The append-only web bulletin board

James Heather and David Lundin
University of Surrey, Guildford, Surrey, UK
{j.heather,d.lundin}@surrey.ac.uk

Abstract

A large number of papers on verifiable electronic voting that have appeared in the literature in recent years have relied heavily on the availability of an append-only web bulletin board. *Despite this widespread requirement, however, the notion of an append-only web bulletin board remains somewhat vague, and no method of constructing such a bulletin board has been proposed.*

This paper fills the gap. We identify the required properties of an append-only web bulletin board, and introduce the concept of certified publishing *of messages to the board. We show how such a board can be constructed in order to satisfy the properties we have identified.*

Finally, we consider how to extend the scheme to make the web bulletin board robust and able to offer assurance to writers of the inclusion of their messages.

Although the work presented here has been inspired and motivated by the requirements of electronic voting systems, the web bulletin board is sufficiently general to allow use in other contexts.

1 Introduction

A number of verifiable electronic voting systems require specific data to be made publicly available after or during the election [1, 2, 3, 4, 5, 6, 7, 8, 10, 11, 12, 13, 14, 15, 17, 18]. Sometimes the means of publication is left undiscussed; where it is raised, it is often referred to as an *append-only web bulletin board*. The existence of such a publication vehicle is typically assumed, but the required properties of the web bulletin board are usually not given significant air time, and certainly no attempt has been made at a systematic treatment or at providing a mechanism for implementing an append-only web bulletin board. This is perhaps rather surprising considering the sheer number of papers that rely on the existence of something along these lines.

Our aim here is to identify the properties that the append-only web bulletin board needs to possess, and then to show how one can be built.

1.1 The web bulletin board

The basic idea that emerges from reading through the many papers listed above is as follows. There are three types of agent involved in the system: the *web bulletin board*, *readers*, and *writers*. The web bulletin board needs to allow various parties—the writers—to publish information on the board, so that it can be read by any of the readers. The board is *append-only* in the sense that once something is published it should never be removed or altered, and when something new is published it should be placed at the end of the ordered sequence of messages listed on the board. If something is inserted out of sequence, removed, or altered, this needs to be detectable.

The board is not responsible for generating the content, of course; that is down to the parties that write to the board. When they write something, they provide a digital signature to enable others to verify the origin of the message. The board is, however, responsible for ensuring that signatures are correct, and for ensuring that the content of the board does not change after publication. In the context of a national election, there would presumably be laws governing these responsibilities, with severe sanctions for contravention.

It is, of course, very hard to build a system that can guarantee that information written to it cannot be lost. If the published information is stored in only one place, and that repository suffers catastrophic failure, it might not be possible to recover it, though it might be possible to prove that information has indeed gone missing. For this reason, the first part of this paper looks only at being able to *detect* corruption rather than being able to *prevent* it. This issue of fault tolerance and disaster recovery is one that we will return to in a later section.

The possibility of collusion between a writer of a

message and the board itself should be borne in mind. We will ensure that nothing can be published to the board unless it is signed by both the writer and the board; but even if the two collude, we should still have protection against insertion, deletion or alteration of messages.

1.2 Motivation

The main motivating context, as we have suggested, is that of electronic voting.

1.2.1 Electronic voting

In a verifiable electronic voting system, there are usually various parties who collectively transform the encrypted votes into an election result. Verifiability of the system then rests on allowing these parties to publish certain information that enables anyone (voters, the political parties, the media, election observers, and so on) to check various claims; for example:

- all the encrypted votes [6, 7, 17] might be published, so that voters can check that their votes have been included in the process;
- all the decrypted votes might be published (without anyone knowing the link from encrypted vote to decrypted vote), so that anyone can check the tallying;
- those involved in the decryption might publish zero-knowledge proofs [19] or other information as evidence that they have done their jobs without underhand tactics [6, 7].

1.2.2 Other applications

Although electronic voting provides the primary motivation, our scheme has other applications. For instance:

1. *Auctions.* In the auction context the bidder wishes to place a bid at a particular time, based on a current (opened or closed) bid and receive proof of receipt of the bid. The auctioneer wants protection against allegations of malfeasance, and thus needs to publish a proof that the sequence of bids has not been manipulated in any way.
2. *Auditable discussion boards.* It may be desirable to create a web-based discussion board or forum that provides an auditable history of the discussion.
3. *System logs.* In some contexts, it might be useful to have a verifiable online log of the activities of a distributed system. Typically, log files are written as plain text files, with full trust invested in the system writing the log; sometimes we may wish to weaken the level of trust required in the logger.
4. *Petitions.* One current fashion seems to be signing of online petitions. However, there is usually no security provided at all: no-one can verify that the signatures are not faked, and those who do sign cannot verify that the text of the petition is not subsequently changed. It would clearly increase trust in the final signed petition if we could find a way round these problems.

