A privacy-friendly architecture for future cloud computing

Ronald Petrlic*, Stephan Sekula and Christoph Sorge

Department of Computer Science,
University of Paderborn,
33098 Paderborn, Germany
Email: ronald.petrlic@uni-paderborn.de
Email: sekula@live.uni-paderborn.de
Email: christoph.sorge@uni-paderborn.de
*Corresponding author

Abstract: We present a privacy-friendly architecture for a future cloud computing scenario where software licensing and software payment plays a major role. We show how digital rights management as a technical solution for software licensing can be achieved in a privacy-friendly manner. In our scenario, users who buy software from software providers and execute it at computing centres stay anonymous. At the same time, our approach guarantees that software licences are bound to users and that their validity is checked before execution. Thus, digital rights management constitutes an incentive for software providers to take part in such a future cloud computing scenario. We employ a software re-encryption scheme so that computing centres are not able to build profiles of their users – not even under a pseudonym. We make sure that malicious users are unable to relay software to others.

Keywords: future cloud computing; digital rights management; privacy protection; anonymous payment.


Biographical notes: Ronald Petrlic is currently pursuing his PhD in Computer Science at the University of Paderborn. He received his Master of Science degree in Computer Science from the University of Paderborn, Germany. His areas of interest are privacy protection in DRM for cloud computing and privacy protection in smart metering.

Stephen Sekula completed his Bachelors degree from the University of Paderborn, Germany. He is especially interested in IT and network security as well as privacy protection. Further, he wants to focus on human-computer interaction (user experience).

Christoph Sorge has a PhD from Karlsruhe Institute of Technology. He worked as a research scientist at NEC Laboratories Europe before joining the University of Paderborn as an Assistant Professor in 2010. His research interests include network security and privacy, both from a technical and legal perspective.

This paper is a revised and expanded version of a paper entitled ‘Privacy-preserving DRM for cloud computing’ presented at the ‘26th International Conference on Advanced Information Networking and Applications Workshops’, Fukuoka, March 2012.

1 Introduction

At the moment, cloud computing is a trend that is said to be revolutionising the IT world in an unprecedented way. Simply put, it is a provision of unused computing resources. However, the convenience of this provision is what makes up the advantage for both the provisioning computing centres with unutilised resources and the customers in need for resources but afraid of high (initial) investments in self-hosting computing centres. Cloud computing is not only about hardware resource provisioning, a service called utility computing, but it also allows for the provisioning of Software as a Service (SaaS). The data centre’s hardware and software is actually what Armbrust et al. (2009) call a ‘cloud’. Despite all the advantages of cloud computing, there are still some issues that are not resolved yet, such as security aspects of provided services and privacy issues concerning users’ data. Moreover, as in the case of SaaS, business models of cloud computing providers are predominantly based on financing the software provision via target-oriented user advertisement – as users’ preferences can be easily observed by providers when they
analyse their software usage in combination with the data processed by the users. In a future cloud computing scenario, the business models for cloud providers might change as users do not accept target-oriented advertisement any longer but are willing to pay a small amount of money for each software execution. Moreover, (potential) users might be afraid of putting their personal or company-related data into the cloud as it is not clear how well that data is protected by the providers or in which countries that data is being processed. Thus, one can imagine a user who wants to use a certain software provided by an SaaS provider but is not willing, or not allowed as restricted by law, to use it because the SaaS provider operates its computing centre in a country where the user does not want its personal data to get processed. In this case, a decoupling of the software provision and the software execution is desirable. This means that there are software providers that develop and provide software and users are free to choose the computing centre where they want to execute the software. The separation between the provision of software and its execution facilitates the market entrance of software providers and computing centres as they can focus on their core business.

As soon as the software provider is not in control of the execution of the software any longer, some sort of copyright protection is needed. As Bauckhage (2003) points out, copyright protection is the only way the producer of digital goods, in our case the software provider providing software, is able to regain the initial investments and keep the status of monopolistic power against competition until innovative goods displace its own: i.e. the software provider needs an incentive to develop software and take part in such a future cloud computing scenario.

Digital Rights Management (DRM) enables the software provider to provide users differentiated price models for their software, reaching from execute at most $n$-times-like models to flat-rate models. Thus, software providers do not get paid just once for the provision of software, they rather sell licences that restrict users in using the software and making them pay for another licence if they want to use the software regularly. For users, on the other hand, differentiated price models allow them to choose the proper licence for a software they expect best fits their demands. For example, a user who knows that he/she will execute the software just a couple of times will rather buy a cheaper licence that allows the execution of the software at most $n$ times than a licence that allows flat-rate usage of the software.

As users pay software providers for the provision of software and computing centres that execute the software, a proper anonymous payment scheme has to be found that best fits such a future cloud computing scenario. Most importantly, this payment scheme must not violate any properties of the DRM solution. Hence, we consider payment methods that take users’ privacy into account and thereby enable them to pay software in an anonymous fashion. In an anonymous payment scheme, there usually are three entities to consider. These are the bank, the user (sometimes also called customer) and the merchant. In the considered scenario, the cloud providers can be considered as merchants, since each one of them offer users software or services which they can purchase. Software providers should receive wages for the provided software licences, and computing centres offer their services to execute software that require payment.

