t. A real world analogy would be the money that is needed to buy a property, e.g a car or a house, now, is always not enough, to buy the very same property in 2 years, e.g. due to inflation.

Let us consider a toy example as follows. Supposed the discount rate is 50% (hourly rate), at the beginning of the protocol, a party deposit 100$ into the blockchain, after 1 hour, the party withdraws 50$, and after 2 hours, the party withdraw the rest 50$. The net present value for the party in this case, would be

1. +50/(1+0.5) = 33.3$ (first withdrawal)
2. +50/[(1+0.5)*(1+0.5)] = 22.2$ (second withdrawal)

thus 55.5$. The cost of participation will be thus

1. -(-100)$ (deposit)
2. -(+55.5)$ (net present value at the end)

= 44.5$.

The Payment Interest. The basic fixed interest rate model (used for home mortgages) is sufficient to show the marked financial unfairness of some protocols. Tab. 5.1 reports the December 2019 rates used by US depository institutions, measured in *basis points* (bps, 1/100th of 1%). Those are the rates at which depository institutions (*e.g.* commercial banks) can deposit money in each other (in the US) to

[2] In general, the discount rate may depend on the risk aversions of the players [LVM16], or the confidence in the certainty of future payments [BIW15]. The net present value may also have different functional forms (*e.g.* exponential, hyperbolic, etc.) or different values for borrowing or receiving money [ALR+01].

[3] For simplicity, we think of a round as a single time unit.

Table 5.1. US depository inst. rates in 2019 (per annum).

Fund	Min	Max	Median	Median Hour	Median Minute
EFFR	245	155	238	0.0272	0.0005
SOFR	525	152	238	0.0272	0.0005

A basis point, bps, is 1/100th of 1%. It is the standard unit of measure for interest rates at which depository institutions can deposit money in each other to adjust their capital requirements.

adjust their capital requirements. A quantity of money paid or received by a party P_i after a time r is cumulatively discounted at a constant discount rate δ, *i.e.* $\eta_i(r)$ can be computed as follows.

Conversion of Interest Rate
To convert the Overnight Rate δ_{d} (per annum) to Hourly Rate δ_{h} and Minute Rate δ_{m}, and using those to compute the corresponding payment interest, one needs to follow several intermediate steps:

1. Convert the Overnight Rate δ_{d} into *continuous* time, *i.e.* $\delta = \ln(1 + \delta_{\mathsf{d}})$ (where $\ln(\cdot)$ is the natural logarithm); using the NYFed's Secured Overnight Financing Overnight Rate $\delta_{\mathsf{d}} = 238 = 2.38\% = 0.0238$, $\delta := \ln(1 + 0.0238) = 0.0235$;
2. Multiply the continuous rate by $\frac{1}{365 \cdot 24}$ for the Hourly Rate δ_{h} or $\frac{1}{365 \cdot 24 \cdot 60}$ for the Minute Rate δ_{m}; then convert back to *discrete* time by taking $e^{r\delta}$ to obtain the payment interest factor,
3. Hence $\eta_i(r) := e^{-r\delta_{\mathsf{h}}}$ if we are using the Hourly Rate
4. (or $\eta_i(r) := e^{-r\delta_{\mathsf{m}}}$ if we are using the Minute Rate);

Notice that we deliberately do not consider the value that parties might give to protocol's outputs (*i.e.*, obtaining the output may be significantly more valuable to party i than party j). This issue is definitely relevant from the viewpoint of *protocol participants* to decide whether the whole MPC hassle (with or without penalties) is worth the bother. However, we believe that the outcome's valuation should be at least fair from the viewpoint of a *protocol designer*: all parties being equal the construction should be fair for them all, and they should not be discriminated by going first, last, or third.[4]

5.2.2 The Escrow Functionality

Our functionality $\mathcal{F}^*_{\mathsf{escrow}}$ (Fig. 5.1) is meant to capture any n-party protocol in the hybrid model with a so-called *escrow* ideal functionality in which: (i) Each player P_i can deposit a certain number of coins in the escrow (possibly multiple times); (ii) At some point the functionality might pay P_i with some coins from the escrow. In concrete instantiations, case (ii) can happen either because P_i claims back a previous deposit, or as a refund corresponding to some event triggered by another party (*e.g.*, in case of aborts).

Fix an execution of any protocol π in the $\mathcal{F}^*_{\mathsf{escrow}}$-hybrid model. For each message $(\mathtt{deposit}, \mathsf{coins}(d_{i,r}), *)$ sent by P_i, we add an entry $(r, d_{i,r})$ into an array $\mathcal{D}_i$ to

[4] In a formal model this could be simply achieved using an utility function so that instead of $(-d_{i,r} + q_{i,r}) \cdot \eta_i(r)$ we have $\mathcal{U}((-d_{i,r} + q_{i,r}) \cdot \eta_i(r))$.

The **Escrow Functionality** $\mathcal{F}^*_{\mathsf{escrow}}$ runs with security parameter 1^λ, parties $\mathsf{P}_1, \ldots, \mathsf{P}_n$, and adversary Sim corrupting a subset of parties. Its behavior is unspecified, except for the following:

- Upon input $(\texttt{deposit}, sid, ssid, \mathsf{coins}(d_{i,r}), *)$ from an honest party P_i at round r, record $(\texttt{deposit}, sid, ssid, i, r, d_{i,r}, *)$ and send it to all parties.
- During round r the functionality might send $(\texttt{refund}, sid, ssid, r, \mathsf{coins}(q_{i,r}), *)$ to party P_i and $(\texttt{refund}, sid, ssid, r, q_{i,r}, *)$ to all parties.
- The functionality is not allowed to create coins, *i.e.* at any round r the following invariant is maintained:

$$\sum\nolimits_{i\in[n],r'\leq r} q_{i,r'} \leq \sum\nolimits_{i\in[n],r'\leq r} d_{i,r'}$$

Figure 5.1. The family of escrow functionalities.

book keep all deposits P_i made. For the commands $(\texttt{refund}, r, q_{i,r}, *)$ received by P_i, we maintain an array $\mathcal{R}_i$ of entries $(r, q_{i,r})$ keeping track of all claims/refunds P_i received.

Deposits/refunds can appear in an arbitrary order; we only keep track of the round in which those are made. Apart from these commands, and the impossibility of creating money from nothing, the behavior of $\mathcal{F}^*_{\mathsf{escrow}}$ is unspecified. As we argue later, common ideal functionalities used in cryptographically fair MPC with penalties are of the escrow type.

The functionality $\mathcal{F}^*_{\mathsf{escrow}}$ captures inter-temporal economic choices (*i.e.* a party can abort or continue the protocol), and formalizes a notion of fairness grounded in the economic literature. The experimental evidence about inter-temporal economic choices [BRY89, AW97, BIW15, LVM16] is that money paid or received "now" has a greater value than the same amount of money received or paid "later". At the end all parties could still be made whole, but whoever was forced by the protocol to pay into escrow at *noticeably different times*, or held deposits of *noticeably different sizes*, would clearly call unfair play.[5]

5.2.3 Financial Fairness

Financial fairness then says that, even in a run of π with possibly corrupted parties, the net present cost of participation χ_i associated to each *honest player* is the same. Here, we make no assumption on η_i, but one may limit fairness to specific, empirical, forms of η_i (*e.g.*, known to hold for poker players).

Definition 25 (Financial fairness)**.** Consider an n-party protocol π that is cryptographically fair with penalties in the $\mathcal{F}^*_{\mathsf{escrow}}$-hybrid model, and let $(\mathcal{D}_i, \mathcal{R}_i)_{i\in[n]}$ be as described above. We say that π is *financially fair* if for every possible discount rate function $\eta(r) \in [0, 1]$, for all transcripts resulting from an arbitrary execution

[5] One could argue that these deposits are comparable to security deposits, as those used in the U.S. for interest-bearing accounts, with the interest accrued to the depositor's benefit. That is not true for the deposits used in penalty protocols based on cryptocurrencies: once the deposits are locked, they cannot be used, and therefore no interest is accrued to the depositor.

of π (with possibly corrupted parties), and for all $i, j \in [n]$ such that P_i and P_j are honest, it holds that $\chi_i = \chi_j$ where the *net present cost of participation* χ_i is defined in Eq. (5.1).

Def. 25 could be weakened by considering specific discount rates $\eta(r)$ (*e.g.* financial fairness with hyperbolic discount).

5.2.4 Instances of Escrow

Two instances of $\mathcal{F}^*_{\mathsf{escrow}}$ are commonly used for designing state-of-the-art penalty protocols: the Claim-Or-Refund functionality $\mathcal{F}^*_{\mathsf{CR}}$ and the Multi-Lock functionality $\mathcal{F}^*_{ML}$. Both functionalities can be implemented using both the Bitcoin network and Ethereum smart contracts.

$\mathcal{F}^*_{\mathsf{CR}}$ [BK14], allows a sender P_i to *conditionally* send coins to a receiver P_j, where the condition is formalized as a circuit $\phi_{i,j}$ with time-lock τ: P_j can obtain P_i's deposit by providing a satisfying assignment w within time τ, otherwise P_i can have his deposit refunded at time $\tau + 1$.

The **Claim-or-Refund Functionality** $\mathcal{F}^*_{\mathsf{CR}}$ runs with security parameter 1^λ, parties $\mathsf{P}_1, \ldots, \mathsf{P}_n$, and ideal adversary Sim.

Deposit Phase: Upon receiving the tuple $(\texttt{deposit}, sid, ssid, i, j, \phi_{i,j}, \tau, \mathsf{coins}(d))$ from P_i, record the message $(\texttt{deposit}, sid, ssid, i, j, \phi_{i,j}, \tau, d)$ and send it to all parties. Ignore any future deposit messages from P_i to P_j.

Claim Phase: After round τ, upon receiving $(\texttt{claim}, sid, ssid, i, j, \phi_{i,j}, \tau, d, w)$ from P_j, check if: (1) a tuple $(\texttt{deposit}, sid, ssid, i, j, \phi_{i,j}, \tau, d)$ was recorded, and (2) if $\phi_{i,j}(w) = 1$. If both checks pass, send $(\texttt{claim}, sid, ssid, i, j, \phi_{i,j}, \tau, d, w)$ to all parties, send $(\texttt{claim}, sid, ssid, i, j, \phi_{i,j}, \tau, \mathsf{coins}(d))$ to P_j, and delete the record $(\texttt{deposit}, sid, ssid, i, j, \phi_{i,j}, \tau, d)$.

Refund Phase: In round $\tau + 1$, if the record $(\texttt{deposit}, sid, ssid, i, j, \phi_{i,j}, \tau, d)$ was not deleted, then send $(\texttt{refund}, sid, ssid, i, j, \phi_{i,j}, \tau, \mathsf{coins}(d))$ to P_i, and delete the record $(\texttt{deposit}, sid, ssid, i, j, \phi_{i,j}, \tau, d)$.

$\mathcal{F}^*_{ML}$ [KB14] allows n parties to atomically agree on a timeout τ, circuits $\phi_1, \ldots, \phi_n$, and a deposit d. Hence, if P_i within round τ reveals to everyone a valid witness w_i for ϕ_i, it can claim its deposit back; otherwise, at round $\tau + 1$, the deposit of P_i is split among all other players.

The **Multi-Lock Functionality** $\mathcal{F}^*_{ML}$ runs with security parameter 1^λ, parties $\mathsf{P}_1, \ldots, \mathsf{P}_n$, and adversary Sim.

Lock Phase: Wait to receive $(\mathsf{lock}, i, D_i = (d, sid, ssid, \phi_1, \ldots, \phi_n, \tau), \mathsf{coins}(d))$ from each P_i and record $(\mathsf{locked}, sid, ssid, i, D_i)$. Then, if $\forall i, j : D_i = D_j$ send message $(\mathsf{locked}, sid, ssid)$ to all parties and proceed to the Redeem Phase. Otherwise, for all i, if the message $(\mathsf{locked}, sid, ssid, i, D_i)$ was recorded, then delete it, send message $(\texttt{abort}, sid, ssid, i, \mathsf{coins}(d))$ to P_i and terminate.

Redeem Phase: In round τ, upon receiving a message $(\mathsf{redeem}, sid, ssid, i, w_i)$ from P_i, if $\phi(w_i) = 1$ then delete $(\mathsf{locked}, sid, ssid, i, D_i)$, send $(\mathsf{redeem}, sid, ssid, \mathsf{coins}(d))$ to P_i and $(\mathsf{redeem}, sid, ssid, i, w_i)$ to all parties.

