Dynamic Accumulator Based Discretionary Access Control for Outsourced Storage with Unlinkable Access
(Short Paper)

Daniel Slamanig

Department of Engineering and IT, Carinthia University of Applied Sciences,
Primoschgasse 10, 9020 Klagenfurt, Austria
d.slamanig@cuas.at

Abstract. In this paper we are interested in privacy preserving discretionary access control (DAC) for outsourced storage such as increasingly popular cloud storage services. Our main goal is to enable clients, who outsource data items, to delegate permissions (`read`, `write`, `delete`) to other clients such that clients are able to unlinkably and anonymously perform operations on outsourced data items when holding adequate permission. In contrast to recent approaches based on oblivious RAM, oblivious transfer combined with anonymous credentials or attribute based encryption, we propose a solution based on dynamic accumulators. In doing so, our approach naturally reflects the concept of access control lists (ACLs), which are a popular means to implement DAC.

1 Introduction

Ensuring confidentiality, integrity and authenticity when outsourcing organizational data(bases) to untrusted third parties has been a research topic for many years [7,10,12]. With the growing popularity of cloud computing, security in distributed access to data outsourced by "ordinary users" becomes also relevant. This is underpinned by the fact that so called cloud storage services increasingly gain in popularity. Besides confidentiality issues, i.e. for many types of data it may be valuable that the cloud provider (CP) solely has access to encrypted data but is still able to perform operations like searches on encrypted data [11], many recent works focus on more subtle privacy issues, i.e. unlinkable and potentially anonymous *access* to and *operations* on data stored in the cloud [3,4,9,13,14].

Some works [3,4,8] thereby focus on mandatory access control (MAC), i.e. access control policies for stored data are specified by the cloud provider, and others [9,14] on discretionary access control (DAC). In the latter scenario, clients can store data in the cloud and delegate access permissions to other clients - thereby specifying access control on their own - without the CP being able to determine who is sharing with whom, link operations (reads, writes) of clients together and to identify the users. Nevertheless, the CP can be sure that access control is enforced, i.e. clients need to have adequate permissions for the data.

A.D. Keromytis (Ed.): FC 2012, LNCS 7397, pp. 215–222, 2012.

Our Contribution. In the DAC setting, the access control in a system is enforced by a trusted reference monitor. A commonly used approach is to employ access control lists (ACLs), whereas every data item has its associated ACL representing a list of users with their corresponding permissions which can be modified dynamically. Thus, data owners can add or remove other users and their permissions to or from an ACL. A user who wants to perform an operation on a data item has to authenticate to the system and the reference monitor decides (using the corresponding ACL) whether he is allowed to perform the operation. It is straightforward to use pseudonyms in ACLs to hide the real identities of users in this setting. However, all operations of a user within the system can be linked to the user's pseudonym and achieving *unlinkability* is not that straightforward. We solve this problem and basically our approach is to stick with ACLs, but to "modify" ACLs in a way that the reference monitor 1) can still decide if a user is allowed to perform the operation, 2) users can delegate/revoke access rights to/from other users but 3) the reference monitor (CP) is *not able to identify users* as well as *link operations conducted by users together*.

We provide two solutions to this problem. The first solutions has the drawback that convincing the "reference monitor" of holding the respective permission has proof complexity $O(k)$, where k is the number of authorized users. The key idea is to have one ACL for every type of permission and data item and the ACL contains commitments to "pseudonyms". A user essentially proves to the "reference monitor" that he possesses *one* valid pseudonym in the ACL without revealing which one. The second approach reduces the proof complexity to $O(1)$ and uses a similar idea, whereas ACLs are represented by cryptographic accumulators [1]. A cryptographic accumulator allows to represent a set by a single value (the accumulator) whose size is independent of the size of the set. For every accumulated value of this set one can compute a witness and having this witness one can prove in zero-knowledge that one holds a witness corresponding to one accumulated value without revealing which one. Dynamic accumulators [6] in addition allow an efficient update of an accumulator by adding elements to (and possibly deleting elements from) it along with efficient update of the remaining witnesses. In particular, our second construction relies on a dynamic accumulator with efficient updates proposed by Camenisch et al. in [5], whereas efficient updates mean that witnesses can be updated without the knowledge of accumulator-related secret information by any party.

