Controlling Data-Flow in the Cloud

Mandy Weißbach and Wolf Zimmermann
Institute of Computer Science, University of Halle
06120 Halle (Saale), Germany
Email: {weissbach, zimmermann}@informatik.uni-halle.de

***Abstract*—A big obstacle for using cloud services is that users have no control over the locations where their data are stored or processed, respectively. This paper presents a program analysis approach that enables clients to negotiate services with undesired locations. Clients may only use services that guarantee not to use (directly or indirectly) services on undesired locations for processing or storing the clients' data. In order to increase trust in the answers given by services during the negotiation process, a cryptographic approach similar to Web page certification is proposed. We show that a static data-flow analysis combined with a cryptographic approach ensures that clients' data do not reach undesired locations in the cloud.**

Keywords- data-flow; service-level agreement; cloud security.

I. INTRODUCTION

One major obstacle in using cloud services is that clients have no control where their data are being stored and processed. National data protection laws may require from clients to satisfy some standards. For example, EU-directives imply that it is illegal to pass personal data to environments where the access to data cannot be controlled [1]. Recently, there is an even stronger proposal [2]. This may apply towards storing data as well as to the results from processing data. However, if cloud servers located at different locations, they need to obey national laws on the server's location, and these might be rather different than the location of the cloud users and therefore there might be unauthorized access to clients' data that might be legal in the cloud servers country.

Unfortunately, encryption of data is only a solution when data are just stored (see, e.g., [3]), but it is currently not a solution when they are being processed [4] (some work on directly processing encrypted data exist, but it is just at the beginning and it is not clear whether this research will be successful at the end). Therefore, it is crucial that cloud service users can require that their data are stored in certain locations or exclude some locations for storing and processing their data. But, cloud service providers themselves prefer where to store the data of the service users. Even worse, they may use other cloud services which themselves may use other services and so on. Thus, it seems almost impossible to control where data are processed and stored. Thus, several authors see this issue as one of the major challenges in cloud computing [5], [6], [7].

In this paper, we propose a service-level agreement approach to ensure that the data of cloud service users are not processed or stored at undesired locations. Typical service-level agreement (SLA) approaches such as, e.g., reliability or response time can be measured by service users. If the chosen service violates its assured service quality, the service user is enabled to use alternative services. However, the problem in this work has different characteristics: (i) it is not measurable whether data are not being stored and processed at undesired locations, (ii) a violation cannot be observed by service users, and (iii) if a cloud service violates directly or indirectly the SLA, there already is a possible threat for the service user, i.e., the damage is sustained. Thus, service violations in the context of this work should be avoided, and service users have to trust the agreement.

We tackle the problem of avoiding storing or processing data at undesired location by data-flow analysis. In particular, this analysis ensures that either data do not leave the cloud server hosting the cloud service or the data are only transferred (possibly in processed form) to cloud services ensuring that the data received are neither stored nor processed at undesired locations. This approach enables the cloud service to provide the correct agreement. However, a malicious service may give the wrong answer. We propose cryptographic methods analogous to web page certification in order to increase the trust into the negotiated service-level agreement.

The paper is organized as follows: an example of a service model is provided and explained in Section II. In Section III, a data-flow analysis with respect to the given example is done. Section IV proposes a cryptographic approach in order to increase the trust in the answer given by the service model of Section II. On top of that, Section V presents an approach to choose dynamically a service that can be trusted. Section VI disscuses related work and Section VII concludes this work.

II. APPROACH

This section demonstrates the underlying approach. In our service model, we assume that each service A provides a set of functions, denoted by $Provided_A$. This might be given as a WSDL-description (Web Services Description Language). Furthermore each service A might use other services. We assume that this is not hard-coded in the implementation of A, but there is a variable I a where I contains the set of functions that is called on a, and a can be bound