Compensation Phase: In round $\tau + 1$, for all $i \in [n]$, if $(\mathsf{locked}, sid, ssid, i, D_i)$ was recorded but not yet deleted, then delete it and send the message $(\mathsf{payout}, sid, ssid, i, j, \mathsf{coins}(\frac{d}{n-1}))$ to every party $\mathsf{P}_j \neq \mathsf{P}_i$.

5.3 Penalty Protocols

5.3.1 Protocols Description

In what follows we briefly describe the penalty protocols on a high level, we invite the readers to find their details in the respective papers.

Ladder (L) [BK14]. Let f be the function being computed, and x_i be the private input of party P_i. At the beginning, the players run a cryptographically *unfair*, off-chain, MPC protocol for a derived function $\tilde{f}$ that: (i) computes the output $y = f(x_1, \ldots, x_n)$; (ii) divides y into n shares[6] $\sigma_1, \ldots, \sigma_n$; (iii) computes a commitment γ_i (with opening α_i) to each share σ_i, and gives (σ_i, α_i) to the i-th party and $(\gamma_j)_{j\in[n]}$ to every player.

Then, the players engage in a sequence of "claim-or-refund" transactions divided into two phases. During the Deposit Phase, each player conditionally sends some coins to another party via $\mathcal{F}^*_{\mathsf{CR}}$. These transactions are parameterized by the values γ_i, and require the receiving player to reveal the opening α_i before a fixed timeout, during the Claim Phase, in order to "claim" the reward (thus compensating honest players),[7] which otherwise will be refunded to the sender who will lose the coins sent to the honest parties without being able to redeem the coins received from them (*i.e.*, a penalty to the dishonest player). Finally, every party either reconstructs the output or receives a monetary compensation.

More in details, the Deposit Phase of Protocol L consists of Roof/Ladder Deposits, as illustrated below for $n = 4$:

$$\textbf{ROOF}\colon\ \mathsf{P}_j \xrightarrow[d,\tau_4]{\phi_{j,4}} \mathsf{P}_4 \text{ (for } j \in \{1,2,3\})$$

$$\textbf{LADDER}\colon\ \mathsf{P}_4 \xrightarrow[3d,\tau_3]{\phi_{4,3}} \mathsf{P}_3$$

$$\mathsf{P}_3 \xrightarrow[2d,\tau_2]{\phi_{3,2}} \mathsf{P}_2$$

$$\mathsf{P}_2 \xrightarrow[d,\tau_1]{\phi_{2,1}} \mathsf{P}_1$$

where $\mathsf{P}_i \xrightarrow[d,\tau]{\phi_{i,j}} \mathsf{P}_j$ indicates that P_i deposits d coins that can be claimed by P_j before time τ, as long as P_j sends to $\mathcal{F}^*_{\mathsf{CR}}$ a valid witness w for the predicate $\phi_{i,j}$. Importantly, the protocol requires that the claims happen in reverse order w.r.t. the deposits. Assume that P_3 is malicious and aborts the protocol during the Claim Phase. In such a case, P_1 would claim d coins from P_2 at round τ_1, whereas

[6] The secret sharing scheme ensures that an attacker corrupting up to $n-1$ players obtains no information on the output y at the end of this phase.

[7] More precisely, in L the condition requires the recipient i to publish the share σ_j and coins α_j such that $\mathsf{Commit}(\sigma_j; \alpha_j) = \gamma_j$ for each $j \leq i$.

P_2 would claim $2d$ coins from P_3 at round τ_2. If P_3 aborts (and thus it does not provide a valid witness), P_4 is refunded $3d$ coins at round τ_4. After that, at round $\tau_5 > \tau_4$, each $\mathsf{P}_{i\leq 3}$ is refunded d coins (from the roof deposits). Thus, P_3 loses $2d$ coins, while each $\mathsf{P}_{i\leq 2}$ is compensated with d coins.[8]

Locked Ladder (`LL`) [KMB15a]. This protocol is specifically tailored for playing distributed poker. To support multiple stages, a locking mechanism is designed to penalize aborting players in each phase of computation (in `L` players are penalized only during the Claim Phase). Moreover, `LL` yields a more complicated deposit sequence, as it requires additional deposits (called Bootstrap and Chain Deposits) to force the first party to start the new stage of the poker ideal functionality. See Section 5.6.3 for more details. The efficiency of this protocol has been further improved in [BKM17] using Ethereum smart contracts.

Compact Ladder (`CL`) [KVV16]. To prevent the explosion in script complexity (note that in `L` the witness in the last round is n times larger than the witness in the first round), protocol `CL` uses a trick that makes the size of the witness independent of the number of players. The basic idea is to replace $(\sigma_1, \ldots, \sigma_n)$ with a secret sharing $(\mathsf{k}_1, \ldots, \mathsf{k}_n)$ of the secret key k for a symmetric encryption scheme, and to reveal to every party an encryption c of the output y.

Planted Ladder (`PL`) [KVV16]. This protocol extends `L` to reactive functionalities by stacking multiple instances of `L`, *i.e.* an n-party r-stage functionality is handled as a run of `L` with $r \cdot n$ parties, using additional deposits to force the next stage to start (the so-called Underground Deposits). As a result, `PL` requires more transactions and very high deposits from each player. We illustrate the total amount of coins locked by each player in every round in Fig. 5.10 (see Section 5.6.3).To improve efficiency, one can replace `CL` with `L`; we denote the resulting protocol as `CPL`.

Amortized Ladder (`AL`) [KB16]. This protocol aims at performing multiple MPCs using a single instance of a penalty protocol. The sequence of deposits/claims is the same as in `LL`, except that all the deposits/claims happen in parallel.

Multi-Lock (`ML`) [KB14]. This protocol relies on the ideal functionality $\mathcal{F}^*_{ML}$, instead of $\mathcal{F}^*_{\mathsf{CR}}$, in order to realize the "claim-or-refund" transactions. The latter allows to manage multiple deposits/claims in an atomic fashion, thus resulting in an improved round complexity.

Insured MPC (`IMPC`) [BDD20]. This protocol follows the same blueprint of `ML`, *i.e.* `IMPC` manages multiple deposits/claims in an atomic fashion. However, the protocol further improves the efficiency in the evaluation of the commitments in the off-chain MPC and the on-chain reconstruction phase using publicly-verifiable additively homomorphic commitments.

[8] Player P_4 has not moved, and thus it is not compensated.

Other Protocols. The idea of guaranteeing cryptographic fairness through monetary compensation was originally studied in the setting of e-cash or central bank systems [BCE+07, Lin09, KL12], and implemented using Bitcoin by Andrychowicz *et al.* [ADMM14]. Other penalty protocols also exist for the concrete case of cryptographic lotteries [ADMM14, BK14, MB17]. Another type of penalty protocol is the one introduced in Hawk [KMS+16, Appendix G, Section B]. This construction follows the blueprint of ML except that it employs a *semi-trusted manager* in order to enforce a correct cash distribution. For further discussions see [AHM18].

5.3.2 An Illustrative (New) Protocol

To illustrate the trade-offs that a protocol must face, we provide a simple example of another protocol that achieves the same efficiency of the CL protocol by Kumaresan and Bentov [KB14]. Namely, we can design a new penalty protocol that combines ideas from [GIM+10] and [KB14, KB16], to obtain a constant-round penalty protocol with $O(n)$ transactions and script complexity $O(n\lambda)$. Looking ahead, this protocol is both cryptographically and financially fair; however, security only holds in the sense of sequential composition (although in the plain model and under standard assumptions).

Compact Multi-Lock (CML) We rely on some standard cryptographic primitives (cf. Chapter 2): (i) an n-party secret sharing scheme $(\mathsf{Share}, \mathsf{Recon})$ with message space $\{0,1\}^\lambda$ and share space $\{0,1\}^k$; (ii) a secret-key encryption scheme $(\mathsf{Enc}, \mathsf{Dec})$ with secret keys in $\{0,1\}^\lambda$ and message space $\{0,1\}^m$, where m is the output size of the function f; and (iii) a non-interactive commitment Commit with message space $\{0,1\}^k$.

Let f be the function to be computed. The protocol proceeds in two phases. In the first phase, the parties run a cryptographically unfair MPC for a derived function $\tilde{f}$ that samples a random key k, secret shares k into shares $\mathsf{k}_1, \ldots, \mathsf{k}_n$, commits to each share k_i individually obtaining a commitment γ_i, and finally encrypts the output $f(x_1, \ldots, x_n)$ using the key k yielding a ciphertext c. The output of P_i is $((\gamma_j)_{j\in[n]}, c, \mathsf{k}_i, \alpha_i)$, where α_i is the randomness used to generate γ_i. During the second phase, the players use $\mathcal{F}^*_{ML}$ to reveal the shares of the key k in a fair manner, thus ensuring that every player can reconstruct the key and decrypt the ciphertext c to obtain the output of the function.

This protocol requires a constant number of rounds, $O(n)$ transactions, and script complexity $O(n\lambda)$ where λ is a security parameter. The latter holds true so long as the secret key k for the symmetric encryption scheme is short (in fact, of length λ independent of m).

5.3.3 Extensions

Reactive CML, In the reactive setting, we have to securely realize ideal functionalities whose computation proceeds in stages. After each phase, the players receive a partial output and a state, which both influence the choice of the inputs for the next stage. Normally, simulation-based security for non-reactive functionalities implies

the same flavor of security for reactive ones [HL10]. However, for secure computation with penalties the naive approach (*i.e.* run an independent instance of the penalty protocol for each stage of the functionality) fails in case of aborts after a given stage is concluded (and before the next stage starts) as one needs a mechanism to force the parties to continue to the next stage.

Let π be a protocol for securely realizing (with aborts) a reactive functionality: during each stage, every player independently computes its next message as a function of its current input and transcript so far, and broadcasts this message to all other parties. Kumaresan *et al.* [KVV16] show how to deal with such protocols using PL. We can do the same for CL by leveraging the power of $\mathcal{F}_{ML}^{*}$. In particular, we consider an invocation of $\mathcal{F}_{ML}^{*}$ for each stage of π, where during the Lock Phase each player P_i specifies a circuit ϕ_i that checks the correctness of P_i's next message w.r.t. the protocol transcript so far. This is possible so long as the underlying MPC is publicly verifiable, a property also used in [KVV16]. The price to pay is a larger communication/round complexity.

The above requires to augment the $\mathcal{F}_{ML}^{*}$ functionality so that each player can deposit $\mathsf{coins}(d \cdot \bar{\tau})$, where $\bar{\tau}$ is the maximum number of stages in a run of π, and claim back at most $\mathsf{coins}(d)$ for each stage (*i.e.*, after revealing its next message).

BoBW CML, A drawback of cryptographic fairness with penalties is that a *single* corrupted player can cause the protocol to abort (at the price of compensating the other players). Ideally, we would like to have a protocol such that when $s_1 < n/2$ players are corrupted the protocol achieves full security, whereas in case the number of corrupted parties is $s_2 \geq n/2$, the protocol achieves security with aborts or, even better, fairness with penalties. This yields so-called best-of-both-worlds (BoBW) security [IKK+11], which is known to be impossible in the parameters regime $s_1 + s_2 \geq n$.

Kumaresan *et al.* [KVV16] provide a dual mode protocol achieving BoBW security for any $s_1 + s_2 < n$. We can easily adapt their approach—and, in fact, even simplify it thanks to the power of $\mathcal{F}_{ML}^{*}$—to our CML protocol, by using the following modifications: (i) The function $\tilde{f}$ now computes an (s_2+1, n)-threshold secret sharing of the symmetric key k; (ii) At the end of the Reconstruction Phase, all honest parties broadcast their share. Modification (i) ensures that, at the end of the Share Distribution Phase, an attacker controlling at most s_2 parties has no information on the output. The dual-mode protocol of [KVV16] relies on PL, as L does not guarantee compensation to honest parties that did not reveal their share (say, due to an abort during the protocol); this eventuality is not possible given the atomic nature of $\mathcal{F}_{ML}^{*}$, and thus no further change to CML is required to obtain fairness with penalties in the presence of s_2 corrupted parties. Modification (ii) is needed because when $s_1 < n - s_2$ parties are corrupted, there are at least $n - s_2 \geq s_1 + 1$ honest parties, which allows everyone to obtain the output by correctness of the secret sharing scheme.

5.4 Comparison over Security and Efficiency

5.4.1 Security Assumptions

Protocols L, ML, LL, AL and PL satisfy UC security, and IMPC satisfies (G)UC security with the global RO functionality. The situation is different for CL (and CPL as well). Recall that here the players start by engaging in an off-chain, unfair, MPC protocol whose output for party P_i includes (c, k_i), where c is an encryption of the output y under a symmetric key k, and k_i is a share of the key.