The privacy protection challenge has been pointed out by skeptics of cloud computing and is seen as a serious obstacle for cloud computing by Armbrust et al. (2009). They argue that if sensitive data is stored encrypted within the cloud, they might be even more secure than within a local data centre. Pearson (2009) comes up with the ‘top six recommended privacy practices for cloud system designers, architects, developers and testers’ that head for the same direction. The minimisation of personal information sent to and stored in the cloud and the protection of personal information in the cloud, besides a maximisation of user control – just to mention a few of the tips – are seen as an answer to the privacy protection challenge. A fully homomorphic encryption scheme that allows operations to be conducted on the encrypted data without revealing any (plain) data to the cloud provider might be used. However, an actual scheme, as proposed by Gentry (2009), requires a high computational complexity and is thus impractical. Even though such an approach might reduce the privacy challenge, as even the cloud provider that works with the (sensitive) data is not able to access the data in clear text, we think that such measures fall short of privacy protection. One has to keep in mind that not only the data itself may be ‘personal’, but the fact of who accesses data – even if encrypted – may pose a privacy threat by itself as well. Most cloud services require customers’ identification and authentication. Thus, the cloud service provider knows which customer accesses its service how often, etc. In a future cloud computing scenario, usage data must be seen as personal data as well and a privacy-friendly architectural – as presented in this paper – needs to focus on protecting the user from any parties gaining that data as well.

This is a revised version of the paper by Petrlic and Sorge (2012). We come up with a privacy-friendly architecture for future cloud computing. Our main contributions in this paper are:

- We are the first to come up with an integrated solution that provides both a technical solution for a very flexible DRM as well as an anonymous payment scheme that supports variable charging for software and thus best fits a future cloud computing scenario.
- Our solution prevents any party from building a user profile (not even under pseudonym).

The remainder of this paper is structured as follows. Related work is covered in Section 2. In Section 3 we come up with the requirements for a future cloud computing system. We present the architecture and the corresponding protocol of our proposed concept in Section 4. We discuss the concept in Section 5 before we conclude and give an outlook in Section 6.

2 Related work

Perlman et al. (2010) present a privacy-preserving DRM concept that allows users to buy content without revealing
A privacy-friendly architecture for future cloud computing

which content is bought. Their first scheme is based on anonymous cash. The distributor blindly signs cash that is bought by users. Users spend the cash to buy content from the same distributor. Their second scheme is based on blind decryption. Users ask the distributor to decrypt an encrypted decryption key to decrypt the content. The decryption key is encrypted with the distributor’s public key, so only the distributor is able to decrypt the key when the user pays for the content. To prevent the distributor from learning which content the user will decrypt with the key – as each content is encrypted with a different key – the user blinds the encrypted key during the request. The authors conclude that the second scheme is more efficient than the one based on anonymous cash. A shortcoming of their concept is that it does not allow for different licence models – models that limit the number of executions of the content are not supported, as the licence cannot be checked each time the content is executed. Instead, the content decryption key is bought once and can be used as many times as wanted. Moreover, the blind decryption scheme lacks variable charging for content.

Conrado et al. (2004) present a privacy-preserving DRM concept that allows users to anonymously buy content from a provider and that prevents the tracking of users while the content is accessed. Users’ identities are disassociated from their content. The authors introduce temporary pseudonyms that are managed by users’ smart cards. Their main assumptions are that users contact the provider by means of anonymous channels, that the smart cards do not reveal any Personally Identifiable Information (PII), and that content can only be accessed by compliant devices – devices that behave according to the DRM rules. The provider may be able to build user profiles by learning the association between the smart card’s public key and the content ID during the buying phase. Tracking is only possible for a limited number of transactions as the user’s pseudonyms change periodically. The authors also present a solution to support countable rights licences, i.e. where the content can be accessed only a limited number of times. For that purpose, they introduce an additional domain manager device that creates personalised user rights. As the authors conclude, the introduction of this additional device introduces some complexity.

Research concerning enforceability of DRM is mainly on the trusted environment of customers’ machines that should check the licence before decrypting the digital content and prohibiting easy copying after the decryption (Erickson, 2003; Reid and Caelli, 2005).

An extension of the privacy-preserving DRM scheme presented by Petric and Sorge (2012) is presented by Petric (2012b). It is suggested to employ an adapted version of proxy re-encryption according to Ateniese et al. (2006) instead of the combination of secret sharing and homomorphic encryption. As the scheme presented by Petric (2012b) operates in the same scenario as by Petric and Sorge (2012), the suggestions in terms of integration into a future cloud computing scenario made in the paper at hand also do apply if the approach based on proxy re-encryption is used.

Petrlic (2012a) has come up with a trusted computing-based DRM solution for cloud computing. The author assumes that software is packed into Virtual Machines (VMs) in the future and the VMs are sent to computing centres for execution. The licence enforcement is performed by a DRM system within the VM in this approach. The VM attests its authenticity and integrity towards the computing centre before being executed so that the computing centre can be sure that the VM is not compromised and performs its task of licence checking.

Another solution towards privacy-preserving DRM in cloud computing is presented by Joshi and Petric (2013). Their approach, which provides unlinkability of software executions, is based on a combination of ring signatures with an anonymous recipient scheme. Secret sharing allows the disclosure of user identities only in the case of fraud. One of the first anonymous payment schemes was presented by Chaum (1985). This payment scheme is based on blind signatures which allow users to withdraw and use money without the bank being able to trace them. The payment method makes use of digital signatures based on the RSA algorithm (Rivest et al., 1978). Another payment scheme was presented by Tewari et al. (1998). It uses the technique of secret splitting (see Shamir, 1979) to shadow a user’s identity. This payment method can be used in an offline fashion so as to reduce required communication with the bank. However, the scheme by Tewari et al. requires each entity that wants to receive payments to own a trusted device issued by a central entity. This adds security but burdens the user.

