Universally Composable Simultaneous Broadcast against a Dishonest Majority and Applications

Myrto Arapinis
The University of Edinburgh
Edinburgh, United Kingdom
marapini@ed.ac.uk

Ábel Kocsis
The University of Edinburgh
Edinburgh, United Kingdom
abelkcss@gmail.com

Nikolaos Lamprou
The University of Edinburgh
Edinburgh, United Kingdom
nikolaoslabrou@yahoo.gr

Liam Medley
Royal Holloway University of London
Egham, United Kingdom
liam.medley.2018@live.rhul.ac.uk

Thomas Zacharias
The University of Edinburgh
Edinburgh, United Kingdom
tzachari@ed.ac.uk

ABSTRACT

Simultaneous broadcast (SBC) protocols, introduced in [Chor et al., FOCS 1985], constitute a special class of broadcast channels which, besides consistency, guarantee that all senders broadcast their messages independently of the messages broadcast by other parties. SBC has proved extremely useful in the design of various distributed computing constructions (e.g., multiparty computation, coin flipping, electronic voting, fair bidding). As with any communication channel, it is crucial that SBC security is composable, i.e., it is preserved under concurrent protocol executions. The work of [Hevia, SCN 2006] proposes a formal treatment of SBC in the state-of-the-art Universal Composability (UC) framework [Canetti, FOCS 2001] and a construction secure assuming an honest majority.

In this work, we provide a comprehensive revision of SBC in the UC setting and improve the results of [Hevia, SCN 2006]. In particular, we present a new SBC functionality that captures *both simultaneity and liveness* by considering a broadcast period such that (i) within this period all messages are broadcast independently and (ii) after the period ends, the session is terminated without requiring full participation of all parties. Next, we employ time-lock encryption (TLE) over a standard broadcast channel to devise an SBC protocol that realizes our functionality against any adaptive adversary corrupting up to *all-but-one* parties. In our study, we capture synchronicity via a global clock [Katz et al., TCC 2013], thus lifting the restrictions of the original synchronous communication setting used in [Hevia, SCN 2006]. As a building block of independent interest, we prove the first TLE protocol that is *adaptively* secure in the UC setting, strengthening the main result of [Arapinis et al., ASIACRYPT 2021].

Finally, we formally exhibit the power of our SBC construction in the design of UC-secure applications by presenting two interesting use cases: (i) distributed generation of uniform random strings, and (ii) decentralized electronic voting systems, without the presence of a special trusted party.

CCS CONCEPTS

• **Security and privacy → Distributed systems security**; **Cryptography**; **Formal security models**.

KEYWORDS

Secure Broadcast, Universal Composability, Time-Lock Encryption

ACM Reference Format:
Myrto Arapinis, Ábel Kocsis, Nikolaos Lamprou, Liam Medley, and Thomas Zacharias. 2023. Universally Composable Simultaneous Broadcast against a Dishonest Majority and Applications. In *ACM Symposium on Principles of Distributed Computing (PODC '23), June 19–23, 2023, Orlando, FL, USA.* ACM, New York, NY, USA, **??** pages. https://doi.org/10.1145/3583668.xxxxxxx

1 INTRODUCTION

Communication over a broadcast channel guarantees consistency of message delivery, in the sense that all honest parties output the same message, even when the sender is malicious. Since its introduction by Pease et al. [**?**], broadcast has been a pivotal concept in fault tolerant distributed computing and cryptography. From a property-based security perspective, broadcast communication dictates that every honest party will output some value (termination) that is the same across all honest parties (agreement) and matches the sender's value, when the sender is honest (validity). The first efficient construction was proposed by Dolev and Strong [**?**]. In particular, the Dolev-Strong broadcast protocol deploys public-key infrastructure (PKI) to achieve property-based security against an adversary corrupting up to $t < n$ parties, where n is the number of parties. In the context of simulation-based security though, Hirt and Zikas [**?**] proved that broadcast under $t > \frac{n}{2}$ corruptions (dishonest majority) is impossible, even assuming a PKI. In the model of [**?**], the adversary may adaptively corrupt parties within the duration of a round (*non-atomic communication model*). Subsequently, Garay et al. [**?**] showed that in the weaker setting where a party cannot be corrupted in the middle of a round (*atomic communication model*), PKI is sufficient for realizing adaptively secure broadcast against an adversary corrupting up to $t < n$ parties.

An important class of protocols that has attracted considerable attention is the one where broadcast is *simultaneous*, i.e., all senders transmit their messages independently of the messages broadcast

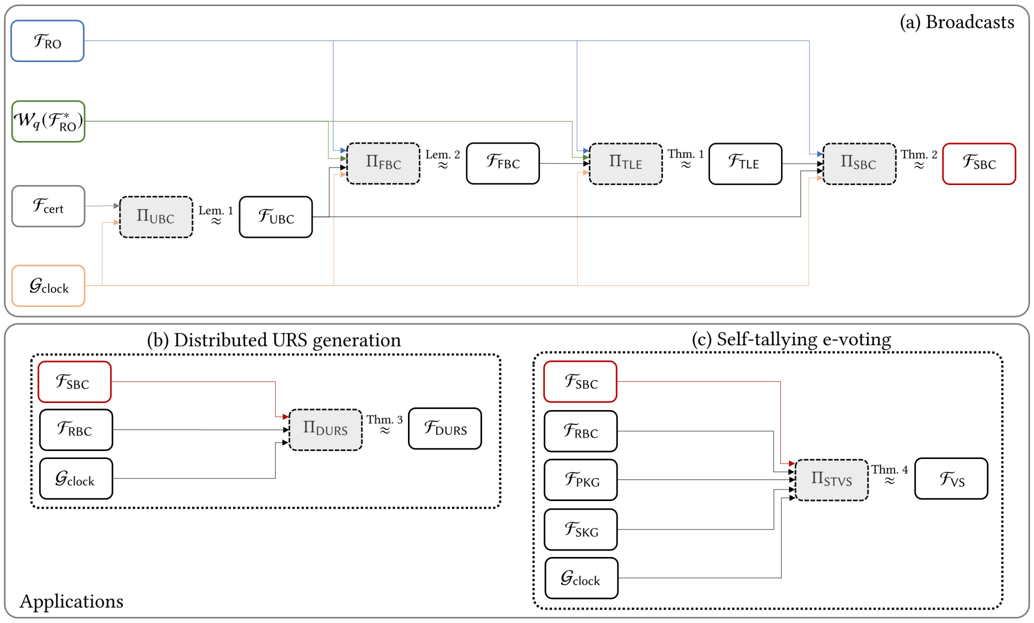
(a) Broadcasts
$\mathcal{F}_{\mathrm{RO}}$
$\mathcal{W}_q(\mathcal{F}^*_{\mathrm{RO}})$
$\mathcal{F}_{\mathrm{cert}}$
$\mathcal{G}_{\mathrm{clock}}$
Π_{UBC}
Lem. 1 ≈
$\mathcal{F}_{\mathrm{UBC}}$
Π_{FBC}
Lem. 2 ≈
$\mathcal{F}_{\mathrm{FBC}}$
Π_{TLE}
Thm. 1 ≈
$\mathcal{F}_{\mathrm{TLE}}$
Π_{SBC}
Thm. 2 ≈
$\mathcal{F}_{\mathrm{SBC}}$
(b) Distributed URS generation
$\mathcal{F}_{\mathrm{SBC}}$
$\mathcal{F}_{\mathrm{RBC}}$
$\mathcal{G}_{\mathrm{clock}}$
Π_{DURS}
Thm. 3 ≈
$\mathcal{F}_{\mathrm{DURS}}$
(c) Self-tallying e-voting
$\mathcal{F}_{\mathrm{SBC}}$
$\mathcal{F}_{\mathrm{RBC}}$
$\mathcal{F}_{\mathrm{PKG}}$
$\mathcal{F}_{\mathrm{SKG}}$
$\mathcal{G}_{\mathrm{clock}}$
Π_{STVS}
Thm. 4 ≈
$\mathcal{F}_{\mathrm{VS}}$
Applications

Figure 1: Overview of the paper's contributions. We denote Π_{X} the X, and $\mathcal{F}_{\mathsf{X}}$ the ideal functionality capturing the security requirements for X, and in UC fashion we write $\Pi_{\mathsf{X}} \approx \mathcal{F}_{\mathsf{X}}$ to denote that the protocol Π_{X} realizes the ideal functionality $\mathcal{F}_{\mathsf{X}}$ (thus, ensuring the same security properties). Our results rely on the following hybrid functionalities: (i) the global clock $\mathcal{G}_{\mathrm{clock}}$, (ii) the random oracles $\mathcal{F}_{\mathrm{RO}}$ and $\mathcal{F}^*_{\mathrm{RO}}$, (iii) the wrapper $\mathcal{W}_q(\cdot)$, (iv) the certification $\mathcal{F}_{\mathrm{cert}}$ modelling a PKI, (v) the relaxed broadcast $\mathcal{F}_{\mathrm{RBC}}$ that allows a single message to be broadcast in an unfair manner (and can be realized via $\mathcal{F}_{\mathrm{cert}}$ and $\mathcal{G}_{\mathrm{clock}}$, cf. Fact ??), (vi) the public key threshold key generation $\mathcal{F}_{\mathrm{PKG}}$, and (vii) the signature key generation $\mathcal{F}_{\mathrm{SKG}}$.

by other parties. The concept of simultaneous broadcast (SBC) was put forth by Chor et al. [?] and has proved remarkably useful in the design of various distributed computing constructions (e.g., multi-party computation, coin flipping, electronic voting, fair bidding). The works of Chor and Rabin [?] and Gennaro [?] improve the round complexity of [?] from linear to logarithmic, and from logarithmic to constant (in n), respectively. From a security modeling aspect, Hevia and Micciancio [?] point out the hierarchy between the SBC definitions in [? ? ?] as [?]⇒[?]⇒[?] (from strongest to weakest). Specifically, the simulation-based definition of [?] implies sequentially composable security. Under the definition of [?], Faust et al. [?] present a construction with a performance gain in the presence of repeated protocol runs. All the aforementioned SBC solutions [? ? ? ?] achieve security that tolerates $t < \frac{n}{2}$ corruptions (honest majority).

The concept of SBC that retains security under concurrent executions has been formally investigated by Hevia [?]. Namely, [?] proposes a formal SBC treatment in the state-of-the-art Universal Composability (UC) framework of Canetti [?] along with a construction that has constant round complexity and is secure assuming an honest majority. Composable security is crucial for any broadcast channel functionality that serves as a building block for distributed protocol design and is a primary goal of our work.