2 Related Work and Background

In this section we briefly present three different approaches bearing some similarities with the one proposed in this paper, but employing entirely different building blocks. Then, we present the concept of dynamic accumulators.

Anonymous Credentials. Camenisch et al. [3,4] use anonymous credentials within oblivious transfer protocols to access items from a database at a server. Thereby, the server defines the access control policies but neither learns which items a user accesses nor which attributes or roles the user has. Still, he is able to

enforce access control. An approach supporting complex access control policies such as the Brewer-Nash or the Bell-LaPadula model based on oblivious transfer and so called stateful anonymous credentials is proposed in [8].

Oblivious RAM. In [9], Franz et al. present an oblivious RAM (ORAM) based approach, which enables an owner of a database to outsource the database to an untrusted storage service. Thereby, the data owner can delegate read and write permissions to clients and clients can only perform operations on data items when they possess appropriate permissions. A key feature of their so called delegated ORAM solution is that the storage service does not learn how often data items are accessed by a user while access control is still enforced. Additionally, their approach employs symmetric encryption to provide data confidentiality. However, revocation of access rights is not explicitly realized. They propose to encrypt data items with a fresh key and to use broadcast encryption to distribute the key amongst all remaining authorized clients, which is rather involved.

Attribute Based Encryption. Very recently Zarandioon et al. [14] introduced K2C, an approach for hierarchical (file system like) cryptographic cloud storage, which can be implemented (like the approach presented here) on top of existing cloud services like Amazon S3[1]. Here, clients can organize their encrypted data items hierarchically at an untrusted storage provider and delegate `read` and `write` permissions to other clients. The approach is based on key-policy attribute based encryption (KP-ABE) and signatures from KP-ABE to provide anonymous access. Although quite elegant, the revocation of permissions in this approach, as above, requires re-encryption of data items w.r.t. updated policies and distribution of respective keys to all remaining authorized clients.

Re-encryption (even when using lazy revocation on write accesses) is a cumbersome task and we avoid this by guaranteeing that revoked clients will no longer be able to even read data items, since they will not be able to successfully pass the prove protocol with the "reference monitor" at the CP any longer.

Dynamic Accumulator with Efficient Updates. In our construction we make use of a dynamic accumulator with efficient updates introduced in [5]. Accumulatable values are in the set $\{1, \ldots, n\}$, by V we denote the set of values contained in the accumulator, V_w represents status information about the accumulator, $state_U$ are state information containing some parameters for the accumulator and the set U represents all elements that were ever accumulated. We provide an abstract definition below (see [5] for technical details):

$\mathsf{AccGen}(1^k, n)$ generates an accumulator key pair (sk, pk), an initially empty accumulator $acc_\emptyset$, which is capable of accumulating up to n values, and an initial state $state_\emptyset$.

$\mathsf{AccAdd}(sk, i, acc_V, state_U)$ adds value i to the accumulator acc_V and outputs a new accumulator $acc_{V\cup\{i\}}$, a state $state_{U\cup\{i\}}$ and a witness wit_i for value i.

$\mathsf{AccUpdate}(pk, V, state_U)$ outputs an accumulator acc_V for values $V \subset U$.

[1] `http://aws.amazon.com/s3/`

$\mathsf{AccWitUpdate}(pk, wit_i, V_w, acc_V, V, state_U)$ outputs a witness wit'_i for acc_V if wit_i was a witness for acc_{V_w} and $i \in V$.

$\mathsf{AccVerify}(pk, i, wit_i, acc_V)$ verifies whether $i \in V$ using an actual witness wit_i and accumulator acc_V. If this holds it outputs `accept` otherwise `reject`.

Note that the AccVerify algorithm among other parameters gets (i, wit_i)[2] and thus knows who "proves" that his corresponding value i was indeed accumulated. Fortunately, dynamic accumulators are usually designed having in mind that they should come along with efficient proofs to prove in zero-knowledge that a value was accumulated without revealing the value itself.

ZKP of Accumulated Value. Camenisch et al. [5] provide an elegant and efficient ZKP for accumulated values. Therefore, instead of signing the values i using an arbitrary signature scheme, one uses a variant of the weakly secure Boneh-Boyen signature scheme [2] (therefore the AccAdd algorithm has to be modified accordingly [5]). This in combination with a randomization technique allows a user to provide a proof of knowledge (PK) of a randomization value that allows to de-randomize a commitment to value i such that i was signed and i is accumulated in acc_V. The PK can be made non-interactive using the Fiat-Shamir heuristic, whereas the corresponding signature of knowledge will be denoted as spk. Thus, we can modify the AccVerify algorithm to take input parameters (pk, spk, acc_V) and this allows for verification without the necessity of revealing the value i and the witness wit_i.

