Electronic Communications of the EASST
Volume 66 (2013)

Proceedings of the Automated Verification of Critical Systems (AVoCS 2013)

Verifying a Mix Net in CSP

Efstathios Stathakidis, David M. Williams and James Heather

15 pages

Guest Editors: Steve Schneider, Helen Treharne
Managing Editors: Tiziana Margaria, Julia Padberg, Gabriele Taentzer
ECEASST Home Page: http://www.easst.org/eceasst/ ISSN 1863-2122

Verifying a Mix Net in CSP

Efstathios Stathakidis[1]*, **David M. Williams**[2]† **and James Heather**[1]

[1] {e.stathakidis, j.heather}@surrey.ac.uk
Department of Computing, University of Surrey, Guildford, UK
[2] d.m.williams@vu.nl
Theoretical Computer Science, VU University Amsterdam, The Netherlands

Abstract: A Mix Net is a cryptographic protocol that tries to unlink the correspondence between its inputs and its outputs. In this paper, we formally analyse a Mix Net using the process algebra CSP and its associated model checker FDR. The protocol that we verify removes the reliance on a Web Bulletin Board during the mixing process: rather than communicating via a Web Bulletin Board, the protocol allows the mix servers to communicate directly, exchanging signed messages and maintaining their own records of the messages they have received. Mix Net analyses in the literature are invariably focused on safety properties; important liveness properties, such as deadlock freedom, are wholly neglected. This is an unhappy omission, however, since a Mix Net that produces no results is of little use. Here we verify that the Mix Net is guaranteed to terminate, outputting a provably valid mix agreed upon by a majority of mix servers, under the assumption that a majority of them act according to the protocol.

Keywords: Mix Nets, formal methods, model-checking, CSP, FDR

1 Introduction

A *Mix Net* is a cryptographic protocol which conceals the correspondence between the initial vector of encrypted input values and the permuted vector of decrypted values given as output. No protocol participant should be able to link a single encrypted input to its specific corresponding decrypted output, although all participants must be assured that a bijective relationship between inputs and outputs exists. It was first introduced by Chaum [Cha81] for anonymous mail. Usually, a Mix Net consists of a number of mix servers that collectively execute a protocol; most recent Mix Nets rely on a public-key encryption scheme, such as ElGamal [Gam85], that allows re-encryption of ciphertexts. The inputs are submitted under the joint public key of the Mix Net, and each individual mix server receives a list of encrypted messages (ciphertexts), re-encrypts each of them, permutes the resulting vector and outputs the re-encrypted and re-ordered list to the Web Bulletin Board (WBB). The re-encryption and mixing is peformed by each mix server sequentially. To ensure the correctness of the operation to the other mix servers, each mix server produces a "proof of shuffle". The output of the last mix server may be decrypted by a threshold number of tellers and then posted on the WBB for public verification. Golle *et al.* [GJJS04] outline such a Mix Net.

* Author sponsored by the EPSRC under the Trustworthy Voting Systems (TVS) project.
† Author sponsored by the NWO under the Design and Analysis of Secure Distibuted Protocols (DASDiP) project.

Mix Nets have been proposed for use in real-life applications, including anonymous Web browsing and electronic cash payments, RFID tags and untraceable mail systems. Their main application is electronic-voting [Adi08, RBH+09, BCH+12], where they are used to ensure that a voter's vote cannot be tracked and revealed throughout the process, providing voter privacy and anonymity. Although their anonymity has been exhaustively analysed and proved, early Mix Nets were not fault-tolerant as a dishonest mix server could corrupt the execution in order to stop the whole process or result in an invalid or inconsistent output. In this work we are interested in both liveness and safety properties that a Mix Net should fulfil, namely robustness and privacy as described in Section 4.

Furthermore, one assumption often made in the Mix Net literature is the existence of a publicly verifiable and trusted site, called a Web Bulletin Board (WBB) [HL08, WG06, Wik04]. The mix servers communicate with each other via the WBB and the Mix Net achieves universal verifiability as anyone has read access to it and can verify the stored information. However, the bulletin board has proved notoriously slippery to construct in practice. Here, we replace this assumption with two others that are far easier to realise: first, that all communication is performed over authenticated channels; and secondly, the *honest majority* assumption, under which strictly more than half of the mix servers act according to the protocol. On this basis, we remove the need for a WBB during mixing, and verify a Mix Net that allows direct communication between the mix servers.

In what follows, we will retain some notion of a WBB for publication of the final mix data from each server. This is simply because the final result of the mix needs somehow to be published; obviously if there is no way at all of making something available permanently for public consumption then there is no way of effectively completing the mix. However, this is a much weaker assumption than that of its existence during mixing. The final publication problem can be solved by putting the mix data up, suitably signed, on various news organisations' web sites, for example, or releasing it on BitTorrent; but these mechanisms are not appropriate for live communication between mix servers during the mixing process. Essentially, the WBB that we use to publish the final mix data corresponds simply to an assumption that after the mix servers have done their work, the honest servers have a reliable way of getting the message 'out there'.

Our Contributions. Our primary contribution is the formal analysis of a Mix Net that performs mixing with no centralised trusted authority in place but instead enables direct communication between the mix servers using authenticated channels. Rather than posting to, and reading from, a central WBB that records a consistent record of all messages sent, each mix server maintains its own local perspective throughout the rounds of mixing, receiving signed messages broadcast to them by other mix servers and broadcasting their own signed messages to all others. Secondly, we verify that our proposed protocol meets its requirements using the CSP process algebra and the FDR model-checker. We analyse our protocol in the presence of a realistic intruder model, based on Roscoe and Goldsmith's perfect spy [RG97]. It is important to analyse Mix Nets' security properties before using them, as many constructions have been broken after they were introduced. Combined, our results show that our protocol guarantees to terminate and guarantees consensus among a majority of mix servers of the final chain of mixes in the presence of a minority of dishonest mix servers that do not faithfully follow the protocol.

Organisation. We begin, in Section 2, by discussing previous work on Mix Nets. In Section 3, we describe a subset of the CSP process algebra that is sufficient for constructing our model. We propose our protocol in Section 4 that removes the need of the WBB during mixing. We then construct a CSP model of our Mix Net in Section 5. Results of the formal analysis of the model are given in Section 6, and we conclude the paper in Section 7 with a discussion of possible future research directions.