Our contributions. We explore the SBC problem in the context of UC security against a dishonest majority. We improve the results of [?] both from a definitional and a security aspect. In more detail, we achieve the following improvements (cf. Figure **??**(a)):

- We define a new SBC functionality that abstracts communication given an agreed broadcast period, outside of which all broadcast operations are discarded. Our functionality captures (i) *simultaneity*: corrupted senders broadcast without having any information about honest senders' messages; (ii) *liveness*: after the broadcast period ends, termination is guaranteed (with some delay) without the requirement of full participation by all parties. We stress that the latter property is not captured by the functionality of [?], as the adversary (simulator) may wait indefinitely until it allows termination of the execution which happens only after all (honest and corrupted) senders have transmitted their value.
- We provide a construction that realizes our SBC functionality in an optimal way, that is, it preserves *UC security against a Byzantine adversary that can adaptively corrupt up to $t < n$ parties in the non-atomic communication model.* To overcome the impossibility result of [?], besides PKI, we deploy (i) adaptively secure time-lock encryption (TLE) in the UC setting; (ii) a programmable random oracle (RO). Specifically, via TLE (and the programmable RO), senders perform (equivocable) encryptions of their message that can be decrypted by any party when the decryption time comes, with some delay upon the end of the broadcast period. It is easy to see that the semantic security of the TLE ciphertexts that lasts throughout the broadcast period guarantees simultaneity. The broadcast period is set dynamically, by having the first sender of the session (as scheduled by the environment) "wake up" the other parties via the broadcast of a special message.

Although using a programmable RO is standard to enable equivocation in simulation-based security (e.g., in [? ? ? ?]), TLE with adaptive UC security is not available in the literature. To construct it, we rely on the findings of the following papers:

(1) The work of Arapinis et al. [?] that provides a UC treatment of TLE and a protocol that is secure against a static adversary.
(2) The work of Cohen et al. [?] that studies the concept of broadcast and fairness in the context of resource-restricted cryptography [?]. They prove that time-lock puzzles (TLPs) (a notion closely related to TLE) and a programmable RO are sufficient for building stand-alone simulation-based secure broadcast against an adaptive adversary that corrupts up to $t < n$ parties in the non-atomic model. They also show that neither TLPs nor programmable ROs alone are enough to achieve such level of security. In [?], standard broadcast encompasses *fairness*, i.e., an adversary that adaptively corrupts a sender after learning her value cannot change this original value. The weaker notion of *unfair* broadcast [?] can be realized by the Dolev-Strong protocol [?] against $t < n$ adaptive corruptions.

Compared to [?] and [?], we take the following steps: first, we adapt the fair broadcast (FBC) and unfair broadcast (UBC) functionalities in [?] to the UC setting, where multiple senders may perform many broadcasts per round. Then, similar to [? ? ?], we model resource-restriction in UC via wrapper that allows all parties to perform up to a number of RO queries per round. Next, we revisit the FBC protocol in [?] by using the TLE algorithms of [?] instead of an arbitrary time-lock puzzle and show that our instantiation UC-realizes our FBC functionality. Finally, we prove that by deploying the TLE protocol of [?] over our FBC functionality is sufficient to provide an adaptively secure realization of the TLE functionality in [?]. We view the construction of the first adaptively UC secure TLE protocol as a contribution of independent interest. We refer the reader to Section ?? for a detailed discussion of the key subtleties to the design of our composably secure (un)fair broadcast protocols.

- The SBC construction in [?] is over the synchronous communication functionality in [?]. As [?] shows, this functionality does not provide the guarantees expected of a synchronous network (specifically, termination). These limitations are lifted when relying on a (global) clock functionality [?], as we do in our formal treatment. The use of a global clock is the standard way to model loose synchronicity in UC: every clock tick marks the advance of the execution rounds while within a round, communication is adversarially scheduled by the environment.

Armed with our construction, we present two interesting applications of SBC that enjoy adaptive UC security. Namely,

- *Distributed uniform random string generation (Figure ??(b)).* We devise a protocol where a set of parties contribute their share of randomness via our SBC channel. After some delay (upon the end of the broadcast period), the honest parties agree on the XOR of the shares they received as a common uniform random string. We call this *delayed uniform random string* (DURS) generation.
- *Self-tallying e-voting (Figure ??(c)).* Self-tallying voting systems (STVSs) constitute a special class of decentralized electronic voting systems put forth by Kiayias and Yung [?], where the voters can perform the tally themselves without the need for a trusted tallying authority. Most existing efficient STVSs [? ? ?] require a trusted party to ensure election fairness (i.e., no partial results are leaked before the end of the vote casting period). We remove the need of a trusted party in self-tallying elections by modifying the construction in [?] (shown secure in the UC framework). In particular, we deploy our SBC channel for vote casting instead of a bulletin board used in the original protocol.

The proofs of all the theorems and lemmas can be found in the companion full version [?].

2 BACKGROUND

2.1 Network model

We consider synchronous point-to-point communication among n parties in $\mathbf{P}$, where protocol execution is carried out in rounds. The adversary is Byzantine and may adaptively corrupt any number of $t < n$ parties. The corruption is w.r.t. the strong *non-atomic communication model* (cf. [? ?]) where the adversary may corrupt parties in the middle of a round.

2.2 The UC framework

Universal Composability (UC), introduced by Canetti in [?], is a state-of-the-art framework for the formal study of protocols that should remain secure under concurrent executions. In UC, security is captured via the *real world/ideal world* paradigm as follows.

- In the ideal world, an *environment* $\mathcal{Z}$ schedules the execution and provides inputs to the parties that are *dummy*, i.e., they simply forward their inputs to an *ideal functionality* $\mathcal{F}$, which abstracts the studied security notion (e.g., secure broadcast). The functionality is responsible for carrying out the execution given the forwarded input and returns to the party some output along with a destination identity *ID*, so that the dummy party forwards the output to *ID*. By default, we assume that the destination is $\mathcal{Z}$, unless specified explicitly. The execution is carried out in the presence of an ideal adversary $\mathcal{S}$, the *simulator*, that interacts with $\mathcal{F}$ and $\mathcal{Z}$ and controls corrupted parties. We denote by $\mathrm{EXEC}_{\mathcal{F},\mathcal{S},\mathcal{Z}}$ the output of $\mathcal{Z}$ after ending the ideal world execution.
- In the real world, $\mathcal{Z}$ schedules the execution and provides inputs as previously, but now the parties actively engage in a joint computation w.r.t. the guidelines of some protocol Π (e.g., a broadcast protocol). The execution is now in the presence of a real (Byzantine) adversary $\mathcal{A}$ that interacts with $\mathcal{Z}$ and may (adaptively) corrupt a number of parties. We denote by $\mathrm{EXEC}_{\Pi,\mathcal{A},\mathcal{Z}}$ the output of $\mathcal{Z}$ after ending the real world execution.

Definition 1. *We say that a protocol* Π UC-realizes *the ideal functionality* $\mathcal{F}$ *if for every real world adversary* $\mathcal{A}$ *there is a simulator* $\mathcal{S}$ *such that for every environment* $\mathcal{Z}$*, the distributions* $EXEC_{\mathcal{F},\mathcal{S},\mathcal{Z}}$ *and* $EXEC_{\Pi,\mathcal{A},\mathcal{Z}}$ *are computationally indistinguishable.*

According to the UC Theorem, the UC security of Π implies that Π can be replaced by $\mathcal{F}$ in any protocol that invokes Π as a subroutine. Besides, a protocol may use as subroutine a functionality that abstracts some setup notion (e.g., PKI, a random oracle, or a global clock). These setup functionalities maybe *global*, in the sense that share their state across executions of multiple protocols [?]. If a protocol utilizes a set of functionalities $\{\mathcal{F}_1, \ldots, \mathcal{F}_k\}$, then we say that its UC security is argued in the $(\mathcal{F}_1, \ldots, \mathcal{F}_k)$*-hybrid model.*

2.3 Hybrid functionalities

Throughout the paper, λ denotes the security parameter and $\mathsf{negl}(\cdot)$ a negligible function.

The global clock functionality. The global clock (cf. [? ?]) can be read at any moment by any party. For each session, the clock advances only when all the involved honest parties and functionalities in the session make an Advance_Clock request.

The random oracle functionality. The RO functionality (cf. [?]) can be seen as a trusted source of random input. Given a query, it returns a random value. It also updates a local variable $L_{\mathcal{H}}$ in order to return the same value to similar queries. This functionality can be seen as the "idealization" of a hash function.

The certification functionality. The certification functionality (cf. [?]) abstracts a certification scheme which provides signatures bound to *identities*. It provides commands for signature generation and verification, and is tied to a single party (so each party requires a separate instance). It can be realized via an EUF-CMA secure signature scheme combined with a party acting as a trusted certification authority.

The wrapper functionality. We recall the wrapper functionality from [?] in the full version (in the adaptive corruption model), for the special case where the wrapped evaluation functionality is the random oracle $\mathcal{F}_{\mathsf{RO}}$. The wrapper $\mathcal{W}_q$ allows the parties to access $\mathcal{F}_{\mathsf{RO}}$ only up to q times per round (clock tick).

The relaxed broadcast functionality $\mathcal{F}_{\mathsf{RBC}}(\mathbf{P})$.

The functionality initializes a pair of variables $(\mathsf{Output}, \mathsf{Sender})$ as $(\bot, \bot)$. It also maintains the set of corrupted parties, $\mathbf{P}_{\mathsf{corr}}$, initialized as empty.

- Upon receiving $(\mathsf{sid}, \textsc{Broadcast}, M)$ from $P \in \mathbf{P} \setminus \mathbf{P}_{\mathsf{corr}}$, if $(\mathsf{Output}, \mathsf{Sender}) = (\bot, \bot)$, it records the output-sender pair $(\mathsf{Output}, \mathsf{Sender}) \leftarrow (M, P)$ and sends $(\mathsf{sid}, \textsc{Broadcast}, M, P)$ to $\mathcal{S}$.
- Upon receiving $(\mathsf{sid}, \textsc{Broadcast}, M, P)$ from $\mathcal{S}$ on behalf of $P \in \mathbf{P}_{\mathsf{corr}}$, if $(\mathsf{Output}, \mathsf{Sender}) = (\bot, \bot)$, it sends $(\mathsf{sid}, \textsc{Broadcast}, M, P)$ to all parties and $\mathcal{S}$, and halts.
- Upon receiving $(\mathsf{sid}, \textsc{Allow}, \tilde{M})$ from $\mathcal{S}$, if $\mathsf{Sender} \in \mathbf{P}_{\mathsf{corr}}$, it sends $(\mathsf{sid}, \textsc{Broadcast}, \tilde{M}, \mathsf{Sender})$ to all parties and $\mathcal{S}$, and halts. Otherwise, it ignores the message.
- Upon receiving $(\mathsf{sid}_C, \textsc{Advance\_Clock})$ from $P \in \mathbf{P} \setminus \mathbf{P}_{\mathsf{corr}}$, if $\mathsf{Sender} = P$, it sends $(\mathsf{sid}, \textsc{Broadcast}, \mathsf{Output}, \mathsf{Sender})$ to all parties and $\mathcal{S}$, and halts. Otherwise, it returns $(\mathsf{sid}_C, \textsc{Advance\_Clock})$ to P with destination identity $\mathcal{G}_{\mathsf{clock}}$.

