View metadata, citation and similar papers at core.ac.uk

brought to you by CORE

provided by Edinburgh Research Explorer

THE UNIVERSITY of EDINBURGH

Edinburgh Research Explorer

# Privacy-supporting Cloud Computing by In-browser Key Translation

**Citation for published version:**
Arapinis, M, Bursuc, S & Ryan, M 2013, 'Privacy-supporting Cloud Computing by In-browser Key Translation' Journal of Computer Security, vol. 21, no. 6, pp. 847-880. DOI: 10.3233/JCS-130489

**Digital Object Identifier (DOI):**
10.3233/JCS-130489

**Link:**
Link to publication record in Edinburgh Research Explorer

**Document Version:**
Peer reviewed version

**Published In:**
Journal of Computer Security

**General rights**
Copyright for the publications made accessible via the Edinburgh Research Explorer is retained by the author(s) and / or other copyright owners and it is a condition of accessing these publications that users recognise and abide by the legal requirements associated with these rights.

**Take down policy**
The University of Edinburgh has made every reasonable effort to ensure that Edinburgh Research Explorer content complies with UK legislation. If you believe that the public display of this file breaches copyright please contact openaccess@ed.ac.uk providing details, and we will remove access to the work immediately and investigate your claim.

Download date: 05. Apr. 2019

# Privacy-supporting cloud computing by in-browser key translation

Myrto Arapinis Sergiu Bursuc 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 facilitating some data-dependent computations. The technique is based on key translations and mixes in web browsers.

We focus on a particular kind of software-as-a-service, namely, services that support applications, evaluations, and decisions. Such services include job application management, public tender management (e.g., for civil construction), and conference management. We identify the specific security and privacy risks that existing systems pose. We propose a protocol that addresses them, and forms the basis of a system that offers strong security and privacy guarantees.

We express the protocol and its properties in the language of ProVerif, and prove that it does provide the intended properties. We describe an implementation of a particular instance of the protocol called ConfiChair, which is geared to the evaluation of papers submitted to conferences.

## 1 Introduction

Cloud computing refers to the idea that data and programs can be stored centrally, in the cloud, and accessed any time from anywhere through thin clients and lightweight mobile devices. On one hand, cloud computing brings many advantages, including data ubiquity, flexibility of access, and resilience; indeed, it also enhances security: the cloud provider may be able to afford to invest in better and more up-to-date security technologies and practices than the data owner can. On the other hand, however, since cloud computing necessarily puts data outside of the control of the data owner, it inevitably introduces security issues too. cloud computing raises privacy and confidentiality concerns because the service provider has access to all the data, and could accidentally or deliberately disclose it.

Familiar examples of cloud computing include cloud-based storage (such as Dropbox), online documents (such as Google docs), and customer-relationship management systems (such as salesforce.com) are familiar examples. These are commonly known as *software-as-a-service* (SaaS) applications. SaaS applications are being adopted widely throughout business. For example, Google Apps currently supports 40 million individual employees from 4 million businesses that pay US$50 per user per year. Those businesses have outsourced their email, calendar, and documents to Google. Currently, 5000 business sign up to Google Apps each day. But, for the security reasons mentioned, many organisations are not willing to use cloud-based services for business-confidential data. Such organisations typically include lawyers, doctors, insurance companies, banks, engineers, and government. Solutions are sought that would offer such companies some of the benefits of cloud computing, without the confidentiality risk.

If the cloud's role is confined to storing the data on behalf of the owner, then the problem can be solved by encrypting the data, with the owner holding the keys. However, typically one wants

the cloud provider to be able to do non-trivial computations with the data, and therefore the problem is in fact very hard to solve. Such computations may include searches, transformations, selections, and access control decisions. For example:

- The data is an email archive, and the cloud is expected to handle email in a way that depends on its content (e.g., spam filtering, topic classification, compliance with corporate disclosure policies).
- The data is documents and photographs, and the cloud is expected to enforce access rules that depend on the content.
- The data may be scientific data obtained by experiment and observation, and the computation may be to find patterns, or organise the data according to rules. Other examples in this category include data-intensive financial modelling or network traffic modelling.
- The data may be financial data, payroll data, human resource data, banking data, etc., and the computation is usual business processing.
- The data may be documents to review (job applications, research papers, business tenders) and the computation may be to support the review process (sending to reviewers, collating reviews, etc.).

In this paper, we provide a solution for a last-mentioned kind of SaaS application in the list above, namely, services that support applications, evaluations, and decisions. Such services include job application management, public tender management (e.g., for civil construction), and conference management. We identify a set of confidentiality requirements for such SaaS applications, and we propose a way to construct applications which achieves the requirements. The confidentiality guarantees ensure that no-one has access to the application data, beyond the access that is explicitly granted to them by their participation in the service. In particular, this is true about the cloud provider and managers. We describe a protocol in which applicants, evaluators and decision makers interact through their web browsers with the cloud-based management system, to perform the usual tasks of uploading and downloading applications and evaluations. 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 an evaluator doesn't see other evaluations of an application before writing her own. However, all the sensitive data is seen by the cloud only in encrypted form, and the cloud is not able to tell which applicants are being reviewed by which evaluators.

The principal idea behind the protocol is that the cloud provider processes only ciphertexts, and the keys are manipulated by the browsers of the participants in the system. Since often these manipulations involve decrypting data with one key and encrypting with another, we call this "key translation in the browser".

**Our contributions.**

1. We identify a class of cloud-based systems for competition management and propose a general framework to reason about the functionality and privacy of such systems (section 2)
2. We propose a general protocol that provides secrecy and unlinkability properties for cloud-based competition management, relying on a trusted competition manager and in-browser key translation (section 3)
3. We show how the desired privacy properties can be formally expressed and automatically verified with ProVerif (section 4)
4. We present the implementation of a privacy-supporting system for conference management (ConfiChair), which is based on our general protocol (section 5).

Contributions 1-3 are a generalization of a specific protocol for conference management and of its verification, that we have proposed in [9].

## 2 Description of the problem and related work

As mentioned, our target is any SaaS system that supports applications, evaluations, and decisions. There are many systems of this kind in the real world, and increasingly they are cloud-based. We give some examples:

**Conference management systems** are used to support the reviewing and discussion procedures needed to decide which papers are accepted for presentation at conferences. Authors submit papers online, and programme committee members download the papers to evaluate them, and upload their reviews. The conference management system typically also supports online discussion of the reviews and the acceptance decision. EasyChair and the Editor's Assistant (EDAS) are well-known examples of cloud-based conference management systems. For example, EasyChair currently hosts more than 3000 conferences per year, and has about 600,000 users [43].

**Public tender management** is the process in which governments and their agencies collect and evaluate bids (or tenders) to supply services. Bidders submit tenders, and evaluators review and compare the bids in order to accept one or more of them. There may be a pre-qualification stage, in which potential tenderers have to prove their eligibility, and there may be a tender-refinement stage, in which bidders can answer questions and supply more information about their bid [3]. A plethora of tender management systems exists [1, 4, 2].

**Recruitment for employment** is the process of attracting, screening, and selecting a qualified person for a job. Applicants respond to advertisements by making an application; referees and other evaluators comment on the suitability of the applicants, and a panel makes a decision about whom to offer the position. WCN is a UK-based e-recruitment provider.

These systems involve a similar workflow, although they use different terminology and the workflow may differ in small details. We illustrate the terminological differences in the table below:

| *Generic* | *Conference* | *Procurement* | *Employment* |
|---|---|---|---|
| Invitation | Call for papers | Invitation to tender | Job description |
| Panel | Programme committee | Evaluation team | Panel |
| Manager | Programme chair | Manager | Recruiter |
| Panel member | PC member | Evaluator | Referee |
| Application | Paper | Tender | Application |
| Applicant | Author | Bidder | Applicant |
| Evaluation | Review | Evaluation | Evaluation |
| Outcome | Decision | Announcement | Outcome |

In this paper, we use the generic terms in the left-hand column. A panel member will sometimes be called reviewer or evaluator, and the competition manager will sometimes be called chair.

*Competition management workflow.* In general, a competition comprises an initialisation phase, followed by a certain number of rounds. The initialisation consists in the opening of the competition by the chair, the registration of applicants, and the declaration of conflicts between applicants and panel members. A round consists in 5 consecutive phases: first the applicants submit the documents necessary to the current round (Submission phase), then each submission gets assigned to panel members (Assignment phase) for evaluation (Evaluation phase) and discussion (Discussion phase). Finally the applicants get notified the outcome of the current round and may also get instructions for the following round (Notification phase).

Depending on the considered scenario, there may be several rounds to the competition. Each round consists of five phases (Submission, Assignment, Evaluation, Discussion, and Notification)

described below. For example, in the context of conference management, in the first round authors submit papers which get reviewed and discussed by the programme committes. There can also be a second round, usually called rebuttal, where authors are asked to submit their responses to the first reviews, which in turn get reviewed and discussed. Similarily, tender bidding processes usually comprise three rounds, namely the pre-qualification, the bidding and the refinement rounds.

*Security concerns.* The different kinds of competition management systems share similar security concerns. In the case of cloud-based conference management systems, these have already been well-documented [40]. Cloud-based systems such as EasyChair contain 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 system administrators, and could be accidentally or deliberately disclosed. In the case of tender management systems and employment recruitment systems, the risks are similar. The centralised existence of sensitive data about individuals and organisations can create conflicts of interest for the custodians, and also can attract the attention of hackers and crackers.

Note that the data confidentiality issue concerns the cloud system administrator (who administrates the system for all competitions), not the individual competition manager (who is concerned with a single competition). With current systems, the cloud system administrator has access to all the data on the system, across all of the competitions and all of the applicants and evaluators An individual competition manager, on the other hand, has access to the data only for the particular competition of which she is manager. Moreover, an applicant or evaluator that chooses to participate in the competition can be assumed to be willing to trust the manager (for if he didn't, he would not participate); but there is no reason to assume that he trusts or even knows the cloud management system provider.

This paper proposes a competition management protocol which does not allow the cloud infrastructure provider to access any of the data. But solving the problem of data confidentiality has to be done in a way that allows the service provider to offer useful functionality, and in a way that results in a system with good usability features. Thus, 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 cannot 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.

For brevity, we use the term "cloud" to include all roles that are not an explicit part of the service management; that includes the service 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 applications and evaluations.** The cloud does not have access to the content of applications or of their evaluations.
- **Unlinkability of applicant-evaluator.** The cloud *does* have access to the names of applicants and the names of evaluators. This access is required in order to route information correctly, to enforce access control, and to allow a logged-in user to see all his data in a unified way. However, the cloud does not have the ability to tell if a particular applicant was reviewed by a particular evaluator.

**Applicability of the ideas** The protocol we propose is limited to a particular class of SaaS systems, namely those that support applications, evaluations, and decisions. In future work, we intend to explore wider classes of SaaS systems that might also benefit from the technique of in-browser key translation and mixing, such as the following:

- Customer relationship management systems (such as salesforce.com);
- 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.

## 2.1 Threat model

Cloud service providers such as Google or Dropbox should not be considered fully malicious, since it is in their interest to attract new customers and keep existing ones, and to protect their customers from third-party threats. This means that they would not want to launch attacks on their customers which would damage their reputation as a cloud provider. The appropriate attacker model for cloud providers lies somewhere in between the assumption that the cloud is fully trusted (which is the assumption users are obliged to make today) and the assumption that it is fully malicious.

