Universally Composable Simultaneous Broadcast

Alejandro Hevia*

Department of Computer Science, Universidad de Chile
Blanco Encalada 2120, Santiago, Chile
ahevia@dcc.uchile.cl

Abstract. Simultaneous Broadcast protocols allow different parties to broadcast values in parallel while guaranteeing mutual independence of the broadcast values. The problem of simultaneous broadcast was suggested by Chor et al. (FOCS 1985) who proposed a linear-round solution, and later improved by Chor and Rabin (PODC 1987) and Gennaro (IEEE Trans. on Parallel and Distributed Systems 2000). The most efficient solution, in terms of round complexity, is the one due to Gennaro, which is in the common random string model. This construction has constant round complexity but is not very practical, as it requires generic zero-knowledge proofs, non-interactive zero-knowledge proofs of knowledge, and commitment schemes. All the mentioned solutions were proven secure under security definitions with weak or no composition guarantees – only sequential composition for the initial construction by Chor et al.

In this work, we explore the problem of Simultaneous Broadcast under Universally Composable (UC) security (Canetti 2001). We give a definition of Simultaneous Broadcast in this framework, which is shown to imply all past definitions. We also show this notion can be achieved by a computationally efficient, constant-round construction (building on the verifiable secret sharing scheme of Cramer et al. at Eurocrypt 1999), which is secure under an honest majority. Our results rely on (and benefit from) capturing synchronous communication as a functionality within the UC model, as suggested by Canetti (IACR eprint 2005). Indeed, we show that this approach of modeling synchronous communication can lead to better understanding of where synchronicity is needed, and also simpler constructions and proofs.

1 Introduction

Broadcast channels allow one or more senders to efficiently transmit messages to be received by all parties connected to a (physical or virtual) communication network. As a communication primitive, broadcast is fundamental both in the design of network communication protocols, and in the area of secure multiparty computation. The main security property characterizing broadcast communication is consistency: the messages received by all players as a result of a broadcast

* Work done while the author was at the Computer Sc. Dept., U.C. San Diego, USA. Supported in part by NSF grant 0313241. Any opinions, findings, and conclusions or recommendations expressed in this material are those of the author(s) and do not necessarily reflect the views of the National Science Foundation.

transmission operation are guaranteed to be the same. The problem of achieving consistency when implementing broadcast on top of a point-to-point network (commonly known as Byzantine agreement) is central not only in cryptography, but also to the area of fault-tolerant distributed computation. It has received enormous attention (e.g., [25,18,11,7] among many others).

In secure multiparty computation, it is often desirable that the broadcast channel satisfies some additional properties, besides consistency. For example, in applications where multiple senders must broadcast messages at the same time[1] (e.g., when running in parallel many copies of a broadcast protocol with different senders), it is often important to enforce the *simultaneous* transmission of the messages. The goal is that no sender can decide its broadcast message based on the values broadcast by the other players. Achieving this property, also called *independence*, is not as straightforward as it may seem. In general, naive parallel execution of broadcast protocols does not suffice, nor the more sophisticated round efficient approaches presented in [4,27]. Indeed, a common conservative assumption in settings where (multisender) broadcast channels are provided is to assume *rushing* adversaries – adversaries that, at each round, may see the messages sent by the honest parties before sending out the messages for the corrupted parties for the same round [5]. Nonetheless, this independence property does play a fundamental role in the secure multiparty computation protocol of [12] as well as in many important applications (like contract bidding, coin flipping, and electronic voting schemes, as exemplified in [13,17,19]) where this type of broadcast enormously simplifies the design of protocols.

The concept of *simultaneous broadcast* was introduced by Chor et al. in [12], along with a simulated-based definition. In [12], Chor et al. presented protocols that securely implement simultaneous broadcast on top of a network which allows regular broadcast transmission operations, not necessarily satisfying the simultaneity property. For each simultaneous broadcast operation, their protocols require a number of rounds that is linear in the number of parties. Subsequent works [13,19] focused on reducing the round complexity, obtaining simultaneous broadcast protocols with logarithmic [13] or even constant [19] number of rounds (the latter result achieved in the common random string model.) Even as the round-efficiency of the solutions increased, the definitions of security did not remain the same, and they actually became increasingly restricted, as it was pointed out by Hevia and Micciancio [22]. In particular, there it was shown that the protocol of [19], the most round efficient protocol so far, is secure under a definition of security strictly weaker than the original simulation-based definition [12].[2] Nonetheless, the round efficiency of Gennaro's protocol made attractive

[1] Also called *interactive consistency* in [31,4] and *parallel broadcast* in [22]. To avoid confusions, we adopt the term *multisender broadcast* to refer to the operation of multiple senders broadcasting messages at the same time.

[2] Indeed, the definition of simultaneous broadcast proposed in [19] may not exclude protocols that fail to achieve the intuitive notion of independence captured by the simulation-based definition of [12].

the search for either a proof that such protocol achieves a stronger notion of simultaneous broadcast (e.g. [12]), or for a variant of that protocol that does it.

Concurrent Execution and Universal Composability: The development of increasingly complex computing environments brought the concern that previously secure protocols (proven as stand-alone primitives) might not remain secure under stronger adversarial conditions, like parallel or concurrent execution with many (possibly different) other protocols [17], or if invoked by other (possibly unknown) protocols. It was in such context that several security frameworks were developed (see [32,8] for a good survey). In [8], Canetti presented the *Universally Composable (UC) Security* framework, which allows modular description and analysis of protocols under concurrent execution and provides strong composability guarantees. Indeed, the security of UC secure protocols is maintained under general composition with an unbounded number of instances of arbitrary protocols running concurrently. Thus, given the benefits of the UC framework, the security of many cryptographic primitives has been revisited to explore whether these stronger UC guarantees can be achieved, and if so, at what cost (in terms of efficiency or assumptions). As we will see in the next section, simultaneous broadcast can be achieved under UC security not only at no cost, but also with *gains* in terms of efficiency.