2 Previous Work

In this section, we survey previous work in Mix Nets where liveness is of major concern. Robustness is the liveness property regarding successful termination in the presence of faulty mix servers. The first Mix Net introduced by Chaum [Cha81] is not robust, as in the case where one of the mix servers does not work, the execution stops and no output is obtained. Sako and Kilian [SK95] proposed Mix Net constructions that are not robust either: if at least one mix server stops responding, then the entire system stops without outputting a result. Jakobsson [Jak98] presented a practical Mix Net which was believed to be robust until Desmedt and Kurosawa [DK00] found an attack such that at least one malicious mix server can prevent the Mix Net from computing the correct result. In these approaches the mixing can not proceed if a single mix server is unavailable. Distributed consensus protocols, such as [PSL80], guarantee agreement on a vector of values that includes those of all honest nodes. Our requirements of Mix Nets need not be this strong: we require that some mix of threshold length be agreed by a majority of mix servers, although all honest mix servers' mixes need not be included.

Another weakness in the Mix Net literature is the assumption of the existence of a WBB that supervises the mixing process. Each mix server reads what is posted on the WBB, operates on what was read and posts back to the WBB. Most of the constructions are based on the strong assumption that the WBB is authenticated, tamper-proof and resistant to denial-of-service attacks. Wisktröm and Groth [WG06] proposed an adaptively secure Mix Net based on ideal functionalities for a WBB [Wik04]. The existence of such a publicly verifiable site is not realistic and practical in real-life applications, however, as it is a single point of failure. If it is unavailable, the entire process stops and the Mix Net does not complete and does not produce an output.

Although they are the building blocks for constructing secure real-life applications, Mix Nets have been proposed without formal analysis and automated verification. Wikström [Wik04] proved that his Verificatum Mix Net is secure in the universal composable symbolic model, but his analysis excludes constructions with proofs of shuffles. Moreover, his scheme does not satisfy the liveness property, as when a mix server raises a complaint about another's honesty, then the whole process stops and manual intervention is needed in order to exclude the faulty mix server.

Process algebras like π-calculus [AF01] have been used to analyse electronic voting schemes with very limited references to Mix Nets. Kremer *et al.* [KRS10] and Delaune *et al.* [DKR09] used the π-calculus language to model and analyse important voting properties like voter-privacy, coercion-resistance, receipt-freeness and verfiability in the presence of an ideal WBB-based Mix Net, and the use of CSP and its associated refinement checker FDR have proved successful in finding previously unknown flaws in security protocols [Low96, RS01]. We use the CSP/FDR approach to analyse the robustness of a Mix Net, the first such analysis for a Mix Net.

3 Preliminaries

Communicating Sequential Processes (CSP) is a process algebra used for verifying complex concurrent systems. The core of the CSP algebra is a *process*, which is described by the way it communicates with its environment. Processes proceed from one state to another by engaging in *events*. The *alphabet* of a process P, denoted αP, records the set of all visible events that this process may perform. The set of all possible events is denoted by Σ. In CSP, all the communication events are instantaneous and they happen only when both the processes and the environment agree on their occurrence. For a more detailed explanation of CSP we refer the reader to [Ros98].

STOP is the simplest CSP process, which does nothing. The process $a \rightarrow P$ is initially willing to communicate a and then behaves like P. $P \,\Box\, Q$ can act either as P or Q, the choice of which is in the hands of the environment. Replicated external choice replicates the choice over the set $\mathscr{A}$, and is denoted by $\Box_{x \in \mathscr{A}} P(x)$. By $P \parallel_{\mathscr{A}} Q$ we denote generalised parallel, which synchronises P and Q on events lying in the set $\mathscr{A}$. Alphabetised parallel is denoted by $P \,{}_{\alpha P}\|_{\alpha Q}\, Q$, and synchronises P and Q on events lying in the intersection of αP and αQ. We write $\|_{i \in I}\, [\alpha P]\, P(i)$ for the replicated alphabetised parallel composition of processes $P(i)$ indexed over I, where each $P(i)$ is allowed to perform events from αP and the processes are synchronised on the common events. In hiding, $P \setminus \mathscr{A}$, the internal events from $\mathscr{A}$ are hidden from the environment. In renaming, $[\![{}^{a}/_{b}]\!]$, the events b occurring in the process are replaced by the events a.

For two decades the Failures/Divergences Refinement (FDR) checker has been the principal tool for verifying properties of models expressed in CSP. FDR tests whether the CSP model of the system being analysed refines some specification of the system's desired behaviour, which is also written in CSP. We omit details of the datatypes and channel definitions in the CSP model of a Mix Net presented in Section 5, as they can be inferred from the data values ascribed to them; however, we shall provide definitions of certain sets and functions used throughout the model.

The set of all mix servers is denoted by $\mathscr{P}$ and is defined to be $\mathscr{H} \cup \mathscr{D}$, where $\mathscr{H}$ (resp. $\mathscr{D}$) denotes the set of all honest (resp. dishonest) mix servers. By $max(\mathscr{D})$, we denote the dishonest mix server with the highest identity. By $|\mathscr{A}|$, we denote the cardinality of a set $\mathscr{A}$, by $\mathbb{P}\mathscr{A}$ we denote the powerset function as applied to a set $\mathscr{A}$, and we use $\mathscr{A} \setminus \mathscr{B}$ to denote set difference, subtracting all elements of a set $\mathscr{B}$ from a set $\mathscr{A}$. Honest mix servers send broadcast messages to all other mix servers at once, whereas the intruder is able to send different messages to individual mix servers. Thus, it is useful to define a set $\mathscr{R} = \{X \in \mathbb{P}\mathscr{P} \mid |X| = N - 1 \, or \, |X| = 1\}$ of the possible message recipients, where N is the total number of mix servers.

By α, we denote an unmixed vector. By $M_j(m)$, we denote the vector m mixed using mix server j's secret permutaion value, where m is either the unmixed vector α or some mix thereof. By $\rho(M_j(m))$ we denote the zero knowledge proof that the operation producing $M_j(m)$ from m was performed as attested to. Messages, also referred to as chains, then have the form $S_j(M_j(m), \rho(M_j(m)), s)$, where s is its corresponding sub-chain. Chains are so called, as they are extended at each mixing stage, although a better analogy is of a Matryoshka nesting doll. The mix from a previously received chain (or α in the base case) is mixed, the mixing operation attested to and the whole of the previously received chain are all signed together, such that a proof of each operation in the chain is nested within each 'layer' of the message. The length of a message m is denoted by $\#m$ and is calculated by counting these layers of signatures, i.e.,

sub-chains. Messages cannot be longer than the total number of mix servers.

