EURECOM
Department of Networking and Security
Campus SophiaTech
450, route des Chappes
06410 Biot France
FRANCE

Research Report RR-13-281

A P2P Based Usage Control Enforcement Scheme Resilient to Re-injection Attacks

March 2013

Iraklis Leontiadis, Refik Molva, and Melek Önen

Tel : (+33) 4 93 00 81 00
Fax : (+33) 4 93 00 82 00
Email : {Firstname.Name1,Firstname.Name2,Firstname.Name3}@eurecom.fr

[1] EURECOM's research is partially supported by its industrial members: BMW Group, Cisco, Monaco Telecom, Orange, SAP, SFR, Sharp, STEricsson, Swisscom, Symantec, Thales.

A P2P Based Usage Control Enforcement Scheme Resilient to Re-injection Attacks

***Abstract*—Existing privacy controls based on access control techniques do not prevent massive dissemination of private data by malevolent acquaintances of social network, unauthorized duplication of files or personal messages, or persistence of some files in third-party operated storage beyond their deletion by their owners. We suggest a usage control enforcement scheme that allows users to gain control over their data during its entire lifetime and the way it is disseminated in outsourced distributed data storage. The scheme is based on a peer-to-peer architecture whereby a different set of peers is randomly selected for data assignment. Usage control is achieved based on the assumption that at least t out of any set of n peers will not behave maliciously. Such a system would still suffer from re-injection attacks whereby attackers can gain ownership of data and the usage policy thereof by simply re-storing data after slight modification of the content. In order to cope with re-injection attacks the scheme relies on a similarity detection mechanism based on special hash functions. The robustness of the scheme has been evaluated in an experimental setting using a variety of re-injection attacks.**

I. Introduction

With the advent of social networks and cloud computing the processing and storage of private data is more and more outsourced to services operated by third parties. The significant capacity increase and widespread dissemination advantages offered by these services also come with unprecedented security and privacy concerns. Beyond basic exposures that are partially covered by classical security mechanisms such as data confidentiality, authentication, and access control new security and privacy requirements arise due to the sheer volume of data exchanges and the span of dissemination enabled by these services. Existing privacy controls based on access control techniques do not prevent massive dissemination of private data by malevolent acquaintances of social network, unauthorized duplication of files or personal messages, or persistence of some files in third-party operated storage beyond their deletion by their owners. As a result of such exposures, users of these outsourced services lose control over their data thereof. Bestowing users back with the control of their data and over the way it is disseminated within these services can unfortunately not be achieved by means of classical access control mechanisms.

Access control can achieve perfect control over the identity of parties authorized to access the data and the circumstances of the access operation pertaining to time and content but it does not allow for any control over the way these parties make further use of the data. Such a comprehensive control spanning the entire lifetime of each data segment can actually, be assured through a security service called usage control. A contingent, high-level use case example that entails usage control allows a set of users to store their data segments along with the imposed by them policy into a system, such that the usage control policy will be enforced during the entire lifetime of users' data. Usage control guarantees the policy enforcement to every copy of the data segment and the conformation to the policy after the deletion of it without letting unauthorized duplications in third-party storage services.

In this paper, we suggest an original solution to tackle a special case of the usage control problem. Even though a generic usage control solution fitting all possible settings seems infeasible, in a confined environment with a well defined set of subjects, resources and operations, usage control can be achieved. The impact of leaving the system to violate some of the rules would be negligible. The proposed solution defines a P2P system where data management operations are performed and controlled by a subset of peers. The enforcement is assured thanks to the collaboration of peers and based on the assumption that at least t out of any set of n randomly chosen peers will not behave maliciously. In all users who have adopted the P2P network architecture for any operation, the usage control policy enforcement is guaranteed and violations outside this network would not significantly affect the system.

Furthermore, even in such a confined environment, an adversary may try to gain control over data segments by slightly modifying the content and re-submitting the resulting data segments with a different policy. The proposed enforcement mechanism allows peers to detect similarities between any upcoming data and the existing one, thanks to the use of special functions defined as *error tolerant hash function* (ETHF).

In section 2, we define the problem of usage control and depict the idea of our solution. Related work is presented in section 3. In section 4, the preliminaries of our solution are provided and in section 5 we give a detailed description of the scheme. Before closing with our conclusion and future work in section 8, we analyze the security of the proposed mechanism in section 6 and in section 7 we evaluate the correctness of the *error tolerant hash function* in an experimental setup.

II. Problem Statement

A. Usage control

If we could try to give a definition for usage control then this can be summarized as follows: *Enforce compliance with policy during the entire lifetime of each resource.* Usage control's main difference with access control is the notion of continuous policy validation whereas access control is discrete in the sense that there is no policy enforcement between various checkpoints. In contrast, usage control enforces the policy during the time elapsed between checkpoints.