1.1 Our Contributions

In this work, we present an communication and round efficient solution for the simultaneous broadcast problem. Our solution is based on verifiable secret sharing (VSS) [12] and does not uses zero-knowledge proofs, zero-knowledge proofs of knowledge, not commitments schemes as previous constructions [13,19]. Moreover, our construction is provably secure in the Universally Composable framework against computationally-unbounded adaptive adversaries assuming an honest majority and private channels, with some negligible error probability.[3] To achieve this, we introduce a natural definition of Simultaneous Broadcast in the UC framework which implies all previous definitions. Our simultaneous broadcast construction is very efficient: we run one VSS per party in parallel. While the construction is technically simple, proving UC security present some subtleties, like dealing with rushing adversaries or parties simply "copying" someone else's sharing. We overcome some of the problems by defining a synchronous variant of verifiable secret sharing, which we call *Terminating VSS* (TVSS), and building our simultaneous broadcast protocol invoking such TVSS functionality. We then show that, when formalized as UC functionality, TVSS is intrinsically synchronous. A benefit of this approach is that our simultaneous broadcast protocol does not explicitly requires global synchronous communication since all the synchronicity is provided by the Terminating VSS functionality. Our construction and proof exemplifies the approach, first suggested by Canetti

[3] Most of the properties of our solution – namely communication and round complexity, reliability and negligible error probability – are inherited from the VSS used as building block [14].

in [8], of abstracting synchronous communication as a functionality rather than embedding it in the execution model [30,24]. We believe this approach leads to modular analysis, simple protocol design and simpler proofs.

1.2 Related Work

Simultaneous Broadcast: As mentioned above, the simultaneous broadcast problem was put forward by Chor et al. [12] who proposed a simulation-based definition and a linear-round protocol. This protocol essentially executed n sequential VSS protocols, where n is the number of communicating parties. The sequential execution was needed to prevent corrupted parties from broadcasting the same value as an honest party, for instance by reusing (copying) the VSS data sent out by the honest party. Then, Chor and Rabin in [13] showed how to reduce the round complexity to $\mathcal{O}(log(n))$ rounds. Their protocol requires, among other things, that each party first broadcast a commitment of her input and then proves knowledge of the broadcasted value. The reduction in rounds comes from using a clever scheduling technique for doing the proofs – for any two players, there is a step in the protocol where one player acts as prover and the other one acts as verifier of the proof of knowledge. Such a scheduling prevents "copying" the proofs. Finally, Gennaro in [19], working in the common random string model, put forward a protocol that greatly simplifies the one in [13] by showing how to run the proofs of knowledge in parallel, essentially employing non-interactive proofs of knowledge [34].

In terms of the previous definitional work for simultaneous broadcast problem, it turns out that each result in [12,13,19] presents a different definition. Hevia and Micciancio [22] show that these definitions form a strict hierarchy when considered in terms of input distributions. They point out that the strongest definition (in a well-defined sense, see [22]) is the simulation-based notion of [12], which preserves security under sequential composition. The notions in [13,19] targeted stand-alone execution and thus provide no composition guarantees.

Verifiable Secret Sharing: The notion of Verifiable Secret Sharing (VSS) was first proposed by Chor et al. [12] inspired by the need of adding robustness to standard secret sharing (eg. Shamir's [35]) The problem has been extensively studied both in the synchronous setting (e.g. [18,5,20,33,14,1]) and in the asynchronous setting [3,6,11,7]. In the information-theoretic model with adaptive adversaries, Rabin and Ben-Or [33] proposed a VSS secure under an honest majority, by allowing negligible failure probability. Subsequently, in the same model, Cramer et al. [14] improved the information checking protocols of [33] and presented a very efficient constant-round VSS protocol. By instantiating the Terminating VSS required in our construction with the scheme in [14] we obtain our constant-round solution.

Related Protocols and Generic Solutions: The simultaneous broadcast problem is related to the idea of *common-coin* protocols [18,29], where several parties want to generate one or more unbiased coins in a distributed way. Indeed, the constructions proposed in [18,29] involve executing VSS protocols in parallel,

in a similar approach to ours. We remark, however, that the goals are different: while for common-coin generation it suffices that the broadcast value of a single (uncorrupted) value is not correlated to the output of corrupted parties,[4] in the simultaneous broadcast problem we seek to guarantee that every single component of the output vector of the broadcast values remains "uninfluenced" by the other values in the same vector. In addition, our construction must guarantee security under general (UC) composition.

A related, although orthogonal issue, is the problem of *simultaneous termination*, which has been studied by Lindell et al. [27]. The problem arises in the context of parallel composition of multiparty protocols in synchronous networks, where certain protocols (including broadcast protocols) may not terminate in the same round when composed in parallel, thus complicating their sequential composition with other protocols. (We note that, in [27], the term *simultaneous broadcast* is used but with a different meaning than ours, as they refer to what we call multisender broadcast.) The methods in [27] do not attempt to achieve independence (in fact, they do not) since their concern is not adding new functional properties to the resulting parallel protocol, but ensuring that it can be safely composed sequentially with other protocols while preserving round efficiency. Also in the context of composition of broadcast protocols, Lindell et al. discussed some pitfalls in the composition of Authenticated Byzantine Agreement [26]. They show that, unless session identifiers are available, no parallel or concurrent composition of Authenticated Byzantine Agreement is secure if more than one third of the parties are faulty. In our work, we do assume the availability of broadcast channels (in the form of the broadcast functionality $\mathcal{F}_{BC}$) but we put no restrictions on how they are implemented. For example, session identifiers for the broadcast protocols could be initialized using the techniques of Barak et al. [2], or by standard techniques under setup assumptions [10].

Lastly, there are powerful UC constructions for secure multiparty computation of generic protocols (e.g. [10,15]) which could certainly be used to provide a solution to the simultaneous broadcast problem. Indeed, techniques of [10] do provide such a solution tolerating any number of corrupted parties in the common random string model. However, as done in many other examples of multiparty secure computation (eg. threshold signatures, key-exchange, voting), our goal is to look for more specialized, and therefore more efficient, solutions for the simultaneous broadcast problem.