3 Implementing DAC with Unlinkable Access

In this section we present the model, a first (rather inefficient) construction and a detailed description of our main construction. Then, we comment on some aspects and briefly argue about the security.

Model. Let CP be a cloud provider who runs a cloud storage service, which allows clients $C = \{c_1, \ldots, c_n\}$ to store (outsource), retrieve and manipulate data items. Clients access data via a very simple interface as it is quite common in block oriented cloud storage services such as Amazon S3, i.e. storing key-value pairs and supporting the operations `insert`, `read`, `write` and `delete`. Now, the owner of a data item (the client who inserts the data into the cloud storage) should be able to delegate the permissions `read`, `write` and `delete` (`r`,`w` and `d` for short) for single data items to other clients and can also revoke all these permissions whenever necessary.

One main design goal is, that the CP "enforces" the access control, i.e. only allows an operation if the client is able to prove the possession of the respective permission, but at the same time is not able to link different operations of the clients together. Additionally, clients may also stay anonymous as we will discuss later on and will be clear from our construction. We assume that a client can establish a secure communication channel to an owner of data items and vice

[2] When instantiating the accumulator scheme of [5] the values i are actually group elements g_i and can either be made public or the values $g_i||i$ are signed.

versa (for instance by sharing encrypted messages via Amazon's Simple Queue Service). Furthermore, we assume the CP to represent an honest but curious (passive) adversary and that the CP does not collude with clients.

A First Approach. To provide a better understanding, we begin with a first approach: Consider a data owner c_m, who wants to insert a data item d_i at CP. He generates a key pair (sk_{d_i}, pk_{d_i}) of a signature scheme and chooses suitable random values $s_{m,i,r}, r_{m,i,r}$, $s_{m,i,w}, r_{m,i,w}$ and $s_{m,i,d}, r_{m,i,d}$ for an unconditionally hiding commitment scheme, i.e. Pedersen commitments. He computes the commitments $c_{m,i,r} = C(s_{m,i,r}, r_{m,i,r})$ as well as $c_{m,i,w}$ and $c_{m,i,d}$ and signs every single commitment using sk_{d_i}. Then he sends d_i, the verification key pk_{d_i} along with the commitments and respective signatures to CP. CP checks whether the single signatures are valid and creates three empty ACLs, $ACL_{d_i,r}$, $ACL_{d_i,w}$ and $ACL_{d_i,d}$ for `r`, `w` and `d` permissions respectively and adds the commitments to the corresponding ACLs.

If c_m wants to delegate a permission to another client c_j for data item d_i, he simply chooses new random values $s_{j,i,x}, r_{j,i,x}$ for permission $x \in \{\mathtt{r}, \mathtt{w}, \mathtt{d}\}$, computes and signs the commitments and requests CP to add the commitments to the respective ACLs (who accepts this if the signatures are valid). Then, he gives $(s_{j,i,x}, r_{j,i,x})$ as well as the parameters for the commitment scheme to c_j.

Assume a user wants to perform a `r` operation for data item d_i, then he has to retrieve the respective ACL $ACL_{d_i,r}$ (which we assume has k entries) and perform an OR-composition of a ZKP of the opening of a commitment, i.e. $PK\{(\alpha, \beta) : \bigvee_{l=1}^{k}(c_{l,i,x} = C(\alpha, \beta))\}$. This proof is an efficient OR-composition of DL-representation proofs in case of Pedersen commitments, can easily be made non-interactive and succeeds if c_j knows at least one opening for a commitment in $ACL_{d_i,r}$. If the verification of this proof succeeds, CP can allow the `r` operation for data item d_i, but is not able to identify c_j. Nor is CP able to link different operations of c_j together due to employing zero-knowledge OR-proofs. Obviously, if there is only a single commitment in the ACL, then there will be no unlinkability. However, it is straightforward for the data owner to initially insert some dummy commitments into the ACLs, which will provide unlinkability - the CP cannot distinguish between such dummies and real users.

