Secure and Efficient Asynchronous Broadcast Protocols*

(Extended Abstract)

Christian Cachin, Klaus Kursawe, Frank Petzold**, and Victor Shoup

IBM Research, Zurich Research Laboratory
CH-8803 Rüschlikon, Switzerland
{cca,kku,sho}@zurich.ibm.com

Abstract. Broadcast protocols are a fundamental building block for implementing replication in fault-tolerant distributed systems. This paper addresses secure service replication in an asynchronous environment with a static set of servers, where a malicious adversary may corrupt up to a threshold of servers and controls the network. We develop a formal model using concepts from modern cryptography, give modular definitions for several broadcast problems, including reliable, atomic, and secure causal broadcast, and present protocols implementing them. Reliable broadcast is a basic primitive, also known as the Byzantine generals problem, providing agreement on a delivered message. Atomic broadcast imposes additionally a total order on all delivered messages. We present a randomized atomic broadcast protocol based on a new, efficient multi-valued asynchronous Byzantine agreement primitive with an external validity condition. Apparently, no such efficient asynchronous atomic broadcast protocol maintaining liveness and safety in the Byzantine model has appeared previously in the literature. Secure causal broadcast extends atomic broadcast by encryption to guarantee a causal order among the delivered messages. Our protocols use threshold cryptography for signatures, encryption, and coin-tossing.

1 Introduction

Broadcast protocols are a fundamental building block for fault-tolerant distributed systems. A group of servers can offer some service in a fault-tolerant way by using the state machine replication technique, which will mask the failure of any individual server or a fraction of them. In the model with Byzantine faults considered here, faulty servers may exhibit arbitrary behavior or even be controlled by an adversary.

* This work was supported by the European IST Project MAFTIA (IST-1999-11583). However, it represents the view of the authors. The MAFTIA project is partially funded by the European Commission and the Swiss Department for Education and Science.

** Frank Petzold has since left IBM and can be reached at petzold@hepe.com.

J. Kilian (Ed.): CRYPTO 2001, LNCS 2139, pp. 524–541, 2001.

In this paper, we present a modular approach for building robust broadcast protocols that provide reliability (all servers deliver the same messages), atomicity (a total order on the delivered messages), and secure causality (a notion that ensures no dishonest server sees a message before it is scheduled by the system). An important building block is a new protocol for multi-valued Byzantine agreement with "external validation." Our focus is on methods for distributing secure, trusted services on the Internet with the goal of increasing their availability and security. Cryptographic operations are exploited to a greater extent than previously for such protocols because we consider them to be relatively cheap, in particular compared to the message latency on the Internet.

We do not make any timing assumptions and work in a purely asynchronous model with a static set of servers and no probabilistic assumptions about message delays. Our protocols rely on a trusted dealer that is used once to set up the system, but they do not use any additional external constructs later (such as failure detectors or stability mechanisms). We view this as the standard cryptographic model for a distributed system with Byzantine faults. These choices maintain the safety of the service even if the network is temporarily disrupted. This model also avoids the problem of having to assume synchrony properties and to fix timeout values for a network that is controlled by an adversary; such choices are difficult to justify if safety and also security depend on them.

Despite the practical appeal of the asynchronous model, not much research has concentrated on developing efficient asynchronous protocols or implementing practical systems that need consensus or Byzantine agreement. Often, developers of distributed systems avoid the approach because of the result of Fischer, Lynch, and Paterson [9], which shows that consensus is not reachable by protocols that use an a priori bounded number of steps, even with crash failures only. But the implications of this result should not be overemphasized. In particular, there are randomized solutions that use only a *constant expected* number of asynchronous "rounds" to reach agreement [15,7,3]. Moreover, by employing modern, efficient cryptographic techniques and by resorting to the random oracle model, this approach has recently been extended to a practical yet provably secure protocol for cryptographic Byzantine agreement that withstands the maximal possible corruption [6].

Two basic broadcast protocols are *reliable broadcast* (following Bracha and Toueg [4]), which ensures that all servers deliver the same messages, and a variation of it that we call *consistent broadcast*, which only provides agreement among the actually delivered messages. The consistent broadcast primitive used here is particularly useful in connection with a *verifiability* property for the delivered messages, which ensures that a party can transfer a "proof of delivery" to another party in a single flow.

The efficient randomized agreement protocols mentioned before work only for binary decisions (or for decisions on values from small sets). In order to build distributed secure applications, this is not sufficient. One also needs agreement on values from large sets, in particular for ordering multiple messages. We propose a new *multi-valued Byzantine agreement* protocol with an *external* validity

condition and show how it can be used for implementing *atomic broadcast*. External validity ensures that the decision value is acceptable to the particular application that requests agreement; this corrects a drawback of earlier agreement protocols for multi-valued agreement, which could decide on illegal values. Both protocols use digital signatures and additional cryptographic techniques.

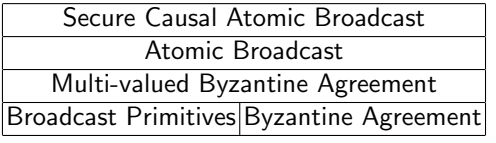
The multi-valued Byzantine agreement protocol invokes only a constant expected number of binary Byzantine agreement sub-protocols on average and achieves this by using a cryptographic common coin protocol in a novel way. It withstands the maximal possible corruption of up to one third of the parties and has expected quadratic message complexity (in the number of parties), which is essentially optimal.

Our atomic broadcast protocol guarantees that a message from an honest party cannot be delayed arbitrarily by an adversary as soon as a minimum number of honest parties are aware of that message. The protocol invokes one multi-valued Byzantine agreement per batch of payload messages that is delivered. An analogous reduction of atomic broadcast to consensus in the crash-fault model has been described by Chandra and Toueg [8], but it cannot be directly transferred to the Byzantine setting.