The set of all messages that can be feasibly sent and received in a protocol run is denoted by $\mathscr{M}$. Messages do not require every mix server to be included, signing some sub-chain within the chain, but the signatures throughout a chain are required to be in strictly increasing order. Moreover, it is not possible for a mix server to deduce how a vector of values has been mixed, even with knowledge of the values before and after mixing as well as the zero knowledge proof. An honest mix server will never sign a mix chain unless it has produced the final mix in the chain. This is not necessarily true of the dishonest mix servers. The message space of $\mathscr{M}$ takes these properties into account, omitting messages that could not feasibly be sent in a protocol run. The outer signatory of a message is verified using the corresponding public key; we use a function $outer(m)$ to return the signatory of a given message m. Likewise, $seq(m)$ returns the outer mix sequence of m. The set of all possible output mixes, $\mathscr{O}$, is taken to be $\{seq(m) \mid m \in \mathscr{M}\}$.

4 Our Proposed Mix Net

In this section we propose our Mix Net, which avoids use of the WBB for communication between mix servers during the mixing process. Our protocol is guaranteed to terminate in the presence of a dishonest minority. Upon termination of the mixing process, the mix servers post partial decryptions to the WBB and at least one valid chain of threshold length will be fully decrypted/agreed upon by a majority of mix servers. The primary difference from current Mix Nets found in the literature is that each mix server maintains their own 'local' record of the chain of mixes instead of maintaining a consistent 'global' record via the WBB.

4.1 Requirements

An honest mix server follows the protocol without deviating from it, produces correct shuffles and valid proofs. It sends the same message to all other mix servers using authenticated channels. Conversely, a dishonest mix server tries to disrupt the protocol such that it does not terminate, or otherwise, upon termination, causes a dispute among the honest mix servers. To try and achieve these goals, the intruder acts as a Byzantine faulty node under the limitation of a perfect cryptography assumption, i.e., the intruder can refuse to send messages, it need not send messages to all other mix servers, and can try to send messages that it can construct but that do not necessarily follow the protocol. The intruder may send different messages to different mix servers.

The output of a Mix Net should be a complete chain of provably valid mixes. A chain is considered complete if its length is strictly greater than $\frac{N}{2}$, where N is the number of servers, and each mix is a proven valid mix of the mix in its sub-chain. We consider the Mix Net robust if it always returns such output, regardless of the behaviour of the dishonest minority. It is conceivable that not every honest server has a mix represented in the final chain; it is a requirement, however, that at least one mix has been produced by an honest server to maintain privacy.

4.2 Protocol description

When the protocol starts, all mix servers are provided with the initial list of unmixed ciphertexts, α, as the initial candidate mix to be mixed. Each mix server has a unique identity between 1 and

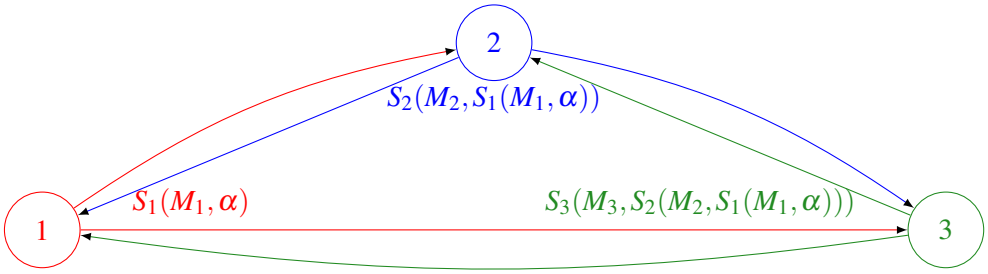

Figure 1: A Mix Net with three honest mix servers

the total number N of mix servers. Mix server 1 is the first *mixer* and all mix servers maintains a counter to record the identity of the current mix server. Initially, mix server 1 re-encrypts α using new randomness and shuffles using its own local and secret permutation, thus producing the mix $M_1(\alpha)$. To prove the correctness of the re-encryption and shuffling, it proves, in zero-knowledge, that $M_1(\alpha)$ was formed correctly from α, producing the proof $\rho(M_1(\alpha))$. It then signs those two values along with α, with its own signing key, to produce $S_1(M_1(\alpha), \rho(M_1(\alpha)), \alpha)$. We shall call such messages chains, where the length of this chain is said to be 1, as it has just one layer of mixing and signing.

The other mix servers acting as *checkers* wait to receive the first mixer's signed output or for a timeout to occur. Each checker extracts the message from inside the signature and checks the proofs against the latest mix. In later rounds they will also have to do this for the inner signatures and inner proofs. If any of the proofs do not check out, then the message is rejected and the checker waits for a message containing valid proofs to arrive or for a timeout to occur. If the message is valid, i.e., is a chain containing a complete sequence of valid proofs, then the length of the chain is calculated by counting the number of layers of signatures. If the length of the chain is equal to or exceeds the length of the checker's current candidate chain, the current candidate chain is updated to be the most recently received chain. Otherwise it remains unaltered. They increment the counter and, if it is their turn, they do the mixing.

Having received the message $S_1(M_1(\alpha), \rho(M_1(\alpha)), \alpha)$ and verified the signature and proof of mixing, the second mix server mixes $M_1(\alpha)$ to produce $M_2(M_1(\alpha))$ and must also produce the associated zero knowledge proof $\rho(M_2(M_1(\alpha)))$. He signs this to produce the message $S_2(M_2(M_1(\alpha)), \rho(M_2(M_1(\alpha))), S_1(M_1(\alpha), \rho(M_1(\alpha)), \alpha))$, which is a complete chain of length two. As chains get longer throughout the protocol, we abbreviate this to $S_2(M_2, S_1(M_1, \alpha))$.

Figure 1 illustrates the messages exchanged in a run of the protocol in the presence of three honest mix servers faithfully following the protocol. Following receipt of the first two messages, the third mix server acts as mixer and sends $S_3(M_3, S_2(M_2, S_1(M_1, \alpha)))$ to the other two mix servers. All the mix servers maintain a record of the valid chains that they sent or received during the protocol that are longer than $\frac{N}{2}$. Subsequently, they shall post all such chains, partially decrypted, onto the WBB. Hence, a polynomial number of different outputs will appear on the WBB. Any chain that has been partially decrypted by at least a majority of mix servers can be used as the final output, with preference given to the longest.

5 Modelling and Formal Analysis in CSP

In this section, we present the processes and specifications used to model and analyse our Mix Net. We describe the processes modelling the honest and dishonest mix servers, and then compose them into models to be checked for robustness and privacy. The complete script can be found at http://www.tvsproject.org/csp/mixnet.csp.

