Resource-Restricted Cryptography: Honest-Majority MPC from a CRS (and No Broadcast)

Juan A. Garay [*] Aggelos Kiayias[†] Rafail M. Ostrovsky[‡] Giorgos Panagiotakos[§]
Vassilis Zikas[¶]

October 30, 2019

Abstract

Traditional bounds on synchronous Byzantine agreement (BA) and secure multi-party computation (MPC) establish that in absence of a private-coin correlated-randomness setup, such as a PKI, protocols can tolerate up to $t < n/3$ of the parties being malicious. The introduction of "Nakamoto style" consensus, based on Proof-of-Work (PoW) blockchains, put forth a somewhat different flavor of BA, showing that even a majority of corrupted parties can be tolerated as long as the majority of the computation resources remain at honest hands. This assumption on honest majority of some resource was also extended to other resources such as stake, space, etc., upon which blockchains achieving Nakamoto-style consensus were built that violated the $t < n/3$ bound in terms of number of party corruptions. The above state of affairs begs the question of whether the seeming mismatch is due to different goals and models, or whether the resource-restricting paradigm can be generically used to circumvent the $n/3$ lower bound.

In this work we study this question and formally demonstrate how the above paradigm changes the rules of the game in cryptographic definitions. First, we abstract the core properties that the resource-restricting paradigm offers by means of a functionality *wrapper*, in the UC framework, which when applied to a standard point-to-point network restricts the ability (of the adversary) to send new messages. We show that such a wrapped network can be implemented using the resource-restricting paradigm—concretely, using PoWs and honest majority of computing power—and that the traditional $t < n/3$ impossibility results fail when the parties have access to such a network.

We then present constructions for BA and MPC, which given access to such a network tolerate $t < n/2$ corruptions without assuming a private correlated randomness setup, but merely a *fresh* Common Reference String (CRS)—i.e., a CRS which becomes available to the parties at the same time as to the adversary. We also show how to remove this freshness assumption by leveraging the power of a random oracle. Our MPC protocol achieves the standard notion of MPC security, where parties might have dedicated roles, as is for example the case in Oblivious Transfer protocols. This is in contrast to existing solutions basing MPC on PoWs, which associate roles to pseudonyms but do not link these pseudonyms with the actual parties.

[*] Texas A&M University, garay@cse.tamu.edu.

[†] University of Edinburgh and IOHK, akiayias@inf.ed.ac.uk. Research party supported by H2020 Grant 780477, PRIViLEDGE.

[‡] University of California, Los Angeles, rafail@cs.ucla.edu. Supported by DARPA SPAWAR contract N66001-15-C-4065.

[§] University of Edinburgh, giorgos.pan@inf.ed.ac.uk.

[¶] University of Edinburgh and IOHK, vzikas@inf.ed.ac.uk.

Contents

1 Introduction

Byzantine agreement (BA), introduced by Lamport, Shostak, and Pease [26], is a fundamental primitive in distributed computing and is at the core of many secure multi-party computation (MPC) protocols. The problem comes in two main flavors, *Consensus* and *Broadcast*—although a number of relaxations have also been proposed. Consensus considers a set of n parties $\mathcal{P} = \{P_1, \ldots, P_n\}$ each of whom has an input x_i, and who wish to agree on an output y (Consistency) such that if $x_i = x$ for all honest parties then $y = x$ (Validity), despite the potentially malicious behavior of up to t of them. In the Broadcast version, on the other hand, only a single party, often called the *sender* has an input x_s, and the goal is to agree on an output y (Consistency) which, when the sender is honest equals x (Validity).

The traditional setting in which the problem was introduced and investigated considers synchronous communication and protocol execution. In a nutshell, this means that the protocol advances in rounds such that: (1) parties have a consistent view of the current round—i.e., no party advances to round $\rho + 1$ before all other parties are finished with their round ρ instructions; and (2) all messages sent in round ρ are delivered to their respective recipients by the beginning of round $\rho + 1$. Furthermore, the underlying communication network is a complete point-to-point authenticated channels network, where every pair (P_i, P_j) of parties is connected by a channel, such that when P_j receives a message on this channel it knows it was indeed sent by P_i (or the adversary, in case P_i is corrupted). We refer to the above setting as the *(standard) LSP setting.*

In this model, Lamport *et al.* proved that there exists no Consensus or Broadcast protocol which can tolerate $t \geq n/3$ Byzantine parties, i.e., parties controlled by a (central) active and malicious adversary. The original formulation considered perfect security (i.e., information-theoretic security with zero error probability) and no correlated randomness shared among the parties.[1] This impossibility result was later extended by Borcherding [7] to computational security—i.e., it was proved to hold even under strong computational assumptions, such as one-way permutations.[2] Furthermore, it applies even when the point-to-point channels used by the parties are secure, i.e., both authenticated and private, and even if we assume an arbitrary public correlated randomness setup and/or a random oracle (RO).[3] (A *public* correlated randomness setup can be viewed as a functionality which samples a string and distributes it to all parties, e.g, a *common reference string* (CRS). This is in contrast to a *private* correlated randomness setup which might keep part of the sampled string private and distribute different parts of it to different parties, e.g., a PKI. For ease of reference we state the above as a corollary:

Corollary 1 (Strong $t \geq n/3$ impossibility [7]). *In the synchronous point-to-point channels setting, there exists no Broadcast protocol tolerating $t \geq n/3$ corrupted parties. The statement holds both in the authenticated and in the secure channels setting, both for unconditional adversaries and assuming (even enhanced) trapdoor permutations, and even assuming an arbitrary public correlated randomness setup and/or a random oracle.*

The effect of BA lower bounds on MPC. MPC allows a set of parties to compute an arbitrary function of their (potentially private) inputs in a secure way even in the presence of an adversary. Ben-Or, Goldwasser and Wigderson [5] presented a protocol which computes any function with perfect

[1] Lamport *et al.* also considered the case of "signed messages." The information-theoretic setting was referred to as the "oral messages" setting.

[2] The original result by Borcherding just treats the case of assumptions sufficient for the existence of existentially unforgeable signatures, but it can easily be extended to arbitrary cryptographic hardness assumptions.

[3] As usual, the implicit assumption here is that no party of adversary can query the RO more times than its running time.

security in the synchronous setting while tolerating $t < n/3$ malicious parties assuming the parties have access to a complete network of instant delivery point-to-point *secure*—i.e., authenticated and private—channels (we shall refer to this model as the *BGW communication model*). The lower bound holds even if a Broadcast channel—i.e., an ideal primitive guaranteeing the input/output properties of Broadcast—is available to the parties. Rabin and Ben-Or [29] proved that if we allow for a negligible error probability and assume broadcast, then there exists a general MPC protocol tolerating up to $t < n/2$ of the parties being corrupted, even if the adversary is computationally unbounded.

Observe, however, that just allowing negligible error probability is not sufficient for circumventing the $t < n/3$ barrier. Indeed, it is straightforward to verify that fully secure MPC as considered in [21, 29]—with fairness and guaranteed output delivery—against malicious/Byzantine adversaries implies Broadcast: Just consider the function which takes input only from a designated party, the sender, and outputs it to everyone.[4] In fact, using the above observation and Corollary 1 directly implies that $t < n/3$ is tight even assuming a computational adversary, secure point-to-point channels, an arbitrary public correlated randomness setup, e.g., a CRS, and/or a random oracle.

The public-key infrastructure (PKI) model. With the exception of perfect security[5], the above landscape changes if we assume a *private correlated randomness setup*, such as a PKI. Indeed, in this case Dolev and Strong [16] proved that assuming a PKI and intractability assumptions implying existentially unforgeable digital signatures (e.g., one way functions) Broadcast tolerating arbitrarily many (i.e., $t < n$) malicious corruptions is possible. We refer to this protocol as *Dolev-Strong Broadcast.* In fact, as shown later by Pfitzmann and Waidner [28], by assuming more complicated correlations—often referred to as a setup for *information-theoretic (pseudo-)signatures*—it is possible to obtain an unconditionally (i.e., information-theoretically) secure protocol for Broadcast tolerating any corrupted minority. Clearly, by plugging the above constructions in [29], we obtain a computationally or even i.t. secure MPC protocol tolerating any dishonest minority in the private correlated randomness setting. Recall that this task was impossible for honest majorities in the public correlated randomness setting.

The blockchain revolution. The introduction and systematic study of blockchains in the *permissionless* setting, such as the Bitcoin blockchain, demonstrated how Consensus and Broadcast can be reached even in settings where a majority of the participants might be adversarial (as long as the majority of the computing power remains honest) and even without a private correlated randomness setup. And although it was proven that such constructions work under the different assumption of honest-majority computing power, a confusion still remained driven mainly by the fact that the investigation of the type of consensus achieved by Bitcoin ("Nakamoto consensus") considered more involved models that closer capture its execution parameters (e.g., "partial synchrony" [17]), and that the Bitcoin backbone protocol [19, 27] was shown to achieve *eventual* consensus, a property closer to the traditional state-machine replication problem from distributed computing [30][6]. In fact, similar approaches were also used for alternative blockchains that relied on assumptions about restricting other resource, such as for example a majority of honest stake ("proof of stake"—PoS) [6, 25, 20], a majority of honest space [25, 15, 3, 6, 13], etc., which were however also analyzed in more complex network settings.

[4] There are some delicate matters to handle when capturing Broadcast as MPC, which will become relevant for out results, but for clarity we defer discussing them for when they are needed.

[5] Since perfect security allows no error probability, a setup does not help.

[6] Although it was also shown in [19] how to achieve the standard version of Consensus, as defined above, but in a way radically different from the existing protocols.

The resource-restricting paradigm. We will use this general term to refer to all the above approaches. Thus, an intriguing question remained:

> *Does Corollary 1 still apply to the standard LSP model (of instant delivery authenticated channels and full synchrony) under the resource-restricting paradigm?*

In this work we first answer this question in the negative by abstracting the essence of the above resource-restricting paradigm as an access restriction on the underlying communication network. Intuitively, the assumption of restricting (the adversary's access to) the relative resource can be captured by disallowing any party—and in particular any adversarial party—to send unboundedly many more new messages than any other party. To avoid ambiguity and allow using the related assumption in higher level constructions, we choose to work on Canetti's Universal Composition framework [9]. In particular, we describe the assumption induced by restricting the resources available to the adversary by means of a functionality *wrapper*, which wraps a communication network and restricts the ability of parties (or the adversary) to send new messages through this network.

We then demonstrate how our wrapper, when applied to the standard instant-delivery synchronous network, makes it impossible for the adversary to launch the attack from [7]. In particular, the classical impossibilities (or even their extension stated in Corollary 1) in the same model as the one they were proven, and with the required properties from the target primitive, do not apply to protocols in this new restricted network.

In order to prove that our network restriction is an appropriate abstraction of the mechanisms implied by the resource-restricting paradigm, we focus on the case of proofs of work (PoW) and prove how to implement the wrapped LSP-style network from a public correlated randomness setup (in particular, any high min-entropy CRS) and an access-restricted random oracle. Concretely, along the lines of the composable analyses of Bitcoin [4], we capture the assumption of honest majority of hashing power by means of a wrapped RO, which allows each party (honest or corrupted) at most q queries per communication round (cf. [19]) for any given q (polynomial in the security parameter).[7]An important consideration of our transformation is the need for a *freshness* property on the assumed CRS. Specifically, our protocol for realizing the wrapped network assumes that the adversary gets access to the CRS at the same time as honest parties do (and crucially relies on this fact). Intuitively, the reason is that our protocol will rely on PoW-style hash puzzles in order to restrict the ability of the adversary to create many new valid messages. Clearly, if the adversary has access to the initial CRS—which will play the role of the genesis block—way before the honest parties do, then he can start potentially precomputing valid messages thus making the implementation of communication restriction infeasible.

