Privacy supporting cloud computing: ConfiChair, a case study

Myrto Arapinis, Sergiu Bursuc, and Mark Ryan

School of Computer Science, University of Birmingham
{m.d.arapinis,s.bursuc,m.d.ryan}@cs.bham.ac.uk

Abstract. Cloud computing means entrusting data to information systems that are managed by external parties on remote servers, in the "cloud", raising new privacy and confidentiality concerns. We propose a general technique for designing cloud services that allows the cloud to see only encrypted data, while still allowing it to perform data-dependent computations. The technique is based on key translations and mixes in web browsers.

We focus on the particular cloud computing application of conference management. We identify the specific security and privacy risks that existing systems like EasyChair and EDAS pose, and address them with a protocol underlying ConfiChair, a novel cloud-based conference management system that offers strong security and privacy guarantees.

In ConfiChair, authors, reviewers, and the conference chair interact through their browsers with the cloud, to perform the usual tasks of uploading and downloading papers and reviews. In contrast with current systems, in ConfiChair the cloud provider does not have access to the content of papers and reviews and the scores given by reviewers, and moreover is unable to link authors with reviewers of their paper.

We express the ConfiChair protocol and its properties in the language of ProVerif, and prove that it does provide the intended properties.

1 Introduction

Cloud computing means entrusting data to information systems that are managed by external parties on remote servers, "in the cloud." Cloud-based storage (such as Dropbox), on-line documents (such as Google docs), and customer-relationship management systems (such as salesforce.com) are familiar examples. Cloud computing raises privacy and confidentiality concerns because the service provider has access to all the data, and could accidentally or deliberately disclose it.

Cloud-based conference management systems such as EasyChair or the Editor's Assistant (EDAS) represent a particularly interesting example [32]. For example, EasyChair currently hosts more than 3000 conferences per year, and therefore contains a vast quantity of sensitive data about the authoring and reviewing performance of tens of thousands of researchers world-wide. This data is in the possession of the EasyChair administrators, and could be accidentally or deliberately disclosed. A conference chair that is thinking of hosting her conference on a cloud-based conference system therefore faces a dilemma: if she uses the system, she adds to this mountain of data and the risks associated with it; if she doesn't use the system, she deprives herself of the advantages of a readily-available, well-engineered system that already has user accounts for the majority of participants in her conference (authors, PC members, and reviewers).

Note that the data confidentiality issue concerns the cloud conference system administrator (who administrates the system for all conferences), not the conference chair (who is concerned with a single conference). The conference system administrator has access to all the data on the system, across thousands of conferences and tens of thousands of authors and reviewers. An individual conference chair, on the other hand, has access to the data only for the particular conference of which she is chair. Moreover, an author or reviewer that chooses to participate in the conference can be assumed to be willing to trust the chair (for if he didn't, he would not participate); but there is no reason to assume that he trusts or even knows the conference management system provider.

In this paper, we identify a set of confidentiality requirements for conference management and propose ConfiChair, a cloud-based conference management protocol that supports them. The confidentiality guarantees ensure that no-one has access to conference data, beyond the access that is explicitly granted to them by their participation in the conference. In particular, this is true about the cloud provider and managers. ConfiChair is loosely modelled on EasyChair or EDAS, but with the additional security guarantees. We describe a protocol in which authors, reviewers and the conference chair interact through their web browsers with the cloud-based management system, to perform the usual tasks of uploading and downloading papers and reviews. The cloud is responsible for fine-grained routing of information, in order to ensure that the right agents are equipped with the right data to perform their task. It is also responsible for enforcing access control, for example concerning conflicts of interest and to ensure that a reviewer doesn't see other reviews of a paper before writing her own. However, all the sensitive data is seen by the cloud only in encrypted form.

For brevity, we use the term "cloud" to include all roles that are not an explicit part of the conference management; that includes the conference management system administrator, the cloud service provider, the network administrator, etc. The security properties that our system provides may be summarised as follows.

- **Secrecy of papers, reviews and scores.** The cloud does not have access to the content of papers or reviews, or the numerical scores given by reviewers to papers.
- **Unlinkability of author-reviewer.** The cloud *does* have access to the names of authors and the names of reviewers. This access is required in order to route information correctly, to enforce access control, and to allow a logged-in researcher to see all his data in a unified way. However, the cloud does not have the ability to tell if a particular author was reviewed by a particular reviewer. In particular, for each encryption of each review or score held by the cloud, either the cloud does not know which author it applies to, or does not know which reviewer submitted it.

Summary of contribution

1. We identify a set of requirements for cloud-based conference management systems, notably privacy requirements such as secrecy and unlinkability.
2. We propose ConfiChair, a conference management protocol that provides the usual functionalities while offering strong privacy guaranties.
3. We show the usability of ConfiChair by providing a prototype implementation. We demonstrate that using ConfiChair is as easy and useful as using EasyChair, except for the requirement of two copy-paste operations (one performed by authors, one performed by reviewers).
4. We formalise the required privacy properties and automatically prove them with ProVerif.

Applicability of the ideas Cloud-based services are being adopted widely throughout business. The following examples raise similar security concerns to those of conference management:

- Customer relationship management systems (such as salesforce.com);
- Cloud-hosted recruitment process services, in which applicants, referees, recruiters and employers interact to process job applications;
- Cloud-based finance and accounting services;
- Social networks, in which users share posts and status updates without wishing that data to be mined by the cloud provider for profiling purposes.

We believe our technique of browser-based key translation and mixnets is readily applicable to these examples too.

2 Description of the problem and related work

Our problem is determined by three conflicting sets of requirements, namely functionality, privacy and usability. As we show below, there is much existing work related to our paper, but it can not be used to solve our problem either because of its complexity, or because of its different perspectives on privacy, or because it does not achieve the required balance between privacy and functionality.

2.1 Desired properties and threat model

Functional requirements. As previously mentioned, we use the term "cloud" to refer to the cloud service provider, conference management system and its administrators, and the network. The responsibilities of the cloud are:

- To collect and store data relevant to the conference, including names of reviewers and authors, papers, reviews and scores.
- To enforce access control in respect of conflicts of interest and ensuring that reviewers see other reviews of a paper only after they have submitted their own.
- To manage the information flow of the conference: from authors, to conference chair, to reviewers and back.
- To notify the authors of the acceptance decision about their papers.

Privacy requirements. We require that the cloud does not know

- the content of submitted papers,
- the content of submitted reviews,
- the scores attributed to submitted papers.

Further, when data is necessarily known to the cloud in order that it can fulfil the functional requirements, we require what we call *unlinkability* property: the cloud is unable to link

- authors to reviewers of their papers

Threat model. It is reasonable to trust the cloud to execute the specified functional requirements. Indeed, an incorrect functionality would be detected in the long run and the users would simply move into another cloud. On the other hand, the cloud may try to violate privacy without affecting functionality, in a way that cannot readily be detected. ConfiChair is designed to remove this possibility. Obviously, there are inherent limitations on any protocol's ability to achieve this. For instance, if the cloud provider was invited to participate as a PC member or a chair, then he necessarily would have access to privileged information. Consequently, the privacy requirements are expected to hold in our threat model only for conferences in which the cloud provider does not participate, except as provider of the cloud service or as author of a paper.

We assume that users are running uncorrupted browsers on malware-free machines. The HTML, Java, and Javascript code that they download is also assumed to be obtained from a trustworthy source and properly authenticated (*e.g.* by digital signatures).

Usability requirements. The system should be as easy to use as present day conference management systems, such as EasyChair, iChair, OpenConf or HotCRP. The cost of security should not be unreasonable waiting time (e.g. for encryption, data download), or software installation on the client-side (e.g. a browser should be sufficient), or complex key management (e.g. public key infrastructure), etc. We discuss more about usability in section 4 which describes our prototype implementation.

2.2 Related work

Generic solutions. Much work has been done that highlights the confidentiality and security risks that are inherent in cloud computing (e.g., [13] includes an overview), and there is now a conference series devoted to that topic [18]. Although the issue is well-known, the solutions described are mostly based on legislative and procedural approaches. Some generic technological solutions have appeared in the literature. The first one uses trusted hardware tokens [33], in which some trusted hardware performs the computations (such as key translations) that involve sensitive data. Solutions based on trusted hardware tokens may work, but appear to have significant scalability issues, and require much more research. Other papers advise designing cloud services to avoid having to store private data, and include measures to limit privacy loss [28].

Fully-homomorphic encryption (FHE) has been suggested as another generic solution to cloud-computing security. FHE is the idea that data processing can be done through the encryption, and has recently been shown to be possible in theory [21]. However, the range of functionality that can be provided through the encryption is not completely general. For example, one cannot extract from a list the items satisfying a given predicate, but one can return a list of encrypted truth values that indicate the items that satisfy the predicate, which is less useful. It is not clear to what extent FHE could alleviate the requirement to perform the browser-side computations of ConfiChair. Moreover, FHE is currently woefully inefficient in practice, and can only be considered usable in very specialised circumstances.

Data confidentiality and access control. Many works consider the problem of restricting the access of data in the cloud to authorised users only. For example, attribute-based encryption [7, 5] allows fine-grained control over what groups of users are allowed to decrypt a piece of data. A different example is work that aims to identify functionally encryptable data, i.e. data that can be encrypted while preserving the functionality of a system [30]. Such systems, and others, aim to guarantee that the cloud, or unauthorised third parties, do not access sensitive data. Our problem requires a different perspective: how to design systems that allow the cloud, i.e. the intruder, to handle sensitive data, but at the same time ensure that sensitive data value links between them are not revealed.