5.1 Honest mix servers

The behaviour of the honest mix servers can be split into four phases: checking before mixing; mixing; checking after mixing; and posting the results to the WBB. The first mixer will of course perform no checking before mixing and the final mixer will do no checking after mixing.

$$
\begin{array}{l}
CHK_1(me, curr, toBeMixed) = \\
\quad \textbf{if } curr < me \textbf{ then} \\
\qquad \left(\begin{array}{l} timeout \rightarrow CHK_1(me, curr+1, toBeMixed) \\ \\ \Box \quad \Box \\ \quad m \in \{m' \in \mathscr{M} \mid outer(m') = curr\} \end{array} \right. \\
\qquad\qquad \left. \begin{array}{l} recv.m \rightarrow \\ timeout \rightarrow \\ \textbf{if } \#m \geq \#toBeMixed \textbf{ then} \\ \qquad CHK_1(me, curr+1, m) \\ \textbf{else } CHK_1(me, curr+1, toBeMixed) \end{array} \right) \\
\quad \textbf{else } MIX(me, curr, toBeMixed)
\end{array}
$$

CHK_1 models the initial phase of checking before mixing. Using *toBeMixed*, the honest checker keeps track of the chain that it will use once it becomes mixer, and also records the identity of the current mixer, *curr*. The checkers are willing to receive any valid mix, represented in CHK_1 by the external choice over valid chains signed by the current mixer, but block receipt of invalid mixes, and mixes signed by a server who is not the current mixer. This abstracts away the behaviour of receiving a mix containing an invalid proof, or an incorrectly signed message. If a checker times out before (resp. after) receiving a valid chain, then *toBeMixed* remains the same (resp. is updated with the new chain if not shorter than the last) and *curr* is incremented.

$$
\begin{array}{l}
MIX(me, curr, toBeMixed) = \\
\left(\begin{array}{l} send.S_{me}(M_{me}, toBeMixed) \rightarrow \\ timeout \rightarrow \\ CHK_2(me, curr+1) \end{array} \right)
\end{array}
$$

$$
\begin{array}{l}
CHK_2(me, curr) = \\
\quad \textbf{if } curr \leq N \textbf{ then} \\
\qquad \left(\begin{array}{l} timeout \rightarrow CHK_2(me, curr+1) \\ \\ \Box \quad \Box \\ \quad m \in \{m' \in \mathscr{M} \mid outer(m') = curr\} \end{array} \right. \\
\qquad\qquad \left. \begin{array}{l} recv.m \rightarrow \\ timeout \rightarrow \\ CHK_2(me, curr+1) \end{array} \right) \\
\quad \textbf{else } done \rightarrow STOP
\end{array}
$$

If the mix server counter is its own, the mix server enters the mixing phase. It performs the mixing using *toBeMixed*, and sends a new chain to the other mix servers. Honest mix servers do not time out before sending a message (we assume that the timeout value is sufficiently large that an honest mix server will have time to complete its work and send the result out). Thereafter, the mix server need not keep track of *toBeMixed* as it will not need to perform any more mixing. The current mix server counter is incremented by one, and the mix server continues to be involved in checking other servers' mixes (that is, it continues on to the CHK_2 process).

In the checking after mixing phase, checkers can time out before or after receiving a valid chain, in a similar manner to the previous checking phase. Once *curr* exceeds the total number

of mix servers, the mixing protocol terminates as modelled by the *done* event. The mix server is now ready to post its results to the WBB. Each mix server must record all chains it sent or received during mixing, and post partial decryptions of those with length greater than $\frac{N}{2}$.

A particular mix could become known to a mix server in a number of ways. It could feasibly have arrived several times, in messages signed by different mix servers. Like the various learnable facts in Roscoe and Goldsmith's perfect spy [Ros98], a mix server starts off being ignorant of each possible mix, as modelled by *IGN*. Note that *IGN* is defined in terms of a mix, *sq*, and not a message. The mix can be *learnt*, after which it becomes known; thereafter it can be *said*. The mix server could also calculate the mix themselves from its *toBeMixed* value, modelled here as *say*, which can only occur in synchrony with the *MIX* process. Otherwise, the protocol terminates before the occurrence of a *learn* or *say* of this mix and the process deadlocks. Once known, a mix could potentially be learnt or said again in the *KNW* process, without changing the state. Following a *done* event, the mix can be posted (to the WBB), as long as the chain is over the threshold length, as defined in *PST*.

$$IGN(sq) = \left(\begin{array}{l} done \rightarrow STOP \\ \Box\ learn.sq \rightarrow KNW(sq) \\ \Box\ say.sq \rightarrow KNW(sq) \end{array}\right) \qquad KNW(sq) = \left(\begin{array}{l} done \rightarrow PST(sq) \\ \Box\ learn.sq \rightarrow KNW(sq) \\ \Box\ say.sq \rightarrow KNW(sq) \end{array}\right) \qquad PST(sq) = \left(\begin{array}{l} \textbf{if}\ \#sq > \frac{N}{2}\ \textbf{then} \\ post.sq \rightarrow PST(sq) \\ \textbf{else}\ STOP \end{array}\right)$$

To realise when a mix is received via a message, we rename each *learn.sq* to all possible *recv.m* in which the message would return the corresponding mix, and likewise we rename *say.sq* to all possible *send.m*. The renamings of all *IGN* processes, one for each possible mix, are composed in parallel, synchronising only on the one event their alphabets share, namely *done*. Subsequently all *send*, *recv* and *done* events are synchronised with the process $CHK_1(me, 1, \alpha)$, which initialises the local perspective of the mix server, *me*, with the current mix server counter set to the first mixer and the *toBeMixed* parameter set to the unmixed value α.

$$HON(me) = CHK_1(me, 1, \alpha) \left|\left| \underset{\{|send,recv,done|\}}{\left(\left|\left|_{sq \in \mathscr{O}}\right.\right.\right.} [\{done\}] IGN(sq) [\![\ {}^{recm.m,send.m}/_{learn.sq,say.sq} \ \mid \ _{m \in \mathscr{M}, seq(m)=sq} \ ]\!] \right)$$

A global perspective of the honest mix server does not just *send* and *recv* messages. Instead it is defined in terms of the connections a mix server shares with others. Hence, each *send* is renamed to an outgoing $comm.me.\mathscr{P}\backslash\{me\}$, distributing the sent message to all mix servers other than *me*. Likewise, each *recv* is renamed to an incoming *comm.x.Z*, where *x* is some mix server other than *me*, and *Z* is either the singleton set containing only *me* or the set of all mix servers other than *x*. Honest mix servers will send to everyone, whereas the dishonest mix server could choose to send different messages to different mix servers or choose to send messages to some mix servers but not others. The mix server does not know how the message arrived: he sees the protocol only in terms of the *HON* process. However, when analysing the system, we can see which messages were distributed to whom by synchronising the various *renHON* processes, which gives a view of the protocol from a networking perspective. The *post* event is renamed in a similar fashion such that groups of mix servers can synchronise on an *agree* event, modelling the threshold decryption.