We note that such freshness of the CRS might be considered a non-standard assumption and seems relevant only in combination with the resource-restricting paradigm. Nonetheless, in Section 6, we discuss how this freshness can be replaced using PoWs on challenges exchanged between parties, along the lines of [1]. The absence of freshness yields a somewhat relaxed wrapper which offers analogous restrictions as our original wrapper, but guarantees only limited transferability of the messages sent, and is not as strict towards the adversary as our original one (i.e., adversarial messages can be transfered more times than honest ones). Still, as we argue, this relaxed wrapper is sufficient for obtaining all the positive results in this work.

The above sheds light on the seemingly confusing landscape, but leaves open the question of how powerful the new assumption of the resource-restricting wrapper (and hence the resource-restricting

[7] The wrapper actually puts a restriction to adversarial parties as honest parties can be restricted by their protocol (cf. [4]).

paradigm in general) is. In particular, although the above demonstrates that the resource-restricting paradigm allows to circumvent the limitation of Corollary 1, it still leaves open the question:

> *Does the resource-restricting methodology allow for fully secure MPC in the public correlated randomness model, and if so, under what assumptions on the number of corrupted parties?*

We investigate the question of whether we can obtain honest majority MPC in this setting, and answer it in the affirmative. (Recall that without the resource-restricting methodology and associated assumptions this is impossible since MPC implied Broadcast.) Note that a consensus impossibility due to Fitzi [18] proved that the $t < n/2$ bound is actually necessary for Consensus in the standard LSP communication model. And the lower bound holds even if we assume a broadcast primitive. In fact, by a simple inspection of the results one can observe that the underlying proof uses only honest strategies (for different selections of corruption sets) and therefore applies even under the resource-restricting paradigm—where, as above, this paradigm is captured by wrapping the network with our communication-restricting wrapper.

Towards the feasibility goal, we provide a protocol which allows us to establish a PKI assuming only our resource-restricted (wrapped) LSP network and one-way functions (or any other assumption which allows for existentially unforgeable signatures). More specifically, we show that our PKI establishment mechanism implements the key registration functionality $\mathcal{F}_{\text{REG}}$ from [10]. Our protocol is inspired by the protocol of Andrychowicz and Dziembowski [1]. Their protocol, however, achieved a non-standard notion of MPC in which inputs are associated to public-keys/pseudonyms. In particular, in the standard MPC setting, computing a function $f(x_1, \ldots, x_n)$ among parties $P_1, \ldots, P_n$ means having each P_i contribute input x_i and output $f(x_1, \ldots, x_n)$—this is reflected both in in the original definitions of MPC [31, 21] and in the UC SFE functionality $\mathcal{F}_{\text{SFE}}$ [9] and the corresponding standalone evaluation experiment from [8]. Instead, in the MPC evaluation from [1], every party P_i is represented by a pseudonym j_i, which is not necessarily equal to i and where the mapping between i and j_i is unknown to the honest participants.[8] Then the party contributing the ℓth input to the computation of f is P_i such that $j_i = \ell$. This evaluation paradigm was termed *pseudonymous MPC* in [24].

It is not hard to see, however, that the above evaluation paradigm makes the corresponding solution inapplicable to classical scenarios where MPC would be applied, where parties have distinguished roles. Examples include decentralized auctions—where the auctioneer should not bid—and asymmetric functionalities such as oblivious transfer. We note in passing that the above relaxation of traditional MPC guarantees seems inherent in the permissionless peer-to-peer setting setting of [1, 24]. Instead, our protocol adapts the techniques from [1] in a white-box manner to leverage the authenticity of our underlying communication network—recall that our protocol is in the (wrapped) BGW communication setting—in order to ensure that the registered public keys are publicly linked to their respective owners. This allows us to evaluate the standard MPC functionality.

Getting from an implementation of $\mathcal{F}_{\text{REG}}$ where the keys are linked to their owners to standard MPC is then fairly straightforward by using the modularity of the UC framework. As proved in [10], $\mathcal{F}_{\text{REG}}$ can be used to realize the certified signature functionality (aka *certification functionality*) $\mathcal{F}_{CERT}$ which, in turn, can be used to realize a Broadcast functionality against even adaptive adversaries [22]. By plugging this functionality into the honest-majority protocol (compiler) by Cramer *et al.* [14]—an adaptation of the protocol from [29] to tolerate adaptive corruptions—we obtain an MPC protocol which is adaptively secure.

Organization of the paper. In Section 2 we discuss our model. In Section 3 we introduce our wrapper-based abstraction of the resource-restricting paradigm and demonstrate how the impossibility from

[8] In fact, $(j_1, \ldots, j_n)$ is a permutation of $(1, \ldots, n)$

Corollary 1 fails when parties can use it. Section 4 presents our implementation of this wrapper from PoWs and a fresh CRS, and Section 5 discusses how to use it to obtain certified digital signatures and MPC. Finally in Section 6 we discuss how to remove the freshness assumption by leveraging PoWs.

2 Model

To allow for a modular treatment and ensure universal composition of our results, we will work in Canetti's UC model [8]. We assume some familiarity of the reader with UC but we will restrict the properties we use to those that are satisfied by any composable security framework. In fact, technically speaking, our underlying framework is the UC with global setups (GUC) [11], as we aim to accurately capture a global notion of time (see below). Nonetheless, the low level technicalities of the GUC framework do not affect our arguments and the reader can treat our proofs as standard UC proofs.

Parties, functionalities, and the adversary and environment are (instances of) interactive Turing machines (ITMs) running in probabilistic polynomial time (PPT). We prove our statements for a static active adversary; however, the static restriction is only for simplicity as our proofs can be directly extended to handle adaptive corruptions. In (G)UC, security is defined via the standard simulation paradigm: In a nutshell, a protocol π realizes a functionality $\mathcal{F}$ (in UC, this is described as emulation of the dummy/ideal $\mathcal{F}$-hybrid protocol ϕ) if for any adversary attacking π there exists a simulator attacking ϕ making the executions of the two protocols indistinguishable in the eyes of any external environment. Note that π might (and in our cases will, as discussed below) have access to its own hybrid functionalities.

Synchrony. We adopt the global clock version of the synchronous UC model by Katz *et al.* [23] as described in [4]. Concretely, we assume that parties have access to a global clock functionality which allows them to advance rounds at the same pace. For generality, we will allow the clock to have a dynamic party set, as in [4].

Global Functionality $\mathcal{G}_{\textsc{clock}}$

The functionality manages the set $\mathcal{P}$ of registered identities, i.e, parties $P = (\mathsf{pid}, \mathsf{sid})$. It also manages the set F of registered functionalities (together with their session identifier). Initially, $\mathcal{P} = \emptyset$ and $F = \emptyset$. For each session sid the clock maintains a variable τ_{sid}. For each identity $P = (\mathsf{pid}, \mathsf{sid}) \in \mathcal{P}$ it manages variable d_P. For each pair $(\mathcal{F}, \mathsf{sid}) \in F$ it manages variable $d_{(\mathcal{F},\mathsf{sid})}$ (all integer variables are initially set to 0).

Synchronization:

- Upon receiving $(\textsc{clock-update}, \mathsf{sid}_C)$ from some party $P \in \mathcal{P}$ set $d_P := 1$; execute *Round-Update* and forward $(\textsc{clock-update}, \mathsf{sid}_C, P)$ to $\mathcal{A}$.
- Upon receiving $(\textsc{clock-update}, \mathsf{sid}_C)$ from some functionality $\mathcal{F} \in F$ in a session sid such that $(\mathcal{F}, \mathsf{sid}) \in F$, set $d_{(\mathcal{F},\mathsf{sid})} = 1$, execute *Round-Update* and return $(\textsc{clock-update}, \mathsf{sid}_C, \mathcal{F})$ to $\mathcal{A}$.
- Upon receiving $(\textsc{clock-read}, \mathsf{sid}_C)$ from any participant (including the environment, the adversary, or any ideal—shared or local—functionality) return $(\textsc{clock-read}, \mathsf{sid}_C, \tau_{\mathsf{sid}})$ to the requestor.

Procedure Round-Update: For each session sid do: If $d_{(\mathcal{F},\mathsf{sid})} = 1$ for all $\mathcal{F} \in F$ and $d_P = 1$ for all honest $P = (\cdot, \mathsf{sid})$ in $\mathcal{P}$, then set $\tau_{\mathsf{sid}} = \tau_{\mathsf{sid}} + 1$ and reset $d_{\mathcal{F}} = 0$ and $d_P = 0$ for all parties $P = (\cdot, \mathsf{sid}) \in \mathcal{P}$.

Communication network. We capture point-to-point authenticated communication, modeling the LSP channels in UC, by means of a multi-party multi-use version of the authenticated channel functionality with instant delivery along the lines of [4]. (The original network from [4] had bounded delay; hence here we need to set this bound to 1.) Note that in this network once an honest party P_i inserts a message to be sent to P_j, the message is buffered, and it is delivered after at most Δ attempts from the receiver (here $\Delta = 1$). Syntactically, we allow the simulator to query the network and learn if a buffered message was received by the respective receiver. This step—despite being redundant in most cases as the simulator should be able to defer this fact by observing the activations forwarded to him—is not only an intuitive addition, as it captures that the adversary is aware of delivery of message, but will also simplify the protocol description and simulation. For completeness, we include the authenticated network functionality below.

Note that the BGW-style secure point-to-point network functionality can be trivially derived by the authenticated one by replacing in the message $(\textsc{sent}, \mathsf{sid}, m, P_i, P_j, mid)$ which the adversary receives upon some m being inserted to the network, the value of m by $\perp$ (of by $|m|$ if this is implemented by standard encryption).

Functionality $\mathcal{F}_{\text{AUTH}}$

The functionality is parameterized by a set of possible senders and receivers, denoted by $\mathcal{P}$, a list $\vec{M}$, and integer variables of the form D_z, where $z \in \{0,1\}^*$, that are dynamically created. For every party $P \in \mathcal{P}$ it maintains a fetch counter f_P. Initially, $\vec{M} := \emptyset$ and $f_P := 0$, for every $P \in \mathcal{P}$.

- Upon receiving $(\textsc{send}, \mathsf{sid}, m, P_j)$ from $P_i \in \mathcal{P}$, set $D_{mid} := 1$ and $\vec{M} = \vec{M}||(m, P_i, P_j, mid)$, where mid is a unique message-ID, and send $(\textsc{sent}, \mathsf{sid}, m, P_i, P_j, mid)$ to $\mathcal{A}$.
- Upon receiving $(\textsc{fetch}, \mathsf{sid})$ from some honest party $P_j \in \mathcal{P}$, increment f_P by 1, set $M' = \emptyset$, and do the following:
  1. For all tuples $(m, P_i, P_j, mid) \in \vec{M}$, set $D_{mid} := D_{mid} - 1$,
  2. for all tuples $(m, P_i, P_j, mid) \in \vec{M}$, where $D_{mid} \leq 0$, delete (m, P_i, P_j, mid) from $\vec{M}$, and add (m, P_i) to M'.
  3. Send $(\textsc{sent}, \mathsf{sid}, M')$ to P_j.
- Upon receiving $(\textsc{fetch-requests}, \mathsf{sid}, P)$ from $\mathcal{A}$, output $(\textsc{fetch-requests}, \mathsf{sid}, f_P)$.