Comparison with previous solutions: In terms of efficiency, the number of rounds required in our construction is equal to the number of rounds of the terminating VSS construction we use, which is the one proposed by Cramer et al. [14]. Similarly, the computational complexity of our construction is n times that of the terminating VSS in [14]. Concretely, our solution requires $\mathcal{O}((k + \log n)n^4)$ bits of communication, and only 14 rounds (or 12 if no faults occur). If the model is extended so parties can use digital signatures, the protocol takes only 7 rounds, although security holds only against computationally-bounded

[4] In all fairness, in general the mentioned protocols do achieve more than that.

adversaries. In comparison, the previous constant round solution [19] uses at comparable number of rounds (seven for the VSS protocol [5], plus six for the computational zero-knowledge proof [20], plus three rounds) but requires the communication of a large number of bits (n copies of a non-interactive zero-knowledge proof of knowledge for a generic NP statement [34], plus n times the communication complexity of the VSS protocol of [20], a total that in most implementations is often orders of magnitude larger than $\mathcal{O}((k + \log n)n^4)$).

In terms of the adversary tolerated our solution is similar to Gennaro's. The construction of [19] works over public channels under computationally-bounded static adversaries and can be made secure under adaptive adversaries using the compiler of [9]. In comparison, assuming secure channels, our solution tolerates computationally unbounded adaptive adversaries, and security in the public channels (and computationally-bounded adversaries) setting can be obtained by standard techniques, like non-committing encryption [9,16]. In terms of resilience, our construction tolerates at most $t < n/2$ corrupted parties as Gennaro's solution. The constructions of Chor et al. [12] and Chor and Rabin [13] tolerate $t < n/4$ and $t < n/2$ respectively.

Organization of the paper: In the next section, we briefly describe the UC framework. Then, in Sect. 3, we describe and justify our formalization of the notion of Terminating VSS (denoted UC-TVSS), our synchronous variant of VSS, and we mention how it can be efficiently implemented. Sect. 4.1 presents our notion of security for simultaneous broadcast (denoted UC-SB), and shows how to implement it from UC-TVSS. We conclude in Sect. 5 by discussing how to extend our results to the public channel model, and how to model simultaneous broadcast under purely asynchronous communication.

2 Preliminaries

Model: Our results are in the Universally Composable framework of Canetti as described in [8]. We briefly and informally outline it here. In the UC framework, the desired properties of cryptographic protocols are defined in terms of tasks or *functionalities*. A functionality is a "trusted third party" that first obtains inputs directly from the parties, performs certain instructions on these inputs, and provides the parties with the appropriate outputs. A protocol securely implements a given cryptographic task, if executing the protocol against a realistic (i.e. real-life) adversary "emulates" the execution of an *ideal process*. In the ideal process, the task is computed by the functionality directly interacting with the parties against a very limited adversary (called the *ideal-adversary*). The notion of "emulation" involves a distinguisher $\mathcal{Z}$ which, by providing the parties with inputs and seeing their outputs, and by interacting with the adversary, attempts to tell whether it is interacting with a real protocol and the real-life adversary, or with the functionality and the ideal-adversary. Good emulation means no such environment is successful. See details and proofs in [8].

In this paper, we consider a network of n parties, $P_1, \ldots, P_n$, connected by perfectly private authenticated channels and a broadcast channel. In the UC

terminology, this translates to working in the $(\mathcal{F}_{SMT}, \mathcal{F}_{BC})$-hybrid model, where $\mathcal{F}_{SMT}$ is the *secure message transmission* functionality [8] and $\mathcal{F}_{BC}$ is the *broadcast channel* functionality, which does not satisfy any "fairness" property (i.e. it allows *rushing*). There is a computationally unbounded adversary that can actively corrupt up to $t < n/2$ parties. We consider adaptive corruptions, where the decision to corrupt a party is made during the execution of the protocol, depending on the data gathered so far. Our protocols allow an error probability negligible in the security parameter k. In terms of notation, we let $[n]$ denote the set $\{1, \ldots, n\}$ and $P_{[n]}$ denote the set of all parties $\{P_1, \ldots, P_n\}$.

3 Terminating VSS

Motivation: One inconvenience of the definition of standard VSS schemes (as in [18] or even the UC variant of [1,8]) for our purposes is that it does not guarantee the protocol terminates if the dealer is corrupted. Nonetheless, all synchronous VSS protocols in the literature (e.g. [5,20,33,18,14,1]) seem to satisfy some form of *terminating* condition: there is a round in the execution in which all parties agree that the sharing phase has "time-out" and the secret is fixed. To capture this property while preserving the possibility that the VSS protocol being used *from* higher-level asynchronous protocols, we define the notion of *Terminating VSS* (TVSS).

TVSS protocols are guaranteed to conclude independently of the dealer's actions. We characterize TVSS as a functionality in the UC framework, $\mathcal{F}_{TVSS}$, which is shown in Fig. 1. Intuitively, $\mathcal{F}_{TVSS}$ extends the VSS functionality so that even if a corrupted dealer D fails to call the functionality, a fixed value is eventually associated to D. In fact, honest parties can "force" the functionality to fix a value for D by sending `EndSharing` messages. Our formulation of $\mathcal{F}_{TVSS}$, is inspired on the UC VSS variant of Abe and Fehr [1], which includes the concept of "spooling" the secret, a syntactic technique that allows the dealer to announce to the adversary – via a `Spool` message – that a new functionality is being called. The adversary is thus able to adaptively corrupt the dealer before the dealer commits to a value.[5] We say a protocol π achieves UC-TVSS if π UC-realizes functionality $\mathcal{F}_{TVSS}$.

The name *Terminating* VSS reflects that a protocol that UC-realizes $\mathcal{F}_{TVSS}$ concludes (terminates) as long as the adversary delivers all sent messages. The adversary can still delay or block some messages forever – but nothing more. In particular, the protocol does not stall even if the corrupted dealer is irresponsive. This adversarial behavior is a concern in protocols that depend on the termination of a VSS subprotocol, even in the authenticated transmission model, as the parties waiting for a successful completion of the VSS functionality may not

[5] A similar technique appears in the formalization of VSS in [8], albeit implicitly in the way the functionality reacts to the corruption messages from the adversary. Another purely syntactic choice is allowing the adversary to trigger the end of the sharing phase – alternative but equivalent formulations are possible (see [21]).