For instance an access control system verifies that a user has the rights required by the policy before authorizing access to a file, but it does not monitor the operations performed by that subject on the data driven from the file during that access operation–whereas a usage control policy enforcement system would also assure that the data obtained through the access operation is used properly, i.e. in accordance with the usage control policy. Thus, an *online social network* (OSN) application that verifies access to personal data as part of user profile, assures access control but not usage control because usage control violations such as duplication or dissemination of personal data by parties authorized for access control, such as friends, cannot be prevented even when required by the owner.

As already introduced in the previous section, the proposed solution is applied to a confined environment whereby all data within the system is protected following usage control policies defined by their respective owner. In a scenario with such a confined environment, let S be a system that implements usage control on a set of data D based on the policy of data owners. Usage control in such a scenario inherently suffers from two limitations that would allow malicious users to evade the usage control on data D by system S.

In the first type of attack, which we define as **bypass** attacks, a legitimate user can escape from usage control enforcement on a piece of data d_i by simply pulling out d_i from S and using it outside S in an unauthorized way. Even though impossible to prevent, the bypass attack has a limited impact if S has a global coverage that makes it inescapable for the overwhelming majority of users. Some OSNs such as Facebook or LinkedIn are inescapable with respect to the inter-personal communication and if these OSNs implement a usage control system like S then the bypass attack on personal data would only have a very limited impact. Therefore, in the sequel of the paper, we assume that given the confined environment **bypass** attacks will not have a strong impact on the security of the system.

Even a system that would benefit from the impact factor to prevent the bypass attacks, would still suffer from the other inherent exposure of usage control system that is the **re-injection** attack. In such an attack an adversary extracts some data d_i that are governed by a usage control policy P, imposed by its owner u_i. Afterwards the malicious user $slightly$ alters the data and tries to re-store data d'_i but now with the same or different usage control policy P'. As such she will present herself as the new owner o'_i of data d'_i abusing the usage control policy system and affecting the dissemination of legitimate users' data by duplicating it.

B. Idea of solution

In order to assure usage control together with preventing **re-injection** attacks, we propose a distributed enforcement mechanism based on a P2P system whereby peers collaborate with each other to assure the enforcement of usage control policies defined by data owners: in the proposed solution, each data is assigned to and managed by a predefined set of n peers whereby at least t of them are considered as being legitimate. The new system hence relies on a threshold solution whereby at least t legitimate nodes collaborate and guarantee the correct enforcement of policies defined for each piece of data.

Yet, such a solution does not protect the system from **re-injection** attacks. Since the decentralized control of each data segment is distributed among a different subset of nodes, any attacker may gain ownership of a data segment by slightly altering some existing data segment and submitting the new version of a new segment with its own policy to the system. Such modification attempts will go undetected because the modified data segment will be considered as a brand new segment and thus it will be assigned to a different set of peer nodes.

Such attacks are avoided thanks to the design of a dedicated data assignment algorithm which detects similarities between any new and already stored data. This newly defined type of algorithm is a specific function which outputs the same fingerprints for slightly modified files. This function is named error tolerant hash function (*ETHF*) an assures that similar data are assigned to the same set of nodes.

Furthermore, the node assignment operation of course cannot be implemented by the user itself: hence, randomly selected peers should agree on this final set of peers assigned to the management of a specific content. Therefore in addition to the need for similarity detection function, the system should define a random generator to select these random peers whose main role is to perform the initial data assignment step. Basic cryptographic hash functions are a good candidate for this preliminary step.

Furthermore, even before the problem of node assignment, one should define the way how content is defined in the system. Indeed, the relevant and unique content has to be extracted from files that may be defined or encoded in different ways. We therefore assume that each file consists of some metadata that includes information about the file and the content itself. This content is used as the input to detect similarity. We assume that the content of the files is human readable encoded text.

To summarize, the proposed usage control mechanism that defines the P2P network as the confined environment protects against re-injection attacks thanks to the use of error-tolerant hash functions that are able to detect similarities. However, the use of such functions is not sufficient in order to fully ensure the control over data. We next give a description of previous related work.

III. Related work

In [12] authors provide the first definition of usage control policy in the sense of ongoing policy enforcement after data release. A set of authorizations, obligations and conditions should be smoothly orchestrated for a usage control policy scheme. Conditions should be validated in accordance with obligations in order to allow authorizations on objects. Zhang *et al* [9] provide a different formalization of usage control using *Petri nets*. In [17],[6] authors propose a policy based usage control language for usage control enforcement. Zhao *et al* [17] in their analysis proposed the notion of timing constraints which advocates an ongoing usage control policy. Both papers lack the definition of a mechanism whereby the enforcement of a usage control policy can be applied in an architecture with malicious users. Janicke *et al* [8] proposed an enforcement scheme which can be considered as being the closest one to the proposed solution in the sense that enforcement is achieved in a distributed environment. Unfortunately, as opposed to our solution the correctness and security of such mechanisms are not evaluated though out a real life data management scenario.