Unfortunately, the encryption must be "non-committing" [CFGN96] for the security proof to go through: the simulator first must send the adversary a bogus ciphertext c (say an encryption of the all-zero string), and when it learns the correct output y, if the adversary did not abort, must explain c as an encryption of y instead. In the plain model, such encryption inherently requires keys as long as the plaintext [Nie02], which would void any efficiency improvement w.r.t. the original L Protocol. To circumvent this problem, [KVV16] builds the encryption c in the ROM, essentially setting $c = \mathsf{Hash}(\mathsf{k}_1 \oplus \ldots \oplus \mathsf{k}_n) \oplus y$, where $\mathsf{k} = \mathsf{k}_1 \oplus \ldots \oplus \mathsf{k}_n$. This allows the simulator to equivocate the ciphertext in a straight-line fashion by programming the random oracle.

A considerable drawback of the hash-based CL Protocol is that it is *not* provably secure in the ROM because one cannot assume that Hash is a random oracle: to run an MPC protocol that computes $c = \mathsf{Hash}(\mathsf{k}_1 \oplus \ldots \oplus \mathsf{k}_n) \oplus y$ we must represent the very hashing algorithm as a circuit. Hence, security only holds under the controversial assumption that the above hash-based encryption scheme is non-committing when the random oracle Hash is instantiated with a real-world hash function (*e.g.*, SHA3). Note that this assumption is much stronger than just requiring that Hash is a random oracle.

Protocol CML follows the same blueprint as ML, but intuitively replaces the ideal functionality $\mathcal{F}^*_{\mathsf{CR}}$ with $\mathcal{F}^*_{\mathsf{ML}}$ in order to improve the round complexity and achieve financial fairness. As a consequence, its security analysis faces a similar issue as the one discussed above. Here, we propose an alternative solution that allows to obtain provable security in the plain model by focusing on standalone, rather than UC security (which in turn implies security under sequential composition). This weakening allows us to replace the non-committing encryption scheme with any semantically secure one, and to solve the issue of equivocating the ciphertext in the security proof by rewinding the adversary (which is not allowed in the UC setting). A similar solution was considered in [GIM+10] for fair MPC without coins.

Note that rewinding in our setting essentially means that the simulator should have the ability to reverse transactions on the blockchain, whereas distributed ledgers are typically immutable. However, there already exist certain blockchains where blocks can be redacted given a secret trapdoor [AMVA17] (and are immutable otherwise). In our case, such a trapdoor would not exist in the real world, but rather it would be sampled by the simulator in the security proof, in a way very similar to standard proofs of security in the common reference string model [BSMP91, CF01]. We further note that previous work also used limited forms of rewinding in the setting of MPC protocols with blockchains. For instance, Choudhuri *et al.* [CGJ19] construct black-box zero-knowledge protocols in a blockchain-hybrid model where

Table 5.2. Efficiency of State-of-the-art penalty protocols

Comparing penalty protocols in terms of round complexity, number of transactions, script complexity/capability, and fairness. We denote by n the number of parties, by m the output size of the function being computed, by r the number of stages in the reactive/-multistage setting, and by λ the security parameter.

Protocol	#Rounds	#TXs	Script Complexity
L	$O(n)$	$O(n)$	$O(n^2m)$
CL	$O(n)$	$O(n)$	$O(n\lambda)$
ML	$O(1)$	$O(n^2)$	$O(n^2m)$
CML	$O(1)$	$O(n^2)$	$O(n\lambda)$
IMPC	$O(1)$	$O(n)$	$O(nm)$
LL	$O(n)$	$O(n^2)$	$O(n^2mr)$
AL	$O(n)$	$O(n^2)$	$O(n^2m\lambda)$
PL	$O(n)$	$O(n)$	$O(n^2mr)$
CPL	$O(n)$	$O(n)$	$O(nr\lambda)$

the simulator is allowed to rewind only during certain slots.

5.4.2 Efficiency

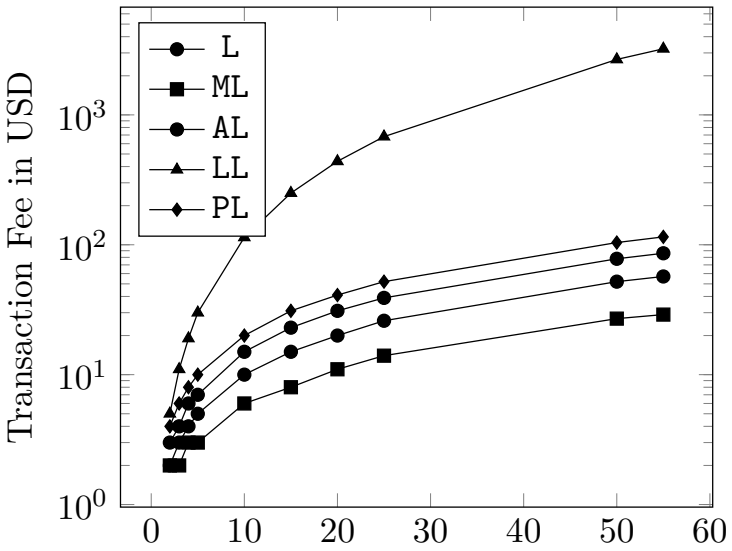
In this section we compare the penalty protocols w.r.t their on-chain efficiency in both an asymptotic and an empirical way (assuming their execution is on a Bitcoin network).In Table 5.2 we can notice that the script complexity of CL, CPL and CMLdoes not depend on the size of the output function, but only on the number of parties n and the security parameter λ, thus leading to a significant efficiency speed-up. As it can be noticed also in Fig. 1.1, CLand CPL are not provably secure, whilst our CML is secure only under standard composition. In terms of asymptotic efficiency we can state that our CML is indeed the most efficient, but due to its real world applicability to a blockchain[9], IMPC can be considered the best protocol among the presented ones.

Empirical efficiency is measured in terms of transaction fees (based on the number of transactions) in Fig. 5.2, execution time (based on the number of rounds) in Fig. 5.3, and script complexity (based on input size in bits) in Fig. 5.4. In our empirical analysis we do not take into account IMPC, CML, and CPL, since concrete efficiency numbers are not provided in the protocol description of IMPC, while CML is not universally composable and the latter not provable at all, and CPL is not applicable in an existing blockchain environment.

For transaction fees we assume the Bitcoin network's commonly used minimum transaction fee of 546 satoshis (1 satoshi = 10^{-8} BTC) and a BTC costs approximately 48k USD (by May 2021) . For the execution time, we use the standard assumption that 1 BTC round is one hour. For script complexity, we assume 80 bits security with input size (shares) of 128 bits and the commitment scheme is SHA-256 (pre-images of 512 bits and outputs of 256 bits). L, ML, and AL are non-reactive protocols. We assume the reactive LL and PL are with 2 stages. For the PL we evaluate both the naive version and the compact version CPL.

We simulate the on-chain efficiency for 2-55 parties using the practical case

[9]recall that sequential composability cannot be run in a blockchain ecosystem.



The most expensive LL requires 12312 transactions, which costs around 3277$ for the 55 parties case, approximately 60$ per party.

Figure 5.2. Transaction fees (based on the number of BTC transactions, where 1 transaction costs 546 satoshis, 1 BTC = 48k USD, by May 2021).

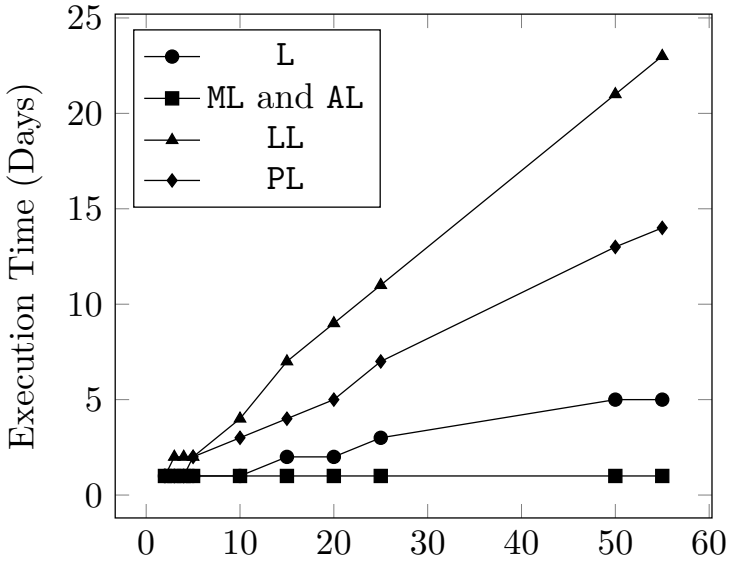
from the Bloomberg dark pool, just to execute one contract in the case of non-reactive functionality, or two in the case of reactive functionality. All of the protocols show acceptable transaction fees, even the most expensive protocol LL, costs only 3277$ for the 55 parties case (which means approximately only 60$ for each party). However, all protocols except ML and AL show unacceptable execution time: ML concludes in the simplest (non-reactive) protocol L takes 5 days to finish for 55 parties to execute 1 contract while the most complicated (reactive) protocol LL requires 23 days to execute 2 contracts; the improved (reactive) protocol PL does reduce the requirement to 14 days but it is still too slow. ML is the protocol with the lowest script complexity in the case of non-reactive while Compact PL is the protocol with the lowest script complexity in the case of reactive. The non-compact version of PL yields the highest script complexity.

5.5 Comparison over Financial Fairness

5.5.1 Deposits Schedule and Illustration of Financial Unfairness

The amount of deposited coins for each player in the Ladder Protocol is illustrated in Fig. 5.5 (for the 4-party case). Observe that P_1 has to deposit only d coins, while P_4 needs to deposit $3d$ coins. Furthermore, P_4 has to lock its coins very early (*i.e.*, at the 2nd round), but can only claim its coins very late (*i.e.*, at the last round). Hence, this protocol is financially *un*fair in the following sense: (i) The amount of deposits are different for each player (*e.g.*, P_1 deposits d coins while P_n deposits $(n-1)d$ coins); and (ii) some players deposit early but can only claim late in the protocol (*e.g.*, P_4 in Fig. 5.5).

While financial *un*fairness is easy to notice (by pure observation) in simple pro-



The simplest protocol L takes 216 rounds, which is 5 days to finish, while the most complicated protocol LL takes 23 days to finish. The improved protocol PL takes 14 days to finish.

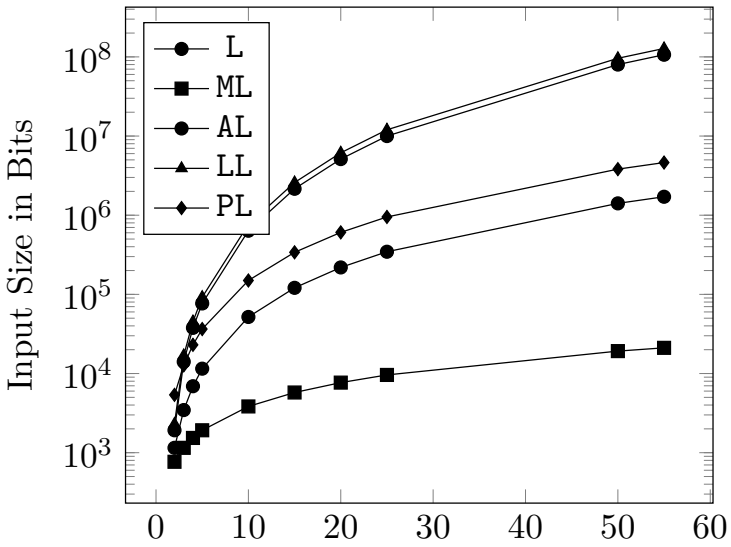
Figure 5.3. Execution time (based on number of protocol rounds, where 1 round = 1 hour).

tocols such as Ladder, it tends to be more difficult to judge whether a penalty protocol is financially fair or not when it yields a more complicated sequence of deposit and claim transactions [KMB15b, KB16, KVV16].

