Modeling a Bulletin Board Service based on Broadcast Channels with Memory

Severin Hauser[1,2], Rolf Haenni[1]

[1] Bern University of Applied Sciences, CH-2501 Biel, Switzerland
{severin.hauser},{rolf.haenni}@bfh.ch
[2] University of Fribourg, CH-1700 Fribourg, Switzerland
severin.hauser@unifr.ch

Abstract. The publication of the election data is fundamental for making electronic voting systems universally verifiable. For this, voting protocols usually rely on a secure bulletin board, which keeps track of the data produced during the protocol execution. This paper presents a general model for implementing such a bulletin board service. The design of the model is based on the concept of an ideal broadcast channel with memory, which transmits messages without loss of information to a present or future receiver. The challenge of implementing a bulletin board service is to approximate the properties of such an ideal channel to the best possible degree. Our model contributes to a better understanding of these properties and may help in designing future bulletin board implementations.

1 Introduction

To achieve universal verifiability, all parties involved in a cryptographic voting protocol must achieve an agreement on the public data created during the protocol execution. This problem can be seen as a *Byzantine agreement problem* [17]. For some voting contexts like boardroom voting, state-of-the-art Byzantine agreement (or *reliable broadcast*) protocols are reasonable solutions. Unfortunately, for many voting contexts these protocols are not well fitted for two reasons. First, these protocols are not sufficiently efficient on a large scale with many computationally limited parties. Secondly, the model for these protocols assumes that all honest parties are available at the moment of the protocol execution. For parties such as voters in real-world political elections, this is not a realistic assumption. Their limited connectivity could even lead to the point where no agreement can be achieved at all. Cryptographic voting is not the only application with this kind of problems. Other applications that have to deal with similar problems are online auctions or cryptographic currencies. From a more general viewpoint, these applications can be regarded as secure multi-party computation problems with a public audit, in which external auditors can check whether the protocol output was computed correctly or not [2].

So instead of applying Byzantine agreement protocols, papers from the cryptographic voting literature often refer to a *broadcast channel with memory* (BCM)

for making the public election data available to everyone. The existence of a BCM is often assumed without providing a detailed definition of what a BCM is and without specifying its properties [6,20]. The lack of proper definitions is a problem for the general understanding of the cryptographic protocols and for analyzing their security properties. To the best of our knowledge, the first proposal for a formal BCM model has been published recently in [14]. According to this model, which defines a BCM as an idealized theoretical construct, messages can be transmitted instantaneously and without loss to a present or future receiver.

In a real-world implementation of a given cryptographic protocol, the theoretical model of a BCM can at best be approximated. A common approach is to substitute the BCM with a service provided by one or more additional protocol parties. The job of these parties is to receive and memorize the messages transmitted over the broadcast channel during the protocol execution. A group of parties providing this service is what we call *bulletin board service* (BBS). Its goal is to guarantee that all submitted messages are recorded, that messages are never deleted or modified, and that the order in which the messages appeared is tracked. Bulletin board implementations with this property are called *append-only*. In addition, some voting protocols require designated board sections for all involved parties [6], while other protocols require that the board rejects messages that are not well-formed [13]. When implementing a BBS, appropriate solutions for such protocol-specific requirements need to be provided in addition to the append-only property. Since a large amount of the available BBS literature focuses on providing solutions for a specific cryptographic protocol, distinguishing the properties derived from the BCM and the ones introduced by the cryptographic protocol is sometimes difficult.

1.1 Contribution and Paper Overview

In Section 2, we introduce formal definitions for various types of channels, including a definition for a broadcast channel with memory. We summarize the model presented in [14] and expand it with the concept of return-link channels. Our definitions describe how such channels behave under ideal circumstances. As such, they serve as a guideline for the design of a bulletin board service, which mimics the ideal behaviour of a broadcast channel with memory under real-world circumstances.

The main contribution of this paper is a proposal for a general BBS model, which we introduce in Section 3. This model is derived from the BCM definition with the goal of providing an analogous functionality and similar guarantees. Based on the necessary communication channels to the rest of the system, we identify several communication roles and describe the tasks and responsibilities of parties fulfilling these roles. We illustrate the generality of the model with some real-world examples.

1.2 Related Work

The idea of publishing the election data on a public bulletin board has a long tradition in the literature of verifiable electronic voting. While almost every existing cryptographic voting protocol uses a BBS as a central communication platform between the parties involved, almost no paper describing such a protocol gives a precise specification of the properties expected from the board. Usually, the existence of an appropriate BBS is just taken for granted, but the BBS itself remains a black box.