1.3 Roadmap

The contribution of this paper is split into Sections 2 and 3. In Section 2, we introduce the *certified publishing* concept, identify the properties required of the web bulletin board, and show that our system satisfies those properties. Section 3 then discusses how to extend this with more thorough robustness properties. Finally, we sum up and give conclusions in Section 4.

2 Certified publishing

The web bulletin board must accept submissions only from accredited writers; similarly it must protect itself from accountability that arises from the published data by keeping proof of the origin of messages.

In this section, we start by listing the properties we require of our web bulletin board; then we introduce our scheme and show that it satisfies these properties.

2.1 Required security properties

We identify a number of security properties that an append-only web bulletin board should satisfy.

A web bulletin board is a sequence $\langle wbb_n \rangle$, where each wbb_i contains a message, along with some metadata about the message. We will leave this metadata abstract for the moment, and give details in Section 2.2 of what metadata the web bulletin board publishes in our scheme.

2.1.1 Unalterable history

Definition 2.1. A web bulletin board has *unalterable history* if, whenever a reader retrieves the contents of the board at time T_0 and again at time T_1, it is able to check that the board it read at T_0 is a prefix of the board at T_1, in the sense that the board at T_1 has the same content as previously, except for possibly having

had messages appended. If this is not the case, the reader can detect that the board has become corrupted.

Definition 2.2. A web bulletin board has *certified publishing* if whenever a reader retrieves the contents of the board, either he can detect corruption of the board, or he will have proof, for each message on the board:

1. of who wrote the message;
2. that the writer intended the message to be published with the stated timestamp and at this point in the board's sequence of messages.

These guarantees should hold even if the bulletin board and the writer collude.

2.1.2 Proof of timeliness of acceptance

The following guarantees enable agents to verify that messages were accepted for publication in a timely fashion.

Timeliness is difficult to tie down in a clean fashion, because networks are usually asynchronous, and we have to allow for latency. Although we would like to say, for instance, that if a reader checks the board at time T, he should not subsequently discover a new message published with an earlier date of writing than T, we must allow for the possibility that a writer has constructed a message with a timestamp of $T - \delta$ for some small δ, and the message is still in transit, and so has not yet appeared on the board. For this reason, we introduce some small, fixed security parameter ϵ that appears as a parameter to the following definitions. Increasing the value of ϵ reduces the number of times the board has to reject a message because its claimed publication date is too old by the time it arrives, but increases the extent to which the web bulletin board can deliberately delay decisions on whether to publish a particular message. We do not anticipate that this will cause problems: it should be possible to choose a value of the order of a few milliseconds and still have decent protection against latency. If a message is rejected because the delay exceeds ϵ, the message can always be sent again with a fresh timestamp.[1]

Definition 2.3. The bulletin board has *timely publication* if, whenever a reader views the web bulletin board at a time T, and a message is subsequently published to the web bulletin board with a claimed publication date that is earlier than $T - \epsilon$, the reader can prove that the board has been corrupted.

[1] In fact, we could get away with sending only a fresh timestamp and signature. This might save a lot of time if the message was very long.

The above definition is not subsumed under Definition 2.1. Here, we are dealing with the *time* of publication; there, we were dealing with the *order*. Definition 2.3 says essentially that once someone has viewed the board, nothing more can be published to the board with a publication date of *before* the time of reading.

Definition 2.4. Suppose that the web bulletin board currently contains λ messages. Suppose further that writer W attempts at time T to write message m as message $\lambda + 1$, and that W' attempts at time T' to write m' as message $\lambda + 1$. If in such cases the later of the two messages always wins—that is, if whenever (without loss of generality) $T + \epsilon < T'$, and the earlier message m is published to the board, W' can prove that the board has become corrupted—then we say that the board has *early rejection*. (If $|T - T'| < \epsilon$ then we get no guarantees.)

This last point initially seems strange: one might expect the earlier message to win over the later message. But it makes very little difference which wins, as long as there is a clear and enforceable policy. The practical upshot of forcing the later message to take priority is that if the web bulletin board is to claim that the first message never arrived, it will have to make this decision before allowing any other writers to submit messages. This prevents the web bulletin board from collecting a pool of potential next messages from various writers, and delaying its decision over which to publish until it has received a favourable one.