Functionality $\mathcal{F}_{TVSS}$

$\mathcal{F}_{TVSS}$ expects its SID to be of the form $sid = (sid', D, \mathcal{P})$, where $\mathcal{P}$ is a list of parties among which sharing is to be performed. It proceeds as follows.

(1) At first activation, initialize $\mathbf{s}$ as $\perp$.

(2) Upon receiving input $(\texttt{Spool}, sid, s)$ from party $D \in \mathcal{P}$, set $\mathbf{s} \leftarrow s$ and send $(\texttt{SpoolRcvd}, sid, D)$ to adversary A. (Any subsequent input `Spool` is ignored.)

(3) Upon receiving input $(\texttt{Share}, sid, s')$ from party $D \in \mathcal{P}$, set $\mathbf{s} \leftarrow s'$ and send $(\texttt{ShareRcvd}, sid, D)$ to adversary A. (Any subsequent input `Share` is ignored.)

(4) Upon receiving input $(\texttt{EndSharing}, sid)$ from uncorrupted party $P \in \mathcal{P}$, record $(\texttt{EndSharing}, P)$ and send $(\texttt{EndSharingRcvd}, sid, P)$ to adversary A.

(5) Upon receiving message $(\texttt{Corrupt}, sid, P)$ from the adversary, do: If $P = D$ then send $\mathbf{s}$ to the adversary. Otherwise, delete record $(\texttt{EndSharing}, P)$, if exists. In both cases, send $(\texttt{Corrupt}, sid)$ message to P.

(6) Upon receiving message $(\texttt{DoEndSharing}, sid)$ from the adversary do: If there is at least one record of the form $(\texttt{EndSharing}, sid, P)$, and
- D is honest and a message $(\texttt{Share}, sid, u)$ from D has been received, or
- D is corrupted,

then send $(\texttt{Shared}, sid)$ to each party in $\mathcal{P}$ and A. (Any subsequent input `Share` or message `DoEndSharing` is ignored.) Otherwise, ignore the message.

(7) Upon receiving input $(\texttt{Open}, sid)$ from uncorrupted party P, output $(\texttt{Opened}, sid, \mathbf{s})$ to P and the adversary A.

Fig. 1. The Terminating VSS (with Spooling) functionality, $\mathcal{F}_{TVSS}$

have access to synchronous communication or any "time-out" mechanism.[6] In this context, *termination* means that, once honest parties are instructed to start executing the TVSS protocol, then no matter the actions of the dealer, π concludes with some output (possibly empty) if the adversary delivers all messages. Discussion: One may argue that termination issue seems to disappear if one considers a synchronous version of the UC model (as done in [30,24]). While synchronous communication among all parties certainly allows to implements time-outs (and thus have default sharings if the dealer does not participate), we believe that "encapsulating" synchronicity *inside* the primitive that requires it is useful as higher-level protocols do not need to be aware of it. Concretely, TVSS not only captures a form of synchronous VSS but also keeps the dependence on synchronous communication modularized, as any reliance on it is explicitly and independently handled inside the TVSS functionality. In particular, even though it is possible to show that any implementation of TVSS requires synchronous communication (at least twice) among the parties running it,[7] a higher-level protocol ρ using the TVSS functionality can run in an asynchronous way. In practice, this means that ρ could be implemented in an asynchronous network

[6] We remark that, in some applications, the parties "waiting" for the completion of a VSS subprotocol may not be the same as the ones executing the VSS protocol.

[7] This is implied by Claim 4.1 where TVSS is shown to be equivalent to functionality $\mathcal{F}_{SYN}$, synchronous communication with guaranteed delivery [8].

where only limited synchronicity is available (say only within certain subsets of the parties, or when synchronous communication can only be provided very infrequently) as long as the subprotocol that realizes the TVSS functionality has "enough" access to the synchronization capability. For example, applications in cluster networks [36] may exploit the advantage of implementing TVSS locally in each cluster (where synchronization is easier) while the inter-cluster protocol ρ can run asynchronously. Even our concrete application of TVSS, building simultaneous broadcast, where each TVSS involves *all* the parties, may benefit from this modular approach: dealers in different TVSS subprotocols could start the execution at different rounds (because of lack of network connectivity, for example) and still be able to achieve simultaneous broadcast. Furthermore, the simplicity of our construction for simultaneous broadcast shows that this approach may also simplify protocol design and security proofs.

3.1 Instantiating TVSS

In this section, we revisit the very efficient VSS protocol presented by Cramer et al. in [14] in a synchronous model of computation with some negligible error probability. The scheme is based on the bivariate solution of Feldman [18,5] and builds on the information checking techniques of Rabin and Ben-Or [33]. The construction is very efficient: the sharing phase takes fourteen rounds and reconstruction takes two rounds, while the total communication cost is $\mathcal{O}((k + \log n)n^3)$ bits for an error probability of $2^{-k+\mathcal{O}(\log n)}$. This construction π_{TVSS} is detailed in [14,21].

In [14], Cramer et al. prove their construction information-theoretically secure against adaptive corruptions under the classical definition of security [18]. The next proposition shows that their protocol can be proven a secure Terminating VSS in the UC hybrid model we consider here if the model includes the synchronous communication (with guaranteed delivery) functionality $\mathcal{F}_{SYN}$ proposed in [8]. Due to space constraints, the proof is omitted (see [21]).

Proposition 1. *Protocol π_{TVSS} UC-securely realizes functionality $\mathcal{F}_{TVSS}$ in the $(\mathcal{F}_{BC}, \mathcal{F}_{SMT}, \mathcal{F}_{SYN})$-hybrid model for $n > 2t$.*

4 UC Simultaneous Broadcast (UC-SB)

In this section, we generalize the simulation-based definition of Simultaneous Broadcast put forward by Chor et al. [12] to the UC framework. We achieve this by providing a *simultaneous broadcast* functionality $\mathcal{F}_{SB}$ (Fig. 2) which is a variant of the synchronous communication functionality [8] that provides "fairness", in the sense that the adversary is not allowed rushing. We say a protocol π achieves UC-SB if π UC-securely implements functionality $\mathcal{F}_{SB}$.