Given the importance of the bulletin board concept in electronic voting, only a remarkably small number of specific papers is devoted to the problem of specifying and implementing a BBS. Peters was one of the first to suggest such a specification and solution [20]. His main focus was on making the bulletin board robust against failures or attacks, using multiple peers and protocols from the multi-party computation literature. In [15, 18], Heather and Lundin made some proposals to ensure the append-only property and to solve the resulting conflicts with the robustness property. Some reports on corresponding implementations have been published later [3, 16]. Another description of a practical BBS implementation is included in the report about the voting system used in the state of Victoria, Australia [4]. In a follow-up paper [9], Culnane and Schneider proposed a robust algorithm for a peered bulletin board and verified its correctness formally. Recently Dold and Grothoff presented a Byzantine consensus protocol that allows to synchronize a set of elements [10]. They use it to implement the bulletin board for an e-voting system that is based on the protocol proposed by Cramer et al. [7].

2 Broadcast Channel With Memory

Many cryptographic voting protocols in the literature assume the existence of a broadcast channel with memory to achieve universal verifiability. The BCM is used by the involved parties to exchange public data during the execution of the protocol. Unfortunately, a proper formal definitions of the core functionalities and properties of a BCM is often entirely missing. This lack of proper definitions leads to problems in the understanding of the cryptographic protocols and their security properties. In this section, based on the notion of a distributed system, we give formal definitions of broadcast channels and broadcast channels with memory. Our model is both a summary and an extension of the BCM model proposed in [14].

2.1 Distributed Systems and Channels

A *distributed system* (Ω, Γ) consists of a finite set of *parties* $\Omega = \{p_1, \ldots, p_n\}$ and a finite set of *channels* $\Gamma = \{c_1, \ldots, c_m\}$. The parties in this system exchange messages over the available channels to achieve some security (and other) objectives in the context of a given problem domain. It is usually assumed that the channels

provide some properties such as authenticity or confidentiality. In our model of a distributed system, we assume—as a general rule—that all channels are *ideal*. This means that they are *noiseless*, possess *unlimited capacity*, and provide a *total message order*. This implies that no message can be lost or modified during transmission, that messages of arbitrary size are transmitted instantaneously, and that no two messages can be sent at the exact same point in time.

Definition 1. A *(ideal) channel* $c \in \Gamma$ of a distributed system (Ω, Γ) is defined by a *sender domain* $S_c \subseteq \Omega$ (the parties that can send messages over c), a *receiver domain* $R_c \subseteq \Omega$ (the parties that can receive messages over c), and a *message space* $\mathcal{M}_c \subseteq \mathcal{M}$ (the messages that can be transmitted over c). If $s \in S_c$ transmits $m \in \mathcal{M}_c$ over c to R_c, then every $r \in R_c$ receives m instantaneously when m is sent. Parties $p \in \Omega \setminus R_c$ not from the receiver domain can observe the transmission of m over c, but can not learn any information about m itself (except its length). On the other hand, parties $p \notin \Omega$ not belonging to the distributed system do not have access to the channels and can therefore not even observe the transmission of m.

The general definition of an ideal channel includes a number of useful limiting cases, which are important in cryptographic protocols. We call $c \in \Gamma$ a *public channel*, if $S_c = \Omega$. This means that every party in the system is able to send messages over c. Similarly, c is called *broadcast channel*, if $R_c = \Omega$.[3] In this case, every message transmitted is received by all parties in the system. If $S_c = \{s\}$ consists of a single sender $s \in \Omega$, then c is an *authentic channel*. Receiving m over such an authentic channel guarantees that s is the author of m. Similarly, if $R_c = \{r\}$ consists of a single receiver $r \in \Omega$, then c is a *confidential channel*. In this case, the channel guarantees that no party other that r learns anything about m (beyond its length). If the sender and receiver domains are identical, i.e., if $S_c = R_c$, we speak of a *closed group channel*. In this case, every member of the closed group can send and receive messages over c.

Some of the above properties are mutually exclusive. For instance, a broadcast channel can not be confidential and a public channel can not be authentic (except for $|\Omega| = 1$). On the other hand, there are a number of useful combinations that are very common in cryptographic protocols. Most importantly, if c is authentic and confidential at the same time, i.e., if both $S_c = \{s\}$ and $R_c = \{r\}$ consist of a single party only, it is called a *secure channel*. A secure channel is called *untappable* in the special case of $\Omega = \{s, r\}$. This implies that no other party can observe the transmission of messages between s and r.

Definition 2. A *return-link channel* is a channel $c \in \Gamma$ of a distributed system (Ω, Γ) that creates temporary *return-links* from every receiver $r \in R_c$ of a

[3] In the literature, broadcast channels are defined in many different ways, for example as a $(m, \lambda, \ldots, \lambda) \mapsto (m, m, \ldots, m)$, where λ denotes an empty message. Such a *broadcast functionality* is an important building block for designing secure multi-party computation protocols in the presence of active adversaries. Assuming a public-key infrastructure, such broadcast channels can be implemented for any number of malicious parties using a signature scheme [12].

transmitted message $m \in \mathcal{M}_c$ to the sender $s \in S_c$ of the message. Return-links can be used by a receiver for sending a response $\ell \in \mathcal{L}_c$ to the (possibly unknown) sender s, where $\mathcal{L}_c$ denotes the message domain of the return-links created by c. By submitting ℓ over the return-link, r does not learn more about s other than $s \in S_c$. The transmission of ℓ from r to s is instantaneous and noiseless. Parties other than s and r can observe the transmission, but they do not learn anything about ℓ.