Figure 2: The functionality $\mathcal{F}_{\mathsf{RBC}}$ interacting with the parties in P and the simulator $\mathcal{S}$.

The relaxed broadcast functionality. In Figure ??, we present the relaxed broadcast functionality $\mathcal{F}_{\mathsf{RBC}}$ (for a single message) in [?] that is the stepping stone for realizing unfair broadcast (cf. Subsection ??) which, in turn, is in the core of the design of the fair and simultaneous broadcast constructions. The functionality captures agreement, but only a weak notion of validity, i.e., if a sender is *always* honest and broadcasts a message M, then every honest party will output the value M. In addition, we modify the original description of $\mathcal{F}_{\mathsf{RBC}}$ by forcing the delivery of the message to all parties, when the sender (i) is initially corrupted, or (ii) remains honest in the execution and completes her part by forwarding an Advance_Clock message. This was implicit in [?]. As presented in [? ?], $\mathcal{F}_{\mathsf{RBC}}$ can be realized based on the Dolev-Strong protocol [?] and a UC-secure signature scheme. Formally,

Fact 1 ([? ?]). *There exists a protocol Π_{RBC} that UC-realizes $\mathcal{F}_{\mathsf{RBC}}$ in the $(\mathcal{F}_{\mathsf{cert}}, \mathcal{G}_{\mathsf{clock}})$-hybrid model against an adaptive adversary corrupting $t < n$ parties (in the non-atomic model).*

2.4 Time-lock encryption

To realize our secure SBC we will mobilise a special type of encryption, called *time-lock encryption* (TLE). TLE allows one to encrypt a message M for a set amount of time τ. Decryption requires a witness w whose computation is inherently sequential. [?] provides a UC treatment of the TLE primitive, and a TLE scheme that is UC secure against static adversaries. We will revisit TLE in the presence of adaptive adversaries in Section ??.

The time-lock encryption (TLE) functionality. In Figure ??, we present the TLE functionality from [?]. Here, $\mathsf{leak}(\cdot)$ is a function over time slots that captures the timing advantage of the adversary in intercepting the TLE ciphertexts, and delay is an integer that express the delay of ciphertext generation.

The Astrolabous TLE scheme. We utilize the algorithms of the Astrolabous TLE scheme from [?]. Given Astrolabous, $\mathcal{F}_{\mathsf{TLE}}$ is UC-realized in the static corruption model as stated below.

Fact 2 ([?]). *Let $\mathcal{F}_{\mathsf{BC}}$ be the broadcast functionality defined in [?]. There exists a protocol that UC-realizes $\mathcal{F}_{\mathsf{TLE}}^{\mathsf{leak},\mathsf{delay}}$ in the $(\mathcal{W}_q(\mathcal{F}^*_{\mathsf{RO}}), \mathcal{F}_{\mathsf{RO}}, \mathcal{F}_{\mathsf{BC}}, \mathcal{G}_{\mathsf{clock}})$-hybrid model against a static adversary corrupting $t < n$ parties, with leakage function $\mathsf{leak}(\mathsf{Cl}) = \mathsf{Cl} + 1$ and $\mathsf{delay} = 1$, where $\mathcal{F}_{\mathsf{RO}}$ and $\mathcal{F}^*_{\mathsf{RO}}$ are distinct random oracles.*

3 UC (UN)FAIR BROADCAST AGAINST DISHONEST MAJORITIES

The prior work of Cohen *et al.* [?] studies broadcast fairness in a stand-alone fashion. Here, we revisit the concept of broadcast fairness in the setting of UC security, where protocol sessions may securely run concurrently or as subroutines of larger protocols; and in each session, every party can send of multiple messages. We provide a comprehensive formal treatment of the notions of *unfair broadcast* (UBC) and *fair broadcast* (FBC) that will be the stepping stones for the constructions of the following sections.

3.1 Unfair broadcast definition and realization

The UBC functionality. We consider a relaxation of FBC, captured by the notion of unfair broadcast introduced in [?]. We present the UBC functionality in Figure ??. Informally, in UBC, the adversary (simulator) is allowed to receive the sender's message before broadcasting actually happens, and (unlike in FBC) adaptively corrupt the sender to broadcast a message of its preference.

The time-lock encryption functionality $\mathcal{F}_{\mathsf{TLE}}^{\mathsf{leak},\mathsf{delay}}(\mathbf{P})$.

The functionality initializes the list of recorded message/ciphertext L_{rec} as empty and defines the tag space TAG. It also maintains the set of corrupted parties, $\mathbf{P}_{\mathrm{corr}}$, initialized as empty.

▪ Upon receiving $(\mathsf{sid}, \textsc{Enc}, M, \tau)$ from $P \notin \mathbf{P}_{\mathrm{corr}}$, it reads the time Cl and does:

(1) If $\tau < 0$, it returns $(\mathsf{sid}, \textsc{Enc}, M, \tau, \perp)$ to P.
(2) It picks tag $\overset{\$}{\leftarrow}$ TAG and it inserts the tuple $(M, \mathsf{Null}, \tau, \mathsf{tag}, \mathsf{Cl}, P) \rightarrow L_{\mathrm{rec}}$.
(3) It sends $(\mathsf{sid}, \textsc{Enc}, \tau, \mathsf{tag}, \mathsf{Cl}, 0^{|M|}, P)$ to $\mathcal{S}$. Upon receiving the token back from $\mathcal{S}$ it returns $(\mathsf{sid}, \textsc{Encrypting})$ to P.

▪ Upon receiving $(\mathsf{sid}, \textsc{Update}, \{(c_j, \mathsf{tag}_j)\}_{j=1}^{p(\lambda)})$ from $\mathcal{S}$, for all $c_j \neq \mathsf{Null}$ it updates each tuple $(M_j, \mathsf{Null}, \tau_j, \mathsf{tag}_j, \mathsf{Cl}_j, P)$ in L_{rec} to $(M_j, c_j, \tau_j, \mathsf{tag}_j, \mathsf{Cl}_j, P)$.

▪ Upon receiving $(\mathsf{sid}, \textsc{Retrieve})$ from P, it reads the time Cl and does:

(1) For every tuple $(M, \mathsf{Null}, \tau, \mathsf{tag}, \mathsf{Cl}', P) \in L_{\mathrm{rec}}$ such that $\mathsf{Cl} - \mathsf{Cl}' \geq \mathsf{delay}$, it picks $c \overset{\$}{\leftarrow} \{0,1\}^{p'(\lambda)}$ and updates the tuple as $(M, c, \tau, \mathsf{tag}, \mathsf{Cl}', P)$.
(2) It sets $C := \{(M, c, \tau)\}_{(M,c,\tau,\cdot,\mathsf{Cl}',P) \in L_{\mathrm{rec}}: \mathsf{Cl}-\mathsf{Cl}' \geq \mathsf{delay}}$.
(3) It returns $(\mathsf{sid}, \textsc{Encrypted}, C)$ to P.

▪ Upon receiving $(\mathsf{sid}, \textsc{Dec}, c, \tau)$ from $P \notin \mathbf{P}_{\mathrm{corr}}$, if $c \neq \mathsf{Null}$:

(1) If $\tau < 0$, it returns $(\mathsf{sid}, \textsc{Dec}, c, \tau, \perp)$ to P. Else, it reads the time Cl from $\mathcal{G}_{\mathrm{clock}}$ and:
  (a) If $\mathsf{Cl} < \tau$, it sends $(\mathsf{sid}, \textsc{Dec}, c, \tau, \textsc{More\_Time})$ to P.
  (b) If $\mathsf{Cl} \geq \tau$, then
    – If there are two tuples $(M_1, c, \tau_1, \cdot, \cdot, \cdot), (M_2, c, \tau_2, \cdot, \cdot, \cdot)$ in L_{rec} such that $M_1 \neq M_2$ and $c \neq \mathsf{Null}$ where $\tau \geq \max\{\tau_1, \tau_2\}$, it returns to P $(\mathsf{sid}, \textsc{Dec}, c, \tau, \perp)$.
    – If no tuple $(\cdot, c, \cdot, \cdot, \cdot, \cdot)$ is recorded in L_{rec}, it sends $(\mathsf{sid}, \textsc{Dec}, c, \tau)$ to $\mathcal{S}$. Upon receiving $(\mathsf{sid}, \textsc{Dec}, c, \tau, M)$ back from $\mathcal{S}$ it stores $(M, c, \tau, \mathsf{Null}, 0, \mathsf{Null})$ in L_{rec} and returns $(\mathsf{sid}, \textsc{Dec}, c, \tau, M)$ to P.
    – If there is a unique tuple $(M, c, \tau_{\mathrm{dec}}, \cdot, \cdot, \cdot)$ in L_{rec}, then if $\tau \geq \tau_{\mathrm{dec}}$, it returns (sid, $\textsc{Dec}, c, \tau, M)$ to P. Else, if $\mathsf{Cl} < \tau_{\mathrm{dec}}$, it returns (sid, $\textsc{Dec}, c, \tau, \textsc{More\_Time})$ to P. Else, if $\mathsf{Cl} \geq \tau_{\mathrm{dec}} > \tau$, it returns $(\mathsf{sid}, \textsc{Dec}, c, \tau, \textsc{Invalid\_Time})$ to P.

▪ Upon receiving $(\mathsf{sid}, \textsc{Leakage})$ from $\mathcal{S}$, it reads the time Cl from $\mathcal{G}_{\mathrm{clock}}$ and returns $(\mathsf{sid}, \textsc{Leakage}, (\{(M, c, \tau)\}_{\forall (M,c,\tau,\cdot,\cdot,\cdot) \in L_{\mathrm{rec}}: \tau \leq \mathsf{leak}(\mathsf{Cl})} \cup \{(M, c, \tau, \mathsf{tag}, \mathsf{Cl}, P) \in L_{\mathrm{rec}}\}_{\forall P \in \mathbf{P}_{\mathrm{corr}}}))$ to $\mathcal{S}$.