Intuitively, the definition of $\mathcal{F}_{SB}$ guarantees independence as the adversary cannot access any honest party's input until the broadcast is authorized to proceed, when it is "too late". Notice also that the functionality guarantees output delivery. In some applications, it may be useful to relax this condition.

Functionality $\mathcal{F}_{SB}$

$\mathcal{F}_{SB}$ expects its SID to be of the form $sid = (sid', \mathcal{P})$, where $\mathcal{P}$ is a list of parties among which broadcast is to be performed. It proceeds as follows.

(1) Upon receiving input $(\texttt{Broadcast}, sid, m)$ from party $P \in \mathcal{P}$, record (P, m) and output (sid, P) to the adversary. (If P later becomes corrupted then the record (P, m) is deleted.)

(2) Upon receiving message $(\texttt{Proceed}, sid, N)$ from the adversary, do: If there exist uncorrupted parties $P \in \mathcal{P}$ for which no record (P, m) exists then ignore the message. Else:
   1. Interpret N as the list of messages sent by corrupted parties. That is, $N = \{(S_i, m_i)\}$ where each $S_i \in \mathcal{P}$ is corrupted, and m_i is a message.
   2. Prepare a vector $\mathbf{m} = (m_i)_{i \in \mathcal{P}}$ of messages sent by all parties in $\mathcal{P}$, both corrupted and honest.

(3) Send $(\texttt{Broadcast}, sid, \mathbf{m})$ to the adversary.

(4) Upon receiving input $(\texttt{Receive}, sid)$ from a party $P \in \mathcal{P}$, output $(\texttt{Received}, sid, \mathbf{m})$ to P.

Fig. 2. The simultaneous broadcast functionality, $\mathcal{F}_{SB}$

UC-SB AND PREVIOUS SIMULTANEOUS BROADCAST DEFINITIONS: It is not hard to see that UC-SB implies the (stand-alone) simulation-based definition of simultaneous broadcast in [12]. This is immediate since UC security implies stand-alone security [8]. Then, by the results of [22], it holds that UC-SB implies all the other notions of Simultaneous Broadcast [13,19].

4.1 A Generic Construction of UC-SB from UC-TVSS

In this section, we present our main construction. We show how to implement simultaneous broadcast (UC-SB) using Terminating VSS (UC-TVSS). The construction is simple: each party first runs the share phase of the TVSS in parallel; once all sharings have concluded (terminated), each party starts the reconstruction phase, gather all other parties' secrets and output the vector of values. (see Fig. 3). Moreover, the construction works for any t; the final condition of honest majority comes from instantiating $\mathcal{F}_{TVSS}$ with π_{TVSS} (Prop. 1).

Theorem 1. *Protocol π_{SB} UC-securely realizes $\mathcal{F}_{SB}$ in the $\mathcal{F}_{TVSS}$-hybrid model.*

Proof. Let A be a real-life adversary for π_{SB}. Note that A expects to interact with n parties running π_{SB} with access to n copies of functionality $\mathcal{F}_{TVSS}$. Given A, the ideal adversary S simulates the execution of protocol π_{SB} for adversary A by simulating the parties and functionalities as follows. Let $P_1, \ldots, P_n$ denote the simulated parties, $\tilde{P}_1, \ldots, \tilde{P}_n$ the ideal-world parties. Let sid^* be the session identifier under which each (simulated) party is first invoked (by the environment $\mathcal{Z}$), and $\mathcal{F}^1_{TVSS}, \ldots, \mathcal{F}^n_{TVSS}$ be the (simulated) n copies of functionality $\mathcal{F}_{TVSS}$, where $\mathcal{F}^k_{TVSS}$ denotes the functionality invoked by P_k with session identifier

Protocol π_{SB}

Private Inputs: Each P_i holds $x_i \in X$
Public Input: session identifier sid^*
Private Outputs: a vector $\mathbf{y}_i \in (X \cup \{\perp\})^n$ for each P_i
Each party $P_i \in \mathcal{P}$ runs sequentially the following steps:

(1) For each $j \in P_{[n]}$, set $sid_j \leftarrow (sid^*, P_j, P_{[n]})$.
(2) Send $(\texttt{Spool}, sid_i, x_i)$ to $\mathcal{F}_{TVSS}$.
(3) Send $(\texttt{Share}, sid_i, x_i)$ to $\mathcal{F}_{TVSS}$.
(4) For each $j \in P_{[n]}$, send $(\texttt{EndSharing}, sid_j)$ to $\mathcal{F}_{TVSS}$.
(5) Upon receiving $(\texttt{Shared}, sid_j)$ from $\mathcal{F}_{TVSS}$, record $(\texttt{Shared}, sid_j)$. Repeat this step until there is a record $(\texttt{Shared}, sid_j)$ for each $j \in [n]$
(6) For each $j \in P_{[n]}$, send $(\texttt{Open}, sid_j)$ to $\mathcal{F}_{TVSS}$.
(7) Upon receiving $(\texttt{Opened}, sid_j, v_j)$ from $\mathcal{F}_{TVSS}$, record (sid_j, y_j). Once a record (sid_j, y_j) for each $j \in [n]$ exists, output vector $\mathbf{y}_i = (y_j)_{j \in [n]}$ and halt.