Unlinkability. In many applications it is important that links between participants, data, or transactions are kept hidden. In RFID-based systems [15] or in privacy enhancing identity management systems [17] for example, an important requirement is that two transactions of a same agent should not be linkable in order to prevent users from being tracked or profiled. Another exemplar application that requires unlinkability is electronic voting: a voter must not be linked to the vote that he has cast [19]. Moreover, like scores or identities in our case study, a vote is at the same time functional (to be counted) and sensitive (to be private). Voting systems achieve unlinkability by relying either on mix nets [25, 24], or on restricted versions of homomorphic encryption that allow the addition of plaintexts [3, 6]. Our proposed protocol also relies on mixing, showing how that idea can be adapted to new application areas.

Other systems identify applications where the cloud can be provided with "fake" data without affecting functionality [22]. In that case, privacy of "real" data may be preserved, without the cloud being able to detect the substitution. That is a stronger property than what we aim for, and at the same time the solution proposed in [22] is restricted to very specific applications. In particular, a conference management system can not function correctly with "fake" data provided to the cloud.

Conference management. There has been work exposing particular issues with conference management systems, related to data secrecy, integrity and access control [26, 31]. These are also important concerns, but that are quite orthogonal to ours, where we are interested in system design for ensuring unlinkability properties. More importantly, none of these works considers our threat model, where the attacker is the cloud.

3 The protocol

3.1 Description

The protocol is informally described in Figures 1-3 on the following pages. Some details, such as different tags for messages in each phase of the conference, are left out, but the detailed formal definition is given in Appendix A. The main idea of the protocol is to use a symmetric key K_{Conf} that is shared among the members of the programme committee. This key will be used to encrypt sensitive data before uploading it to the cloud. However, the cloud needs access to some sensitive data, like the reviewers of a paper, in order to implement the functional requirements of the protocol. To reveal that data to the cloud, without compromising privacy, our protocol makes use of the fact that different types of data are needed by the cloud at different phases of the conference. Thus, in transitioning from one phase to another, the conference chair can hide the links between authors and reviewers. He does so by performing a random mix on the data he needs to send to

the cloud before moving to the next phase. Each conference has a public key, that authors use to encrypt symmetric keys, that in turn serve to encrypt papers.

Notation As we just explained, the privacy of participants relies on the chair performing random mixes of the data he sends to the cloud. This is specified in Figures 1-3 by representing the databases $\mathrm{DB}^r_{\mathrm{Keys}}$, $\mathrm{DB}^r_{\mathrm{notf}}$ as random permutations (denoted by by $\leftarrow_{\mathcal{R}}$) of sets rather than as lists, *i.e.* with no order on their entries.

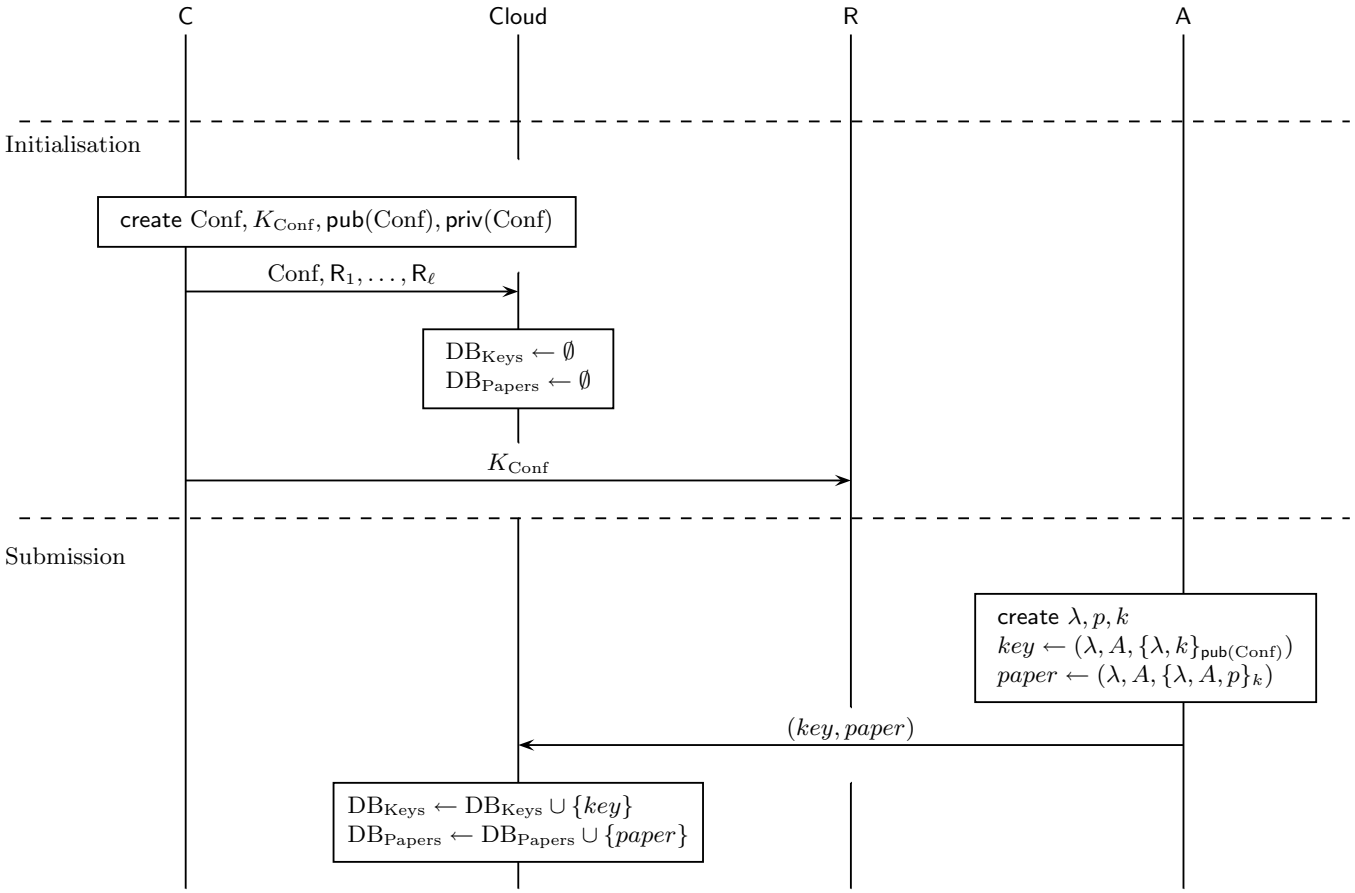
In the description of the ConfiChair protocol in Figures 1-3, we haven't included a biding phase. Although this phase is of great practical importance, it is conceptually similar to reviewing and discussion phases, and can be handled in a similar way. Indeed, provding a biding phase in a way that would preserve the users' privacy would also rely on the chair performing a reencryption mix on the papers before sending them to the reviewers through the cloud.

3.2 Discussion

Distribution of the reviewing symmetric key The privacy properties of our protocol rely on the sharing of a symmetric key K_{Conf} among the members of the programme committee in such a way that the cloud does not get hold of K_{Conf}. Here we suggest a few possible solutions in the context of our application, reflecting different trade-offs between security and usability. Our protocol is independent of which of the three solutions is adopted:
Public keys. Each reviewer may be expected to have a public key. Then, the symmetric key can be encrypted with each of the chosen reviewer's public key and uploaded to the cloud. The distribution can be made more flexible and efficient by relying on key distribution protocols like [11]. An important issue in this setting is the authentication of public keys of reviewers invited to participate in the conference. This may be done either relying on a hierarchical certification model such as *PKI* or, what is more probable in the case of conference management, on a distributed web of trust, such as that of *PGP*.
Token. In this solution, each reviewer generates a symmetric key k_{R} and uploads $\{k_{\mathsf{R}}\}_{\mathsf{pub}(\mathrm{Conf})}$ to the cloud. Then, the chair sends $\{K_{\mathrm{Conf}}\}_{k_{\mathsf{R}}}$ to the reviewer using a channel that is outside the control of the cloud. He does this by checking the reviewer's authenticated email address and sends $\{K_{\mathrm{Conf}}\}_{k_{\mathsf{R}}}$ to that address. The reviewer then decrypts this token to obtain K_{Conf}. In this case, even if the cloud has access later to a reviewer's email, it cannot compromise the privacy properties that our protocol ensures.
Email. If we assume that email infrastructure is not in the control of the cloud service provider that hosts ConfiChair (as is most probably the case in conference management), the key K_{Conf} could be sent to reviewers directly by email. In that case, if the email of a reviewer is compromised later on, its privacy for the conference Conf is also compromised. Note that it is only the key K_{Conf} that must be sent by email, all the rest of the protocol being executed in the cloud.

Computation in the cloud We stress that non-trivial computation takes place in the cloud, namely routing of messages, and optionally collection of statistics. It is essential for usability and take-up of the proposed system that these computations are done by the cloud. The difficulties in designing the protocol thus lie in releasing the necessary information for the cloud to perform these operations without compromising the users of our system's privacy. In particular, the link between the sender of a message (*e.g.* the author of a paper) and the end receiver of this message (*e.g.* the reviewer of this paper) should remain private and this although it is the cloud that is responsible for routing messages.