Return links are expected to be available only for a short time after the transmission of a message. The exact purpose of sending back a response is not further specified, but in most cases it will be something like an acknowledgement, receipt, status report, error message, etc. In protocols relying on such responses, return-link channels are useful to reduce the total amount of necessary channels between the parties. They are also useful to return the response directly to the actual sender $s \in S_c$, even if S_c contains multiple parties and therefore s is unknown to r. Return-links with such properties are widely available in the real world, for example in the case of TCP connections. An example of a distributed system with 10 parties and 3 channels is shown in Figure 1. Regular channels are depicted as single-headed arrows and return-link channels as double-headed arrows.

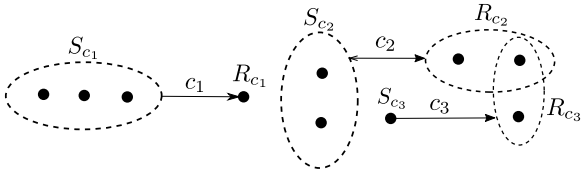


Fig. 1: Example of a distributed system with a confidential channel c_1, a return-link channel c_2, and an authentic channel c_3.

2.2 Broadcast Channel with Memory

In our ideal model of noiseless channels with unlimited capacities, we assume that every submitted message reaches every receiver from the receiver domain instantaneously, independently of the receiver's actual availability and capacity to process the incoming message. In a non-ideal setting, receivers might not always be capable of processing the messages the moment they arrive. They might even miss some incoming messages entirely. In cryptographic protocols, in which broadcast channels are used to spread information to everyone, this imperfection can cause complicated coordination problems.

To allow a receiver to recover from messages lost during the protocol execution, we introduce the concept of a channel with memory. The idea is that

the transmission of a message $m \in \mathcal{M}_c$ over a channel $c \in \Gamma$ is performed by two operations $s : \mathsf{Send}_c(m)$ and $r : \mathbf{M}_c \leftarrow \mathsf{Receive}_c()$. The former is invoked by the sender $s \in S_c$ and the latter by the receiver $r \in R_c$. A channel with memory guarantees that the messages and the order in which they have been sent are stored and never lost. For this, the channel maintains an internal state, called *channel history* $\mathbf{M}_c$, which is initialized by $\mathbf{M}_c \leftarrow \langle\rangle$ and updated by $\mathbf{M}_c \leftarrow \mathbf{M}_c \| \langle m \rangle$ each time a new message $m \in \mathcal{M}_c$ is sent. This idea is further formalized in the following definition.

Definition 3. A channel $c \in \Gamma$ of a distributed system (Ω, Γ) with sender domain $S_c \subseteq \Omega$ and receiver domain $R_c \subseteq \Omega$ is called *channel with memory*, if every $s \in S_c$ can perform the operation $s : \mathsf{Send}_c(m)$ to send a message $m \in \mathcal{M}_c$ over the channel and every $r \in R_c$ can perform $r : \mathbf{M}_c \leftarrow \mathsf{Receive}_c()$ to receive the current channel history $\mathbf{M}_c \in \mathcal{M}_c^*$ of all messages sent so far. An ideal channel with memory has unlimited capacity, i.e., $\mathbf{M}_c$ can get arbitrarily large.

The concept of a channel with memory applies to all particular channel types described before. For example, a *broadcast channel with memory* (BCM) gives every party permanent access to all the messages sent over this channel. In an authentic BCM, it is guaranteed that every message included in the channel history has been sent by the same single sender. In a public BCM, the senders of the messages are unknown within Ω (except for $|\Omega| = 1$). Authentic and public broadcast channels with memory are the most useful instances in cryptographic voting protocols, for example to provide authentic broadcasting to the election authorities and public broadcasting to the voters. If a protocol provides multiple authentic broadcast channels with different senders, it may be necessary to augment Definition 3 to support a common history over multiple channels. For this, we refer to the definition of a *bundled broadcast channel with memory* (BBCM), which additionally keeps track of the sender of every transmitted message [14].

3 Bulletin Board Service

The concept of a BCM as described in the previous section is an idealized theoretical construct, for which no one-to-one practical implementation exists in the real world. For this reason, cryptographic protocols that require such a channel need a substitution that provides an equivalent functionality and similar guarantees. Knowing that the exact same properties of an ideal broadcast channel with memory can at best be approximated by a practical implementation, designing such a substitution is a very delicate problem on its own. A common approach in the literature is to add one or multiple additional parties offering the service of a *bulletin board* to the other parties of the distributed system. In this section, we introduce a general model for such a *bulletin board service* (BBS). We first describe the desired properties of a BBS and the basic functionality. Then we introduce various roles for the parties involved and show how several examples from the real world fit into the general model.

