Towards practical privacy-preserving Digital Rights Management for Cloud Computing

Nakul Joshi  
University of Southern California  
Los Angeles, CA 90089  
Email: nakuljos@usc.edu

Ronald Petrlic  
University of Paderborn  
33098 Paderborn  
Email: ronald.petrlic@uni-paderborn.de

Abstract—We propose a privacy-preserving digital rights management scheme for (future) cloud computing. Users buy software from software providers and execute it at computing centers. Our solution allows software providers to provide different license models, like execute at most \( n \)-times models. Users’ anonymity and unlinkability of actions are preserved and thus, profile building is not even possible under (a) pseudonym.

Privacy protection in the honest-but-curious model is achieved by combining ring signatures with an anonymous recipient scheme. We employ secret sharing in a unique manner that allows the software provider to expose the user’s identity if the user commits fraud, e.g. by exceeding the execution limit \( n \).

I. INTRODUCTION

Computing services are moving towards the cloud in reaction to demand for greater portability and convenience of use. Software providers, though, will want guarantees that their intellectual property rights will not be infringed upon before they invest in software development for the cloud. Copyright protection allows a producer of digital goods to regain the initial investments and keep the status of monopolistic power against competition until innovative goods displace its own. [2]

Digital Rights Management (DRM) provides, among other things, (cryptographic) methods for content provision, content safekeeping, license phrasing/offer creation, content distribution, booking, payment, authorization, and content consumption [12]. Thus, DRM is a popular approach, at least on the part of content providers, to provide copyright protection. Some DRM systems bind the digital content that shall be “protected” to the users’ platforms, whereas others bind the content to users. Content providers providing licenses that allow the execution of content at most \( n \) times need to keep track of executions. Thus, if trusted computing platforms can be assumed, local lists can be used to keep track of executions and the execution environment enforces the license policies, i.e. prohibits the content execution if the terms are not met any longer. However, state-of-the-art DRM systems often require platform/user authentication towards the content provider before granting execution authorization. [11]

Such DRM systems expose users to tracking. Content providers know exactly which users are accessing certain content. The detailed consumption profiles that can be built may be used for targeted advertisement. To prevent usage surveillance in cloud computing, privacy protection shall be considered during DRM system development. It should not be “a property that can be added on later”. [11]

In certain areas, software licensing will change in the future towards a pay-per-use model. Through the popularity of cloud computing, hybrid models of software licensing have emerged, e.g. a token model\(^1\) that is based on prepaid tokens which can be consumed for future software executions. [22] Such models allow for discounts on volume execution licenses. In the automotive industry, for example, cloud computing is seen as an important factor to support different applications such as engineering simulations. The automotive industry can benefit from “new” software licensing models as software is paid only for its actual/planned usage. [14] However, the usage of certain software by the automotive industry may help the executing computing center to draw conclusions about new business segments, etc. to be explored. Thus, profile building prevention—as aimed for in this paper—is not only important for privacy protection but to preserve business secrecy as well.

Attempts have been made to create privacy-protecting copyright enforcement mechanisms, as discussed in Sect. II, but they do not allow for flexible license policies. In particular, no efficient solution has been presented that permits an execute at most \( n \)-times model in a privacy-preserving manner. Our main contribution is to solve the gap between privacy protection asked for by users and flexible license models allowing differentiated pricing for software providers. We come up with a solution in this paper that provides the following properties:

- Different license models, like execute at most \( n \)-times models, are supported
- The enforcement does not reveal any personal information, like software usage patterns
- Users’ identities can be revealed only in case of fraud, i.e. if license terms are violated
- The scheme is practicable in terms of computation/communication overhead

The remainder of this paper is structured as follows. Related

\(^1\)This model is denoted by execute at most \( n \)-times model in this paper.
work is covered in Sect. II and the preliminaries our concept is based on are covered in Sect. III. We present our system model in Sect. IV and our concept in Sect. V. The concept is discussed in Sect. VI before we conclude in Sect. VII.

II. RELATED WORK

Conrado et al. [8] and Perlman et al. [16] have come up with privacy-preserving DRM solutions that allow users to buy content anonymously from a content provider and access the content without being tracked. However, the solution presented in [8] relies on smart cards and thus lacks practicality, whereas the solution presented in [16], based on blind decryption, has limited support for different license models, e.g. execute at most n-times models are not supported.