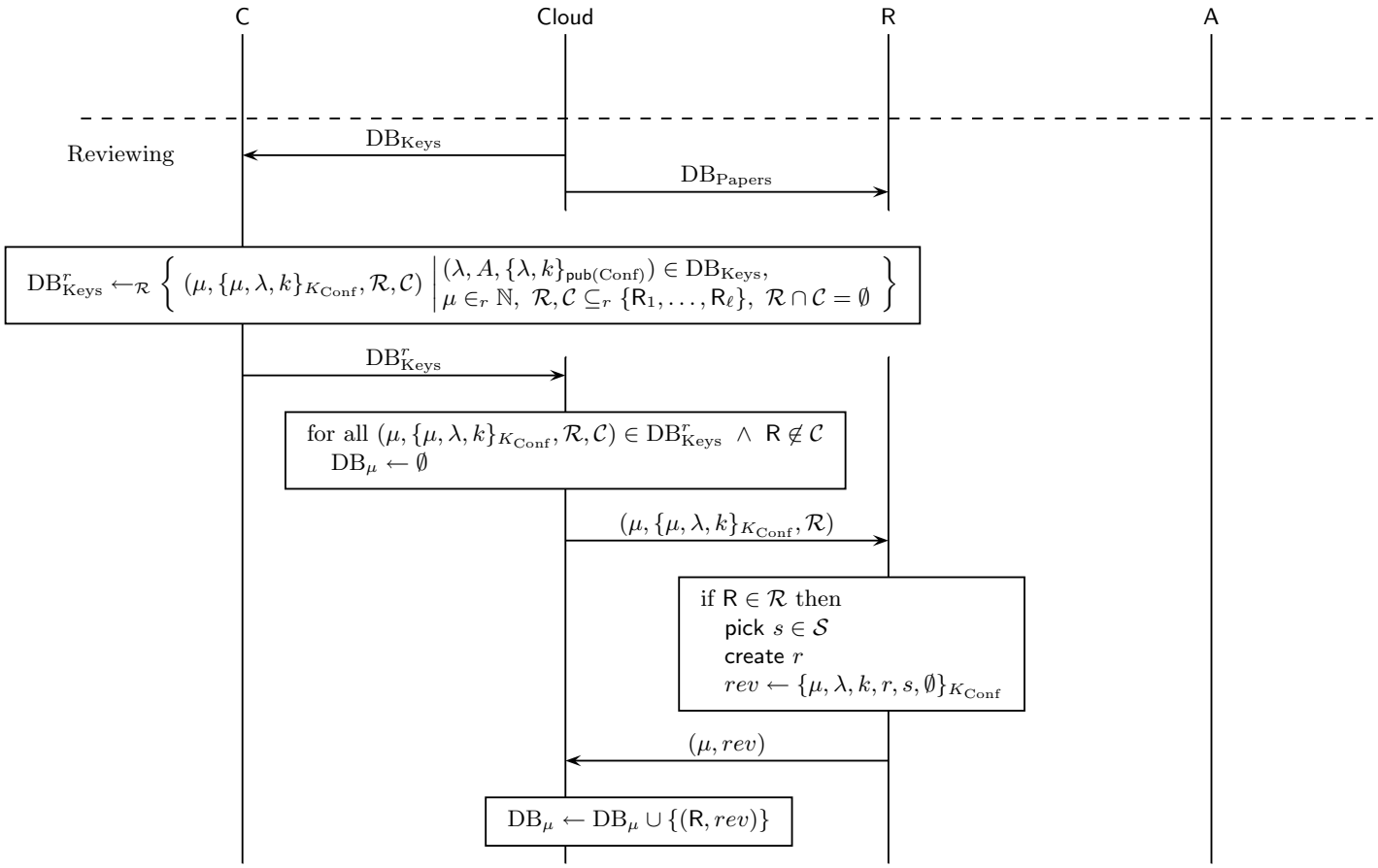
Optionally, the protocol can be extended to allow the cloud to collect statisticts or other anonymised data about the conference, its authors, papers, and reviewers. This can be achieved by adding code which extracts this information during the manipulations performed by the chair's browser. For example, along with the computation of $\mathrm{DB}^r_{\mathrm{notf}}$ in Figure 3.1, the chair could also compute the average score $as_\mu = (s_1 + \cdots + s_{n_\mu})/n_\mu$ for each paper and return the vector $(as_\mu)_\mu$



Initialisation. The conference chair C generates the symmetric key K_{Conf}, a public key pub(Conf) and a corresponding private key priv(Conf). The symmetric key is then shared among the reviewers in a way that does not reveal it to the cloud (see section 3.2). Then C requests from the cloud the creation of the conference named Conf, sending along the names of the chosen reviewers $\mathsf{R}_1, \ldots, \mathsf{R}_\ell$ for the programme committee.

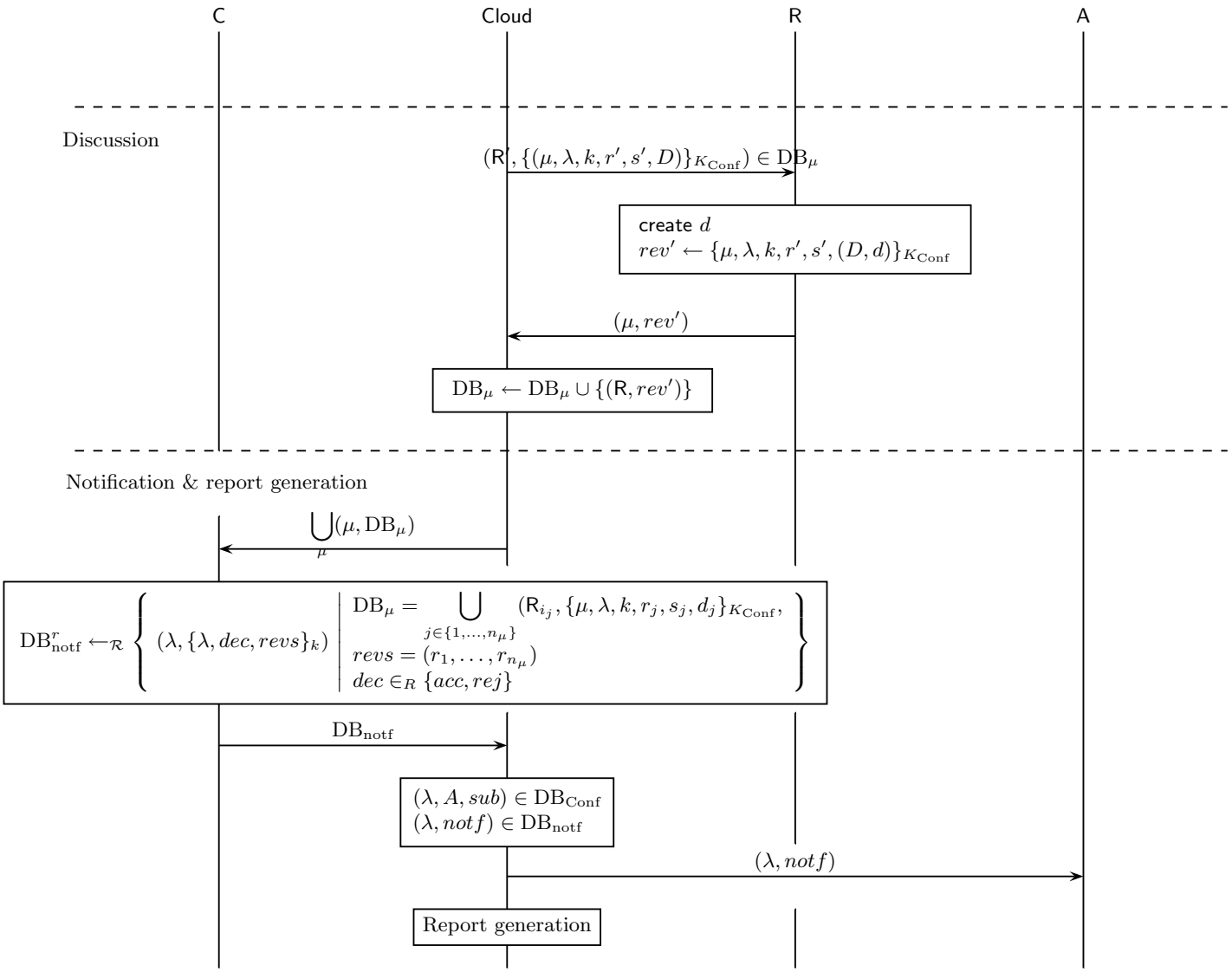
Submission. An author A creates a paper p and a symmetric key k. He uploads to the cloud p encrypted with k and k encrypted with pub(Conf). An identifier λ is used to refer to this encrypted submission. The first role of the key k is to provide a symmetric key for the encryption of papers. The second role of k will be to encrypt the reviews assigned to p, for the notification that will be sent through the cloud back to the author. The cloud creates two corresponding databases: one with encrypted submission keys and one with encrypted papers.

Fig. 1. ConfiChair: initialisation and submission phases



Reviewing. The chair downloads the database with encrypted keys, decrypts them using the private key priv(Conf) of the conference and encrypts them back with the shared symmetric key K_{Conf}. A new identifier μ is introduced for each paper. C also assigns the reviewers $\mathcal{R}$ to review the paper corresponding to μ, and declares the conflicts $\mathcal{C}$ restricting the set of reviewers that are allowed to see the data concerning μ. Finally, he mixes the elements in $\mathrm{DB}^r_{\mathrm{Keys}}$ before sending it to the cloud. The cloud filters the submissions according to these choices and sends them to reviewers.
The reviewers download the database with papers and can decrypt the papers for which they hold the keys. They read the papers, and for the papers they have been assigned to review (i.e. $R \in \mathcal{R}$), they write reviews, pick scores from S, and upload them in encrypted form back to the cloud. Note that the cloud is told to what identifier μ this encrypted review refers to. This allows the cloud to manage the data flow, without being able to link μ with λ, and hence the reviewer with the author.

Fig. 2. ConfiChair: reviewing phase



Discussion. The reviews of each paper are submitted to the programme committee members (except for the conflicting reviewers) for discussion. Each reviewer can read a submitted review and the ongoing discussion D and add a comment d to it. Reviewers upload their comments together with the review to which they refer to in encrypted form back to the cloud.