 ISBN: 978-1-61208-216-5

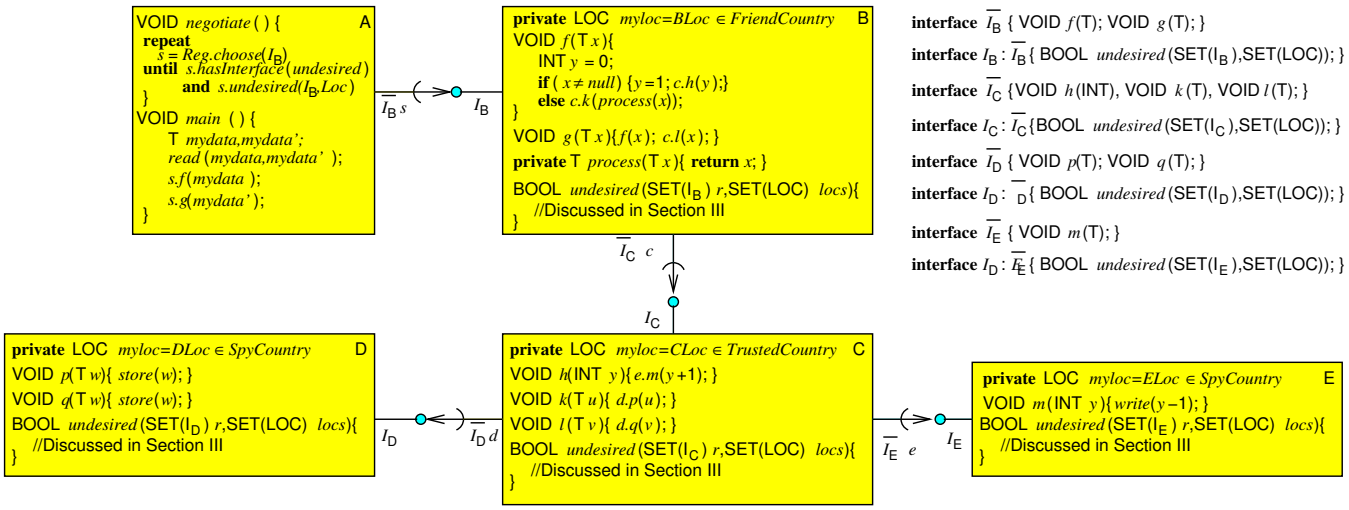

Figure 1. Storing Data at Undesired Locations

(dynamically) to a service X that provides at least I, i.e., $I \subseteq Provided_X$. Functions in I are called *required functions* of A w.r.t. a. The set of candidate services must be published and we assume that a registry Reg maintains all published services. For the purpose of this and the next section, we assume that the use structure is acyclic. Section IV shows how this assumption can be dropped.

Example 1: Consider services A and B in Fig. 1. $A.s$ is bound to service B and $B.c$ is bound to service C. The provided interface of B is $Provided_B = \{f, g, undesired\}$. The required functions of A w.r.t. s are $\{f, g\}$. The required functions of B w.r.t. c are $\{h, k, l\}$. ■

Remark: It is part of service-level agreement approaches that negotiations bind services to these variables. For simplicity, we only consider a set of functions for the selection of candidate services. However, this can easily be replaced by other match-making approaches, e.g., it can be based on contracts or adapters can be included. □

In the context of this paper a client would like to negotiate an agreement that a selected service guarantees to avoid data-flow from the client's data to a set Loc of undesired locations. This ensures that the client's data are not stored at undesired locations. For the purpose of negotiation, service A may offer a function $undesired \in Provided_A$ that returns true iff data flows via some operations o from the provided interface of A to services at undesired locations. It is sufficient to take into account only the set $S \subseteq Provided_A$ of operations used by the client. If A uses another service B, it needs to ask B (via B's function $undesired$) whether it can guarantee that A's data do not flow to a location in $l \in Loc$. Obviously, this needs only to be guaranteed for those operations of B where B passes (possibly processed) data of A. For simplicity, we assume that each service X knows its location and this location is stored in a constant $X.myloc$.

Example 2: Consider the services A, B, C, D, and E in Fig. 1. Service A would like to use service B. Service B is located in *FriendCountry*. B itself uses service C located in $TrustedCountry$ while C uses serviced D and E located in $SpyCountry$. For the example, we assume that all services (except possibly A) are published.