We also define and implement a variation of atomic broadcast called *secure causal atomic broadcast*. This is a robust atomic broadcast protocol that tolerates a Byzantine adversary and also provides secrecy for messages up to the moment at which they are guaranteed to be delivered. Thus, client requests to a trusted service using this broadcast remain confidential until they are answered by the service and the service processes the requests in a causal order. This is crucial in our asynchronous environment for applying the state machine replication method to services that involve confidential data.

Secure causal atomic broadcast works by combining an atomic broadcast protocol with robust threshold decryption. The notion and a heuristic protocol were proposed by Reiter and Birman [17], who called it "secure atomic broadcast" and also introduced the term "input causality" for its main property. Recent progress in threshold cryptography allows us to present an efficient robust protocol together with a security proof in the random oracle model.

In accordance with the comprehensive survey of fault-tolerant broadcasts by Hadzilacos and Toueg [10], we define and implement our protocols in a modular way, with reliable and consistent broadcasts and Byzantine agreement as primitives. This leads to the following layered architecture:



Important for the presentation of our broadcast protocols is our formal model of a modular protocol architecture, where a number of potentially corrupted parties communicate over an insecure, asynchronous network; it uses complexity-theoretic concepts from modern cryptography. This makes it possible to easily

integrate the formal notions for encryption, signatures, and other cryptographic tools with distributed protocols. The model allows for quantitative statements about the running time and the complexity of protocols; the essence of our definition is to bound the number of steps taken by participants on behalf of a protocol *independently* from network behavior. In view of the growing importance of cryptography for secure distributed protocols, a unified formal model for both is a contribution that may be of independent interest.

Organization of the Paper. For lack of space, only the most important results are described in this extended abstract. It begins with a brief account of the formal model and definitions for binary Byzantine agreement and consistent broadcast. Then it presents validated Byzantine agreement and an implementation for the multi-valued case, which is extended to atomic broadcast. More details, in particular the formal model, detailed definitions and proofs, the discussion of related work, and the descriptions of reliable broadcast and secure causal atomic broadcast, can be found in the full version [5].

2 Model

2.1 Overview of the Formal Model

Our system consists of a collection of n interactive Turing machines, of which t are (statically) corrupted by an adversary, modeled by an arbitrary Turing machine. There is a trusted dealer that has distributed some cryptographic keys initially, but it is not used later. Our model differs in two respects from other models traditionally used in distributed systems with Byzantine faults: (1) In order to use the proof techniques of complexity-based cryptography, our model is *computational*: all parties and the adversary are constrained to perform only feasible, i.e., polynomial-time, computations. This is necessary for using formal notions from cryptography in a meaningful way. (2) We make no assumptions about the network at all and leave it under complete control of the adversary. Our protocols work only to the extent that the adversary delivers messages faithfully. In short, *the network is the adversary.* The differences become most apparent in the treatment of termination, for which we use more concrete conditions that together imply the traditional notion of "eventual" termination.

We define termination by bounding a *statistic* measuring the amount of work that honest, uncorrupted parties do on behalf of a protocol. In particular, we use the communication complexity of a protocol for this purpose, which is defined as the length of all protocol messages that are "associated" to the protocol instance. We use the term *protocol message* for messages that the parties send to each other to implement a protocol, in contrast to the *payload messages* that are the subject of the (reliable, consistent, atomic ...) broadcasts among all parties. The specification of a protocol requires certain things to happen under the condition that all protocol messages have been delivered; thus, bounding the length (and also the number) of protocol messages generated by uncorrupted parties ensures that the protocol has actually terminated under this condition.

As usual in cryptography, we prove security with respect to all polynomial-time adversaries. Our notion of an *efficient* (deterministic) protocol requires that the statistic is *uniformly bounded* by a fixed polynomial independent of the adversary. We also define the corresponding notion of a *probabilistically uniformly bounded statistic* for randomized protocols; the expected running time of such a protocol can be derived from this. Both notions are closed under modular composition of protocols, which is not trivial for randomized protocols.

Our model uses the adversary in two roles: to invoke new instances of a protocol through input actions (as an application might do) and to deliver protocol messages (modeling the network).

For simplicity, all protocol messages delivered by the adversary are assumed to be authenticated (implementing this is straightforward in our model).

2.2 Byzantine Agreement

We give the definition of *(binary) Byzantine agreement* (or *consensus* in the crash-fault model) here as it is needed for building atomic broadcast protocols. It can be used to provide agreement on independent *transactions*.

The Byzantine agreement protocol is activated when the adversary delivers an input action to P_i of the form $(ID, \texttt{in}, \texttt{propose}, v)$, where $v \in \{0, 1\}$. When this occurs, we say P_i *proposes* v for transaction ID. A party terminates the Byzantine agreement protocol (for transaction ID) by generating an output action of the form $(ID, \texttt{out}, \texttt{decide}, v)$. In this case, we say P_i *decides* v for transaction ID. Let any protocol message with tag ID or $ID|\ldots$ that is generated by an honest party be *associated* to the agreement protocol for ID.

Definition 1 (Byzantine agreement). *A protocol solves* Byzantine agreement *if it satisfies the following conditions except with negligible probability:*

Validity: *If all honest parties that are activated on a given ID propose v, then any honest party that terminates for ID decides v.*[1]

Agreement: *If an honest party decides v for ID, then any honest party that terminates decides v for ID.*

Liveness: *If all honest parties have been activated on ID and all associated messages have been delivered, then all honest parties have decided for ID.*

Efficiency: *For every ID, the communication complexity for ID is probabilistically uniformly bounded.*

2.3 Cryptographic Primitives

Apart from ordinary digital signature schemes, we use collision-free hashing, pseudo-random generators, robust non-interactive dual-threshold signatures [18], threshold public-key encryption schemes [19], and a threshold pseudo-random function [13,6]. Definitions can be found in the full version [5].

[1] We use the term "validity" for this condition in accordance with most of the literature on Byzantine agreement. Alternatively, one might also adopt the terminology of fault-tolerant broadcasts [10] and instead call it "integrity," to emphasize that it is a general safety condition (in contrast to a liveness condition).