Figure 3: The functionality $\mathcal{F}_{\mathsf{TLE}}^{\mathsf{leak},\mathsf{delay}}$ parameterized by the security parameter λ, a leakage function leak, a delay variable delay, interacting with the parties in P, the simulator $\mathcal{S}$, and global clock $\mathcal{G}_{\mathrm{clock}}$.

The unfair broadcast functionality $\mathcal{F}_{\mathsf{UBC}}(\mathbf{P})$.

The functionality initializes list L_{pend} of pending messages as empty. It also maintains the set of corrupted parties, $\mathbf{P}_{\mathrm{corr}}$, initialized as empty.

▪ Upon receiving $(\mathsf{sid}, \textsc{Broadcast}, M)$ from $P \in \mathbf{P} \setminus \mathbf{P}_{\mathrm{corr}}$, it picks a unique random tag from $\{0,1\}^{\lambda}$, adds the tuple (tag, M, P) to L_{pend} and sends $(\mathsf{sid}, \textsc{Broadcast}, \mathsf{tag}, M, P)$ to $\mathcal{S}$.

▪ Upon receiving $(\mathsf{sid}, \textsc{Broadcast}, M, P)$ from $\mathcal{S}$ on behalf of $P \in \mathbf{P}_{\mathrm{corr}}$, it sends $(\mathsf{sid}, \textsc{Broadcast}, M)$ to all parties and $\mathcal{S}$.

▪ Upon receiving $(\mathsf{sid}, \textsc{Allow}, \mathsf{tag}, \tilde{M})$ from $\mathcal{S}$, if there is a tuple $(\mathsf{tag}, \cdot, P) \in L_{\mathrm{pend}}$ such that $P \in \mathbf{P}_{\mathrm{corr}}$, it does:

(1) It sends $(\mathsf{sid}, \textsc{Broadcast}, \tilde{M})$ to all parties and $(\mathsf{sid}, \textsc{Broadcast}, \tilde{M}, P)$ to $\mathcal{S}$.
(2) It deletes $(\mathsf{tag}, \cdot, P)$ from L_{pend}.

▪ Upon receiving $(\mathsf{sid}_C, \textsc{Advance\_Clock})$ from $P \in \mathbf{P} \setminus \mathbf{P}_{\mathrm{corr}}$ it does:

(1) It reads the time Cl from $\mathcal{G}_{\mathrm{clock}}$. If this is the first time that P has sent a $(\mathsf{sid}_C, \textsc{Advance\_Clock})$ message during round Cl, then for every $(\mathsf{tag}, M, P) \in L_{\mathrm{pend}}$, it does:
  (a) It sends $(\mathsf{sid}, \textsc{Broadcast}, M)$ to all parties and $(\mathsf{sid}, \textsc{Broadcast}, M, P)$ to $\mathcal{S}$.
  (b) It deletes (tag, M, P) from L_{pend}.
(2) It returns $(\mathsf{sid}_C, \textsc{Advance\_Clock})$ to P with destination identity $\mathcal{G}_{\mathrm{clock}}$.

Figure 4: The functionality $\mathcal{F}_{\mathsf{UBC}}$ interacting with the parties in P and the simulator $\mathcal{S}$.

The UBC protocol. In Figure **??**, we present a simple protocol that utilizes multiple instances of $\mathcal{F}_{\mathsf{RBC}}$ (cf. Figure **??**) to realize concurrent unfair broadcast executions. The invocation to the $\mathcal{F}_{\mathsf{RBC}}$ instances replaces the composition of multiple Dolev-Strong runs.

By the description of Π_{UBC}, the Universal Composition Theorem [**?**], and Fact **??**, we get the following lemma.

Lemma 1. *There exists a protocol that UC-realizes $\mathcal{F}_{\mathsf{UBC}}$ in the $(\mathcal{F}_{\mathrm{cert}}, \mathcal{G}_{\mathrm{clock}})$-hybrid model against an adaptive adversary corrupting $t < n$ parties.*

The FBC functionality. Our FBC functionality $\mathcal{F}_{\mathsf{FBC}}^{\Delta,\alpha}$ has the FBC functionality in [**?**] as a reference point, and extends [**?**] to the setting where any party can send of multiple messages per round. In FBC, the adversary (simulator) can receive the sender's message before its broadcasting actually happens. However, even if it adaptively corrupts the sender, the adversary cannot alter the original message that has been "locked" as the intended broadcast value.

The functionality is parameterized by two integers: (i) a *delay* Δ, and (ii) an *advantage* α of the simulator $\mathcal{S}$ to retrieve the broadcast messages compared to the parties. Specifically, if a message is requested to be broadcast at time Cl^*, then $\mathcal{F}_{\mathsf{FBC}}^{\Delta,\alpha}$ will send it to the parties at time $\mathsf{Cl}^* + \Delta$, whereas $\mathcal{S}$ can obtain it at time $\mathsf{Cl}^* + \Delta - \alpha$.

The unfair broadcast protocol $\Pi_{\mathsf{UBC}}(\mathcal{F}_{\mathsf{RBC}}, \mathbf{P})$.

Every party P maintains two counters $\mathsf{total}^P, \mathsf{count}^P$, initialized to 0.

▪ Upon receiving $(\mathsf{sid}, \textsc{Broadcast}, M)$ from $\mathcal{Z}$, the party P does:

(1) She increases count^P and total^P by 1.
(2) She sends $(\mathsf{sid}, \textsc{Broadcast}, M)$ to $\mathcal{F}_{\mathsf{RBC}}^{P,\mathsf{total}^P}$.

▪ Upon receiving $(\mathsf{sid}, \textsc{Broadcast}, M^*, P^*)$ from $\mathcal{F}_{\mathsf{RBC}}^{P^*,\cdot}$, the party P forwards $(\mathsf{sid}, \textsc{Broadcast}, M^*)$ to $\mathcal{Z}$.

▪ Upon receiving $(\mathsf{sid}_C, \textsc{Advance\_Clock})$ from $\mathcal{Z}$, the party P reads the time Cl from $\mathcal{G}_{\mathsf{clock}}$. If this is the first time that she has received a $(\mathsf{sid}_C, \textsc{Advance\_Clock})$ command during round Cl, she does:

(1) For every $j = 1, \ldots, \mathsf{count}^P$, she sends $(\mathsf{sid}_C, \textsc{Advance\_Clock})$ to $\mathcal{F}_{\mathsf{RBC}}^{P,\mathsf{total}^P-(\mathsf{count}^P-j)}$. Namely, P instructs $\mathcal{F}_{\mathsf{RBC}}^{P,\mathsf{total}^P-(\mathsf{count}^P-j)}$ to broadcast her j-th message for the current round Cl.
(2) She resets count^P to 0.
(3) She forwards $(\mathsf{sid}_C, \textsc{Advance\_Clock})$ to $\mathcal{G}_{\mathsf{clock}}$.

Figure 5: The protocol Π_{UBC} with the parties in P.

3.2 Fair broadcast definition and realization

The functionality associates each Broadcast request with a unique random tag, marks the request as "pending", and informs $\mathcal{S}$ of the senders' activity by leaking the tag and the sender's identity to $\mathcal{S}$. After $\Delta - \alpha$ rounds, $\mathcal{S}$ can perform an Output_Request and obtain the message that corresponds to a specific tag. However, at this point and unlike in UBC, the message becomes "locked" and $\mathcal{S}$ cannot alter it with a message of its choice, even if the sender gets adaptively corrupted. Besides, by performing a Corruption_Request, $\mathcal{S}$ can obtain the pending messages of all corrupted parties, so that it can update the state of the corresponding simulated party with the actual pending messages. The simulator may change the original message of a broadcast request with a value of its choice only if (i) the associated sender is corrupted and (ii) the original message is not locked. The message delivery to the parties happens when the parties forward an Advance_Clock message for the round that is Δ time after the broadcast request occurred. The functionality is formally presented in Figure **??**.

The FBC protocol. The (stand-alone) FBC protocol proposed in [**?**] is not UC secure. In Figure **??**, we present our protocol that realizes concurrent fair broadcast executions. As in [**?**], we deploy (a) UBC, (b) time-lock puzzles (instantiated by the TLE algorithms in [**?**]) to achieve broadcast fairness, and (c) a programmable RO to allow equivocation (also applied in [**? ? ?**]).

In order to construct FBC in a setting with recurring and arbitrary scheduled broadcast executions, several technical issues arise. The key challenge here is to ensure that messages are retrieved by all parties in the same round. Our protocol carefully orchestrates TLE encryption, emission, reception, and TLE decryption of messages broadcast in UBC manner w.r.t. the global clock to achieve this. The UC-secure protocol Π_{FBC} encompasses the following key features:

The fair broadcast functionality $\mathcal{F}_{\mathsf{FBC}}^{\Delta,\alpha}(\mathbf{P})$.

The functionality initializes the list L_{pend} of (unlocked) pending messages as empty, the list L_{lock} of locked messages as empty, and a variable Output as $\perp$. It also maintains the set of corrupted parties, $\mathbf{P}_{\mathsf{corr}}$, initialized as empty.

▪ Upon receiving $(\mathsf{sid}, \textsc{Broadcast}, M)$ from $P \in \mathbf{P} \setminus \mathbf{P}_{\mathsf{corr}}$ or $(\mathsf{sid}, \textsc{Broadcast}, M, P)$ from $\mathcal{S}$ on behalf of $P \in \mathbf{P}_{\mathsf{corr}}$, it reads the time Cl from $\mathcal{G}_{\mathsf{clock}}$, picks a unique random tag from $\{0,1\}^\lambda$, and adds the tuple $(\mathsf{tag}, M, P, \mathsf{Cl})$ to L_{pend}. Then, it sends $(\mathsf{sid}, \textsc{Broadcast}, \mathsf{tag}, P)$ to $\mathcal{S}$.

▪ Upon receiving $(\mathsf{sid}, \textsc{Output\_Request}, \mathsf{tag})$ from $\mathcal{S}$, it reads the time Cl from $\mathcal{G}_{\mathsf{clock}}$. If there is a tuple $(\mathsf{tag}, M, P, \mathsf{Cl}^*) \in L_{\mathsf{pend}}$ such that $\mathsf{Cl} - \mathsf{Cl}^* = \Delta - \alpha$, it adds $(\mathsf{tag}, M, P, \mathsf{Cl}^*)$ to L_{lock}, removes it from L_{pend}, and sends $(\mathsf{sid}, \textsc{Output\_Request}, \mathsf{tag}, M, P, \mathsf{Cl}^*)$ to $\mathcal{S}$.