The random oracle functionality. As is typical in the proof-of-work literature, we will abstract puzzle-friendly hash functions by means of a random oracle functionality.

Functionality $\mathcal{F}_{\text{RO}}$

The functionality is parameterized by a security parameter λ and a set of parties $\mathcal{P}$. It maintains a (dynamically updatable) map H that is initially empty.

- Upon receiving $(\textsc{Eval}, \mathsf{sid}, x)$ from some party $P \in \mathcal{P}$ (or from $\mathcal{A}$ on behalf of a corrupted P), do the following:
  1. If $H[x] = \perp$, sample a value y uniformly at random from $\{0,1\}^\lambda$, and set $H[x] := y$.
  2. Return $(\textsc{Eval}, \mathsf{sid}, x, H[x])$ to the requestor.

Furthermore, following [4], we will use the wrapper to capture the assumption that no party gets more than q queries to the RO per round. This wrapper in combination with the honest majority of parties captures the assumption that the adversary does not control a majority of the systems hashing power.

Wrapper Functionality $\mathcal{W}^q_{\text{RO}}(\mathcal{F})$

The wrapper functionality is parameterized by a set of parties $\mathcal{P}$, and an upper bound q which restricts the $\mathcal{F}$-evaluations of each corrupted party per round. (To keep track of rounds the functionality registers with the global clock $\mathcal{G}_{\text{CLOCK}}$.) The functionality manages the variable τ and the current set of corrupted miners $\mathcal{P}$. For each party $P \in \mathcal{P}$ it manages variables q_P. Initially, $\tau = 0$.

General:

- The wrapper stops the interaction with the adversary as soon as the adversary tries to exceed its budget of q queries per corrupted party.

Relaying inputs to the random oracle:

- Upon receiving $(\text{EVAL}, \mathsf{sid}, x)$ from $\mathcal{A}$ on behalf of a corrupted party $P \in \mathcal{P}'$, then first execute *Round Reset*. Then, set $q_P := q_P + 1$ and only if $q_P \leq q$ forward the request to $\mathcal{F}_{\text{RO}}$ and return to $\mathcal{A}$ whatever $\mathcal{F}_{\text{RO}}$ returns.
- Any other request from any participant or the adversary is simply relayed to the underlying functionality without any further action and the output is given to the destination specified by the hybrid functionality.

Standard UC Corruption Handling:

- Upon receiving $(\text{CORRUPT}, \mathsf{sid}, P)$ from the adversary, set $\mathcal{P}' := \mathcal{P}' \cup \mathcal{P}$. If P has already issued $t > 0$ random oracle queries in this round, set $q_P := t$. Otherwise set $q_P := 0$.

Procedure Round-Reset:

Send $(\text{CLOCK-READ}, \mathsf{sid}_C)$ to $\mathcal{G}_{\text{CLOCK}}$ and receive $(\text{CLOCK-READ}, \mathsf{sid}_C, \tau')$ from $\mathcal{G}_{\text{CLOCK}}$. If $|\tau' - \tau| > 0$ (i.e., a new round started), then set $q_P := 0$ for each participant $P \in \mathcal{P}$ and set $\tau := \tau'$.

Correlated randomness setup. Finally, we make use of the CRS functionality [12], which models a public correlated randomness setup.

Functionality $\mathcal{F}^{\mathcal{D}}_{\text{CRS}}$

When activated for the first time on input $(\text{RETRIEVE}, \mathsf{sid})$, choose a value $d \leftarrow \mathcal{D}$, and send $(\text{RETRIEVE}, d)$ back to the activating party. In each other activation return the value d to the activating party.

3 Inapplicability of Strong BA Impossibility

In this section we present our abstraction of the resource-restricting paradigm as a communication-restricting wrapper for the underlying communication network, and show that the strong BA impossibility (Corollary 1) does not apply to this wrapped network. In partcular, as we discussed, in [7] it was argued that assuming $3t \geq n$, no private correlated randomness setup, the existence of signatures, and authenticated point-to-point channels, no protocol solves the broadcast problem. In this section, we show that if parties have access to a simple channel that is restricted in such a way that spam or sybil attacks are infeasible, the impossibility proof of [7] does not go through.

3.1 Modeling a Communication-Restricted Network

Our filtering wrapper restricts the per-round accesses of each party to the functionality, in a probabilistic manner. In more detail, for parameters p, q, each party has a quota of q SEND requests per round, each of them succeeding with probability p. Note that after a message has been sent through the filter, the sender, as well as the receiver, can re-send the same message for free. This feature captures the fact that if a message has passed the filtering mechanism once, it should be freely allowed to circulate in the network. We explicitly differentiate this action in our interface, by introducing the RESEND request; parties have to use RESEND to forward for free messages they have already received.

Wrapper Functionality $\mathcal{W}^{p,q}_{\text{FLT}}(\mathcal{F})$

The wrapper functionality is parameterized $p \in [0, 1]$ and $q \in \mathbb{N}$, which restrict the probability of success and number of $\mathcal{F}$-evaluations of each party per round, respectively, and a set of parties $\mathcal{P}$. It manages the round integer variable τ, the current set of corrupted parties $\tilde{\mathcal{P}}$, and a list $\mathcal{T}$. For each party $P \in \mathcal{P}$, it manages the integer variable t_P.
Initially $\tau := 0$, $T := \emptyset$, and $t_P := 0$, for each $P \in \mathcal{P}$.

Filtering:

- Upon receiving $(\text{SEND}, \mathsf{sid}, m, P_j)$ from party $P_i \in \mathcal{P}$, execute *Round-Reset*, and do the following:
  - Set $t_{P_i} := t_{P_i} + 1$. If $t_{P_i} \leq q$, with probability p, do:
    1. Add (m, P_i) to $\mathcal{T}$ and output $(\text{SUCCESS}, \mathsf{sid})$ to P_i,
    2. on response $(\text{CONTINUE}, \mathsf{sid}, m)$ from P_i, forward $(\text{SEND}, \mathsf{sid}, m, P_j)$ to $\mathcal{F}$.

    In any other case, send $(\text{FAIL}, \mathsf{sid})$ to P_i.
- Upon receiving $(\text{RESEND}, \mathsf{sid}, m, P_j)$ from honest party $P_i \in \mathcal{P} \setminus \tilde{\mathcal{P}}$, if $(m, P_i) \in \mathcal{T}$ then forward $(\text{SEND}, \mathsf{sid}, m, P_j)$ to $\mathcal{F}$.
- Upon receiving $(\text{RESEND}, \mathsf{sid}, m, P_J)$ from $\mathcal{A}$ on behalf of corrupted $P_i \in \tilde{\mathcal{P}}$, if $(m, P) \in \mathcal{T}$ for some $P \in \mathcal{P}$, then forward $(\text{SEND}, \mathsf{sid}, m, P_j)$ to $\mathcal{F}$.
- Upon $\mathcal{F}$ sending $(\text{SENT}, \mathsf{sid}, m, P_i)$ to P_j, add (m, P_j) to $\mathcal{T}$ and forward the message to P_j.

Standard UC Corruption Handling:

- Upon receiving $(\text{CORRUPT}, \mathsf{sid}, P)$ from the adversary, set $\tilde{\mathcal{P}} \leftarrow \tilde{\mathcal{P}} \cup \mathcal{P}$.

General:

- Any other request from (resp. towards) any participant or the adversary, is simply relayed to the underlying functionality (resp . any participant of the adversary) without any further action.

Procedure Round-Reset:

- Send $(\text{CLOCK-READ}, \mathsf{sid}_C)$ to $\mathcal{G}_{\text{CLOCK}}$ and receive $(\text{CLOCK-READ}, \mathsf{sid}_C, \tau')$ from $\mathcal{G}_{\text{CLOCK}}$.
- If $|\tau' - \tau| > 0$, then set $t_P := 0$ for each $P \in \mathcal{P}$ and set $\tau := \tau'$.

3.2 The Impossibility Theorem, Revisited

Next, we show that if parties have access to $\mathcal{W}^{p,q}_{\text{FLT}}(\mathcal{F}_{\text{AUTH}})$, for some noticeable p and $q \geq 1$, the BA impossibility proof [7] does not go through. The proof relies on the fact that the adversary can simulate the behavior of multiple honest parties. In a nutshell, we describe a protocol where parties send messages through $\mathcal{W}^{p,q}_{\text{FLT}}(\mathcal{F}_{\text{AUTH}})$, and thus due to the restricted number of SEND attempts the adversary has at his disposal, it will be impossible for him to simulate multiple parties running this protocol.

Lemma 2. *Let $n = 3, t = 1$, p be a noticeable function, and $q \geq 1$. There exists a polynomial time protocol in the $(\mathcal{G}_{\text{CLOCK}}, \mathcal{F}_{\text{AUTH}}, \mathcal{W}^{p,q}_{\text{FLT}}(\mathcal{F}_{\text{AUTH}}), \mathcal{F}_{\text{SIG}})$-hybrid model that invalidates the $t \geq n/3$ BA impossibility theorem of [7].*

Proof. The impossibility proof considers the class of full information protocols, where if some party receives a message at some round r, it signs the message with its own signing key, and sends it to all other parties. We are going to show a subclass of protocols that use $\mathcal{W}^{p,q}_{\text{FLT}}(\mathcal{F}_{\text{AUTH}})$ and are not captured by the proof.

We first briefly recall the proof in [7] for the case $n = 3$ and $t = 1$. The proof is based on constructing three scenarios $\sigma_1, \sigma_2, \sigma_3$, where broadcast cannot possibly be achieved. Let the sender be P_1. We proceed to describe $\sigma_1, \sigma_2, \sigma_3$. In σ_1, P_1 has input 0 and P_2 is corrupted. In σ_2, P_1 has input 1 and P_3 is corrupted. In σ_3, P_1 is corrupted.

By Validity, it follows that in σ_1 P_2 should output 0, and in σ_2 P_3 should output 1, no matter the behavior of the adversary. Moreover, due to the Agreement (Consistency) property, the output of P_2 and P_3 in σ_3 must be the same. The proof then proceeds to describe a way of making the view of P_2 (resp. P_3) indistinguishable in scenarios σ_1 (resp. σ_2) and σ_3, and thus reaching a contradiction since they are going to decide on different values in σ_3.

The main idea is for P_2 in σ_1 to behave as if P_1 had input 1, by creating a set of fake keys and changing the signatures of P_1 to the ones with the fake keys and different input where possible. Since there is no PKI, P_3 cannot tell whether P_1 is corrupted and sends messages signed with different keys to P_2, or if P_2 is corrupted. Symmetrically, P_3 in σ_2 simulates P_1 with input 0. Finally, P_1 in σ_3 simulates *both behaviors*, i.e., P_1 running the protocol honestly with input 1 in its communication with P_2, and P_1 with input 0 in its communication with P_3. This is exactly where the impossibility proof does not go through anymore.

For the moment, assume that we are in the setting where $p = 1 - \mathsf{negl}(\lambda)$ and $q = 1$. Let Π be a full information protocol, where in the first round the sender P_1 uses $\mathcal{W}^{1-\mathsf{negl}(\lambda),1}_{\text{FLT}}(\mathcal{F}_{\text{AUTH}})$ to transmit its message to the other two parties. Further, assume that this message is different for the cases where the sender input is 0 and 1, with probability α. It follows that P_1 has to send two *different* messages to parties P_2 and P_3 at the first round of σ_3, with probability α. However, this is not possible anymore, as the network functionality only allows for one new message to be send by P_1 at each round, with overwhelming probability. Hence, with probability α the impossibility proof cannot go through anymore.