Notification. After reviewing, ranking and possible discussions, the chair of the conference makes a decision and, for each paper, creates a notification including the decision and the reviews. This notification is encrypted with the corresponding symmetric key k that was created by the author during the submission phase. The encrypted notification is uploaded to the cloud, who is also given in unencrypted form the identifier λ that refers to this submission. Again, this allows the cloud to manage the information flow, without compromising the privacy of the authors. Henceforth, the cloud is able to route the notification messages to the corresponding authors.

Fig. 3. ConfiChair: discussion and notification phases

to the cloud. (Such optional features must be carefully designed to avoid weakening the security properties, and are not considered in our formal model in Secrion 5.)

Efficiency and usability It may seem that there is a considerable amount of work to be done by the chair, especially in the transition between phases. As we discuss in the next section, this does not have to be evident to the chair. Our experiments with our prototype show that the browser can transparently execute the protocol.

4 Implementation

The ideal implementation of our protocol would look and feel very similar to existing cloud-based conference management systems such as OpenConf, EDAS and EasyChair, and should require no additional software beyond a web browser.

We constructed a prototype implementation [29], in order to discover any potential problems with a practical implementation and to find how much time and memory such a system may require, both on server-side and on client-side.

Overview. We implemented the ConfiChair prototype so that only a browser is necessary for participating as an author, a reviewer, or a chair. Overall, our prototype of ConfiChair feels very similar to current web-based management systems. A user of the system can perform his usual tasks by simply clicking a few buttons.

For example, to submit a paper an author logs to his ConfiChair account, selects the link for the conference to which he wants to submit, clicks the *new submission* button, selects the PDF file of his paper and clicks the *submit* button to complete his submission. All the key generation and the secure storing, as well as the encryption dictated by our protocol is transparently performed by the browser. The only aspect not currently performed by the browser is the retrieval of the conference public-key $\mathsf{pub}(\mathrm{Conf})$; this key must be manually input by the author (by copy-paste from the call for papers for example).

Similarily, the chair of a conference wanting to create a ConfiChair page for his conference Conf, loggins to his ConfiChair account and clicks the *create new conference* button. His browser will transparently, generate and securily store the keys K_{Conf}, $\mathsf{pub}(\mathrm{Conf})$, and $\mathsf{priv}(\mathrm{Conf})$.

Performance. The system is expected to handle hundreds of papers without overhead on the chair. In particular, browser-side re-encryption and mixing while transitioning between phases should not take more than a few minutes. From that perspective, the results of our experiments with the prototype implementation are promising. They are presented in figure 4. Also, experiments with any number of random files can be easily re-run on [29]. The tables show the waiting time for a corresponding number of papers when transiting between phases: always within a few minutes. The submissions in our experiments are PDF files of scientific papers, thus reflecting a real-case situation.

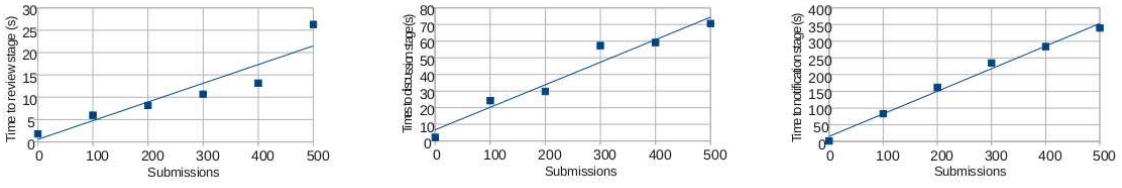


Fig. 4. Performance evaluation. The time taken for transitioning to the review stage is about 25s for 500 papers. The times for the other two transitions are about 70s and 350s.

Transparency of key management. To hide the complexity of the encryption keys from the user, these are managed and retrieved by the browser transparently when logging to ConfiChair. The login procedure implemented relies on each user having an identity id and a secret password psw_{id} from which the browser derives two keys: the ConfiChair account key $\mathsf{Kdf}_1(psw_{id})$ to authenticate the user to the the cloud provider, and a second key $\mathsf{Kdf}_2(psw_{id})$ used to encrypt the key purse of the user. This key purse contains the set of keys generated by the browser in previous accesses to the ConfiChair system, for example submission keys if the user has used ConfiChair as an author in the past, or conference keys if he has used it as a programme committee member.

When submitting a paper, the author's browser generates a symmetric key k which it uses to automatically encrypt the paper before sending it to the cloud. This key k is in turn added to the key purse of the user, which is uploaded encrypted with $\mathsf{Kdf}_2(psw_{authorid})$ to the cloud. To the submitter, this does not look like anything other than a normal file upload. Similarly, when the chair moves the conference to the review stage, it appears to be just like clicking on a normal link, since the chair's browser has already retrieved from the cloud the chair's key purse, and decrypted it with $\mathsf{Kdf}_2(psw_{chairid})$, and can then transparently decrypt and reencrypt the submissions according to the protocol.

In this way, the only key that needs to be securely backed up by a user id is his ConfiChair password psw_{id}. All the other keys are stored in encrypted form in the cloud, and retrieved when needed by his browser.

Currently, the authors need to copy and paste from the call for papers the public key of the conference $\mathsf{pub}(\text{Conf})$ to which they want to submit, and the reviewers need to copy and paste from their email the shared-key of the conference $K(\text{Conf})$ for which they are reviewers.

Future improvements In contrast to the ideal system that we envisage, our prototype requires the use of a Java plug-in for the users' browsers, since a Java applet is used to provide cryptographic routines. These routines are called from the JavaScript code using LiveConnect. An alternative to Java would be to use HTML5 which, unlike previous versions of HTML, provides the necessary features to implement ConfiChair, such as the possibility to program on-the-fly stream encryption and decryption. However, as the experiences of [3, 4, 23] suggest, Java applets are not necessarily an impediment in the usability and the take-up of the system. Moreover, while the use of the Java plug-in may look unattractive to some, it presents the following two advantages over HTML5:

- HTML5 is not yet widely adopted. Only the Chrome browser currently supports all the necessary features of HTML5 that an implementation of ConfiChair would require.
- In order for the user to trust the code that their web browser runs, the code should be reviewed, certified and signed by one or more trusted parties. The user's web browser would then verify the certificates without the user's intervention. Currently Java applets are the only way to achieve this.

All these implementation platform related issues will be further investigated in the future, for a real implementation and deployment of the ConfiChair protocol.

5 Formal model and verification

It is difficult to ascertain whether or not a cryptographic protocol satisfies its security requirements. Numerous deployed protocols have subsequently been found to be flawed, *e.g.* the Needham-Schroeder public-key protocol [27], the public-key Kerberos protocol [14], the PKCS#11 API [12], or the BAC protocol in e-Passports [16]. In this section, we formally show that ConfiChair does satisfy the announced security properties. The formal verification of the protocol has been done automatically using the ProVerif tool [8, 10]. The ProVerif specification of the ConfiChair protocol is available online [29]. The verification requires a rigorous description of the protocol in the ProVerif calculus as well as formal definitions of the desired properties, each discussed in detail in the following section.

5.1 The process calculus

The ProVerif calculus [8, 10] is a language for modelling distributed systems and their interactions. It is a dialect of the applied pi calculus [2]. In this section, we briefly review the basic ideas and concepts of the ProVerif calculus.

Terms The calculus assumes an infinite set of *names*, $a, b, c, \ldots$, an infinite set of *variables*, $x, y, z, \ldots$ and a finite *signature* Σ, that is, a finite set of *function symbols* each with an associated arity. Function symbols are divided in two categories, namely *constructors* and *destructors*. Constructors are used for building messages from other messages, while destructors are used for analysing messages and obtaining parts of the messages they are applied to. Names and variables are messages. A new message M may be built by applying a constructor $f \in \Sigma$, to names, variables and other messages, $M_1, \ldots, M_n$, and denoted as usual $f(M_1, \ldots, M_n)$. A term evaluation D is built by applying any function symbol $g \in \Sigma$ (constructor or destructor) to variables, messages or term evaluations, $D_1, \ldots, D_n$, denoted $g(D_1, \ldots, D_n)$. The semantics of a destructor g of arity n is given by a finite set of rewrite rules of the form $g(M_1, \ldots, M_n) \to M_0$, where $M_0, M_1, \ldots, M_n$ are messages that only contain constructors and variables. Constructors and destructors are used to model cryptographic primitives such as *shared-key* or *public-key encryption*. The ProVerif calculus uses tuples of messages $(M_1, \ldots, M_n)$, keeping the obvious projection rules implicit.

In the following, and for the purpose of modelling the ConfiChair protocol presented in section 3, we will consider the signature

$$\Sigma = \{\mathsf{senc}_{/3}, \mathsf{sdec}_{/2}, \mathsf{pub}_{/1}, \mathsf{aenc}_{/3}, \mathsf{adec}_{/2}, \mathsf{subm}_{/0}, \mathsf{initrv}_{/0}, \mathsf{revw}_{/0}, \mathsf{dsc}_{/0}, \mathsf{ntf}_{/0}, \mathsf{one}_{/0}, \mathsf{two}_{/0}\}$$

where senc (*resp.* aenc) is a constructor of arity 3 that models the randomised shared-key (*resp.* randomised public-key) encryption primitive, sdec (*resp.* adec) is the corresponding destructor of arity 2, and pub is a constructor of arity 1 that models the public key associated to the private key given in argument. The signature also contains the constants initrv, revw, dsc, and ntf corresponding to the tags used to label messages originating from different phases of the protocol; and the constants one and two representing the possible scores for papers. The semantics of the two destructors is given by the two following rules

$$\mathsf{sdec}(x, \mathsf{senc}(x, y, z)) \to z$$
$$\mathsf{adec}(x, \mathsf{aenc}(\mathsf{pub}(x), y, z)) \to z$$

We model the probabilistic shared-key encryption of the message m with the key k by $\mathsf{senc}(k, r, m)$, where the r is fresh for every encryption; and the probabilistic public-key encryption of the message m with the pubic key corresponding to the secret key k by $\mathsf{aenc}(\mathsf{pub}(k), r, m)$.