Fig. 3. Simultaneous Broadcast protocol in the $\mathcal{F}_{TVSS}$-hybrid model

$sid_k = (sid^*, P_k, P_{[n]})$.[8] Adversary S maintains a set N with the corrupted parties and their inputs, initially $N \leftarrow \emptyset$, and proceeds as follows. If A corrupts any party P_i before the party has submitted a `Share` message to $\mathcal{F}^i_{TVSS}$ then S corrupts ideal-world party $\tilde{P}_i$, obtains its input x_i, and pass it to A. If A instructs corrupted party P_i to submit $(\texttt{Share}, x'_i)$ to $\mathcal{F}^i_{TVSS}$, then S simulates the operation, and adds $(\tilde{P}_i, x'_i)$ to N. For all uncorrupted parties P_k, S sets P_k's input to an arbitrary value (eg. $x'_k \leftarrow \perp$) and simulates P_k's interaction with $\mathcal{F}^k_{TVSS}$ by simulating both, party and functionality. Notice that S can do such simulation without the real P_k's input because adversary A's view of the interaction between P_k and $\mathcal{F}^k_{TVSS}$ during the *share* phase of TVSS (steps (1)-(6) of Fig. 1) is independent of P_k's input. Indeed, consider the event E_k defined as "$\mathcal{F}^k_{TVSS}$ has at least one record $(\texttt{EndSharing}, P)$ *and then* it receives a message `DoEndSharing` from A". As long as P_k is corrupted anytime before E_k is true, S can proceed as before, that is, S obtains x_k from corrupting $\tilde{P}_k$ and pass it to A. Notice, however, that adversary A must corrupt a party P_i before P_i sends out message `Share` to $\mathcal{F}^i_{TVSS}$ if A wants to change the value submitted by P_i.

At some point in the simulation, A may send a `DoEndSharing` message to some TVSS functionality. Then, S partitions the simulated parties in four sets. These sets are dynamic in the sense that S may *move* parties from one set to another depending on the subsequent instructions of A. The corrupted parties are partitioned into B_{Sh} and its complement, where B_{Sh} is the set of parties which have submitted a message `Share` to $\mathcal{F}_{TVSS}$. (Notice that for all $P_i \in B_{Sh}$, N contains an entry $(\tilde{P}_i, x'_i)$.) Similarly, any honest party P_i is either in G_{Sh} or its complement, where G_{Sh} is the set of parties that have submitted a message `Shared` to its $\mathcal{F}^i_{TVSS}$. Notice that if $P_i \in G_{Sh}$, then P_i has sent

[8] Notice that, such functionality may also be invoked (and instantiated) by some other party P_j on message `EndSharing` if P_k is not activated by $\mathcal{Z}$.

or is about to send a `EndSharing` message. Let $B_{\texttt{end}}$ (resp. $G_{\texttt{end}}$) be the set of corrupted (resp. uncorrupted) parties whose corresponding TVSS functionalities have at least one record of the form $(\texttt{EndSharing}, P_j)$. Assume A sends a message `DoEndSharing` to functionality $\mathcal{F}_{TVSS}^k$. Then,

(1) If $P_k \notin B_{\texttt{end}} \cup G_{\texttt{end}}$, that is, $\mathcal{F}_{TVSS}^k$ has no `EndSharing` records, then S does nothing (since those messages are ignored by the TVSS functionality).

(2) If $P_k \in G_{\texttt{end}}$ but $P_k \notin G_{Sh}$, that is, $\mathcal{F}_{TVSS}^k$ has one or more `EndSharing` records but P_k has yet to submit a `Share` request to $\mathcal{F}_{TVSS}^k$, then S does nothing (since those messages are ignored by the TVSS functionality).

(3) If $P_k \in B_{\texttt{end}} \cap B_{Sh}$ or if $P_k \in G_{\texttt{end}} \cap G_{Sh}$, then S simulates $\mathcal{F}_{TVSS}^k$'s execution by having the functionality send messages $(\texttt{Shared}, sid_k)$ to all parties P_i and the adversary A. If $P_k \in B_{\texttt{end}} \cap \overline{B_{Sh}}$, then S does the same but also adds $(P_k, \perp)$ into set N.

(4) If A instructs a corrupted party P_i to send a message `Open` to some $\mathcal{F}_{TVSS}^k$, then S honestly simulates the functionality.

We also let $J \subseteq (G_{\texttt{end}} \cap G_{Sh}) \cup B_{\texttt{end}}$ be the set of parties to whose functionality A has sent a message `DoEndSharing`. S continues the simulation following the above rules (possibly moving parties into G_{Sh}, B_{Sh}, $G_{\texttt{end}}$, $B_{\texttt{end}}$, and J as new messages are delivered by A) until $J = [n]$. Assume this happens when A sends a message `DoEndSharing` to $\mathcal{F}_{TVSS}^k$. *Before* applying rule 3 from above, S sends $(\texttt{Proceed}, sid^*, N)$ to ideal functionality $\mathcal{F}_{SB}$, and obtains $(\texttt{Broadcast}, sid^*, \mathbf{m})$. S uses $\mathbf{m}$ to set the secret in each simulated (uncorrupted) $\mathcal{F}_{TVSS}^i$ to $s_i = m_i$, where $\mathbf{m} = (m_1, \ldots, m_n)$. Only then S applies rule 3 from above for party P_k. ¿From then on, S honestly simulates the execution of π_{SB} for A.

We claim that the simulation is perfect. Indeed, observe that adversary A's view before set J becomes equal to $[n]$ is independent of the input of the simulated parties, as it consists of the corrupted parties' inputs, and messages $(\texttt{SpoolRcvd}, sid_i, P_i)$, $(\texttt{ShareRcvd}, sid_i, P_i)$, $(\texttt{EndSharingRcvd}, sid_i, P_i)$, and $(\texttt{Shared}, sid_j, P_i)$ for one or more party $P_i \in (G_{\texttt{end}} \cap G_{Sh}) \cup B_{\texttt{end}}$. The crucial observation is that no uncorrupted party P_k issues an `Open` message unless P_k has received `Shared` messages for all parties. This only happens if `DoEndSharing` messages have been received by each functionality $\mathcal{F}_{TVSS}^j$, $P_j \in J$, which only happens *after* J is set to $[n]$. At that point, the ideal adversary S has obtained the inputs for all parties, so the adversary's view from then on is identical to the real-world experiment. Notice also that once adversary A sends `DoEndSharing` messages to each functionality $\mathcal{F}_{TVSS}^j$, $P_j \in J = [n]$, A cannot issue a `Share` message for any (corrupted) party P_i. This is because P_i must also be in $J \subseteq (G_{\texttt{end}} \cap G_{Sh}) \cup B_{\texttt{end}}$ which implies P_i is either in $G_{\texttt{end}} \cup B_{\texttt{end}}$, and therefore functionality $\mathcal{F}_{TVSS}^i$ has successfully executed step (6) where $(\texttt{Shared}, sid_i)$ was sent out to all parties; after this step, no new `Share` or`DoEndSharing`input is accepted by $\mathcal{F}_{TVSS}^i$. This concludes the proof.