2.2 The history

The implementation of the bulletin board in our scheme is as follows. The web bulletin board stores a sequence $\langle wbb_n \rangle$, indexed starting from 1, where each $wbb_i = \langle m_i, T_i, W_i, H_i, WSign_i, BSign_i \rangle$, where m_i is a message, T_i is a timestamp, W_i is the name of a writer, and H_i is a hash, and $WSign_i$ and $BSign_i$ are signed terms. The intention is that T_i will store the writer W_i's timestamp at the time of writing message m_i, H_i is a hash that identifies the message as occurring at this point in the sequence, $WSign_i$ is the writer's commitment to the message, and $BSign_i$ is the board's commitment to accepting the message for publication.

Definition 2.5. Such a sequence is called a *history*.

Definition 2.6. The web bulletin board is required to ensure that its history always satisfies the following invariant:

1. $H_i = H(m_i.T_i.W_i.H_{i-1})$, where $H_0 = 0$;

2. $WSign_i = S_{W_i}(H_i)$;

3. $BSign_i = S_B(WSign_i.T_i')$, with T_i' being the web bulletin board's timestamp at the time of signing;

4. $T_i \leq T_i' < T_i + \epsilon$.

A history that has these properties is called a *consistent* history.

Lemma 2.1. *If a history is consistent, then any prefix of the history is also consistent.*

Proof. The proof is a simple induction on the length of the history. □

Definition 2.7. We will use 'wbb_λ' to denote the last element of $\langle wbb_n \rangle$ (that is, the most recent entry on the web bulletin board). The current *state hash* is the value of H_λ. This is the hash value that the next writer will need to use as the third component in constructing $H_{\lambda+1}$.

2.3 Assumptions

There are various assumptions that are required in order to ensure that the web bulletin board achieves the security properties we desire.

In practice, the following assumption would normally be ensured by putting some adequate private key infrastructure (PKI) in place.

Assumption 2.1. All agents know the public keys of all writers and of the web bulletin board itself, but each writer's secret key is known only to that writer, and the web bulletin board's secret key is known only to the web bulletin board.

We need further to assume that the cryptography does its job adequately. Assumptions 2.1 and 2.2 together mean that a message signed with S_{W_i} must have originated with W_i, and a message signed with S_B must have originated with the web bulletin board. Assumption 2.1 is enough to ensure that anyone who sees a signed message can verify the signature.

Assumption 2.2. Signed messages can be produced only by an agent who knows the signing key.

The most important consequence of the following assumption is that the hash function we are using is treated as injective. This is obviously not strictly true of a hash function; however, a good collision-free hash function will effectively achieve this for us by ensuring that hashes do not accidentally collide, and that agents are unable to produce two distinct terms that hash to the same value.

Assumption 2.3. The terms form a free algebra. Essentially, this means that concatenation of terms is associative, and that any two syntactically distinct terms have different values.

2.4 Reading

All information written to the web bulletin board is considered public, and so anyone can act as a reader. The fact that this is usually termed a *web* bulletin board suggests that the transfer is to be done over HTTP, although that will not concern us here. It is also quite possible that, for efficiency reasons, readers might want to retrieve only part of the contents of the board rather than the whole of it, but again, we are not here concerned with questions of how to present the material to the reader. For the purposes of this paper, we shall assume that readers retrieve all of $\langle wbb_n \rangle$ whenever required.

Whenever anything is read from the web bulletin board, it also returns a signed and dated copy of the current state hash:

$$\text{Message 1.} \quad B \to R \ : \ \langle wbb_n \rangle .S_B(H_\lambda.T_B)$$

This makes it impossible for the web bulletin board subsequently to change what was on the board before this point. If the web bulletin board tries to do so, R can prove that the board has been corrupted by producing $S_B(H_\lambda, T_B)$. We will return later to this point.

2.5 Writing

Writing to the board involves three stages: getting the current state hash; sending the message to the board; getting a receipt. The protocol looks like this:

$$\begin{array}{ll} \text{Message 1.} & B \to W \ : \ S_B(H_\lambda, T_B) \\ \text{Message 2.} & W \to B \ : \ m.T.W.H.S_W(H) \\ \text{Message 3.} & B \to W \ : \ S_B(S_W(H).T') \end{array}$$

where $H = H(m.T.W.H_\lambda)$.

In the first message, the board sends the current state hash H_λ to the writer, signed and dated. The writer rejects Message 1 if T_B is more than ϵ old.