We will write $D \Downarrow M$ if the term evaluation D can be reduced to the message M by applying some destructor rules. For example, if we consider the following term evaluation E and message N

$$E = \mathsf{senc}(k, r, \mathsf{sdec}(k', \mathsf{senc}(k', r', s)))$$
$$N = \mathsf{senc}(k, r, s),$$

by application of the first rewrite rule given above, we have $E \Downarrow N$.

Processes Processes are built according to the grammar given below, where M is a message, D is a term evaluation, n is a name, and c is a channel name.

$P, Q, R ::=$	processes
0	null process
$P \mid Q$	parallel composition
$!P$	replication
$\text{new}\, n; P$	name restriction
$\text{let}\ M = D\ \text{in}\ P\ \text{else}\ Q$	term evaluation

$\text{in}(c, M); P$	message input
$\text{out}(c, M); P$	message output

Replication handles the creation of an unbounded number of instances of a process. The process let $M = D$ in P else Q tries to evaluate D and matches the result with M; if this succeeds, then the variables in M are instantiated accordingly and P is executed; otherwise Q is executed. We will omit the else branch of a let when the process Q is 0. Names that are introduced by a new construct are *bound* in the subsequent process, and they represent the creation of fresh data. Variables that are introduced in the term M of an input or of a let construct are *bound* in the subsequent process, and they represent the reception or computation of fresh data. Names and variables that are not bound are called *free*. We denote by $\text{fn}(P)$, respectively $\text{fv}(P)$, the free names, respectively free variables, that occur in P.

Notation. A process definition P will sometimes be denoted by $P(\vec{v})$, where $\vec{v}$ is a vector of free variables that occur in P and that can be seen as parameters for the process P. Then we will abbreviate the process let $\vec{v} = \vec{w}$ in P simply by $P(\vec{w})$, and we will say that $P(\vec{w})$ is an instance of $P(\vec{v})$.

Example 1. The process A models the authors' part of the ConfiChair protocol.

$$
\begin{array}{lll}
A & \stackrel{\text{def}}{=} & \text{new}\, ida; !A'(ida) \\
A'(xida) & \stackrel{\text{def}}{=} & \text{new}\, p;\ \text{new}\, k;\ A''(xida, p, k) \\
A''(yida, yp, yk) & \stackrel{\text{def}}{=} & \text{new}\, l;\ \text{new}\, r1;\ \text{new}\, r2; \\
 & & \text{in}(cpbk, xpbk); \\
 & & \text{let}\ k\_subm = (l, ida, \mathsf{aenc}(xpbk, r1, (\mathsf{subm}, l, k)))\ \text{in}\ (\\
 & & \text{let}\ p\_subm = (l, ida, \mathsf{senc}(k, r2, (l, ida, pap)))\ \text{in}\ (\\
 & & \text{out}(c, (k\_subm, p\_subm)); \\
 & & \text{in}(c, xn)))
\end{array}
$$

An author with identity ida can submit many times to many different conferences ($!A'(ida)$). For each submission he generates the paper p and the submission key k, he chooses the conference he wants to submit to, fetches the corresponding public ($\text{in}(cpbk, xpbk)$), and generates the identifier l (corresponding to the λ in the diagrams of Section 3). He then builds the submission message ((k_subm, p_subm)) as described in the diagrams of Section 3, and sends his submission to the cloud on the public channel c. He finally waits for the notification ($\text{in}(c, xn)$).

Altogether, the ConfiChair protocol can be fully modelled by the process

$$CC \stackrel{\text{def}}{=} \text{new}\, cshk;\ \text{new}\, cpbk;\ (!C \mid !R \mid !A)$$

The subprocesses C, R, and A model the behaviour of a conference chair, a reviewer, and an author respectively. A is fully detailed above, and C and R are detailed in Appendix A. We consider a general system CC with an unbounded number of chairs, reviewers, and authors. In CC, $cshk$ is the private channel (discussed in the first paragraph of Section 3.2) on which the shared-keys of conferences are sent to reviewers. The channel $cpbk$ is an authenticated channel from which the authors can access the public key of a conference in order to submit a paper. Although this channel is restricted to model that the public keys of conferences should be authenticated, the chair (as detailed in Appendix A) also publishes on the public channel c the public key of his conference, for anyone including the attacker to submit papers to it.

Semantics. The possible actions of the environment are captured by *evaluation contexts*. A *context* is a process with a hole. We denote by $C[A]$ the process obtained by filling $C[\_]$'s hole with the process A. An *evaluation context* is a context whose hole is not under a replication, a conditional, an input, or an output.

We define the operational semantics of the process calculus by the means of two relations: structural equivalence and internal reductions. *Structural equivalence* ($\equiv$) is the smallest equivalence

relation on processes closed under α-conversion of bound names and bound variables, application of evaluation contexts and of standard rules (see [10] for full definition) that capture the associativity, the commutativity and the interplay of $|$ and ν.

Internal reduction ($\rightarrow$) is the smallest relation closed under structural equivalence, application of evaluation contexts and such that

$$\overline{c}\langle M\rangle.P \mid c(x).Q \rightarrow P \mid Q\{^{M}/_{x}\}$$
$$\text{let } M = D \text{ in } P \text{ else } Q \rightarrow P\sigma, \text{ if } D \Downarrow N \ \&\ \sigma = \mu(M, N)$$
$$\text{let } M = D \text{ in } P \text{ else } Q \rightarrow Q, \text{ otherwise}$$

where we let $\mu(M, N)$ denote the substitution that matches M with N, if such a substitution exists. We write $\rightarrow^*$ for an arbitrary (possibly zero) number of internal reductions.

Observational equivalence We write $A \Downarrow c$ when A can evolve into a process that can send a message on c, that is, when $A \rightarrow^* C[\overline{c}\langle M\rangle.P]$ for some term M, some evaluation context $C[\_]$ that does not bind c, and some process P.

Definition 1. *Observational equivalence ($\approx$) is the largest symmetric relation $\mathcal{R}$ between processes such that $A\ \mathcal{R}\ B$ implies:*

1. *if $A \Downarrow c$, then $B \Downarrow c$.*
2. *if $A \rightarrow^* A'$ then, for some B', we have $B \rightarrow^* B'$ and $A'\ \mathcal{R}\ B'$;*
3. *$C[A]\ \mathcal{R}\ C[B]$ for all closing evaluation contexts $C[\_]$.*

We will express secrecy and unlinkability as the observational equivalence of two processes, that share the same operational structure and differ only on data that they handle. Formally, such two processes are represented in the ProVerif calculus as a *bi-process*. A bi-process is constructed in the same way as a process, with the single difference that the binary operator choice may be used in term evaluations. A bi-process P represents two processes with the same structure in the following sense: from P we can obtain a first process $\mathrm{fst}(P)$, by replacing all occurrences of $\mathsf{choice}[M_1, M_2]$ with M_1; a second process $\mathrm{snd}(P)$ can be obtained by replacing all occurrences of $\mathsf{choice}[M_1, M_2]$ with M_2, for all terms M_1, M_2. For example, the bi-process $\mathrm{out}(c, \mathsf{choice}[a, b])$ represents in ProVerif a test of whether $\mathrm{out}(c, a) \approx \mathrm{out}(c, b)$, which is false.

5.2 Properties and analysis

In this work, we prove using the ProVerif tool, that the ConfiChair protocol satisfies the intended secrecy and unlinkability properties informally described in section 2.1. The purpose of this section is to formally define these properties, and to show how they can be automatically verified with ProVerif. We define both secrecy and unlinkability properties as equivalences between processes adapting the classical approach of [34, 1, 19] to our context.

To express security properties we will need to consider particular authors and reviewers in interaction with the rest of the system. For this we consider a hole in the process CC, where we can plug any process, *i.e.* we let:

$$CC[\_] \stackrel{\text{def}}{=} \text{new } cshk;\ \text{new } cpbk;\ (!C \mid !R \mid !A \mid \_\ )$$

To express authors and reviewers who submit some specific data (of which the privacy will be tested), we consider the processes:

- $A_{pap}(ida, p, k)$ - for an author whose identity is ida and that behaves like a regular author, with the single difference that amongst other submissions, he also submits the paper p with the corresponding submission-key k.
- $R_{sc}(idr, sc)$ - for a reviewer whose identity is idr and that behaves like a regular reviewer, with the single difference that amongst other reviews, he also reviews a paper to which it attributes the score sc.

– $R_{rev}(idr, k, rev)$ - for a reviewer whose identity is idr and that behaves like a regular reviewer, with the single difference that amongst other reviews, he also reviews the paper corresponding to the submission-key k, and writes the review rev.

The formal definition of these processes is detailed in Appendix B.

Secrecy properties To formalise the considered secrecy properties, we rely on the notion of strong secrecy defined in [9].

Paper secrecy We say that a conference management protocol satisfies strong secrecy of papers if, even if the cloud initially knows p_1 and p_2, the cloud cannot make a distinction between an execution of the protocol where an author submitted the paper p_1 and an execution where he has submitted the paper p_2.