For the case where p is noticeable and $q \geq 1$, we can design a similar protocol that cannot be captured by the proof. The protocol begins with a first "super round" of size $\frac{\lambda}{pq}$ regular rounds, where each party should successfully send its first message m at least $\frac{3\lambda}{4}$ times using $\mathcal{W}^{p,q}_{\text{FLT}}(\mathcal{F}_{\text{AUTH}})$ for it to be considered valid. Since the functionality allows sending the same message twice for free, the sequence of $\frac{3\lambda}{4}$ messages is encoded as follows: $(m, 1), \ldots, (m, \frac{3\lambda}{4})$.

Next, we analyze the probability that $\mathcal{A}$ can use the strategy described in the impossibility proof in [7]. Note that each party can query $\mathcal{W}^{p,q}_{\text{FLT}}(\mathcal{F}_{\text{AUTH}})$ up to λ/p times during the super round. We will show that: (i) honest parties will be able to send $\frac{3\lambda}{4}$ messages with overwhelming probability, and (ii) that the adversary in σ_3 will not be able to send the $2 \cdot \frac{3\lambda}{4}$ messages it has to. Let random variable X_i be 1 if the i-th query to $\mathcal{W}^{p,q}_{\text{FLT}}(\mathcal{F}_{\text{AUTH}})$ of some party P succeeds, and 0 otherwise. Also, let $X = \sum_{i=1}^{\lambda/p} X_i$. It holds that $\mathbb{E}[X] = p \cdot \lambda/p = \lambda$. By an application of the Chernoff bound, for $\delta = \frac{1}{4}$, it holds that

$$\Pr[X \leq (1-\delta)\mathbb{E}[X]] = \Pr[X \leq \frac{3\lambda}{4}] \leq e^{-\Omega(\lambda)}.$$

Hence, with overwhelming probability each party will be able to send at least $\frac{3\lambda}{4}$ messages in the first $\frac{\lambda}{pq}$ rounds. On the other hand, we have that

$$\Pr[X \geq (1+\delta)\mathbb{E}[X]] = \Pr[X \geq \frac{5\lambda}{4}] \leq e^{-\Omega(\lambda)}.$$

Hence, no party will be able to send more than $\frac{5\lambda}{4}$ messages in the first super round. This concludes the proof, since the adversary, in order to correctly follow the strategy described before, must send in total $\frac{6\lambda}{4} (> \frac{5\lambda}{4})$ messages in the first super round. Thus, with overwhelming probability it is going to fail to do so. Finally, note that the length of the super round is polynomial, since $1/p$ is bounded by some polynomial. Thus, the theorem follows. □

The proof of Corollary 1 works along the same lines as the proof of [7]; since only public correlated randomness is assumed, nothing prevents the adversary from simulating an honest party. Finally, we note that the same techniques used above can also be used to refute an appropriate adaptation of Corollary 1, where parties have access to $\mathcal{W}^{p,q}_{\textsc{flt}}(\mathcal{F}_{\textsc{auth}})$.

4 Implementing a Communication-Restricted Network

In this section we describe our implementation of $\mathcal{W}^{p,q}_{\textsc{flt}}(\mathcal{F}_{\textsc{auth}})$ that is based on the resource-restricted RO functionality $\mathcal{W}^{q}_{\textsc{ro}}(\mathcal{F}_{\text{RO}})$ and a standard authenticated network. As discussed in the introduction, we also make use of an enhanced version of the $\mathcal{F}_{\textsc{crs}}$ functionality, where it is guaranteed that the adversary learns the shared string after the honest parties. We capture this restriction as a wrapper $\mathcal{W}_{\textsc{fresh}}(\mathcal{F}^{\mathcal{D}}_{\textsc{crs}})$ which does not allow the adversary to learn the CRS before the round honest parties are spawned. W.l.o.g., in the rest of the paper we are going to assume that all parties are spawned at round 1.

Our protocol makes uses of the proof-of-work construction of [2]. Every time a party wants to send a new message, it tries to find a hash of the message and some nonce, that is smaller than some target value, and if successful it forwards this message through $\mathcal{F}_{\textsc{auth}}$ to the designated recipient. Moreover, if it has received such a message and nonce, it can perform a RESEND by forwarding this message through $\mathcal{F}_{\textsc{auth}}$. To be sure that the adversary does not precompute small hashes before the start of the protocol, and thus violates the SEND quota described in the wrapper, parties make use of the string provided by $\mathcal{W}^{\mathcal{D}}_{\textsc{fresh}}(\mathcal{F}_{\textsc{crs}})$, where $\mathcal{D}$ will be a distribution with sufficient high min-entropy. They use this string as a prefix to any hash they compute, thus effectively disallowing the adversary to use any of the small hashes it may have precomputed.

Protocol `Wrapped-Channel`$^{D,q}(P)$

Initialization:

- We assume that P is in the party set of $\mathcal{W}^{q}_{\textsc{ro}}(\mathcal{F}_{\text{RO}})$, $\mathcal{F}_{\textsc{auth}}$, and $\mathcal{W}_{\textsc{fresh}}(\mathcal{F}^{\mathcal{D}}_{\textsc{crs}})$. The protocol maintains a list of valid message/nonce/hash tuples $\mathcal{T}$, initially empty, and a counter t initially set to 0. When P is first activated, it gets the CRS from $\mathcal{W}_{\textsc{fresh}}(\mathcal{F}^{\mathcal{D}}_{\textsc{crs}})$, and uses it as a prefix of all messages it sends to $\mathcal{W}^{q}_{\textsc{ro}}(\mathcal{F}_{\text{RO}})$. For simplicity, we avoid explicitly including this term bellow.

Message Exchange:

- Upon receiving $(\textsc{send}, \mathsf{sid}, m, P')$, execute *Round-Reset*, set $t := t+1$, and if $t > q$ output $(\textsc{fail}, \mathsf{sid})$ to P. Otherwise, do the following:

  1. Send $(\textsc{eval}, \mathsf{sid}, (m, r))$ to $\mathcal{W}^{q}_{\textsc{ro}}(\mathcal{F}_{\text{RO}})$, where $r \leftarrow \{0,1\}^{\lambda}$.
  2. On response $(\textsc{eval}, \mathsf{sid}, (m, r), v)$, if $(v > D)$, output $(\textsc{fail}, \mathsf{sid})$ to P.

3. Otherwise, store (m, r, v) in $\mathcal{T}$, and send $(\textsc{success}, \mathsf{sid})$ to P. On response $(\textsc{continue}, \mathsf{sid})$, pick r', v' such that (m, r', v') is the lexicographically smallest such entry in $\mathcal{T}$, and send $(\textsc{send}, \mathsf{sid}, (m, r', v'), P')$ to $\mathcal{F}_{\text{AUTH}}$.

- Upon receiving $(\textsc{resend}, \mathsf{sid}, m, P')$, let $M := \{(r, v) : (m, r, v) \in \mathcal{T}\}$, and do the following:
  1. If $M \neq \emptyset$, then pick the lexicographically smallest (r, v) in M, and send $(\textsc{send}, \mathsf{sid}, (m, r, v), P')$ to $\mathcal{F}_{\text{AUTH}}$.
  2. Otherwise, output $(\textsc{fail}, \mathsf{sid})$ to P.
- Upon receiving $(\textsc{fetch}, \mathsf{sid})$, forward the message to $\mathcal{F}_{\text{AUTH}}$.
- Upon receiving $(\textsc{sent}, \mathsf{sid}, (m, r, v), P')$ from $\mathcal{F}_{\text{AUTH}}$, send $(\textsc{eval}, \mathsf{sid}, (m, r))$ to $\mathcal{W}^{q}_{\text{RO}}(\mathcal{F}_{\text{RO}})$. On response $(\textsc{eval}, \mathsf{sid}, (m, r), v')$, if $(v \leq D)$ and $(v' = v)$, add (m, r, v) to $\mathcal{T}$ and output $(\textsc{sent}, \mathsf{sid}, m, P')$.
- Upon receiving $(\textsc{fetch-requests}, \mathsf{sid})$, forward the message to $\mathcal{F}_{\text{AUTH}}$, and output its response.

Procedure Round-Reset:
Send $(\textsc{clock-read}, \mathsf{sid}_C)$ to $\mathcal{G}_{\text{CLOCK}}$ and receive $(\textsc{clock-read}, \mathsf{sid}_C, \tau')$ from $\mathcal{G}_{\text{CLOCK}}$. If $|\tau' - \tau| > 0$, then set $t := 0$ and $\tau := \tau'$.

Next, we prove that $\texttt{Wrapped-Channel}^{D,q}$ UC realizes the $\mathcal{W}^{p,q}_{\text{FLT}}(\mathcal{F}_{\text{AUTH}})$ functionality, for appropriate values of p. The main idea of the proof is that the simulator is going to simulate new messages sent through the ideal functionality in the eyes of $\mathcal{A}$, by appropriately programming the random oracle. All other actions can be easily simulated.

Lemma 3. *Let $p := \frac{D}{2^\lambda}$, and $\mathcal{D}$ be a distribution with min-entropy at least $\omega(\log(\lambda))$. The protocol $\texttt{Wrapped-Channel}^{D,q}$ UC-realizes functionality $\mathcal{W}^{p,q}_{\text{FLT}}(\mathcal{F}_{\text{AUTH}})$ in the $(\mathcal{W}^{q}_{\text{RO}}(\mathcal{F}_{\text{RO}}), \mathcal{F}_{\text{AUTH}}, \mathcal{W}_{\text{FRESH}}(\mathcal{F}^{\mathcal{D}}_{\text{CRS}}))$-hybrid model.*

Proof. We consider the following simulator that is parameterized by some real-world adversary $\mathcal{A}$:

Simulator $\mathcal{S}_1$

The simulator manages a set of parties P. It sets up an empty network buffer $\vec{M}$, an empty random oracle table H, and a table of received messages $\mathcal{T}$. The simulator also manages integer variables of the form D_z, where $z \in \{0,1\}^*$, that are dynamically created, and f_P, for $P \in \mathcal{P}$. Initially, $\vec{M}$ is empty, and $f_P := 0$, for $P \in \mathcal{P}$.

Simulating the CRS:
- Sample a value from $\mathcal{D}$ once, and only output it after the round the protocol starts.

Simulating the Random Oracle:
- As in the protocol above, we always include the CRS value as a prefix of all messages to $\mathcal{W}^{q}_{\text{RO}}(\mathcal{F}_{\text{RO}})$. Again, for clarity we avoid explicitly including this term bellow.
- Upon receiving $(\textsc{eval}, \mathsf{sid}, u)$ for $\mathcal{W}^{q}_{\text{RO}}(\mathcal{F}_{\text{RO}})$ from $\mathcal{A}$ on behalf of corrupted $P \in \mathcal{P}$, do the following:
  1. If $H[u]$ is already defined, output $(\textsc{eval}, \mathsf{sid}, u, H[u])$,
  2. If u is of the form (m, r), send $(\textsc{send}, \mathsf{sid}, m, P)$ to $\mathcal{W}^{p,q}_{\text{FLT}}(\mathcal{F}_{\text{AUTH}})$ on behalf of P. On response $(\textsc{fail}, \mathsf{sid})$, set $H[u]$ to a uniform value in $\{0,1\}^\lambda$ larger than D. On response $(\textsc{success}, \mathsf{sid})$, set $H[u]$ to a uniform value in $\{0,1\}^\lambda$ smaller or equal to D. Output $(\textsc{eval}, \mathsf{sid}, v, H[u])$.
  3. Otherwise, set $H[u]$ to a uniform value in $\{0,1\}^\lambda$ and output $(\textsc{eval}, \mathsf{sid}, u, H[u])$.