In Message 2, the writer sends the message he wishes to publish, along with a newly generated hash, based on the message, the time of writing, the writer's name, and the current state hash; he also sends a signed version of this hash. The web bulletin board rejects Message 2 if $T - T_B > \epsilon$. On receipt of Message 2, the web bulletin board checks the signature, and thus obtains proof that the writer really did write the message at this point; it also checks the hash. The writer's signature commits

to the message, to the timestamp, and to the current state hash; this guarantees that the writer intended the message to appear as the next message in the sequence after the one that resulted in this state hash.

Finally, the board sends back a signed and dated copy of the writer's signed hash. The writer will reject Message 3 if $T' - T > \epsilon$. The writer also checks the signature, and gets proof that the web bulletin board has accepted the message as appearing next in the sequence. If anything else appears in place of m, the writer can produce m and the web bulletin board's signature to show that the web bulletin board has deleted or altered the message.

Of course, when the writer sends Message 2, he has no guarantee of receiving a Message 3 at all, and thus no guarantee of getting proof that the message has been accepted.

At the end of the protocol, provided that the signature and the hash are both correct, and provided that T is fresh, the web bulletin board appends $\langle m.T.W.H.S_W(H).S_B(S_W(H).T')\rangle$ to the history. The new state hash now becomes H.

Proposition 2.2. *If the history was consistent before appending this new message, it will still be consistent afterwards.*

Proof. By inspection. □

2.6 Analysis of security properties

We now consider the security properties we set out in Definitions 2.1 to 2.4, and show that our implementation of the bulletin board satisfies those properties. We start with some preliminary results.

Lemma 2.3. *Any reader who reads the entire history of the web bulletin board has enough information to check that it is consistent.*

Proof. All signatures are immediately verifiable because by Assumption 2.1 all agents have all public keys. The only remaining question is whether the hashes can be verified; this amounts to asking whether the reader knows the information being hashed, in order to reconstruct the hash and check that the values are equal. But this is trivially true. All agents know H_0, because $H_0 = 0$. Then, for every other hash value $H_{k+1} = H(m_{k+1}.T_{k+1}.W_{k+1}.H_k)$, the first three components inside the hash appear in the clear in entry $k+1$ of the web bulletin board, and the last appears in entry k. Finally, anyone can check that $T_i' - T < \epsilon$ for all i. □

We define equivalence of histories based on whether the messages they store, along with the times they were written and the names of the writers, are the same.

Definition 2.8. Suppose that we have two web bulletin board histories $\langle wbb_n\rangle$ and $\langle wbb^*_{n^*}\rangle$.

Suppose that the ith terms of the histories are $\langle m_i, T_i, W_i, H_i, WSign_i, BSign_i\rangle$ for the first history, and $\langle m^*_i, T^*_i, W^*_i, H^*_i, WSign^*_i, BSign^*_i\rangle$ for the second. We say that the two histories are *equivalent* if and only if

1. $n = n^*$;
2. for all $1 \leq i \leq n$, we have $m_i = m^*_i$, $T_i = T^*_i$, $W_i = W^*_i$.

Remark. This notion of equivalence is an equivalence relation.

Lemma 2.4. *Two histories have the same state hash if and only if the histories are equivalent.*

Proof. We first show that two histories with the same state hash are equivalent. To do this, we prove the contrapositive: that two inequivalent consistent web bulletin board histories have different state hashes.

Suppose that we have two consistent histories $\langle wbb_n\rangle$ and $\langle wbb'_{n'}\rangle$, where the ith term of the former is $\langle m_i, T_i, W_i, H_i, WSign_i, BSign_i\rangle$ and the ith term of the latter is $\langle m^*_i, T^*_i, W^*_i, H^*_i, WSign^*_i, BSign^*_i\rangle$. If they are inequivalent, then they differ on some m_i, T_i or W_i, or else the histories are of different length. Since the two histories are consistent, the state hashes for each are correctly constructed from these m_i, T_i and W_i terms; that is, with the first history, for each $1 \leq i \leq n$ we have

$$H_i = H(m_i.T_i.W_i.H_{i-1})$$

and similarly for the second web bulletin board.

A simple mathematical induction on n, together with Assumption 2.3, now establishes that $H_n \neq H^*_{n^*}$.