To formally capture this, we construct from $CC[\_]$ two processes: in the first one the hole is filled with an author that submits the publicly known (*i.e.* free) paper p_1, and in the second one the hole is filled with that author submitting the publicly known (*i.e.* free) paper p_2. We verify using ProVerif that these two processes are observationally equivalent:

$$CC[\text{new}\, ida;\ \text{new}\, k;\ A_{pap}(ida, p_1, k)] \ \approx\ CC[\text{new}\, ida;\ \text{new}\, k;\ A_{pap}(ida, p_2, k)]$$

Score secrecy Similarly, in order to verify the strong secrecy of scores, we build from $CC[\_]$ one process in which the hole is filled with a reviewer that attributes one to some paper, and one process in which the hole is filled with the reviewer attributing two to it.

$$CC[\text{new}\, idr;\ R_{sc}(idr, \mathsf{one})] \ \approx\ CC[\text{new}\, idr;\ R_{sc}(idr, \mathsf{two})]$$

Review secrecy The definition of secrecy of reviews is a bit more subtle. Reviews are sent to the authors at the notification phase, and the attacker could very well have submitted a paper. He would then rightfully obtain the review to his paper. So the property we want to formalise is that an attacker doesn't get to see the reviews of other authors' papers. In other words, review secrecy holds if, even if the cloud initially knows rev_1 and rev_2, the cloud cannot distinguish an execution of the protocol where a reviewer to a paper not submitted by the attacker writes the review rev_1 from an execution where the reviewer writes the review rev_2.

To capture this, we construct from $CC[\_]$ two processes. In the first one, the hole is filled with an honest author that submits a paper p with the corresponding submission-key k and a reviewer reviewing this paper and writing the publicly known (*i.e.* free) review rev_1. In the second one, an honest author that submits a paper p with the corresponding submission-key k and a reviewer reviewing this paper and writing the publicly known (*i.e.* free) review rev_2. For review secrecy to hold, the following equivalence must hold:

$$CC\left[\begin{array}{l}\text{new}\, ida;\ \text{new}\, p; \text{new}\, k; \text{new}\, idr;\\ (A_{pap}(ida, p, k) \mid\ R_{rev}(idr, k, rev_1))\end{array}\right] \ \approx\ CC\left[\begin{array}{l}\text{new}\, ida;\ \text{new}\, p; \text{new}\, k; \text{new}\, idr;\\ (A_{pap}(ida, p, k) \mid\ R_{rev}(idr, k, rev_2))\end{array}\right]$$

Analysis We used the ProVerif tool to prove that the equivalences described above hold, and thus that as announced ConfiChair does provide secrecy of papers, scores and reviews. The ProVerif source code for each of these equivalences is available online [29].

Author-reviewer unlinkability This property aims to guarantee that the links between a given author and the reviewers of his papers remain hidden from the cloud. To formalise it one could ask whether two processes are in observational equivalence: one in which ida's paper is reviewed by a reviewer idr_1, and another in which ida's paper is reviewed by a reviewer idr_2.

However, similarly to privacy in electronic voting [19], definitions of unlinkability are a bit more tricky. Since the identities of the authors that submit papers are revealed to the cloud at

submission time, and the identities of the reviewers are published when the review is submitted, unlinkability can not be ensured when there is a single reviewer, or a single author.

In order to give robust definitions of unlinkability we need to consider conferences with at least two reviewers and at least two authors submitting papers to it that are being reviewed by these reviewers. It is the chair's task to ensure that this is indeed the case. Accordingly, there is in the formal model a processes C_{ar} that ensures that at each stage of the conference at least two authors and two reviewers have executed their role. The detailed definition of C_{ar} is given in appendix B.

We prove that there is no observable difference between the case where reviewer idr_1 reviews ida_1's paper and reviewer idr_2 reviews ida_2's paper (left-handside process), and the case where reviewer idr_2 reviews ida_1's paper and reviewer idr_1 reviews ida_2's paper (right-handside process):

$$CC\left[\begin{array}{l}\text{new } p_1; \text{new } p_2; \text{new } k_1; \text{new } k_2;\\ \text{new } rev_1; \text{new } rev_2\\ C_{ar}(k_1, k_2, idr_1, idr_2) \mid\\ A_{pap}(ida_1, p_1, k_1) \mid A_{pap}(ida_2, p_2, k_2) \mid\\ R_{rev}(idr_1, k_1, rev_1) \mid R_{rev}(idr_2, k_2, rev_2)\end{array}\right] \approx CC\left[\begin{array}{l}\text{new } p_1; \text{new } p_2; \text{new } k_1; \text{new } k_2;\\ \text{new } rev_1; \text{new } rev_2\\ C_{ar}(k_1, k_2, idr_2, idr_1) \mid\\ A_{pap}(ida_1, p_1, k_1) \mid A_{pap}(ida_2, p_2, k_2) \mid\\ R_{rev}(idr_1, k_2, rev_1) \mid R_{rev}(idr_2, k_1, rev_2)\end{array}\right]$$

Analysis We used the ProVerif tool to prove that the equivalence described above hold, and thus that as announced ConfiChair does provide author-reviewer unlinkability. The ProVerif source code for this equivalence is available online [29].

A few words on the ProVerif code This section discusses an encoding technique that we use in order to help ProVerif handle the commutativity of the | operator.

As it has already been pointed in [20], although the equivalence $\text{out}(c, a) \mid \text{out}(c, b) \approx \text{out}(c, b) \mid \text{out}(c, a)$ trivially holds by commutativity of the parallel operator, ProVerif cannot prove it. Indeed, according to ProVerif the biprocess $\text{out}(c, \mathsf{choice}[a, b]) \mid \text{out}(c, \mathsf{choice}[b, a])$ corresponding to this equivalence doesn't satisfy uniformity under reductions, and therefore the equivalence cannot be proved by ProVerif.

In the ProVerif equivalence queries that we verified, we used the technique suggested in [20], consisting in swapping data so that ProVerif doesn't need to invoke the commutativity of the | operator to establish the considered equivalences. This technique uses the fact that the biprocesses $\mathcal{C}[\text{out}(c, M) \mid \text{out}(c, N)]$ and $\mathcal{C}[\text{out}(c, \mathsf{choice}[M, N]) \mid \text{out}(c, \mathsf{choice}[N, M])]$ are equivalent (*i.e.* the first satisfies observational equivalence if and only if the second does too), where $\mathcal{C}[\_]$ is any context (not necessarily an evaluation context).

When applying this technique to our trivial example we obtain the following biprocess for which ProVerif can now prove that observational equivalence holds:

$$\begin{array}{l}\text{let } x = \mathsf{choice}[\mathsf{choice}[a, b], \mathsf{choice}[b, a]] \text{ in}\\ \text{let } y = \mathsf{choice}[\mathsf{choice}[b, a], \mathsf{choice}[a, b]] \text{ in}\\ \text{out}(c, x) \mid \text{out}(c, y)\end{array}$$

This biprocess corresponds to the equivalence

$$\text{out}(c, a) \mid \text{out}(c, b) \approx \text{out}(c, a) \mid \text{out}(c, b)$$

which holds trivially, and can be established without having to invoke the commutativity of the | operator.

In our application, we use this technique when we communicate reviewer names to the cloud, while testing for the unlinkability of author-reviewer.

6 Conclusion

The accumulation of sensitive data on servers around the world is a major problem for society, and will be considerably exacerbated by the anticipated take-up of cloud-computing technology. The

fact that confidential data about the authoring and reviewing performance of tens of thousands of researchers across thousands of conferences is stored by well-known cloud-based systems serves to show how widespread and ubiquitous the problem is [32].

We have introduced a general technique that can be used to address this problem in a wide variety of circumstances, namely, the technique of translating between keys and mixing data in a trustworthy browser. We have proposed ConfiChair, a conference management system that uses this technique to obtain strong privacy properties while having all the advantages of cloud computing. In ConfiChair, the cloud sees sensitive data only in encrypted form, with no single person holding all the encryption keys (our protocol uses a different key for each conference). The conference chair's browser decrypts data with one key and encrypts it with possibly another one, while mixing and re-randomising to ensure unlinkability properties.

We are able to state and prove strong secrecy and unlinkability properties for ConfiChair. The protocol still enables the cloud provider to route information to the necessary chairs, reviewers and authors, to enforce access control, and optionally to perform statistics collection. We have demonstrated that the cryptography and key management can be handled by a regular web browser [29] (specifically, we used LiveConnect). We plan to continue developing our prototype into a complete system.

An important design decision in ConfiChair is the fact that a single key K_{Conf} is used to encrypt all the information for the conference Conf. Stronger secrecy properties could be obtained if a different key were used for different subsets of reviewers and papers, but this would be at the cost of simplicity. Using a single key per conference seems to strike a good balance between usability and security. Finer-grained access control is implemented (as on current systems) by the cloud, e.g. for managing the conflicts of interest.

In further work, we intend to apply the ideas to work with other cloud-computing applications (such as those mentioned in the introduction), and to provide a framework for expressing secrecy and unlinkability properties in a more systematic way.

Acknowledgments. Thanks to Joshua Phillips for much help with the implementation and typesetting. We also gratefully acknowledge financial support from EPSRC via the projects *Trust Domains* (TS/I002529/1) and *Trustworthy Voting Systems* (EP/G02684X/1).