Simulating the Network:

- Upon receiving $(\textsc{send}, \mathsf{sid}, u, P')$ for $\mathcal{F}_{\text{AUTH}}$ from $\mathcal{A}$ on behalf of corrupted $P \in \mathcal{P}$, do the following:
  1. If u is of the form (m, r, v), $H[(m, r)]$ is defined, $H[(m, r)] = v$, and $v \leq D$, add (u, P) to $\mathcal{T}$, and send $(\textsc{resend}, \mathsf{sid}, m, P')$ to $\mathcal{W}^{p;q}_{\text{FLT}}(\mathcal{F}_{\text{AUTH}})$ on behalf of P. On response $(\textsc{sent}, \mathsf{sid}, m, P, P', mid)$, set $D_{mid} = 1$ and $\vec{M} = \vec{M}||(u, P, P', mid)$, and send $(\textsc{sent}, \mathsf{sid}, u, P, P', mid)$ to $\mathcal{A}$.
  2. Otherwise, send $(\textsc{sent}, \mathsf{sid}, u, P, P', mid)$ to $\mathcal{A}$, where mid is a unique message-ID.
- Upon receiving $(\textsc{fetch-requests}, \mathsf{sid}, P)$ for $\mathcal{F}_{\text{AUTH}}$ from $\mathcal{A}$, execute *Network-Update* and output $(\textsc{fetch-requests}, \mathsf{sid}, P, f_P)$.

Interaction with $\mathcal{W}^{p;q}_{\text{FLT}}(\mathcal{F}_{\text{AUTH}})$:

- Upon receiving $(\textsc{sent}, \mathsf{sid}, m, P, P', mid)$ from $\mathcal{W}^{p;q}_{\text{FLT}}(\mathcal{F}_{\text{AUTH}})$, execute *Network-Update*, and do the following :
  1. If $(\nexists (r', v') : ((m, r', v'), P) \in \mathcal{T})$, pick an r uniformly at random from $\{0,1\}^\lambda$ and set $H[(m, r)] := v$, where v is a uniform value in $\{0,1\}^\lambda$ smaller or equal to D. Then, add $((m, r, v), P)$ to $\mathcal{T}$,
  2. otherwise, pick r, v such that $((m, r, v), P)$ is the lexicographically smallest such entry in $\mathcal{T}$.

  Add $((m, r, v), P, P', mid)$ to $\vec{M}$, set $D_{mid} = 1$, and output $(\textsc{sent}, \mathsf{sid}, (m, r, v), P, P', mid)$ to $\mathcal{A}$.

Procedure Network-Update: For each $P \in \mathcal{P}$, send $(\textsc{fetch-requests}, \mathsf{sid}, P)$ to $\mathcal{W}^{p;q}_{\text{FLT}}(\mathcal{F}_{\text{AUTH}})$. On response $(\textsc{fetch-requests}, \mathsf{sid}, P, f'_P)$, if $f'_P > f_P$, set $f_P := f'_P$ and do the following

1. For all tuples $(u, P', P, mid) \in \vec{M}$, set $D_{mid} := D_{mid} - 1$.
2. For all tuples $(u, P', P, mid) \in \vec{M}$, where $D_{mid} \leq 0$, delete (u, P', P, mid) from $\vec{M}$, and add (u, P_j) to $\mathcal{T}$.

We will argue that for every PPT adversary $\mathcal{A}$ in the real world, no PPT environment $\mathcal{Z}$ can distinguish between the real execution against $\mathcal{A}$ and the ideal execution against $\mathcal{S}_1$.

First, let E_1 denote the event where honest parties in the real world, and on input SEND, repeat a query to the random oracle. Each time an honest party issues a new RO query, a random string of size λ bits is sampled. The probability that the same string is sampled twice in a polynomial execution is negligible in λ. Moreover, E_1 implies this event. Hence, the probability of E_1 happening in a polynomially bounded execution is at most $\mathsf{negl}(\lambda)$. Note, that if E_1 does not occur, the distribution of SEND commands invoked by honest parties that succeed is identical in the real and the ideal world.

Next, we turn our attention to adversarial attempts to send a new message. Let E_2 be the event where $\mathcal{A}$ sends a message of the form (m, r, v) to $\mathcal{F}_{\text{AUTH}}$, such that it hasn't queried (m, r) on the random oracle and $H[(m, r)] = v$. The probability of this event happening, amounts to trying to guess a random value sampled uniformly over an exponential size domain, and is $\mathsf{negl}(\lambda)$. Moreover, if E_2 does not occur, the adversary can only compute new "valid" messages by querying the RO. Define now E_3 to be the event where the adversary makes a query to the RO containing the CRS value, before round 1. By the fact that the CRS value is sampled by a high min-entropy distribution, and that $\mathcal{A}$ is PPT, it is implied that $\Pr[E_3] \leq \mathsf{negl}(\lambda)$. Hence, if E_2 and E_3 do not occur, the distribution of adversarially created messages is identical in both worlds.

Now if E_1, E_2, E_3 do no occur, the view of the adversary and the environment in both worlds is identical, as all requests are perfectly simulated. By an application of the union bound, it is easy to see that $\neg(E_1 \vee E_2 \vee E_3)$ occurs with only negligible probability. Hence, the real and the ideal execution are statistically indistinguishable in the eyes of $\mathcal{Z}$, and the theorem follows. □

Corollary 4. *Let $n = 3, t = 1$, p be a noticeable function, $q \geq 1$, and any distribution $\mathcal{D}$ with min-*

entropy at least $\omega(\log(\lambda))$*. Then, there exist a polynomial time protocol in the* $(\mathcal{G}_{\text{CLOCK}}, \mathcal{W}^q_{\text{RO}}(\mathcal{F}_{\text{RO}}),$ $\mathcal{F}_{\text{AUTH}}, \mathcal{W}_{\text{FRESH}}(\mathcal{F}^{\mathcal{D}}_{\text{CRS}}), \mathcal{F}_{\text{SIG}})$*-hybrid model, that invalidates the proof of the impossibility theorem of [7].*

5 Implementing a Registration Functionality

In this section, we show how to implement a key registration functionality (cf. [10]) in the resource restricted setting, and in the presence of an honest majority of parties.

5.1 The Registration Functionality

The registration functionality $\mathcal{F}_{\text{REG}}$ allows any party to submit a key, which all other parties can later retrieve. Our specific formulation, is parameterized by an integer r that specifies the round after which key retrieval becomes available. Note, that $\mathcal{F}_{\text{REG}}$ does not guarantee that the keys submitted belong to the corresponding parties, i.e., a corrupted party can submit a key it saw another party submit.

Following the paradigm of [4] to deal with synchrony, $\mathcal{F}_{\text{REG}}$ also has a MAINTAIN command, which is parameterized by an implementation dependent function $\mathsf{predict\text{-}time}$. We use this mechanism, to capture the behavior of the real world protocol with respect to $\mathcal{G}_{\text{CLOCK}}$, and appropriately delay $\mathcal{F}_{\text{REG}}$ from sending its clock update until all honest parties get enough activations. In more detail, $\mathsf{predict\text{-}time}$ takes as input a timed honest input sequence of tuples $\vec{I}^T_H = (\ldots, (x_i, id_i, \tau_i), \ldots)$, where x_i is the i-th input provided to $\mathcal{F}_{\text{REG}}$, by honest party id_i at round τ_i. We say that a protocol Π has a *predictable synchronization pattern*, if there exists a function $\mathsf{predict\text{-}time}$ such that for any possible execution of Π, with timed honest input sequence $\vec{I}^T_H$, $\mathsf{predict\text{-}time}(\vec{I}^T_H) = \tau + 1$ if all honest parties have received enough activations to proceed to round $\tau + 1$.

Functionality $\mathcal{F}^r_{\text{REG}}$

The functionality is parameterized by a set of parties $\mathcal{P}$, and an integer r. It maintains integer variables τ, d_u, and a owner/key set $\mathcal{T}$. Initially, $\mathcal{T}$ is empty and τ is equal to 0.

Upon receiving any input I from any party or the adversary, send $(\text{CLOCK-READ}, \mathsf{sid}_C)$ to $\mathcal{G}_{\text{CLOCK}}$. On response $(\text{CLOCK-READ}, \mathsf{sid}_C, t')$, if $|\tau' - \tau| > 0$, set $\tau := \tau'$, $d_u := 0$. Then, if I was received from an honest party $P \in \mathcal{P} \setminus \tilde{\mathcal{P}}$, set $\vec{\mathcal{I}}^T_H := \vec{\mathcal{I}}^T_H || (I, P_i, \tau)$. Depending on the input I and the ID of the sender, execute the respective code:

- On input $I = (\text{SUBMIT}, \mathsf{sid}, v)$ from honest party P, if there is no v' such that $(P, v') \in \mathcal{T}$, add (P, v) to $\mathcal{T}$ and send $(\text{SUBMIT}, \mathsf{sid}, v)$ to $\mathcal{A}$.
- On input $I = (\text{SUBMIT}, \mathsf{sid}, v)$ from corrupted party P, if $\tau \leq r$ and there is a v' such that $(P, v') \in \mathcal{T}$, delete it and add (P, v) instead. Then, send $(\text{SUBMIT}, \mathsf{sid}, v)$ to $\mathcal{A}$.
- On input $I = (\text{RETRIEVE}, \mathsf{sid})$ from party P, if $\tau > r$, output $(\text{RETRIEVE}, \mathsf{sid}, \mathcal{T})$ to P.
- Upon receiving $(\text{MAINTAIN}, \mathsf{sid})$ from honest party P, if all honest parties have submitted an input, $\mathsf{predict\text{-}time}(\vec{I}^T_H) > \tau$, and $d_u = 0$, set $d_u := 1$ and send $(\text{CLOCK-UPDATE}, \mathsf{sid}_C)$ to $\mathcal{G}_{\text{CLOCK}}$. Otherwise, send (I,ID) to $\mathcal{A}$.

5.2 The Protocol

To implement the above functionality we follow an adaptation of the protocol from [1], with the difference that instead of relating keys to pseudonyms, parties are able to create a PKI relating keys to identities. First, we deal with a technical issue.

Our protocol contains commands that perform a sequence of operations. It is possible that during the execution of this operation, the party will lose the activation. Following the formulation of [4], we perform some of the commands in an interruptible manner. That is, a command I is I-*interruptible* executed, if in case activation is lost, an anchor is stored so that in the next invocation of this command it continues from the place it stopped in the previous activation. For more details on how implement this mechanism, we refer to [4].

Next, we give an informal description of the protocol, which makes use of $\mathcal{W}_{\text{FLT}}(\mathcal{F}_{\text{AUTH}})$, $\mathcal{F}_{\text{AUTH}}$, $\mathcal{G}_{\text{CLOCK}}$, and the signature functionality $\mathcal{F}_{\text{SIG}}$ of [10], adapted for many signers and being responsive, i.e., the one who is issuing a command is not losing its activation, as for example is done in the context of the key evolving signature functionality $\mathcal{F}_{\text{KES}}$ of [3].