3.1 Guarantees

In Section 2, we introduced the concept of a BCM as a channel with ideal properties. For a BBS to offer similar properties, we identified a number of guarantees that seem to be crucial for a BBS to provide. Knowing that the ideal BCM properties can at best be approximated, it is important to have at least a clear understanding of some realistic goals and an overview of the possibilities for reaching them. For each guarantee introduced below, we refer to the corresponding BCM property, from which it is derived.

Authentication. This addresses the fact, that an ideal BCM c only allows parties from its sender domain S_c to submit messages. For the BBS, this means that the sender of a message must be authenticated to ensure that only messages from parties belonging to the sender domain are accepted. If the authentication evidence provided by the sender is transferable to third parties, it can be recorded together with the message and forwarded to third parties on request. Digital signatures are examples of such transferable authentication evidence, which ensures sender authenticity without relying on trust in the BBS.

Non-Discrimination. When a message is transmitted over an ideal BCM, parties are discriminated only with respect to the channel's sender domain. Therefore, the BBS needs to ensure that no party from the sender domain is excluded from accessing the interface provided by the service for submitting a message. Similarly, it must be ensured that no party is discriminated against retrieving the set of all submitted messages. Known solutions for this are based on the assumption that at least some of the parties responsible for accepting the incoming messages and disseminating the recorded messages behave correctly.

Message Ordering. An ideal channel with memory records and returns the transmitted messages in perfect chronological order. Therefore, the BBS has to provide an equivalent mechanism which ensures a unique message ordering even if a large amount of messages is submitted almost simultaneously. This message ordering has to be immutable and everyone must be able to verify its correctness. Since submitting messages over real-world channels always implies some delay and the BBS might need time for processing them, implementing a perfect chronological order is very difficult. Therefore, the goal of a BBS implementation is to provide an order that approximates the perfect chronological ordering as close as possible.

Message Well-Formedness The message space $\mathcal{M}_c$ of a BCM c restricts the type and format of the transmitted messages. This restriction can be transferred easily to a BBS by performing corresponding checks for each incoming message. Valid messages $m \in \mathcal{M}_c$ are accepted and recorded, whereas invalid messages $m \notin \mathcal{M}_c$ are rejected.

Uniqueness. An ideal BCM c has a unique and unchangeable channel history $\mathbf{M}_c$ consisting of all previously submitted messages in chronological order. Therefore, the BBS also has to ensure that set of recorded messages is unique and can not be altered, i.e., all parties retrieving the board content will receive compatible views. More precisely, if two parties request the board content at two different points in time $t_1 < t_2$, then the list of messages retrieved at t_1 must be a prefix of the list retrieved at t_2.

Completeness. With an ideal BCM c, submitted messages are recorded instantaneously and added to the channel history without any delay. This implies that the BCM always returns the complete channel history $\mathbf{M}_c$ of all messages submitted so far. Therefore, a BBS also needs to ensure that submitted messages are processed as quickly as possible and that requests are always responded with the complete board content of all recorded messages. The maximal necessary time for a message to be processed by the BBS is denoted by Δ. This value is an important characteristics of a given BBS implementation.

3.2 Basic Model and Functionality

Let (Ω, Γ) be a distributed system with a single broadcast channel with memory $c_{\mathsf{BCM}} \in \Gamma$. Replacing c_{BCM} by a BBS means to introduce an extended distributed system (Ω', Γ'), where $\Omega' = \Omega \cup \Phi$ denotes the extended set of parties and $\Gamma' = (\Gamma \setminus \{c_{\mathsf{BCM}}\}) \cup \Psi$ the updated set of channels. The elements of Φ and Ψ are called *bulletin board parties* and *bulletin board channels*, respectively. The BBS must be designed in a way that all parties from $S_{c_{\mathsf{BCM}}}$ have access to a channel in Ψ that connects them with the bulletin board parties for submitting a message to the BBS. Similarly, all parties from Ω must have access to a channel for receiving the channel history $\mathbf{M}_{\mathsf{BCM}}$ from the bulletin board parties. The bulletin board parties themselves may be mutually connected over additional (possibly authentic or confidential) channels to coordinate their current state of memory. To accomplish the substitution of c_{BCM}, it is crucial that Ψ introduces no new channel with memory, i.e., that all channels in Γ' can be realized using standard communication and network technology.

For a BBS to provide the same functionality as c_{BCM}, it needs to provide operations similar to $\mathsf{Send}_{c_{\mathsf{BCM}}}(m)$ and $\mathsf{Receive}_{c_{\mathsf{BCM}}}()$. To avoid confusion between channel and service, we call them $\mathsf{Post}_{\mathsf{BBS}}(p)$ and $\mathsf{Get}_{\mathsf{BBS}}()$, where $p = (m, \alpha)$ contains the broadcast message $m \in \mathcal{M}_{\mathsf{BCM}}$ and some meta-data $\alpha \in \mathcal{A}$. The purpose and format of α depends on the concrete realization of the BBS, but it often contains some *authentication evidence* such as a digital signature, that can be verified by third parties. The pair p itself is called *post* and the process of submitting p to the BBS is called *posting*. If a party posts p to the BBS, it sends it over one of the available bulletin board channels to one or multiple bulletin board parties, which are responsible for the further processing of p.