For the reverse direction, we must show that two equivalent histories have the same state hash. This is also a simple induction. If the histories are equivalent then they are of the same length; suppose our two histories are $\langle wbb_n\rangle$ and $\langle wbb'_n\rangle$. Now an induction on n establishes the result. If $n = 0$, then the state hash in each case is 0. Now suppose that the result holds for all histories of length k. When $n = k + 1$, the state hash for the first board is $H_{k+1} = H(m_{k+1}.T_{k+1}.W_{k+1}.H_k)$, and for the second it is $H'_{k+1} = H(m'_{k+1}.T'_{k+1}.W'_{k+1}.H'_k)$. But the equivalence of the histories tells us that $m_{k+1}.T_{k+1}.W_{k+1} = m'_{k+1}.T'_{k+1}.W'_{k+1}$, and the inductive hypothesis tells us that $H_k = H'_k$; thus, $H_{k+1} = H'_{k+1}$, and the histories have the same state hash. □

We are now in a position to consider the security properties we require of our web bulletin board.

2.6.1 Unalterable history

Retrieving the contents of the web bulletin board, as we have already seen, returns to the reader R the sequence $\langle wbb_n \rangle . S_B(H_\lambda, T_B)$.

By performing appropriate checks when reading the board, the reader can confirm that the board has not been corrupted. By remembering the signed state hash $S_B(H_\lambda, T_B)$, the reader will have enough information to be able to detect later if anything he has already read has changed.

Theorem 2.5. *The web bulletin board has unalterable history (Definition 2.1).*

Proof. Suppose a reader retrieves the contents of the board at time T_0. He will receive, along with the contents $\langle wbb0_{n_0} \rangle$ of the board, $S_B(H_{\lambda_0}, T_0)$. The reader then checks that H_{λ_0} is indeed the state hash of $\langle wbb0_{n_0} \rangle$; he also checks that $\langle wbb0_{n_0} \rangle$ is consistent. Having made these checks, he need store only $S_B(H_{\lambda_0}, T_0)$; he need not cache the entire board.

If he later retrieves the contents $\langle wbb1_{n_1} \rangle$ of the board at T_1, he will receive $S_B(H_{\lambda_1}, T_1)$. He performs the same checks on $\langle wbb1_{n_1} \rangle$ as he did when he retrieved the board the first time.

If all of these checks pass, then $\langle wbb1_{n_1} \rangle$ is consistent. If $n_1 < n_0$ then the later board is shorter than it was, and the reader will know that the board has been corrupted, because something must have been deleted for the board to have shortened. Otherwise, he now considers the sequence consisting of the first n_0 elements of $\langle wbb1_{n_1} \rangle$. This is a prefix of the later board, and so by Lemma 2.1 this is also consistent.

He now looks at this prefix and considers the last element $\langle m_{n_0}, T_{n_0}, W_{n_0}, H_{n_0}, WSign_{n_0}, BSign_{n_0} \rangle$. The state hash of the prefix is H_{n_0}. By Lemma 2.4, the state hash of this prefix is the same as that of $\langle wbb0_{n_0} \rangle$ if and only if the prefix is equivalent to $\langle wbb0_{n_0} \rangle$; so he checks that $H_{n_0} = H_{\lambda_0}$. If this is the case, then he knows that no message or message timestamp or message origin has been altered, and that nothing has been deleted or inserted before this point. If not, he will know that the board has been corrupted. □

Note that this works even in the presence of collusion between the web bulletin board and the writers. Even if the writers produce old signatures (with old timestamps) for the web bulletin board to insert into the sequence, the change of history will mean that the state hash of the prefix will change, and the reader will be able to detect this.

Theorem 2.6. *The web bulletin board has certified publishing (Definition 2.2).*

Proof. When a reader retrieves the contents $\langle wbb_n \rangle$ of the web bulletin board, he first checks that the history is consistent. If not, he is able to detect that the board has been corrupted. If it is consistent, then each element is of the form $\langle m_i, T_i, W_i, H_i, WSign_i, BSign_i \rangle$, where $WSign_i = S_{W_i}(H(m_i.T_i.W_i.H_{i-1}))$. This signature is enough to show that writer W_i created the message—by Assumption 2.1, only W_i has the signing key, and the only time a writer signs a message is when sending one to the board for publication. The writer chooses the timestamp when creating the signature, so the reader knows that the writer intended this to be the timestamp associated with the message.

That the writer intended the message to appear at this point in the sequence is clear from the fact that the writer was prepared to use H_{i-1} as the last component inside the hash to be signed: this value, H_{i-1}, was the state hash before m_i was added to the history. □

2.6.2 Proof of timeliness of acceptance

Theorem 2.7. *The web bulletin board has timely publication (Definition 2.3).*

Proof. If a reader views the board at time T, he will obtain a value of the form $S_B(H_\lambda, T_B)$ from the web bulletin board, with $T < T_B < T + \epsilon$. But this commits the web bulletin board to the claim that at time T_B, the state hash was H_λ. This, by Lemma 2.4, corresponds to some particular history $\langle wbb_\lambda \rangle$.