$$\begin{aligned} renHON(me) = HON(me) [\![& \ {}^{comm.me.\mathscr{P}\backslash\{me\}.m}/_{send.m} \mid _{m \in \mathscr{M}} \ ]\!] \\ [\![& \ {}^{comm.x.Z.m}/_{recv.m} \mid _{m \in \mathscr{M}, x \in \mathscr{P}\backslash\{me\}, Z \in \mathscr{R}, x \notin Z, me \in Z} \ ]\!] \\ [\![& \ {}^{agree.Z.sq}/_{post.sq} \mid _{sq \in \mathscr{O}, Z \in \mathbb{P}\mathscr{P}, |Z| > \frac{N}{2}, \#sq > \frac{N}{2}} \ ]\!] \end{aligned}$$

Our final models are constructed using alphabetised parallel composition. As things stand, if both are honest, mix server 1 will communicate with mix server 2 each time 1 sends a mix to

all mix servers via $comm.1.\mathscr{P}\backslash\{1\}$. However, although mix server 2 is willing to receive this message, it is also willing to communicate via $comm.1.\{2\}$ as, locally, it is unable to distinguish the two types of communication. The alphabetised parallel composition will ensure that no events $comm.1.\{2\}$ ever occur in the model, as long as we define appropriate alphabets. The alphabet of mix server *me* is defined below.

$$\begin{aligned}\alpha HON(me) = &\{comm.me.Z.m \mid_{m\in\mathscr{M},Z\in\mathscr{R},me\notin Z}\} \cup \{comm.y.Z.m \mid_{m\in\mathscr{M},y\in\mathscr{P}\backslash\{me\},Z\in\mathscr{R},me\in Z,y\notin Z}\} \\ &\cup\{agree.Z.m \mid_{m\in\mathscr{M},Z\in\mathbb{P}\mathscr{P},|Z|>\frac{N}{2},me\in Z,\#m>\frac{N}{2}}\} \cup \{timeout, done\}\end{aligned}$$

We can already compose a system in which all mix servers are honest, that is $\mathscr{D} = \emptyset$.

$$ALLHON = \Big\|_{i\in\mathscr{H}} [\alpha HON(i)]\, renHON(i)$$

To perform a more rigorous analysis we must of course include an intruder model.

5.2 Dishonest mix servers

We use the Roscoe and Goldsmith's perfect *Spy* process [RG97], more specifically its encoding in [Ros98], as the basis of the intruder (dishonest mix server) behaviour, although we use a different renaming as we are not considering a Dolev Yao attacker [DY83]. The *Spy* can receive messages sent to it over *learn*, infer events based on messages received and its initial knowledge and *say* messages that it has inferred that may be accepted by the honest mix servers.

Upon receipt of a message, the intruder can sometimes make a number of independent inferences in any order. Roscoe and Goldsmith's perfect spy avoids refinement checking an unnecessarily large state space caused by this combinatorial explosion. Whenever an inference is made, the intruder does not lose the ability to perform any action that could be done prior to the inference. This observation allows one to force the intruder immediately to perform every possible inference in some arbitrary order before taking any other action. This causes no observable difference in the intruder's behaviour, but avoids otherwise troublesome state space explosion.

Furthermore, deductions need only be made for the misbehaving agent's set of learnable facts. That is, the intruder can never learn a fact that it knew initially or that cannot be deduced from its initial knowledge and all the messages it could hear. Roscoe and Goldsmith's *Spy* elegantly omits inferences of such facts. Each deduction function operates on a subset of the set of facts conforming to a particular type. Two deduction rules are sufficient for our purpose. The first allows a signed fact to be learnt from knowledge of the fact and the signing key, and to deduce the fact from knowledge of the signed fact and the signatory's public key. The second allows a sub-chain to be deduced from a chain. Remember $S_2(M_2, S_1(M_1, \alpha))$ abbreviates $S_2(M_2(M_1(\alpha)), \rho(M_2(M_1(\alpha))), S_1(M_1(\alpha), \rho(M_1(\alpha)), \alpha))$. From knowledge of this message and mix server 2's public key the intruder can extract a sequence of three values: the mix, the proof and the sub-chain. Even with knowledge of the mixed values before and after this round of mixing, as well as the zero knowledge proof, the mixer's secret permutation value is not revealed. Hence, in our abstract representation of this message, the intruder cannot deduce the value M_2, but can deduce the sub-chain included in this chain, or the chain as signed by the intruder, or can extend the chain with an extra mix as signed by the intruder.

$$\begin{aligned}&deductions1(X) = \{(\{m, sk(i)\}, S_i(m)), (\{S_i(m), dual(sk(i))\}, m) \mid S_i(m) \in X\} \\ &deductions2(X) = \{(\{m, m'\}, (m, m')), (\{(m, m')\}, m') \mid (m, m') \in X\}\end{aligned}$$

The only other change we make to *Spy* is to redefine the intruder's initial knowledge. This consists of all mix server identities, the secret permutation values of all dishonest mix servers, all mix servers' public keys for verifying signatures, and the initial unmixed vector of values.

$$Known' = \mathscr{P} \cup \{M_i, sk(i) \mid i \in \mathscr{D}\} \cup \{pk(i) \mid i \in \mathscr{P}\} \cup \{\alpha\}$$

The *Spy* process has in its alphabet *learn* allowing information to be received and *say* allowing messages to be sent. Internally, *infer* actions allow messages to be inferred such that additional *say* actions may become possible, i.e., additional messages can be sent once they have been deduced by the intruder. In order to compose *Spy* with the honest mix server processes *renHON*, we must rename *Spy* such that it appropriately communicates with the other mix servers.

$$renSPY = Spy[\![\; comm.x.\mathscr{P}\backslash\{x\}.m, say.m, comm.max(\mathscr{D}).\{x\}.m \,/\, learn.m, say.m, say.m \mid m \in \mathscr{M}, x \in \mathscr{H} \;]\!]$$
$$[\![\; agree.Z.seq(m) \,/\, say.m \mid m \in \mathscr{M}, Z \in \mathbb{P}\mathscr{P}, |Z| > \tfrac{N}{2}, Z \backslash \mathscr{H} \neq \emptyset, \#m > \tfrac{N}{2} \;]\!]$$