3 System model

In this section we present the basic approach of a privacy-friendly architecture for future cloud computing, discuss the assumptions and point out the requirements.

3.1 Basic concept

We assume that in a future cloud computing scenario, there need to be some dedicated parties – we call them service providers – that act as agents for the users to facilitate the use of cloud computing. Those service providers handle the storage of the users’ software within the cloud. They are also responsible for checking the licences before allowing the software execution.

Users authorise a service provider to buy a certain software from a software provider. However, the service provider should not learn which software is bought. The software provider provides the service provider with an encrypted version of the software together with the corresponding licence. Each time the user wants the software to be executed he/she instructs the service provider to initialise the software execution at a certain computing centre – the user may freely choose the computing centre each time. The service provider checks whether the licence allows the execution and provides the computing centre with the encrypted software to be executed. Both the user and the service provider must cooperate to reconstruct the key that is used by the computing centre to decrypt the
software. Our proposed protocol is based on a combination of secret sharing and homomorphic encryption. The service provider is paid for storing the encrypted software and performing the licence checking. The software provider is paid for the software provision and the computing centre is paid for the software execution.

3.2 Assumption
We have the following assumptions for our concept.

3.2.1 Availability of a public key infrastructure (PKI)
The PKI is needed to issue certificates to the involved parties. We denote the user’s public key by \((e_u, N_u)\) and the corresponding private key by \((d_u, N_u)\). The service provider possesses the key pair \((e_{SP}, N_{SP})\) and \((d_{SP}, N_{SP})\), the software provider the pair \((e_{SWP}, N_{SWP})\), \((d_{SWP}, N_{SWP})\) and the computing centre the pair \((e_{CC}, N_{CC})\), \((d_{CC}, N_{CC})\). As the user needs to be authenticated by the service provider to initiate the buying, payment and execution of software, the service provider needs to know the (registered) users’ public keys. The service provider must not be able to trace back and identify users based on their public keys – only in case of fraud is the relation between the public key and the user identity disclosed by the PKI. The user needs to know the public keys of the software providers he/she buys software from and those of the computing centres that should execute the software.

3.2.2 Honest- but-curious service provider
We assume that a service provider follows the protocol but at the same time tries to gain as much information from the protocol as possible – i.e. a service provider may try to find out which software is bought and executed by the user. Thus, users can not trust their service provider concerning profile building of their software usage.

3.2.3 Multitude of software provided by software provider
If a software provider offered only a single product, it would be easy for the service provider to learn which product is bought by its users from that software provider.

3.2.4 Certified computing centres
Computing centres must be certified by a trusted third party. The certification states that computing centres do not use the software to make money by providing the execution to other users who have not paid for the software. Note that we do not want to restrict the actual used sort of certification. We can assume that contracts are in place that forbid computing centres such bad practices. Failure to comply can have legal ramifications. We are aware of technologies like trusted computing, which could help in enforcing contract terms to a certain extent – at least theoretically. However, this paper’s focus is not on the enforcement of such terms.

3.2.5 No cooperation to break user privacy
The service provider must not share its knowledge about users, meaning the pseudonym, with the software provider or the computing centre. Further, the entities involved in the payment process do not cooperate in order to get access to such information. Otherwise they would be able to build user profiles under a pseudonym. Note that users are free to decide which service providers, software providers and computing centres to choose and to trust, e.g., based on their reputation. Thus, users could choose those entities from different countries in order to minimise the likelihood of cooperation among them. Therefore, this assumption is realistic in a practical scenario.

3.3 Requirements

3.3.1 Economics
The shift from cloud computing towards future cloud computing entails a new way of software licensing. Users will ask for software renting models rather than buying software – as this can be cheaper for users when they execute the software only several times. Price models that allow the execution of software at most n times or up to a certain day are imaginable. Furthermore, users will not want that bought software is bound to a certain computing centre – they will require being able to freely choose the computing centre where the software shall be executed, based on price, SLAs, etc.

The only exception is that a user is not allowed to execute the software on its own platform. This would allow for a bypassing of the DRM scheme in place. This is why we rely on certified computing centres, as pointed out in Section 3.2.4.

If the software provider disappears from the market, users who have bought the software and have a proper licence available should still be able to execute the software. This would not be the case in state-of-the-art cloud computing where users cannot use the software any longer as soon as the SaaS provider abandons its service.

3.3.2 Security
In order to protect software providers, users must not be able to relay software to others. In order to allow software providers to be sure that licences are checked and thus that their provided software is not executed illegally, some party – which can be held responsible – must check the licence before the user is allowed to execute the software. No data must leak to unauthorised parties and actions must be assignable to users.

3.3.3 Payment aspects
Since the usage of a payment scheme could introduce further problems into the system, there are additional aspects that should be considered. First, digital money, often also referred to as ‘coins’, is easily copyable. Therefore, a payment method has to be used that enables the bank to trace impostors.

The payment scheme should work in an offline fashion so as to reduce communication overhead and the load on the bank:
i.e. the merchant is not required to contact the bank to check the authenticity of the coin whenever receiving a coin from a user.