▪ Upon receiving $(\mathsf{sid}, \textsc{Corruption\_Request})$ from $\mathcal{S}$, it sends $(\mathsf{sid}, \textsc{Corruption\_Request}, \langle(\mathsf{tag}, M, P, \mathsf{Cl}^*) \in L_{\mathsf{pend}} : P \in \mathbf{P}_{\mathsf{corr}}\rangle)$ to $\mathcal{S}$.

▪ Upon receiving $(\mathsf{sid}, \textsc{Allow}, \mathsf{tag}, \tilde{M}, \tilde{P})$ from $\mathcal{S}$, it does:

(1) If there is no tuple $(\mathsf{tag}, M, \tilde{P}, \mathsf{Cl}^*)$ in L_{pend} or L_{lock}, it ignores the message.
(2) If $\tilde{P} \in \mathbf{P} \setminus \mathbf{P}_{\mathsf{corr}}$ or $(\mathsf{tag}, M, \tilde{P}, \mathsf{Cl}^*) \in L_{\mathsf{lock}}$, it ignores the message.
(3) If $\tilde{P} \in \mathbf{P}_{\mathsf{corr}}$ and $(\mathsf{tag}, M, \tilde{P}, \mathsf{Cl}^*) \in L_{\mathsf{pend}}$ (i.e., the message is not locked), it sets $\mathsf{Output} \leftarrow \tilde{M}$. If there is no tuple $(\mathsf{tag}, \cdot, \cdot, \cdot)$ in L_{lock}, it adds $(\mathsf{tag}, \mathsf{Output}, \tilde{P}, \mathsf{Cl}^*)$ to L_{lock} and removes $(\mathsf{tag}, M, \tilde{P}, \mathsf{Cl}^*)$ from L_{pend}. It sends $(\mathsf{sid}, \textsc{Allow\_OK})$ to $\mathcal{S}$.

▪ Upon receiving $(\mathsf{sid}_C, \textsc{Advance\_Clock})$ from $P \in \mathbf{P} \setminus \mathbf{P}_{\mathsf{corr}}$, it does:

(1) It reads the time Cl from $\mathcal{G}_{\mathsf{clock}}$.
(2) Let $L \leftarrow L_{\mathsf{pend}} @ L_{\mathsf{lock}}$ be the concatenation of the two lists. It sorts L lexicographically w.r.t. the second coordinate (i.e. messages) of its tuples.
(3) For every tuple $(\mathsf{tag}^*, M^*, P^*, \mathsf{Cl}^*) \in L$, if $\mathsf{Cl} - \mathsf{Cl}^* = \Delta$, it sends $(\mathsf{sid}, \textsc{Broadcast}, M^*)$ to P.
(4) It returns $(\mathsf{sid}_C, \textsc{Advance\_Clock})$ to P with destination identity $\mathcal{G}_{\mathsf{clock}}$.

Figure 6: The functionality $\mathcal{F}_{\mathsf{FBC}}^{\Delta,\alpha}$ interacting with the parties in P and the simulator $\mathcal{S}$, parameterized by delay Δ and simulator advantage α.

(1) Resource-restriction is formalized via a wrapper $\mathcal{W}_q(\mathcal{F}_{\mathsf{RO}}^*)$ that allows a party or the adversary to perform up to q parallel queries per round (cf. [**? ? ?**] for similar formal treatments).
(2) To take advantage of parallelization that the wrapper offers, parties generate puzzles for creating TLE ciphertexts (and solve the puzzles of the ciphertexts they have received) only when

they are about to complete their round. I.e., when receiving an ADVANCE_CLOCK command by the environment, they broadcast in UBC manner all their messages (TLE encrypted with difficulty set to 2 rounds and equivocated) for the current round. Observe that if without such restriction and allow senders broadcast their messages upon instruction by the environment, then this would lead to an "waste of resources"; so, parties would not be able to broadcast more than q messages per round and/or they would not have any queries left to proceed to puzzle solution.

(3) For realization of $\mathcal{F}_{\mathsf{FBC}}$, a message must be retrieved by all parties in the same round. Hence, we require that parties, when acting as recipients, begin decryption (puzzle solving) *in the round that follows* the one they received the associated TLE ciphertext. Otherwise, the following may happen: let parties A, B, and C complete round Cl first, second, and third, respectively. If B broadcasts an encrypted message M, then, unlike C, A will have exhausted its available resources (RO queries) by the time she receives M. As a result, C is able to retrieve M at round $\mathsf{Cl}+1$ (by making the first set of q RO queries in Cl and the second set in $\mathsf{Cl}+1$) whereas A not earlier than $\mathsf{Cl}+2$ (by making the first set in $\mathsf{Cl}+1$ and the second in $\mathsf{Cl}+2$).

(4) The reason that we impose time difficulty of two rounds instead of just one is rather technical. Namely, if it was set to one round, then the number of queries required for puzzle solution is q. However, a rushing real-world adversary may choose to waste all of its resources to decrypt a TLE ciphertext *in the same round* that the ciphertext was intercepted. In this case, the simulator would not have time for equivocating the randomness hidden in the underlying puzzle and simulation would fail.

The protocol is formally described in Figure ??. The core idea of the construction is the following: to broadcast a message M in a fair manner, the sender chooses a randomness ρ and creates a TLE ciphertext c of ρ. Then, she queries the RO on ρ to receive a response η, computes $M \oplus \eta$, and broadcasts $(c, M \oplus \eta)$ via $\mathcal{F}_{\mathsf{UBC}}$. When decryption time comes, any recipient can decrypt c as ρ, obtain η via a RO query on ρ, and retrieve M by an XOR operation.

In the following lemma, we prove that our FBC protocol UC-realizes $\mathcal{F}_{\mathsf{FBC}}^{\Delta,\alpha}$ for delay $\Delta = 2$ and advantage $\alpha = 2$. Namely, the parties retrieve the messages after two rounds and the simulator two rounds earlier (i.e., in the same round).

The fair broadcast protocol $\Pi_{\mathsf{FBC}}(\mathcal{F}_{\mathsf{UBC}}, \mathcal{W}_q(\mathcal{F}^*_{\mathsf{RO}}), \mathcal{F}_{\mathsf{RO}}, \mathbf{P})$.

The protocol utilizes the TLE algorithms $(\mathsf{AST.Enc}, \mathsf{AST.Dec})$ described in [?]. Every party P maintains (i) a list L^P_{pend} of messages pending to be broadcast, (ii) a list L^P_{wait} of received ciphertexts waiting to be decrypted, and (iii) a list L^P of messages ready to be delivered. All three lists are initialized as empty.

▪ Upon receiving (sid, BROADCAST, M) from $\mathcal{Z}$, P adds M to L^P_{pend}.

▪ Upon receiving (sid, BROADCAST, (c^*, y^*)) from $\mathcal{F}_{\mathsf{UBC}}$, P reads the time Cl from $\mathcal{G}_{\mathsf{clock}}$ and adds (c^*, y^*, Cl) to L^P_{wait}.

▪ Upon receiving (sid, ADVANCE_CLOCK) from $\mathcal{Z}$, the party P reads the time Cl from $\mathcal{G}_{\mathsf{clock}}$. If this is the first time that P has received (sid, ADVANCE_CLOCK) for time Cl, she does:

(1) For every M in L^P_{pend}, she picks puzzle randomness $r^M_0 || \cdots || r^M_{2q-1} \xleftarrow{\$} (\{0,1\}^\lambda)^{2q}$.
(2) For every $(c^*, y^*, \mathsf{Cl}-1)$ in L^P_{wait}, she parses c^* as $(2, c^*_2, c^*_3)$ and c^*_3 as $(r^*_0, z^*_1, \ldots, z^*_{2q})$. For every $(c^{**}, y^{**}, \mathsf{Cl}-2)$ in L^P_{wait}, she parses c^{**} as $(2, c^{**}_2, c^{**}_3)$ and c^*_3 as $(r^{**}_0, z^{**}_1, \ldots, z^{**}_{2q})$.
(3) She makes all available q queries $Q_0, \ldots, Q_{q-1}$ to $\mathcal{W}_q(\mathcal{F}^*_{\mathsf{RO}})$ for Cl and gets responses $R_0, \ldots, R_{q-1}$, respectively, where
   - $Q_0 = \big(\cup_{M \in L^P_{\mathsf{pend}}} \{r^M_0, \ldots, r^M_{2q-1}\}\big) \bigcup \big(\cup_{(c^*,y^*,\mathsf{Cl}-1)\in L^P_{\mathsf{wait}}} \{r^*_0\}\big) \bigcup \big(\cup_{(c^{**},y^{**},\mathsf{Cl}-2)\in L^P_{\mathsf{wait}}} \{z^{**}_q \oplus h^{**}_{q-1}\}\big)$.
   - $R_0 = \big(\cup_{M \in L^P_{\mathsf{pend}}} \{h^M_0, \ldots, h^M_{2q-1}\}\big) \bigcup \big(\cup_{(c^*,y^*,\mathsf{Cl}-1)\in L^P_{\mathsf{wait}}} \{h^*_0\}\big) \bigcup \big(\cup_{(c^{**},y^{**},\mathsf{Cl}-2)\in L^P_{\mathsf{wait}}} \{h^{**}_q\}\big)$.
   - For $j \geq 1$, $Q_j = \big(\cup_{(c^*,y^*,\mathsf{Cl}-1)\in L^P_{\mathsf{wait}}} \{z^*_j \oplus h^*_{j-1}\}\big) \bigcup \big(\cup_{(c^{**},y^{**},\mathsf{Cl}-2)\in L^P_{\mathsf{wait}}} \{z^{**}_{j+q} \oplus h^{**}_{j+q-1}\}\big)$.
   - For $j \geq 1$, $R_j = \big(\cup_{(c^*,y^*,\mathsf{Cl}-1)\in L^P_{\mathsf{wait}}} \{h^*_j\}\big) \bigcup \big(\cup_{(c^{**},y^{**},\mathsf{Cl}-2)\in L^P_{\mathsf{wait}}} \{h^{**}_{j+q}\}\big)$.[a]
(4) For every M in L^P_{pend}:
   (a) She chooses a random value ρ from the TLE message space;
   (b) She encrypts as $c \leftarrow \mathsf{AST.Enc}(\rho, 2)$ using RO responses $(h^M_0, \ldots, h^M_{2q-1})$ obtained by querying $\mathcal{W}_q(\mathcal{F}^*_{\mathsf{RO}})$ on Q_0.
   (c) She queries $\mathcal{F}_{\mathsf{RO}}$ on ρ and receives a response η.
   (d) She computes $y \leftarrow M \oplus \eta$.
   (e) She deletes M from L^P_{pend}, and sends (sid, BROADCAST, (c, y)) to $\mathcal{F}_{\mathsf{UBC}}$.
(5) For every $(c^{**}, y^{**}, \mathsf{Cl}-2)$ in L^P_{wait}:
   (a) She sets the decryption witness as $w^{**}_2 \leftarrow (h^{**}_0, \ldots, h^{**}_{q-1})$.
   (b) She computes $\rho^{**} \leftarrow \mathsf{AST.Dec}(c^{**}, w^{**}_2)$.
   (c) She queries $\mathcal{F}_{\mathsf{RO}}$ on ρ^{**} and receives a response η^{**}.
   (d) She computes $M^{**} \leftarrow y^{**} \oplus \eta^{**}$ and adds M^{**} to L^P.
   (e) She deletes $(c^{**}, y^{**}, \mathsf{Cl}-2)$ from L^P_{wait}.
(6) She sorts L^P lexicographically.
(7) For every M^{**} in L^P, she returns (sid, BROADCAST, M^{**}) to $\mathcal{Z}$.
(8) She resets L^P as empty.
(9) She sends (sid$_C$, ADVANCE_CLOCK) to $\mathcal{F}_{\mathsf{UBC}}$. Upon receiving (sid$_C$, ADVANCE_CLOCK) from $\mathcal{F}_{\mathsf{UBC}}$, she forwards (sid$_C$, ADVANCE_CLOCK) to $\mathcal{G}_{\mathsf{clock}}$ and completes her round.