To be as general as possible, we assume that posts are processed in blocks $b = (\{p_1, \ldots, p_s\}, \beta)$, where $\beta \in \mathcal{B}$ denotes some meta-data added to the block by the bulletin board parties. The main purpose of β is to provide some *publishing*

evidence such as a signed hash chain, which again can be verified by third parties. Note that the posts included in a block are unordered. The internal processing of a block b by the bulletin board parties is called *publication* of b.

Depending on the block size, there are different publication modes. If the block size s is a fixed value for all blocks, it means that the incoming posts are buffered until the block size is reached. If the blocks are created periodically, for example one block every minute, we obtain blocks of different sizes. Corresponding publication modes are called *buffered publication* and *periodical publication*, respectively. A fixed block size $s = 1$ is an important special case of buffered publication, in which individual posts are published immediately. This particular mode is called *immediate publication*. The selected publication mode is an important characteristics of a concrete BBS implementation.

The bulletin board parties are responsible for keeping track of all the processed blocks of posts. This internal state of the BBS consisting of all blocks is called *board history*. One can think of it as an initially empty list $\mathbf{B} \leftarrow \langle\rangle$, which is updated to $\mathbf{B} \leftarrow \mathbf{B} \| \langle b \rangle$ each time a new block b has been formed. This means that $\mathbf{B}$ contains the list of all blocks published by the BBS so far. Therefore, $\mathbf{B}$ is also the expected return value of $\mathsf{Get}_{\mathsf{BBS}}()$, which then enables the derivation of the channel history $\mathbf{M}_{\mathsf{BCM}}$ by extracting the messages from the individual posts in the blocks. Note that immediate publication is the only mode that implies a unique message ordering. This is because the order over the blocks is fixed and in this case each block contains only one message.

3.3 Basic Roles

Every party involved in a BBS has a certain role with corresponding tasks. As the specific roles and tasks depend greatly of the protocol used to run the BBS, we can not introduce the roles and tasks in a general way. However, given the general goal of providing two basic operations $\mathsf{Post}_{\mathsf{BBS}}(p)$ and $\mathsf{Get}_{\mathsf{BBS}}()$, we identified four basic *communication roles*, which depend on how a given bulletin board party is involved in communicating with the main protocol parties. To provide the basic functionality, the BBS must provide at least two channels, one for collecting the posts submitted by parties from the sender domain $S_{c_{\mathsf{BCM}}}$ and one for disseminating the board history to Ω. Without loss of generality, we assume that both channels have a return-link (see explanations given below). An auxiliary channel for broadcasting additional information about the current board state may be necessary to achieve some of the guarantees. The receiver and sender domains of these three channels define three different communication roles, which we call *collector*, *disseminator*, and *broadcaster*. The channels are denoted by $c_C, c_D, c_B \in \Psi$, respectively.

Figure 2 shows the communication roles of the bulletin board parties and illustrates how the parties interact with the rest of the system over the associated channels. The figure also shows how these roles could overlap depending on the protocol. Bulletin board parties not involved in the communication to the rest of the system define an additional role. We call them *associates* and the set of associates is denoted by Φ_A. Below, we give a more detailed description of each

communication role, the interface they provide to the rest of the system, and the attributed tasks.

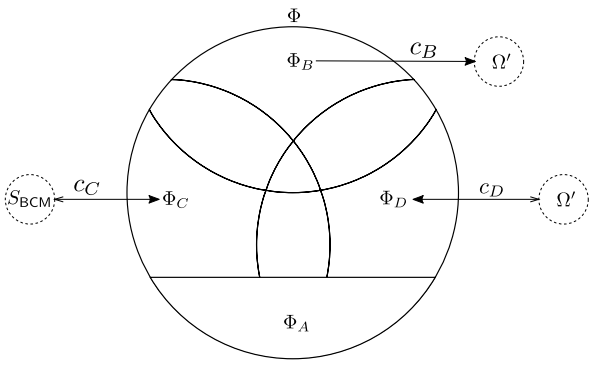


Fig. 2: Illustration of the communication roles and channels of a BBS. Corresponding sets of parties are denoted by Φ_A, Φ_B, Φ_C, and Φ_D. The channels c_C and c_D each have a return-link. The channel c_D is a public channel and the channel c_B is a broadcast channel.