Suppose that the board subsequently publishes a new message, not present in $\langle wbb_\lambda \rangle$, with a claimed publication date of T_0, where $T_0 + \epsilon < T$. This involves placing an entry into the history of the form $\langle m, T_0, W, H, WSign, BSign \rangle$, where $BSign = S_B(S_W(H(m.T_0.H').T_0{}'))$, with $T_0{}' < T_0 + \epsilon$. But now $T_0{}' < T_0 + \epsilon < T < T_B$.

But this commits the web bulletin board to the claim that at time $T_0{}'$, the state hash corresponded to a history that includes this new entry. Since $T_0{}' < T_B$, the history at $T_0{}'$ should have been a prefix of the history at T_B. But this means that the history at T_B should also have included the new entry, contrary to our previous assumption. □

Theorem 2.8. *The web bulletin board has early rejection (Definition 2.4).*

Proof. Early rejection is an immediate consequence of timely publication and the fact that writing a message involves first reading the state hash from the board.

The first message of the writing protocol is

$$\text{Message 1.} \quad B \to W \ : \ S_B(H_\lambda, T_B)$$

which returns the state hash H_λ to the writer. If another writer now manages to publish a message using the same state hash but at a time earlier than $T_B - \epsilon$, the same argument as that used in Theorem 2.7 will enable the writer to show that the board has become corrupted. □

2.7 Summary

In this section, we have developed a scheme for implementing the append-only web bulletin board, and shown that it satisfies the security properties we required of it.

This is already sufficiently powerful to meet the demands of most, if not all, of the systems that have assumed the existence of an append-only web bulletin board. It provides an implementation that guarantees that the bulletin board cannot manipulate the messages on the board and hope to escape detection.

The bulletin board as presented thus far, however, does not provide any liveness guarantees. There is nothing to stop the board from refusing to communicate with one or more agents. Although the board cannot manipulate the history of previously published messages, it can certainly prevent them from being published or read.

The main motivation for developing an append-only web bulletin board is that many electronic voting systems require such a board. Most of the voting systems that have been proposed are rather weak at present on their ability to recover from disaster, and are unable to cope if one of the core components of the system crashes or refuses to perform its function; consequently, if the board were to be used for one of these applications, it would not significantly weaken the liveness properties of the system as a whole.

However, it is clearly desirable to make the web bulletin board as robust as possible. It would be better if we could construct a distributed board in such a way that guarantees can be made about publication and retention of messages even in the event of one or more machines crashing or becoming compromised.

The aim of the next section is to make some progress towards constructing such a board.

3 Robust publishing

Although the writer and the web bulletin board both seek to gain proof from the other of their correct function, the principal weakness of the scheme presented thus far in the paper is that the web bulletin board may suffer some catastrophic failure that prohibits it from fulfilling its duty to publish the data it has accepted from the writers, or the web bulletin board may stage a denial of service attack, by refusing to communicate with some or all of the readers or writers, with the same result.

3.1 Web bulletin board peers

A natural way of improving the scheme is to create a distributed web bulletin board, consisting of some number of geographically disparate linked peers, each run by a separate organisation. We can then simply replicate the published data across all of them, and if one fails, the others can still function and fulfil the duty of the *collective*.

However, although there exist many practical methods for replicating databases across a set of servers, these do not offer the amount of trust that we seek. If, for example, a particular web bulletin board peer B_x successfully replicates the data it has accepted from the writers and issued certificates for to another peer B_y this does not necessarily give the writers any higher level of confidence in the correct publishing of the data, because they have not been issued a certificate by the web bulletin board *collective* but merely by a single peer. If B_x were to fail—perhaps by having its private key compromised!—and a writer, with a certificate proving the receipt by that peer of a message, were to complain, there would (still) be no way of recovering the messages that had not been replicated to other peers. The writer must be guaranteed, at the time of writing, that its message will survive because it has been replicated to a large enough number of web bulletin board peers.

The approach we adopt here is to require that as long as some threshold set k out of n bulletin board peers survive and function correctly, the integrity of the election should be guaranteed by the collective. In terms of the scheme presented in the previous section, this means that the certificate issued by the web bulletin board to the writer is issued by the web bulletin board peers as a collective. This is facilitated by a threshold cryptography scheme.