The protocol is structured in 3 different phases. In the first phase, parties attempt to send enough messages containing a pair of verification keys $(pk, \hat{pk})$ through $\mathcal{W}_{\text{FLT}}(\mathcal{F}_{\text{AUTH}})$ to all other parties. The first key is generated by the parties themselves, while the second key corresponds to the input they receive through the SUBMIT command (we assume here that it is a verification key, but that does not need to be the case). This phase ends after a predetermined number of rounds. The second phase that takes $n+1$ rounds. Parties depending on when they received the different messages assign them a grade, from 0 for the earliest, to n for the latest. To ensure that these grades differ by at most one for the same key, they immediately send keys they received to all other parties. This allows them to establish a form of a graded PKI, denoted by $\mathcal{K}$ in the protocol, where parties are proportionally represented, and which can be later used in the third phase to do broadcast through an adaptation of the Dolev-Strong protocol.

Unlike [1], at the end of the first phase, all parties sign their input $\hat{pk}$ with their key pk, and send it to all other parties. The receiving parties add the key send and signed to a list, denoted by $\mathcal{M}$. This way honest parties can relate a key to their identity. The set $\mathcal{M}$ is then broadcast by each party in the third phase. Finally, by using a majority rule, parties are able to agree on set of keys and identities, denoted by $\mathcal{N}$ in the protocol, which is going to be what they are going to output when they get a RETRIEVE command.

Protocol `Graded-Agreement`(P)

Initialization:

- We assume that P is registered to $\mathcal{G}_{\text{CLOCK}}$ and is in the party sets of $\mathcal{W}^q_{\text{FLT}}(\mathcal{F}_{\text{RO}})$, $\mathcal{F}_{\text{AUTH}}$ and $\mathcal{F}_{\text{SIG}}$. The protocol maintains a list $\mathcal{K}$ of key/grade pairs, a list $\mathcal{M}$ of key/owner tuples, a list $\mathcal{N}$ of key/owner pairs, and a list $\mathcal{T}$ of message/key pairs, all initially empty, keys $pk, \hat{pk}$, initially set to $\perp$, and integer variables $\tau := 0, r := \frac{4n^2\lambda}{\min(1,pq)}$, $c := 1$.

Upon receiving any input I from any party or the adversary, send $(\text{CLOCK-READ}, \mathsf{sid}_C)$ to $\mathcal{G}_{\text{CLOCK}}$. On response $(\text{CLOCK-READ}, \mathsf{sid}_C, t')$, if $|\tau' - \tau| > 0$, set $\tau := \tau'$ and $d_r, d_u := 0$, and do the following:

- Upon receiving $(\text{MAINTAIN}, \mathsf{sid})$, execute in a $(\text{MAINTAIN}, \mathsf{sid})$-interruptible manner the following:

  1. If $d_r = 0$, then:
     - If $0 < \tau \leq r$, execute *PowGeneration.*
     - Else if $r < \tau \leq r + n + 1$, execute *KeyAgreement.*
     - Else, execute *Broadcast.*
     - Set $d_r := 1$.
  2. Else if $d_r = 1$ and $d_u = 1$, set $d_r := 2$ and send $(\text{CLOCK-UPDATE}, \mathsf{sid}_C)$ to $\mathcal{G}_{\text{CLOCK}}$.
  3. Else, set $d_r := 2$.

- Upon receiving $(\text{SUBMIT}, \mathsf{sid}, v)$, if $\tau > 0$ or $d_r = 0$, set $\hat{pk} := v$.

- Upon receiving $(\textsc{Retrieve}, \mathsf{sid})$, if $\tau > r + 2n$, output $\mathcal{N}$.
- Upon receiving $(\textsc{clock-update}, \mathsf{sid}_C)$, if $d_r = 2$ and $d_u = 0$, set $d_u := 1$ and send $(\textsc{clock-update}, \mathsf{sid}_C)$ to $\mathcal{G}_{\textsc{clock}}$. Otherwise, set $d_u := 1$.

Procedure PoWGeneration: If $pk = \perp$, then send $(\textsc{KeyGen}, \mathsf{sid})$ to $\mathcal{F}_{\textsc{sig}}$, and on response $(\textsc{Verification Key}, \mathsf{sid}, v)$, set $pk := v$. If $\hat{pk} = \perp$, give the activation to $\mathcal{Z}$, and in the next activation repeat this step. Otherwise,do the following:

1. Repeat q times: Send $(\textsc{send}, \mathsf{sid}, (pk, c), P)$ to $\mathcal{W}^{p,q}_{\textsc{flt}}(\mathcal{F}_{\textsc{auth}})$. On response $(\textsc{success}, \mathsf{sid})$, increase c by 1, and for each $P' \in \mathcal{P}$ send $(\textsc{resend}, \mathsf{sid}, (pk, i), P')$ through $\mathcal{W}^{p,q}_{\textsc{flt}}(\mathcal{F}_{\textsc{auth}})$.
2. If $\tau = r$, send $(\textsc{Sign}, \mathsf{sid}, pk, (\hat{pk}, P'))$ to $\mathcal{F}_{\textsc{sig}}$. On response, $(\textsc{Signed}, \mathsf{sid}, pk, (\hat{pk}, P'), \sigma)$, for each $P' \in \mathcal{P}$ send $(\textsc{send}, \mathsf{sid}, (pk, \hat{pk}, \sigma), P')$ to $\mathcal{F}_{\textsc{auth}}$.

Procedure KeyAgreement: Do the following:

1. Send $(\textsc{fetch}, sid)$ to $\mathcal{W}_{\textsc{flt}}(\mathcal{F}_{\textsc{auth}})$.
2. On response, $(\textsc{sent}, \mathsf{sid}, \vec{M})$ from $\mathcal{W}_{\textsc{flt}}(\mathcal{F}_{\textsc{auth}})$, for every subset of messages in $\vec{M}$ of the form $M' = \{(\textsc{sent}, \mathsf{sid}, (pk', \hat{pk}', P_i), P'_i)\}_{i \in [(1-\delta)pqr)]}$, for δ equal to $1/4t$, if no entry of the form $(pk', \cdot)$ exists in $\mathcal{K}$, add $(pk', \tau - (r+1))$ to $\mathcal{K}$ and forward the messages in M' to all other parties through $\mathcal{W}_{\textsc{flt}}(\mathcal{F}_{\textsc{auth}})$.
3. If $\tau = r + 1$, send $(\textsc{fetch}, sid)$ to $\mathcal{F}_{\textsc{auth}}$. On response $(\textsc{sent}, \mathsf{sid}, \vec{M}')$ from $\mathcal{F}_{\textsc{auth}}$, for every message in $\vec{M}$ of the form $(\textsc{sent}, \mathsf{sid}, (pk', \hat{pk}', \sigma), P')$, if there exists a entry of the form $(pk', \cdot)$ in $\mathcal{K}$, send $(\textsc{Verify}, \mathsf{sid}, pk', (\hat{pk}', P'), \sigma)$ to $\mathcal{F}_{\textsc{sig}}$. On response $(\textsc{Verified}, \mathsf{sid}, pk', (\hat{pk}', P'), \sigma, f)$, if $f = 1$, add $(\hat{pk'}, P')$ to $\mathcal{M}$.

Procedure Broadcast: Do the following:

1. If $\tau = r + n + 2$, set $m := (\mathcal{M}, pk)$, and send $(\textsc{Sign}, \mathsf{sid}, pk, m)$ to $\mathcal{F}_{\textsc{sig}}$. On response, $(\textsc{Signed}, \mathsf{sid}, pk, m, \sigma)$, send $(\textsc{send}, \mathsf{sid}, (m, pk, (pk, \sigma)), P')$ to every party $P' \in \mathcal{P}$ through $\mathcal{F}_{\textsc{auth}}$.
2. If $r + n + 2 < \tau \leq r + 2n + 2$, send $(\textsc{fetch}, sid)$ to $\mathcal{F}_{\textsc{auth}}$. On response, $(\textsc{sent}, \mathsf{sid}, \vec{M})$ from $\mathcal{F}_{\textsc{auth}}$, do the following:
   (a) For every message in $\vec{M}$ of the form $(\textsc{sent}, \mathsf{sid}, (m', pk_1, (pk_1, \sigma_1), \ldots, (pk_k, \sigma_k)), P')$, for $k = \tau - (r + n + 2)$, send $(\textsc{Verify}, \mathsf{sid}, pk_i, (m, pk_i), \sigma_i)$ to $\mathcal{F}_{\textsc{sig}}$, for $i \in [k]$. If for all responses of the form $(\textsc{Verified}, \mathsf{sid}, pk_i, (m, pk_i), \sigma_i, f_i)$, for $i \in [k]$, it holds that $f_i = 1$, and $(pk_i, g_i) \in \mathcal{K}$ and $g_i \leq k$, add (m', pk_1) to $\mathcal{T}$.
   (b) For every new entry (m', pk'_1) in $\mathcal{T}$, send $(\textsc{Sign}, \mathsf{sid}, pk, (m', pk'_1))$ to $\mathcal{F}_{\textsc{sig}}$. On response, $(\textsc{Signed}, \mathsf{sid}, pk, (m', pk'_1), \sigma)$, add (pk_0, σ) to the relevant message, and forward it to all other parties through $\mathcal{F}_{\textsc{auth}}$.
3. If $\tau = r + 2n + 2$, do the following:
   (a) For every pk_i, where $\exists m \neq m' : (m, pk_i), (m', pk_i) \in \mathcal{T}$, delete all entries of the form $(\cdot, pk_i)$ from $\mathcal{T}$.
   (b) For every $P' \in \mathcal{P}$, if there exists a unique key $\hat{pk'}$, where at least $n/2$ entries of $\mathcal{T}$, contain an entry of the form $(\hat{pk}', P')$, and do not contain any other entry of the form $(\cdot, P')$, add $(\hat{pk'}, P')$ to $\mathcal{N}$.

First, we note that there exists a function $\mathsf{predict\text{-}time}$ for protocol `Graded-Agreement` that successfully predicts when honest parties are done for the round. To see this, notice that honest parties lose their activation in a predictable manner when they get $\textsc{Maintain}$ as input. Moreover, the simulator has all the information it needs to simulate honest parties: it can simulate functionalities $\mathcal{W}_{\textsc{flt}}(\mathcal{F}_{\textsc{auth}})$ and $\mathcal{F}_{\textsc{auth}}$, as well as the signatures generated by honest parties. Finally, due to the properties of the protocol, also proved in [1], parties are going to reliably broadcast their key set $\mathcal{M}$, and thus all provide the same responses on a $\textsc{Retrieve}$ command from $\mathcal{Z}$. We proceed to state our theorem.

Theorem 5. *Let $n > 2t$, p be a noticeable function, $q \in \mathbb{N}^+$. The protocol* `Graded-Agreement` *UC-realizes functionality $\mathcal{F}_{\mathrm{REG}}^{\frac{4n^2\lambda}{\min(1,pq)}+2n+3}$ in the $(\mathcal{G}_{\mathrm{CLOCK}}, \mathcal{F}_{\mathrm{AUTH}}, \mathcal{W}_{\mathrm{FLT}}^{p,q}(\mathcal{F}_{\mathrm{AUTH}}), \mathcal{F}_{\mathrm{SIG}})$-hybrid model.*

Proof. Let $r = \frac{4n^2\lambda}{\min(1,pq)}$ and w.l.og., $p \cdot q \leq 1$. We start by making some observations about the protocol.