Collector. The collectors are responsible for providing the $\mathsf{Post}_{\mathsf{BBS}}(p)$ operation to the sender domain $S_{c_{\mathsf{BCM}}}$ of the BCM c_{BCM}. Upon receiving a new post $p = (m, \alpha)$ from a party of the sender domain $S_{c_{\mathsf{BCM}}}$ over the channel c_C, the collectors have to check the authenticity and conformity of the message $m \in \mathcal{M}_{\mathsf{BCM}}$ and the meta-data $\alpha \in \mathcal{A}$. In case some check fails, an error message is returned over the return-link of c_C to the sender and the procedure aborts. Otherwise, p is added to the current block for further processing and some response $\gamma \in \mathcal{C}$ (acknowledgment, receipt, status report, etc.) is sent back to the sender. Returning such a response over the return-link extends the signature of the Post-operation as follows:

$$\gamma \leftarrow \mathsf{Post}_{\mathsf{BBS}}(p).$$

Disseminator. The disseminators are responsible for providing the $\mathsf{Get}_{\mathsf{BBS}}()$ operation to the main protocol parties. To avoid unnecessary restrictions, we extend the sender domain of c_D from Ω to $\Omega' = \Omega \cup \Phi$, which implies that c_D becomes a public channel in (Ω', Γ'). It can be used by every involved party to submit a request for the current board history to the disseminators. The return-link of c_D is required for returning the current board history $\mathbf{B}$ to the party requesting it. In addition to returning $\mathbf{B}$, the BBS may also produce and return some meta-data $\delta \in \mathcal{D}$ concerning the request. The following extended signature summarizes the Get-operation:

$$\mathbf{B}, \delta \leftarrow \mathsf{Get}_{\mathsf{BBS}}().$$

Invoking this operation will always return the complete set of blocks of the current board history, even if only some particular blocks or posts are of interest. In a productive environment, where $\mathbf{B}$ could grow into a very large set, this solution might not be very practical. Therefore, we propose an extended Get-operation,

$$\mathbf{B}_q, \delta \leftarrow \mathsf{Get}_{\mathsf{BBS}}(q),$$

which accepts a *query* $q \in \mathcal{Q}$ as input parameter. The query is applied to the board history and only the subset $\mathbf{B}_q \subseteq \mathbf{B}$ of blocks satisfying the query is returned.

Broadcaster. As the disseminators act only on request, the spreading of the board history $\mathbf{B}$ is somewhat limited. Since responding with $\mathbf{B}$ to a request is like a commitment to the current board state, not getting requests over a long period means that no commitments are made for a long time. Under such circumstances, guaranteeing completeness is more difficult. Therefore, we propose an additional broadcast channel c_B for spreading information about the board history to a larger group. The broadcasters are responsible for using this channel, for example each time a new block is added to the board history, or periodically, to broadcast some information $\phi = f(\mathbf{B})$ derived from $\mathbf{B}$ to everyone (for example the current header of a hash chain).

Associate. The group of associates in a BBS only communicates internally with the other bulletin board parties. They support the BBS in achieving its guarantees, for example by issuing signatures for each newly added block. They can also be in charge of maintaining the database, in which the current board state is stored, or of replicating this database for backup purposes.

Some particular bulletin board parties may fulfill the additional role of a *bulletin board trustee*. The set of all trustees is denoted by $\Phi_T \subseteq \Phi$. They are responsible for establishing the trust assumptions of the service, which are necessary for providing the desired guarantees. We assume that each trustee is in possession of a private signature key and that corresponding public keys are publicly known. An important tasks of a trustee is to sign every change made to the board history, i.e., each time a new block is added. In this case, signatures issued by the trustees may be added to the block's meta-data β, possibly together with a digital time-stamp, or they may be returned to the sender as part of the acknowledgment γ. In a similar way, signatures may be created and added to the response δ whenever a party requests the current board history $\mathbf{B}$. Figure 3 shows an example of a BBS with three collectors $\Phi_C = \{p_1, t_1, t_3\}$, three disseminators $\Phi_D = \{p_2, t_2, t_3\}$, three associates $\Phi_A = \{p_3, t_4, t_5\}$, and one broadcaster $\Phi_B = \{p_4\}$. There are five trustees $\Phi_T = \{t_1, t_2, t_3, t_4, t_5\}$.

3.4 BBS Examples

To show that the roles and guarantees introduced in this section can be applied to existing BBS implementations and that they are useful for a better understanding, we sketch here three examples and highlight their properties. An illustrative overview of these examples and the parties involved is given in Figure 4.

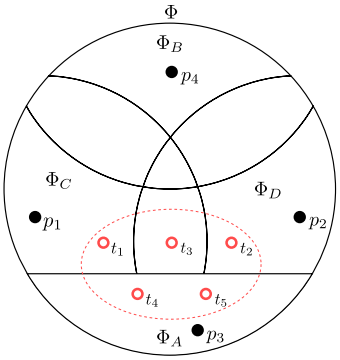


Fig. 3: Example of a BBS with parties $\Phi = \{p_1, p_2, p_3, p_4, t_1, t_2, t_3, t_4, t_5\}$ and trustees $\Phi_T = \{t_1, t_2, t_3, t_4, t_5\}$. We use red circles to represent trustees and black dots for ordinary bulletin board parties.