[a] Namely, the first query includes all puzzle generation queries required for the TLE of every message that will be broadcast by P. The j-th query includes (i) all j-th step puzzle solving queries for decrypting messages received in round $\mathsf{Cl}-1$ and (ii) all $(q+j)$-step puzzle solving queries for decrypting messages received in round $\mathsf{Cl}-2$. The queries are computed as described in Subsection ??. As a result, the decryption witness for each TLE ciphertext can be computed in two rounds (upon completing all necessary $2q$ hashes).

Figure 6: The protocol Π_{FBC} with the parties in P.

LEMMA 2. *The protocol* Π_{FBC} *in Figure* ?? *UC-realizes* $\mathcal{F}^{2,2}_{\mathsf{FBC}}$ *in the* $(\mathcal{F}_{\mathsf{UBC}}, \mathcal{W}_q(\mathcal{F}^*_{\mathsf{RO}}), \mathcal{F}_{\mathsf{RO}}, \mathcal{G}_{\mathsf{clock}})$*-hybrid model against an adaptive adversary corrupting* $t < n$ *parties.*

4 UC TIME-LOCK ENCRYPTION AGAINST ADAPTIVE ADVERSARIES

In this section, we strengthen the main result of [?] (cf. Fact ??), presenting the first UC realization of $\mathcal{F}_{\mathsf{TLE}}$ against adaptive adversaries. Specifically, we prove that the TLE construction in [?] is UC secure when deploying $\mathcal{F}_{\mathsf{FBC}}$ as the hybrid that establishes communication among parties. In more details, the TLE construction in [?] requires that an encryptor broadcasts her TLE ciphertext to all other parties to allow them to begin solving the associated time-lock puzzle for decryption. The following theorem shows that FBC is sufficient to guarantee adaptive security of the TLE protocol.

Theorem 1. *Let Δ, α be integers s.t. $\Delta \geq \alpha \geq 0$. The protocol Π_{TLE} in Figure ?? UC-realizes $\mathcal{F}_{\mathsf{TLE}}^{\mathsf{leak},\mathsf{delay}}$ in the $(\mathcal{W}_q(\mathcal{F}^*_{\mathsf{RO}}), \mathcal{F}_{\mathsf{RO}}, \mathcal{F}_{\mathsf{FBC}}^{\Delta,\alpha}, \mathcal{G}_{\mathsf{clock}})$-hybrid model, where $\mathsf{leak}(\mathsf{Cl}) = \mathsf{Cl} + \alpha$ and $\mathsf{delay} = \Delta + 1$.*

5 SIMULTANEOUS BROADCAST

In this section, we present our formal study of the simultaneous broadcast (SBC) notion in the UC framework, which comprises a new functionality $\mathcal{F}_{\mathsf{SBC}}$ and a TLE-based construction that we prove it UC-realizes $\mathcal{F}_{\mathsf{SBC}}$. Our approach revisits and improves upon the work of Hevia [?] w.r.t. several aspects. In particular,

(1) We consider SBC executions where honest parties agree on a well-defined *broadcast period*, outside of which all broadcast messages are ignored. This setting is plausible, as simultaneity suggests that no sender broadcasts a message depending on the messages broadcast by other parties. If there is no such broadcast period, then liveness and simultaneity are in conflict, as a malicious sender could wait indefinitely until all honest parties are forced to either (i) abort, or (ii) reveal their messages before all (malicious) senders broadcast their values. On the contrary, within an agreed valid period, honest parties can safely broadcast knowing that every invalid message will be discarded. Unlike [?], participation of all parties is not necessary for the termination of the protocol execution. In Section ??, we propose practical use cases where our SBC setting is greatly desired.

(2) The SBC functionality of [?] is designed w.r.t. the synchronous communication setting in [?]. As shown in [?], this setting has limitations (specifically, guarantee of termination) that are lifted when using $\mathcal{G}_{\mathsf{clock}}$. In our formal treatment, synchronicity is captured in the state-of-the-art $\mathcal{G}_{\mathsf{clock}}$-hybrid model.

(3) The SBC construction in [?] is proven secure only against adversaries that corrupt a minority of all parties. By utilizing TLE, our work introduces the first SBC protocol that is UC secure against any adversary corrupting $t < n$ parties.

The SBC functionality. Our SBC functionality $\mathcal{F}_{\mathsf{SBC}}$ (cf. Figure ??) interacts with $\mathcal{G}_{\mathsf{clock}}$ and is parameterized by a broadcast time span Φ, a message delivery delay Δ and a simulator advantage α. Upon first Broadcast request, it sets the current global time as the beginning of the broadcast period that lasts Φ rounds. If a Broadcast request was made by an honest sender P, then the functionality leaks only the sender's identity. All Broadcast requests are recorded as long as they are made within the broadcast period. The recorded messages are issued to each party (resp. the simulator) Δ rounds (resp. $\Delta - \alpha$ rounds) after the end of the period.

The simultaneous broadcast functionality $\mathcal{F}_{\mathsf{SBC}}^{\Phi,\Delta,\alpha}(\mathbf{P})$.

The functionality initializes the list L_{pend} of pending messages as empty and two variables $t_{\mathsf{start}}, t_{\mathsf{end}}$ to $\perp$. It also maintains the set of corrupted parties, $\mathbf{P}_{\mathsf{corr}}$, initialized as empty.

▪ Upon receiving $(\mathsf{sid}, \textsc{Broadcast}, M)$ from $P \in \mathbf{P} \setminus \mathbf{P}_{\mathsf{corr}}$ or $(\mathsf{sid}, \textsc{Broadcast}, M, P)$ from $\mathcal{S}$ on behalf of $P \in \mathbf{P}_{\mathsf{corr}}$, it does:

(1) It reads the time Cl from $\mathcal{G}_{\mathsf{clock}}$.
(2) If $t_{\mathsf{start}} = \perp$, it sets $t_{\mathsf{start}} \leftarrow \mathsf{Cl}$ and $t_{\mathsf{end}} \leftarrow t_{\mathsf{start}} + \Phi$.
(3) If $t_{\mathsf{start}} \leq \mathsf{Cl} < t_{\mathsf{end}}$, it does :
  (a) It picks a unique random tag from $\{0,1\}^{\lambda}$.
  (b) If $P \in \mathbf{P} \setminus \mathbf{P}_{\mathsf{corr}}$, it adds $(\mathsf{tag}, M, P, \mathsf{Cl}, 0)$ to L_{pend} and sends $(\mathsf{sid}, \textsc{Sender}, \mathsf{tag}, 0^{|M|}, P)$ to $\mathcal{S}$. Otherwise, it adds $(\mathsf{tag}, M, P, \mathsf{Cl}, 1)$ to L_{pend} and sends $(\mathsf{sid}, \textsc{Sender}, \mathsf{tag}, M, P)$ to $\mathcal{S}$.

▪ Upon receiving $(\mathsf{sid}, \textsc{Corruption\_Request})$ from $\mathcal{S}$, it sends $(\mathsf{sid}, \textsc{Corruption\_Request}, \langle(\mathsf{tag}, M, P, \mathsf{Cl}^*, 0) \in L_{\mathsf{pend}} : P \in \mathbf{P}_{\mathsf{corr}}\rangle)$ to $\mathcal{S}$.

▪ Upon receiving $(\mathsf{sid}, \textsc{Allow}, \mathsf{tag}, \tilde{M}, \tilde{P})$ from $\mathcal{S}$, it does:

(1) It reads the time Cl from $\mathcal{G}_{\mathsf{clock}}$.
(2) If $t_{\mathsf{start}} \leq \mathsf{Cl} < t_{\mathsf{end}}$ and there is a tuple $(\mathsf{tag}, M, \tilde{P}, \mathsf{Cl}^*, 0) \in L_{\mathsf{pend}}$ and $\tilde{P} \in \mathbf{P}_{\mathsf{corr}}$, it updates the tuple as $(\mathsf{tag}, \tilde{M}, \tilde{P}, \mathsf{Cl}^*, 1)$ and sends $(\mathsf{sid}, \textsc{Allow\_OK})$ to $\mathcal{S}$. Otherwise, it ignores the message.