3 Broadcast Primitives: Verifiable Consistent Broadcast

Our multi-valued agreement protocol builds on top of a *consistent broadcast* protocol, which is a relaxation of Byzantine *reliable broadcast* [12]. Consistent broadcast provides a way for a distinguished party to send a message to all other parties such that two parties never deliver two conflicting messages for the same sender and sequence number. In other words, it maintains consistency among the actually delivered payloads with the same senders and sequence numbers, but makes no provisions that two parties do deliver the payloads. Such a primitive has also been used by Reiter [16].

Broadcasts are parameterized by a tag ID, which can also be thought of as identifying a broadcast "channel," augmented by the identity of the sender, j, and by a sequence number s. We restrict the adversary to submit a request for consistent broadcast tagged with $ID.j.s$ to P_i only if $i = j$ and at most once for every sequence number.

A consistent broadcast protocol is activated when the adversary delivers an input action to P_j of the form $(ID.j.s, \texttt{in}, \texttt{c-broadcast}, m)$, with $m \in \{0,1\}^*$ and $s \in \mathbb{N}$. When this occurs, we say P_j *consistently broadcasts* m *tagged with* $ID.j.s$. Only the sender P_j is activated like this. The other parties are activated when they perform an explicit *open* action for instance $ID.j.s$ in their role as receivers (this occurs implicitly in our system model when they **wait for** an output tagged with $ID.j.s$).

A party terminates a consistent broadcast of m tagged with $ID.j.s$ by generating an output action of the form $(ID.j.s, \texttt{out}, \texttt{c-deliver}, m)$. In this case, we say P_i *consistently delivers* m *tagged with* $ID.j.s$. For brevity, we also the terms *c-broadcast* and *c-deliver*.

Definition 2 (Authenticated Consistent Broadcast). *A protocol for* authenticated consistent broadcast *satisfies the following conditions except with negligible probability:*

Validity: *If an honest party has* c-broadcast m *tagged with* $ID.j.s$*, then all honest parties* c-deliver m *tagged with* $ID.j.s$*, provided all honest parties have been activated on* $ID.j.s$ *and the adversary delivers all associated messages.*

Consistency: *If some honest party* c-delivers m *tagged with* $ID.j.s$ *and another honest party* c-delivers m' *tagged with* $ID.j.s$*, then* $m = m'$*.*

Authenticity: *For all* ID*, senders* j*, and sequence numbers* s*, every honest party* c-delivers *at most one message* m *tagged with* $ID.j.s$*. Moreover, if* P_j *is honest, then* m *was previously* c-broadcast *by* P_j *with sequence number* s*.*

Efficiency: *For any* ID*, sender* j*, and sequence number* s*, the communication complexity of instance* $ID.j.s$ *is uniformly bounded.*

The provision that the "adversary delivers all associated messages" is our quantitative counterpart to the traditional "eventual" delivery assumption.

A party P_i that has delivered a payload message using consistent broadcast may want to inform another party P_j about this. Such information might be

useful to P_j if it has not yet delivered the message, but can exploit this knowledge to deliver the payload message itself, maintaining consistency. We call this property the *verifiability* of a consistent broadcast.

Informally, we use *verifiability* like this: when P_j claims that it is not yet in a state to *c-deliver* a particular payload message m, then P_i can send a single protocol message to P_i and when P_j processes this, it will *c-deliver* m immediately.

Definition 3 (Verifiability). *A consistent broadcast protocol is called* verifiable *if the following holds, except with negligible probability: When an honest party has* c-delivered *m tagged with $ID.j.s$, then it can produce a single protocol message M that it may send to other parties such that any other honest party will* c-deliver *m tagged with $ID.j.s$ upon receiving M (provided the other party has not already done so before).*

We call M the message that *completes* the verifiable broadcast. This notion implies that there is a polynomial-time computable predicate $V_{ID.j.s}$ that the receiving party can apply to an arbitrary bit string for checking if it constitutes a message that completes a verifiable broadcast tagged with $ID.j.s$.

A protocol for verifiable authenticated consistent broadcast (denoted VCBC) is given in the full version [5]. It is inspired by the "echo broadcast" of Reiter [16] and based on a threshold signature scheme. Its message complexity is $O(n)$ and its bit complexity is $O(n(|m|+K))$, assuming the length of a threshold signature and a signature share is at most K bits.

4 Validated Byzantine Agreement

The standard notion of validity for Byzantine agreement implements a binary decision and requires that only if *all* honest parties propose the same value, this is also the agreement value. No particular outcome is guaranteed otherwise. Obviously, this still ensures that the agreement value was proposed by *some* honest party for the binary case. But it does not generalize to multi-valued Byzantine agreement, and indeed, all previous protocols for multi-valued agreement [15,20, 14] may fall back to a default value in this case, and decide for a value that *no* honest party proposed.

We solve this problem by introducing an *external validity* condition, which requires that the agreement value is legal according to a global, polynomial-time computable predicate, known to all parties and determined by the particular higher-level application.

Validated Byzantine agreement generalizes the primitive of *agreement on a core set* [2], which is used in the information-theoretic model for a similar purpose (a related protocol was also developed by Ben-Or and El-Yaniv [1]).

4.1 Definition

Suppose there is a global polynomial-time computable predicate Q_{ID} known to all parties, which is determined by an external application. Each party may

propose a value v that should satisfy Q_{ID} and perhaps contains validation information. The agreement domain is not restricted to binary values.

A validated Byzantine agreement protocol is activated by a message of the form $(ID, \texttt{in}, \texttt{v-propose}, v)$, where $v \in \{0,1\}^*$. When this occurs, we say P_i *proposes* v for transaction ID. We assume the adversary activates all honest parties on a given ID at most once. W.l.o.g., honest parties propose values that satisfy Q_{ID}.