Conrado et al. [7] present another privacy-preserving solution for identity-based DRM systems. Licenses are bound to public keys, serving as users’ identities. However, neither the content providers that issue the licenses nor the devices that check the licenses learn the identities. The authors propose to blind the users’ public keys within the licenses. The scheme thus provides privacy protection, however it does not support licenses that allow an execution of content at most n-times.

Lee et al. [15] employ proxy re-encryption to make DRM interoperable, i.e. allowing a playback of content on several devices. Content providers manage and trace their digital content and request fees for the re-encryption of content, which is problematic in terms of privacy protection.

Petrlíč et al. [18] present a privacy-preserving DRM solution for cloud computing that does not require any trusted devices and which allows different license models. However, their protocol (based on secret sharing and homomorphic encryption) is inefficient in terms of communication overhead. Petrlíč [17] improves the protocol by making use of proxy re-encryption [1]. Moreover, the re-encryption algorithm is randomized—which provides indistinguishability of first-level ciphertexts under the condition that the same second-level ciphertext is re-encrypted for the same party several times.

III. PRELIMINARIES

A. Proxy re-encryption

Proxy re-encryption allows a proxy to transform a message encrypted under A’s public key into the message encrypted under B’s public key. The proxy cannot see the message in plain text during the process. The delegation is done by means of a re-encryption key generated by A using B’s public key and A’s own private key. We make use of unidirectional proxy re-encryption as proposed by Ateniese et al. [1], which allows A to delegate to B without requiring B to delegate to A. The scheme is based on the ElGamal cryptosystem [10].

B. Secret sharing

Secret sharing allows a party to create an arbitrarily large number of shares \( (n) \) out of a secret, such that a party that possesses a sufficient number of shares \( (k) \) can recover the secret. Parties with only \( k - 1 \) shares, however, do not learn anything about the secret except an upper bound on its length.

Such a scheme is called \((k, n)\)-threshold scheme as \( k \) or more shares out of \( n \) are needed to recover the secret. We employ the \((k, n)\)-threshold scheme proposed by Shamir [21].

C. Anonymous recipient encryption

Anonymous recipient encryption, as proposed by Waters et al. [23], allows a party to create multiple public keys under which messages can be encrypted, such that all the messages can be decrypted by a single corresponding private key. Further, the public keys are incomparable, i.e. the public keys cannot be connected to one another, preserving the anonymity of the recipient in an efficient fashion that does not require the storage of large numbers of keypairs. We employ the scheme based on the ElGamal cryptosystem [10], which has been proven to be secure in the random oracle model.

D. Ring signatures

Ring signatures allow data to be certified such that it can be guaranteed to have been signed by a member of a certain group. The signing party cannot be revealed, i.e. the signatures are signer-ambiguous. Further, multiple documents certified by the same member cannot be linked to one another. The scheme presented by Rivest et al. [20] is proven to be secure in the random oracle model. Unlike group signature schemes, like e.g. [5], the scheme we employ [20] does not require any setup phase and there is no group coordinator which could reveal the identity of the signer. All the public keys of the possible signers need to be included in the signature.

E. Anonymous payment

An anonymous payment scheme allows a user to anonymously retrieve digital coins from a bank. No party, not even the bank, is able to connect the coins to the user who paid for the coins, afterwards. We employ Chaum’s scheme [4], which is based on blind signatures. The bank blindly signs the digital coins using the RSA signature scheme [19] and debits the money from the user’s account.

IV. SYSTEM MODEL

A. Problem Statement

Users purchase software licenses from software providers. They may execute this software at any of several computing centers that have already obtained copies of the software from the software providers. There exists a trusted third party (TTP) that verifies licenses. It is necessary to protect both the software provider’s intellectual property rights as well as the user’s privacy. None of the parties involved must be able to study the user’s usage pattern.

B. Architecture and Basic Approach

The architecture and approach of our system are shown in Fig. 1. The user first anonymously purchases a license for a software from the software provider. To execute the software, the user requests the TTP for a software execution token. The TTP validates the license and then issues a certified token. On presenting the token to the computing center, the computing center validates the token and executes the software.
C. Notations

We denote the set of users as $\mathbb{U}$ and a single user as $u \in \mathbb{U}$. Similarly, we have $swp \in SWP$ and $cc \in CC$ to denote a software provider and a computing center. A certain software from the catalog $SW$ offered by $swp$ is denoted by $sw$. Finally, the trusted third party is denoted by $ttp$.