▪ Upon receiving $(\mathsf{sid}_C, \textsc{Advance\_Clock})$ from $P \in \mathbf{P} \setminus \mathbf{P}_{\mathsf{corr}}$, it does:

(1) It reads the time Cl from $\mathcal{G}_{\mathsf{clock}}$.
(2) If this is the first time it has received a $(\mathsf{sid}_C, \textsc{Advance\_Clock})$ message from P during round Cl, then
  (a) If it has received no other $(\mathsf{sid}_C, \textsc{Advance\_Clock})$ message during round Cl,
    (i) If $\mathsf{Cl} = t_{\mathsf{end}}$, then it does:
      (A) It updates every tuple $(\cdot, \cdot, P^*, \cdot, 0) \in L_{\mathsf{pend}}$ such that $P^* \in \mathbf{P} \setminus \mathbf{P}_{\mathsf{corr}}$ as $(\cdot, \cdot, P^*, \cdot, 1)$ (to guarantee the broadcast of messages from always honest parties).
      (B) It sorts L_{pend} lexicographically according to the second coordinate (messages).
    (ii) If $\mathsf{Cl} = t_{\mathsf{end}} + \Delta - \alpha$, it sends $(\mathsf{sid}, \textsc{Broadcast}, \langle(\mathsf{tag}, M)\rangle_{(\mathsf{tag}, M, \cdot, \cdot, 1) \in L_{\mathsf{pend}}})$ to $\mathcal{S}$.
  (b) If $\mathsf{Cl} = t_{\mathsf{end}} + \Delta$, it sends $(\mathsf{sid}, \textsc{Broadcast}, \langle M\rangle_{(\cdot, M, \cdot, \cdot, 1) \in L_{\mathsf{pend}}})$ to P.
(3) It returns $(\mathsf{sid}_C, \textsc{Advance\_Clock})$ to P with destination identity $\mathcal{G}_{\mathsf{clock}}$.

Figure 7: The functionality $\mathcal{F}_{\mathsf{SBC}}^{\Phi,\Delta,\alpha}$ interacting with the parties in P and the simulator $\mathcal{S}$, parameterized by time span Φ, delay Δ, and simulator advantage α.

The SBC protocol $\Pi_{\mathsf{SBC}}(\mathcal{F}_{\mathsf{UBC}}, \mathcal{F}_{\mathsf{TLE}}^{\mathsf{leak},\mathsf{delay}}, \mathcal{F}_{\mathsf{RO}}, \Phi, \Delta, \mathbf{P})$.

Every party P maintains a list L^{P}_{pend} of messages under pending encryption and a list L^{P}_{rec} of received ciphertexts both initialized as empty, and four variables t^{P}_{awake}, t^{P}_{end}, τ^{P}_{rel}, first^{P}, all initialized to $\perp$. All parties understand a special message 'Wake_Up' that is not in the broadcast message space.

▪ Upon receiving (sid, Broadcast, M) from $\mathcal{Z}$, the party P does:

(1) If $t^{P}_{\mathsf{awake}} = \perp$, she sets $\mathsf{first}^{P} \leftarrow M$ and sends (sid, Broadcast, Wake_Up) to $\mathcal{F}_{\mathsf{UBC}}$.
(2) If $t^{P}_{\mathsf{awake}} \neq \perp$, she does:
  (a) She reads the time Cl from $\mathcal{G}_{\mathsf{clock}}$.
  (b) If $\mathsf{Cl} \geq t^{P}_{\mathsf{end}} - \mathsf{delay}$, she ignores the message[a].
  (c) She chooses a randomness $\rho \xleftarrow{\$} \{0,1\}^{\lambda}$.
  (d) She adds (ρ, M) in L^{P}_{pend}.
  (e) She sends (sid, Enc, ρ, τ^{P}_{rel}) to $\mathcal{F}_{\mathsf{TLE}}^{\mathsf{leak},\mathsf{delay}}$.

▪ Upon receiving (sid, Broadcast, Wake_Up) from $\mathcal{F}_{\mathsf{UBC}}$, if $t^{P}_{\mathsf{awake}} = \perp$, the party P does:

(1) She reads the time Cl from $\mathcal{G}_{\mathsf{clock}}$.
(2) She sets $t^{P}_{\mathsf{awake}} \leftarrow \mathsf{Cl}$, $t^{P}_{\mathsf{end}} \leftarrow t^{P}_{\mathsf{awake}} + \Phi$, and $\tau^{P}_{\mathsf{rel}} \leftarrow t^{P}_{\mathsf{end}} + \Delta$ (i.e., all parties agree on the start and end of the broadcast period, as well as the time-lock decryption time).
(3) If $\mathsf{first}^{P} \neq \perp$, she parses the (unique) pair in L^{P}_{pend} that contains first^{P} as $(\rho, \mathsf{first}^{P})$. Then, she sends (sid, Enc, ρ, τ^{P}_{rel}) to $\mathcal{F}_{\mathsf{TLE}}^{\mathsf{leak},\mathsf{delay}}$ (this check is true only if P broadcasts her first message when acting as the first sender in the session).

▪ Upon receiving (sid, Broadcast, (c^*, τ^*, y^*)) from $\mathcal{F}_{\mathsf{UBC}}$, if $\tau^* = \tau^{P}_{\mathsf{rel}}$ and for every $(c', y') \in L^{P}_{\mathsf{rec}} : c' \neq c^* \wedge y' \neq y^*$, then the party P adds (c^*, y^*) to L^{P}_{rec}.

▪ Upon receiving (sid_C, Advance_Clock) by $\mathcal{Z}$, the party P does:

(1) She reads the time Cl from $\mathcal{G}_{\mathsf{clock}}$. If this is not the first time she has received a (sid_C, Advance_Clock) command during round Cl, she ignores the message.
(2) If $t^{P}_{\mathsf{awake}} \leq \mathsf{Cl} < t^{P}_{\mathsf{end}}$, she sends (sid, Retrieve) to $\mathcal{F}_{\mathsf{TLE}}^{\mathsf{leak},\mathsf{delay}}$ to obtain the encryptions of messages that she requested delay rounds earlier. Upon receiving (sid, Encrypted, T) from $\mathcal{F}_{\mathsf{TLE}}^{\mathsf{leak},\mathsf{delay}}$, she does:
  (a) She parses T as a list of tuples of the form $(\rho, c, \tau^{P}_{\mathsf{rel}})$.
  (b) For every $(\rho, c, \tau^{P}_{\mathsf{rel}}) \in T$ such that there is a pair $(\rho, M) \in L^{P}_{\mathsf{pend}}$, she does:
    (i) She queries $\mathcal{F}_{\mathsf{RO}}$ on ρ and receives a response η.
    (ii) She computes $y \leftarrow M \oplus \eta$.
    (iii) She sends (sid, Broadcast, $(c, \tau^{P}_{\mathsf{rel}}, y)$) to $\mathcal{F}_{\mathsf{UBC}}$.
(3) If $\mathsf{Cl} = \tau^{P}_{\mathsf{rel}}$, then for every $(c^*, y^*) \in L^{P}_{\mathsf{rec}}$, she does:
  (a) She sends (sid, Dec, c^*, τ^{P}_{rel}) to $\mathcal{F}_{\mathsf{TLE}}^{\mathsf{leak},\mathsf{delay}}$. Upon receiving (sid, Dec, c^*, τ^{P}_{rel}, ρ^*) from $\mathcal{F}_{\mathsf{TLE}}^{\mathsf{leak},\mathsf{delay}}$, if $\rho^* \notin \{\perp, \text{More\_Time}, \text{Invalid\_Time}\}$, she queries $\mathcal{F}_{\mathsf{RO}}$ on ρ^* and receives a response η^*.
  (b) She computes $M^* \leftarrow y^* \oplus \eta^*$.
  (c) She sends (sid, Broadcast, M^*) to $\mathcal{Z}$.
(4) She sends (sid_C, Advance_Clock) to $\mathcal{F}_{\mathsf{UBC}}$. Upon receiving ($\mathsf{sid}_C$, Advance_Clock) from $\mathcal{F}_{\mathsf{UBC}}$, she forwards ($\mathsf{sid}_C$, Advance_Clock) to $\mathcal{G}_{\mathsf{clock}}$ and completes her round.

[a]The reason is that due to TLE ciphertext generation time (delay rounds), if $\mathsf{Cl} \geq t^{P}_{\mathsf{end}} - \mathsf{delay}$, then the message would not be ready for broadcast before t^{P}_{end}.

Figure 8: The protocol Π_{SBC} with the parties in P.

The SBC protocol. Our SBC protocol (cf. Figure **??**) is over $\mathcal{F}_{\mathsf{UBC}}$ and deploys $\mathcal{F}_{\mathsf{TLE}}^{\mathsf{leak},\mathsf{delay}}$ to achieve simultaneity, and $\mathcal{F}_{\mathsf{RO}}$ for equivocation. In the beginning, the first sender notifies via $\mathcal{F}_{\mathsf{UBC}}$ the other parties of the start of the broadcast period via a special 'Wake_Up' message. By the properties of UBC, all honest parties agree on the time frame of the broadcast period that lasts Φ rounds. During the broadcast period, in order to broadcast a message M, the sender chooses a randomness ρ and interacts with $\mathcal{F}_{\mathsf{TLE}}^{\mathsf{leak},\mathsf{delay}}$ to obtain a TLE ciphertext c of ρ (after delay rounds). By default, c is set to be decrypted Δ rounds after the end of the broadcast period. Then, she makes an RO query for ρ, receives a response η and broadcasts c and $M \oplus \eta$ via $\mathcal{F}_{\mathsf{UBC}}$. Any recipient of $c, M \oplus \eta$ can retrieve the message Δ rounds after the end of the broadcast period by (i) obtaining ρ via a decryption request of c to $\mathcal{F}_{\mathsf{TLE}}^{\mathsf{leak},\mathsf{delay}}$, (ii) obtaining η as a RO response to query ρ, and (iii) computing $M \leftarrow (M \oplus \eta) \oplus \eta$.

Theorem 2. *Let* $\mathsf{leak}(\cdot)$, delay *be the leakage and delay parameters of* $\mathcal{F}_{\mathsf{TLE}}$. *Let* Φ, Δ *be positive integers such that* $\Phi > \mathsf{delay}$ *and* $\Delta > \max_{\mathsf{Cl}^*}\{\mathsf{leak}(\mathsf{Cl}^*) - \mathsf{Cl}^*\}$. *The protocol* Π_{SBC} *in Figure* **??** *UC-realizes* $\mathcal{F}_{\mathsf{SBC}}^{\Phi,\Delta,\alpha}$ *in the* $(\mathcal{F}_{\mathsf{UBC}}, \mathcal{F}_{\mathsf{TLE}}^{\mathsf{leak},\mathsf{delay}}, \mathcal{F}_{\mathsf{RO}}, \mathcal{G}_{\mathsf{clock}})$*-hybrid model against an adaptive adversary corrupting* $t < n$ *parties, where the simulator advantage is* $\alpha = \max_{\mathsf{Cl}^*}\{\mathsf{leak}(\mathsf{Cl}^*) - \mathsf{Cl}^*\} + 1$.