On the Synchronicity of Simultaneous Broadcast and TVSS: We conclude this section showing that Simultaneous Broadcast is essentially as strong as synchronous communication, namely $\mathcal{F}_{SYN}$. One direction is provided by the

reduction to UC-TVSS described above, which says that any solution for UC-TVSS can be used to achieve UC-SB. Notice also that $\mathcal{F}_{SYN}$ implies UC-TVSS by Prop. 1. The other direction holds because UC-SB can be used to implement $\mathcal{F}_{SYN}$ as follows: first parties (non-simultaneously) broadcast their values, and then use simultaneous broadcast to transmit the same values (i.e. those broadcasted non-simultaneously before). Thus, the following claim holds.

Claim. Let π be a protocol that UC-securely realizes $\mathcal{F}_{SB}$ in the $\mathcal{F}_{SMT}$-hybrid model. Then, there exists a protocol that UC-securely realizes $\mathcal{F}_{SYN}$ in the $(\mathcal{F}_{SB}, \mathcal{F}_{SMT})$-hybrid model.

5 Extensions

Removing secure channels: Our protocol for simultaneous broadcast is only secure in the secure channel model. To obtain a protocol secure in the public channel model (i.e. authenticated channels), we can use known techniques, like those proposed by Lysyanskaya [28] which require secure erasures, or non-committing encryption [9,16]. For the case of static corruption is much simpler, as encrypting the messages with a semantic secure encryption scheme suffices.

Functionality $\mathcal{F}_{ASB}$

$\mathcal{F}_{SB}$ expects its SID to be of the form $sid = (sid', \mathcal{P}, t)$, where $\mathcal{P}$ is a list of parties among which broadcast may potentially be performed, and $t < n$ is an integer, where $n \stackrel{\text{def}}{=} |\mathcal{P}|$. It proceeds as follows.

(1) Upon receiving input $(\texttt{Broadcast}, sid, m)$ from party $P \in \mathcal{P}$, record (P, m) and output (sid, P) to the adversary. (If P later becomes corrupted then the record (P, m) is deleted.)

(2) Upon receiving a message $(\texttt{Proceed}, sid, N, W)$ from the adversary, do: If W is a subset of parties in $\mathcal{P}$ of size at least $n - t$, and there exist honest parties $P \in W$ for which no record (P, m) exists then ignore the message. Else:
   1. Interpret N as the list of messages sent by corrupted parties. That is, $N = \{(S_i, m_i)\}$ where $S_i \in W$ and S_i is corrupted, and m_i is a message.
   2. Prepare a vector $\mathbf{m} = (m_i)_{i \in W}$ of messages sent by all parties in W, both corrupted and honest.

(3) Send $(\texttt{Broadcast}, sid, \mathbf{m})$ to the adversary.

(4) Upon receiving input $(\texttt{Receive}, sid)$ from a party $P \in \mathcal{P}$, send $(\texttt{Received}, sid, \mathbf{m})$ as delayed output to P.

Fig. 4. The asynchronous simultaneous broadcast functionality, $\mathcal{F}_{ASB}$

Asynchronous Simultaneous Broadcast (UC-ASB): It is well known that in an asynchronous network, no functionality that depends on all the inputs can be computed [31]. This is because it is impossible to distinguish between failed processes (those instructed to not send messages) and very slow processes. Therefore, no process can afford to wait for messages coming from more than $n - t$

distinct other processes. In this section, we adapt the functionality of Simultaneous Broadcast to comply with this restriction, at the cost of weakening the guarantee that all players can participate in the broadcast (which is unavoidable). We remark that, nonetheless, the modified functionality $\mathcal{F}_{ASB}$ (Fig. 4) still preserves the intuitive notion of *independence*, as long as parties that do not participate in the broadcast are not allowed to contribute later with their inputs. We say a protocol π achieves UC-ASB if π UC-securely realizes $\mathcal{F}_{ASB}$.

We claim (without proof) that there exists a simple construction that achieves UC-ASB for the case $n > 3t$. It suffices to run first the initial phase of the secure multiparty computation of Ben-Or et al. [6]. Spelled out, first, parties run n parallel copies of the *ultimate secret sharing* protocol; then the protocol for *agreement on a common subset* is run. (Both protocols are described in [6].) In this way, all parties agree on the set W of parties that have properly shared their input. The reconstruction protocol is executed next, where the secrets of all parties in W is reconstructed. For computationally bounded adversaries, a similar approach can be obtained using the initialization phase of the computationally efficient construction of [23]. It is an open problem whether more communication efficient solutions exist.