D. Assumptions

We assume that none of the parties involved collude to build user profiles, and that $swp$ can trust $ttp$ and $cc$ to enforce licenses. We also assume a system of honest-but-curious entities which follow the protocol as described correctly but which have an incentive to store any data acquired in the process and glean information from it. The threat of a dishonest $ttp$ is that it could reveal identities of users in the system.

Transferrability of licenses is not an issue in our scenario and thus is outside the scope of this paper. Even in a fully non-anonymous environment, a user could “transfer” a license to another user by attaching the other user’s input data on an anonymous environment, a user could “transfer” a license to another user by attaching the other user’s input data on an anonymous environment.

E. Requirements

1) Req. I: User Anonymity: $swp$ and $cc$ must not obtain $u$’s personally identifying information (PII), i.e., any information that are related or relatable to $u$. Such information would be e.g., name, address, credit card details, IP address, etc.

2) Req. II: Unlinkability of $sw$ to $u$: $ttp$ knows $u$’s identity and is contacted for a new token by $u$ before each $sw$ execution. Thus, to prevent $ttp$ from profile building, $ttp$ must not find out which $sw$ is executed by $u$. Furthermore, $ttp$ must not find out which $swp$ provides $u$ with software, as this would allow $ttp$ to draw conclusions about the type of software.

3) Req. III: Unlinkability of software purchases: $swp$ must not be able to connect different purchases by the same user to one another, nor must it be able to observe the software usage pattern of any user. Thus, $swp$ must not be able to sufficiently distinguish whether two purchases are related or not.

4) Req. IV: Unlinkability of software executions: $cc$ must not be able to connect multiple software executions from $u$ to one another, even if they are of the same $sw$. Thus, $cc$ must not be able to sufficiently distinguish whether two software executions are related or not.

5) Req. V: License Enforcement: Only users who have been authorized to run the software upon purchase of a license should be able to run the software—under the terms specified by the license. The $swp$ should be able to specify restrictions such as execution limits. The $swp$ should be able to identify users exceeding execution limits.

6) Req. VI: Practicality: The solution must be efficient, i.e., the computational overhead must be kept low, and $u$ in particular cannot be expected to perform any large computation.

V. PROPOSED CONCEPT

Our scheme consists of the following phases.

A. Initialization

In the initial set-up of the system, users, computing centers, and software providers all register with $ttp$. Users need to provide their identity, i.e., name and email address. Computing centers need to get certified and confirm that they will not illegally relay users’ software to others. The $ttp$ creates a list of approved $swps$, which is later on used for license validation. This list is regularly updated and published whenever new $swps$ join or leave the system. The $swps$ must confirm that they do not sign any software from third parties. Approved $ccs$ obtain copies of the software from $swps$. This download in advance makes sense for software that is assumed to be executed by many users later on. Each $swp$ sets up an anonymous receiver encryption scheme, as described in Sect. III. It provides a private key $swp_{priv}$ and allows the creation of anonymous public keys $SWP_{pub}$.

B. Purchase

In this phase, $u$ anonymously pays $swp$ using the anonymous payment scheme presented in Sect. III, and acquires a license for $sw$. An anonymization network such as Tor [9] is used by $u$ to contact $swp$. The communication is secure, i.e., TLS-protected [13]—$swp$ authenticates towards $u$.

1) Goals: The license must be generated in such a manner that $ttp$ can verify that executions meet the terms of the license (to comply with Req. V). The license must also include the identity of the software, but this should not be visible to $ttp$ (to comply with Req. II).