Single-Party BBS. The setup as shown in Figure 4a is the simplest possible one. It consists of a single trusted bulletin board party, which is responsible for everything. This setting is comparable to a classical central database, which stores the application data and responds to queries. Variations of this simple setup can be found in many implementations of verifiable voting systems [1,5,11]. To guarantee important properties such as completeness, non-discrimination, message-ordering, and uniqueness, this type of BBS relies completely on its single party to be honest. In most practical systems, the decision to adopt such a simple BBS design was due to lack of time or resources.

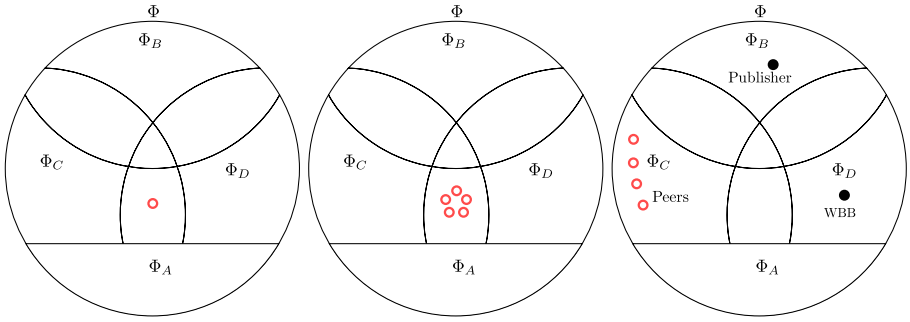
In most single-party implementations of this kind, posts are published immediately, i.e., the block size is equal to 1. The maximal publication time Δ is therefore relatively low. In a proposal by Heather and Lundin [15], in order to guarantee a unique message-ordering, the party submitting a post $p = (m, \alpha)$ first retrieves the current hash chain header from the board and incorporates it into α. The problem is that this may create race condition between multiple parties trying to submit a post simultaneously.

Multi-Party BBS. A setup with multiple trusted bulletin board parties of the same type often emerges from protocols that provide solutions to the *Byzantine Agreement Problem* [17]. All parties are assumed to act identically, and as a group, they are responsible for the proper functioning of the BBS. In the setup shown in Figure 4b, all bulletin board parties act as collectors and disseminators. To guarantee important properties such as completeness, message-ordering, non-discrimination, and uniqueness, agreement protocols between the parties usually require more than $^2/_3$ of the involved parties to be honest. Such agreement protocols are relatively complex and thus limit the throughput of the system. Peters proposed a design of such a setup based on a protocol by Reiter [20, 21], and Beuchat presented an implementation of Peter's approach [3]. An other

implementation is shown by Dold and Grothoff in [10]. They take a different approach as they use a protocol that allows for agreement on sets of messages. This way they only need to reach agreement between all bulletin board parties at the end of the voting period.

Into this category also belong most of the blockchain implementations presented until today, for example the blockchain used in the digital currency *BitCoin* [19]. There, so-called *miners* create a blockchain based on the *proof of work* concept and with an average block size of approximately 1800 transactions every ten minutes. This ensures uniqueness as long as the honest miners control more than half of the computation power. Because BitCoin integrates the miners into the protocol they also cover authentication and message well-formedness as honest miners will only accept blocks containing valid transactions. The other guarantees are addressed by different mechanics of the BitCoin protocol.

vVote System Bulletin Board. Culnane and Schneider presented in [9] a BBS proposal for the *vVote Verifiable Voting System* [8]. As shown in Figure 4c, they introduce parties of different types and for different roles. There is a group of parties, called the *peers*, which act as collectors and trustees. Another party, called the *web bulletin board* (WBB), acts as disseminator and is therefore responsible for spreading the board history. Finally, a party called *publisher* uses a traditional printed newspaper for broadcasting daily the current hash chain header. By creating a new block only once a day, the protocol works with periodic publication. Completeness is ensured by the fixed schedule of the block creation and by the information printed in the newspaper. Uniqueness, non-discrimination and message ordering are guaranteed as long as more than $2/3$ of the peers are honest.



(a) Single-Party BBS (b) Multi-Party BBS (c) vVote

Fig. 4: Examples of existing BBS implementations.

4 Conclusion

In this paper, we presented and extended formal definitions for various types of channels. This also includes a broadcast channel with memory, which is often required in cryptographic voting protocols. Since a broadcast channel with memory has no direct implementation in the real world, it needs to be substituted by something that is often called bulletin board. Based on our definitions, we introduced a model for a bulletin board service. This model describes the guarantees this service must provide and what kind of roles exists inside the service. We showed that the model helps to understand existing bulletin board implementations by discussing some examples.