In the literature, there is already an attacker model in between these two extremes, namely, the "honest-but-curious" attacker (Figure 1). It says that the attacker launches passive attacks but not active ones. We don't believe this is the right attacker model for cloud providers, however. There is no reason to suppose a cloud provider will refrain from active attacks, if it can get away from it. We adopt the term "malicious-but-cautious attacker" for our threat model. In this model, the cloud provider is assumed to be malicious, in the sense of actively trying to mount attacks against the privacy of cloud users. At the same time, the cloud provider is assumed to be cautious of not leaving any verifiable evidence of its misbehaviour. The malicious-but-cautious attacker model is already implicitly used, e.g., in electronic voting. An election manager typically can cheat, but doing so would be detected by tests that voters use to detect election integrity [33, 19] or voter coercion [28]. This attacker model is also related to the *covert adversary* of [10], but it additionally assumes that the cloud provider acts to protect its users from third-party attacks.

**Privacy requirements.** In terms of our SaaS competition-management application, this means that the cloud provider will be willing to act maliciously if it can do so in a way which won't be detected by the users. As previously mentioned, we use the term "cloud" to refer to the cloud service provider, competition management system and its administrators, and the network. The protocol must protect the secrecy from the cloud of

- the content of submitted applications,
- the content of submitted evaluations,

Further, when data is necessarily known to the cloud in order that it can fulfil the functional requirements, we require what we call the *unlinkability* property: the cloud is unable to determine if a particular applicant is being evaluated by a particular evaluator or not; that is, the cloud cannot infer

- the link between applicants and evaluators

However, we can assume that the cloud provider will

- Collect and store data relevant to the competition, including names of evaluators and applicants, as well as applications and evaluations.
- To enforce access control in respect of conflicts of interest and ensuring that evaluators see other evaluations of an application only after they have submitted their own.
- Manage the information flow of the competition: from applicants, to competition manager, to evaluators and back.
- Notify the applicants of the outcome of their applications.

If it failed to carry out these responsibilities, it would risk that the users would detect it, and move their custom elsewhere. We assume it is not willing to take this risk.


Figure 1: A *malicious* provider is assumed to be willing to use any available strategy to attack, no matter what the consequence. A *malicious but cautious* provider is assumed to launch no attacks that would leave readily verifiable evidence of the attack. An *honest but curious* provider is assumed to launch no attacks that would leave any trace (whether verifiable or not); he confines himself to passive attacks. An *honest* provider is assumed to launch no attacks.

**Assumptions.** We assume that the cloud provider is not invited to participate as an evaluator or a manager, and does not collude with any such party. Our properties could not possibly hold without this assumption, since the provider would necessarily have access to privileged information. However, we allow the cloud provider to participate as an applicant. It would not be reasonable to forbid this, since there is no eligibility requirement on being an applicant. To summarise, the cloud provider does participates in the protocol only as provider of the cloud service or as applicant corresponding to an application, or both.

We also 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).

We do not consider timing attacks and other kinds of side-channel attacks. (Proving resistance to such attacks is quite a specialised topic, and there are emerging some standard defences that can be deployed.)

**Usability requirements.** The system should be as easy to use as present-day web-based competition management systems, such as EasyChair, iChair, eTenderer, VHTender, or WCN. 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 5 which describes a prototype implementation of an instance of our protocol.

## 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., [20] includes an overview), and there is now a conference series devoted to that topic [25]. 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 [41], 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 [36].

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 [27]. 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. For these reasons, we think FHE does not solve the problem of confidentiality from the cloud provider for this kind of SaaS application. 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 [13, 11] 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 [38]. 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 [22] or in privacy enhancing identity management systems [24] for example, an important requirement is that two transactions from the same agent should not be linkable in order to prevent users from being tracked or profiled. Another prominent example of an application that requires unlinkability is electronic voting: a voter must not be linked to the vote that he has cast [26]. Moreover, like user 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 [32, 31], or on restricted versions of homomorphic encryption that allow the addition of plaintexts [7, 12]. Our proposed protocol also relies on mixing, showing how this idea can be adapted to new application areas.

Other systems identify applications where the cloud can be provided with "fake" data without affecting functionality [29]. 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 [29] is restricted to very specific applications. In particular, a conference management system cannot 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 [34, 39]. These are also important concerns, but that are orthogonal to ours, where we are interested in a system design for ensuring confidentiality from the cloud provider.

# 3 The protocol

The main idea of the protocol is that the cloud provider sees only ciphertexts. When applications are submitted, they are encrypted with keys that are themselves encrypted with the public key of the application system; later, those keys are decrypted and encrypted with a symmetric key that is available to panel members. Since the cloud is not trusted to have the secret keys or the plaintext, this decryption and encryption is done by the manager's browser. Later in the protocol, there are other similar episodes in which the manager's browser decrypts some ciphertexts and encrypts some of the data, possibly again with a different key. We call these episodes "key translations in the browser".

## 3.1 Description

The protocol is informally described in Figures 2-4 on the following pages. The protocol roles are: applicant, manager and evaluator (also called panel member). All the actions are performed through the cloud, except the transmission of a symmetric key, to be secretly shared among the panel members, and the transmission of the public key of the conference to applicants, that we have to assume authentic. The cloud may perfom any actions with the messages that it gets by running the protocol. What is illustrated in diagrams are some of the actions that are required from the cloud in order to implement the desired functionality.

Some details of the protocol, such as different tags for messages in each phase of the competition, are left out of the diagrams, but they are present in our formal specification presented in section 4. The main idea of the protocol is to use a symmetric key $K_{\text{Comp}}$ that is privately

shared among the members of the panel. 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 identity of the panel member in charge of evaluating a particular application, 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 competition. Thus, in transitioning from one phase to another, the manager can hide the links between applicants and panel members. He does so by performing a random mix on the data stored in the cloud before moving to the next phase. Each competition has a public key, that applicants use to encrypt symmetric keys, that in turn serve to encrypt submissions.

**Notation.** In Figures 2-4 we use the following notations:

- $\{m\}_k$ to represent the encryption of a term $m$ with a key $k$;
- $L \leftarrow \mathsf{mix}\ S$ to denote that $L$ is a random permutation of the elements in $S$;
- $n \in_r M$ to denote that $n$ is selected among elements in the set $M$ by a process that can either be random, or depend on some non-deterministic choice of a participant in the protocol;
- $L \leftarrow \{M \mid \phi\}$ to denote that the elements of the $L$ are constructed as specified by an expression $M$, whose sub-expressions have been defined in $\phi$. For example, $\{\{s\}_k \mid s \in M, k \in_r \mathbb{N}\}$ represents the set of encryptions of elements of $M$, with a key that is randomly chosen for each element of $M$;
- "$\mathsf{for\ all}\ T \in S;\ \ \mathcal{E}$" means that for all messages of a set $S$, that should be of the form specified by $T$, the actions in $\mathcal{E}$ should be executed; similarly, "$\mathsf{for\ several}\ T \in S;\ \ \mathcal{E}$" specifies that the actions $\mathcal{E}$ should be executed for a subset of $S$, chosen non-deterministically;
- the diagrams specify the workflow for a particular panel member $P$ and a particular applicant $A$. There is therefore an implicit universal quantification over all panel members and all applicants for a conference.

### 3.1.1 Initialisation (Figure 2).

The initialisation phase consists in four steps: opening, registration, key translation and conflicts declaration.

**Opening.** The manager of competition Comp generates the symmetric key $K_{\mathrm{Comp}}$, a public key $\mathsf{pub}(\mathrm{Comp})$ and a corresponding private key $\mathsf{priv}(\mathrm{Comp})$. The symmetric key is then shared among the members of the panel in a way that does not reveal it to the cloud (see section 3.2.1). Then the manager requests from the cloud the creation of the competition named Comp, sending along the names of the chosen panel members $\mathsf{P}_1, \ldots, \mathsf{P}_\ell$.

**Registration.** An applicant $\mathsf{A}$ should first register his application for the competition, before submitting any documents for evaluation. A registration consists in the identity of the applicant, a fresh number $\lambda$ to identify the application and a fresh symmetric key $k$. The triple $(\lambda, A, k)$ is encrypted with $\mathsf{pub}(\mathrm{Comp})$ and is uploaded to the cloud, along with $\lambda$ and $A$ in clear text. The first role of the key $k$ will be to encrypt documents submitted by $A$, and that are part of the application identified with $\lambda$. The second role of $k$ will be to encrypt the outcome of each round, for the notification that will be sent through the cloud back to the applicant.

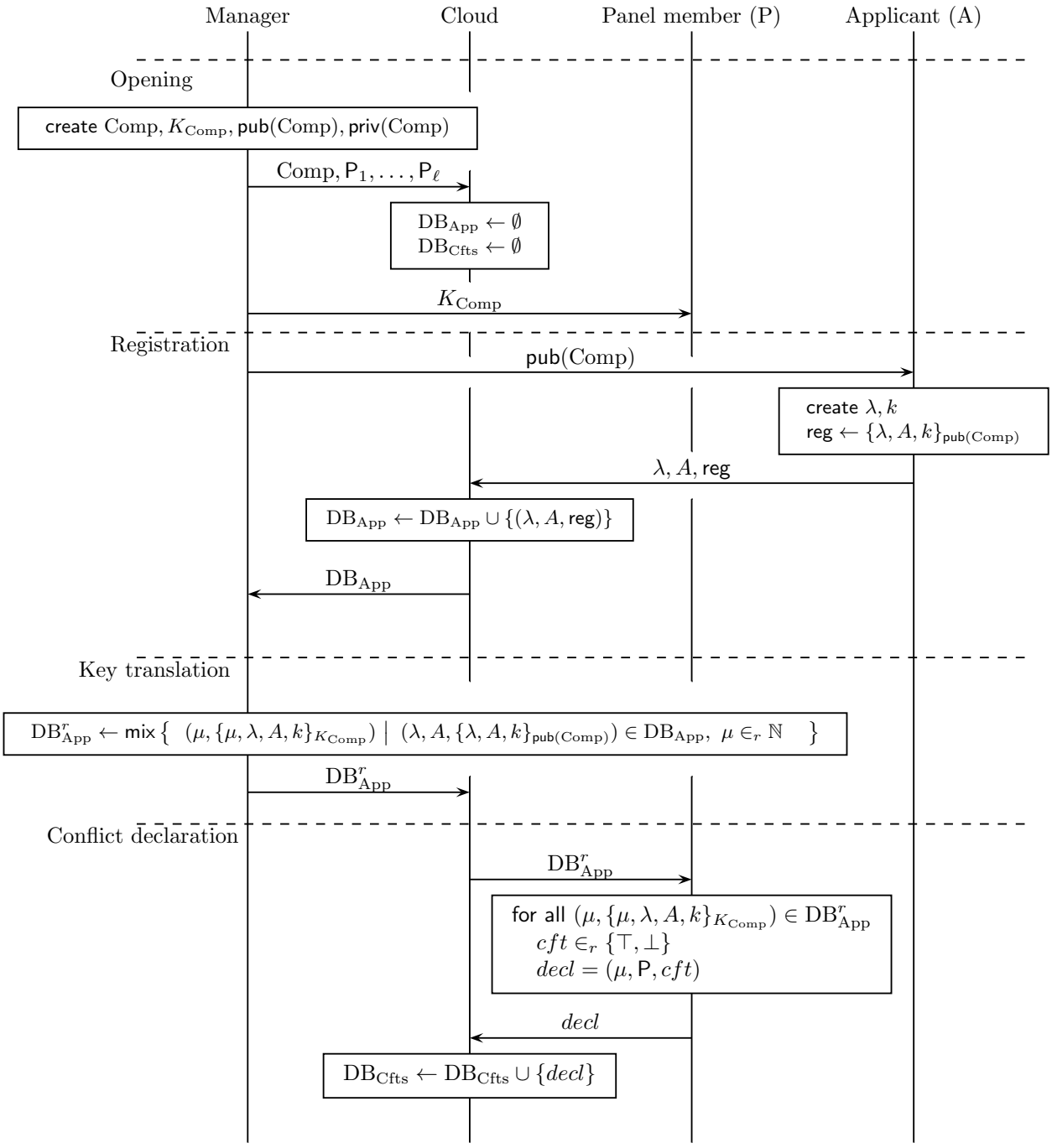


Figure 2: **Initialisation: opening, registration, and conflicts declaration phases**

**Key translation.** The role of this step is to re-encrypt the received applications, replacing the public key of the conference with the symmetric key shared between the panel members. The manager downloads the database of applications, decrypts the application blob using the private key priv(Comp) of the competition and creates a new application blob encrypted with the shared symmetric key $K_{\text{Comp}}$. A new identifier $\mu$ is introduced for each application, its role being that of a public and anonymous identifier for that application. Finally, the database of blobs, together with their identifiers, is mixed, resulting in an anonymised databased $\text{DB}^r_{\text{App}}$, that the manager uploads back to the cloud.