2) Keys: The keys used in this phase are:

   - For each purchase, $u$ generates and maintains a temporary keypair $(u_{pub}, u_{priv})$
   - $swp$ maintains the permanent signature keypair $(swp_{sign_{pub}}, swp_{sign_{priv}})$
   - $swp$ creates the anonymous recipient public key $swp_{pub}^3$

3) Protocol: $u$ sends a license request for $sw$ to $swp$, containing the following data:

   - software id
   - digital coins worth the amount to spend for $sw$
   - $u_{pub}$, the temporary public key

   The $swp$ generates a license $L$, including the following information, and returns it to $u$:

   - The terms of the license, in plaintext
- The software id, signed under \(swp\)-\(sign_{prv}\) and then encrypted under \(u_{pub}\)
- The anonymous public key \(swp^i_{pub}\)
- A unique identifying number for the license
- A ring signature on \(L\) of SWP

The terms need to be included in the license so that \(ttp\) can check them. The software id is signed so that \(cc\) can check that the execution token belongs to the proper \(sw\). However, the id needs to be encrypted under \(u_{'s public key as \(ttp\) must not see it (to comply with Req. II). The (newly generated) anonymous public key is needed so that \(cc\) can send the execution token to \(swp\) at the end without allowing \(ttp\) to find out \(swp\)’s identity—if \(swp\) used the same public key each time, \(ttp\) would know from whom \(u\) is buying \(sw\) and thus violating Req. II. The unique identifying number for the license does not tell anything about the software identity but is needed so that \(ttp\) can keep track of different licenses for the same \(u\).

The whole license is certified with a ring signature of SWP. Thus, \(ttp\) will be able to check whether \(L\) has been signed by one of the approved \(swps\) but it will not know by which \(swp\).

C. Token Request

In this phase, the license is verified and \(ttp\) permits \(u\) to execute \(sw\) at \(cc\), by issuing \(u\) a token.

1) Goals: \(ttp\) must be able to check that the license is valid before issuing a token (to comply with Req. V). However, \(ttp\) must not see the software id (to comply with Req. II). \(u\) must not be able to duplicate tokens, and \(cc\) should be able to check which software the token was issued for. Further, \(cc\) must not be able to find out who is running the software and should not be able to link multiple executions by \(u\) to each other (to comply with Req. IV). Lastly, there must be a mechanism by which exceeding any execution limits specified in the license can be detected (also to comply with Req. V).

2) Keys: The keys used in this phase are:
- \(cc_{pub}\), the permanent public key of \(cc\)
- \(r_{u\rightarrow cc}\), a re-encryption key generated by \(u\) after choosing a \(cc\) and acquiring \(cc_{pub}\)
- \(ttp_{priv}\), \(ttp\)’s permanent private key

3) Protocol: \(u\) requests a token from \(ttp\) by sending it \(L\) and \(r_{u\rightarrow cc}\). \(ttp\) verifies the license by checking the signature and validating the terms of the license.

\(ttp\) uses \(r_{u\rightarrow cc}\) to re-encrypt the (encrypted) software id so that it is encrypted under \(cc_{'s public key \(cc_{pub}\) later on. The proxy re-encryption ensures that \(ttp\) can use the encrypted software id provided by \(swp\) to create a token with a software id that is only visible to \(cc\). Further, it allows \(u\) to use a different \(cc\) for each execution without having to acquire a new license. The re-encryption algorithm needs to be randomized, as presented in [17], so that \(cc\) cannot relate encrypted software ids to each other based on the same ciphertexts.

If this is the first request for the given license (as identified by the unique license id), and if the license specifies an execution limit of \(n\), then \(ttp\) internally assigns the user a pseudonym \(p\) and sets up an \(n+1\)-threshold secret-sharing scheme, where \(p\) is the secret. With this scheme, \(ttp\) can create a number of secret shares \(P\). Those shares can be used later on, as explained in Sect. V-E, to identify users that exceeded the execution limit. \(ttp\) can then generate a token, which includes:

- A timestamp to prevent replay attacks by \(u\)
- The re-encrypted software id from the license, now encrypted under \(cc_{pub}\)
- Metadata that will allow \(swp\) to check if execution limits are exceeded, in case of a limited-execution license:
  - \(group-id\), an identifier that allows \(swp\) to group tokens from the same license together
  - A randomly generated pseudonym share, \(p_i \in P\)
  - The execution limit

This metadata is encrypted under \(swp^i_{pub}\), which is retrieved from the license. Note that \(ttp\) does not know for which \(swp\) this metadata is encrypted.

The token is signed by \(ttp\) under \(ttp_{priv}\).