Suppose that client A wants to avoid storing its data neither in their original nor in processed form at servers in *SpyCountry*. Thus, before client A actually uses service B it would like to know whether data passed to B are never stored (neither in original nor in processed form) at a server in $SpyCountry$. Let Loc be the set of servers in $SpyCountry$. The procedure $negotiation$ searches for a published service B offering at least the operations specified in I_B where I_B is the set of functions of the required service that are called from A. For the purpose of negotiation, A calls $undesired(I_B, Loc)$ because A calls $b.f(mydata)$ and $b.g(mydata')$, if b is bound to service B. B calls functions $h, k, l \in Provided_C$ if c is bound to a service C. A call of $B.f$ implies that data of A flow to the calls $c.k(z)$ and $c.l(x)$, but there is no flow from data of A to the call $c.h(y)$. Thus, the call $undesired(I_B, Loc)$ must return $true$ only if $B.myloc \notin Loc$ and $undesired(\{k, l\}, Locs) = true$. Note that function h needs not to be considered because $mydata$ does not flow to y in the call $c.h(y)$. The functions $k, l \in Provided_C$ call $p, q \in Provided_D$ if $C.d$ is bound to a service D. The arguments of the calls $c.k(z)$ and $c.l(x)$ flow to the calls $d.p(u)$ and $d.q(v)$, respectively. Thus, there is a flow from the data of A to service D located in $SpyCountry$ which could store these data. Therefore, the negotiation must fail. $C.undesired(\{k, l\}, Locs)$ must return $false$ and therefore $B.undesired(I_B, Loc)$ returns $false$, i.e., A cannot use B.

Suppose there would be an alternative service D' with the same implementations of p and q, but its location

is in $MyCountry$. Then, if $C.d$ is bound to D' instead of D, $D'.undesired(\{p,q\}, Loc)$ returns $true$ because D' does not use other services. Thus, in this case $C.undesired(\{k,l\}, Locs)$ can return $true$ and therefore also $B.undesired(I_B, Loc)$ returns $true$. Hence, A can use B. Note, that $C.e$ is still bound to service E in $SpyCountry$ but there is no flow from the data of A to E because E is only used in h and there is no flow from data of A to the call $c.h(y)$ in B. ■

In general, if a service A negotiates with a service B for avoiding undesired locations Loc, A must additionally provide the set of functions $O \subseteq Provided_B$ used by A. If B calls functions provided by other services, and data from A flow to B then, B must ensure that these services are neither located in an undesired location nor passed directly or indirectly to a service located in an undesired country. Therefore, B has to negotiate with the required services for assuring that the data of A are never passed to services in an undesired location.

Remark: A cyclic use-relation would lead to non-terminating negotiations. One possibility to overcome this problem is that after a certain time, the negotiation with a service B is interrupted and another service is considered for negotiation, i.e., the function $undesired$ terminates after a certain time and returns $false$. □

III. Data-Flow Analysis

For the implementation of $undesired$, the data-flow from the provided functions to the required functions needs to be analyzed. For such a data-flow analysis, we refer to [8] (an interprocedural def-use-chain is needed). Let x be a parameter of a function $f \in Provided_A$ provided by a service A. The result of the program analysis is a predicate $DEP_{e,x}$ for each argument e of a call f of a required function of A. $DEP_{e,x}$ is true if the value of e depends on x.

Example 3: Consider services A and B in Fig. 1. The function $f \in Provided_B$ has a parameter x of type T. If x is not $null$ the function $h \in Provided_C$ is called. So $DEP_{x,y} = false$ because the value of the argument y does not depend on x.

Remark: The program analysis is *conservative*, i.e., the value of e might be independent of x although the program analysis computes $DEP_{e,x} = true$. However, if $DEP_{e,x} = false$, then it is guaranteed that the value of e is independent of x. An exact computation of $DEP_{e,x}$ would be undecidable. □

Let $T\ f(T_1\ x_1, \ldots, T_n\ x_n) \subseteq Provided_A$. Then, the *slice of* f consists of all set of required functions of A that are called with an argument depending on one of the parameters x_i, i.e.,

$Slice_f \triangleq \{p : \exists s.p(e_1, \ldots, e_k) \in A \bullet \exists i, j \bullet DEP_{e_i,x_j}\}$

The *slice of a set of functions* $S \subseteq Provided_A$ is defined by

$Slice_S \triangleq \bigcup_{f \in S} Slice_f$

Let B be a service that is used by A and $S \subseteq Provided_A$. Then

$Called_{S,B} \triangleq Slice_S \cap Provided_B$

is the set of functions of B called by A that may depend on a parameter of a function in S. For these functions, it must hold that B doesn't pass the data directly or indirectly to a service at undesired locations. Thus, the requirement for service B is

$UnDes_{B,S,L} \triangleq Called_{S,B} = \emptyset \vee B.undesired(Called_{S,B}, L) = true$

Example 4: Consider service B in Fig. 1. The $Slice$ of the function $f \in Provided_B$ and $g \in Provided_B$ are defined by

$Slice_f \triangleq \{k\}$ and $Slice_g \triangleq \{l\}$

because there is no dataflow from x to y ($DEP_{x,y} = false$), which means $h \notin Slice_f$. With $Slice_f$ and $Slice_g$, it is

$Slice_{f,g} = \{k, l\}$.