**Conflicts declaration.** The panel members can download any anonymised application from $\text{DB}^r_{\text{App}}$ and decrypt the blob to identify the corresponding author. They can then declare either the existence if a conflict of interest (represented by the constant $\top$), or the absence of a conflict of interest (represented by the constant $\bot$). The declaration is attached to the anonymous identifier $\mu$, so that the cloud can subsequently filter out the evaluations that are not to be received by panel members with conflicts of interest.

### 3.1.2 A round (Figures 3 and 4).

A competition may have several rounds, each round consisting of five steps: submission, assignment, evaluation, discussion and notification.

**Submission.** An applicant A creates a document $s^i$ for submision at round $i$. The submission $s^i$ is uploaded to the cloud encrypted with the key $k$ generated at the registration phase. The identifier $\lambda$ is used to link the submission $s^i$ with a registered application, so that the panel members can access all submissions that are part of the same application.

**Assignment.** After applicants have uploaded their submissions for a round $i$, the panel members can start bidding for applications. A positive bid is represented by the constant $\top$. The bid and the identity of the reviewer are encrypted and the resulting blob is associated with the identifier $\mu$ corresponding to the respective application. This information is sent to the manager through the cloud. The manager can then assign applications to reviewers that have bid for them, by uploading triples of the form $(\mu, \mathsf{P}, \mathsf{assigned\_blob})$ back to the cloud. The public information $\mu, \mathsf{P}$ allows the cloud to manage the information flow, by sending only necessary data to reviewers. The private information contained in assigned_blob allows reviewers to verify that the assignment has been indeed performed by the chair.

**Evaluation.** In the evaluation step, the panel members obtain from the cloud the applications that have been assigned to them. They create an evaluation which is attached to the application, and a corresponding encrypted blob is sent back to the cloud. Note that the cloud is told to what identifier $\mu$ this encrypted evaluation refers. This allows the cloud to manage the data flow, without being able to link $\mu$ with $\lambda$, and hence the panel member with the applicant.

**Discussion.** The evaluations of each application can be downloaded by panel members (except those in conflict) for further discussion. Each member of the panel can read a submitted evaluation and the ongoing discussion $D$ and add a comment $d$ to it. This is implemented through the cloud and through decryptions/encryptions exactly as in the evaluation step. The discussion is iterative: the panel members can add several comments for an application within a single round.

**Notification.** For each application, the manager creates a notification including the outcome of the current round $i$ and the evaluations written by the panel members. It can also include instructions for the next round. The outcome is encrypted with the applicant's symmetric key $k$ (chosen at registration time). The encrypted notifications along with the submission identifier

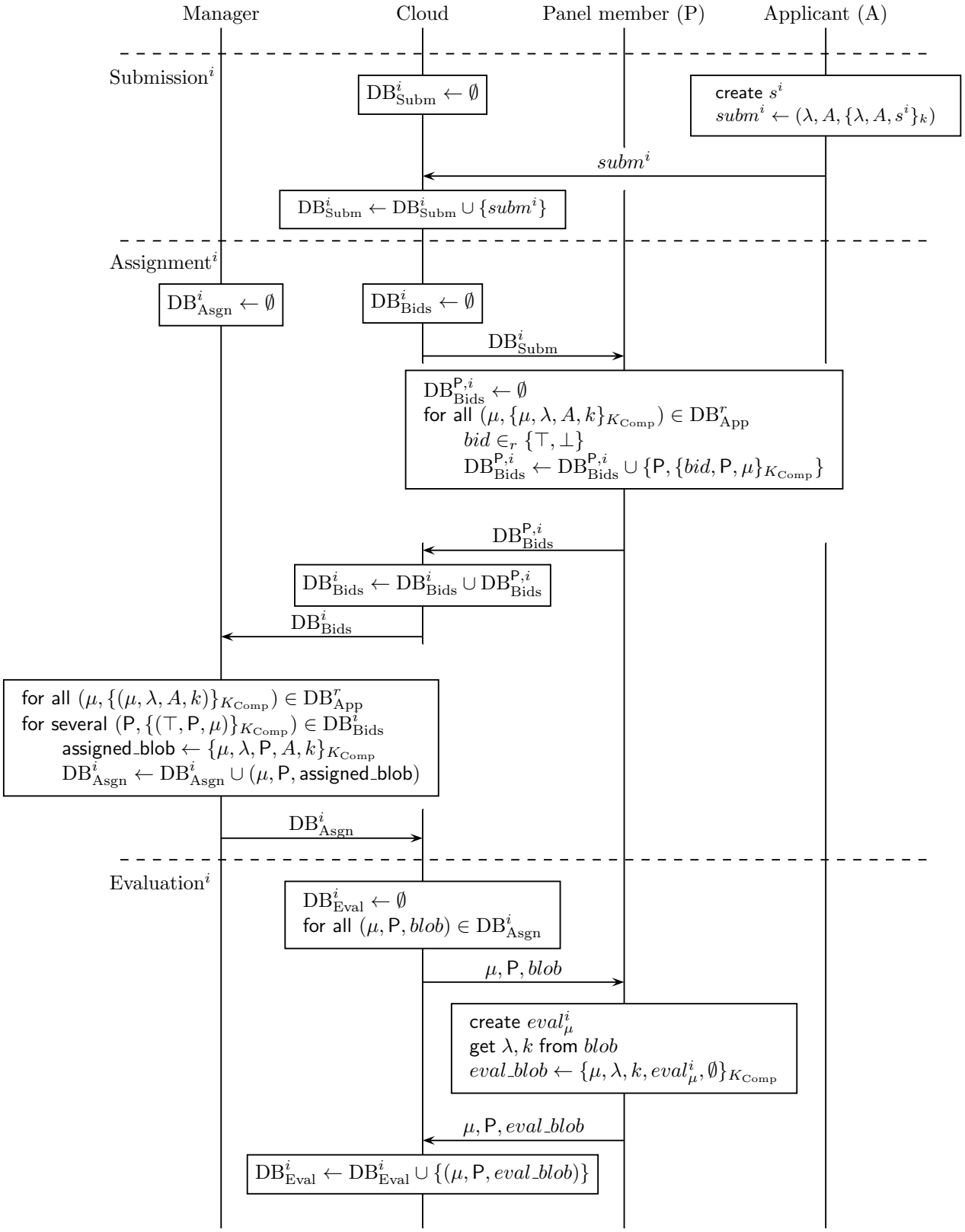


Figure 3: **Round $i$: submission, assignment, and evaluation phases**

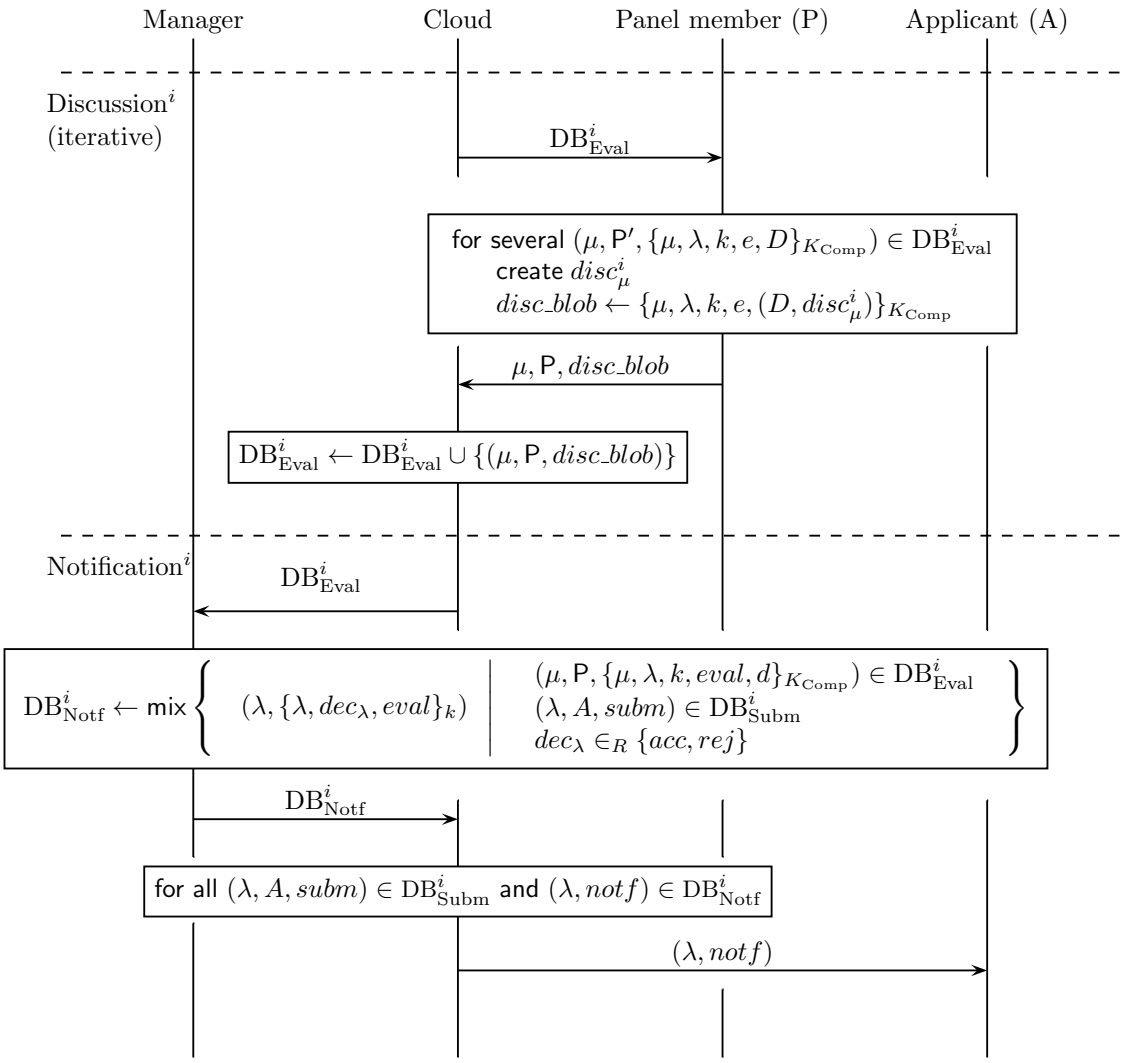


Figure 4: **Round $i$: discussion and notification phases**

$\lambda$ are mixed and uploaded to the cloud. This allows the cloud to manage the information flow without compromising the privacy of the applicants.

## 3.2 Discussion

### 3.2.1 Distribution of the competition symmetric key.

The privacy properties of our protocol rely on the sharing of a symmetric key $K_{\text{Comp}}$ among the members of the panel in such a way that the cloud does not get hold of $K_{\text{Comp}}$. Here we suggest a few possible solutions in the context of our applications, reflecting different trade-offs between security and usability. Our protocol is independent of which of the three solutions is adopted:

*(1) Public keys.* Each member of the panel may be expected to have a public key. Then, the symmetric key can be encrypted with each of the chosen panel member's public key and uploaded to the cloud. The distribution can be made more flexible and efficient by relying on key distribution protocols like [17]. An important issue in this setting is the authentication of public keys of members invited to participate in the competition panel. This may be done either relying on a hierarchical certification model such as *PKI* or, what is more probable in the case of competition management, on a distributed web of trust, such as that of *PGP*.