Each *say* event is renamed to $comm.max(\mathscr{D}).\{x\}$. Locally, the mix servers are unable to distinguish where messages originated from, and so it does not matter which of the identities of the dishonest mix servers the intruder uses to send messages. As such, we reduce the number of transitions by enabling the intruder to use only the highest numbered dishonest mix server, without limiting the intruder's capability to attack the protocol. The intruder can send different messages to different mix servers, so each message is sent to an individual mix server—that is, to the singleton set $\{x\}$, where x is the identity of some honest mix server. If the intruder wishes, like the honest servers, to send the same message to all mix servers, he simply unicasts the same message to each honest mix server. The intruder learns all messages sent to any of the dishonest mix servers' identities, so it is not necessary for, say, dishonest mix server 1 to send messages to dishonest mix server 2. As the intruder hears all messages sent to any of the mix servers identities, and all honest mix servers send to all mix servers other than themselves, *learn* events are renamed to $comm.x.\mathscr{P}\backslash\{x\}$ where x is again the identity of some honest mix server. We may wish our intruder to contribute to the group agreements of the final output, so the *say* events should also be renamed to $agree.Z$ events, where Z is any set of servers that contains the identity of at least one dishonest mix server.

Our use of the alphabetised composition operator ensures that the mix servers are synchronised on appropriate events.

$$\alpha SPY = \{comm.y.\mathscr{P}\backslash\{y\}.m \mid_{m \in \mathscr{M}, y \in \mathscr{H}}\} \cup \{comm.x.Z.m \mid_{m \in \mathscr{M}, x \in \mathscr{D}, Z \in \mathscr{R}, x \notin Z}\}$$
$$\cup \{agree.Z.m \mid_{m \in \mathscr{M}, Z \in \mathbb{P}\mathscr{P}, |Z| > \frac{N}{2}, Z \backslash \mathscr{H} \neq \emptyset, \#m > \frac{N}{2}}\}$$

Now that we have constructed processes that capture the behaviour of the honest and dishonest mix servers, we can compose them together for analysis.

5.3 Requirements and assertions

We shall construct three systems for the analysis of the protocol. Each system involves a set of honest mix servers that follow the protocol faithfully, and may include an intruder that has control over a subset of mix servers, receiving any message sent to these mix servers.

Initially, we analyse *ALLHON*, in which all mix servers faithfully follow the protocol. In this case, we expect all servers to agree on the longest of the mix chains that is sent on the network, received in the final round. Each mix server partially decrypts this chain and publishes the result.

If more than half of the mix servers post a partial decryption of the same chain to the WBB, then this decrypted chain of mixes will be considered as the final output of the mix. This output is modelled in our system as a synchronous *agree* event among a majority of mix servers agreeing on the output message. We wish to check that it is guaranteed that some *agree* event occurs.

$RBST = agreement \rightarrow RBST$

$RBST \sqsubseteq_{FD} (ALLHON \setminus \Sigma\{|\ agree\ |\})[\![^{agreement}/_{agree.X.sq}\ |_{X\in\mathbb{P}\mathscr{P},sq\in\mathscr{O}}\ ]\!]$

Using the Failures/Divergences model (*FD*), we check that it is not possible for our model to diverge or deadlock before the occurrence of the *agreement* event. As all events other than *agree* have been hidden and all *agree* events renamed to *agreement*, this checks that no loops or deadlocks exist prior to the performance of some *agree*. Thus, this captures our robustness requirement that agreement on some complete and valid mix is guaranteed.

Once we introduce the intruder model, it will be possible that some chains have been received by some mix servers but not by others. We check the same robustness requirement of this model.

$$RBST \sqsubseteq_{FD}((ALLHON\ {}_{\alpha ALLHON}\|_{\alpha SPY}\ renSPY) \setminus \Sigma\{agree\})[\![^{agreement}/_{agree.X.sq}\ |_{X\in\mathbb{P}\mathscr{P},sq\in\mathscr{O}}\ ]\!]$$
$$where\ \alpha ALLHON = \bigcup(\{\alpha HON(i) \mid i \in \mathscr{H}\})$$

We perform the same check on a model in which the intruder outputs no chain, in an attempt to stop any chain being agreed upon by a threshold of mix servers.

$$RBST \sqsubseteq_{FD}((ALLHON\ {}_{\alpha ALLHON}\|_{\alpha SPY}\ (renSPY \underset{\{|agree|\}}{\|}\ STOP)) \setminus \Sigma\{agree\})[\![^{agreement}/_{agree.X.sq}\ |_{X\in\mathbb{P}\mathscr{P},sq\in\mathscr{O}}\ ]\!]$$

We also require a message to be private as described in Section 4.1. For this we check that no majority of mix servers can agree on a non-private mix.

$$PRIV = \Box_{\substack{X\in\mathbb{P}\mathscr{P}\\ y\in\{sq\in\mathscr{O}|private(sq),\#sq>\frac{N}{2}\}}} agree.X.y \rightarrow PRIV$$

$$PRIV \sqsubseteq_{T} (ALLHON\ {}_{\alpha ALLHON}\|_{\alpha SPY}\ renSPY) \setminus \Sigma\{agree\}$$

These assertions are sufficient for guaranteeing that our protocol terminates and guarantees consensus among a majority of mix servers of a complete chain of mixes in the presence of fewer than $\frac{N}{2}$ mix servers that do not faithfully follow the protocol. We check these assertions in FDR, considering all possible combinations of honest and dishonest mix servers, with up to five mix servers in total, and a majority of them being honest. In all cases FDR confirmed that our models satisfy the specified requirements.

6 Results of Analysis

In this section, we verify the protocol against liveness and safety properties. We start with all mix servers being honest. When we check the protocol with five honest mix servers, as expected, we find that they all agree on the same chain of valid mixes. Moreover, the privacy of the mixes is guaranteed. The trace below illustrates this behaviour. Each of the traces discussed in this section were obtained via simulation of the models in ProBE, a CSP animator that allows a user to explore how a process behaves. In this first trace, each honest mix server times out when sending out a new signed message.