A party terminates a validated Byzantine agreement protocol by generating a message of the form $(ID, \texttt{out}, \texttt{v-decide}, v)$. In this case, we say P_i *decides* v for transaction ID.

We say that any protocol message with tag ID that was generated by an honest party is *associated* to the validated Byzantine agreement protocol for ID. An agreement protocol may also invoke sub-protocols for low-level broadcasts or for Byzantine agreement; in this case, all messages associated to those protocols are associated to ID as well (such messages have tags with prefix $ID|\ldots$).

Definition 4 (Validated Byzantine Agreement). *A protocol solves* validated Byzantine agreement *with predicate Q_{ID} if it satisfies the following conditions except with negligible probability:*

External Validity: *Any honest party that terminates for ID decides v such that $Q_{ID}(v)$ holds.*

Agreement: *If some honest party decides v for ID, then any honest party that terminates decides v for ID.*

Liveness: *If all honest parties have been activated on ID and all associated messages have been delivered, then all honest parties have decided for ID.*

Integrity: *If all parties follow the protocol, and if some party decides v for ID, then some party proposed v for ID.*

Efficiency: *For every ID, the communication complexity for ID is probabilistically uniformly bounded.*

A variation of the validity condition is that an application may prefer one class of decision values over others. Such an agreement protocol may be *biased* and *always* choose the preferred class in cases where other values would have been valid as well.

Validated Byzantine agreement is often used with arguments that consist of a "value" part v and a separate "proof" π that establishes the validity of v. If v is a single bit, we call this the problem of *binary validated agreement*; a protocol for this task is used below.

In fact, we will need a binary validated agreement protocol that is "biased" towards 1. Its purpose is to detect whether there is some validation for 1, so it suffices to guarantee termination with output 1 if $t+1$ honest parties know the corresponding information at the outset. Formally, a *binary validated Byzantine agreement protocol biased towards 1* is a protocol for validated Byzantine agreement on values in $\{0,1\}$ such that the following condition holds:

Biased Validity: If at least $t+1$ honest parties propose $v=1$ then any honest party that terminates for ID decides $v=1$.

We describe two related protocols for multi-valued validated Byzantine agreement below: Protocol VBA, described in Section 4.3, needs $O(n)$ rounds and invokes $O(n)$ binary agreement sub-protocols; this can be improved to a constant expected number of rounds, resulting in Protocol VBAconst, which is described in Section 4.4. But first we discuss the binary case.

4.2 Protocols for the Binary Case

It is easy to see that any binary asynchronous Byzantine agreement protocol can be adapted to external validity and can also be biased.

For example, in the protocol of Cachin, Kursawe, and Shoup [6] one has to "justify" the pre-votes of round 1 with a valid "proof" π. The logic of the protocol guarantees that either a decision is reached immediately or the validations for 0 and for 1 are seen by all parties in the first two rounds. Furthermore, the protocol can be biased towards 1 by modifying the coin such that it always outputs 1 in the first round.

4.3 A Protocol for the Multi-valued Case

We now describe Protocol VBA that implements multi-valued validated Byzantine agreement.

The basic idea is that every party proposes its value as a candidate value for the final result. One party whose proposal satisfies the validation predicate is then selected in a sequence of binary Byzantine agreement protocols and this value becomes the final decision value. More precisely, the protocol consists of the following steps.

Echoing the proposal (lines 1–4): Each party P_i *c-broadcasts* the value that it proposes to all other parties using verifiable authenticated consistent broadcast. This ensures that all honest parties obtain the same proposal value for any particular party, even if the sender is corrupted. Then P_i waits until it has received $n-t$ proposals satisfying Q_{ID} before entering the agreement loop.

Agreement loop (lines 5–20): One party is chosen after another, according to a fixed permutation Π of $\{1,\ldots,n\}$. Let a denote the index of the party selected in the current round (P_a is called the "candidate"). Each party P_i carries out the following steps for P_a:

1. Send a `v-vote` message to all parties containing 1 if P_i has received P_a's proposal (including the proposal in the vote) and 0 otherwise (lines 6–11).
2. Wait for $n-t$ `v-vote` messages, but do not count votes indicating 1 unless a valid proposal from P_a has been received—either directly or included in the `v-vote` message (lines 12–13).
3. Run a *binary* validated Byzantine agreement biased towards 1 to determine whether P_a has properly broadcast a valid proposal. Vote 1 if P_i has received a valid proposal from P_a and add the protocol message

Protocol VBA **for party** P_i**, tag** ID**, and validation predicate** Q_{ID}

LET $V_{ID|a}\big((v,\rho)\big)$ BE THE FOLLOWING PREDICATE:

$V_{ID|a}\big((v,\rho)\big) \equiv (v=0)$ **or**
$\big(v=1$ **and** ρ completes the verifiable authenticated c-broadcast of a message $(\texttt{v-echo}, w_a)$ with tag $ID.a.0$ such that $Q_{ID}(w_a)$ holds$\big)$

UPON RECEIVING MESSAGE $(ID, \texttt{in}, \texttt{v-propose}, w)$:

1: *verifiably authenticatedly c-broadcast* message $(\texttt{v-echo}, w)$ tagged with $ID|\texttt{vcbc}.i.0$
2: $w_j \leftarrow \perp \quad (1 \le j \le n)$
3: **wait for** $n-t$ messages $(\texttt{v-echo}, w_j)$ to be *c-delivered* with tag $ID|\texttt{vcbc}.j.0$ from distinct P_j such that $Q_{ID}(w_j)$ holds
4: $l \leftarrow 0$
5: **repeat**
6: $l \leftarrow l+1; a \leftarrow \Pi(l)$
7: **if** $w_a = \perp$ **then**
8: send the message $(ID, \texttt{v-vote}, a, 0, \perp)$ to all parties
9: **else**
10: let ρ be the message that completes the c-broadcast with tag $ID|\texttt{vcbc}.a.0$
11: send the message $(ID, \texttt{v-vote}, a, 1, \rho)$ to all parties
12: $u_j \leftarrow \perp; \rho_j \leftarrow \perp \quad (1 \le j \le n)$
13: **wait for** $n-t$ messages $(ID, \texttt{v-vote}, a, u_j, \rho_j)$ from distinct P_j such that $V_{ID|a}\big((u_j,\rho_j)\big)$ holds
14: **if** there is some $u_j = 1$ **then**
15: $v \leftarrow (1, \rho_j)$
16: **else**
17: $v \leftarrow (0, \perp)$
18: propose v for $ID|a$ in binary validated Byzantine agreement biased towards 1, with predicate $V_{ID|a}$
19: **wait for** the agreement protocol to decide some (b, σ) for $ID|a$
20: **until** $b = 1$
21: **if** $w_a = \perp$ **then**
22: use σ to complete the verifiable authenticated c-broadcast with tag $ID|\texttt{vcbc}.a.0$ and *c-deliver* $(ID, \texttt{v-echo}, w_a)$
23: output $(ID, \texttt{out}, \texttt{v-decide}, w_a)$
24: **halt**

Fig. 1. Protocol VBA for multi-valued validated Byzantine agreement.

that completes the verifiable broadcast of P_a's proposal to validate this vote. Otherwise, if P_i has received $n - t$ `v-vote` messages containing 0, vote 0; no additional information is needed. If the agreement decides 1, exit from the loop (lines 14–20).

Delivering the chosen proposal (lines 21–24): If P_i has not yet *c-delivered* the broadcast by the selected candidate, obtain the proposal from the value returned by the Byzantine agreement.

The full protocol is shown in Figure 1.

Theorem 1. *Given a protocol for biased binary validated Byzantine agreement and a protocol for verifiable authenticated consistent broadcast, Protocol* VBA *provides multi-valued validated Byzantine agreement for* $n > 3t$.

The message complexity of Protocol VBA is $O(tn^2)$ if Protocol VCBC [5] is used for verifiable consistent broadcast and the binary validated Byzantine agreement is implemented according to Section 4.2.

If all parties propose v and π that are together no longer than L bits, the communication complexity in the above case is $O(n^2(tK + L))$, assuming the length of a threshold signature and a signature share is at most K bits. For a constant fraction of corrupted parties, however, both values are cubic in n.

4.4 A Constant-Round Protocol for Multi-valued Agreement

In this section we present Protocol VBAconst, which is an improvement of the protocol in the previous section that guarantees termination within a constant expected number of rounds. The drawback of Protocol VBA above is that the adversary knows the order Π in which the parties search for an acceptable candidate, i.e., one that has broadcast a valid proposal. Although at least one third of all parties are guaranteed to be accepted, the adversary can choose the corruptions and schedule messages such that none of them is examined early in the agreement loop.

The remedy for this problem is to choose Π randomly during the protocol *after* making sure that enough parties are already committed to their votes on the candidates. This is achieved in two steps. First, one round of commitment exchanges is added before the agreement loop. Each party must commit to the votes that it will cast by broadcasting the identities of the $n - t$ parties from which it has received valid `v-echo` messages (using at least authenticated consistent broadcast). Honest parties will later only accept `v-vote` messages that are consistent with these commitments. The second step is to determine the permutation Π using a threshold coin-tossing scheme that outputs a pseudo-random value, after enough votes are committed. Taken together, these steps ensure that the fraction of parties which are guaranteed to be accepted are distributed randomly in Π, causing termination in a constant expected number of rounds.

The details of Protocol VBAconst are described in Figure 2 as modifications to Protocol VBA.

Protocol VBAconst **for party** P_i**, tag** ID**, and validation predicate** Q_{ID}

Modify Protocol VBA for party P_i, tag ID, and validation predicate Q_{ID} as follows:

1. Initialize and distribute the shares for an $(n, t+1)$-threshold coin-tossing scheme $\mathcal{C}_1$ with k''-bit outputs during system setup. Recall that this defines a pseudorandom function F. Let G be a pseudorandom generator according to Section 2.3.
2. Include the following instructions between lines 3 and 4 of Protocol VBA, before entering the agreement loop:

   1: $c_j \leftarrow \begin{cases} 1 & \text{if } w_j \neq \perp \\ 0 & \text{otherwise} \end{cases} \qquad (1 \leq j \leq n)$

   2: $C \leftarrow [c_1, \ldots, c_n]$

   3: *authenticatedly c-broadcast* the message $(\texttt{v-commit}, C)$ tagged with $ID|\texttt{cbc}.i.0$

   4: $C_j \leftarrow \perp \qquad (1 \leq j \leq n)$

   5: **wait for** $n - t$ messages $(\texttt{v-commit}, C_j)$ to be *c-delivered* with tag $ID|\texttt{cbc}.j.0$ such that at least $n - t$ entries in C_j are 1

   6: generate a *coin share* γ of the coin $ID|\texttt{vba}$ and send the message $(ID, \texttt{v-coin}, \gamma)$ to all parties

   7: **wait for** $t + 1$ `v-coin` messages containing shares of the coin $ID|\texttt{vba}$ and combine these to get the value $S = F(ID|\texttt{vba}) \in \{0,1\}^{k''}$

   8: choose a random permutation Π, using the pseudorandom generator G with seed S.
3. Modify the condition for accepting `v-vote` messages (line 13) inside the agreement loop such that $(\texttt{v-vote}, a, 0, \perp)$ from P_j is accepted only if C_j is known and $C_j[a] = 0$. (This involves also waiting for additional messages $(\texttt{v-commit}, C_j)$ to be *c-delivered* as above.)

Fig. 2. Protocol VBAconst for multi-valued validated Byzantine agreement.