*(2) Token.* In this solution, each member of the panel generates a symmetric key $k_{\mathsf{P}}$ and uploads $\{k_{\mathsf{P}}\}_{\mathsf{pub}(\text{Comp})}$ to the cloud. Then, the manager sends $\{K_{\text{Comp}}\}_{k_{\mathsf{P}}}$ to the panel member using a channel that is outside the control of the cloud. He does this by checking the evaluator's authenticated email address and sends $\{K_{\text{Comp}}\}_{k_{\mathsf{P}}}$ to that address. The panel member then decrypts this token to obtain $K_{\text{Comp}}$. In this case, even if the cloud has access later to a panel member's email, it cannot compromise the privacy properties that our protocol ensures.

*(3) Email.* If we assume that email infrastructure is not in the control of the cloud service provider that hosts the competition management system (as is most probably the case), the key $K_{\text{Comp}}$ could be sent to panel members directly by email. In that case, if the email of a panel member is compromised later on, its privacy for the competition Comp is also compromised. Note that it is only the key $K_{\text{Comp}}$ that must be sent by email, all the rest of the protocol being executed through the cloud.

### 3.2.2 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 user privacy. In particular, the link between the sender of a message (*e.g.* the applicant behind a submission) and the end receiver of this message (*e.g.* the panel member evaluating this submission) 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 competition, its applicants, submissions, and panel members. This can be achieved by adding code which extracts this information during the manipulations performed by the manager's browser. For example, along with the computation of $\text{DB}^{r}_{\text{Notf}}$ in Figure 4, the manager could also compute the average score $as_{\mu} = (s_1 + \cdots + s_{n_{\mu}})/n_{\mu}$ for each submission and return the vector $(as_{\mu})_{\mu}$ to the cloud. (Such optional features must be carefully designed to avoid breaking the security properties, as those features are not considered in our formal model in Section 4.)

### 3.2.3 Efficiency and usability

It may seem that there is a considerable amount of work to be done by the manager, especially in the transition between phases. As we discuss in section 5, this does not have to be evident to

the manager. Our experiments with our prototype implementation of a competition management system following this protocol show that the browser can transparently execute the protocol.

# 4 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 [35], the public-key Kerberos protocol [21], the PKCS#11 API [18], or the BAC protocol in e-Passports [23]. In this section, we formally show that our protocol satisfies the claimed security properties. The formal verification of the protocol has been done automatically using the ProVerif tool [14, 16], and the corresponding ProVerif scripts are available online [37]. The verification requires a rigorous description of the protocol in a process calculus as well as formal definitions of the desired properties, each discussed in detail in the following sections.

## 4.1 The process calculus

The ProVerif calculus [14, 16] is a language for modelling distributed systems and their interactions. It is a dialect of the applied pi calculus [6]. In this section, we briefly review the basic ideas and concepts of the ProVerif calculus.

### 4.1.1 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* $\Sigma$, 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. The ProVerif calculus also uses tuples of messages $(M_1, \ldots, M_n)$, keeping the obvious projection rules implicit.

We use constructors and destructors to model the cryptographic primitives of *symmetric-key* and *public-key encryption*. Specifically, we consider the signature

$$\Sigma_{\mathsf{ciph}} = \{\mathsf{senc}_{/3}, \mathsf{sdec}_{/2}, \mathsf{pub}_{/1}, \mathsf{aenc}_{/3}, \mathsf{adec}_{/2}\}$$

where $\mathsf{senc}$ (*resp.* $\mathsf{aenc}$) is a constructor of arity 3 that models the randomised shared-key (*resp.* randomised public-key) encryption primitive, $\mathsf{sdec}$ (*resp.* $\mathsf{adec}$) is the corresponding destructor of arity 2, and $\mathsf{pub}$ is a constructor of arity 1 that models the public key associated to the private key given in argument.

The semantics of destructors in $\Sigma_{\mathsf{ciph}}$ is given by the following rules

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

We model the probabilistic shared-key encryption of the message $m$ with the key $k$ by $\mathsf{senc}(k, r, m)$, where $r$ models the random input for the encryption algorithm. The probabilistic public-key encryption of the message $m$ with the pubic key $\mathsf{pub}(k)$ is modeled 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

$N$

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

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

### 4.1.2 Processes

Processes are built according to the grammar given below, where $M, N$ are terms, $D$ is a term evaluation and $n$ is a name.

| $P, Q, R ::=$ | processes |
|---|---|
| $0$ | null process |
| $P \mid Q$ | parallel composition |
| $!P$ | replication |
| $\text{new}\, n; P$ | name restriction |
| let $M = D$ in $P$ else $Q$ | term evaluation |
| $\text{in}(N, M); P$ | message input |
| $\text{out}(N, M); P$ | 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 $P$.

We also adopt a syntactic sugar of ProVerif that allows pattern matching. In the atomic action $\text{in}(M, N)$, the term $N$ can have subterms preceded by $=$, which implies a matching constraint on the input. For example, the input action $\text{in}(c, (= a, y)))$ expects a pair whose first element must be equal to the name $a$, and whose second one is assigned to the variable $y$.

*Semantics.* The possible actions of the environment are captured by *evaluation contexts*. A *context* is a process with a hole. For a context $C[\_]$, we denote by $C[A]$ the process obtained by filling its 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 $\alpha$-conversion of bound names and bound variables, application of evaluation contexts and of standard rules (see [16] for full definition) that capture the associativity, the commutativity and the interplay of $\mid$ and $\nu$.

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

$$\begin{aligned} &\text{out}(N, M').P \mid \text{in}(N, M).Q \rightarrow P \mid Q\sigma, \text{ if } \exists\sigma \; s.t. \; M\sigma = M' \\ &\text{let } M = D \text{ in } P \text{ else } Q \rightarrow P\sigma, \text{ if } \exists M' \; \exists\sigma \; s.t. \; D \Downarrow M' \;\&\; M\sigma = M' \\ &\text{let } M = D \text{ in } P \text{ else } Q \rightarrow Q, \text{ otherwise} \end{aligned}$$

We write $\rightarrow^*$ for an arbitrary (possibly zero) number of internal reductions.

### 4.1.3 Observational equivalence

For a process $A$ and a name $c$, we write $A \Downarrow c$ when $A$ can evolve into a process that can send a message on $c$, that is, when $A \rightarrow^* C[\text{out}(c, M).P]$ for some term $M$, some evaluation context $C[\_]$ that does not bind $c$ (i.e. $c \notin \text{bn}(C[\_])$), and some process $P$.

**Definition 1** *[6] 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 $B \Downarrow c$, for any channel $c$;*
2. *for any process $A'$, if $A \rightarrow^* A'$ then, for some process $B'$, 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.

## 4.2 The protocol in the process calculus

*Syntactic sugars.* To improve readability, we sometimes replace the construct " $\mathsf{let}\ U = V\ \mathsf{in}\ P\ \mathsf{else}\ Q$" with " $\mathsf{if}\ \ U = V\ \mathsf{then}\ P\ \mathsf{else}\ Q$". Furthermore, we will occasionally combine several instructions $\mathsf{if\ U = V\ then\ P\ else\ Q}$ into a single instruction, by gathering the tests into a single disjunctive formula.

We will also make use of the following syntactic sugar of ProVerif: in the term $M$ from the expressions "let $M = D$ in $P$ else $Q$" and "$\mathrm{in}(c, M)$" one is allowed to have subterms of the form "$=N$", for some term $N$. This emphasises the fact that the term $N$ has been previously defined in the process, and therefore the instantiation of $M$ has to be consistent with the previous instantiation of $N$. Formally, this semantics can be expressed by a combination of fresh variables and equality tests. For instance, the process $\mathrm{in}(c, x); \mathrm{in}(c, = x); P$ means that the second input on the channel $c$ must be equal to the first input. Relying on the original syntax, this becomes: $\mathrm{in}(c, x); \mathrm{in}(c, y); \mathsf{if}\ x = y\ \mathsf{then}\ P$.

### 4.2.1 Message tags and communication channels

We introduce the set of constants $\Sigma_{\mathsf{tags}} \cup \Sigma_{\mathsf{nums}}$, where

$$\begin{array}{rcl} \Sigma_{\mathsf{tags}} & = & \{\mathsf{reg}, \mathsf{subm}, \mathsf{initround}, \mathsf{cft}, \mathsf{bid}, \mathsf{assigned}, \mathsf{evl}, \mathsf{dsc}, \mathsf{ntf}\} \\ \Sigma_{\mathsf{nums}} & = & \{\mathsf{zero}, \mathsf{one}, \mathsf{two}\} \end{array}$$

The constants in $\Sigma_{\mathsf{tags}}$ correspond to tags used to label messages in different phases of the protocol. The constants in $\Sigma_{\mathsf{nums}}$ represent small numbers, used for instance to declare bids or conflicts. The initial round of the protocol is represented by the constant $\mathsf{zero}$. Then, if a round is represented by the term $t$, the next round is represented by the term $\mathsf{succ}(t)$, where $\mathsf{succ}$ is an unary function symbol.

Puting all the function symbols together, we obtain the signature

$$\Sigma_{\mathsf{CC}} = \Sigma_{\mathsf{ciph}} \cup \Sigma_{\mathsf{tags}} \cup \Sigma_{\mathsf{nums}} \cup \{\mathsf{succ}\}$$

where $\Sigma_{\mathsf{ciph}}$ has been introduced in section 4.1.1. We also consider the rewrite rules for symmetric and public-key encryption that are introduced in section 4.1.1.

To model communication channels, we consider the following names:

- $c$ - a public name that represents the communication channel with the cloud
- $c_{\mathsf{shk}}$ - a private name that represents a private channel that is used to distribute the symmetric key shared by the panel members
- $c_{\mathsf{pbk}}$ - a private name that represents an authenticated channel where the applicants can obtain the public key of the competition
- $c_{\mathsf{round}}$ - a private name for a channel used by the manager to announce the current round

**Communication through the cloud.** Several points are worthy of mention about our modeling of the cloud channel:

1. The name $c$ is public, which corresponds to the fact that the attacker can intercept and inject any messages that pass through the cloud.

2. The specification of the protocol requires sets of ciphertexts, e.g. $S = \{s_1, \ldots, s_n\}$, to be communicated through the cloud. For instance, in the key translation phase of diagram 1, the chair has to download a list of ciphertexts and upload another list. Since the formal model has to take into account all the possible values of $n \geq 1$, we represent the transmission of $S$ by a several separate transmissions of $s_1, \ldots, s_n$. The message tags will ensure that, upon reception, each ciphertext $s_1, \ldots, s_n$ will be handled consistently as belonging to the set $S$.

3. The semantics of ProVerif (i.e. internal reduction in section 4.1.2) does not specify how any non-determinism that results from multiple inputs and outputs on the channel $c$ is resolved. In fact, ProVerif explores all the possible ways of scheduling the input and output actions, as long as they are allowed by the tests performed by the receiving agents. Therefore, if the protocol is secure in the ProVerif model, an implementation is secure in presence of any particular scheduler that cannot control the actions of the honest agents (the chair in particular).