$\langle comm.1.\{2,3,4,5\}.S_1(M_1,\alpha),\ timeout,$
$comm.5.\{1,3,4,5\}.S_2(M_2,S_1(M_1,\alpha)),\ timeout,$
$comm.3.\{1,2,4,5\}.S_3(M_3,S_2(M_2,S_1(M_1,\alpha))),\ timeout,$
$comm.4.\{1,2,3,5\}.S_4(M_4,S_3(M_3,S_2(M_2,S_1(M_1,\alpha)))),\ timeout,$
$comm.5.\{1,2,3,4\}.S_5(M_5,S_4(M_4,S_3(M_3,S_2(M_2,S_1(M_1,\alpha))))),\ timeout,$
$done,\ agree.\{1,2,3,4,5\}.M_5(M_4(M_3(M_2(M_1(\alpha)))))\rangle$

In fact, after the done event, any subset of mix servers larger than three can agree on any of the three candidate mixes, $M_5(M_4(M_3(M_2(M_1(\alpha)))))$, $M_4(M_3(M_2(M_1(\alpha))))$ or $M_3(M_2(M_1(\alpha)))$. In this case $M_5(M_4(M_3(M_2(M_1(\alpha)))))$ will be considered the final output as it is the longest mix agreed upon by a majority of mix servers.

More interesting traces result from checking the protocol with three honest and two dishonest mix servers. Analysing the protocol with the two last mix servers being dishonest leads to a larger state space because (i) the intruder learns to say more throughout the execution, and (ii) the honest mix servers accept longer messages and therefore many more messages throughout time, so the intruder has lots of choices about what to send in the last two rounds.

The intruder can send different messages to different mix servers or can follow the protocol faithfully. In the latter, we end up with the same trace as above. When the intruder behaves dishonestly, we guarantee that a sufficient number of mix servers agree on the same output and at least one of them is honest, which means that the privacy is satisfied. In the case where the dishonest mix servers simply time out, all of them or any subset of length four or three would agree on the chain produced by the honest mix servers, e.g., $agree.\{1,2,3\}.M_3(M_2(M_1(\alpha)))$.

A more curious behaviour is illustrated below. Here, the intruder acting as dishonest mix server 2 times out without sending any message. Later, acting as 5, the intruder sends a valid but short message to an honest mix server, and sends two valid but differing messages to the other honest mix servers. Honest mix servers 1, 3 and 4 follow the protocol faithfully: 1 constructs the initial mix, 3 mixes what it received directly from 1 following the timing out of 2, and 4 mixes what it received from 3. Dishonest 5 then proceeds to send the valid but differing mixes to the honest mix servers.

$\langle comm.1.\{2,3,4,5\}.S_1(M_1,\alpha),\ timeout,$
$timeout,$
$comm.3.\{1,2,4,5\}.S_3(M_3,S_1(M_1,\alpha)),\ timeout,$
$comm.4.\{1,2,3,5\}.S_4(M_4,S_3(M_3,S_1(M_1,\alpha))),\ timeout,$
$comm.5.\{1\}.S_5(M_5,\alpha),$
$comm.5.\{3\}.S_5(M_2,S_4(M_4,S_3(M_3,S_1(M_1,\alpha)))),$
$comm.5.\{4\}.S_5(M_5,S_4(M_4,S_3(M_3,S_1(M_1,\alpha)))),\ timeout,$
$done,\ agree.\{1,3,4\}.M_4(M_3(M_1(\alpha)))\rangle$

If the dishonest mix servers refrain from posting any partial decryptions then, as illustrated in the trace, the only possible output is a chain of valid mixes mixed only by the honest mix servers. If the dishonest mix servers are willing to post partial decryptions, then it is possible that other mixes will be agreed upon as the final output. Dishonest 2 and 5 are able to agree with honest 3 (resp. 4) on the mix $M_2(M_4(M_3(M_1(\alpha))))$ (resp. $M_5(M_4(M_3(M_1(\alpha)))))$. As 2 received a mix that was shorter than a mix previously received, the only threshold length mix he is willing to decrypt is $M_4(M_3(M_1(\alpha)))$. In any case the privacy is maintained, as each of these possible outputs include mixing by all three honest mix servers.

A more interesting behaviour occurs when the dishonest mix servers swap valid mixes constructed by honest mix servers with valid mixes generated by dishonest mix servers. The reader should note that the dishonest 2 can swap a valid mix by the first mix server with a valid mix generated by himself and send it to all honest mix servers. Honest 3 mixes and signs what it received from the second mix server and forward it to the others; honest 4 then does likewise. On the other hand, dishonest 5 replaces what it received from honest 4 with his own valid mix and sends the corresponding complete and valid chain to the other mix servers.

$\langle comm.1.\{2,3,4,5\}.S_1(M_1,\alpha),\ timeout,$
$comm.5.\{1\}.S_2(M_2,\alpha),$
$comm.5.\{3\}.S_2(M_2,\alpha),$
$comm.5.\{4\}.S_2(M_2,\alpha),\ timeout,$
$comm.3.\{1,2,4,5\}.S_3(M_3,S_2(M_2,\alpha)),\ timeout,$
$comm.4.\{1,2,3,5\}.S_4(M_4,S_3(M_3,S_2(M_2,\alpha))),\ timeout,$
$comm.5.\{1\}.S_5(M_5,S_3(M_3,S_2(M_2,\alpha))),$
$comm.5.\{3\}.S_5(M_5,S_3(M_3,S_2(M_2,\alpha))),$
$comm.5.\{4\}.S_5(M_5,S_3(M_3,S_2(M_2,\alpha))),\ timeout,$
$done,\ agree.\{1,2,3,4,5\}.M_5(M_3(M_2(\alpha)))\rangle$

In this case, as in the other, three or more mix servers agree upon mixes of threshold length, i.e., length three or more. The best the intruders can do is to swap valid mixes, such that the agreed output $M_5(M_3(M_2(\alpha)))$ excludes the mixes of honest 1 and 4. Honest 3's mix is still guaranteed to be a mix within the agreed upon chain, maintaining privacy. It is also possible that an agreement is made on $M_4(M_3(M_2(\alpha)))$ as all mix servers also have this valid mix, which is equal in length to $M_5(M_3(M_2(\alpha)))$. Either, or both, can be agreed to be the final output of the Mix Net as both mixes decrypt to the same vector of values, only ordered differently. Moreover, both are of threshold length include a mix by at least one honest mix server.

All these traces arise from our specification of the protocol and might not have been appreciated without this formal analysis. However, none of them breaks the requirements. The best the intruder could do was to replace some but not all of the honest mix servers' mixes with valid mixes of his own.

7 Conclusion and Future Directions

For the first time, we have conducted a formal analysis of a Mix Net, using CSP/FDR. We showed how to remove the need of a trusted authority during the mixing phase, and instead introduced direct communication between the mix servers. Our analysis demonstrates that our Mix Net is guaranteed to terminate, and output a provably valid mix agreed upon by a majority of mix servers, as long as a majority of them act according to the protocol. Moreover, at least one honest mix server's mix operation is guaranteed to be included in the output chain of mixes.