If the 4-party Ladder protocol was to be run on Bitcoin (that requires 8 rounds in total, thus $\tau = 8$) , a round would last approximately 60 minutes.We assume a very optimistic scenario: participants could borrow money from NYFed's SOFR to run the protocol (see Section 5.2.1, or alternatively that could be their opportunity costs). In essence they are wealthy, risk neutral and worth essentially cheap credit. Normal humans would require much higher interest rates as the empirical evidence shows [BRY89, AW97, BIW15]. The discount (minute) rate would be $\delta_{\mathsf{m}} = 0.0005$ for each player. To represent the *net present cost of participation* of each participant as basis points we simply set $d = 10000$. Thus, using Eq. (5.1), we have (in bps):

$$\chi_1(8) := -\left(-d \cdot e^{-\delta_{\mathsf{m}} \cdot 60} + d \cdot e^{-\delta_{\mathsf{m}} \cdot 5 \cdot 60}\right) \approx 0.11$$
$$\chi_2(8) := -\left(-d \cdot e^{-\delta_{\mathsf{m}} \cdot 60} - d \cdot e^{-\delta_{\mathsf{m}} \cdot 4 \cdot 60} + d \cdot e^{-\delta_{\mathsf{m}} \cdot 6 \cdot 60}\right) \approx 0.19$$
$$\chi_3(8) := -\left(-d \cdot e^{-\delta_{\mathsf{m}} \cdot 60} - 2d \cdot e^{-\delta_{\mathsf{m}} \cdot 3 \cdot 60} + 3d \cdot e^{-\delta_{\mathsf{m}} \cdot 7 \cdot 60}\right) \approx 0.38$$
$$\chi_4(8) := -\left(-3d \cdot e^{-\delta_{\mathsf{m}} \cdot 2 \cdot 60} + 3d \cdot e^{-\delta_{\mathsf{m}} \cdot 8 \cdot 60}\right) \approx 0.49.$$

Roughly speaking, this means that P_4 loses 0.0049% of the base deposit d just to participate, and additionally the participation cost of P_4 is approximately 4 times higher than that of P_1. If one is to use the Ladder protocol in a real world use case (*e.g.*, dark pool futures trading), the base deposit d would be necessarily at least the notional value of one futures contract (*i.e.*, 1 million dollars in case of Eurodollar



ML is the protocol with the lowest script complexity in the case of non-reactive while AL is the protocol with the lowest script complexity in the case of reactive. We do not consider CL and CPL since they do not satisfy the security requirements needed to be run on an existing blockchain, and IMPC because of lack of concrete efficiency numbers.

Figure 5.4. Script complexity (input size in bits).

futures); as such, even with only 4 parties, P_4 would lose \$50 just to participate to trade a single contract!

Unfairnness is manifest already (yet seems small and bearable) in this example with a handful of participants (only 4). However, in a Bloomberg Tradebook:

- The number of parties could be 55 in an average trading day; as such, the protocol would last 110 rounds, and thus, $\chi_{55}(110) := -(-54d \cdot e^{\delta_{\mathsf{m}} \cdot 2 \cdot 60} + 54d \cdot e^{\delta_{\mathsf{m}} \cdot 110 \cdot 60}) \approx 159(bps)$, *i.e.* P_{55} would lose 1.59%, which is almost \$16K, just to participate to trade a single contract!
- The number of messages (protocol executions) could reach 6000 in an average trading day; as such, even with 4 parties, P_4 would spend \$300K a day.
- Combining the numbers, the last party P_{55} would spend \$96M just to participate to an average trading day, which is unacceptably different from the first party's cost which is only around \$60K (\$10 per contract, as the cost of P_1 is only 0.1 bps) .

5.5.2 Theoretical Analysis of Financial Fairness

We formally prove that the family of Ladder Protocols does not meet financial fairness as per our definition. The latter is achieved by interpreting L, LL, CL, PL and AL as MPC protocols in the $\mathcal{F}^*_{\mathsf{escrow}}$-hybrid model, and by carefully analyzing the sequences of deposits/refunds made/received by each participant.

Theorem 9. *For any $n \geq 2$, and penalty amount $q > 0$, the following holds for the n-party Ladder protocol π_{L} from [BK14]:*

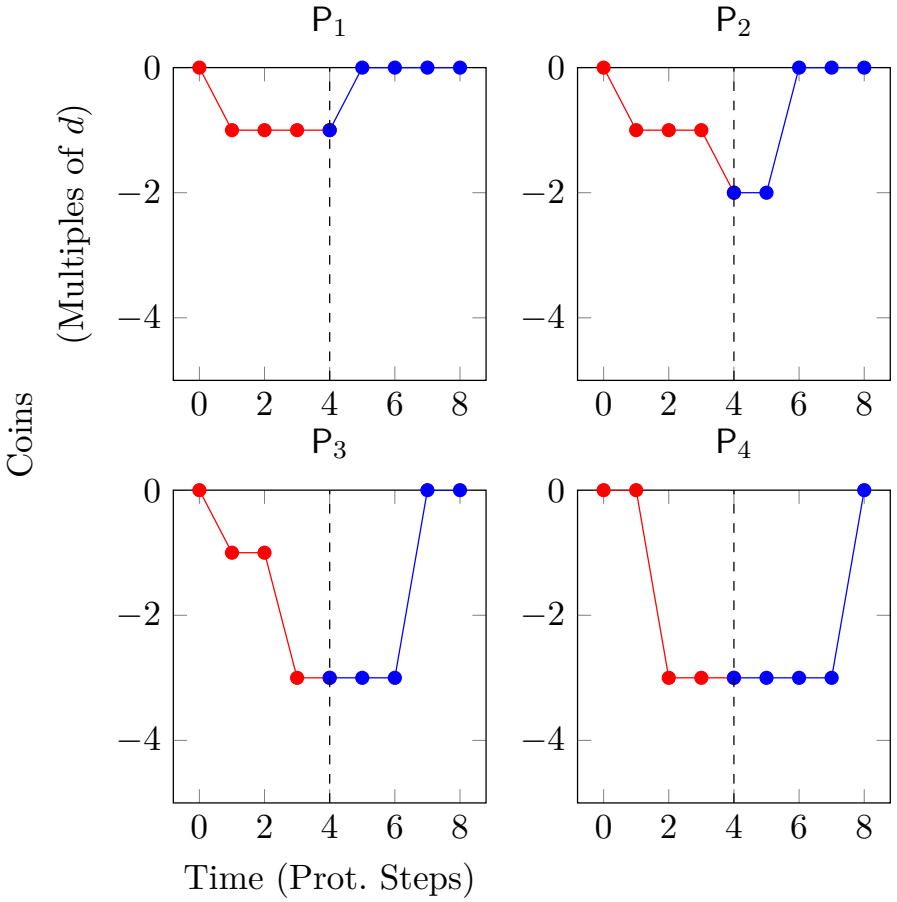


Figure 5.5. Coins locked in a run of the 4-party Ladder Protocol during the Deposit Phase (in red) and the Claim Phase (in blue).

- *If $\eta = 1$, the protocol is financially fair.*
- *If $\eta \neq 1$, the protocol is* not *financially fair.*

Proof. Consider the hybrid ideal functionality $\mathcal{F}^*_{\mathsf{CR}}$ described in Sec. 5.2.4; intuitively this functionality allows a sender P_i to *conditionally* send coins to a receiver P_j, where the condition is formalized as a circuit $\phi_{i,j}$ with time-lock τ: The receiver P_j can obtain P_i's deposit by providing a satisfying assignment w within time τ, otherwise P_i can have his deposit refunded at time $\tau + 1$.

$\mathcal{F}^*_{\mathsf{CR}}$ clearly belongs to the family $\mathcal{F}^*_{\mathsf{escrow}}$ described in Section 5.2.2: the Deposit Phase corresponds to the $(\texttt{deposit}, *, *)$ commands in $\mathcal{F}_{\mathsf{escrow}}$, whereas the Claim/Refund Phase corresponds to the $(\texttt{refund}, *, *)$ commands in $\mathcal{F}_{\mathsf{escrow}}$.

Given $\mathcal{F}^*_{\mathsf{CR}}$, protocol π_{L} proceeds as follows. First, each player P_i (except for P_n) uses $\mathcal{F}^*_{\mathsf{CR}}$ to make a deposit of $\mathsf{coins}(q)$ to P_n, with predicate[10] ϕ_n. After all these deposits are made, each party P_i with $i = n, \ldots, 2$ in sequence uses $\mathcal{F}^*_{\mathsf{CR}}$ to make a deposit of $\mathsf{coins}((i-1)q)$ to P_{i-1} and with predicate ϕ_{i-1}. Let us call these deposits Tx_{i-1}, and denote by $\mathsf{Tx}_{n,i}$ the initial deposits to P_n. Finally, the deposits

[10]We do not specify the predicates, as those are immaterial for characterizing the financial fairness of the protocol.

are claimed in reverse order: First, P_1 claims Tx_1, then P_2 claims Tx_2, until P_n claims $\mathsf{Tx}_{n,i}$ for each $i \neq n$. Aborts are handled as follows: If P_{i+1} does not make Tx_i, each party $\mathsf{P}_{j \leq i}$ does not make Tx_{j-1} and waits to receive the refund from $\mathsf{Tx}_{n,j}$, whereas each $\mathsf{P}_{j>i}$ keeps claiming Tx_j as described above.

For simplicity we consider all parties are honest. The loss of party P_i is:

$$\chi_{i \neq n} = -q \cdot \eta(1) - (i-1) \cdot q \cdot \eta(n-i+2) + i \cdot q \cdot \eta(n+i).$$
$$\chi_n = -(n-1) \cdot q \cdot \eta(2) + (n-1) \cdot q \cdot \eta(2n).$$

When $\eta = 1$, we have $\chi_1 = \ldots = \chi_n = 0$. However, since, *e.g.*, $\chi_1 \neq \chi_2$ for any choice of $\eta < 1$, we conclude that π_{L} is not financially fair whenever $\eta \neq 1$. □

The ML protocol is financially fair. [11]

Theorem 10. *For any $n \geq 2$, ML [BK14] is financially fair.*

Proof. It is easy to see that $\mathcal{F}^*_{ML}$ belongs to the family $\mathcal{F}^*_{\mathsf{escrow}}$. Then, financial fairness immediately follows by the fact that the loss of the i-th player can be computed as follows:

$\chi_i = -(n-1)q \cdot \eta(1) + (n-1)q \cdot \eta(r) + s \cdot q \cdot \eta(r+1)$, where $s \leq n-1$ is the number of corrupted parties that did not redeem a valid witness in the fair reconstruction phase. □

5.5.3 Experimental Analysis of Financial Fairness

We experimentally analyze how the different penalty protocols behave in terms of their inter-temporal choices.[12].

- **The Multi-Lock Protocol.** ML is straightforward in this respect. Every party deposits the same amount of coins at the same time, and can withdraw it as soon as s/he has revealed the secret. The same holds for CML.
- **The Ladder Protocols**. The L, LL, PL, and AL protocols have inter-temporal payment schedules which clearly differ in both amount and duration of deposits per party.[13] To show the difference we implemented a script that simulates the penalty protocol transaction schedule.

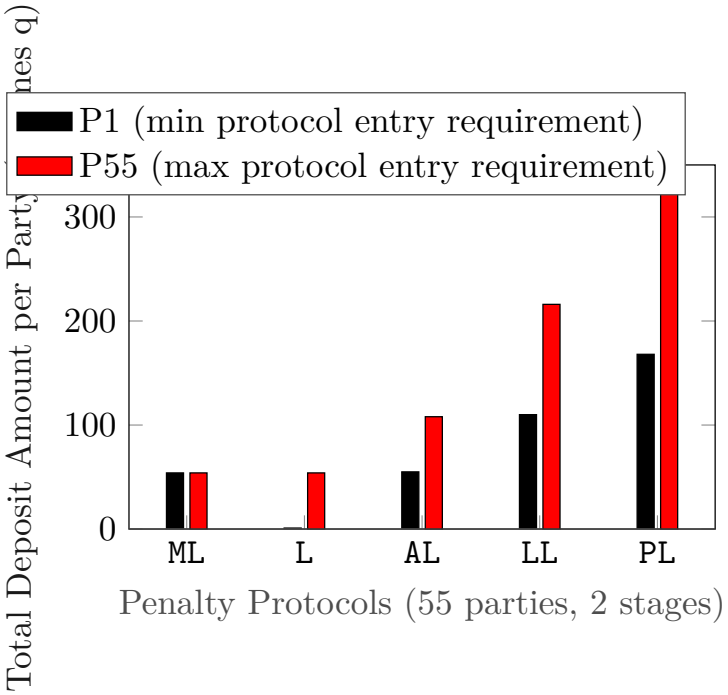
Setting the Stage

We consider the 55-party realistic case with a minimum penalty chip of q to be consistent with the cited papers. Notice that the unfairness phenomena are amplified with n parties (*e.g.* a lit futures trading venue would comprise up to 500 parties). If the protocol supports reactive functionalities (*i.e.*, LL and PL), we limit the number of stages to 2. For each protocol L, LL, PL, AL, ML, we first simulate the sequence of deposits $\{d_{i,t}\}$ (with q being the base unit used for penalization) and withdraws

[11] IMPC [BDD18] can similarly be shown to be financially fair.