Finally, payments should be efficiently executable while maintaining security – reducing the computational overhead. Thus, the number of required digital signatures per transaction should be minimal.

3.3.4 Privacy protection

Cloud computing is problematic as personal data is stored and processed at any computing centre in the cloud. Note that this applies for companies as well – however, in this case we rather talk about business secrets than about privacy. In most cases, it is not clear in which countries the computing centres are, which privacy regulations apply, etc. We are addressing requirements concerning privacy protection within the cloud for our scenario of software buying and execution.

Privacy protection is the protection of personally identifiable information (PII): i.e. ‘information which can be used to distinguish or trace an individual’s identity’ (McCallister et al., 2010). Personal data is ‘information relating to an identified or identifiable natural person’ (Directive 95/46/EC, 1995). There may exist certain information that does not (directly) allow for an identification of individuals but that relate to natural persons and thus have to be handled with care. In our scenario the PII are identities that reveal who buys and executes software. Personal information is information like which software is bought and how often it is executed. To prevent the revealing of PII we ask for anonymity, and to prevent the revealing of personal data we ask for prevention of profile building.

Anonymity means that there is no way to relate individual information to an identified or identifiable natural person. In our scenario, neither the software provider that sells software nor the computing centre that executes software must be able to find out who buys and executes software. The users should also stay anonymous towards the service provider that handles the user interactions with the software provider and the computing centre. Furthermore, even payments towards these entities should not reveal the users’ real identities.

Profile building means that a party is able to relate actions to certain users. There is a special form of profile building called profile building under pseudonym. This form of profile building does not relate actions to certain users but relates them to certain pseudonyms. A pseudonym is an identifier other than the real name that is relatable to a certain person if the mapping is known. Even though an attacker does not know the mapping and thus is not able to directly map the pseudonym to a certain person, having a profile under pseudonym may suffice to differentiate between users and indirectly identify a certain person again. Such profiles under pseudonym may be built if a user appears towards a party under the same pseudonym more than once. Note that preventing profile building under pseudonym implies preventing profile building (under real identity).

In our scenario, profile building prevention means that users cannot be tracked during software buying and software execution. If a user buys software from a certain software provider more than once, the software provider must not be able to learn that this user bought software before. If a user executes a certain software more than once at some computing centre, it must not be possible for the computing centre to relate those executions to each other: i.e. executions must be unlinkable to each other. Profile building prevention under pseudonym is given if unlinkability of actions (Pfitzmann and Kohntopp, 2009), such as software buying and software execution, is met.

3.3.5 Posed requirements

Table 1 provides an overview of the requirements.

| Req. I | Different licence models have to be supported. |
| Req. II | The software licence must not bind the software execution to a certain computing centre. |
| Req. III | The software must be protected so that it cannot be relayed by a user. |
| Req. IV | Data confidentiality against unauthorised parties and non-repudiation of users’ actions must be achieved. |
| Req. V | Fraud must be easily detectable. |
| Req. VI | The payment scheme should be offline. |
| Req. VII | The number of required digital signatures per transaction and coin should be small. |
| Req. VIII | The user should stay anonymous towards the service provider, software provider and computing centre. |
| Req. IX | Profile building (under pseudonym) must not be possible for any party. |

Note that the requirements pointed out in this section are not application-specific, i.e. they apply both for enterprises and for private persons participating in cloud computing. From an economic and security point of view, there is no difference whether a company provides software for cloud computing or whether an individual provides software – both companies and individuals take part for economic reasons and they require some sort of licence management. The same is true for privacy protection as well. On the one hand individuals are concerned about their personally identifiable information and they do not want their habits, in the form of personal data, to get exposed to the public. On the other hand, ‘privacy’ protection for companies is at first hand a protection of business secrets. Any information about internal processes leaked to the outside might reveal information that could be vital for competitors.

4 Concept

In this section we cover the protocol of our future cloud computing concept.

4.1 Software buying

Figure 1 shows the payment process of the software buying. The protocol used was designed by Tewari et al. (1998). In
the scenario that is dealt with, the service provider, software provider, as well as computing centre can be regarded as merchants. This is the case since, in theory, each of these entities is entitled to being paid by the customer. The service provider serves as a proxy for the user so that the user is able to communicate anonymously with the software provider and computing centre. Software providers sell software licences that require the user to pay them as well. Finally, computing centres provide servers that execute software on the user’s behalf so that this service may as well require payment. The payment scheme by Tewari et al. (1998) requires each merchant to possess what is called a POS device. POS stands for ‘Point of Sale’. These devices are issued by a central entity called the issuer that also mints coins. Since these devices are involved in every payment transaction, they need to be trusted. An important aspect about the payment method designed by Tewari et al. is that coins can have denominations. Therefore, different from other anonymous payment schemes, payments of larger values could be conducted with fewer transactions, effectively lowering communication and computation overhead.