References

1. M. Abe and S. Fehr. Adaptively secure feldman VSS and applications to universally-composable threshold cryptography. In *Advances in Cryptology – CRYPTO*, LNCS 3152, pages 317–334. Springer-Verlag, 2004.
2. B. Barak, Y. Lindell, and T. Rabin. Protocol initialization for the framework of universal composability. Cryptology ePrint Archive, Report 2004/006, 2004. `http://eprint.iacr.org/`.
3. M. Ben-Or, R. Canetti, and O. Goldreich. Asynchronous secure computation. In *ACM STOC'93*, pages 52–61. ACM Press, 1993.
4. M. Ben-Or and R. El-Yaniv. Resilient-optimal interactive consistency in constant time. *Distributed Computing*, 16(4):249–262, 2003.
5. M. Ben-Or, S. Goldwasser, and A. Wigderson. Completeness theorems for non-cryptographic fault-tolerant distributed computations. In *ACM STOC'88*, pages 1–10. ACM Press, 1988.
6. M. Ben-Or, B. Kelmer, and T. Rabin. Asynchronous secure computations with optimal resilience (extended abstract). In *ACM PODC'94*, pages 183–192, 1994.
7. C. Cachin, K. Kursawe, F. Petzold, and V. Shoup. Secure and efficient asynchronous broadcast protocols. In *Advances in Cryptology – CRYPTO*, LNCS 2139, pages 524–541. Springer-Verlag, 2001.
8. R. Canetti. Universally composable security: A new paradigm for cryptographic protocols. Report 2000/067, Cryptology ePrint Archive, January 2005. Full version of that in IEEE Symposium on Foundations of Computer Science (FOCS'01).
9. R. Canetti, U. Feige, O. Goldreich, and M. Naor. Adaptively secure multi-party computation. In *ACM STOC'96*. ACM Press, 1996.
10. R. Canetti, Y. Lindell, R. Ostrovsky, and A. Sahai. Universally composable two-party and multi-party secure computation. In *ACM STOC'02*, 2002.
11. R. Canetti and T. Rabin. Fast asynchronous Byzantine agreement with optimal resilience (extended abstract). In *ACM STOC'93*, pages 42–51. ACM Press, 1993.

12. B. Chor, S. Goldwasser, S. Micali, and B. Awerbuch. Verifiable secret sharing and achieving simultaneity in the presence of faults. In *IEEE Symposium on Foundations of Computer Science (FOCS'85)*, pages 383–395. IEEE CS, 1985.
13. B. Chor and M. O. Rabin. Achieving independence in logarithmic number of rounds. In *ACM Symposium on Principles of Distributed Computing (PODC'87)*, pages 260–268. ACM Press, 1987.
14. R. Cramer, I. Damgård, S. Dziembowski, M. Hirt, and T. Rabin. Efficient multiparty computations secure against an adaptive adversary. In *Advances in Cryptology – EUROCRYPT'99*, pages 311–326. Springer-Verlag, 1999.
15. I. Damgård and J. B. Nielsen. Universally composable efficient multiparty computation from threshold homomorphic encryption. In *Advances in Cryptology – CRYPTO*, LNCS 2729, pages 247–264. Springer-Verlag, 2003.
16. I. Damgård and J.B. Nielsen. Improved non-committing encryption schemes based on a general complexity assumption. In *Advances in Cryptology – CRYPTO*, LNCS 1880, pages 432–450. Springer-Verlag, 2000.
17. D. Dolev, C. Dwork, and M. Naor. Nonmalleable cryptography. *SIAM Journal on Computing*, 30(2):391–437, April 2001.
18. P. Feldman and S. Micali. An optimal probabilistic protocol for synchronous byzantine agreement. *SIAM Journal on Computing*, 26(4):873–933, 1997.
19. R. Gennaro. A protocol to achieve independence in constant rounds. *IEEE Transactions on Parallel and Distributed Systems*, 11(7):636–647, July 2000.
20. O. Goldreich, S. Micali, and A. Wigderson. Proofs that yield nothing but their validity or all languages in NP have zero-knowledge proof systems. *Journal of the ACM*, 38(3):691–729, July 1991.
21. A. Hevia. Universally composable simultaneous broadcast. Available from `http://www.dcc.uchile.cl/~ahevia/pubs/`, 2006. Full version of this paper.
22. A. Hevia and D. Micciancio. Simultaneous broadcast revisited. In *ACM PODC'05*, pages 324–333. ACM Press, 2005.
23. M. Hirt, J. B. Nielsen, and B. Przydatek. Cryptographic asynchronous multiparty computation with optimal resilience (extended abstract). In *Advances in Cryptology - EUROCRYPT'05*, LNCS 3494, pages 322–340. Springer-Verlag, 2005.
24. D. Hofheinz and J. Muller-quade. A synchronous model for multi-party computation and the incompleteness of oblivious transfer. Available from http://eprint.iacr.org/2004/016, 2004.
25. L. Lamport, R. Shostak, and M. Pease. The Byzantine generals problem. *ACM Transactions on Programming Languages and Systems*, 4(3):382–401, July 1982.
26. Y. Lindell, A. Lysyanskaya, and T. Rabin. On the composition of authenticated byzantine agreement. In *ACM STOC'02*, pages 514–523. ACM Press, 2002.
27. Y. Lindell, A. Lysyanskaya, and T. Rabin. Sequential composition of protocols without simultaneous termination. In *ACM PODC'02*, pages 203–212, 2002.
28. A. Lysyanskaya. Threshold cryptography secure against the adaptive adversary, concurrently. Report 2000/019, Cryptology ePrint Archive, 2000.
29. S. Micali and T. Rabin. Collective coin tossing without assumptions nor broadcasting. In *Advances in Cryptology – CRYPTO*, LNCS 537, pages 253–266, 1990.
30. J. B. Nielsen. *On Protocol Security in the Cryptographic Model*. Ph.D. thesis, Aarhus University, 2003.
31. M. Pease, R. Shostak, and L. Lamport. Reaching agreements in the presence of faults. *Journal of the ACM*, 27(2):228–234, April 1980.
32. B. Pfitzmann and M. Waidner. A model for asynchronous reactive systems and its application to secure message transmission. In *IEEE Symposium on Security and Privacy (S&P-01)*, pages 184–201. IEEE CS, 2001.

33. T. Rabin and M. Ben-Or. Verifiable secret sharing and multiparty protocols with honest majority. In *ACM STOC'89*, pages 73–85. ACM Press, 1989.
34. A. De Santis and G. Persiano. Zero-knowledge proofs of knowledge without interaction (extended abstract). In *IEEE Symposium on Foundations of Computer Science (FOCS'92)*, pages 427–436. IEEE CS, 1992.
35. A. Shamir. How to share a secret. *Communications of the ACM*, 22(11), 1979.
36. L. von Ahn, A. Bortz, and N.J. Hopper. k-Anonymous message transmission. In *ACM Conference on Computer and Communication Security – CCS'03*, pages 122–130. ACM Press, 2003.