### 4.2.2 The applicant process

We model the role of an applicant by the following process:

$$
\begin{array}{rcl}
\mathsf{Applicant} & = & \mathrm{new}\,\mathsf{idA};!\,\mathsf{AppReg} \\
\mathsf{AppReg} & = & \textbf{(* Create applicant data *)} \\
 & & \mathrm{new}\,\lambda; \mathrm{new}\,k; \mathrm{new}\,r_1; \\
 & & \textbf{(* Obtain the public key of the conference *)} \\
 & & \mathrm{in}(c_{\mathsf{pbk}}, x_{\mathsf{pbk}}); \\
 & & \textbf{(* Upload registration data *)} \\
 & & \mathrm{out}(c, (\lambda, \mathsf{idA}, \mathsf{aenc}(x_{\mathsf{pbk}}, r_1, (\mathsf{reg}, \lambda, \mathsf{idA}, k)))); \\
 & & !\,\mathsf{Submission} \\
\mathsf{Submission} & = & \textbf{(* Wait for the next round, create submission data *)} \\
 & & \mathrm{in}(c_{\mathsf{round}}, x_{\mathsf{round}}); \mathrm{new}\,s; \mathrm{new}\,r_2; \\
 & & \textbf{(* Upload an encrypted submission *)} \\
 & & \mathrm{let}\ \mathsf{submission} = (\lambda, \mathsf{idA}, \mathsf{senc}(k, r_2, ((\mathsf{subm}, x_{\mathsf{round}}), \lambda, \mathsf{idA}, s))) \ \mathrm{in} \\
 & & \mathrm{out}(c, \mathsf{submission})
\end{array}
$$

An applicant can register to several competitions (the process !AppReg). During registration, the applicant generates an identifier $\lambda$ and a submission key $k$, and obtains from the authenticated channel $c_{\mathsf{pbk}}$ the public key $x_{\mathsf{pbk}}$ of the conference. The ciphertext $\mathsf{aenc}(x_{\mathsf{pbk}}, r, k)$ is then uploaded to the cloud, using the public cloud channel $c$. After registration, the applicant can submit any number of applications (the process !Submission). For each submission, the applicant first determines the current round of the competition (stored in $x_{\mathsf{round}}$), creates a new submission $s$, and then uploads to the cloud the corresponding ciphertext for this submission, that is $\mathsf{senc}(k, r_2, ((\mathsf{subm}, x_{\mathsf{round}}), \lambda, \mathsf{idA}, s))$. Note that we attach a tag $(\mathsf{subm}, x_{\mathsf{round}})$ to the plaintext in order to specify that it corresponds to a submission made in round $x_{\mathsf{round}}$.

### 4.2.3 The manager process

We model the manager of the competition by the following process:

$$\mathsf{Manager} \quad = \quad \mathsf{Init} \mid !\mathsf{Round}$$

where Init models the actions of the manager in the initialisation phase (figure 2 from section 3) and Round models the actions of the manager in each round of the competition (figures 3 and

4 from section 3). In the initialisation phase, the manager creates the new keys $k_{\mathsf{shk}}, k_{\mathsf{conf}}$ for that conference, shares the symmetric key $k_{\mathsf{shk}}$ with the panel members over the private channel $c_{\mathsf{shk}}$, and communicates the public key of the conference $\mathsf{pub}(k_{\mathsf{conf}})$ to applicants and to the cloud:

$$\mathsf{Init} \;=\; \mathrm{new}\, k_{\mathsf{shk}}; \mathrm{new}\, k_{\mathsf{conf}}; (!\mathrm{out}(c_{\mathsf{shk}}, k_{\mathsf{shk}}) \mid !\mathrm{out}(c_{\mathsf{pbk}}, \mathsf{pub}(k_{\mathsf{conf}})) \mid \mathrm{out}(c, \mathsf{pub}(k_{\mathsf{conf}})) \mid \mathsf{KeyTrans})$$

In addition, the process KeyTrans models the key translation step, that is performed after the registration phase is completed:

$$\begin{aligned}
\mathsf{KeyTrans} \;=\;& \textbf{(* Download registration data from the cloud *)}\\
& \mathrm{in}(c, (x_\lambda, x_{\mathsf{idA}}, \mathsf{key_blob}));\\
& \mathrm{let}\ (= \mathsf{reg}, = x_\lambda, = x_{\mathsf{idA}}, x_{\mathsf{key}}) = \mathsf{adec}(k_{\mathsf{conf}}, \mathsf{key_blob})\ \mathrm{in}\\
& \textbf{(* Create an anonymous identifier } \mu \textbf{ *)}\\
& \mathrm{new}\, \mu; \mathrm{new}\, r;\\
& \textbf{(* Upload the key translation to the cloud *)}\\
& \mathrm{let}\ \mathsf{key_tran} = (\mu, \mathsf{senc}(k_{\mathsf{shk}}, r, ((\mathsf{initround}, \mathsf{zero}), \mu, x_\lambda, x_{\mathsf{idA}}, x_{\mathsf{key}})))\ \mathrm{in}\\
& \mathrm{out}(c, \mathsf{key_tran});
\end{aligned}$$

In KeyTrans, the manager downloads a registration tuple from the cloud, creates a new nonce $\mu$ to anonymously identify this tuple and re-encrypts the key blob with the key $k_{\mathsf{shk}}$. The resulting anonymized key translation, assigned to the variable key_tran, is uploaded to the cloud.

Note that the specification of the protocol also postulates that the manager should mix the ciphertexts before uploading the key translations back to the cloud. We do not explicitly model the mix as an action of the chair, but its semantics is realized by the non-deterministic nature of outputs and parallel composition in ProVerif.

Each round of the procotol, described in diagrams 2 and 3, consists of several phases: submission, assignement, evaluation, discussion, notification. The manager only participates in the assignement phase and in the notification phase, and also manages the transition from one round to another. This is modeled by the processes RoundTr, Assign and Notify described below:

$$\begin{aligned}
\mathsf{Round} \;=\;& \mathsf{RoundTr} \mid !\mathsf{Assign} \mid !\mathsf{Notify}\\
\mathsf{RoundTr} \;=\;& \mathrm{out}(c_{\mathsf{round}}, \mathsf{zero}) \mid !\mathrm{in}(c_{\mathsf{round}}, x); \mathrm{out}(c_{\mathsf{round}}, x) \mid !\mathrm{in}(c_{\mathsf{round}}, x); \mathrm{out}(c_{\mathsf{round}}, \mathsf{succ}(x))\\
\mathsf{Assign} \;=\;& \textbf{(* Download an application blob and a bid blob from the cloud*)}\\
& \mathrm{in}(c, (x_\mu, \mathsf{app_blob})); \mathrm{in}(c, (x_{\mathsf{idR}}, \mathsf{bid_blob}));\\
& \textbf{(* Decrypt and check that the bid corresponds to the application *)}\\
& \mathrm{let}\ ((= \mathsf{initround}, = x_{\mathsf{round}}), = x_\mu, x_\lambda, x_{\mathsf{idA}}, x_{\mathsf{key}}) = \mathsf{sdec}(k_{\mathsf{shk}}, \mathsf{subm_blob})\ \mathrm{in}\\
& \mathrm{let}\ ((= \mathsf{bid}, = x_{\mathsf{round}}), = \mathsf{one}, = x_{\mathsf{idR}}, = x_\mu) = \mathsf{sdec}(k_{\mathsf{shk}}, \mathsf{bid_blob})\ \mathrm{in}\\
& \textbf{(* Construct an assignement blob and upload it to the cloud *)}\\
& \mathrm{new}\, r;\\
& \mathrm{let}\ \mathsf{assigned_blob} = \mathsf{senc}(k_{\mathsf{shk}}, r, ((\mathsf{assigned}, x_{\mathsf{round}}), x_\mu, x_\lambda, x_{\mathsf{idR}}, x_{\mathsf{idA}}, x_{\mathsf{key}}))\ \mathrm{in}\\
& \mathrm{let}\ \mathsf{assignement} = (x_\mu, x_{\mathsf{idR}}, \mathsf{assigned_blob})\ \mathrm{in}\ \mathrm{out}(c, \mathsf{assignement})\\
\mathsf{Notify} \;=\;& \textbf{(* Download a reviewed submission and make a notification decision *)}\\
& \mathrm{in}(c, (y_\mu, y_{\mathsf{blob}})); \mathrm{in}(c, y_{\mathsf{ntf}}); \mathrm{new}\, r;\\
& \mathrm{let}\ ((= \mathsf{dsc}, = x_{\mathsf{round}}), = y_\mu, y_\lambda, y_{\mathsf{key}}, y_{\mathsf{eval}}, y_{\mathsf{disc}}) = \mathsf{sdec}(k_{\mathsf{shk}}, y_{\mathsf{blob}})\ \mathrm{in}\\
& \textbf{(* Send the notification to the author through the cloud *)}\\
& \mathrm{let}\ \mathsf{notification} = (y_\lambda, \mathsf{senc}(y_{\mathsf{key}}, r, ((\mathsf{ntf}, x_{\mathsf{round}}), y_\lambda, y_{\mathsf{ntf}}, y_{\mathsf{eval}}))\ \mathrm{in}\ \mathrm{out}(c, \mathsf{notification})
\end{aligned}$$

In the Assign process, the chair downloads an encrypted application and an encrypted bid from the cloud, verifies if the bid corresponds to the considered application and creates a new blob (stored in assigned_blob) that assigns the reviewer to the desired application. The assignment blob is uploaded to the cloud, annotated with the pair $(x_\mu, x_{\mathsf{idR}})$, so that the cloud can forward the assigned blob to the corresponding reviewer $x_{\mathsf{idR}}$. In the Notify process, the manager obtains the evaluations and discussions from the application blob, makes a decision about the application, and sends the notification to the author. The notification is sent through the cloud and it is encrypted with the submission key that is associated to the considered application.

### 4.2.4 The evaluator process

For every conference, an evaluator first registers its identity with the cloud and obtains the shared symmetric key of the conference from the chair. Next, the evaluator can obtain applications from the cloud and participate in a number of evaluation rounds, as specified in the Evaluator process below.