Each coin contains a transaction list. This list in turn contains transaction items that consist of a fixed number of identity strings (see Figure 2). These identity strings are the result of a secret splitting (see Shamir, 1979) technique which splits the user’s ID into two parts using a one-time pad. An identity string is thus the randomly generated one-time pad $R$ and the result of $R \oplus ID$ (here, $\oplus$ denotes the XOR operation). Every coin has as many transaction items in its transaction list as it had owners. Therefore, if the transaction list of a coin has $m$ transaction items, the coin has been traded $(m - 1)$ times. The last of these transaction items is in clear, so that the ID of the current coin owner can be found by using the XOR operation. All of the other transaction items are blinded though, which means that either the one-time pad or the result of the XOR has been randomly set to 0. This is the task of the POS device that blinds the transaction items during the payment process so that the merchant cannot acquire the user’s ID. If the user committed fraud (i.e. copied a coin), however, the user’s identity can be revealed with a very high probability (see Section 5.3). In theory, implementing this payment method in the described scenario could take place in two different ways. Since the POS devices themselves shadow the user’s identity, they serve as a proxy between users and merchants. Therefore one could on the one hand think of the user being able to directly pay each above-mentioned entity separately (i.e. service provider, software provider and computing centre). This has the advantage that the anonymous payment scheme provides an additional layer of privacy for the user, since the service provider will not obtain any information about the user’s payments. On the other hand, since service providers already take on the role of the user’s proxy, payments could also be conducted by using them. This way, the user would pay the service provider which in turn pays the software provider and computing centre (dotted line in Figure 1). This is possible only since the payment scheme described by Tewari et al. (1998) allows for coins to circulate in the system for a specific time. The validity period and the number of payments a coin is involved in can limit this time. Nevertheless, using the latter method of paying every entity in the system, the user would be required to tell the service provider the amount of money he/she owes the software provider and the computing centre. If those entities offer products with varying prices, the service provider may deduce information about which product the user has purchased. In this case, instead of using an anonymous payment scheme, a regular credit card payment could be used. But, since we want to prevent service providers from getting to know payment information, the user should pay the service provider, software provider and computing centre directly, using their respective POS devices.

Figure 1 The payment process in the described setting (customer denotes the user of the system) (see online version for colours)
The user signs the software buying messages and the service provider verifies the validity. Furthermore, it exchanges a key $K_{USWP}$ with the software provider by using the DH key exchange.

4.2 Software and licence retrieval

Figure 3 shows the software and licence retrieval protocol. This protocol is executed after the software buying. The involved parties are once again the user, the service provider and the software provider.

Figure 3 Message sequence chart showing the software and licence retrieval protocol executed between user, service provider and software provider

4.2.1 Software encryption

To prevent the encrypted software from being related to a specific software by the service provider by analysing its length, padding is employed to the software before encryption. In Step 1 the software provider provides the service provider with the encrypted software $SW_x$ – denoted by $Enc_{K_x}(SW_x)$. The employed encryption scheme, based on modular addition (+) and specified in Table 2, is as follows: $Enc_{K_x}(SW_x) = SW_x + K_x \mod M$, where $M$ is a large integer.

<table>
<thead>
<tr>
<th>Transaction list:</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Transaction item 1:</strong></td>
</tr>
<tr>
<td>$i_L$</td>
</tr>
<tr>
<td>1</td>
</tr>
<tr>
<td>2</td>
</tr>
<tr>
<td>3</td>
</tr>
<tr>
<td>n-1</td>
</tr>
<tr>
<td>n</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th><strong>Transaction item m:</strong></th>
</tr>
</thead>
<tbody>
<tr>
<td>$i_L$</td>
</tr>
<tr>
<td>1</td>
</tr>
<tr>
<td>2</td>
</tr>
<tr>
<td>3</td>
</tr>
<tr>
<td>n-1</td>
</tr>
<tr>
<td>n</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Current owner:</th>
</tr>
</thead>
<tbody>
<tr>
<td>$i_L \oplus i_H = 8892577$</td>
</tr>
</tbody>
</table>

4.2.2 Secret sharing

The idea behind the encryption of the software is that neither the user nor the service provider gets the (symmetric) decryption key $K_x$ needed to decrypt $SW_x$. This is because the user is not trusted by the software provider. If the user had the key, he/she would be able to decrypt the software and share it with others. The service provider does not get the key because otherwise it would know which software has been bought by the user and how often it is executed. We propose a scheme that is based on secret sharing – the decryption key is subdivided into two parts, one for the user and one for the service provider, $ShareU_{K_x}$ and $ShareSP_{K_x}$.

We employ an additive secret sharing scheme, i.e. $K_x = ShareU_{K_x} + ShareSP_{K_x}$, where $ShareU_{K_x}$ is chosen randomly and $ShareSP_{K_x}$ is chosen accordingly.

Table 2 Additively homomorphic encryption

<table>
<thead>
<tr>
<th>Encryption</th>
<th>Decryption</th>
</tr>
</thead>
<tbody>
<tr>
<td>Message $m \in [0, M - 1]$, where $M$ is a large integer</td>
<td>$Dec(c, k, M) = c - k \pmod{M}$</td>
</tr>
<tr>
<td>randomly generated keystream $k \in [0, M - 1]$</td>
<td></td>
</tr>
<tr>
<td>$c = (m + k) \pmod{M}$</td>
<td></td>
</tr>
</tbody>
</table>

Source: Castelluccia et al. (2005)

$SW_x \in [0, M - 1]$ and $K_x \in [0, M - 1]$ has to apply. The key $K_x$ is unique, i.e. whenever a software is bought, the software provider generates a new key with the RC4 stream cipher. We employ the scheme for the software encryption without making use of the homomorphism property – the reason why we still employ this scheme will become clear when we cover the software re-encryption in Section 4.3.2.
4.2.3 Software licence

The encrypted software comes with the corresponding licence licence in Step 1. Licence is a data structure that contains the following fields: encrypted software identification, validity period, pricing model, digital signature and $\text{ShareSP}_K$. The encrypted software identification does not contain information about the software but rather helps the service provider identifying the user’s encrypted software. The validity period and the pricing model (e.g., the number of allowed software executions or flat rate) are checked by the service provider each time the software is intended to be executed. The licence is digitally signed by the software provider. $\text{ShareSP}_K$ is the service provider’s share for the software decryption key $K$. The relation between user and licence is managed by the service provider – the software provider is not aware of the relation between user and licence and thus the relation is not stored within the licence.