In our model, we included an intruder based on Roscoe and Goldsmith's perfect spy that is able to control some minority of mix servers. The dishonest mix servers can collaborate by sharing knowledge between them. We have shown that our system remains free of deadlocks and infinite loops, and is guaranteed to output a chain of valid mixes of length at least equal to the number of honest mix servers. We included three different intruder models and verified the protocol against them. In all cases we proved that its liveness and safety properties hold. Thanks to the data-independence results by Lazic and Roscoe [LR99], a correctness result on a Mix Net with x messages applies to Mix Nets with an arbitrary number of messages.

The number of mix servers we have used in this paper is typical for applications like electronic voting. In Victoria [BCH+12], the election commission will be running the elections with five mix servers, under the assumption that at least four of them are honest. Our analysis covers this real-world case and more.

One future goal is to analyse the protocol under a stronger intruder model. To introduce a

Dolev-Yao attacker that has full control of the network [DY83], and that could intercept and block every message, would clearly violate robustness. It is therefore of interest to consider to what extent the assumptions made about the threat environment can be relaxed whilst still maintaining the properties we have checked. For example, it may be pertinent to replace the current synchronous communication between the honest mix servers with asynchronous communication over resilient channels in the manner proposed in [WRF12]. Here, the intruder can intercept a message sent between the honest mix servers and delay it from arriving, but cannot indefinitely block the message's receipt, as long as the recipient is always willing to receive it. There remains outstanding work to scale the approach proposed in [WRF12] to cover our models, which have state space and alphabets that are currently too large to be feasibly checked in this manner.

Acknowledgements: We would like to thank Chris Culnane and Steve Schneider, University of Surrey, and Wan Fokkink, VU University Amsterdam, for their pertinent comments. We also thank the insightful remarks of the anonymous reviewers.

Bibliography

[Adi08] B. Adida. Helios: Web-based open-audit voting. In *Proceedings of the 17th USENIX Security Symposium (Security '08)*. 2008.

[AF01] M. Abadi, C. Fournet. Mobile values, new names, and secure communication. In Hankin and Schmidt (eds.), *POPL*. Pp. 104–115. ACM, 2001.

[BCH+12] C. Burton, C. Culnane, J. Heather, T. Peacock, P. Y. A. Ryan, S. Schneider, S. Srinivasan, V. Teague, R. Wen, Z. Xia. A Supervised Verifiable Voting Protocol for the Victorian Electoral Commission. In Kripp et al. (eds.), *Electronic Voting*. LNI 205, pp. 81–94. GI, 2012.

[Cha81] D. Chaum. Untraceable Electronic Mail, Return Addresses, and Digital Pseudonyms. *Commun. ACM* 24(2):84–88, 1981.

[DK00] Y. Desmedt, K. Kurosawa. How to Break a Practical MIX and Design a New One. In Preneel (ed.), *EUROCRYPT*. LNCS 1807, pp. 557–572. Springer, 2000.

[DKR09] S. Delaune, S. Kremer, M. Ryan. Verifying privacy-type properties of electronic voting protocols. *Journal of Computer Security* 17(4):435–487, 2009.

[DY83] D. Dolev, A. C.-C. Yao. On the security of public key protocols. *IEEE Transactions on Information Theory* 29(2):198–207, 1983.

[Gam85] T. E. Gamal. A public key cryptosystem and a signature scheme based on discrete logarithms. *IEEE Transactions on Information Theory* 31(4):469–472, 1985.

[GJJS04] P. Golle, M. Jakobsson, A. Juels, P. F. Syverson. Universal Re-encryption for Mixnets. In Okamoto (ed.), *CT-RSA*. LNCS 2964, pp. 163–178. Springer, 2004.

[HL08] J. Heather, D. Lundin. The Append-Only Web Bulletin Board. In Degano et al. (eds.), *Formal Aspects in Security and Trust*. LNCS 5491, pp. 242–256. Springer, 2008.

[Jak98] M. Jakobsson. A Practical Mix. In *Advances in Cryptology – EuroCrypt '98*. Pp. 448–461. Springer-Verlag, London, UK, 1998.

[KRS10] S. Kremer, M. Ryan, B. Smyth. Election Verifiability in Electronic Voting Protocols. In Gritzalis et al. (eds.), *ESORICS*. LNCS 6345, pp. 389–404. Springer, 2010.

[Low96] G. Lowe. Breaking and Fixing the Needham-Schroeder Public-Key Protocol Using FDR. In Margaria and Steffen (eds.), *TACAS*. Lecture Notes in Computer Science 1055, pp. 147–166. Springer, 1996.

[LR99] R. Lazic, B. Roscoe. Data Independence with Generalised Predicate Symbols. In Arabnia (ed.), *PDPTA*. Pp. 319–326. CSREA Press, 1999.

[PSL80] M. C. Pease, R. E. Shostak, L. Lamport. Reaching Agreement in the Presence of Faults. *J. ACM* 27(2):228–234, 1980.

[RBH+09] P. Y. A. Ryan, D. Bismark, J. Heather, S. Schneider, Z. Xia. Prêt à Voter: a voter-verifiable voting system. *IEEE Transactions on Information Forensics and Security* 4(4):662–673, 2009.

[RG97] A. W. Roscoe, M. Goldsmith. The perfect spy for model-checking crypto-protocols. In *Proceedings of DIMACS workshop on the design and formal verification of crypto-protocols*. 1997.

[Ros98] A. W. Roscoe. *The theory and practice of concurrency*. Prentice Hall, 1998.

[RS01] P. Y. A. Ryan, S. A. Schneider. *Modelling and analysis of security protocols*. Addison-Wesley-Longman, 2001.

[SK95] K. Sako, J. Kilian. Receipt-Free Mix-Type Voting Scheme - A Practical Solution to the Implementation of a Voting Booth. In Guillou and Quisquater (eds.), *EUROCRYPT*. LNCS 921, pp. 393–403. Springer, 1995.

[WG06] D. Wikström, J. Groth. An Adaptively Secure Mix-Net Without Erasures. In Bugliesi et al. (eds.), *ICALP (2)*. LNCS 4052, pp. 276–287. Springer, 2006.

[Wik04] D. Wikström. A Universally Composable Mix-Net. In Naor (ed.), *TCC*. LNCS 2951, pp. 317–335. Springer, 2004.

[WRF12] D. M. Williams, J. de Ruiter, W. Fokkink. Model Checking under Fairness in ProB and Its Application to Fair Exchange Protocols. In Roychoudhury and D'Souza (eds.), *ICTAC*. Lecture Notes in Computer Science 7521, pp. 168–182. Springer, 2012.