In [13] authors propose usage control enforcement targeted for the X11 graphical user interface management daemon in Linux, Unix and Mac operating systems. Their solution is based on data flow tracking in between different resources. In [10], Kumari *et. al.* enforce usage control policies in the application level of a web browser by evaluating it in a web based online social network plugin. Harvan *et al* [4] implement a data flow control mechanism with system calls interposition by controlling them with a monitoring mechanism. Even though the aforementioned practical usage control enforcement mechanisms are implemented in different levels of a system our solution identifies and copes with specific attacks which have a serious impact on the security of the usage control mechanism such as **re-injection** and **bypass** attacks.

Some other works such as in [2] propose the use of similarity detection algorithms for an optimized data storage and lookup operation in P2P systems. Our work significantly differs from [2] since *EHF* are used for usage control and proved to be secure against re-injection attacks.

IV. Preliminaries

In order to introduce the proposed scheme, we first describe the tools which will further be used as the main building blocks of the proposed usage control mechanism.

We consider the scenario whereby a user U_i wishes to store a file F_i to further share it with some other users. In order to enforce the control over the file F_i, the owner U_i defines a set of policy rules P_i.

A. Peer to peer network.

As previously mentioned, the proposed solution implements usage control within a P2P network which is considered as a confined environment with a global coverage: we assume that the impact of bypass attacks hence is limited.

In this P2P system, data lookup, data retrieval and all other operations related to data management follow a protocol based on Distributed Hash Tables (DHT) [16]. A DHT associates the stored data with a key. Each key is assigned to a subset of nodes who corresponds to the peers that are responsible of storing the corresponding data and enforcing the correct usage of it. The mapping between the key and the subset of nodes in a specific protocol is based on the use of a specific hash function which is described in the next section.

The correctness and security of the proposed usage control scheme relies on the legitimate behavior of a corresponding peers responsible of controlling data.

$Lookup$ and $retrieval$ operations for a certain data object are distributed among n peers whereby at least t of them do not behave maliciously.

B. Error tolerant hash function.

In the proposed solution, the hash functions that define the mapping between certain data and the subset of nodes which will store it is an error tolerant hash function which will allow peers to detect similarities between data pieces.

As opposed to cryptographic hash functions which given a slightly modified input return a totally different digest than the original one, an error tolerant hash function ($ETHF$) is resilient to some changes on the input and is defined as follows:

Definition 1: $\mathcal{H}_\mathrm{s}$ is an $ETHF$ if and only if satisfies the following properties:

1) ***resiliency to changes****: given two files x_1, x_2 that are similar, that is, only a small percentage δ of their content is different, it exists σ, such that the hamming distance is less than σ, i.e: $HD(\mathcal{H}_\mathrm{s}(\mathrm{x}_1), \mathcal{H}_\mathrm{s}(\mathrm{x}_2)) \leq \sigma$, where HD corresponds to the hamming distance.*
2) ***first pre-image resistance****: given the result of $\mathcal{H}_\mathrm{s}(\mathrm{x})$ it is hard for an adversary to reconstruct x.*
3) ***collision resistance****: it is hard for an attacker two find two different files x_1, x_2 such that $\mathcal{H}_\mathrm{s}(\mathrm{x}_1)$=$\mathcal{H}_\mathrm{s}(\mathrm{x}_2)$*

Thanks to the aforementioned properties an $ETHF$ assures the correctness and security of the usage control policy enforcement scheme. By assuring that a file and a slightly modified version thereof will be assigned to the same set of peer nodes, resiliency to changes helps detect re-injection attacks. First pre-image resistance prevents an attacker from determining data files that would be assigned to colluding peers. Collision resistance on the other hand prevents false alarms while detecting re-injection attacks.

Similarity detection was the focus of several research activities [14], [11]. One of the most performant solution [5] which nowadays is widely used is Charikar's Simhash algorithm [1]. This algorithm is used to check similarities between web documents. The approach consists of creating a sequence of tokens in such a way that each web page is treated as an m-dimensional vector by extracting a set of features from the input. Authors apply random projections of the vector to a single vector using randomizations. The similarity of two documents depends on the similarity of the positions at the projection vector.

The Simhash algorithm can be divided into the following four sequential phases. Figure 1 illustrates an example of the way Simhash operates.

Feature extraction During this first phase, a set of k features is extracted from the input file. For example given the following text input "*Our university is a graduate school*" when the features are sequential words of the text grouped in sets of 3 words the output becomes: {*“Our”, ”uni”, ‘ver ”,”sit”, “y_i ”,”is_”, “a_g”, “rad”, “uat”, “e_s”, “coo”, ”l__”*}

Hashing: Each feature is then hashed with a cryptographic hash function and represented as a l-bit array digest.