Acknowledgments This research has been supported by the Hasler Foundation (project no. 14028). We thank the anonymous reviewers for their reviews and appreciate their valuable comments and suggestions.

References

1. Adida, B.: Helios: Web-based open-audit voting. In: Van Oorschot, P. (ed.) SS'08, 17th USENIX Security Symposium. pp. 335–348. San Jose, USA (2008)
2. Baum, C., Damgård, I., Orlandi, C.: Publicly auditable secure multi-party computation. In: Abdalla, M., De Prisco, R. (eds.) SCN'14, 9th International Conference on Security and Cryptography for Networks. pp. 175–196. LNCS 8642, Amalfi, Italy (2014)
3. Beuchat, J.: Append-Only Web Bulletin Board. Master's thesis, Bern University of Applied Sciences, Biel, Switzerland (2012)
4. Burton, C., Culnane, C., Heather, J., Peacock, T., Ryan, P.Y.A., Schneider, S., Srinivasan, S., Teague, V., Wen, R., Xia, Z.: A supervised verifiable voting protocol for the Victorian electoral commission. In: Kripp, M., Volkamer, M., Grimm, R. (eds.) EVOTE'12, 5th International Workshop on Electronic Voting. pp. 81–94. No. P-205 in Lecture Notes in Informatics, Bregenz, Austria (2012)
5. Clarkson, M.R., Chong, S., Myers, A.C.: Civitas: Toward a secure voting system. In: SP'08, 29th IEEE Symposium on Security and Privacy. pp. 354–368. Oakland, USA (2008)
6. Cramer, R., Gennaro, R., Schoenmakers, B.: A secure and optimally efficient multi-authority election scheme. In: Fumy, W. (ed.) EUROCRYPT'97, 16th International Conference on the Theory and Application of Cryptographic Techniques. pp. 103–118. LNCS 1233, Konstanz, Germany (1997)
7. Cramer, R., Gennaro, R., Schoenmakers, B.: A secure and optimally efficient multi-authority election scheme. European Transactions on Telecommunications 8(5), 481–490 (1997)
8. Culnane, C., Ryan, P.Y.A., Schneider, S., Teague, V.: vVote: A verifiable voting system. ACM Transactions on Information and System Security 18(1), 3:1–3:30 (2015)
9. Culnane, C., Schneider, S.: A peered bulletin board for robust use in verifiable voting systems. In: CSF'14, 27th Computer Security Foundations Symposium. pp. 169–183. Vienna, Austria (2014)

10. Dold, F., Grothoff, C.: Byzantine set-union consensus using efficient set reconciliation. In: Wicker, S.B., Engel, D. (eds.) ARES'16, 11th International Conference on Availability, Reliability and Security. pp. 29–38. Salzburg, Austria (2016)
11. Dubuis, E., Fischli, S., Haenni, R., Hauser, S., Koenig, R.E., Locher, P., Ritter, J., von Bergen, P.: Verifizierbare Internet-Wahlen an Schweizer Hochschulen mit UniVote. In: Horbach, M. (ed.) INFORMATIK 2013, 43. Jahrestagung der Gesellschaft für Informatik. pp. 767–788. LNI P-220, Koblenz, Germany (2013)
12. Goldreich, O.: The Foundations of Cryptography – Volume II: Basic Applications. Cambridge University Press (2004)
13. Haenni, R., Koenig, R.E.: A generic approach to prevent board flooding attacks in coercion-resistant electronic voting schemes. Computers & Security 33, 59–69 (2013)
14. Hauser, S., Haenni, R.: Implementing broadcast channels with memory for electronic voting systems. JeDEM – eJournal of eDemocracy and Open Government 8(3), 61–79 (2016)
15. Heather, J., Lundin, D.: The append-only web bulletin board. In: Degano, P., Guttman, J., Martinelli, F. (eds.) FAST'08, 5th International Workshop on Formal Aspects in Security and Trust. pp. 242–256. LNCS 5491, Malaga, Spain (2008)
16. Krummenacher, R.: Implementation of a Web Bulletin Board for E-Voting Applications. Project report, Hochschule für Technik Rapperswil (HSR), Switzerland (2010)
17. Lamport, L., Shostak, R., Pease, M.: The byzantine generals problem. ACM Transactions on Programming Languages and Systems 4, 382–401 (1982)
18. Lundin, D., Heather, J.: The robust append-only web bulletin board. Tech. rep., University of Surrey, Guildford, U.K. (2008)
19. Nakamoto, S.: Bitcoin: A peer-to-peer electronic cash system. Anonymous publication (2009)
20. Peters, R.A.: A Secure Bulletin Board. Master's thesis, Department of Mathematics and Computing Science, Technische Universiteit Eindhoven, The Netherlands (2005)
21. Reiter, M.K.: Secure agreement protocols: Reliable and atomic group multicast in Rampart. In: CCS'94, 2nd ACM Conference on Computer and Communications Security. pp. 68–80. Fairfax, USA (1994)