In order to revoke permission x for d_j for client c_j, the data owner simply provides the opening information of the commitment $(s_{j,i,x}, r_{j,i,x})$ along with the signature for the respective commitment to the CP. Then, the CP computes the commitment, checks the signature and if the verification holds removes the commitment from $ACL_{d_i,x}$.

Our Main Construction. The above presented approach is very simple, but has some drawbacks. Let k be the number of clients in an ACL, then 1) the representation of every ACL has size $O(k)$, 2) clients have to retrieve k commitments prior to every operation and most importantly 3) the proof complexity of client's OR-proofs is $O(k)$. In contrast, within the approach presented below all these complexities are $O(1)$ and thus independent of the number of clients. Before going into details, we provide an abstract description of the operations. The additional input $params_{\mathsf{Acc}}$ will be discussed subsequently.

$\mathsf{Store}(id_i, d_i)$: The owner of data item d_i identified by id_i stores (id_i, d_i) at CP.

$\mathsf{Delegate}(c_j, id_i, per, params_{\mathsf{Acc}})$: Delegate permission $per \in \{\mathtt{r}, \mathtt{w}, \mathtt{d}\}$ for data item identified by id_i to client c_j.

$\mathsf{Revoke}(c_j, id_i, per, params_{\mathsf{Acc}})$: Revoke permission $per \in \{\mathtt{r}, \mathtt{w}, \mathtt{d}\}$ for data item identified by id_i for client c_j.

$\mathsf{Read}(id_i, params_{\mathsf{Acc}})$: Read data item d_i identified by id_i. If the client holds the corresponding permission, d_i will be delivered, otherwise return $\perp$.

$\mathsf{Write}(id_i, d_i', params_{\mathsf{Acc}})$: Modify data item d_i identified by id_i to d_i'. If the client has the corresponding permission, d_i' will be written, otherwise return $\perp$.

$\mathsf{Delete}(id_i, params_{\mathsf{Acc}})$: Delete data item d_i identified by id_i. If the client has the corresponding permission, d_i will be deleted, otherwise return $\perp$.

Below, we provide a more detailed description of the operations involved in our construction and the meaning of the parameters $params_{\mathsf{Acc}}$:

Store. A data owner who wants to insert (id_i, d_i) at the CP needs to specify the maximum numbers of clients for every permission. Let us assume that he sets this number for $\mathtt{r}$, $\mathtt{w}$ and $\mathtt{d}$ to n. Then he runs $\mathsf{AccGen}(1^k, n)$ three times and obtains $(sk_{d_i,x}, pk_{d_i,x}, acc_{\emptyset,d_i,x}, state_{\emptyset,d_i,x})$ for $x \in \{\mathtt{r}, \mathtt{w}, \mathtt{d}\}$ and adds himself (represented by value 1, the first accumulatable value) to all accumulators by running $\mathsf{AccAdd}(sk_{d_i,x}, 1, acc_{\emptyset,d_i,x}, state_{\emptyset,d_i,x})$ and sends (id_i, d_i) along with $(pk_{d_i,x}, acc_{\{1\},d_i,x}, state_{\{1\},d_i,x})$ and bookkeeping information V_{d_i}, V_{w,d_i} to the CP. He stores $sk_{d_i,x}$, the witnesses $wit_{1,d_i,x}$ and V_{d_i}, V_{w,d_i}.

Delegate. A data owner who wants to delegate permission $x \in \{\mathtt{r}, \mathtt{w}, \mathtt{d}\}$ for data item d_i to client c_j proceeds as follows: He parses $params_{\mathsf{Acc}}$ (which can be retrieved from CP) as $(pk_{d_i,x}, acc_{V,d_i,x}, state_{U,d_i,x})$. Using $state_{U,d_i,x}$ he determines a value z not already accumulated and obtains the updated accumulator $acc_{V\cup\{z\},d_i,x}$, updated state information $state_{U\cup\{z\},d_i,x}$ and a witness $wit_{z,d_i,x}$ by running $\mathsf{AccAdd}(sk_{d_i,x}, z, acc_{V,d_i,x}, state_{U,d_i,x})$. The data owner securely communicates $(z, wit_{z,d_i,x})$ to client c_j and stores the signature part of $wit_{z,d_i,x}$ for revocation purposes. Then, he sends $(acc_{V,d_i,x}, state_{U,d_i,x})$ along with updated bookkeeping information to the CP.