$$
\begin{array}{rcl}
\mathsf{Evaluator} & = & \mathrm{new}\,\mathsf{idR};\,!\,\mathsf{RevReg} \\
\mathsf{RevReg} & = & \textbf{(* Register to the cloud and obtain } k_{\mathsf{shk}} \textbf{ from the chair *)} \\
 & & \mathrm{out}(c, \mathsf{idR}); \mathrm{in}(c_{\mathsf{shk}}, x_{\mathsf{shk}}); \\
 & & \textbf{(* Declare conflicts and execute several rounds of reviewing *)} \\
 & & !\mathsf{Conflicts} \mid !\ (\ \mathrm{in}(c_{\mathsf{round}}, x_{\mathsf{round}}); (!\ \mathsf{Bids} \mid !\ \mathsf{Review} \mid !\ \mathsf{Discussion})) \\
\mathsf{Conflicts} & = & \textbf{(* Download encrypted applications and determine conflicts *)} \\
 & & \mathrm{in}(c, (x_\mu, x_{\mathsf{blob}})); \\
 & & \mathrm{let}\ ((= \mathsf{initround}, = \mathsf{zero}), = x_\mu, x_\lambda, x_{\mathsf{idA}}, x_{\mathsf{key}}) = \mathsf{sdec}(x_{\mathsf{shk}}, x_{\mathsf{blob}})\ \mathrm{in}\ \mathrm{in}(c, x_{\mathsf{cft}}); \\
 & & \textbf{(* Upload declared conflicts to the cloud *)} \\
 & & \mathrm{out}(c, (x_\mu, \mathsf{idR}, x_{\mathsf{cft}})) \\
\mathsf{Bids} & = & \textbf{(* Download encrypted applicatons *)} \\
 & & \mathrm{in}(c, (x_\mu, x_{\mathsf{blob}})); \\
 & & \mathrm{let}\ ((= \mathsf{initround}, = x_{\mathsf{round}}), = x_\mu, x_\lambda, x_{\mathsf{idA}}, x_{\mathsf{key}}) = \mathsf{sdec}(x_{\mathsf{shk}}, x_{\mathsf{blob}})\ \mathrm{in} \\
 & & \textbf{(* Upload an encrypted bid *)} \\
 & & \mathrm{in}(c, x_{\mathsf{bid}}); \mathrm{new}\, r; \\
 & & \mathrm{let}\ \mathsf{bid\_blob} = (\mathsf{idR}, \mathsf{senc}(x_{\mathsf{shk}}, r, ((\mathsf{bid}, x_{\mathsf{round}}), x_{\mathsf{bid}}, \mathsf{idR}, x_\mu)))\ \mathrm{in}\ \mathrm{out}(c, \mathsf{bid\_blob}) \\
\mathsf{Evaluate} & = & \textbf{(* Download an assigned application *)} \\
 & & \mathrm{in}(c, (x_\mu, = \mathsf{idR}, x_{\mathsf{blob}})); \\
 & & \mathrm{let}\ ((= \mathsf{assigned}, = x_{\mathsf{round}}), = x_\mu, x_\lambda, = \mathsf{idR}, x_{\mathsf{idA}}, x_{\mathsf{key}}) = \mathsf{sdec}(x_{\mathsf{shk}}, x_{\mathsf{blob}})\ \mathrm{in} \\
 & & \textbf{(* Create an evaluation for the submission ...*)} \\
 & & \mathrm{new}\, eval; \mathrm{new}\, r; \\
 & & \textbf{(* ... and upload it encrypted to the cloud *)} \\
 & & \mathrm{let}\ \mathsf{eval\_blob} = (x_\mu, \mathsf{idR}, \mathsf{senc}(x_{\mathsf{shk}}, r, ((\mathsf{evl}, x_{\mathsf{round}}), x_\mu, x_\lambda, x_{\mathsf{key}}, eval, \mathsf{zero})))\ \mathrm{in} \\
 & & \mathrm{out}(c, \mathsf{eval\_blob}) \\
\mathsf{Discussion} & = & \textbf{(* Download any application *)} \\
 & & \mathrm{in}(c, (y_\mu, y_{\mathsf{blob}})); \\
 & & \mathrm{let}\ ((= \mathsf{evl}, = x_{\mathsf{round}}), = y_\mu, y_\lambda, y_k, y_r, y_s, y_d) = \mathsf{sdec}(x_{\mathsf{shk}}, y_{\mathsf{blob}})\ \mathrm{in} \\
 & & \textbf{(* Extend the discussion and upload the revised application *)} \\
 & & \mathrm{new}\, disc; \mathrm{new}\, r; \\
 & & \mathrm{let}\ \mathsf{disc\_blob} = (y_\mu, \mathsf{idR}, \mathsf{senc}(x_{\mathsf{shk}}, r, ((\mathsf{dsc}, x_{\mathsf{round}}), y_\mu, y_\lambda, y_k, y_e, (y_d, d))))\ \mathrm{in} \\
 & & \mathrm{out}(c, \mathsf{disc\_blob}).
\end{array}
$$

In the process Bid, the evaluator obtains an application blob from the cloud, decrypts it using the shared key of evaluators, makes a decision whether to bid or not for the paper, and uploads the encrypted bid back to the cloud. Similarly, in the process Evaluate, the evaluator downloads an encrypted application, verifies that it has been assigned to him, and uploads back to the cloud an encrypted evaluation. Finally, the Discussion process models the fact that any evaluator can discuss any paper, even if it has not been assigned to him.

### 4.2.5 Model of the protocol

The protocol is modeled by the process CC (Confidentiality in the Cloud), where we put together all the elements that have been previously described:

$$\mathsf{CC} \quad = \quad \mathrm{new}\, c_{\mathsf{shk}};\ \mathrm{new}\, c_{\mathsf{pbk}};\ \mathrm{new}\, c_{\mathsf{round}}; (!\ \mathsf{Manager} \mid !\ \mathsf{Evaluator} \mid !\ \mathsf{Applicant})$$

Note that we do not fix the number of competitions, managers, evaluators or applicants: our results hold for any number of instances of the protocol. We declare $c_{\mathsf{shk}}$ to be private, because the

corresponding channel is assumed to be private. The construct new $c_{\mathsf{pbk}}$ ensures that the data sent over $c_{\mathsf{pbk}}$ is authentic (the intruder cannot write data on $c_{\mathsf{pbk}}$), which allows applicants to obtain the correct public key of the competition from Manager. In addition to sending the public key on $c_{\mathsf{pbk}}$, the manager uploads the public key to the cloud (on the public channel of the cloud $c$), thus making it public. We do not model explicitly the actions of the cloud, because all its possible actions, including those that are required for the functionality of our protocol, are considered by the universally quantified evaluation, that is implicit in the definition of our properties (cf sections 4.1.3 and 4.3).

## 4.3 The properties in the process calculus

In this section we explain how the desired secrecy and unlinkability properties are formally defined in the process calculus. We define both secrecy and unlinkability in terms of observational equivalence, extending the classical approach of [42, 5, 26], and adapting it to our context.

When we prove an equivalence, we prove that a certain behaviour of the left process can be indistinguishably mimicked by the right process, and vice versa. A concern when using equivalence to model security properties arises if there is more non-determinism in our model than is permitted in reality. If this happens, it might be that some of the behaviours on one side required to mimic behaviours on the other side are possible in the model but not possible in reality; this could imply that the equivalence holds in the model but not in reality. This is an instance of the more general problem that there is a gap between the model and reality. We try to minimise that gap. Specifically, we ensure that we don't introduce any spurious non-determinism for the witness applications and the witness reviews in our model. We ensure that the only non-determinism is the communication of messages within each phase, which is also non-determinism allowed in reality.

To express the desired security properties, we will need to consider particular applicants and evaluators 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 consider the context:

$$\mathsf{CC}[\_] \;=\; \mathrm{new}\, c_{\mathsf{shk}};\ \mathrm{new}\, c_{\mathsf{pbk}};\ \mathrm{new}\, c_{\mathsf{round}};\ (!\ \mathsf{Manager} \mid !\ \mathsf{Evaluator} \mid !\mathsf{Applicant} \mid \_\ )$$

To express particular applicants, evaluators or actions of the manager, we will define so-called *witness* processes, which are instances of the corresponding general processes. To be able to define various instances in a generic way, we will consider parametrized processes, denoted by $P(a_1, \ldots, a_n)$, where $a_1, \ldots, a_n$ are free names. The idea is that from the specification of $P(a_1, \ldots, a_n)$, we can derive several instances of $P$, by replacing $a_1, \ldots, a_n$ with desired paramaters. The process obtained from $P(a_1, \ldots, a_n)$ by replacing $a_1, \ldots, a_n$ with $b_1, \ldots, b_n$ is denoted by $P(b_1, \ldots, b_n)$.

### 4.3.1 Submission secrecy

Our protocol ensures the secrecy of submissions made by applicants. To express this property formally, we consider a witness applicant process where the identity of the applicant and the content of one of the submissions is fixed. We denote this (parametrized) process by $\mathsf{Applicant}^{\mathsf{s}}(\mathsf{idA}_{\mathsf{w}}, s_{\mathsf{w}})$, and it is defined as follows, where we present in bold font the parts of the process that differ from the general Applicant process:

$$
\begin{array}{rcl}
\mathsf{Applicant}^{\mathsf{s}}(\mathsf{idA}_{\mathsf{w}}, \mathsf{s}_{\mathsf{w}}) & = & \textbf{(* We fix the identity of the author *)} \\
& & \textbf{let idA} = \mathbf{idA_{wtn}} \textbf{ in } ! \, \mathbf{AppReg^{s}} \\
\mathsf{AppReg}^{\mathsf{s}} & = & \mathrm{new}\, \lambda; \mathrm{new}\, k; \mathrm{new}\, r_1; \mathrm{in}(c_{\mathsf{pbk}}, x_{\mathsf{pbk}}); \\
& & \mathrm{out}(c, (\lambda, \mathsf{idA}, \mathsf{aenc}(x_{\mathsf{pbk}}, r_1, (\mathsf{reg}, \lambda, \mathsf{idA}, k)))); \\
& & \textbf{(* General submissions and one witness submission *)} \\
& & (!\mathsf{Submission}) \mid \mathbf{Submission^{s}} \\
\mathsf{Submission}^{\mathsf{s}} & = & \mathrm{in}(c_{\mathsf{round}}, x_{\mathsf{round}}); \mathrm{new}\, r_2; \\
& & \textbf{(* Uploading the witness submission *)} \\
& & \mathrm{let}\ \mathsf{submission} = (\lambda, \mathsf{idA}, \mathsf{senc}(k, r_2, ((\mathsf{subm}, x_{\mathsf{round}}), \lambda, \mathsf{idA}, \mathbf{s_w}))) \ \mathrm{in} \\
& & \mathrm{out}(c, \mathsf{submission})
\end{array}
$$

Then, following [15], we say that a competition management protocol satisfies secrecy of applications if, even if the cloud initially knows the content of two particular submissions $s^1_{\mathsf{w}}$ and $s^2_{\mathsf{w}}$, the cloud cannot make a distinction between an execution of the protocol where an applicant submitted an application containing $s^1_{\mathsf{w}}$ and an execution where the same applicant has submitted $s^2_{\mathsf{w}}$. To formally capture this for our specification, we construct from $\mathsf{CC}[\_]$ two processes: in the first one the hole is filled with a witness applicant process with identity $\mathsf{idA}_{\mathsf{w}}$ that submits $s^1_{\mathsf{w}}$, and in the second one the hole is filled with an applicant process with identity $\mathsf{idA}_{\mathsf{w}}$ that submits $s^2_{\mathsf{w}}$. Both $s^1_{\mathsf{w}}$ and $s^2_{\mathsf{w}}$ are modeled as free names, capturing the fact that the two submissions are known to the cloud.

**Submission secrecy**

$$
\mathsf{CC}[\mathsf{Applicant}^{\mathsf{s}}(\mathsf{idA}_{\mathsf{w}}, s^1_{\mathsf{w}})] \ \approx \ \mathsf{CC}[\mathsf{Applicant}^{\mathsf{s}}(\mathsf{idA}_{\mathsf{w}}, s^2_{\mathsf{w}})]
$$

### 4.3.2 Evaluation secrecy

Our protocol ensures the secrecy of evaluations that are assigned by an evaluator to a given submission. Accordingly, we will have a process for a witness evaluator process that will allow us to consider a particular evaluation that is being produced by an evaluator for a given application. Furthermore, since evaluations are sent back to applicants at the notification phase, we also need to assume an honest applicant in our definitions. Otherwise, the attacker would trivially know the content of an evaluation (one given to one of his own submissions). To model that honest applicant, we define a witness applicant process, that we denote by $\mathsf{Applicant}^{a}(\mathsf{idA}_{\mathsf{w}}, \lambda_{\mathsf{w}}, k_{\mathsf{w}})$, and that allows us to express that one of the applications from $\mathsf{idA}_{\mathsf{w}}$ has identifier $\lambda_{\mathsf{w}}$ and submission key $k_{\mathsf{w}}$:

$$
\begin{array}{rcl}
\mathsf{Applicant}^{\mathsf{a}}(\mathsf{idA}_{\mathsf{w}}, \lambda_{\mathsf{w}}, \mathsf{k}_{\mathsf{w}}) & = & \textbf{(* General applications and one witness application *)} \\
& & \textbf{let idA} = \mathbf{idA_{w}} \textbf{ in } (!\mathbf{AppReg}) \mid \mathbf{AppReg^{a}} \\
\mathsf{AppReg}^{\mathsf{a}} & = & \mathrm{new}\, r_1; \mathrm{in}(c_{\mathsf{pbk}}, x_{\mathsf{pbk}}); \\
& & \textbf{(* Registering the witness application *)} \\
& & \mathrm{out}(c, (\lambda, \mathsf{idA}, \mathsf{aenc}(x_{\mathsf{pbk}}, r_1, (\mathsf{reg}, \lambda_{\mathbf{w}}, \mathsf{idA}, \mathbf{k_w})))); \\
& & ! \, \mathsf{Submission}
\end{array}
$$