[12] While it seems that results could be derived with pencil and paper as all protocols are known, the simulation shows results that are not obvious as even for a small number of parties (55) the numbers of rounds can be very large 300).

[13] IMPC follows the same deposit/withdrawal blueprint as ML.



This figure shows the total amount of deposit locked by the penalty protocols before the withdrawal phase. CRYPTO'14 `L`, CCS'14 `ML`, and CCS'16 `AL` are non-reactive penalty protocols, while CCS'15 `LL` and CCS'16 `PL` are reactive ones. Among the non-reactive protocols, `L` requires significantly different amounts from the first party (always q) and the last party which shells 54 times as much and should shell $(n-1)q$ for n parties. The reactive `LL` and `PL` require disproportionate deposits. For each stage of computation, maximum 54 out of 55 parties may be compensated if one party cheats. Therefore one would only expect $118q$ for two stages of computation (to compensate the other 54 honest parties in 2 stages) rather than 216 or 320+.

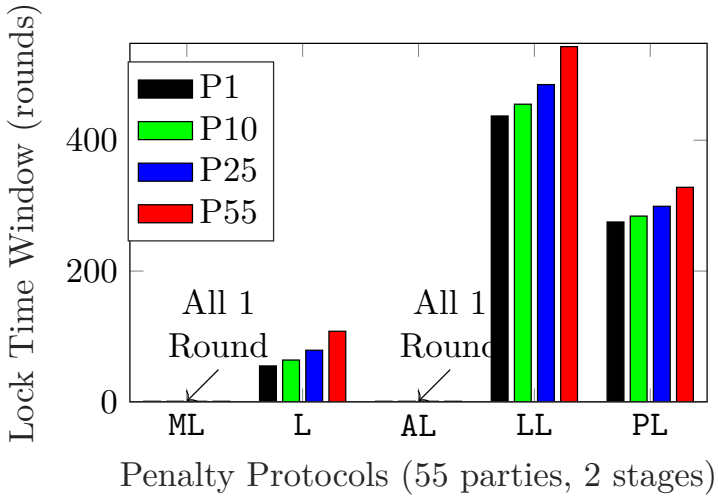
Figure 5.6. Total Amount of Deposit per Party and Protocol

$\{q_{i,t}\}$ of each P_i from an honest execution of the protocol. We show only the results for P_1, P_{10}, P_{25}, and P_{55}.[14]

Fig. 5.6 reports the total deposited amount of P_1 (the minimum entry requirement of a protocol) and P_{55} (the maximum entry requirement of a protocol). All protocols except `ML` require a different amount of deposit from each party. In terms of total amount, `L` is the best protocol... for the first party! The last party has to deposit more than 54x times more. `ML` requires a fixed amount of $(n-1)q$ from each party, while `L` requires such amount from only the two last parties P_{54} and P_{55}. `LL`, `PL` and `AL` require very high amounts of deposits (and again largely different): the worst case party P_{55} has to deposit $216q$ in `LL` and $327q$ in `PL`. Even taking into account the fact that `LL` and `PL` consist of two stages, such deposits look excessive: one would expect to deposit $118q$ for two stages of computation, since we only need to compensate at most 54 parties per stage when one of the 55 parties aborts.

Fig. 5.7 shows the maximum time window that a party has to keep his money

[14] P_1 and P_{55} are the first and the last party, which illustrates the maximum difference possible. P_{10} and P_{25} are representative of the parties that are in the middle.



Among the non-reactive protocols, CCS'14 `ML` and CCS'16 `AL` both conclude in one round. Moreover, `AL` allows multiple MPCs to be done in the one round period. The reactive CCS'15 `LL` and CCS'16 `PL` again require a disproportionate lock time window compared to the non-reactive `L`, `ML`, `AL`.

Figure 5.7. Maximum lock time window (55 parties, 2 stages)

in deposit (starting from the first deposit to the last withdrawal). `ML` and `AL` have the smallest and fairest lock time: only one round. `L` must keep the deposits in more rounds and not the same number of rounds: 55 rounds for the best case P_1 and 108 rounds for the worst case P_{55}. Again both `LL` and `PL` require very high lock windows for the deposits: 543 rounds for `LL` and 328 rounds for `PL`.

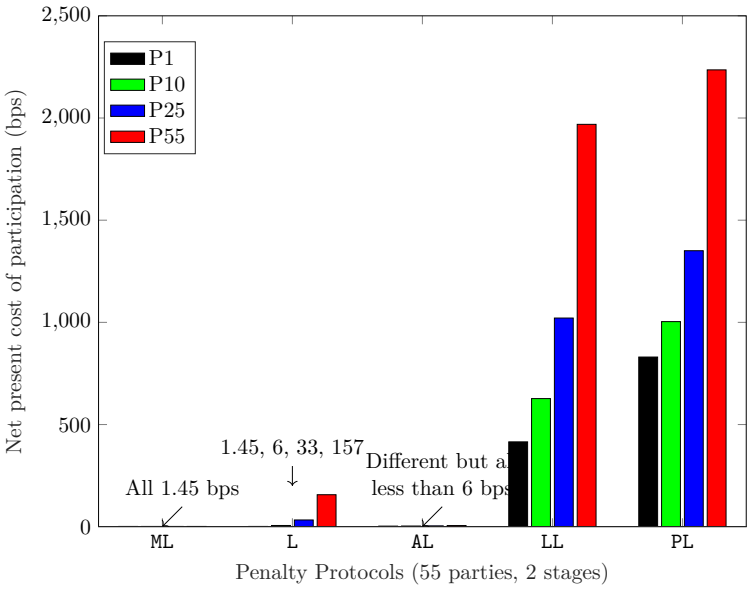
Playing It for Real: Optimistically Unfair

The observations above refer to the worst case but in practice the inter-temporal differences for honest parties might not be noticeable. Checking the behavior of a protocol for honest parties, dubbed *Optimistic Computation* [KB14], is important as a protocol can still be fair for all practical purposes.

To check whether this is the case (it is not), we analyze the financial fairness of each protocol by simulating the *net present cost of participation* χ_i of party P_i (see Eq. (5.1)) in a large sequence of random executions with honest parties:

- Use the sequence of deposits $\{d_{i,r}\}$ and withdraws $\{q_{i,r}\}$ of party i obtained from an honest execution of a protocol at each round r; and the minute ratio derived from the Secured Overnight Financing rate of the New York Fed (238 bps, Tab. 5.1) as this is the going rate among commercial executions, and thus is actually available;

- Simulate each protocol execution on Bitcoin. To convert "rounds" to "Bitcoin time", we use Bitcoin actual network data, *i.e.* the mean and standard deviation of the block generation time (in minutes) for each day from Dec 29, 2018 to June 26, 2019; and consider a round of a protocol to be 6 blocks of



Each bar reports the cost of participation for a party in basis point for an optimistic computation (only honest parties and no aborts). CCS'14 `ML` is the fairest protocol with the smallest (and statistically identical) distribution per party. CCS'16 `AL` provides limited financial fairness even though deposits lock in one round (Fig. 5.7), because each party deposits a different amount (Fig. 5.6). CRYPTO'14 `L` is the least fair.

Figure 5.8. Net present costs of participation

the Bitcoin blockchain for a transaction to be confirmed). From the data, a round can take from 47 minutes to 75 minutes on average.

- Compute the net present cost of participation χ_i of each P_i for each of the 180 days using $q = 10000$ (hence χ_i is captured as basis points), and plot them in Fig. 5.8.[15]

For all cases (both reactive and non-reactive), financial fairness is only achieved in `ML`, as every party locks and releases the same deposit at the same time. `ML` is also the best protocol in terms of net present cost of participation.

For non-reactive cases, `L` yields a huge difference in losses between different parties: χ_1 is around 1.53 bps while χ_4 is around 162.75 bps. This difference is due to the disparity in both the amount of deposits and the time windows in which the deposits are locked (P_1 deposits only q, locked for 55 rounds, while P_{55} deposits $54q$, locked for 108 rounds). The difference is slighter in `AL`: all parties' deposits are locked for one round, but differences between amounts of deposits still exists ($110q$ for P_1 and $216q$ for P_{55}).

For reactive cases, in `LL` and `PL` the party P_1 and the party P_{55} have a large difference in net present costs of participation. Furthermore, the costs for the last party P_{55} are unacceptable in both protocols: more than 2000 bps in `LL`, and more

[15] As the discount rate is small, *i.e.* $\delta_{\mathsf{m}} = 0.0005$, the difference due to slight changes (30 minutes) in transaction confirmation times is negligible. Only for a very large number of transactions it becomes significant.

than 2300 bps in PL. However, a surprising finding is that LL is better than its "improved" version PL in terms of financial fairness. To explain this phenomenon, let us observe that even though LL locks the deposits for a longer time (LL concludes in 543 rounds, while PL needs 328 rounds), the deposit amount is much less (P_{55} deposits $216q$ in LL but $327q$ in PL).

5.6 Simple Fixes Do Not Work

A natural idea to overcome the negative results on financial fairness shown in Section 5.5 would be to simply let parties rotate their roles in different executions, or to select the roles randomly in each execution, with the hope of achieving financial fairness on expectation. Unfortunately, we show here that these approaches are also deemed to fail, except for a finite, very small, numbers of discount rates.

5.6.1 Round Robin

The "round robin" approach considers a global protocol which consists of multiple repetitions in a round robin fashion of a financially *un*fair penalty-based protocol (such as the Ladder protocol L [BK14]). This hopes to fix the unfairness in the penalty-based protocol if the same set of parties are to run the protocol more than once: by shifting the party index in each run, e.g. the last party becomes the first party, the k-th party becomes the k+1-party in the next run, one might be possible to fix the financial unfairness. We note that this is different from the penalty protocols that support multi-stages reactive functionality (such as Locked Ladder LL [KMB15b]), as even though those protocols seem to be based on repeated instances of non-reactive protocol, one cannot shift the party index because it will be insecure.

Unfortunately this solution doesn't work in general even for the simple case in which the reward is the same for all parties.

In the following theorem, we will show that to achieve fairness in the mentioned round robin approach, the parties must be able to obtain a specific limited number of discount rates (e.g. from the banks), depending on the deposit schedule, which is very unpractical (as the discount rates are given by the banks, not asked by the parties).

Theorem 11. *Let an unfair protocol be identified by deposits $d_{i,r}$ for each party $i \in [n]$ which may be rotated to different parties at round robin step $\rho \in [k]$ thus determining a schedule $d_{i,r+\rho}$. There are at most $\tau \cdot k$ specific rates δ that admit a fair round robin global protocol.*

Proof. We observe that for the round-robin protocol to be fair we need to satisfy the following equation for all pair of parties i and j: $\sum_\rho \chi_{i,\rho} e^{-\delta\rho} = \sum_\rho \chi_{j,\rho} e^{-\delta\rho}$

In the simple case where each party have the same reward at the end of the protocol, it can be transformed as follows:

$$\sum_{\rho} \sum_{(r,d_{i,r})} d_{i,r+\rho} \cdot e^{-\delta(\rho+r)} = \sum_{\rho} \sum_{(r,d_{j,r})} d_{i,r+\rho} \cdot e^{-\delta(\rho+r)}$$

By setting $e^{-\delta(\rho+r)}$ equal to $x^{\rho+r}$ we obtain a polynomial equation of degree at most $k\tau$ with integer coefficients equal to $d_{i,r+\rho} - d_{j,r+\rho}$. We can repeat the procedure for all pairs and we obtain at most $n-1$ independent polynomial equations. It is enough that we consider the pairs $1, i$ for all $i \geq 2$. The remaining equations can be derived from those ones. Since the original protocol is unfair, there must be at least one polynomial where at least one of such coefficient for each value of ρ is not zero, hence each polynomial is not identically zero and of rank at least $(k-1)\tau$ and maximum of $k\tau$. Hence each polynomial has at most $k\tau$ zeroes i.e. values of δ that admit a fair round schedule. If the other polynomials are also not identically zero, such values δ must also be zeroes for the other $n-1$ polynomials. □

s

5.6.2 Small Collateral and Repeated Games

Another approach to allow the use of financially unfair protocols in practice might be the use of a small collateral. Then all parties will not worry on a small interested rate on the collateral if they have a choice of a significant reward at the end of the protocol. If the game is repeated several thousands times, e.g. in financial trading, a small collateral might quickly accrue to a significant values and therefore such solution might only hold for games that are not played often.