So only the functions k and l are called with data stemming from A by service B. Hence,

$Called_{\{f,g\},C} \triangleq \{k, l\}$.

In the next step the slices $Slice_k$ and $Slice_l$ are computed:

$Slice_k = \{p\}$ and $Slice_l = \{q\}$.

Therefore, p and q are called with data from A over B:

$Slice_{k,l} = \{p, q\}$.

Considering $Slice_{k,l}$ and the provided functions of the service D and E, only service D is called with data from A:

$Called_{\{k,l\},D} \triangleq \{p, q\}$ and $Called_{\{k,l\},E} \triangleq \emptyset$.

Now, we can compute if there exists a data-flow to an undesired location. Thus,

$UnDes_{D,\{k,l\},SpyCountry} \triangleq false$

because of

$D.myloc \in SpyCountry$ and $Called_{\{k,l\},D} \notin \emptyset$.

Hence,

$UnDes_{E,\{k,l\},SpyCountry} \triangleq true$

because of

$Called_{\{k,l\},E} = \emptyset$.

So,

$UnDes_{C,\{f,g\},SpyCountry} \triangleq false$

because of

$Called_{\{f,g\},C} \notin \emptyset$ and

$C.undesired(Called_{\{f,g\},C}, SpyCountry) = false$.

We can conclude that there is a data-flow from service B over C to D. And D is a service with an undesired location. This violation is produced by service D. Note that although E is called and $E.myloc \in SpyCountry$, this is not omitted, as no data flows from A to E.

Thus, for a service A, the function $undesired$ can be implemented as shown in Fig. 2.

Theorem 1: Let X be a service with an undesired location $X.myloc \in L$. If there is a data-flow from a parameter x of a function $f \in S \subseteq Provided_A$ to X, then $undesired(S, L) = false$.

```
/*@return : false-> data-flow to undesired location(s)
            true-> data-flow only to desired locations*/
BOOL undesired(SET(Provided_A) S, SET(Locations) L) {
   if myloc ∈ L return false;
   foreach service X used by A do
      if ¬UnDes_{X,S,L} return false;
   return true;
}
```

Figure 2. Implementation of *undesired*

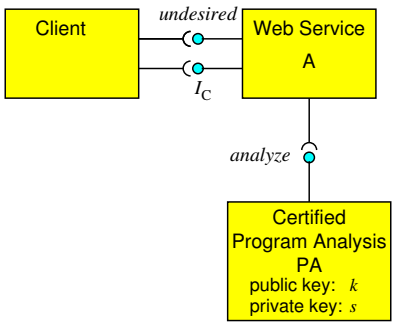

Figure 3. Trusted Agreement

Proof: Let X be a service with $X.myloc \in L$, i.e., the location of X is undesired. The maximal distance $d(Y, X)$ of a service Y to X is the length of the longest cycle-free path from Y to X w.r.t. the use relation. We prove the claim by induction on the maximal distance to X.
BASE CASE: $d(Y, X) = 0$. Then $Y = X$ and therefore $Y.myloc \in L$. For this case, *undesired* returns *false*.
INDUCTIVE CASE: $d(Y, X) > 0$. Let be $S \subseteq Provided_Y$ such that there is a data-flow from a parameter x of a function $f \in S$ to X. Thus, there must be a data-flow (internal to Y) to argument e of function call $z.g(\cdots e \cdots)$ where z is bound to Z and there is a data-flow from the corresponding parameter x of $g \in Provided_Z$ to X. Obviously $d(Z, X) < d(Y, X)$. Thus, by induction hypothesis it holds $undesired(S', L)$ returns *false* for each $S' \subseteq Provided_Z$ with $g \in S'$. By definition, it is $DEP_{e,x} = true$, thus, $g \in Slice_f$ and therefore $g \in Slice_S$. Since $g \in Provided_Z$, it holds $g \in Called_{S,Z}$. Thus, $Called_{S,Z} \neq \emptyset$ and therefore the induction hypothesis implies $UnDes_{Z,S,L} = false$. Since $undesired(S, X)$ only returns *true* if $Y.myloc \notin L$ and $UnDes_{Z,S,L} = true$ for all services used by A, it must return *false*. ■

IV. TRUSTED AGREEMENT

The approach of Sections II and III makes some idealistic assumptions: First, it assumes that each service is not malicious, i.e., it gives the correct answer according to Theorem 1. Second, there are no cycles in the use-relation. In this section, we present an approach to increase the trust in the answer given by a service that is also able to deal with cycles in the use-relation.