Corollary 1. *There exists a protocol that UC-realises* $\mathcal{F}_{\mathsf{SBC}}^{\Phi,\Delta,\alpha}$ *in the* $(\mathcal{F}_{\mathsf{cert}}, \mathcal{W}_q(\mathcal{F}^*_{\mathsf{RO}}), \mathcal{F}_{\mathsf{RO}}, \mathcal{G}_{\mathsf{clock}})$*-hybrid model, where* $\Phi > 3$, $\Delta > 2$, *and* $\alpha = 3$.

6 APPLICATIONS OF SBC

6.1 Distributed random string generation

The delayed uniform random string (DURS) functionality. The DURS functionality is along the lines of the common reference string (CRS) functionality in [?]. The functionality draws a single random string r uniformly at random, and delivers r upon request. The delivery of r is delayed, in the sense that the party who made an early request has to wait until Δ time has elapsed since the first request was made. Besides, the simulator has an advantage α, i.e., it can obtain r (on behalf of some corrupted party) when $\Delta - \alpha$ time has elapsed since the first request was made. The DURS functionality is presented in detail the full version.

The DURS protocol. As a first application, we propose a protocol that employs SBC to realize the DURS functionality described above. The idea is simple: each party contributes its randomness by broadcasting it via SBC to other parties. After SBC is finalized (with delay Δ), all parties agree on the XOR of the received random strings as the generated URS. In addition, the parties agree on the beginning of the URS generation period via a special 'Wake_Up' message broadcast in RBC manner by the first activated party.

Theorem 3. *Let* Δ, Φ, α *be non-negative integers such that* $\Delta > \Phi > 0$ *and* $\Delta - \Phi \geq \alpha$. *The protocol* Π_{DURS} *in Figure* **??** *UC-realizes* $\mathcal{F}_{\mathsf{DURS}}^{\Delta,\alpha}$ *in the* $(\mathcal{F}_{\mathsf{SBC}}^{\Phi,\Delta-\Phi,\alpha}, \mathcal{F}_{\mathsf{RBC}}, \mathcal{G}_{\mathsf{clock}})$*-hybrid model against an adaptive adversary corrupting* $t < n$ *parties.*

6.2 Self-tallying e-voting

The concept of self-tallying elections was introduced by Kiayias and Yung in [?]. In this paradigm, the post-ballot-casting (tally)

The DURS protocol $\Pi_{\mathsf{DURS}}(\mathcal{F}_{\mathsf{SBC}}^{\Phi,\Delta-\Phi,\alpha}, \mathcal{F}_{\mathsf{RBC}}, \mathbf{P})$.

Each party P maintains a variable urs^P initialized to $\perp$ and two flags $f_{\mathsf{wait}}^P, f_{\mathsf{awake}}^P$, initialized to 0.

- Upon receiving (sid, URS) from $\mathcal{Z}$, the party P does:
  (1) If $\mathsf{urs}^P \neq \perp$, it returns (sid, URS, urs^P) to $\mathcal{Z}$. Else,
    (a) If $f_{\mathsf{wait}}^P = 0$, she sets $f_{\mathsf{wait}}^P \leftarrow 1$.
    (b) If $f_{\mathsf{awake}}^P = 0$, she sends (sid, Broadcast, Wake_Up) to $\mathcal{F}_{\mathsf{RBC}}^P$.
- Upon receiving (sid, Broadcast, Wake_Up, P^*) from $\mathcal{F}_{\mathsf{RBC}}^{P^*}$, if $P^* \in \mathbf{P}$ and $f_{\mathsf{awake}}^P = 0$, the party P does:
  (1) She sets $f_{\mathsf{awake}}^P \leftarrow 1$.
  (2) She chooses a randomness $\rho \overset{\$}{\leftarrow} \{0,1\}^\lambda$.
  (3) She sends (sid, Broadcast, ρ) to $\mathcal{F}_{\mathsf{SBC}}^{\Phi,\Delta-\Phi,\alpha}$.
- Upon receiving (sid_C, Advance_Clock) from $\mathcal{Z}$, the party P does:
  (1) If $f_{\mathsf{awake}}^P = 0$, she sends (sid_C, Advance_Clock) to $\mathcal{F}_{\mathsf{RBC}}^P$.
    (a) If $\mathcal{F}_{\mathsf{RBC}}^P$ responds with (sid, Broadcast, Wake_Up, P), she executes steps ??-?? from the Broadcast interface above.
    (b) She sends (sid_C, Advance_Clock) to $\mathcal{G}_{\mathsf{clock}}$.
  Otherwise, she sends (sid_C, Advance_Clock) to $\mathcal{F}_{\mathsf{SBC}}^{\Phi,\Delta-\Phi,\alpha}$. Upon receiving the token from $\mathcal{F}_{\mathsf{SBC}}^{\Phi,\Delta-\Phi,\alpha}$, she sends ($\mathsf{sid}_C$, Advance_Clock) to $\mathcal{G}_{\mathsf{clock}}$.
- Upon receiving (sid, Broadcast, $\langle \rho_1, \dots \rho_k \rangle$) from $\mathcal{F}_{\mathsf{SBC}}^{\Phi,\Delta-\Phi,\alpha}$, if $\mathsf{urs}^P = \perp$, the party P does:
  (1) She sets $\mathsf{urs}^P \leftarrow \bigoplus_{i \in [k]: \rho_i \in \{0,1\}^\lambda} \rho_i$.
  (2) If $f_{\mathsf{wait}}^P = 1$, she sends (sid, URS, urs^P) to $\mathcal{Z}$.

Figure 9: The DURS protocol Π_{DURS} with parties in P.

phase can be performed by any party, removing the need for tallier designation. It was further improved by Groth in [?], and later studied in the UC framework by Szepieniec and Preneel in [?]. To ensure fairness, *i.e.* to prevent intermediary results from being leaked before the end of the casting phase, all these previous works introduce a *trusted control voter* that casts a dummy ballot last, contradicting self-tallying in some sense. In this section, we deploy our SBC channel to solve the fairness challenge in self-tallying elections, lifting the need for this trusted control voter.

The voting system (VS) functionality. The ideal voting system functionality, $\mathcal{F}_{VS}^{\Phi,\Delta,\alpha}$ is presented in Figure ??. It is simply the adaptation of Preneel and Szepieniec's functionality to the global clock model and adaptive corruption [?]. The VS functionality only differs from the SBC functionality in that the individually broadcast ballots are not forwarded to the voters (and simulator), but instead the result (tally) of the election is sent to them. Voters submit their votes to the functionality during the casting period that lasts Φ

The self-tallying protocol $\Pi_{\mathsf{STVS}}(\mathcal{F}_{\mathsf{SBC}}^{\Phi,\Delta,\alpha}, \mathcal{F}_{\mathsf{RBC}}, \mathcal{F}_{\mathsf{PKG}}, \mathcal{F}_{\mathsf{SKG}}, \mathbf{V}$

- Initiate by invoking $\mathcal{F}_{\mathsf{PKG}}$ followed by $\mathcal{F}_{\mathsf{SKG}}$.
- All authorities A_j choose random values $x_{i,j} \leftarrow \mathbb{Z}_{n^2}$ for all voters V_i such that $\sum_i x_{i,j} = 0$. They send these values to $\mathcal{F}_{\mathsf{RBC}}$ but encrypted with that voter's public key. Also,they publish $w^{x_{i,j}}$ for each $x_{i,j}$.
- The scrutineers check that $\sum_i x_{i,j} = 0$ for all j by calculating $\prod_i w^{x_{i,j}} \overset{?}{=} 1$. Also, the scrutineers calculate every voter's verification key $w_i = w^{\sum_j x_{i,j}} = \prod_j w^{x_{i,j}}$.
- All voters V_i read their messages from the authorities and determine their own secret exponent $x_i = \sum_j x_{i,j}$.
- In order to vote, the voters select a public random seed r, for example by querying the random oracle. Next, each voter encrypts his vote using r^{x_i} for randomizer. This encryption is posted to ~~$\mathcal{F}_{\mathsf{BB}}$~~ $\mathcal{F}_{\mathsf{SBC}}^{\Phi,\Delta,\alpha}$ along with a proof that the ballot encrypts an allowable vote and that the correct secret exponent was used, and with a signature on the previous two objects.
- Upon receiving (sid, Broadcast, $\langle b_1, \dots, b_k \rangle$) from $\mathcal{F}_{\mathsf{SBC}}^{\Phi,\Delta,\alpha}$, voters combine all votes and calculate tally *res*.

Figure 10: The protocol Π_{STVS} with voters V. Variant of [?]: instead of posting to the bulletin board $\mathcal{F}_{\mathsf{BB}}$, ballots are posted via $\mathcal{F}_{\mathsf{SBC}}$, removing the need for the trusted control voter.

amount of time from the opening of the election. The functionality does not allow the adversary to read the honestly cast votes or to falsify them. The functionality releases the result of the election when it moves to the tally phase after $\Phi + \Delta$ time has elapsed from the opening of the election. The simulator has an advantage α, i.e., it can obtain the election result (on behalf of some corrupted party) when $\Phi + \Delta - \alpha$ time has elapsed since the opening of the election, (yet after the end of the casting period). The VS functionality is presented in detail in the full version.

The self-tallying VS (STVS) protocol. We deploy an SBC instead of the BB used in the original protocol of Preneel and Szepieniec [?] to ensure fairness, removing the need for the control dummy party. The protocol assumes a public-key generation mechanism formalised by an ideal functionality $\mathcal{F}_{\mathsf{SKG}}$ for the authorities public key and corresponding private key shares, and a voters' key generation functionality $\mathcal{F}_{\mathsf{PKG}}$ for eligibility. The UC-security of Π_{STVS} is similar to the original one [?].

Theorem 4. *Let Δ, Φ, α be non-negative integers such that $\Delta > \Phi > 0$ and $\Delta \geq \alpha$. The protocol Π_{STVS} in Figure ?? UC-realizes $\mathcal{F}_{\mathsf{VS}}^{\Phi,\Delta,\alpha}$ in the $(\mathcal{F}_{\mathsf{SBC}}^{\Phi,\Delta,\alpha}, \mathcal{F}_{\mathsf{RBC}}, \mathcal{F}_{\mathsf{PKG}}, \mathcal{F}_{\mathsf{SKG}}, \mathcal{G}_{\mathsf{clock}})$-hybrid model against an adaptive adversary corrupting $t < n$ parties.*

ACKNOWLEDGMENTS

Zacharias was supported by Input Output (https://iohk.io) through their funding of the Edinburgh Blockchain Technology Lab.