Unfortunately, game theoretic considerations makes such proposition (make a small collateral wrt stakes and rewards so that interest is negligible) less practical than it seems. It only works if *all* parties have a final large reward with certainty. In cases where a party may win a lot and other parties may lose everything, such as poker or financial trading, this is not longer true. We illustrate it for the simple case of two players (Alice going first and Bob going last) and one scoping the full reward leaving the other with nothing.

Playing last and abort if unsatisfied, is a strictly dominating strategy for a single game where the collateral is negligible in comparison to the initial stake, since it is a simple variant of the prisoner dilemma [B$^+$07, Chap.2].

The last player can chose between abort and retrieve the initial stake minus the collateral (if the result is in favor of the first player) or cooperate and retrieve the full reward. The first player can decide to (1a) abort and retrieve the initial stake minus the collateral, cooperate and if the last players cooperate (2a) retrieve nothing for herself or (3a) grab the reward depending on the random outcome of the computation, or (4a) retrieve the initial stake plus the collateral is the last player aborts. In contrast the last player can (2a) retrieve the reward if he cooperates and the outcome is positive or (4a) abort and retrieve the initial stakes minus the collateral if the outcome is zero for him. The option (3a) of cooperating if the outcome is nothing for him is dominated by the the option (4a) of retrieving the initial stake minus the collateral. Hence the Nash equilibrium is first player cooperates, last player aborts if he doesn't win.

In a repeated games with discount rates for later moves (See Section 8.3.3 in [B$^+$07]) both players may cooperate if the discount rate is large enough even if the individual game would have a dominating strategy for defecting (i.e. in our case going last and abort if unsatisfied). Unfortunately, this case is not applicable in our

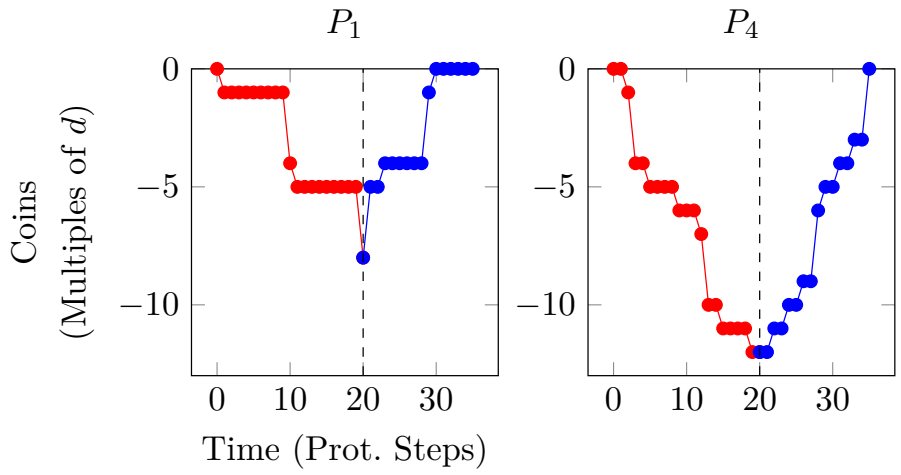


Figure 5.9. Coins locked in a run of the 4-party 2-stages Locked Ladder Protocol during the Deposit Phase (in red) and the Claim Phase (in blue).

scenario as it requires players to have strategies that are contingent on the previous behavior of the game, i.e. players needs to know how the other players played in the previous instance of the game. Since players might join with a new pseudonym, one cannot hold them accountable for repeated aborts. Therefore the repeated game collapses into a sequence of independent games.

5.6.3 Further Examples of Ladder Protocols

Below, we illustrate the 3-party case of the `LL` protocol (without considering Bootstrap Deposits).

$$\textbf{ROOF}\colon P_j \xrightarrow[q,\tau_4]{TT_3} P_3 \text{ (for } j \in \{1,2\})$$
$$\textbf{LADDER}\colon P_3 \xrightarrow[q,\tau_3]{TT_2 \wedge U_{3,2}} P_2 \text{ (Rung Unlock)}$$
$$P_3 \xrightarrow[2q,\tau_2]{TT_2} P_2 \text{ (Rung Climb)}$$
$$P_2 \xrightarrow[q,\tau_2]{TT_1 \wedge U_{2,3}} P_3 \text{ (Rung Lock)}$$
$$\textbf{FOOT}\colon P_2 \xrightarrow[q,\tau_1]{TT_1} P_1$$

The tokens $U_{i,j}$ are necessary to avoid specific attacks, for further details look at

Fig. 5.9 and Fig. 5.10 show the amount of coins locked during the Deposit and Claim Phase of the 4-party `L` and `PL` Protocols. Since `L` and `PL` are for reactive functionalities, in the figures we assume a 2-stages functionality for each protocol. Fig. 5.11 illustrates the amount of coins locked for the `AL` Protocol.

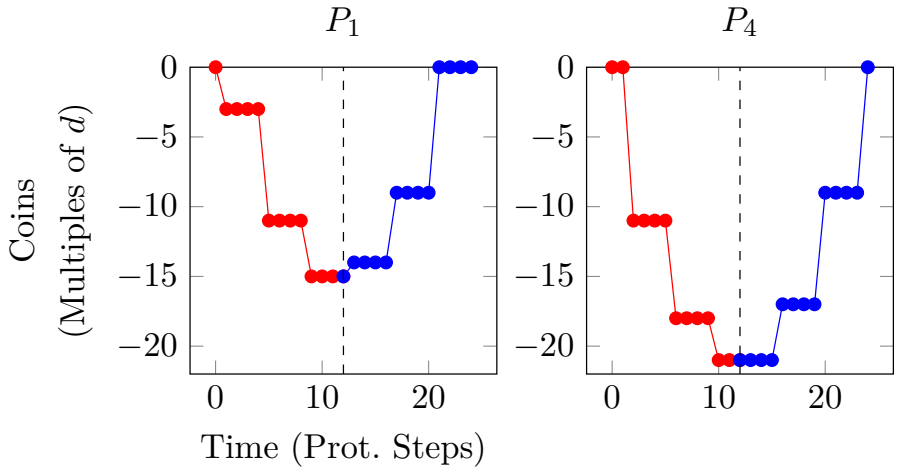


Figure 5.10. Coins locked in a run of the 4-party 2-stages Planted Ladder Protocol during the Deposit Phase (in red) and the Claim Phase (in blue).

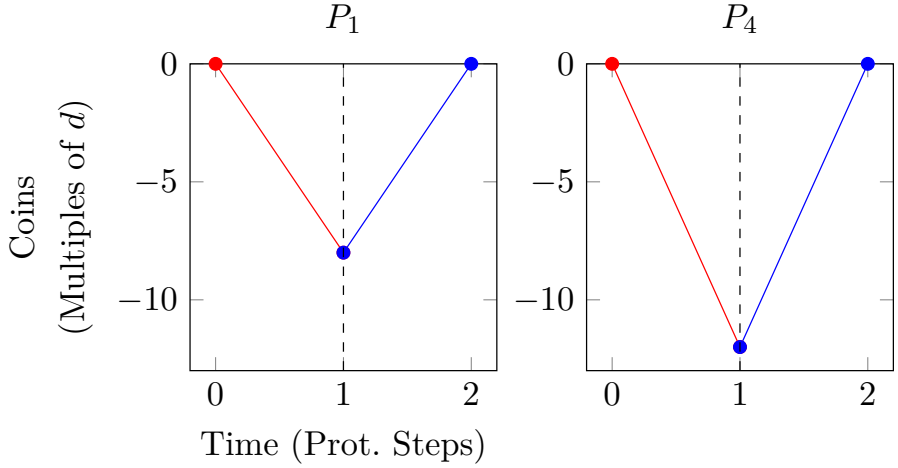


Figure 5.11. Coins locked in a run of the 4-party Amortized Ladder Protocol during the Deposit Phase (in red) and the Claim Phase (in blue).

Chapter 6

Conclusions

We have considered MPC protocols under different perspectives, including new assumptions for low round MPC and publicly verifiable and fair with penalties protocols that are financially sustainable when run with decentralized payments systems relying on smart contracts like Bitcoin or Ethereum blockchains.

In particular, we have shown a construction of maliciously secure oblivious transfer (M-OT) protocol from a certain class of key agreement (KA) and semi-honestly secure OT (SH-OT) protocols that enjoy a property called *strong uniformity* (SU), which informally means that the distribution of the messages sent by one of the parties is computationally close to uniform, even in case the other party is malicious.

When starting with 2-round or 3-round SUSH-OT or SU-KA, we obtain 4-round M-OT, and thus, invoking [BL18], 5-round maliciously secure MPC from standard assumptions including low-noise LPN, LWE, Subset Sum, CDH, DDH, and RSA (all with polynomial hardness).

A significant open problem is constructing round-optimal (i.e., 4-round) maliciously secure MPC from 4-round M-OT. Also, it is a natural question to see whether SU-KA with $r \geq 4$ rounds can be instantiated from even weaker assumptions.

Then, we have focused on public verifiable and asynchronous (in the sense that players can give their contribution to the protocol at any time) MPC protocols implemented on forking blockchains using smart contracts, and how to design such protocols allowing players to be hasty (i.e., without being delayed by finality limitations).

Beyond the double-spending attack, there are other issues that can affect both security and privacy of MPC protocols implemented by a smart contract. On the negative side, we showed that a well-known MPC protocol implemented via smart contracts becomes insecure in the presence of forks and hasty players (because the adversary can play adaptively on a branch of a fork depending on the information observed on the other branch). On the positive side, we have shown smart contracts within on-chain MPC protocols that remain secure even when there are forks and players are hasty.

Moreover we have also discussed how to get fairness with penalties. This allows us to get smart contracts that are simultaneously safe, fair and fast. We have also provided in Section 4.3.2 some experiments to show noticeable improvements of our PCT protocol with respect to the lottery protocol of Andrychowicz et al. in terms

of number of blocks needed for completion of the protocol and gas consumption of the smart contracts.

Finally, we have shown that cryptographically-fair (with penalties) MPC with penalties might be *unfair* when it comes to the amount of money each player has to put into escrow in a run of the protocol. Hence, that the goal is designing penalty protocols that are both cryptographically and financially fair, while at the same time having good efficiency in terms of round complexity, number of transactions, and script complexity.

State-of-the art protocols either achieve low script complexity (at the price of unrealistic assumptions) but not financial fairness [KVV16], or achieve financial fairness but either have high script complexity [KB14] or require trusted third parties [KMS+16]. Alternatively, we also showed that efficiency, cryptographic fairness, and financial fairness are all achievable at the same time and under standard assumptions, so long as one settles for sequential (rather than universal) composability using rewinding-based proofs. The latter might indeed be an option under certain restrictions [CGJ19], or using redactable blockchains [AMVA17].

We make no assumption on the function used to compute the net present value, but in some settings we may want to consider financial fairness only w.r.t. specific discount rates. An extension would be to drop the assumptions, present in the entire literature so far (including our paper) that (a) all parties are compensated equally; and (ii) the adversary compensates all honest parties who do not receive the result of the computation. These two assumptions are apparent, respectively, in the compensation step of $\mathcal{F}^*_{ML}$ (where $(\mathsf{payout}, i, j, \mathsf{coins}(\frac{d}{n-1}))$ is sent to every $\mathsf{P}_j \neq \mathsf{P}_i$), and in the third step of $\mathcal{F}^*_f$ (the `extrapay` step). This approach, known as the "pro-rata" approach for the restitution of mingled funds, however, is not the only possible one [Bur11]. For example, one could use Clayton's rule where "first withdrawals from an account are deemed to be made out of first payments in" [CC14], and return the funds only to the first k parties who deposited. Adjusting to these rules requires simple modifications to the functionalities and the corresponding protocols.

We also shown that the `CML` Protocol only achieves standalone security. We leave it as an open problem to construct a penalty protocol that achieves UC security, with the same efficiency as `CML` and while retaining provable security in the plain model. Note that this question is wide open even in the ROM, as the `CL` Protocol from [KVV16] fails to achieve provable security in the ROM (as discussed in Section 5.4).

Bibliography

[ACJ17] Prabhanjan Ananth, Arka Rai Choudhuri, and Abhishek Jain. A new approach to round-optimal secure multiparty computation. In *Proc. of CRYPTO*, pages 468–499, 2017.

[ADMM14] Marcin Andrychowicz, Stefan Dziembowski, Daniel Malinowski, and Lukasz Mazurek. Secure multiparty computations on bitcoin. In *Proc. of IEEE S&P*, pages 443–458. IEEE, 2014.