The main idea is similar to the verification of web pages, cf. Fig. 3: there is an independent certified program analysis service PA that performs the program analysis and computes the result of *undesired*. The following negotiation protocol increases the trust of the client to the analysis result:

Step 1: *Client* tells A that it would like to negotiate undesired locations
Step 2: A selects a certified program analysis PA and returns PA's public key k to *Client*.
Step 3: *Client* uses k to check whether PA is certified. If this is the case, *Client* encrypts its query $undesired(S, L)$ with k and passes it to A. If k does not belong to a certified program analysis, then *Client* may refuse to choose A or request another program analysis.
Step 4: A passes the encrypted query $undesired(S, L)$ together with its source text to PA. For security reasons, the source text is also encrypted with k.
Step 5: PA first decrypts the query and the source text of A. Then it performs the program analysis according to Section III. Finally, it signs the query $undesired(S, L)$ and the result with its private key s and passes it to A.
Step 6: A passes the signed result to *Client*.
Step 7: *Client* decrypts the signed result with the public key k of PA. Then *Client* verifies whether its query was being analyzed and whether the answer is *true*. If yes, then it accepts A, otherwise it refuses to choose A.

Since *Client* obtains the public key k of a program analysis, it can verify whether the program analysis can be trusted. Furthermore, the encryption of the query in Step 3 keeps it secret to A. Thus, A needs more effort to be malicious because the private key s of PA is required to decrypt the query, which is needed for the manipulation of the query. A possibility of A to be malicious would be that it creates its own (malicious) query q, encrypts it with k and passes it to PA. However, in Step 5 the analysis result together with the query is signed by PA's private key s. Since this key and the original query are secret to A, A is unable to replace the responded and manipulated query q of the PA by the original query. *Client* would discover such a manipulation at Step 7.

Remark: At first glance, it seems to be a severe restriction that A must pass its source text to PA. However, A can choose a program analysis PA that it trusts *before* offering PA to *Client*. □

Sections II and III demonstrate that PA might itself query services B used by A while performing the program analysis. In this case PA has the role of a client and B has the role of the service being queried. Hence, the above protocol can be used to negotiate with B. As PA is able to keep track of the analysis requests of A, it can check for cycles before processing the analysis request. In particular, it checks whether a query $undesired(S, L)$ for A is currently

```
BOOL undesired(SET(Provided_A) S, SET(Locations) L) {
    if myloc ∈ L return false;
    foreach service variable I_x x of A do
        while ¬x.hasInterface(undesired) ∨ ¬UnDes_{x,S,L} do {
            x = Reg.choose(I_x)
            if x = null return false;
        }
    return true;
}
```

Figure 4. Choosing an Adequate Service

being analyzed, i.e., whether there is an open analysis request $undesired(S', L)$ with $S' \subseteq S$. If yes, it can return immediately $true$. This is valid because if there is a data-flow from S' to an undesired location l, then there must be another call of a provided function to a service B with a data-flow to an undesired location.

Remark: We assume that the services are installed correctly and that we can use the advantages of trusted cloud federations [9], [10] in order to avoid that hackers change the implementation of the services. □

V. Dynamically Choosing a Service

In the scenarios of the previous sections, if a client requests a service A for avoiding data-flow to undesired locations and A itself may request for avoiding data-flow to undesired locations, then the chosen service B is not changed. However, service A might decide to choose an alternative service that fulfills the requirements for A. Thus, instead of returning $false$ if a possible data-flow to an undesired location is discovered, cf. Fig. 2, it can be searched for a service that guarantees that there is no data-flow to an undesired location, cf. Fig. 4. If there is no such service (i.e., $Reg.choose(I_x) = null$) then $false$ is returned. If $undesired(S, L)$ returns $true$ each service variable x of A is bound to a service X. Thus, for each call $x.f(\cdots) \in Slice_S$, there is no data-flow to an undesired location.