Theorem 2. *Given a protocol for biased binary validated Byzantine agreement and a protocol for verifiable consistent broadcast, Protocol* VBAconst *provides multi-valued validated Byzantine agreement for* $n > 3t$ *and invokes a constant expected number of binary Byzantine agreement sub-protocols.*

The expected message complexity of Protocol VBAconst is $O(n^2)$ if Protocol VCBC [5] is used for consistent verifiable broadcast and the binary validated Byzantine agreement is implemented according to Section 4.2.

If all parties propose v and π that are together no longer than L bits, the expected communication complexity in the above case is $O(n^3 + n^2(K + L))$, assuming a digital signature is K bits. The n^3-term, which results from broadcasting the commitments, has actually a very small hidden constant because the commitments can be represented as bit vectors.

5 Atomic Broadcast

Atomic broadcast guarantees a total order on messages such that honest parties deliver all messages with a common tag in the same order. It is well known that

protocols for atomic broadcast are considerably more expensive than those for reliable broadcast because even in the crash-fault model, atomic broadcast is equivalent to consensus [8] and cannot be solved by deterministic protocols. The atomic broadcast protocol given here builds directly on multi-valued validated Byzantine agreement from the last section.

5.1 Definition

Atomic broadcast ensures that all messages broadcast with the same tag *ID* are delivered in the same order by honest parties; in this way, *ID* can be interpreted as the name of a broadcast "channel." The total order of atomic broadcast yields an implicit labeling of all messages.

An atomic broadcast is activated when the adversary delivers an input message to P_i of the form $(ID, \texttt{in}, \texttt{a-broadcast}, m)$, where $m \in \{0,1\}^*$. When this occurs, we say P_i *atomically broadcasts* m *with tag* ID. "Activation" here refers only to the broadcast of a particular payload message; the broadcast channel *ID* must be opened before the first such request.

A party terminates an atomic broadcast of a particular payload by generating an output message of the form $(ID, \texttt{out}, \texttt{a-deliver}, m)$. In this case, we say P_i *atomically delivers* m *with tag* ID. To distinguish atomic broadcast from other forms of broadcast, we will also use the terms *a-broadcast* and *a-deliver*.

The acknowledgement mechanism needed for composition of atomic broadcast with other protocols is omitted from this extended abstract.

Again, the adversary must not request an *a-broadcast* of the same payload message from any particular party more than once for each *ID* (however, several parties may *a-broadcast* the same message).

Atomic broadcast protocols should be *fair* so that a payload message m is scheduled and delivered within a reasonable (polynomial) number of steps after it is *a-broadcast* by an honest party. But since the adversary may delay the sender arbitrarily and *a-deliver* an a priori unbounded number of messages among the remaining honest parties, we can only provide such a guarantee when at least $t + 1$ honest parties become "aware" of m. Our definitions of validity and of fairness require actually that only after $t + 1$ honest parties have *a-broadcast* some payload, it will be delivered within a reasonable number of steps. This is also the reason for allowing multiple parties to *a-broadcast* the same payload message—a client application might be able to satisfy this precondition through external means and achieve guaranteed fair delivery in this way. Fairness can be interpreted as a termination condition for the broadcast of a particular payload m.

The *efficiency* condition (which ensures fast termination) for atomic broadcast differs from the protocols discussed so far because the protocol for a particular tag cannot terminate on its own. It merely stalls if no more undelivered payload messages are in the system and must be terminated externally. Thus, we cannot define efficiency using the absolute number of protocol messages generated. Instead we measure the progress of the protocol with respect to the number of messages that are *a-delivered* by honest parties. In particular, we

require that the number of associated protocol messages does not exceed the number of *a-delivered* payload messages times a polynomial factor, independent of the adversary.

We say that a protocol message is *associated* to the atomic broadcast protocol with tag *ID* if and only if the message is generated by an honest party and tagged with *ID* or with a tag $ID|\ldots$ starting with *ID*. In particular, this encompasses all messages of the atomic broadcast protocol with tag *ID* generated by honest parties and all messages associated to basic broadcast and Byzantine agreement sub-protocols invoked by atomic broadcast.

Fairness and efficiency are defined using the number of payload messages in the "implicit queues" of honest parties. We say that a payload message m is *in the implicit queue of a party P_i (for channel ID)* if P_i has *a-broadcast* m with tag *ID*, but *no honest party* has *a-delivered* m tagged with *ID*. The *system queue* contains any message that is in the implicit queue of *some* honest party. We say that one payload message in the implicit queue of an honest party P_i is *older* than another if P_i *a-broadcast* the first message before it *a-broadcast* the second one.

When discussing implicit queues at particular points in time, we consider a sequence of events $E_1, \ldots, E_{k'''}$ during the operation of the system, where each event but the last one is either an *a-broadcast* or *a-delivery* by an honest party. The phrase "at time τ" for $1 \leq \tau \leq k'''$ refers to the point in time just *before* event E_τ occurs.

Definition 5 (Atomic Broadcast). *A protocol for* atomic broadcast *satisfies the following conditions except with negligible probability:*

Validity: *There are at most t honest parties with non-empty implicit queues for some channel ID, provided the adversary opens channel ID for all honest parties and delivers all associated messages.*

Agreement: *If some honest party has* a-delivered *m tagged with ID, then all honest parties* a-deliver *m tagged with ID, provided the adversary opens channel ID for all honest parties and delivers all associated messages.*

Total Order: *Suppose an honest party P_i has* a-delivered $m_1, \ldots, m_s$ *with tag ID, a distinct honest party P_j has* a-delivered $m'_1, \ldots, m'_{s'}$ *with tag ID, and $s \leq s'$. Then $m_l = m'_l$ for $1 \leq l \leq s$.*

Integrity: *For all ID, every honest party* a-delivers *a payload message m at most once tagged with ID. Moreover, if all parties follow the protocol, then m was previously* a-broadcast *by some party with tag ID.*