To define a particular evaluator, we consider the process $\mathsf{Evaluator}^{e}(\mathsf{idR}_{\mathsf{w}}, \lambda_{\mathsf{w}}, k_{\mathsf{w}}, \mathsf{eval}_{\mathsf{w}})$, that specifies an evaluator whose identity is $\mathsf{idR}_{\mathsf{w}}$ and behaves like a regular evaluator, with the single difference that among general applications, $\mathsf{idR}_{\mathsf{w}}$ also evaluates the withness application whose identifier is $\lambda_{\mathsf{w}}$ and whose key is $k_{\mathsf{w}}$, to which it assigns the evaluation $\mathsf{eval}_{\mathsf{w}}$:

$$
\begin{array}{lcl}
\mathsf{Evaluator}^{e}(\mathsf{idR}_\mathsf{w}, \lambda_\mathsf{w}, k_\mathsf{w}, \mathsf{eval}_\mathsf{w}) & = & \textbf{let } \mathsf{idR} = \mathsf{idR}_\mathsf{w} \textbf{ in } ! \ \mathsf{RevReg}^{\mathsf{e}} \\
\mathsf{RevReg}^{\mathsf{e}} & = & \mathrm{out}(c, \mathsf{idR}); \mathrm{in}(c_{\mathsf{shk}}, x_{\mathsf{shk}}); \\
& & \textbf{(* The only difference to } \mathsf{RevReg} \textbf{ is the } \mathsf{Evaluate}^{\mathsf{e}} \textbf{ process *)} \\
& & !\mathsf{Conflicts} \mid ! \ (\ \mathrm{in}(c_{\mathsf{round}}, x_{\mathsf{round}}); (! \ \mathsf{Bids} \mid \textbf{Evaluate}^{\mathbf{e}} \mid ! \ \mathsf{Discussion})) \\
\mathsf{Evaluate}^{\mathsf{e}} & = & \mathrm{in}(c, (x_\mu, = \mathsf{idR}, x_{\mathsf{blob}})); \\
& & \mathrm{let}\ ((= \mathsf{assigned}, = x_{\mathsf{round}}), = x_\mu, x_\lambda, = \mathsf{idR}, x_{\mathsf{idA}}, x_{\mathsf{key}}) = \mathsf{sdec}(x_{\mathsf{shk}}, x_{\mathsf{blob}}) \ \mathrm{in} \\
& & \textbf{if } (x_\lambda, x_{\mathsf{key}}) = (\lambda_\mathsf{w}, k_\mathsf{w}) \textbf{ then} \\
& & (\quad \textbf{(* Evaluation of the witness submission *)} \\
& & \quad \mathrm{new}\, r; \\
& & \quad \textbf{(* The value of the evaluation is specific *)} \\
& & \quad \mathrm{let}\ \mathsf{rev} = (x_\mu, \mathsf{idR}, \mathsf{senc}(x_{\mathsf{shk}}, r, ((\mathsf{evl}, x_{\mathsf{round}}), x_\mu, \lambda_\mathbf{w}, k_\mathbf{w}, \mathsf{eval}_\mathbf{w}, \mathsf{zero}))) \ \mathrm{in} \\
& & \quad \mathrm{out}(c, \mathsf{rev}) \quad ) \\
& & \mathrm{else} \\
& & (\quad \textbf{(* Evaluation of a general submission *)} \\
& & \quad \mathrm{new}\, eval; \mathrm{new}\, r; \\
& & \quad \textbf{(* The value of the evaluation is generic *)} \\
& & \quad \mathrm{let}\ \mathsf{rev} = (x_\mu, \mathsf{idR}, \mathsf{senc}(x_{\mathsf{shk}}, r, ((\mathsf{evl}, x_{\mathsf{round}}), x_\mu, x_\lambda, x_{\mathsf{key}}, eval, \mathsf{zero}))) \ \mathrm{in} \\
& & \quad \mathrm{out}(c, \mathsf{rev}) \quad )
\end{array}
$$

Then, we say that a competition management protocol satisfies evaluation secrecy, if the cloud cannot make a distinction between the case when the witness evaluator $\mathsf{idR}_\mathsf{w}$ has evaluated an application with $\mathsf{eval}^1_\mathsf{w}$ or with $\mathsf{eval}^2_\mathsf{w}$. To make this property even stronger, we assume that the evaluations $\mathsf{eval}^1_\mathsf{w}$ and $\mathsf{eval}^2_\mathsf{w}$ are public. Relying on the witness processes that we have defined so far, we can express evaluation secrecy as the following equivalence:

**Evaluation secrecy**

$$
\begin{array}{l}
\mathrm{new}\, \lambda_\mathsf{w};\ \mathrm{new}\, k_\mathsf{w};\ \mathsf{CC}[\mathsf{Applicant}^{\mathsf{a}}(\mathsf{idA}_\mathsf{w}, \lambda_\mathsf{w}, k_\mathsf{w}) \mid \mathsf{Evaluator}^{\mathsf{e}}(\mathsf{idR}_\mathsf{w}, \lambda_\mathsf{w}, k_\mathsf{w}, \mathsf{eval}^1_\mathsf{w})] \quad \approx \\
\mathrm{new}\, \lambda_\mathsf{w};\ \mathrm{new}\, k_\mathsf{w};\ \mathsf{CC}[\mathsf{Applicant}^{\mathsf{a}}(\mathsf{idA}_\mathsf{w}, \lambda_\mathsf{w}, k_\mathsf{w}) \mid \mathsf{Evaluator}^{\mathsf{e}}(\mathsf{idR}_\mathsf{w}, \lambda_\mathsf{w}, k_\mathsf{w}, \mathsf{eval}^2_\mathsf{w})]
\end{array}
$$

### 4.3.3 Applicant-evaluator unlinkability

Applicant-evaluator unlinkability means that the cloud cannot link a given applicant to the evaluators of his application. For the property to hold, it must be the case that the competition has at least two applicants, whose identities we denote by $\mathsf{idA}^1_\mathsf{w}$ and $\mathsf{idA}^2_\mathsf{w}$, and two evaluators, whose identities we denote by $\mathsf{idR}^1_\mathsf{w}$ and $\mathsf{idR}^2_\mathsf{w}$. Then, to express unlinkability, we require that the two executions where

- $\mathsf{idA}^1_\mathsf{w}$ submits an application that is evaluated by $\mathsf{idR}^1_\mathsf{w}$ and $\mathsf{idA}^2_\mathsf{w}$ submits an application that is evaluated by $\mathsf{idR}^2_\mathsf{w}$
- $\mathsf{idA}^1_\mathsf{w}$ submits an application that is evaluated by $\mathsf{idR}^2_\mathsf{w}$ and $\mathsf{idA}^2_\mathsf{w}$ submits an application that is evaluated by $\mathsf{idR}^1_\mathsf{w}$

are indistinguishable from the point of view of the cloud. In order to specify such two executions formally, we refine the manager process $\mathsf{Manager}$ such that certain applications can be assigned to certain evaluators determined apriori. Specifically, we refine the process $\mathsf{Assign}$, by defining a parametrized process $\mathsf{Assign}^{\mathsf{ae}}(\mathsf{idA}^1_\mathsf{w}, \mathsf{idA}^2_\mathsf{w}, \mathsf{idR}^1_\mathsf{w}, \mathsf{idR}^2_\mathsf{w})$. This process ensures that some submissions from $\mathsf{idA}^1_\mathsf{w}$ are evaluated by $\mathsf{idR}^1_\mathsf{w}$ and some submissions from $\mathsf{idA}^2_\mathsf{w}$ are evaluated by $\mathsf{idR}^2_\mathsf{w}$. Furthermore, this process also ensures that no submission from $\mathsf{idA}^1_\mathsf{w}$ is evaluated by $\mathsf{idR}^2_\mathsf{w}$ and no submission from $\mathsf{idA}^2_\mathsf{w}$ is evaluated by $\mathsf{idR}^1_\mathsf{w}$. Both the former and the latter properties are important to ensure that there is indeed a significant link between $\mathsf{idA}^i_\mathsf{w}$ and $\mathsf{idR}^i_\mathsf{w}$, for $i \in \{1, 2\}$.

$\mathsf{Assign}^{\mathsf{ae}}(\mathsf{idA}^1_{\mathsf{w}}, \mathsf{idA}^2_{\mathsf{w}}, \mathsf{idR}^1_{\mathsf{w}}, \mathsf{idR}^2_{\mathsf{w}}) =$
**(* Download an application and a bid from the cloud *)**
$\mathrm{in}(c, (x_\mu, \mathsf{app\_blob})); \mathrm{in}(c, (x_{\mathsf{idR}}, \mathsf{bid\_blob}));$
$\mathrm{let}\ ((= \mathsf{initround}, = x_{\mathsf{round}}), = x_\mu, x_\lambda, x_{\mathsf{idA}}, x_{\mathsf{key}}) = \mathsf{sdec}(k_{\mathsf{shk}}, \mathsf{subm\_blob})\ \mathrm{in}$
$\mathrm{let}\ ((= \mathsf{bid}, = x_{\mathsf{round}}), = \mathsf{one}, = x_{\mathsf{idR}}, = x_\mu) = \mathsf{sdec}(k_{\mathsf{shk}}, \mathsf{bid\_blob})\ \mathrm{in}$
**(* Restrictions on applications assigned to evaluators: *)**
**if** $x_{\mathsf{idR}} \notin \{\mathsf{idR}^1_{\mathsf{w}}, \mathsf{idR}^2_{\mathsf{w}}\} \vee$ **(* Either $x_{\mathsf{idR}}$ is not a witness evaluator *)**
$x_{\mathsf{idA}} \notin \{\mathsf{idA}^1_{\mathsf{w}}, \mathsf{idA}^2_{\mathsf{w}}\} \vee$ **(* or $x_{\mathsf{idA}}$ is not a witness applicant *)**
$(x_{\mathsf{idR}}, x_{\mathsf{idA}}) = (\mathsf{idR}^1_{\mathsf{w}}, \mathsf{idA}^1_{\mathsf{w}}) \vee$ **(* or an app. of $\mathsf{idA}^1_{\mathsf{w}}$ is assigned to $\mathsf{idR}^1_{\mathsf{w}}$ *)**
$(x_{\mathsf{idR}}, x_{\mathsf{idA}}) = (\mathsf{idR}^2_{\mathsf{w}}, \mathsf{idA}^2_{\mathsf{w}}) \vee$ **(* or an app. of $\mathsf{idA}^2_{\mathsf{w}}$ is assigned to $\mathsf{idR}^2_{\mathsf{w}}$ *)**
**then** $\mathrm{new}\, r;$
$\mathrm{let}\ \mathsf{assigned\_blob} = \mathsf{senc}(k_{\mathsf{shk}}, r, ((\mathsf{assigned}, x_{\mathsf{round}}), x_\mu, x_\lambda, x_{\mathsf{idR}}, x_{\mathsf{idA}}, x_{\mathsf{key}}))\ \mathrm{in}$
$\mathrm{let}\ \mathsf{assignement} = (x_\mu, x_{\mathsf{idR}}, \mathsf{assigned\_blob})\ \mathrm{in}\ \mathrm{out}(c, \mathsf{assignement})$

Let us discuss the tests that are part of the specification $\mathsf{Assign}^{\mathsf{ae}}$. First, if the test $x_{\mathsf{idR}} \notin \{\mathsf{idR}^1_{\mathsf{w}}, \mathsf{idR}^2_{\mathsf{w}}\}$ is true, it means that the evaluator whose bid is considered is not a witness evaluator, and therefore there is no restriction that needs to be imposed on the applications that are assigned to $x_{\mathsf{idR}}$. Similarly, if the test $x_{\mathsf{idA}} \notin \{\mathsf{idA}^1_{\mathsf{w}}, \mathsf{idA}^2_{\mathsf{w}}\}$ is true, there is not further restriction to be imposed on this assignement. Otherwise, it must be the case that $x_{\mathsf{idR}}$ is a witness evaluator and $x_{\mathsf{idA}}$ is a witness applicant. In that case, we allow the assignement to go through only if an application from $\mathsf{idA}^1_{\mathsf{w}}$ is assigned to $\mathsf{idR}^1_{\mathsf{w}}$, or if an application from $\mathsf{idA}^2_{\mathsf{w}}$ is assigned to $\mathsf{idR}^2_{\mathsf{w}}$.