References

1. Martín Abadi. Security protocols and their properties. In *Foundations of Secure Computation, NATO Science Series*, pages 39–60. IOS Press, 2000.
2. Martín Abadi and Cédric Fournet. Mobile values, new names, and secure communication. In *Proceedings of the 28th ACM Symposium on Principles of Programming Languages (POPL'01)*, pages 104–115, January 2001.
3. Ben Adida. Helios: Web-based open-audit voting. In Paul C. van Oorschot, editor, *USENIX Security Symposium*, pages 335–348. USENIX Association, 2008.
4. Ben Adida, Olivier Pereira, Olivier De Marneffe, and Jean-Jacques Quisquater. Electing a university president using open-audit voting: Analysis of real-world use of helios. In *In Electronic Voting Technology/Workshop on Trustworthy Elections (EVT/WOTE*, 2009.
5. Randolph Baden, Adam Bender, Neil Spring, Bobby Bhattacharjee, and Daniel Starin. Persona: an online social network with user-defined privacy. In Pablo Rodriguez, Ernst W. Biersack, Konstantina Papagiannaki, and Luigi Rizzo, editors, *SIGCOMM*, pages 135–146. ACM, 2009.
6. Olivier Baudron, Pierre-Alain Fouque, David Pointcheval, Jacques Stern, and Guillaume Poupard. Practical multi-candidate election system. In *PODC*, pages 274–283, 2001.
7. John Bethencourt, Amit Sahai, and Brent Waters. Ciphertext-policy attribute-based encryption. In *IEEE Symposium on Security and Privacy*, pages 321–334. IEEE Computer Society, 2007.
8. Bruno Blanchet. An efficient cryptographic protocol verifier based on Prolog rules. In *Computer Security Foundations Workshop (CSFW'01)*, 2001.
9. Bruno Blanchet. Automatic proof of strong secrecy for security protocols. In *IEEE Symposium on Security and Privacy*, pages 86–, 2004.
10. Bruno Blanchet, Martín Abadi, and Cédric Fournet. Automated verification of selected equivalences for security protocols. *Journal of Logic and Algebraic Programming*, 2007.

11. Dan Boneh, Craig Gentry, and Brent Waters. Collusion resistant broadcast encryption with short ciphertexts and private keys. In Victor Shoup, editor, *CRYPTO*, volume 3621 of *Lecture Notes in Computer Science*, pages 258–275. Springer, 2005.
12. Matteo Bortolozzo, Matteo Centenaro, Riccardo Focardi, and Graham Steel. Attacking and fixing PKCS#11 security tokens. In *ACM Conference on Computer and Communications Security*, pages 260–269, 2010.
13. Rajkumar Buyya, Chee Shin Yeo, Srikumar Venugopal, James Broberg, and Ivona Brandic. Cloud computing and emerging IT platforms: Vision, hype, and reality for delivering computing as the 5th utility. *Future Generation Computer Systems*, 25(6):599 – 616, 2009.
14. Iliano Cervesato, Aaron D. Jaggard, Andre Scedrov, Joe-Kai Tsay, and Christopher Walstad. Breaking and fixing public-key kerberos. *Inf. Comput.*, 206:402–424, February 2008.
15. C. Chatmon, T. van Le, and T. Burmester. Secure anonymous RFID authentication protocols. Technical Report TR-060112, Florida Stat University, Department of Computer Science, 2006.
16. Tom Chothia and Vitaly Smirnov. A traceability attack against e-passports. In *Financial Cryptography*, 2010.
17. Sebastian Clauß, Dogan Kesdogan, Tobias Kölsch, Lexi Pimenidis, Stefan Schiffner, and Sandra Steinbrecher. Privacy enhancing identity management: Protection against re-identification and profiling. In *Proceedings of the 2005 ACM Workshop on Digital Identity Management*, 2005.
18. Cloud Security Alliance. *Secure Cloud*. `www.cloudsecurityalliance.org/sc2010.html`, 2010.
19. Stéphanie Delaune, Steve Kremer, and Mark D. Ryan. Verifying privacy-type properties of electronic voting protocols. *Journal of Computer Security*, 17(4):435–487, July 2009.
20. Stéphanie Delaune, Mark D. Ryan, and Ben Smyth. Automatic verification of privacy properties in the applied pi calculus. In *Proceedings of the 2nd Joint iTrust and PST Conferences on Privacy, Trust Management and Security (IFIPTM'08)*, volume 263, pages 263–278. IFIP International Federation for Information Processing, 2008.
21. C. Gentry. Fully homomorphic encryption using ideal lattices. In *41st ACM Symposium on Theory of Computing (STOC)*, 2009.
22. Saikat Guha, Kevin Tang, and Paul Francis. NOYB: Privacy in online social networks. In *In Proceedings of the First ACM SIGCOMM Workshop on Online Social Networks*, 2008.
23. Yan Huang and David Evans. Private editing using untrusted cloud services. In *Second International Workshop on Security and Privacy in Cloud Computing, Minneapolis, Minnesota. 24 June 2011.*
24. Markus Jakobsson, Ari Juels, and Ronald L. Rivest. Making mix nets robust for electronic voting by randomized partial checking. In Dan Boneh, editor, *USENIX Security Symposium*, pages 339–353. USENIX, 2002.
25. Ari Juels, Dario Catalano, and Markus Jakobsson. Coercion-resistant electronic elections. In Vijay Atluri, Sabrina De Capitani di Vimercati, and Roger Dingledine, editors, *WPES*, pages 61–70. ACM, 2005.
26. Swee-Won Lo, Raphael C.-W. Phan, and Bok-Min Goi. On the security of a popular web submission and review software (WSaR) for cryptology conferences. In *WISA'07: Proceedings of the 8th international conference on Information security applications*, pages 245–265, Berlin, Heidelberg, 2007. Springer-Verlag.
27. Gavin Lowe. An attack on the Needham-Schroeder public-key authentication protocol. *Information Processing Letters*, 56(3):131–133, 1996.
28. Siani Pearson, Yun Shen, and Miranda Mowbray. A privacy manager for cloud computing. In Martin Gilje Jaatun, Gansen Zhao, and Chunming Rong, editors, *Cloud Computing, First International Conference, CloudCom 2009, Beijing, China, December 1-4, 2009. Proceedings*, pages 90–106, 2009.
29. Joshua Phillips and Matt Roberts. ConfiChair - prototype privacy-supporting conference management system. `https://confichair.markryan.eu`.
30. Krishna P. N. Puttaswamy, Christopher Kruegel, and Ben Y. Zhao. Silverline: Toward data confidentiality in third-party clouds. Technical Report 08, University of California Santa Barbara, 2010.
31. Hasan Qunoo and Mark Ryan. Modelling dynamic access control policies for web-based collaborative systems. In *Data and Applications Security and Privacy XXIV*, volume 6166 of *LNCS*, pages 295–302, 2010.
32. Mark D. Ryan. Cloud computing privacy concerns on our doorstep. *Communications of the ACM*, 54(1):36–38, 2011.
33. Ahmad-Reza Sadeghi, Thomas Schneider, and Marcel Winandy. Token-based cloud computing. In Alessandro Acquisti, Sean W. Smith, and Ahmad-Reza Sadeghi, editors, *Trust and Trustworthy Computing, Third International Conference, TRUST 2010, Berlin, Germany, June 21-23, 2010. Proceedings*, pages 417–429, 2010.
34. Steve Schneider and Abraham Sidiropoulos. CSP and anonymity. In *ESORICS*, pages 198–218, 1996.

A Formal definition of the protocol

We will make extensive use of the following syntactic sugar of ProVerif: in the expression M from the construct "let $M = D$ in P else Q" one is allowed to have subexpressions of the form "$=N$", for some term N. In that case, the subexpressions "$=N$" is an additional constraint on the matching of M and D, requiring that the term at the corresponding position in M matches also with N.

```
(************************************ SIGNATURE & EQUATIONAL THEORY ************************************)
free c.
fun pub/1.
fun aenc/3.
reduc adec(x, aenc(pub(x), y, z)) = z.
fun senc/3.
reduc sdec(x, senc(x, y, z)) = z.
fun one/0. fun two/0.
fun subm/0. fun initrev/0. fun revw/0. fun dsc/0. fun ntf/0. fun true/0.

(******************************************** CHAIR ********************************************)
let C = new shkconf; new prvconf; ((C_init) | (!(C_review)) | (!(C_notify))).

let C_init =
    ((!out(cshkconf, shkconf)) | (out(cpubconf, pub(prvconf))) | (!out(c, pub(prvconf)))).

let C_review =
    new mu;
    new rand;
    in(c, xsubmission);
    let (xlambda, xidauthor, xblob) = xsubmission in (
    let (=subm, xlambda', xsubmissionkey) = adec(prvconf, xblob) in (
    in(c, xreviewer);
    in(c, xconflict);
    let for_rev = (mu, xreviewer, xconflict, senc(shkconf, rand, (initrev, xreviewer, mu, xlambda',
                                                                  xsubmissionkey))) in (
    out(c, for_rev)))).

let C_notify =
    new rand;
    in(c, zsubmission);
    let (zmu, zreviewer, zblob) = zsubmission in (
    let (=dsc, zmu', zlambda, zsubmissionkey, zreview, zscore, zdiscussion) = sdec(shkconf, zblob) in (
    in(ccoin, zntf);
    let for_notf = (zlambda, senc(zsubmissionkey, rand, (ntf, zlambda, zntf, zreview))) in(
    out(c, for_notf)))).

(******************************************** REVIEWER ********************************************)
let R = new idreviewer; out(c, idreviewer); (!(R_main)).

let R_main =
    in(cshkconf, xshkconf); out(c, idreviewer); out(c, idreviewer); in(c, xdbpapers);
    ((!R_review) | (!R_discuss)).

let R_review =
    new rand;
    in(c, xsubmission);
    let (xmu, =idreviewer, xblob) = xsubmission in (
    let (=initrev, =idreviewer, xmu', xlambda, xsubmissionkey) = sdec(xshkconf, xblob) in (
    in(ccoin, xscore);
    new review;
    let subm_reviewed = (xmu, idreviewer, senc(xshkconf, rand, (revw, xmu', xlambda, xsubmissionkey,
                                                                review, xscore))) in (
    out(c, subm_reviewed)))).

let R_discuss =
    new rand';
    in(c, yreview);
    let (ymu, yblob) = yreview in (
    let (=revw, ymu', ylambda, ysubmissionkey, yr, ys) = sdec(xshkconf, yblob) in (
    new discussion;
    let subm_discussion = (ymu, idreviewer, senc(xshkconf, rand', (dsc, ymu', ylambda, ysubmissionkey,
                                                                   yr, ys, discussion))) in (
    out(c, subm_discussion)))).

(******************************************** AUTHOR ********************************************)
let A = new idauthor; (!(A_create)).

let A_create =
    new lambda; new paper; new submissionkey; (A_submit).
```

```
let A_submit =
    new rand1;  new rand2;
    in(cpubconf, xpubconf);
    let key = (lambda, idauthor, aenc(xpubconf, rand1, (subm, lambda, submissionkey))) in (
    let pap = (lambda, idauthor, senc(submissionkey, rand2, (lambda, idauthor, paper))) in (
    out(c, (key, pap));
    in(c, xntf))).

(********************************************** COIN **********************************************)
let Coin = ((!(out(ccoin, one))) | (!(out(ccoin, two)))).

(********************************************** CC **********************************************)
process new cshkconf; new cpubconf; new ccoin; ((!C) | (!R) | (!A) | Coin)
```