Fairness: *Fix a particular protocol instance with tag ID. Consider the system at any point in time τ_0 where there is a set $\mathcal{T}$ of $t+1$ honest parties with non-empty implicit queues, let $\mathcal{M}$ be the set consisting of the oldest payload message for each party in $\mathcal{T}$, and let S_0 denote the total number of distinct payload messages* a-delivered *by any honest party so far. Define a random variable U as follows: let U be the total number of distinct payload messages* a-delivered *by honest parties at the point in time when the first message in $\mathcal{M}$ is* a-delivered *by any honest party, or let $U = S_0$ if this never occurs. Then $U - S_0$ is uniformly bounded.*

Efficiency: *For a particular protocol instance with tag ID, let X denote its communication complexity, and let Y be the total number of distinct payload messages that have been* a-delivered *by any honest party with tag ID. Then, at any point in time, the random variable $X/(Y+1)$ is probabilistically uniformly bounded.*

5.2 A Protocol for Atomic Broadcast

Our Protocol ABC for atomic broadcast uses a secure digital signature scheme $\mathcal{S}$ and proceeds as follows. Each party maintains a FIFO queue of not yet *a-delivered* payload messages. Messages received to *a-broadcast* are appended to this queue whenever they are received. The protocol proceeds in asynchronous global rounds, where each round r consists of the following steps:

1. Send the first payload message w in the current queue to all parties, accompanied by a digital signature σ in an `a-queue` message.
2. Collect the `a-queue` messages of $n-t$ distinct parties and store them in a vector W, and propose W for validated Byzantine agreement.
3. Perform multi-valued Byzantine agreement with validation of a vector of tuples $W = [(w_1, \sigma_1), \ldots, (w_n, \sigma_n)]$ through the predicate $Q_{ID|\mathsf{abc}.r}(W)$ which is true if and only if for at least $n-t$ distinct tuples j, the string σ_j is a valid $\mathcal{S}$-signature on $(ID, \texttt{a-queue}, r, j, w_j)$ by P_j.
4. After deciding on a vector V of messages, deliver the union of all payload messages in V according to a deterministic order; proceed to the next round.

In order to ensure liveness of the protocol, there are actually two ways in which the parties move forward to the next round: when a party receives an *a-broadcast* input message (as stated above) and when a party with an empty queue receives an `a-queue` message of another party pertaining to the current round. If either of these two messages arrive and contain a yet undelivered payload message, and if the party has not yet sent its own `a-queue` message for the current round, then it enters the round by appending the payload to its queue and sending an `a-queue` message to all parties.

The detailed description of Protocol ABC is given in Figure 3. The FIFO queue q is an ordered list of values (initially empty). It is accessed using the operations *append*, *remove*, and *first*, where *append*(q, m) inserts m into q at the end, *remove*(q, m) removes m from q (if present), and *first*(q) returns the first element in q. The operation $m \in q$ tests if an element m is contained in q.

A party waiting at the beginning of a round simultaneously **waits for** an `a-broadcast` and an `a-queue` message containing some $w \notin d$ in line 2. If it receives an *a-broadcast* request, the payload m is appended to q. If only a suitable `a-queue` protocol message is received, the party makes w its own message for the round, but does not append it to q.

Theorem 3. *Given a protocol for multi-valued validated Byzantine agreement and assuming $\mathcal{S}$ is a secure signature scheme, Protocol* ABC *provides atomic broadcast for $n > 3t$.*

Protocol ABC **for party** P_i **and tag** ID

LET $Q_{ID|\texttt{abc}.r}$ BE THE FOLLOWING PREDICATE:

$$Q_{ID|\texttt{abc}.r}([(w_1,\sigma_1),\ldots,(w_n,\sigma_n)]) \equiv \text{for at least } n-t \text{ distinct } j,\ \sigma_j \text{ is a valid } \mathcal{S}\text{-signature by } P_j \text{ on } (ID, \texttt{a-queue}, r, j, w_j)$$

INITIALIZATION:

$q \leftarrow []$ {FIFO queue of messages to *a-broadcast*}
$d \leftarrow \emptyset$ {set of *a-delivered* messages}
$r \leftarrow 0$ {current round}

UPON RECEIVING MESSAGE $(ID, \texttt{in}, \texttt{a-broadcast}, m)$:

if $m \notin d$ **and** $m \notin q$ **then**
  $append(q, m)$

FOREVER:

1: $w_j \leftarrow \perp; \sigma_j \leftarrow \perp \quad (1 \le j \le n)$
2: **wait for** $q \neq []$ **or** a message $(ID, \texttt{a-queue}, r, l, w_l, \sigma_l)$ received from P_l such that $w_l \notin d$ and σ_l is a valid signature from P_l
3: **if** $q \neq []$ **then**
4: $w \leftarrow first(q)$
5: **else**
6: $w \leftarrow w_l$
7: compute a digital signature σ on $(ID, \texttt{a-queue}, r, i, w)$
8: send the message $(ID, \texttt{a-queue}, r, i, w, \sigma)$ to all parties
9: **wait for** $n-t$ messages $(ID, \texttt{a-queue}, r, j, w_j, \sigma_j)$ such that σ_j is a valid signature from P_j (including the message from P_l above)
10: $W \leftarrow [(w_1,\sigma_1),\ldots,(w_n,\sigma_n)]$
11: propose W for multi-valued validated Byzantine agreement for $ID|\texttt{abc}.r$ with predicate $Q_{ID|\texttt{abc}.r}$
12: **wait for** the validated Byzantine agreement protocol to decide some $V = [(v_1,\tau_1),\ldots,(v_n,\tau_n)]$ for $ID|\texttt{abc}.r$
13: $b \leftarrow \bigcup_{j=1}^{n} v_j$
14: **for** $m \in (b \setminus d)$, in some deterministic order **do**
15: output $(ID, \texttt{out}, \texttt{a-deliver}, m)$
16: $d \leftarrow d \cup \{m\}$
17: $remove(q, m)$
18: $r \leftarrow r + 1$

Fig. 3. Protocol ABC for atomic broadcast using multi-valued validated Byzantine agreement.