Now, we consider the process $\mathsf{Manager}^{\mathsf{ae}}(\mathsf{idA}^1_{\mathsf{w}}, \mathsf{idA}^2_{\mathsf{w}}, \mathsf{idR}^1_{\mathsf{w}}, \mathsf{idR}^2_{\mathsf{w}})$, that is obtained from the process $\mathsf{Manager}$ by replacing the sub-process $\mathsf{Assign}$ with the process $\mathsf{Assign}^{\mathsf{ae}}(\mathsf{idA}^1_{\mathsf{w}}, \mathsf{idA}^2_{\mathsf{w}}, \mathsf{idR}^1_{\mathsf{w}}, \mathsf{idR}^2_{\mathsf{w}})$. Additionally, we consider the processes $\mathsf{Applicant}^{\mathsf{id}}(\mathsf{idA})$ and $\mathsf{Evaluator}^{\mathsf{id}}(\mathsf{idR})$, that allow to fix the identity of an applicant and respectively of an evaluator. Then, we can express applicant-evaluator unlinkability by the following observational equivalence:

**Applicant-evaluator unlinkability**

$$\mathsf{CC}\left[\begin{array}{l}\mathsf{Applicant}^{\mathsf{id}}(\mathsf{idA}^1_{\mathsf{w}}) \mid \mathsf{Applicant}^{\mathsf{id}}(\mathsf{idA}^2_{\mathsf{w}}) \mid \\ \mathsf{Evaluator}^{\mathsf{id}}(\mathsf{idR}^1_{\mathsf{w}}) \mid \mathsf{Evaluator}^{\mathsf{id}}(\mathsf{idR}^2_{\mathsf{w}}) \mid \\ \mathsf{Manager}^{\mathsf{ae}}(\mathsf{idA}^1_{\mathsf{w}}, \mathsf{idA}^2_{\mathsf{w}}, \boxed{\mathsf{idR}^1_{\mathsf{w}}, \mathsf{idR}^2_{\mathsf{w}}})\end{array}\right] \approx \mathsf{CC}\left[\begin{array}{l}\mathsf{Applicant}^{\mathsf{id}}(\mathsf{idA}^1_{\mathsf{w}}) \mid \mathsf{Applicant}^{\mathsf{id}}(\mathsf{idA}^2_{\mathsf{w}}) \mid \\ \mathsf{Evaluator}^{\mathsf{id}}(\mathsf{idR}^1_{\mathsf{w}}) \mid \mathsf{Evaluator}^{\mathsf{id}}(\mathsf{idR}^2_{\mathsf{w}}) \mid \\ \mathsf{Manager}^{\mathsf{ae}}(\mathsf{idA}^1_{\mathsf{w}}, \mathsf{idA}^2_{\mathsf{w}}, \boxed{\mathsf{idR}^2_{\mathsf{w}}, \mathsf{idR}^1_{\mathsf{w}}})\end{array}\right]$$

**Automated analysis.** We use the ProVerif tool to prove that all three eqivalence properties defined above are true in our model. The complete specification of these properties in ProVerif is available online [37].

# 5 Implementation of ConfiChair

In order to evaluate the performance and usability of the protocol, we have implemented it for conference management systems in a system we call ConfiChair. 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 [37], 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.

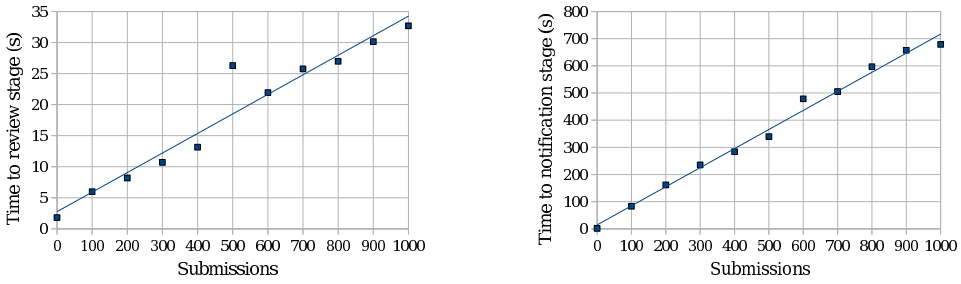


Figure 5: Performance evaluation. The time taken for transitioning to the review stage is about 25s for 500 papers. The time for transitioning to the notification phase is about 350s for 500 papers. These figures average over several runs.

## 5.1 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 $\mathsf{pub}(\mathrm{Comp})$; 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 Comp, logs into his ConfiChair account and clicks the *create new conference* button. His browser will transparently generate and securely store the keys $K_{\mathrm{Comp}}$, $\mathsf{pub}(\mathrm{Comp})$, and $\mathsf{priv}(\mathrm{Comp})$.

## 5.2 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 5. Also, experiments with any number of random files can be easily re-run on [37]. 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.

These performance figures are of course worse than EasyChair, where no cryptography is used other than the usual session encryption using TLS, and in particular the phase transitions take negligible time. Using ConfiChair involves a small performance hit, which is noticed only by the chair during the phase transitions (figure 5). Users and reviewers don't notice any performance hit, because the cryptography time is negligible compared with the time the users themselves take.

## 5.3 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 $\mathsf{pub}(\mathrm{Comp})$ to which they want to submit, and the reviewers need to copy and paste from their email the shared-key of the conference $K_{\mathrm{Comp}}$ for which they are reviewers.

### 5.4 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 [7, 8, 30] 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.

## 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. Confidential data about the applications and evaluations of different types of bids and tenders is currently stored on a relatively small set of cloud-based competition management systems. For example, in the case of conference management, the authoring and reviewing performance of tens of thousands of researchers across thousands of conferences is stored by one or two well-known cloud-based systems [40].

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 a protocol underlying a competition management system

that uses this technique to obtain strong privacy properties while having all the advantages of cloud computing. In this system, 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 competition). The competition manager'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 the protocol. It still enables the cloud provider to route information to the necessary managers, evaluators and applicants, to enforce access control, and optionally to perform statistics collection. In our implementation of ConfiChair, we have demonstrated that the cryptography and key management can be handled by a regular web browser [37] (specifically, we used LiveConnect). We plan to continue developing our prototype into a complete system.

An important design decision in our protocol and in ConfiChair is the fact that a single key $K_{\mathrm{Comp}}$ is used to encrypt all the information for the competition. Stronger secrecy properties could be obtained if a different key were used for different subsets of evaluators and applicants, but this would be at the cost of simplicity. Using a single key per competition 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] eTenderer. `http://www.etenderer.com/public.aspx`.

[2] INDECO tender management system. `http://www.indeco.co.uk/index.php?id=tendermanagementsystem`.

[3] Tendering for public contracts. A guide for small businesses. `http://www.bis.gov.uk/files/file39469.pdf`.

[4] VHTender - tender management system. `http://www.vhsoft.com/products/vhtender.htm`.

[5] Martín Abadi. Security protocols and their properties. In *Foundations of Secure Computation, NATO Science Series*, pages 39–60. IOS Press, 2000.

[6] 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.

[7] Ben Adida. Helios: Web-based open-audit voting. In Paul C. van Oorschot, editor, *USENIX Security Symposium*, pages 335–348. USENIX Association, 2008.

[8] 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.

[9] Myrto Arapinis, Sergiu Bursuc, and Mark Ryan. Privacy supporting cloud computing: Confichair, a case study. In Pierpaolo Degano and Joshua D. Guttman, editors, *POST*, volume 7215 of *Lecture Notes in Computer Science*, pages 89–108. Springer, 2012.

[10] Y. Aumann and Y. Lindell. Security against covert adversaries: Efficient protocols for realistic adversaries. In *Theory of Cryptography*, 2007.

[11] 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.

[12] Olivier Baudron, Pierre-Alain Fouque, David Pointcheval, Jacques Stern, and Guillaume Poupard. Practical multi-candidate election system. In *PODC*, pages 274–283, 2001.

[13] 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.

[14] Bruno Blanchet. An efficient cryptographic protocol verifier based on Prolog rules. In *Computer Security Foundations Workshop (CSFW'01)*, 2001.

[15] Bruno Blanchet. Automatic proof of strong secrecy for security protocols. In *IEEE Symposium on Security and Privacy*, pages 86–, 2004.

[16] Bruno Blanchet, Martín Abadi, and Cédric Fournet. Automated verification of selected equivalences for security protocols. *Journal of Logic and Algebraic Programming*, 2007.

[17] 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.

[18] 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.

[19] Sergiu Bursuc, Gurchetan S. Grewal, and Mark Dermot Ryan. Trivitas: Voters directly verifying votes. In Aggelos Kiayias and Helger Lipmaa, editors, *VOTE-ID*, volume 7187 of *Lecture Notes in Computer Science*, pages 190–207. Springer, 2011.

[20] 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.

[21] 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.

[22] 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.

[23] Tom Chothia and Vitaly Smirnov. A traceability attack against e-passports. In *Financial Cryptography*, 2010.

[24] Sebastian Clauß, Dogan Kesdogan, Tobias Kölsch, Lexi Pimenidis, Stefan Schiffner, and Sandra Steinbrecher. Privacy enhancing identity management: Protection against reidentification and profiling. In *Proceedings of the 2005 ACM Workshop on Digital Identity Management*, 2005.

[25] Cloud Security Alliance. *Secure Cloud*. `www.cloudsecurityalliance.org/sc2010.html`, 2010.

[26] 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.

[27] C. Gentry. Fully homomorphic encryption using ideal lattices. In *41st ACM Symposium on Theory of Computing (STOC)*, 2009.

[28] Gurchetan S. Grewal, Mark Dermot Ryan, Sergiu Bursuc, and Peter Y. A. Ryan. Caveat coercitor: Coercion-evidence in electronic voting. In *IEEE Symposium on Security and Privacy*, pages 367–381. IEEE Computer Society, 2013.

[29] 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.

[30] 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.*

[31] 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.

[32] 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.

[33] Steve Kremer, Mark Ryan, and Ben Smyth. Election verifiability in electronic voting protocols. In Dimitris Gritzalis, Bart Preneel, and Marianthi Theoharidou, editors, *ESORICS*, volume 6345 of *Lecture Notes in Computer Science*, pages 389–404. Springer, 2010.

[34] 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.

[35] Gavin Lowe. An attack on the Needham-Schroeder public-key authentication protocol. *Information Processing Letters*, 56(3):131–133, 1996.

[36] 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.

[37] Joshua Phillips and Matt Roberts. ConfiChair - prototype privacy-supporting conference management system. `http://www.confichair.org`.

[38] 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.

[39] 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.

[40] Mark D. Ryan. Cloud computing privacy concerns on our doorstep. *Communications of the ACM*, 54(1):36–38, 2011.

[41] 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.

[42] Steve Schneider and Abraham Sidiropoulos. CSP and anonymity. In *ESORICS*, pages 198–218, 1996.

[43] Andrei Voronkov et al. EasyChair Conference System. `http://www.easychair.org`.