Remark: The search for an adequate service in the registry Reg might take a long time. An alternative would be to bound the number of tries to find an adequate service. □

The problem with this approach is that the service A needs to know the undesired locations. Thus, encrypting the analysis request with PA's public key prevents A from choosing alternative services according to Fig. 4. However, the program analysis PA could choose an alternative service on behalf of A. Thus PA could tell A which services it can choose. This dynamic choice can be achieved by changing the last step of Step 5 in Section IV: A positive answer is passed to A as in Step 5. However, if PA's answer is negative, it performs the procedure in Fig. 4. Any query $undesired(Called_{S,b}, L)$ to services b used by A is passed by A to the service bound to b. The result is passed back to PA which can tell A whether to bind the chosen service to b. If PA finishes the procedure in Fig. 4, it passes its final answer to A.

Thus, the answer is partially not being kept secret to the service being analyzed. However, the final answer is still kept secret and the client can still verify the final answer. The undesired locations and functions used by the client are still kept secret to the service being analyzed.

VI. Related Work

There is a lot of work on data security in the cloud. These works ensure data integrity ([11], [12], [13], [14], [15]), i.e., no malicious service or cloud attack changes the client's data or that this can be discovered by the client, respectively. These works assume (similar to our work) that there is an independent auditor. The works [12], [14] discuss privacy issues w.r.t. the auditor. [11] considers data-flow. They do not perform a static data-flow analysis but monitor data-flow between services in order to detect malicious services.

Works on privacy leaks on smart phones are closer to this work [16], [17]. These works analyze whether private data leave smart phone applications. While [16] uses a monitoring approach, [17] uses a static data-flow analysis approach. In contrast to our approach, they analyze the software executed on the smart phone, but they also forbid data leaving the smart phone that are stored in trusted locations.

Song et al. [18] investigates data-flow analysis in the context of service computing. In contrast to our work, they analyze data-flow correctness, e.g., whether each business process implementing a service receives the data it needs.

VII. Conclusions

In this work, we have shown how static data-flow analysis can ensure that clients data don't reach undesired locations (directly or indirectly) via software services. For this, an independent trusted program analysis is required. The approach turns out to be a negotiation approach similar to service-level agreements. The client sends a request to a service candidate whether it can guarantee to avoid data-flow to services on undesired locations. For this, an independent program analysis service signs the analysis result with its private key. Therefore, the client can verify whether a trusted program analysis has being performed. In order to prevent malicious services, the analysis request (in addition to the analysis result) is kept secret by encrypting it with the public key of the certified program analysis.

If we assume that no service is malicious (but possibliy erroneous), then there is no need for keeping the analysis request secret. If a service uses other services, it can look for alternatives that ensure themselves the avoidance of data-flow to undesired locations. In order to keep the analysis request secret to the services being analyzed, the service selection can be performed by the program analysis.

One might argue that a drawback of our approach is that the services must pass their source code to a certified program analysis. However, this certified program analysis is the only service that knows the source code and its the service that can choose the program analysis it trusts.

 ISBN: 978-1-61208-216-5

A more serious problem is that extremely malicious services send a source text to a certified program analysis that differs from their implementation. This could be prevented by two different approaches: first, there might be other program analyzes (e.g., that the service is doing something reasonable) and this analysis requests are also encrypted. If a service does not know what property is being analyzed, it is more difficult to prepare itself for cheating. A second possibility would be a combination of a monitoring approach (e.g., similar to [16]) with a randomized testing approach as used for checking data integrity (see, e.g., [15]): For the latter, the program analyzer can generate test cases (based on the source text it knows) and tests the service using these test cases. The monitoring approach monitors the data leaving the service and their corresponding destination services. Any difference between the data-flow analysis results and the destination services is a hint that the analyzed service is malicious, and the certified program analysis can give a negative answer to the client. It is subject to future work to detail these ideas. To check the performance of the proposed approach, the implementation of a tool is also a subject for future work.

ACKNOWLEDGMENT

The authors would like to thank the anonymous reviewers for their helpful comments.

REFERENCES

[1] European Commission and others, "Directive 95/46/ec of the european parliament and of the council of 24 october 1995 on the protection of individuals with regard to the processing of personal data and on the free movement of such data," *Official Journal of the European Communities*, vol. 23, p. 31, 1995.

[2] European Commission, `http://eur-lex.europa.eu/LexUriServ/`, 2012, last accessed May 2012.