Claim 1. *The set $\mathcal{K}$ of each honest party, at the end of round $r+1$, will contain the keys of all other honest parties, with overwhelming probability in λ.*

Proof. We first show that the claim holds for a single honest party. Let random variable X_i be equal to 1, if the i-th invocation of SEND by some honest party P is successful, and 0 otherwise. It holds that $\Pr[X_i = 1] = p$, and that $X_1, \ldots, X_{r \cdot q}$ is a set of independent random variables; each party invokes SEND $r \cdot q$ times up to round r. Let $X = \sum_{i=1}^{rq} X_i$. By an application of the Chernoff bound, it holds that:

$$\Pr[X < (1 - \frac{1}{4t})pqr] = \Pr[X < (1 - \frac{1}{4t})E[X]] \leq e^{-\Omega(\lambda)}$$

Hence, with overwhelming probability each honest party will send at least $(1 - \frac{1}{4t})pqr$ messages, and by an application of the union bound the claim follows. ⊣

In addition to the previous claim we also note two things: (i) The grade of each suck key will be 0, and (ii) due to the unforgeability property of the signature scheme all honest parties will add the associated key $\hat{pk}$ and the correct owner of key pk in $\mathcal{M}$. This two facts will be useful later, when we will argue that all honest keys make it to the final list of keys $\mathcal{N}$, along with their correct owner.

Next, we show that the total number of keys generated will be at most n.

Claim 2. *At the end of round $r + n + 1$, the set $\mathcal{K}$ of each honest party will contain at most n elements, with overwhelming probability.*

Proof. As before let $Z = \sum_{i=1}^{qt(r+n)} Z_i$, denote the successful attempts by the adversary to send a message through $\mathcal{W}_{\mathrm{FLT}}(\mathcal{F}_{\mathrm{AUTH}})$. Note that, starting from round 1, she has $r + n$ rounds in her disposal to send messages. First, after some computations we can show that:

$$(1 + \frac{1}{4t})E[Z] = (1 + \frac{1}{4t})pqt(r + n) \leq (1 - \frac{1}{4t})pqr(t + 1)$$

Then, by the Chernoff bound, it holds that:

$$\Pr[Z > (1 - \frac{1}{4t})pqr(t + 1)] \leq \Pr[Z > (1 + \frac{1}{4t})E[Z]] \leq e^{-\Omega(\lambda)}$$

This implies that with overwhelming probability the set $\mathcal{K}$ of any honest party will contain at most t keys. Due to the previous claim, $\mathcal{K}$ will also contain at most $n - t$ honest keys. Hence, the claim follows. ⊣

It is easy to see that if an honest party adds a key to $\mathcal{K}$ with grade $g < n$, due to the forwarding of the relevant messages for this key in the network, all honest parties will have add key in their keyset with grade at most $g + 1$.

Using all facts proved above, we can now proceed and show that during the Broadcast phase of the protocol, all honest parties will reliably broadcast set $\mathcal{M}$. Moreover, the adversary will not be able to confuse them about her broadcast input, if any. We start by arguing about the values broadcast by honest parties.

Claim 3. *At the end of round $r + 2n + 2$, the set $\mathcal{N}$ of each honest party will contain the keys of all honest parties, along with their correct identity, with overwhelming probability.*

Proof. Let P be some honest party, $(pk, \widehat{pk})$ be her public keys, $\mathcal{K}', \mathcal{M}'$ be her key sets, and $m = \mathcal{M}'$. By our previous claim, all honest parties will have added $(pk, 0)$ to their key set $\mathcal{K}$. Moreover, they will all receive the message $(\widehat{pk}, P)$ signed w.r.t. pk at round $r + 1$ by party P, and thus include $(\widehat{pk}, P)$ in $\mathcal{M}$. Note, that no honest party will include another entry related to P, as P will not send any other such message. Moreover, all parties will receive (m, pk, σ), where σ is a valid signature for m. Hence, they will all add (m, pk) to $\mathcal{T}$. Again, due to unforgeability, they will not add any other entry related to pk in $\mathcal{T}$. Hence, since $\mathcal{T}$ has at most n elements (one for each key) $(\widehat{pk}, P)$ will be the only entry that appears exactly once, with respect to P, in at least $n/2$ sets of $\mathcal{T}$. Thus, all honest parties will add (pk, P) in $\mathcal{N}$, and the claim follows. $\dashv$

Next, we argue that the key sets $\mathcal{N}$ of all honest parties will be the same.

Claim 4. *At the end of round $r + 2n + 2$, all honest parties will have the same set $\mathcal{N}$, with at most one entry per party, with overwhelming probability.*

Proof. First, we argue that all honest parties have same set $\mathcal{T}$ at the end of round $r + 2n + 2$. For the sake of contradiction assume that the opposite was true. This would imply that some honest party P has added $(m, pk) \in \mathcal{T}$ at some round r', while some other party P' has not. We take two cases. If $r' < r + 2n + 2$, then P will forward the messages relevant to entry (m, pk) to all other parties, and they will all add (m, pk) to $\mathcal{T}$. On the other hand, if $r' = r + 2n + 2$, it means that (m, pk) is signed by n keys in the set $\mathcal{K}$ of P, and by our previous claim at least one of these keys was of an honest party. Thus, this party must have accepted this message earlier, and by the previous argument all other honest parties will also have received and accepted this message. This is a contradiction, and hence the honest parties agree on their entries in $\mathcal{T}$.

Next, notice that the deletion of entries from $\mathcal{T}$, when a key is associated with two different messages that happens in the Broadcast procedure, does not change the fact that all honest parties agree on these sets, since if a message is included in $\mathcal{T}$ to begin with, the relevant key is also part of $\mathcal{K}$. Now, since all parties agree on $\mathcal{T}$, and $\mathcal{N}$ is a function of $\mathcal{T}$, it is implied that they will also agree on $\mathcal{N}$. Also, by construction each party P is associated with at most one key in $\mathcal{N}$. The claim follows. $\dashv$

Our last two claims imply that all parties agree on $\mathcal{N}$, each honest party will be correctly represented, and at one most key will be assigned to each identity. Having established these properties of the protocol, we next give a sketch of the simulator, which we denote by $\mathcal{S}_2$. The first thing the simulator must deal with is clock updates. In the ideal world, clock updates sent by the $\mathcal{Z}$ to honest parties, are directly forwarded to $\mathcal{G}_{\text{CLOCK}}$, which in turn forwards them to $\mathcal{S}_2$. This is not the case in the real world, parties sent updates to the $\mathcal{G}_{\text{CLOCK}}$, only after a sufficient number of Maintain and clock-update inputs have been provided by $\mathcal{Z}$. The way we simulate this behavior, is by having $\mathcal{S}_2$ deduce exactly when honest parties would sent their update in the real world, by keeping track of when $\mathcal{F}_{\text{REG}}$ will sent its clock update in the ideal world, and the activations it gets after a Maintain command has been issued to $\mathcal{F}_{\text{REG}}$ or a clock-update command has been issued to

$\mathcal{G}_{\text{CLOCK}}$. Note, that a new round starts only after either of the two commands has been issued, and $\mathcal{S}_2$ has been activated.

Since $\mathcal{S}_2$ can simulate when parties are done for each round, it can easily simulate the interaction of $\mathcal{A}$ with $\mathcal{W}_{\text{FLT}}(\mathcal{F}_{\text{AUTH}})$ and $\mathcal{F}_{\text{AUTH}}$. It does that by simulating the behavior of honest parties. All information needed to do this are either public, or in the case of the signatures of the honest parties, can be generated by the simulator itself. Note, care has been taken, so that never throughout the protocol $\mathcal{S}_2$ has to sign anything with the keys input to $\mathcal{F}_{\text{REG}}$ for honest parties; it only signs with the keys generated by the parties themselves.

Finally, the simulator has to tell $\mathcal{F}_{\text{REG}}$ which keys different parties have registered. It decides that, after round $r + 2n + 2$, where the parties have agreed on their key sets $\mathcal{N}$, and thus have agreed on which keys each party has registered. By our analysis above, this is always the case with overwhelming probability in λ. □

As discussed in the introduction, getting from an implementation of $\mathcal{F}_{\text{REG}}$ where the keys are linked to their owners to standard MPC is fairly straightforward by using the modularity of the UC framework. As proved in [10], $\mathcal{F}_{\text{REG}}$ can be used to realize the certified signature functionality (aka *certification functionality*) $\mathcal{F}_{\text{CERT}}$ which, in turn, can be used to realize a Broadcast functionality against even adaptive adversaries [22]. By plugging this functionality into the honest-majority protocol (compiler) by Cramer *et al.* [14]—an adaptation of the protocol from [29] to tolerate adaptive corruptions—we obtain an MPC protocol which is adaptively secure.

Corollary 6. *Let $n > 2t$, p be a noticeable function, $q \in \mathbb{N}^+$. Then there exists a protocol that UC-realizes functionality $\mathcal{F}_{\text{MPC}}$ in the ($\mathcal{G}_{\text{CLOCK}}$, $\mathcal{F}_{\text{AUTH}}$, $\mathcal{W}^{p,q}_{\text{FLT}}(\mathcal{F}_{\text{AUTH}})$, $\mathcal{F}_{\text{SIG}}$)-hybrid model.*

6 Removing the Freshness Assumption

So far, we have assumed that all parties, including the adversary, get access to the CRS at the same time, i.e., when the protocol starts. In this section, we give a high level overview of how our analysis can be adapted to the case where we remove the fresh CRS and instead assume a collision-resistant hash function. The protocol we devise is based on techniques developed initially in [1].

The main function of the CRS in the implementation of $\mathcal{W}_{\text{FLT}}(\mathcal{F}_{\text{AUTH}})$, is to ensure that all parties agree on which hash evaluations are "fresh," i.e., performed after the CRS became known. Consequently, sent messages are fully transferable, in the sense that they can be forwarded an arbitrary number of times and still be valid. Without a CRS we have to sacrifice full transferability and instead settle with a limited version of the property (cf. [28]).

Next, we describe the filtering functionality we can implement[9] in this setting, denoted $\mathcal{W}_{\text{FLT-LIM}}(\mathcal{F}_{\text{AUTH}})$. The functionality has the same syntax as $\mathcal{W}_{\text{FLT}}(\mathcal{F}_{\text{AUTH}})$, with one difference: each message sent is accompanied by a grade g, which signifies the number of times that this message can be forwarded by different parties and is also related to when the message was initially sent. For example, if party P_1 receives a message with grade 2, the message can be forwarded to party P_2 with grade 1, and party P_2 can forward to party P_3 with grade 0. Party P_3 cannot forward the message any further, while party P_2 can still forward the message to any other party it wants to. Moreover, the initial grade assigned to a message sent using the SEND command is equal to the round that this command was issued minus 1, i.e., messages with higher grades can be computed at later rounds, for honest parties. The adversary has a small advantage: the initial grade of messages he send is equal to the current round. Next, we formally describe $\mathcal{W}_{\text{FLT-LIM}}$.

[9] In fact, we have to slightly weaken the functionality to enforce the participation of honest parties, the same way we do for the $\mathcal{F}_{\text{REG}}$ functionality in Section 5. For clarity, we omit this change.