D. Execution

\(u\) anonymously submits the token to \(cc\) using Tor. \(cc\) validates the token by checking \(ttp\)’s signature and the timestamp. Then \(cc\) decrypts the re-encrypted software id using its own private key \(cc_{priv}\) and checks its validity by verifying it with \(swp_{sign_{prv}}\). It executes the software specified by the id. User input may be included with the execution request. For larger amounts of data, the \(cc\) will need to be given some means of accessing \(u_{'s data on an external data storage.

E. Token Collection

Tokens are kept by \(cc\) for a certain period of time, perhaps a month. \(cc\) removes the timestamps from the tokens. The tokens are then sent\(^3\) to \(swp\). \(swp\) extracts the metadata from the tokens and decrypts them using \(swp_{priv}\). Tokens are then grouped by their identifiers \(group-id\), and counted. If they exceed the execution limit, which is also part of the tokens, the tokens can be used to reveal \(u_{'s pseudonym. The pseudonym can be revealed by executing the secret sharing scheme as at least \(n+1\) are collected for the license allowing \(n\) executions and \(n+1\) is the threshold value. The offender can be identified once \(swp\) knows the pseudonym by contacting \(ttp\) to reveal the mapping from the pseudonym to the identity.

VI. EVALUATION AND DISCUSSION

We discuss our proposed scheme by evaluating whether the requirements as pointed out in Sect. IV-E are met.

\(^3\)In the anonymous recipient scheme of [23], the sending party knows nothing about the receiver but the anonymous public key. Thus, the message has to be broadcast to all possible recipients, who each try their own private keys on the message. However, in our scenario, the encrypting party \(ttp\) again only knows the recipient’s anonymous public key, but the sending party \((cc)\) knows the identity of \(swp\). Thus a broadcast is not needed and direct transmission can be used.
A. User Anonymity

Neither swp nor cc obtain any PII of the user. The user anonymously buys software using an anonymous payment scheme and the temporary user public key $u_{pub}$ does not contain any PII either. Moreover, $u$ contacts swp and cc using the anonymization network Tor. Thus, we meet Req. I.

B. Unlinkability of swp to u

As ttp knows $u$’s identity, it must not find out which software is used by $u$. During the purchase phase (Sect. V-B), swp encrypts the software id under $u$’s public key so that ttp cannot retrieve the id from the license. Further, $u$ provides ttp with the re-encryption key to re-encrypt the encrypted software id. As the employed re-encryption scheme [20] does not allow the proxy (i.e. ttp in our case) to gain any information about the plain text from the re-encryption process, as performed during the token request phase (Sect. V-C), ttp does not gain any information about the software. We propose to use the anonymous recipient scheme so that ttp does not know which swp will be the receiver of the tokens during the token collection phase (Sect. V-E). The public keys in the employed scheme [23] are incomparable. This is important as the information about swp would allow ttp to draw conclusions about the software, e.g. certain swps provide only very specialized software. Furthermore, the license is signed by swp using a ring signature scheme. Thus, ttp can only verify that the license was signed by one of the swps (SWP), but it cannot find out which swp signed the license as the signatures in the employed scheme [20] are signer-ambiguous and multiple licenses certified by the same swp cannot be linked to one another. Thus, ttp cannot find out which sw is used by $u$ or which swp provides the software and we meet Req. II.

C. Unlinkability of software purchases

If swp was able to link two purchases to each other, it would be able to build a user profile (under pseudonym). Thus, we require unlinkability for software purchases. $u$ creates a new temporary public key for each purchase, so swp cannot link the purchases of $u$ to each other based on the public keys. The swp does not take part during the token request and software execution phase. During the token collection phase (Sect. V-E) swp receives all the tokens from cc. As the timestamps are removed from the tokens before transmission to swp, they do not reveal any information that could be used to build a usage profile. The token grouping only allows swp to learn how often the software has been executed—which might be a desired information for swps. Only in the case that $u$ executes sw that is allowed to be executed at most $n$ times more than $n$ times, the threshold secret sharing allows swp to reconstruct the user’s pseudonym. The secret sharing scheme [21] does not allow the reconstruction of the pseudonym with less than $n + 1$ tokens. Thus, the user’s pseudonym is protected as long as he/she does not illegally execute the software more often than having paid for. Given two purchases $purchase_1$ and $purchase_2$, swp cannot do better than random guessing to decide whether those two purchases are linked to each other, i.e. are from the same $u$. Thus, we meet Req. III.