B Formal definition of witnessing processes

To formalise our secrecy and unlinkability properties, we need to consider particular instances of authors and reviewers called witnesses (of the considered property). The witnessing authors are instances of the process A_{pap} defined bellow. According to the considered property, the witnessing reviewers are instances of the processes R_{sc} or R_{rev}, whose intuitive meaning is given in section 5.2.

B.1 Definition of A_{pap}

The process $A_{pap}(xida, xp, xk)$ models an author whose identity is $xida$ and is ready to submit new papers, as well as the paper xp encrypted with the key xk given in argument:

1. $A_{pap}(xida, xp, xk) \stackrel{\text{def}}{=}$
2. $\quad((!A\_create) \mid (A_{pap}\_create))$
3.
4. $A_{pap}\_create \stackrel{\text{def}}{=}$
5. $\quad$let $idauthor = xida$ in (
6. $\quad$let $paper = xp$ in (
7. $\quad$let $submissionkey = xk$ in (
8. $\quad A\_submit)))$

where *A_create* and *A_submit* are as defined in Appendix A.

B.2 Definition of R_{sc}

The process $R_{sc}(xidr, xsc)$ models a reviewer with identity $xidr$, ready to submit many reviews, as well as a review to a paper that gets attributed the score xsc given in argument:

1. $R_{sc}(xidr, xsc) \stackrel{\text{def}}{=}$
2. $\quad$let $idreviewer = xidr$ in (
3. $\quad\text{out}(c, xidr)$
4. $\quad((!R\_main) \mid (R_{sc}\_main)))$
5.
6. $R_{sc}\_main \stackrel{\text{def}}{=}$
7. $\quad\text{in}(cshkconf, xshkconf);\ \text{out}(c, idreviewer);\ \text{out}(c, idreviewer);\ \text{in}(c, xdbpapers);$
8. $\quad((!R\_review) \mid (R_{sc}\_review) \mid (!R\_discuss))$
9.
10. $R_{sc}\_review \stackrel{\text{def}}{=}$
11. $\quad$new $rand$;
12. $\quad\text{in}(c, xsubmission)$
13. $\quad$let $(xmu, = xidr, xblob) = xsubmission$ in (
14. $\quad$let $(= initrev, = xidr, xmu', xlambda, xsubmissionkey) = \mathsf{sdec}(xshkconf, xblob)$ in (
15. $\quad(*\ \text{in}(ccoin, xscore);\ *)$
16. $\quad$let $xscore = xsc$ in (
17. $\quad$new $review$;
18. $\quad$let $subm\_reviewed =$ in
19. $\quad(xmu, xidr, senc(xshkconf, rand, (revw, xmu', xlambda, xsubmissionkey, review, xscore)))$ in (
20. $\quad\text{out}((, c), subm\_reviewed)))))$

where *R_main*, *R_review* and *R_discuss* are as defined in Appendix A. Note that R_{sc}_review and the normal behaviour *R_review*, only differ in the score attributed to the paper under review. Normally, *i.e.* in *R_review*, the reviewer picks randomly a score. In R_{sc}_review this is not the case. We have commented the original choice of the score (line 15), and instead have added line 16 to assign xsc given in argument to the variable $xscore$.

B.3 Definition of R_{rev}

The definition of R_{rev} is very similarly to the one of R_{sc}. The process $R_{rev}(xidr, xk, xrev)$ models a reviewer with identity $xidr$, ready to review many papers, as well as a paper encrypted with the key xk and to write the review $xrev$ for it:

$R_{rev}(xidr, xk, xrev) \stackrel{\text{def}}{=}$
  let $idreviewer = xidr$ in (
  $\text{out}(c, xidr)$
  $((!R\_main) \mid (R_{rev}\_main)))$

$R_{rev}\_main \stackrel{\text{def}}{=}$
  $\text{in}(cshkconf, xshkconf);\ \text{out}(c, idreviewer);\ \text{out}(c, idreviewer);\ \text{in}(c, xdbpapers);$
  $((!R\_review) \mid (R_{rev}\_review) \mid (!R\_discuss))$

$R_{rev}\_review \stackrel{\text{def}}{=}$
  new $rand$;
  $\text{in}(c, xsubmission)$
  let $(xmu, = xidr, xblob) = xsubmission$ in (
  let $(= initrev, = xidr, xmu', xlambda, = xk) = \mathsf{sdec}(xshkconf, xblob)$ in (
  $\text{in}(ccoin, xscore)$;
  $(*$ new $review;\ *)$
  let $review = xrev$ in (
  let $subm\_reviewed =$ in
  $(xmu, xidr, senc(xshkconf, rand, (revw, xmu', xlambda, xsubmissionkey, review, xsc)))$ in (
  $\text{out}((, c), subm\_reviewed)))))$

B.4 Definition of C_{ar}

The process $C_{ar}(xk1, xk2, xidr1, xidr2)$ models the chair of a conference who has assigned the reviewer $xidr1$ to the paper encrypted with the key $xk1$, and the reviewer $xidr2$ to the paper encrypted with the key $xk2$:

$C_{ar}(xk1, xk2, xidr1, xidr2) \stackrel{\text{def}}{=}$
  new $shkconf$; new $prvconf$;
  $((C\_init)|(!(C\_review))|(!(C\_notify))$
  $(C_{ar}\_review)|(C_{ar}\_notify)|$
  $(R_{rev}(xidr1, xk1, rev1))|(R_{rev}(xidr2, xk2, rev2))|(A_{pap}(ida1, p1, xk1))|(A_{pap}(ida2, p2, xk2))).$

$C_{ar}\_review \stackrel{\text{def}}{=}$
  new $rand1$; new $rand2$;
  new $mu1$; new $mu2$;
  $\text{in}(c, xsubmission1); \text{in}(c, xsubmission2);$
  let $(xlambda1, xidauthor1, xblob1) = xsubmission1$ in (
  let $(xlambda2, xidauthor2, xblob2) = xsubmission2$ in (
  let $(= subm, xlambda1', = wtn(k1)) = adec(prvconf, xblob1)$ in (
  let $(= subm, xlambda2', = wtn(k2)) = adec(prvconf, xblob2)$ in (
  $\text{in}(c, = idr1)$;
  $\text{in}(c, = idr2)$;
  $\text{in}(c, xconflict1)$;
  $\text{in}(c, xconflict2)$;
  let $for\_rev1 =$ in
  $(mu1, \mathsf{choice}[idr1, idr2], senc(shkconf, rand1, (initrev, \mathsf{choice}[idr1, idr2], mu1, xlambda1', wtn(k1))))$ in (
  let $for\_rev2 =$ in
  $(mu2, \mathsf{choice}[idr2, idr1], senc(shkconf, rand2, (initrev, \mathsf{choice}[idr2, idr1], mu2, xlambda2', wtn(k2))))$ in (
  $out(c, \mathsf{choice}[for\_rev1, for\_rev2])|out(c, \mathsf{choice}[for\_rev2, for\_rev1]))))))).$

$C_{ar}\_notify \stackrel{\text{def}}{=}$
  $\text{in}(c, zsubmission1); \text{in}(c, zsubmission2);$
  let $(zmu1, zreviewer1, zblob1) = zsubmission1$ in (
  let $(zmu2, zreviewer2, zblob2) = zsubmission2$ in (
  let $(= dsc, zmu1', zlambda1, = \mathsf{choice}[wtn(k1), wtn(k2)], zreview1, zscore1, zdiscussion1) =$ in
  $\mathsf{sdec}(shkconf, zblob1)$ in (
  let $(= dsc, zmu2', zlambda2, = \mathsf{choice}[wtn(k2), wtn(k1)], zreview2, zscore2, zdiscussion2) =$ in
  $\mathsf{sdec}(shkconf, zblob2)$ in (
  $\text{in}(ccoin, zntf1); \text{in}(ccoin, zntf2);$
  new $rand1$; new $rand2$;
  let $for\_notf1 = (zlambda1, senc(\mathsf{choice}[wtn(k1), wtn(k2)], rand1, (ntf, zlambda1, zntf1, zreview1)))$ in (
  let $for\_notf2 = (zlambda2, senc(\mathsf{choice}[wtn(k2), wtn(k1)], rand2, (ntf, zlambda2, zntf2, zreview2)))$ in (
  $\text{out}(c, \mathsf{choice}[for\_notf1, for\_notf2])|\text{out}(c, \mathsf{choice}[for\_notf2, for\_notf1]))))))).$