Wrapper Functionality $\mathcal{W}^{p,q}_{\text{FLT-LIM}}(\mathcal{F})$

The wrapper functionality is parameterized $p \in [0,1]$ and $q \in \mathbb{N}$, which restrict the probability of success and number of $\mathcal{F}$-evaluations of each party per round, respectively, and a set of parties $\mathcal{P}$. It manages the round integer variable τ, the current set of corrupted parties $\tilde{\mathcal{P}}$, and a list $\mathcal{T}$. For each party $P \in \mathcal{P}$, it manages the integer variable t_P. Initially $\tau := 0$, $T := \emptyset$, and $t_P := 0$, for each $P \in \mathcal{P}$.

Filtering:

- Upon receiving $(\textsc{send}, \mathsf{sid}, m, P_j)$ from party $P_i \in \mathcal{P}$, execute *Round-Reset*, and do the following:
  - Set $t_{P_i} := t_{P_i} + 1$. If $t_{P_i} \leq q$, with probability p, do:
    1. If P_i is honest, let[a] $g := \tau$. Otherwise, let $g := \tau - 1$.
    2. Add (m, P_i, g) to $\mathcal{T}$, and output $(\textsc{success}, \mathsf{sid})$ to P_i,
    3. on response $(\textsc{continue}, \mathsf{sid}, m)$ from P_i, forward $(\textsc{send}, \mathsf{sid}, (m, g), P_j)$ to $\mathcal{F}$.

    In any other case, send $(\textsc{fail}, \mathsf{sid})$ to P_i.
- Upon receiving $(\textsc{resend}, \mathsf{sid}, m, g, P_j)$ from honest party $P_i \in \mathcal{P} \setminus \tilde{\mathcal{P}}$, if $(m, P_i, g) \in \mathcal{T}$ and $g > 0$, then forward $(\textsc{send}, \mathsf{sid}, (m, g), P_j)$ to $\mathcal{F}$.
- Upon receiving $(\textsc{resend}, \mathsf{sid}, m, g, P_J)$ from $\mathcal{A}$ on behalf of corrupted $P_i \in \tilde{\mathcal{P}}$, if for some $g' \geq g$ and some $P \in \mathcal{P}$, $(m, P, g') \in \mathcal{T}$, and $g > 0$, forward $(\textsc{send}, \mathsf{sid}, (m, g), P_j)$ to $\mathcal{F}$.
- Upon $\mathcal{F}$ sending $(\textsc{sent}, \mathsf{sid}, (m, g), P_i)$ to P_j, add $(m, P_j, g - 1)$ to $\mathcal{T}$ and forward the message to P_j.

Standard UC Corruption Handling:

- Upon receiving $(\textsc{corrupt}, \mathsf{sid}, P)$ from the adversary, set $\tilde{\mathcal{P}} \leftarrow \tilde{\mathcal{P}} \cup \mathcal{P}$.

General:

- Any other request from (resp. towards) any participant or the adversary, is simply relayed to the underlying functionality (resp . any participant of the adversary) without any further action.

Procedure Round-Reset:
Send $(\textsc{clock-read}, \mathsf{sid}_C)$ to $\mathcal{G}_{\text{CLOCK}}$ and receive $(\textsc{clock-read}, \mathsf{sid}_C, \tau')$ from $\mathcal{G}_{\text{CLOCK}}$. If $|\tau' - \tau| > 0$, then set $t_P := 0$ for each $P \in \mathcal{P}$ and set $\tau := \tau'$.

[a] g is a local variable.

The way we can implement this functionality is by introducing a challenge-exchange procedure to protocol `Wrapped-Channel`: parties multicast random strings, then in the next round hash the ones they received with a collision-resistant hash function and multicast them again, etc. At each round they use the hash they computed as a prefix to the queries they will be making to the restricted RO functionality. If successful, they multicast the hashed value, along with a pre-image of the hash, in order for other parties to be sure that the hash that *they* multicast earlier was used in the computation, and thus ensure freshness. Obviously, in the first round of the protocol parties cannot send any message as they haven't yet exchange any random challenges, in the second round the messages can be transfered one time, in the third twice, and so on.

Next, observe that $\mathcal{W}_{\text{FLT-LIM}}(\mathcal{F}_{\text{AUTH}})$ is sufficient to implement $\mathcal{F}_{\text{REG}}$. The protocol is similar to Protocol `Graded-Agreement`, with the only difference being that parties start trying to send messages through $\mathcal{W}_{\text{FLT-LIM}}(\mathcal{F}_{\text{AUTH}})$ after n rounds have passed. Moreover, in order for a key to be accepted as valid, the messages that accompany it must have a sufficiently high grade. The rest of the protocol is exactly the same. The analysis of [1] is built on this idea, and we point there for more details.

As a result, we are able to implement $\mathcal{F}_{\text{REG}}$, and subsequently $\mathcal{F}_{\text{MPC}}$, without having to assume a "fresh" CRS. With the techniques described above, the following theorem can be proven.

Theorem 7. *Let $n > 2t$ and $q \in \mathbb{N}^+$. Then, there exists a protocol that UC-realizes functionality $\mathcal{F}_{\text{MPC}}$ in the $(\mathcal{G}_{\text{CLOCK}}, \mathcal{F}_{\text{AUTH}}, \mathcal{W}^q_{\text{RO}}(\mathcal{F}_{\text{RO}}), \mathcal{F}_{\text{SIG}})$-hybrid model, assuming the existence of collision resistant hash functions.*

References

[1] M. Andrychowicz and S. Dziembowski. Pow-based distributed cryptography with no trusted setup. In R. Gennaro and M. Robshaw, editors, *Advances in Cryptology - CRYPTO 2015 - 35th Annual Cryptology Conference, Santa Barbara, CA, USA, August 16-20, 2015, Proceedings, Part II*, volume 9216 of *Lecture Notes in Computer Science*, pages 379–399. Springer, 2015.

[2] A. Back. Hashcash. http://www.cypherspace.org/hashcash, 1997.

[3] C. Badertscher, P. Gazi, A. Kiayias, A. Russell, and V. Zikas. Ouroboros genesis: Composable proof-of-stake blockchains with dynamic availability. In *Proceedings of the 2018 ACM SIGSAC Conference on Computer and Communications Security, CCS 2018, Toronto, ON, Canada, October 15-19, 2018*, pages 913–930, 2018.

[4] C. Badertscher, U. Maurer, D. Tschudi, and V. Zikas. Bitcoin as a transaction ledger: A composable treatment. pages 324–356, 2017.

[5] M. Ben-Or, S. Goldwasser, and A. Wigderson. Completeness theorems for non-cryptographic fault-tolerant distributed computation (extended abstract). In J. Simon, editor, *Proceedings of the 20th Annual ACM Symposium on Theory of Computing, May 2-4, 1988, Chicago, Illinois, USA*, pages 1–10. ACM, 1988.

[6] I. Bentov, R. Pass, and E. Shi. Snow white: Provably secure proofs of stake. Cryptology ePrint Archive, Report 2016/919, 2016. `http://eprint.iacr.org/2016/919`.

[7] M. Borcherding. Levels of authentication in distributed agreement. In Ö. Babaoğlu and K. Marzullo, editors, *Distributed Algorithms*, pages 40–55, Berlin, Heidelberg, 1996. Springer Berlin Heidelberg.

[8] R. Canetti. Security and composition of multiparty cryptographic protocols. 13(1):143–202, Jan. 2000.

[9] R. Canetti. Universally composable security: A new paradigm for cryptographic protocols. pages 136–145, 2001.

[10] R. Canetti. Universally composable signature, certification, and authentication. In *17th IEEE Computer Security Foundations Workshop, (CSFW-17 2004), 28-30 June 2004, Pacific Grove, CA, USA*, page 219, 2004.

[11] R. Canetti, Y. Dodis, R. Pass, and S. Walfish. Universally composable security with global setup. pages 61–85, 2007.

[12] R. Canetti and M. Fischlin. Universally composable commitments. In *Advances in Cryptology - CRYPTO 2001, 21st Annual International Cryptology Conference, Santa Barbara, California, USA, August 19-23, 2001, Proceedings*, pages 19–40, 2001.

[13] J. Chen and S. Micali. Algorand. *arXiv preprint arXiv:1607.01341*, 2016.

[14] R. Cramer, I. Damgård, S. Dziembowski, M. Hirt, and T. Rabin. Efficient multiparty computations secure against an adaptive adversary. pages 311–326, 1999.

[15] B. David, P. Gazi, A. Kiayias, and A. Russell. Ouroboros praos: An adaptively-secure, semi-synchronous proof-of-stake blockchain. pages 66–98, 2018.

[16] D. Dolev and H. R. Strong. Authenticated algorithms for byzantine agreement. *SIAM J. Comput.*, 12(4):656–666, 1983.

[17] C. Dwork, N. Lynch, and L. Stockmeyer. Consensus in the presence of partial synchrony. *Journal of the ACM (JACM)*, 35(2):288–323, 1988.

[18] M. Fitzi. *Generalized communication and security models in Byzantine agreement.* PhD thesis, ETH Zurich, 2002.

[19] J. A. Garay, A. Kiayias, and N. Leonardos. The bitcoin backbone protocol: Analysis and applications. In *Advances in Cryptology - EUROCRYPT 2015 - 34th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Sofia, Bulgaria, April 26-30, 2015, Proceedings, Part II*, pages 281–310, 2015.

[20] Y. Gilad, R. Hemo, S. Micali, G. Vlachos, and N. Zeldovich. Algorand: Scaling byzantine agreements for cryptocurrencies. Cryptology ePrint Archive, Report 2017/454, 2017. `http://eprint.iacr.org/2017/454`.

[21] O. Goldreich, S. Micali, and A. Wigderson. How to play any mental game or A completeness theorem for protocols with honest majority. pages 218–229, 1987.

[22] M. Hirt and V. Zikas. Adaptively secure broadcast. pages 466–485, 2010.

[23] J. Katz, U. Maurer, B. Tackmann, and V. Zikas. Universally composable synchronous computation. In *Theory of Cryptography - 10th Theory of Cryptography Conference, TCC 2013, Tokyo, Japan, March 3-6, 2013. Proceedings*, pages 477–498, 2013.

[24] J. Katz, A. Miller, and E. Shi. Pseudonymous broadcast and secure computation from cryptographic puzzles. Cryptology ePrint Archive, Report 2014/857, 2014. `http://eprint.iacr.org/2014/857`.

[25] A. Kiayias, A. Russell, B. David, and R. Oliynykov. Ouroboros: A provably secure proof-of-stake blockchain protocol. pages 357–388, 2017.

[26] L. Lamport, R. E. Shostak, and M. C. Pease. The byzantine generals problem. *ACM Trans. Program. Lang. Syst.*, 4(3):382–401, 1982.

[27] R. Pass, L. Seeman, and abhi shelat. Analysis of the blockchain protocol in asynchronous networks. Cryptology ePrint Archive, Report 2016/454, 2016. `http://eprint.iacr.org/2016/454`.

[28] B. Pfitzmann and M. Waidner. Unconditional byzantine agreement for any number of faulty processors. In A. Finkel and M. Jantzen, editors, *STACS 92, 9th Annual Symposium on Theoretical Aspects of Computer Science, Cachan, France, February 13-15, 1992, Proceedings*, volume 577 of *Lecture Notes in Computer Science*, pages 339–350. Springer, 1992.

[29] T. Rabin and M. Ben-Or. Verifiable secret sharing and multiparty protocols with honest majority (extended abstract). pages 73–85, 1989.

[30] F. B. Schneider. Implementing fault-tolerant services using the state machine approach: A tutorial. *ACM Comput. Surv.*, 22(4):299–319, 1990.

[31] A. C.-C. Yao. Protocols for secure computations (extended abstract). pages 160–164, 1982.