The message complexity of Protocol ABC to broadcast one payload message m is dominated by the number of messages in the multi-valued validated Byzantine agreement; the extra overhead for atomic broadcast is only $O(n^2)$ messages. The same holds for the communication complexity, but the proposed values have length $O(n(|m| + K))$, assuming digital signatures of length K bits.

With Protocol VBAconst from Section 4.4, the total expected message complexity is $O(n^2)$ and the expected communication complexity is $O(n^3|m|)$ for an atomic broadcast of a single payload message.

6 Secure Causal Atomic Broadcast

Secure causal atomic broadcast is a useful protocol for building secure applications that use state machine replication in a Byzantine setting. It provides atomic broadcast, which ensures that all recipients receive the same sequence of messages, and also guarantees that the payload messages arrive in an order that maintains "input causality," a notion introduced by Reiter and Birman [17]. Informally, input causality ensures that a Byzantine adversary may not ask the system to deliver any payload message that depends in a meaningful way on a yet undelivered payload sent by an honest client. This is very useful for delivering client requests to a distributed service in applications that require the contents of a request to remain secret until the system processes it. Input causality is related to the standard causal order, which goes back to Lamport [11]; causality is a useful safety property for distributed systems with crash failures, but is actually not well defined in the Byzantine model [10].

Input causality can be achieved if the sender encrypts a message to broadcast with the public key of a threshold cryptosystem for which all parties share the decryption key [17]. The ciphertext is then broadcast using an atomic broadcast protocol; after delivering it, all parties engage in an additional round to recover the message from the ciphertext.

The definition and an implementation of secure causal atomic broadcast on top of atomic broadcast can be found in the full version [5].

References

1. M. Ben-Or and R. El-Yaniv, "Interactive consistency in constant time." Manuscript, 1991.
2. M. Ben-Or, R. Canetti, and O. Goldreich, "Asynchronous secure computation," in *Proc. 25th Annual ACM Symposium on Theory of Computing (STOC)*, 1993.
3. P. Berman and J. A. Garay, "Randomized distributed agreement revisited," in *Proc. 23th International Symposium on Fault-Tolerant Computing (FTCS-23)*, pp. 412–419, 1993.
4. G. Bracha and S. Toueg, "Asynchronous consensus and broadcast protocols," *Journal of the ACM*, vol. 32, pp. 824–840, Oct. 1985.
5. C. Cachin, K. Kursawe, F. Petzold, and V. Shoup, "Secure and efficient asynchronous broadcast protocols." Cryptology ePrint Archive, Report 2001/006, Mar. 2001. `http://eprint.iacr.org/`.

6. C. Cachin, K. Kursawe, and V. Shoup, "Random oracles in Constantinople: Practical asynchronous Byzantine agreement using cryptography," in *Proc. 19th ACM Symposium on Principles of Distributed Computing (PODC)*, pp. 123–132, 2000. Full version available from Cryptology ePrint Archive, Report 2000/034, `http://eprint.iacr.org/`.
7. R. Canetti and T. Rabin, "Fast asynchronous Byzantine agreement with optimal resilience," in *Proc. 25th Annual ACM Symposium on Theory of Computing (STOC)*, pp. 42–51, 1993. Updated version available from `http://www.research.ibm.com/security/`.
8. T. D. Chandra and S. Toueg, "Unreliable failure detectors for reliable distributed systems," *Journal of the ACM*, vol. 43, no. 2, pp. 225–267, 1996.
9. M. J. Fischer, N. A. Lynch, and M. S. Paterson, "Impossibility of distributed consensus with one faulty process," *Journal of the ACM*, vol. 32, pp. 374–382, Apr. 1985.
10. V. Hadzilacos and S. Toueg, "Fault-tolerant broadcasts and related problems," in *Distributed Systems* (S. J. Mullender, ed.), New York: ACM Press & Addison-Wesley, 1993. An expanded version appears as Technical Report TR94-1425, Department of Computer Science, Cornell University, Ithaca NY, 1994.
11. L. Lamport, "Time, clocks, and the ordering of events in a distributed system," *Communications of the ACM*, vol. 21, pp. 558–565, July 1978.
12. L. Lamport, R. Shostak, and M. Pease, "The Byzantine generals problem," *ACM Transactions on Programming Languages and Systems*, vol. 4, pp. 382–401, July 1982.
13. M. Naor, B. Pinkas, and O. Reingold, "Distributed pseudo-random functions and KDCs," in *Advances in Cryptology: EUROCRYPT '99* (J. Stern, ed.), vol. 1592 of *Lecture Notes in Computer Science*, Springer, 1999.
14. K. J. Perry, "Randomized Byzantine agreement," *IEEE Transactions on Software Engineering*, vol. 11, pp. 539–546, June 1985.
15. M. O. Rabin, "Randomized Byzantine generals," in *Proc. 24th IEEE Symposium on Foundations of Computer Science (FOCS)*, pp. 403–409, 1983.
16. M. Reiter, "Secure agreement protocols: Reliable and atomic group multicast in Rampart," in *Proc. 2nd ACM Conference on Computer and Communications Security*, 1994.
17. M. K. Reiter and K. P. Birman, "How to securely replicate services," *ACM Transactions on Programming Languages and Systems*, vol. 16, pp. 986–1009, May 1994.
18. V. Shoup, "Practical threshold signatures," in *Advances in Cryptology: EUROCRYPT 2000* (B. Preneel, ed.), vol. 1087 of *Lecture Notes in Computer Science*, pp. 207–220, Springer, 2000.
19. V. Shoup and R. Gennaro, "Securing threshold cryptosystems against chosen ciphertext attack," in *Advances in Cryptology: EUROCRYPT '98* (K. Nyberg, ed.), vol. 1403 of *Lecture Notes in Computer Science*, Springer, 1998.
20. R. Turpin and B. A. Coan, "Extending binary Byzantine agreement to multivalued Byzantine agreement," *Information Processing Letters*, vol. 18, pp. 73–76, 1984.