[3] R. Seiger, S. Groß, and A. Schill, "Seccsie: A secure cloud storage integrator for enterprises," in *13th IEEE Conference on Commerce and Enterprise Computing (CEC)*. IEEE, 2011, pp. 252–255.

[4] L. Wei, H. Zhu, Z. Cao, W. Jia, and A. Vasilakos, "Seccloud: Bridging secure storage and computation in cloud," in *Distributed Computing Systems Workshops (ICDCSW), 2010 IEEE 30th International Conference on*. IEEE, 2010, pp. 52–61.

[5] R. Chow, P. Golle, M. Jakobsson, E. Shi, J. Staddon, R. Masuoka, and J. Molina, "Controlling data in the cloud: outsourcing computation without outsourcing control," in *Proceedings of the 2009 ACM workshop on Cloud computing security*. ACM, 2009, pp. 85–90.

[6] M. Dikaiakos, D. Katsaros, P. Mehra, G. Pallis, and A. Vakali, "Cloud computing: Distributed internet computing for it and scientific research," *IEEE Internet Computing*, vol. 13, no. 5, pp. 10–13, 2009.

[7] S. Pearson, "Taking account of privacy when designing cloud computing services," in *ICSE Workshop on Software Engineering Challenges of Cloud Computing, 2009*. IEEE, 2009, pp. 44–52.

[8] A. V. Aho, M. S. Lam, R. Sethi, and J. D. Ullman, *Compilers: Principles, Techniques, and Tools (2nd Edition)*. Boston, MA, USA: Addison-Wesley Longman Publishing Co., Inc., 2006.

[9] A. Celesti, F. Tusa, M. Villari, and A. Puliafito, "Federation establishment between clever clouds through a saml sso authentication profile," *International Journal on Advances in Internet Technology*, vol. 4, no. 12, pp. 14–27, 2011, ISSN: 1942-2652.

[10] A. Celesti, F. Tusa, M. Villari, and A. Puliafito, "Evaluating a distributed identity provider trusted network with delegated authentications for cloud federation," in *PROCEEDINGS of The Second International Conference on Cloud Computing, GRIDs, and Virtualization (Cloud Computing 2011)*. IARIA, 2011, pp. 79–85, ISBN: 978-1-61208-153-3.

[11] J. Du, W. Wei, X. Gu, and T. Yu, "Runtest: assuring integrity of dataflow processing in cloud computing infrastructures," in *Proceedings of the 5th ACM Symposium on Information, Computer and Communications Security*. ACM, 2010, pp. 293–304.

[12] C. Wang, Q. Wang, K. Ren, and W. Lou, "Privacy-preserving public auditing for data storage security in cloud computing," in *INFOCOM, 2010 Proceedings IEEE*. IEEE, 2010, pp. 1–9.

[13] M. Tribhuwan, V. Bhuyar, and S. Pirzade, "Ensuring data storage security in cloud computing through two-way handshake based on token management," in *2010 International Conference on Advances in Recent Technologies in Communication and Computing (ARTCom)*. IEEE, 2010, pp. 386–389.

[14] Z. Hao, S. Zhong, and N. Yu, "A privacy-preserving remote data integrity checking protocol with data dynamics and public verifiability," *IEEE Transactions on Knowledge and Data Engineering*, vol. 23, no. 9, pp. 1432–1437, 2011.

[15] Y. Liang, Z. Hao, N. Yu, and B. Liu, "Randtest: Towards more secure and reliable dataflow processing in cloud computing," in *2011 International Conference on Cloud and Service Computing (CSC)*. IEEE, 2011, pp. 180–184.

[16] W. Enck, P. Gilbert, B. Chun, L. Cox, J. Jung, P. McDaniel, and A. Sheth, "Taintdroid: An information-flow tracking system for realtime privacy monitoring on smartphones," in *Proceedings of the 9th USENIX conference on Operating systems design and implementation*. USENIX Association, 2010, pp. 1–6.

[17] M. Egele, C. Kruegel, E. Kirda, and G. Vigna, "Pios: Detecting privacy leaks in ios applications," in *Proceedings of the Network and Distributed System Security Symposium*, 2011.

[18] W. Song, X. Ma, S. Cheung, H. Hu, and J. Lü, "Preserving data flow correctness in process adaptation," in *Services Computing (SCC), 2010 IEEE International Conference on*. IEEE, 2010, pp. 9–16.

 ISBN: 978-1-61208-216-5