Revoke. A data owner who wants to revoke permission $x \in \{\mathtt{r}, \mathtt{w}, \mathtt{d}\}$ for data item d_i and client c_j proceeds as follows: The data owner parses $params_{\mathsf{Acc}}$ as z, where z represents the value accumulated for c_j in $acc_{V,d_i,x}$. Then he sends z along with the signature for the corresponding witness to CP, who checks the signature. If the verification holds, CP runs $\mathsf{AccUpdate}(pk_{d_i,x}, V \setminus \{z\}, state_{U,d_i,x})$ and stores the resulting accumulator $acc_{V\setminus\{z\},d_i,x}$, otherwise CP terminates.

Read/Write/Delete. A client who wants to perform operation $x \in \{\mathtt{r}, \mathtt{w}, \mathtt{d}\}$ for data item d_i first parses $params_{\mathsf{Acc}}$ as $(pk_{d_i,x}, wit_{d_i,x}, V_w, acc_{V,d_i,x}, V, state_{U,d_i,x})$. Then he has to check whether the accumulator $acc_{V,d_i,x}$ has changed, i.e. a new client was added or a client was revoked. If this is the case, the user has to run $\mathsf{AccWitUpdate}(pk_{d_i,x}, wit_{d_i,x}, V_w, acc_{V,d_i,x}, V, state_{U,d_i,x})$ to compute the updated witness $wit'_{d_i,x}$. Then, he uses the actual witness to compute a signature of knowledge spk to prove that the value corresponding to the witness was accumulated. He then sends spk to CP and the CP runs $\mathsf{AccVerify}(pk_{d_i,x}, spk, acc_{V,d_i,x})$.

If it returns `accept`, then CP depending on the operation either delivers d_i to the client (`read`), overwrites d_i with d_i' provided by the client (`write`) or deletes d_i along with corresponding accumulators and bookkeeping information (`delete`) and terminates otherwise.

Remark. Delegate and Revoke operations need to be authorized, since otherwise "ACLs" could be maliciously manipulated. We have omitted this above, but this can be efficiently realized by signing the values sent to the CP at the end of the two above mentioned operations. For the sake of convenience and efficiency, the data owner can use the Boneh-Boyen signature scheme whose respective keys are part of the private and public key of the accumulator respectively.

Confidentiality and Integrity of Stored Data. When storing encrypted data, all that data owners have to do is to additionally send the respective encryption key along with the witness to the user. Note that we do not require re-encryption (as in [9,14]) since revoked users will no longer be able to access data items. To provide integrity verification, one can store signatures or HMACs along with data items and distribute the keys together with the witnesses to clients.

Security Analysis. First, we consider *security against malicious clients*: All clients other than the data owner do not know $sk_{d_i,x}$. Thus they will not succeed in producing new witnesses, i.e. authorizing unauthorized clients, or trigger unauthorized Delegate or Revoke operations to manipulate the accumulator. Consequently, clients can only perform operations on data objects with permissions they have been granted. Secondly, we consider *security against a curious CP*: The CP does not learn the identities of clients (when they obtain a permission, they are only identified by the value to be accumulated - which is unrelated to their identity). Furthermore, due to employing ZKPs in the AccVerify protocol to prove the possession of witnesses, the respective witnesses and corresponding values are not disclosed. Consequently, clients conduct operations in an unlinkable and anonymous fashion.

4 Extensions and Future Work

Hierarchical Delegation. It may be desirable to augment simple DAC in a way that clients who have obtained permissions for some data from a data owner are able to delegate the obtained permissions for this data to further clients on their own. But then, data owners very likely would like to recursively revoke granted permission. For instance, if the data owner has provided permission x to client c_i and c_i has granted permission x to c_j, then revoking permission x for c_i should immediately imply revoking permission x for c_j. This can be realized as follows: If the data owner wants to allow further delegation for a specific data item, permission x and client c_i, he simply gives m witnesses to this client and remembers the corresponding values and signatures. Client c_i can then delegate $m-1$ permissions x to other clients (or give other clients more witnesses for further delegation). If the data owner revokes the permission x for client c_i, then he simply "removes" all m witnesses from the respective accumulator.