3.2 Threshold cryptography scheme

For this improved version of the web bulletin board, we will assume that some particular threshold cryptography scheme, such as ElGamal [9] or Paillier [16], has been agreed upon, and that each peer has been given a secret share of a threshold signing key. The scheme

that we present here is not dependent on the particular cryptographic mechanism used.

We require two things from the threshold scheme: first, that it is a public key scheme, meaning that an encryption under a generally available public key can only be decrypted by the secret (threshold) private key; and secondly, that the private key can be split into n parts in such a way that a threshold subset of k key holders can co-operate to perform the decryption, but $k-1$ key holders together still have no useful information about the threshold key.

3.3 Distributed history: synchronized

The information stored by each peer is exactly the same information as was stored in Section 2. The bulletin board's signing key S_B is now the threshold key, split among the n peers; any k of these can together sign a message.

The rough idea for writing a message is that the writer should send his message to a peer of his choice; the peer will then form a threshold set of peers who can sign the receipt; and this signed receipt is then returned to the writer as proof of acceptance and publication of the message. All of the k peers involved in signing now add the message to their own history, and the message is also sent to the other $n-k$ peers for publication on their boards too. Each of those peers must accept the message as authentic, because it has already been signed by the threshold key.

This is nearly sufficient to give us what we want, but there are a couple of loose ends that want tying up.

3.3.1 Locking the peers

If the peers are to stay synchronized, it is important that we do not have two messages written concurrently. But we need a threshold set of peers to sign a message, and this gives us a neat way round the problem. We start by stipulating that a threshold set must contain more than half of the peers; that is, that $2k > n$. In this way, we can ensure that two threshold sets can never be constructed concurrently.

Peers should never allow themselves to be part of two signing sets at the same time. They may still respond to read requests when in the middle of a signing operation, but they may not respond to other writing requests.

(Coding this would need to be done carefully, of course, to prevent deadlock when two peers each try to construct a threshold set to get a message signed.)

3.3.2 Publish or be damned

When a message is signed by a threshold set, it must then be distributed to the other $n-k$ peers. One must ask what happens if a peer refuses to publish a message on its board.

By this point, the message has been accepted by a threshold set, and is therefore deemed to have been certified by the collective. A peer must not refuse to publish a certified message on its board. If it does so, this constitutes breaking its contract, and it should thereafter be dropped by the other peers.

3.4 Distributed history: unsynchronized

Another possibility for maintaining a distributed web bulletin board is to drop the requirement that all peers should keep track of all of the messages. It is possible to construct the board in such a way that writing to one peer results in a signature from a threshold set, and a consequent guarantee that the message has been replicated to those peers, but not necessarily to all n peers. This involves changing the structure somewhat from that presented in Section 2. What we get is a web bulletin board that maintains a *local* append-only structure, in the sense that each peer keeps an ordered sequence of messages, but the web bulletin board is no longer *globally* append-only.

Following this approach has two interesting consequences. First, the writing of messages need no longer be done strictly sequentially: two or more messages can be written concurrently. In a very large-scale system, this could be a considerable advantage, because locking a threshold set of peers might be a time-consuming operation.

Secondly, in order to make sure one has read all of the messages, one now needs to consult $n-k+1$ peers. (Each message is guaranteed to be written on k boards, so we need to check enough boards to leave only $k-1$ boards unchecked.) This might substantially increase the bandwidth requirements of the readers.

Whether one chooses the synchronized distributed web bulletin board or the unsynchronized distributed web bulletin board depends entirely on the nature of the application. If one requires a strict ordering of all messages to be maintained, one must use the synchronized board; this would apply to an auction, or audited discussion board. For an online petition, the ordering of the signatures is presumably not crucial, and the unsynchronized board would work well.

For many electronic voting systems, the ordering of messages is unimportant. Often what is required is simply a record of all encrypted votes, or some such.

In this context, either option would work, but the unsynchronized board might be more efficient.

4 Conclusion

In this paper, we have provided a way of implementing the append-only web bulletin board whose existence has been assumed in so much of the electronic voting literature.

We introduced the notion of *certified publishing*, in which a writer and a bulletin board are protected from false allegations of misconduct. We then introduced our scheme, and demonstrated that it satisfies the properties that one requires of an append-only web bulletin board.

Finally, in Section 3, we discussed how to distribute the board among a number of peers to make it robust in the face of system failure or deliberate misconduct.

Future work will focus on the distributed web bulletin board and on its security properties. We aim to define the liveness properties we would expect of the two flavours of distributed board, and then prove that they satisfy those properties.