[ADMM16] Marcin Andrychowicz, Stefan Dziembowski, Daniel Malinowski, and Lukasz Mazurek. Secure multiparty computations on bitcoin. *Commun. ACM*, 59(4):76–84, 2016.

[AHM18] Sarah Azouvi, Alexander Hicks, and Steven J. Murdoch. Incentives in security protocols. In *Proc. of SPW*, pages 132–141, 2018.

[AIR01] William Aiello, Yuval Ishai, and Omer Reingold. Priced oblivious transfer: How to sell digital goods. In *Proc. of EUROCRYPT*, pages 119–135, 2001.

[Ale03] Michael Alekhnovich. More on average case vs approximation complexity. In *Proc. of IEEE FOCS*, pages 298–307, 2003.

[ALR+01] George-Marios Angeletos, David Laibson, Andrea Repetto, Jeremy Tobacman, and Stephen Weinberg. The hyperbolic consumption model: Calibration, simulation, and empirical evaluation. *J. of Econ. Persp.*, 15(3):47–68, 2001.

[ALZ13] Gilad Asharov, Yehuda Lindell, and Hila Zarosim. Fair and efficient secure multiparty computation with reputation systems. In *Proc. of ASIACRYPT*, pages 201–220, 2013.

[AMVA17] Giuseppe Ateniese, Bernardo Magri, Daniele Venturi, and Ewerton R. Andrade. Redactable blockchain - or - rewriting history in bitcoin and friends. In *Proc. of IEEE EuroS&P*, pages 111–126, 2017.

[AOZZ15] Joël Alwen, Rafail Ostrovsky, Hong-Sheng Zhou, and Vassilis Zikas. Incoercible multi-party computation and universally composable receipt-free voting. In *Proc. of CRYPTO*, pages 763–780, 2015.

[AW97] Martin Ahlbrecht and Martin Weber. An empirical study on intertemporal decision making under risk. *Mgmt. Sci.*, 43(6):813–826, 1997.

[B+07] Ken Binmore et al. *Playing for real: a text on game theory.* Oxford Uni. Press, 2007.

[BCD+09] Peter Bogetoft, Dan Lund Christensen, Ivan Damgård, Martin Geisler, Thomas P. Jakobsen, Mikkel Krøigaard, Janus Dam Nielsen, Jesper Buus Nielsen, Kurt Nielsen, Jakob Pagter, Michael I. Schwartzbach, and Tomas Toft. Secure multiparty computation goes live. In *Proc. of Fin. Crypto.*, pages 325–343, 2009.

[BCE+07] Mira Belenkiy, Melissa Chase, C. Christopher Erway, John Jannotti, Alptekin Küpçü, Anna Lysyanskaya, and Eric Rachlin. Making p2p accountable without losing privacy. In *Proc. of ACM WPES*, pages 31–40, 2007.

[BD18] Zvika Brakerski and Nico Döttling. Two-message statistically sender-private OT from LWE. In *Proc. of TCC*, pages 370–390, 2018.

[BDD18] Carsten Baum, Bernardo David, and Rafael Dowsley. Insured MPC: efficient secure multiparty computation with punishable abort. *IACR Cryptology ePrint Archive*, 2018:942, 2018.

[BDD20] Carsten Baum, Bernardo David, and Rafael Dowsley. Insured MPC: efficient secure computation with financial penalties. In *Proc. of Fin. Crypto.*, pages 404–420, 2020.

[BG17] Vitalik Buterin and Virgil Griffith. Casper the friendly finality gadget. *CoRR*, abs/1710.09437, 2017.

[BGJ+18] Saikrishna Badrinarayanan, Vipul Goyal, Abhishek Jain, Yael Tauman Kalai, Dakshita Khurana, and Amit Sahai. Promise zero knowledge and its applications to round optimal MPC. In *Proc. of CRYPTO*, pages 459–487, 2018.

[BGM+18] Christian Badertscher, Juan A. Garay, Ueli Maurer, Daniel Tschudi, and Vassilis Zikas. But why does it work? A rational protocol design treatment of bitcoin. In *Proc. of EUROCRYPT*, pages 34–65, 2018.

[BGW88] Michael Ben-Or, Shafi Goldwasser, and Avi Wigderson. Completeness theorems for non-cryptographic fault-tolerant distributed computation (extended abstract). In *Proc. of ACM STOC*, pages 1–10, 1988.

[BHP17] Zvika Brakerski, Shai Halevi, and Antigoni Polychroniadou. Four round secure computation without setup. In *Proc. of TCC*, pages 645–677, 2017.

[BIW15] Jeffrey R Brown, Zoran Ivković, and Scott Weisbenner. Empirical determinants of intertemporal choice. *J. of Financ. Econ.*, 116(3):473–486, 2015.

[BK14] Iddo Bentov and Ranjit Kumaresan. How to use bitcoin to design fair protocols. In *Proc. of CRYPTO*, pages 421–439, 2014.

[BKM17] Iddo Bentov, Ranjit Kumaresan, and Andrew Miller. Instantaneous decentralized poker. In *Proc. of ASIACRYPT*, pages 410–440, 2017.

[BL18] Fabrice Benhamouda and Huijia Lin. k-round multiparty computation from k-round oblivious transfer via garbled interactive circuits. In *Proc. of EUROCRYPT*, pages 500–532, 2018.

[BLS01] Dan Boneh, Ben Lynn, and Hovav Shacham. Short signatures from the weil pairing. In *Proc. of ASIACRYPT*, pages 514–532, 2001.

[BMTZ17] Christian Badertscher, Ueli Maurer, Daniel Tschudi, and Vassilis Zikas. Bitcoin as a transaction ledger: A composable treatment. In *Proc. of CRYPTO*, pages 324–356, 2017.

[BOSS20] Carsten Baum, Emmanuela Orsini, Peter Scholl, and Eduardo Soria-Vazquez. Efficient constant-round MPC with identifiable abort and public verifiability. In *Proc. of CRYPTO*, pages 562–592, 2020.

[BR96] Mihir Bellare and Phillip Rogaway. The exact security of digital signatures - how to sign with RSA and Rabin. In *Proc. of EUROCRYPT*, pages 399–416, 1996.

[BRY89] Uri Benzion, Amnon Rapoport, and Joseph Yagil. Discount rates inferred from decisions: An experimental study. *Mgmt. Sci.*, 35(3):270–284, 1989.

[BSMP91] Manuel Blum, Alfredo De Santis, Silvio Micali, and Giuseppe Persiano. Noninteractive zero-knowledge. *SIAM J. Comput.*, 20(6):1084–1118, 1991.

[Bur11] Andrew Burrows. *The Law of Restitution*. Oxford Uni. Press, 2011.

[BW] Bitcoin Wiki. Confirmation in bitcoin. `https://en.bitcoin.it/wiki/Confirmation`.

[BY92] Mihir Bellare and Moti Yung. Certifying cryptographic tools: The case of trapdoor permutations. In *Proc. of CRYPTO*, pages 442–460, 1992.

[Can00] Ran Canetti. Security and composition of multiparty cryptographic protocols. *J. Cryptol.*, 13(1):143–202, 2000.

[Can01] Ran Canetti. Universally composable security: A new paradigm for cryptographic protocols. In *Proc. of IEEE FOCS*, pages 136–145, 2001.

[Can20] Ran Canetti. Universally composable security. *J. ACM*, 67(5):28:1–28:94, 2020.

[CC00] Christian Cachin and Jan Camenisch. Optimistic fair secure computation. In *Proc. of CRYPTO*, pages 93–111, 2000.

[CC14] Christian Chamorro-Courtland. Demystifying the lowest intermediate balance rule: The legal principles governing the distribution of funds to beneficiaries of a commingled trust account for which a shortfall exists. *Banking and Finance Law Review*, 39, 2014.

[CF01] Ran Canetti and Marc Fischlin. Universally composable commitments. In *Proc. of CRYPTO*, pages 19–40, 2001.

[CFGN96] Ran Canetti, Uriel Feige, Oded Goldreich, and Moni Naor. Adaptively secure multi-party computation. In *Proc. of ACM STOC*, pages 639–648, 1996.

[CGH98] Ran Canetti, Oded Goldreich, and Shai Halevi. The random oracle methodology, revisited (preliminary version). In *Proc. of ACM STOC*, pages 209–218, 1998.

[CGH04] Ran Canetti, Oded Goldreich, and Shai Halevi. The random oracle methodology, revisited. *J. ACM*, 51(4):557–594, 2004.

[CGJ19] Arka Rai Choudhuri, Vipul Goyal, and Abhishek Jain. Founding secure computation on blockchains. In *Proc. of EUROCRYPT*, pages 351–380, 2019.

[CL18] Ran Canetti and Amit Lichtenberg. Certifying trapdoor permutations, revisited. In *Proc. of TCC*, pages 476–506, 2018.

[Cle86] Richard Cleve. Limits on the security of coin flips when half the processors are faulty (extended abstract). In *Proc. of ACM STOC*, pages 364–369, 1986.

[CLOS02] Ran Canetti, Yehuda Lindell, Rafail Ostrovsky, and Amit Sahai. Universally composable two-party and multi-party secure computation. In *Proc. of ACM STOC*, pages 494–503, 2002.

[CNS07] Jan Camenisch, Gregory Neven, and Abhi Shelat. Simulatable adaptive oblivious transfer. In *Proc. of EUROCRYPT*, pages 573–590, 2007.

[COSV17a] Michele Ciampi, Rafail Ostrovsky, Luisa Siniscalchi, and Ivan Visconti. Delayed-input non-malleable zero knowledge and multi-party coin tossing in four rounds. In *Proc. of TCC*, pages 711–742, 2017.

[COSV17b] Michele Ciampi, Rafail Ostrovsky, Luisa Siniscalchi, and Ivan Visconti. Round-optimal secure two-party computation from trapdoor permutations. In *Proc. of TCC*, pages 678–710, 2017.

[CPS18] T.-H. Hubert Chan, Rafael Pass, and Elaine Shi. Pala: A simple partially synchronous blockchain. *IACR Cryptology ePrint Archive*, 2018.

[DDL18] Bernardo David, Rafael Dowsley, and Mario Larangeira. Kaleidoscope: An efficient poker protocol with payment distribution and penalty enforcement. In *Proc. of Fin. Crypto.*, pages 500–519, 2018.

[DN00] Ivan Damgård and Jesper Buus Nielsen. Improved non-committing encryption schemes based on a general complexity assumption. In *Proc. of CRYPTO*, pages 432–450, 2000.

[DY05] Yevgeniy Dodis and Aleksandr Yampolskiy. A verifiable random function with short proofs and keys. In *Proc. of Pub. -Key Crypto.*, pages 416–431, 2005.

[EGL82] Shimon Even, Oded Goldreich, and Abraham Lempel. A randomized protocol for signing contracts. In *Proc. of CRYPTO*, pages 205–210, 1982.

[Eth] Etherscan Team. The ethereum average block time chart. `https://etherscan.io/chart/blocktime`. Accessed: 2020-06-11.

[Fic] Kelvin Fichter. Solidity bls signatures. `https://github.com/kfichter/solidity-bls`. Smart contract implementation of BLS Signatures.

[Gam84] Taher El Gamal. A public key cryptosystem and a signature scheme based on discrete logarithms. In *Proc. of CRYPTO*, pages 10–18, 1984.

[GG17] Rishab Goyal and Vipul Goyal. Overcoming cryptographic impossibility results using blockchains. In *Proc. of TCC*, pages 529–561, 2017.

[GH07] Matthew Green and Susan Hohenberger. Blind identity-based encryption and simulatable oblivious transfer. In *Proc. of ASIACRYPT*, pages 265–282, 2007.

[GHKL11] S. Dov Gordon, Carmit Hazay, Jonathan Katz, and Yehuda Lindell. Complete fairness in secure two-party computation. *J. of ACM*, 58(6):24:1–24:37, 2011.

[GHM+17] Yossi Gilad, Rotem Hemo, Silvio Micali, Georgios Vlachos, and Nickolai Zeldovich. Algorand: Scaling byzantine agreements for cryptocurrencies. In *Proc. of SOSP*, pages 51–68, 2017.

[GIM+10] S. Dov Gordon, Yuval Ishai, Tal Moran, Rafail Ostrovsky, and Amit Sahai. On complete primitives for fairness. In *Proc. of TCC*, pages 91–108, 2010.

[GK12] S. Dov Gordon and Jonathan Katz. Partial fairness in secure two-party computation. *J. Cryptology*, 25(1):14–40, 2012.