Discretionary access control is an admittedly simple but often sufficient access control model. Especially when outsourcing data to popular cloud storage

services, such an access control model is reasonable and can be deployed quite easily. Due to increasing privacy demands, a mechanism - as the one proposed in this paper - can be valuable. We leave the gathering of practical experiences when deploying our construction in this scenario as important future work.

Acknowledgements. We thank the anonymous reviewers for their helpful feedback on the paper. This work has been supported by an internal grant (zentrale Forschungsförderung - ZFF) of the Carinthia University of Applied Sciences.

References

1. Benaloh, J.C., de Mare, M.: One-Way Accumulators: A Decentralized Alternative to Digital Signatures (Extended Abstract). In: Helleseth, T. (ed.) EUROCRYPT 1993. LNCS, vol. 765, pp. 274–285. Springer, Heidelberg (1994)
2. Boneh, D., Boyen, X.: Short Signatures Without Random Oracles. In: Cachin, C., Camenisch, J. (eds.) EUROCRYPT 2004. LNCS, vol. 3027, pp. 56–73. Springer, Heidelberg (2004)
3. Camenisch, J., Dubovitskaya, M., Neven, G.: Oblivious Transfer with Access Control. In: ACM Conference on Computer and Communications Security, pp. 131–140. ACM (2009)
4. Camenisch, J., Dubovitskaya, M., Neven, G., Zaverucha, G.M.: Oblivious Transfer with Hidden Access Control Policies. In: Catalano, D., Fazio, N., Gennaro, R., Nicolosi, A. (eds.) PKC 2011. LNCS, vol. 6571, pp. 192–209. Springer, Heidelberg (2011)
5. Camenisch, J., Kohlweiss, M., Soriente, C.: An Accumulator Based on Bilinear Maps and Efficient Revocation for Anonymous Credentials. In: Jarecki, S., Tsudik, G. (eds.) PKC 2009. LNCS, vol. 5443, pp. 481–500. Springer, Heidelberg (2009)
6. Camenisch, J., Lysyanskaya, A.: Dynamic Accumulators and Application to Efficient Revocation of Anonymous Credentials. In: Yung, M. (ed.) CRYPTO 2002. LNCS, vol. 2442, pp. 61–76. Springer, Heidelberg (2002)
7. Ciriani, V., De Capitani di Vimercati, S., Foresti, S., Jajodia, S., Paraboschi, S., Samarati, P.: Fragmentation and Encryption to Enforce Privacy in Data Storage. In: Biskup, J., López, J. (eds.) ESORICS 2007. LNCS, vol. 4734, pp. 171–186. Springer, Heidelberg (2007)
8. Coull, S.E., Green, M., Hohenberger, S.: Access Controls for Oblivious and Anonymous Systems. ACM Trans. Inf. Syst. Secur. 14(1), 10 (2011)
9. Franz, M., Williams, P., Carbunar, B., Katzenbeisser, S., Peter, A., Sion, R., Sotakova, M.: Oblivious Outsourced Storage with Delegation. In: Danezis, G. (ed.) FC 2011. LNCS, vol. 7035, pp. 127–140. Springer, Heidelberg (2012)
10. Hacigümüs, H., Mehrotra, S., Iyer, B.R.: Providing Database as a Service. In: ICDE. IEEE (2002)
11. Kamara, S., Lauter, K.: Cryptographic Cloud Storage. In: Sion, R., Curtmola, R., Dietrich, S., Kiayias, A., Miret, J.M., Sako, K., Sebé, F. (eds.) FC 2010 Workshops. LNCS, vol. 6054, pp. 136–149. Springer, Heidelberg (2010)
12. Mykletun, E., Narasimha, M., Tsudik, G.: Authentication and Integrity in Outsourced Databases. In: NDSS. The Internet Society (2004)
13. Williams, P., Sion, R., Carbunar, B.: Building Castles out of Mud: Practical Access Pattern Privacy and Correctness on Untrusted Storage. In: ACM Conference on Computer and Communications Security, pp. 139–148. ACM (2008)
14. Zarandioon, S., Yao, D(D.), Ganapathy, V.: K2C: Cryptographic Cloud Storage with Lazy Revocation and Anonymous Access. In: Rajarajan, M., et al. (eds.) SecureComm 2011. LNICST, vol. 96, pp. 59–76. Springer, Heidelberg (2012)