4.2.4 Retrieval

The encrypted software and the licence are not forwarded to the user by the service provider but stored for later usage. A secure channel between the software provider and the user, whose communication is relayed via the service provider, is needed so that the software provider can securely communicate the user’s share $\text{ShareU}_K$ of the decryption key $K$ towards the user in Step 2 – without allowing the service provider to get the share and thus being able to decrypt the software. The software provider employs a symmetric encryption scheme, such as AES (NIST, 2001), and uses the symmetric key $K_{USWP}$ that it exchanged with the user during the buying phase.

4.3 Software execution

Figure 4 shows the software execution protocol. The involved parties are the user, service provider and computing centre. The user encrypts $\text{Msg}_i = (\text{command}, \text{mv}, K_{HU}, \text{coin})$ in Step 1 and sends it to the service provider. The command tells the service provider what to do, like to execute the software at a certain computing centre. The modification vector $\text{mv}$ tells the service provider how to re-encrypt the encrypted software, as covered in Section 4.3.2. The homomorphic user key $K_{HU}$, randomly generated by the user and never reused, is used for the reassembling of the software decryption key, meaning the secret combination, as covered next. The coin denotes the payment for the software execution – compare Section 4.1 for the description of the employed payment scheme.

Message sequence chart showing the software execution protocol executed between the user, service provider and computing centre.

**Figure 4** Message sequence chart showing the software execution protocol executed between the users, service provider and computing centre.
Moreover, in order to support an execute at most n-times model, the service provider needs to keep track of executions. For that purpose, the service provider keeps a list where each software execution of a user is logged. If the value exceeds the specified maximum allowed value, as denoted by pricing model in the software licence in Section 4.2.3, the service provider does not allow the execution of the software.

4.3.1 Secret combination

The user and the service provider send their share values to the computing centre for the computing centre to be able to reconstruct the software decryption key. If the user executes a software more than once at a certain computing centre, the computing centre will be able to build a profile (under pseudonym) if it received the user’s or service provider’s share directly. We employ a secret sharing scheme that is based on the homomorphic encryption scheme as described in Table 2. The computing centre will only get the encrypted share values of the user and the service provider and will be provided with a key that allows the decryption of the software decryption key \(K_U\) only – but not the individual encrypted share values.

In Step 2 the user sends the computing centre its modified share value \(msu\) – encrypted with the user’s homomorphic key \(K_{HU}\). The \(msu\) is computed as: \(msu = ShareU_{K_y} + mv\), where \(msu \in [0, M-1]\) has to apply. The whole message, incorporating a common key \(K_{UCC}\) between the user and the computing centre (chosen randomly by the user) as well, is encrypted with the computing centre’s public key so that the service provider, acting as a relaying node once again, does not get into possession of the encrypted \(msu\) and the common key. The common key is needed for the communication between the user and the computing centre – for the provision of input values and the transmission of the result in the last step.

The service provider generates a homomorphic key \(K_{ISP}\) and computes the aggregated homomorphic key in Step 5: \(K_{aggk} = K_{HU} + K_{ISP}\), where \(K_{aggk} \in [0, M-1]\). \(K_{aggk}\) is unique for each transaction as both \(K_{HU}\) and \(K_{ISP}\) are randomly chosen each time. \(K_{aggk}\) allows the computing centre to decrypt the (new) software decryption key \(K_y\). The details of the software decryption key change and the software re-encryption are explained in Section 4.3.2. In Step 6 the service provider sends the computing centre \(K_{aggk}\) along with the re-encrypted software, the service provider’s \(ShareSP_{K_y}\) (encrypted with \(K_{ISP}\)) as well as the coin to pay for the software execution. The message is encrypted with the computing centre’s public key.

The computing centre retrieves the software key \(K_y\) by decrypting the aggregation of the user’s and the service provider’s encrypted shares with \(K_{aggk}\) in Step 7. According to the specification presented in Table 2, this computation is performed as: \(K_y = (Enc_{K_{aggk}}(msu) + Enc_{K_{ISP}}(ShareSP_{K_y})) - K_{aggk} \mod M\). In Step 8 the computing centre decrypts the software as: \(SW_x = (Enc_{K_y}(SW_x) - K_y) \mod M\). The computing centre executes the software, receives input data from the user (Step 9) and returns the computation result to the user in Step 10. The communication between the user and the computing centre is encrypted with the common key \(K_{UCC}\) between both parties.

4.3.2 Software re-encryption

The re-encryption is performed by the service provider after it has checked the corresponding licence (Step 3). As explained above, the user’s \(ShareU_{K_y}\) is modified by the user, resulting in the modified share value \(msu\), before it is transmitted to the computing centre. This modification is needed as we propose that the service provider modifies the encrypted software each time it is sent to the computing centre. Remember that the software encryption key \(K_y\) is changed for each software purchase by the software provider. If the service provider did not re-encrypt the software, the key would stay static as well – allowing the computing centre to build a user profile under pseudonym.