4.1 The application to electronic voting

A robust web bulletin board and the issuing of encrypted receipts are vital components of verifiable electronic voting systems. When an encrypted receipt is submitted to the web bulletin board by a voting machine/scanner, the web bulletin board collective responds with a certificate indicating that the receipt has been received and published in the robust collective history. This certificate can be printed onto the voter's receipt, giving the individual further opportunities to audit the election. Furthermore, if the voter is able to verify, using the certificate, that the encrypted receipt has been correctly entered onto the web bulletin board, he need not check the receipt on the web bulletin board after the close of the election. This improves the security of the whole election: it means that the integrity of the election requires a smaller number of voters to check their receipts.

4.2 Acknowledgements

Warm thanks to Roger Peel and Zhe Xia for their comments on the scheme presented here.

References

[1] B. Adida and R. Rivest. Scratch & vote: self-contained paper-based cryptographic voting. *Proceedings of the fifth ACM workshop on Privacy in electronic society*, pages 29–40, 2006.

[2] R. Aditya, Lee B, C. Boyd, and E. Dawson. An efficient mixnet-based voting scheme providing receipt-freeness. *Proceedings of TrustBus'04*, pages 152–161, 2004. LNCS 3184.

[3] Roberto Araujo, Ricardo Filipe Custodio, and Jeroen van de Graaf. A verifiable voting protocol based on farnel. *Proceedings of Workshop On Trustworthy Elections (WOTE 2007)*, 2007.

[4] O. Baudron, P.-A. Fouque, D. Pointcheval, J. Stern, and G. Poupard. Practical multi-candidate election system. *Proceedings of the twentieth ACM Symposium on Principles of Distributed Computing (PODC'01)*, pages 274–283, 2001.

[5] J. Benaloh and D. Tuinstra. Receipt-free secret-ballot elections (extended abstract). *Proceedings of the twenty-sixth Symposium on Theory of Computing (STOC'94)*, pages 544–553, 1994.

[6] D. Chaum. Secret ballot receipts: true voter-verifiable elections. *IEEE: Security and Privacy Magazine*, 2(1):38–47, 2004.

[7] D. Chaum, P. Ryan, and S. Schneider. A practical voter-verifiable election scheme. *Proceedings of the tenth European Symposium on Research in Computer Science (ESORICS'05)*, pages 118–139, 2005. LNCS 3679.

[8] R. Cramer, R. Gennaro, and B. Schoenmakers. A secure and optimally efficient multi-authority election scheme. *Advances of Eurocrypt'97*, pages 103–118, 1997. LNCS 1233.

[9] T. ElGamal. A public key cryptosystem and a signature scheme based on discrete logarithms. *IEEE Transactions on IT*, 31(4):467–472, 1985.

[10] K. Fisher, R. Carback, and T. Sherman. Punchscan: Introduction and system definition of a high-integrity election system. In *PRE-PROCEEDINGS*, pages 19 – 29. IAVoSS Workshop On Trustworthy Elections, 2006.

[11] A. Fujioka, T. Okamoto, and K. Ohta. A practical secret voting scheme for large scale elections. *Advances of Auscrypt'92*, pages 244–251, 1992. LNCS 718.

[12] A. Juels, D. Catalano, and M. Jakobsson. Coercion-resistant electronic elections. *Proceedings of the 2005 ACM Workshop on Privacy in the Electronic Society*, pages 61–70, 2005.

[13] B. Lee, C. Boyd, E. Dawson, K. Kim, J. Yang, and S. Yoo. Providing receipt-freeness in mixnet-based voting protocols. *Proceedings of ICISC'03*, pages 245–258, 2003. LNCS 2971.

[14] C. A. Neff and J. Adler. Verifiable e-voting: indisputable electronic elections at polling places. *VoteHere Inc*, 2003.

[15] M. Ohkubo, F. Miura, M. Abe, A. Fujioka, and T. Okamoto. An improvement on a practical secret voting scheme. *Information Security'99*, pages 225–234, 1999. LNCS 1729.

[16] P. Paillier. Public-key cryptosystems based on discrete logarithms residues. *Advances of Eurocrypt'99*, pages 223–238, 1999. LNCS 1592.

[17] Punchscan. http://www.punchscan.org.

[18] R. Rivest. The threeballot voting system, 2006. http://crypto.csail.mit.edu/ rivest/Rivest-TheThreeBallotVotingSystem.pdf.

[19] B. Schneier. *Applied Cryptography: protocols, algorithms, and source code in C.* John Wiley, United States of America, second edition edition, 1996.