D. Unlinkability of software executions

The cc must not be able to connect multiple software executions of the same sw from $u$ to one another, i.e. they need to be unlinkable, to prevent profile building under pseudonym towards cc. cc receives a new execution token (Sect. V-D) from $u$ for each software execution. Those tokens do not contain any information that would allow cc to link them together. The metadata which serves swp during the token collection phase (Sect. V-E) cannot be read by cc as they are encrypted under swp’s anonymous public key. Thus, even in case of fraud, cc cannot reveal $u$’s pseudonym. Given two executions $execution_1$ and $execution_2$, cc cannot do better than random guessing to decide whether those two executions are related to each other, i.e. are from the same $u$. Thus, we meet Req. IV.

E. License Enforcement

To meet the license enforcement demands stated in Sect. IV-E, the right to execute software has to be checked before every single execution. The check is performed by ttp during the token request (Sect. V-C). The ttp checks the terms of the license. Furthermore, the signed and encrypted software id is taken from the license and put into the token so that cc can check whether the issued token belongs to the software to be executed. A timestamp within the token prevents replay attacks where $u$ could try to use the token for an execution more than once. The tokens are certified by ttp so cc will only accept such tokens. As discussed above, if $u$ requests more than $n$ tokens, where $n$ is the limit as stated in the license, swp will be able to reconstruct $u$’s pseudonym after collecting all the tokens. Thus, we meet Req. V asking for license enforcement.

F. Practicality

For the scheme to be used in practice, it has to be efficient and the overhead for the parties has to be low. Some considerations concerning the overhead for the different parties invested during the protocol phases are given in Tab. I. The overhead for $u$ is low in general. The generation of the used keys is done transparently by the DRM software. The concept of anonymous digital cash might, however, involve some challenges for $u$. The ttp’s overhead is relatively low as well—it does not constitute the bottleneck of the system even if it is involved during each software execution. The re-encryption of the software id constitutes the most complex task for ttp. The swp’s tasks do not involve any complex operations. The only drawback of the ring signature is that if the list of approved swps is long, swp needs to include all of those swps in the signature. The high overhead for cc is in terms of communication overhead as cc might need to download a high number of software products during the system initialization. If the software is not downloaded during this phase, it needs to be downloaded on-demand.

To sum it up, our proposed protocol meets all the requirements as stated in Sect. IV-E.
TABLE I
OVERHEAD OF EACH PARTY INVESTED DURING THE PROTOCOL PHASES.

<table>
<thead>
<tr>
<th></th>
<th>low, medium (obtaining digital coins from bank; registration with ttp)</th>
<th>medium (registration of us, ccs, sw, sup)</th>
<th>low (registration with ttp; setup of anonymous recipient scheme)</th>
<th>low (high) (registration with ttp; obtaining sw involves communication overhead)</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Initialization</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>Purchase</strong></td>
<td>low, medium (generation of a new keypair; payment for sw towards sup)</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>Token Request</strong></td>
<td>low (generation of re-encryption key)</td>
<td>medium (license check; re-encryption of software id; token generation; (execution of the secret sharing scheme))</td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>Execution</strong></td>
<td>low (token submission)</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>Token Collection</strong></td>
<td>-</td>
<td>-</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

VII. CONCLUSION AND OUTLOOK

We have come up with a privacy-preserving DRM solution that is practical for a (future) cloud computing scenario. Our solution allows the software provider to provide different price models for software, e.g., an *execute at most n-times* model. Such differentiated price models are not supported by state-of-the-art research approaches towards privacy-preserving DRM.

DRM systems that are used in practice today do mostly not support privacy protection, i.e., it is possible for content providers to link requests to an identity and build usage profiles and thus threatening the right of personal integrity. [6]

Combining different price models and privacy protection is the main contribution of this paper. To achieve this goal, we propose a system that securely puts together different cryptographic primitives such as ring signatures and an anonymous recipient scheme. We propose to employ secret sharing in a unique manner that allows the software provider to expose the user’s identity if the user commits fraud.

ACKNOWLEDGMENT

This work was partially supported by the German Research Foundation (DFG) within the Collaborative Research Centre On-The-Fly Computing (SFB 901). Joshi performed his work on the paper while visiting the University of Paderborn, supported by the RISE program.

REFERENCES