To allow for such a software re-encryption, the user’s share value has to be modified each time the software is executed. This is why we introduced the modification vector \(mv\) in Step 1. The \(mv\) is randomly chosen by the user for each execution. The user’s modified share value \(msu\) is computed by adding the modification vector to the share value. Accordingly, the software is modified, meaning re-encrypted, by adding the same modification vector to the software: \(Enc_{K_y}(SW_x) = Enc_{K_y}(SW_x) + mv\) (Step 4). We denote the fact that the software is now encrypted with the still unknown – key \(K_y\) by the new index ‘\(y\)’. For the decryption to succeed, \((SW_x + mv) \in [0, M-1]\) must hold.

4.4 Improvement of the scheme

Our proposed scheme can be improved in terms of communication overhead. Instead of transmitting \(ShareU_{K_y}\) from the software provider to the user, we may transmit a shorter seed value instead. The seed is used as input to a pseudorandom number generator like RC4 by the user in order to get \(ShareU_{K_y}\). The same approach can be taken for the transmission of the modification vector \(mv\) and the homomorphic user key \(K_{HU}\) from the user to the service provider.

5 Discussion

In this section we discuss our concept in terms of the requirements as pointed out in Section 3.

5.1 Economics

The service provider checks the licence before allowing the software execution. Depending on the licence model, the
service provider decides whether the permission on the part of the software provider, as stated in the licence, is still given. Our concept meets Req. I asking for different licence models. Even if the software provider disappears from the market, the user – given that he/she has a proper licence available – is still able to execute the software as the encrypted software is stored by the service provider and the software provider does not take part during the software execution phase. The user can choose the computing centre where the software is executed. We meet Req. II, the very requirement of flexibility in (future) cloud computing, with our approach.

5.2 Security

The software provider provides the service provider with the encrypted software. The user does not come in possession of the software and is not able to relay it to anyone else. The user needs to contribute its software decryption key share for each software execution. Relaying software would mean for a user to reveal a user’s credential for the service provider, he/she would still need to spy on the share value as well, which should be stored securely by the user. Thus, relaying the authentication credential to another user and arguing that the credential has been retrieved another way might not be an option for a malicious user.

The software providers rely on the service providers to check the licences before initiating the software executions at computing centres. As we have assumed, the service providers are certified by independent parties that check whether the service providers comply with their duties. Any communication between the involved parties is encrypted and particularly the user’s software buying and execution commands are digitally signed to achieve non-repudiation for those actions. Thus, we comply with Req. IV as well.

5.3 Payment aspects

Upon deposit of a coin, the bank checks whether it has stored a coin with the same serial number already. If this is the case, these two coins are compared. Each coin contains a transaction list with transaction items that are the result of the secret splitting technique that is applied upon payment by the POS device of a merchant. Since the blinding process is executed randomly, there is a very high chance to find a matching set of ID parts that reveal the impostor’s ID when brought together. This probability is dependent on the number of times this coin has been spent and the number of identity strings that are contained in one transaction item (compare Figure 5). The more identity strings are used, the higher the probability that fraud is detected, but also the larger the coin. In Figure 6 one can see the increase of the average coin size depending on the number of times a coin has been spent as well as the number of identity strings used in that coin. Hence, there is a trade-off between coin size and security in fraud detection. One could thus consider requiring a larger number of identity strings for coins with a higher denomination in order to be sure that fraud can be detected for these coins (see Tewari et al., 1998). Therefore, we meet Req. V that fraud must be easily detectable.

Figure 5 The probability to detect fraud depending on the number of identity strings used and times the coin has been spent (see online version for colours)

Figure 6 The average size of a coin depending on the number of times it has been spent and the number of identity strings used (see online version for colours)

Each coin contains identifying information of the user which is its splitted ID. Only the last transaction items contained in the transaction list of a coin is unblinded so that a simple XOR operation can be used to figure out the ID of the current owner of a coin. All other transaction items different from the last one are blinded and hence it is not possible to retrieve the ID of previous owners of a coin. This is possible only if fraud has been committed and hence multiple copies of the same coin are available. Then the impostor’s ID can be extracted from those copies of the coin. Thus, received coins need not be deposited with the bank immediately to check them for authenticity, which means that the payment scheme is off-line. Therefore, we meet Req. VI of the used payment method to be offline.

Finally, if we take a look at the performance of the payment scheme, only three digital signatures need to be computed per coin, where each payment will add one computation of a signature. Nevertheless, there need to be eight checks of digital signatures if we take into account the steps from minting a coin until final deposit, if this coin has been used in one transaction. An advantage, which is provided by the payment scheme described by Tewari et al.
(1998), is the use of different denominations. This allows for one coin to be worth 1000, for example, which relativises the amount of signatures needed, since in this example, one would only have to conduct one transaction as compared to 1000 one dollar transactions. In sum, this payment scheme is a rather efficient one, so that we meet Req. VII as well, which calls for the number of required digital signatures per transaction and coin to be small.

Additionally, the user’s privacy is protected even towards the service provider, since an anonymous payment method is used instead of regular credit card payments.