[GKL15] Juan A. Garay, Aggelos Kiayias, and Nikos Leonardos. The bitcoin backbone protocol: Analysis and applications. In *Proc. of EUROCRYPT*, pages 281–310, 2015.

[GKM+00] Yael Gertner, Sampath Kannan, Tal Malkin, Omer Reingold, and Mahesh Viswanathan. The relationship between public key encryption and oblivious transfer. In *Proc. of IEEE FOCS*, pages 325–335, 2000.

[GM84] Shafi Goldwasser and Silvio Micali. Probabilistic encryption. *J. Comput. Syst. Sci.*, 28(2):270–299, 1984.

[GMMM18] Sanjam Garg, Mohammad Mahmoody, Daniel Masny, and Izaak Meckler. On the round complexity of OT extension. In *Proc. of CRYPTO*, pages 545–574, 2018.

[GMPP16] Sanjam Garg, Pratyay Mukherjee, Omkant Pandey, and Antigoni Polychroniadou. The exact round complexity of secure computation. In *Proc. of EUROCRYPT*, pages 448–476, 2016.

[GMPY11] Juan A. Garay, Philip D. MacKenzie, Manoj Prabhakaran, and Ke Yang. Resource fairness and composability of cryptographic protocols. *J. Cryptology*, 24(4):615–658, 2011.

[GMW87] Oded Goldreich, Silvio Micali, and Avi Wigderson. How to play any mental game or A completeness theorem for protocols with honest majority. In *Proc. of ACM STOC*, pages 218–229, 1987.

[GMW91] Oded Goldreich, Silvio Micali, and Avi Wigderson. Proofs that yield nothing but their validity for all languages in NP have zero-knowledge proof systems. *J. ACM*, 38(3):691–729, 1991.

[GS09] Vipul Goyal and Amit Sahai. Resettably secure computation. In *Proc. of EUROCRYPT*, pages 54–71, 2009.

[GS18] Sanjam Garg and Akshayaram Srinivasan. Two-round multiparty secure computation from minimal assumptions. In *Proc. of EUROCRYPT*, pages 468–499, 2018.

[Hai08] Iftach Haitner. Semi-honest to malicious oblivious transfer - the black-box way. In *Proc. of TCC*, pages 412–426, 2008.

[Hat09] Robert Hatch. Reforming the Murky Depths of Wall Street: Putting the Spotlight on the Security and Exchange Commission's Regulatory Proposal Concerning Dark Pools of Liquidity. *The George Washington Law Review*, 78:1032, 2009.

[HHPV18] Shai Halevi, Carmit Hazay, Antigoni Polychroniadou, and Muthuramakrishnan Venkitasubramaniam. Round-optimal secure multi-party computation. In *Proc. of CRYPTO*, pages 488–520, 2018.

[HIK+11] Iftach Haitner, Yuval Ishai, Eyal Kushilevitz, Yehuda Lindell, and Erez Petrank. Black-box constructions of protocols for secure computation. *SIAM J. Comput.*, 40(2):225–266, 2011.

[HILL99] Johan Håstad, Russell Impagliazzo, Leonid A. Levin, and Michael Luby. A pseudorandom generator from any one-way function. *SIAM J. Comput.*, 28(4):1364–1396, 1999.

[HK09] Dennis Hofheinz and Eike Kiltz. The group of signed quadratic residues and applications. In *Proc. of CRYPTO*, pages 637–653, 2009.

[HL10] Carmit Hazay and Yehuda Lindell. *Efficient Secure Two-Party Protocols - Techniques and Constructions*. IS&C. Springer, 2010.

[IKK+11] Yuval Ishai, Jonathan Katz, Eyal Kushilevitz, Yehuda Lindell, and Erez Petrank. On achieving the "best of both worlds" in secure multiparty computation. *SIAM J. on Comp.*, 40(1):122–141, 2011.

[IKO+11] Yuval Ishai, Eyal Kushilevitz, Rafail Ostrovsky, Manoj Prabhakaran, and Amit Sahai. Efficient non-interactive secure computation. In *Proc. of EUROCRYPT*, pages 406–425, 2011.

[IOZ14] Yuval Ishai, Rafail Ostrovsky, and Vassilis Zikas. Secure multi-party computation with identifiable abort. In *Proc. of CRYPTO*, volume 8617, pages 369–386. Springer, 2014.

[IPS08] Yuval Ishai, Manoj Prabhakaran, and Amit Sahai. Founding cryptography on oblivious transfer – Efficiently. In *Proc. of CRYPTO*, pages 572–591, 2008.

[IR88] Russell Impagliazzo and Steven Rudich. Limits on the provable consequences of one-way permutations. In *Proc. of CRYPTO*, pages 8–26, 1988.

[JS07] Stanislaw Jarecki and Vitaly Shmatikov. Efficient two-party secure computation on committed inputs. In *Proc. of EUROCRYPT*, pages 97–114, 2007.

[Kal05] Yael Tauman Kalai. Smooth projective hashing and two-message oblivious transfer. In *Proc. of EUROCRYPT*, pages 78–95, 2005.

[KB14] Ranjit Kumaresan and Iddo Bentov. How to use bitcoin to incentivize correct computations. In *Proc. of ACM CCS*, pages 30–41, 2014.

[KB16] Ranjit Kumaresan and Iddo Bentov. Amortizing secure computation with penalties. In *Proc. of ACM CCS*, pages 418–429, 2016.

[Kil88] Joe Kilian. Founding cryptography on oblivious transfer. In *Proc. of ACM STOC*, pages 20–31, 1988.

[Kil92] Joe Kilian. A note on efficient zero-knowledge proofs and arguments (extended abstract). In *Proc. of ACM STOC*, pages 723–732, 1992.

[KKM12] Saqib A. Kakvi, Eike Kiltz, and Alexander May. Certifying RSA. In *Proc. of ASIACRYPT*, pages 404–414, 2012.

[KL12] Alptekin Küpçü and Anna Lysyanskaya. Usable optimistic fair exchange. *Computer Networks*, 56(1):50–63, 2012.

[KMB15a] Ranjit Kumaresan, Tal Moran, and Iddo Bentov. How to use bitcoin to play decentralized poker. In *Proc. of ACM CCS*, pages 195–206, 2015.

[KMB15b] Ranjit Kumaresan, Tal Moran, and Iddo Bentov. How to use bitcoin to play decentralized poker. In *Proc. of ACM CCS*, pages 195–206, 2015.

[KMS+16] Ahmed E. Kosba, Andrew Miller, Elaine Shi, Zikai Wen, and Charalampos Papamanthou. Hawk: The blockchain model of cryptography and privacy-preserving smart contracts. In *Proc. of IEEE S&P*, pages 839–858, 2016.

[KO97] Eyal Kushilevitz and Rafail Ostrovsky. Replication is NOT needed: SINGLE database, computationally-private information retrieval. In *Proc. of IEEE FOCS*, pages 364–373, 1997.

[KO04] Jonathan Katz and Rafail Ostrovsky. Round-optimal secure two-party computation. In *Proc. of CRYPTO*, pages 335–354, 2004.

[KVV16] Ranjit Kumaresan, Vinod Vaikuntanathan, and Prashant Nalini Vasudevan. Improvements to secure computation with penalties. In *Proc. of ACM CCS*, pages 406–417, 2016.

[KZZ16] Aggelos Kiayias, Hong-Sheng Zhou, and Vassilis Zikas. Fair and robust multi-party computation using a global transaction ledger. In *Proc. of EUROCRYPT*, pages 705–734, 2016.

[Lin08] Yehuda Lindell. Efficient fully-simulatable oblivious transfer. *Chicago J. Theor. Comput. Sci.*, 2008, 2008.

[Lin09] Yehuda Lindell. Legally enforceable fairness in secure two-party communication. *Chicago J. Theor. Comput. Sci.*, 2009, 2009.

[LMRS04] Anna Lysyanskaya, Silvio Micali, Leonid Reyzin, and Hovav Shacham. Sequential aggregate signatures from trapdoor permutations. In *Proc. of EUROCRYPT*, pages 74–90, 2004.

[LPS10] Vadim Lyubashevsky, Adriana Palacio, and Gil Segev. Public-key cryptographic primitives provably as secure as subset sum. In *Proc. of TCC*, pages 382–400, 2010.

[LVM16] Boram Lee and Yulia Veld-Merkoulova. Myopic loss aversion and stock investments: An empirical study of private investors. *J. of Banking & Fin.*, 70:235–246, 2016.

[Lys02] Anna Lysyanskaya. Unique signatures and verifiable random functions from the DH-DDH separation. In *Proc. of CRYPTO*, pages 597–612, 2002.

[LZ13] Yehuda Lindell and Hila Zarosim. On the feasibility of extending oblivious transfer. In *Proc. of TCC*, pages 519–538, 2013.

[LZ18] Yehuda Lindell and Hila Zarosim. On the feasibility of extending oblivious transfer. *J. Cryptology*, 31(3):737–773, 2018.

[MB17] Andrew Miller and Iddo Bentov. Zero-collateral lotteries in bitcoin and ethereum. In *Proc. of EuroS&P Workshops*, pages 4–13, 2017.

[MMNT19] Bernardo Magri, Christian Matt, Jesper Buus Nielsen, and Daniel Tschudi. Afgjort: A partially synchronous finality layer for blockchains. *IACR Cryptology ePrint Archive*, 2019.

[MNN+18] F. Massacci, C. N. Ngo, J. Nie, D. Venturi, and J. Williams. Futuresmex: Secure, distributed futures market exchange. In *Proc. of IEEE SSP*, pages 453–471, 2018.

[MNS16] Tal Moran, Moni Naor, and Gil Segev. An optimally fair coin toss. *J. Cryptology*, 29(3):491–513, 2016.

[MRV99] Silvio Micali, Michael O. Rabin, and Salil P. Vadhan. Verifiable random functions. In *Proc. of IEEE FOCS*, pages 120–130, 1999.

[Nak19] Satoshi Nakamoto. Bitcoin: A peer-to-peer electronic cash system. Technical report, Manubot, 2019.

[Nao91] Moni Naor. Bit commitment using pseudorandomness. *J. Cryptology*, 4(2):151–158, 1991.

[Nie02] Jesper Buus Nielsen. Separating random oracle proofs from complexity theoretic proofs: The non-committing encryption case. In *Proc. of CRYPTO*, pages 111–126, 2002.

[NP01] Moni Naor and Benny Pinkas. Efficient oblivious transfer protocols. In *Proc. of SODA*, pages 448–457, 2001.

[NP05] Moni Naor and Benny Pinkas. Computationally secure oblivious transfer. *J. Cryptology*, 18(1):1–35, 2005.

[ORS15] Rafail Ostrovsky, Silas Richelson, and Alessandra Scafuro. Round-optimal black-box two-party computation. In *Proc. of CRYPTO*, pages 339–358, 2015.

[PS17] Rafael Pass and Elaine Shi. Hybrid consensus: Efficient consensus in the permissionless model. In *Proc. of DISC*, pages 39:1–39:16, 2017.

[PS18] Rafael Pass and Elaine Shi. Thunderella: Blockchains with optimistic instant confirmation. In *Proc. of EUROCRYPT*, pages 3–33, 2018.

[PSS17] Rafael Pass, Lior Seeman, and Abhi Shelat. Analysis of the blockchain protocol in asynchronous networks. In *Proc. of EUROCRYPT*, pages 643–673, 2017.

[PVW08] Chris Peikert, Vinod Vaikuntanathan, and Brent Waters. A framework for efficient and composable oblivious transfer. In *Proc. of CRYPTO*, pages 554–571, 2008.

[Rab81] Michael O. Rabin. How to exchange secrets by oblivious transfer. Technical report, Harvard University, 1981.

[Reg05] Oded Regev. On lattices, learning with errors, random linear codes, and cryptography. In *Proc. of ACM STOC*, pages 84–93, 2005.

[RT] Randao Team. Randao: A dao working as rng of ethereum. `https://github.com/randao/randao`.

[SSV19] Alessandra Scafuro, Luisa Siniscalchi, and Ivan Visconti. Publicly verifiable proofs from blockchains. In *Proc. of Pub. -Key Crypto.*, pages 374–401, 2019.

[Yao82] Andrew Chi-Chih Yao. Protocols for secure computations (extended abstract). In *Proc. of IEEE FOCS*, pages 160–164, 1982.

[Yao86] Andrew Chi-Chih Yao. How to generate and exchange secrets (extended abstract). In *Proc. of IEEE FOCS*, pages 162–167, 1986.