5.4 Privacy protection

We discuss the privacy aspect of our concept in terms of anonymity and profile building prevention. As we have pointed out in Section 3.3.4, anonymity is about not revealing any PII and profile building prevention is about not revealing personal data, such as which software is executed how often by some party. Note that we have also stated in Section 3.3.4 that if we can prevent profile building under pseudonym, we implicitly prevent profile building (under real identity).

The user only directly communicates with the service provider – the other parties do not get identifying information like user data, network addresses, etc. The service provider does not forward the digital signatures corresponding to the software buying and execution commands towards the software provider and computing centre. The anonymous payment scheme by Tewari et al. (1998) is used to pay for the software, its storing and licence checking and its execution. Since the POS devices blind the user’s identity, the latter can be sure that he/she will stay anonymous towards his transaction partners as long as they do not commit fraud. We meet Req. VIII which asks for anonymity.

The software provider takes part only during the software buying and the software and licence retrieval. There is no way for the software provider to build a profile of its (unknown) users under pseudonym. This would be possible for the software provider if it checked the licence by itself as it could include information into the licences that allowed for a unique identification.

We prevent the computing centre from profile building under pseudonym by changing the user’s software decryption key share and the associated software re-encryption each time the software is sent to the computing centre. If we did not change the share, the computing centre would know that the software is executed by a user that executed it before. The change of the share value and the associated software re-encryption also prevents collaborating computing centres from profile building.

The modified user’s share value and the service provider’s share value are sent towards the computing centre encrypted by employing a homomorphic encryption scheme. The computing centre must not learn any single (modified) share value. The homomorphic keys $K_{KH}$, $K_{ISP}$ and the aggregated key $K_{agg}$ are also changed for every transaction. The computing centre is not able to re-identify previously used keys. The key renewal guarantees that the homomorphic encryption scheme is provably secure, as proven by Castelluccia et al. (2005).

The service provider must not learn which software is executed by a certain user to prevent profile building under pseudonym – note that the service provider knows the user’s pseudonym as the service provider stores the encrypted software for the user. $SW_j$ is encrypted with $K_j$ by the software provider and sent to the service provider. As the service provider only gets its own share value of the key, it is not able to reconstruct the key $K_j$ and decrypt the software on its own. The service provider does not learn the user’s (modified) share value as it is transmitted encrypted. The secrecy of the software encryption scheme is based on the secrecy of Castelluccia et al. (2005). The scheme is proven to be provably secure if the employed key is not reused. The key $K_j$ to encrypt $SW_j$ is unique – we employ the RC4 stream cipher to generate the keystream. The re-encryption employs a new modification vector $mv$ each time. The service provider does not learn the result of the software execution as the computing centre encrypts the message with $K_{CC}$ (known by the user and the computing centre).

The service provider cannot act as an MITM between the user and the software provider during the software buying as the user obtains the software provider’s public key via an independent channel, such as the PKI, and encrypts the message towards the software provider with this key. This is true for the communication between the user and the computing centre as well and thus the service provider is not able to fool the user into executing its software within its own computing centre and thereby learning the software.

The payment scheme does not reveal any important information of the user since his/her ID is blinded upon payment. Therefore no entity can track a user only because of a payment transaction that this user conducted. Regarding that the software provider can charge different prices for different software, we proposed that the user pays the service provider, software provider and computing centre separately: i.e. the service provider does not learn how much the user pays to the software provider and thus cannot conclude which software is bought.

As we have assumed, the parties must not cooperate to break the user’s privacy. In case the service provider and the software provider cooperate, we lose profile building prevention under pseudonym against the service provider as well as against the software provider. The service provider knows the user’s pseudonym – not the real identity, though – and the software provider knows which software is sold. The same is true if the service provider and the computing centre cooperate as the computing centre knows which software is being executed as well. This is why a data exchange between the service provider and the software provider as well as the computing centre must be prohibited by contract or law.

A privacy-friendly architecture for future cloud computing
To sum it up, we meet Req. IX which asks for the prevention of profile building (under pseudonym).

5.5 Correctness of the protocol

The scheme by Castelluccia et al. (2005) is additively homomorphic due to the commutative property of addition. The software provider provides the service provider with \((SW_i + K_y) \mod M\). The service provider additionally adds \(mv\) to the encrypted software. We required \(SW_i \in [0, M-1], K_y \in [0, M-1]\) as well as \((SW_i + mv) \in [0, M-1]\). This last requirement is necessary as the modification of the software must not lead to an overflow. \(K_y\) is computed by the computing centre as: \(K_y = ((\text{ShareSP}_{K_y} + \text{ShareU}_{K_y} + \text{ShareH}_{K_{ID}}) \mod M\). The decryption of the software is performed as: \(SW_i = (SW_i + K_y + mv - K_y) \mod M\).

In summary, our protocol is correct and we meet all requirements as stated in Section 3.

6 Conclusion and outlook

We have pointed out that privacy protection will become more important in the cloud computing scenarios of the future. At the same time, DRM as well as proper payment concepts are crucial for software providers to take part in future cloud computing. One major design goal of our privacy-friendly architecture is the protection of users’ privacy. The homomorphic encryption-based secret sharing scheme, combined with the software re-encryption scheme makes sure that users stay anonymous towards software providers and computing centres and that profile building is not possible – not even under pseudonym – for any party. The introduced payment scheme even supports privacy protection and provides a solution to detect fraud as well.

Acknowledgements

This work was partially supported by the German Research Foundation (DFG) within the Collaborative Research Centre ‘On-The-Fly Computing’ (SFB 901)

References