Accumulation: The set of all digests is accumulated in the following way: Given the set of the binary digests from the previous step an $l \times k$ matrix is constructed. An addition operation is performed at the elements of each column by treating each “0” as -1 and each “1” as 1.

Reduction: Depending on the sign of the numerical value of each element in the array that was constructed from the previous step, the final fingerprint is calculated using the sign of each value in the table. For each negative value or zero a 0 is assigned, and 1 otherwise.

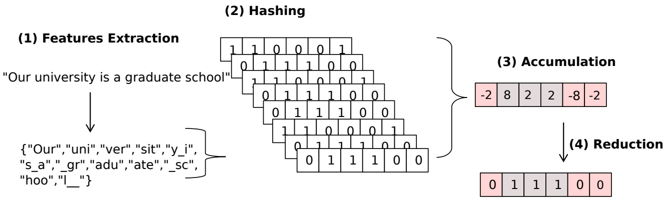


Fig. 1: *Simhash's* phases. In phase 1 the **features extraction** functionality extracts the features from the file given as input. Next the **hashing** procedure occurs whereby all the features are encrypted using cryptographic hash functions. Afterward in phase 3 the **accumulation** operation takes places and in the end from the **reduction** phase the final Simhash digest is computed based on the sign of each number element from the previous phase

C. Error correcting code.

Since the proposed solution consists of a complete data management scheme that assures usage control, this management scheme should of course ensure data reliability additionally. Therefore, a redundancy mechanism becomes a basic building block of the system. Our solution implements an error correcting code (ECC) [15] which encodes a k symbol message into n symbols such that given any k symbols the original message can be reconstructed using the corresponding decoding function. We denote the encoding function as

$\mathcal{E}nc : \{0,1\}^k \to \{0,1\}^n$ and the decoding function as $\mathcal{D}ec : \{0,1\}^k \to \{0,1\}^k$.

D. Content Extractor.

Even though two files may look different following a similarity checking mechanism, their actual content still can be the same. This occurs due to the different representation of a file. Configuration data and layout parameters may result on different representations of the file but the content still can remain the same. The extractor Ext separates the *Data* D from the *metadata* M of a *file* F. We refer to this operation as content extraction implemented by a function Ext. Furthermore when the P2P network is asked to retrieve $content$ the $InvExt()$ function reconstructs the file from both its $content$ and its $metadata$.

V. The proposed Mechanism

A. Overview

As mentioned in the previous section, the proposed solution relies on the existence of a peer to peer (P2P) network. Therefore, the main data management operations are executed through this P2P network following the steps defined in the newly proposed protocol. In this particular P2P network, nodes can have four roles:

- *Producers* UP basically are nodes that wish to share some data in the network. The *producer* generates content and becomes the owner of this specific content. It also specifies the usage control policy rules for the retrieval and the usage of this specific content.
- *Consumers* UC "*consume*" content. These are nodes that wish to retrieve some data. *Consumers* receives the required content only if they fulfill the requirements defined by the policy rules sticked with the relevant content.
- *Caretakers* CT are responsible of both storing content and verifying whether a *consumer* is authorized for the specific usage of the data based on the respective policy defined by the *producer*.
- *Initiators* $\mathcal{I}$ define the set of *caretakers* that are responsible of a specific content upon reception of storage request. They also separate the content data $\mathcal{D}$ and metadata $\mathcal{MD}$ from the file $\mathcal{F}$.

The proposed mechanism is mainly defined by two operations, namely the $storage$ and the $retrieval$. Assuming that not all nodes are legitimate, the operations defined at both phases are distributed among a set of n *caretaker nodes* and such operations are successful only if a threshold number of caretakers collaborate. This threshold number is set regarding the trust degree on the network. During the *storage* phase, the newly proposed protection mechanism assures that similar data are stored and managed by the same set of *caretakers*. The similarity verification is performed using the error tolerant hash function that was defined in section IV-B . During the *retrieval* phase, the *consumer* contacts each relevant *caretaker* which in turn verifies whether the *consumer* fulfills the requirements originating from the policy rules of the targeted content. In the following section, we describe each operation in details.

Throughout the paper we are using the following notations: $\mathcal{F}$, $\mathcal{D}$, MD and $\mathcal{P}$ respectively denote the file that is to be stored in the system, the content of the file, the metadata that includes information needed for the reconstruction of the file and the policy corresponding to the usage of the file. that of the file. $\mathcal{H}_{\mathrm{s}}$ denotes the *Error tolerant hash function* which is the main building block of the protocol and H denote a cryptographic hash function. Gen indicates the *Content Extractor* which separates the data $\mathcal{D}$ from the metadata MD of a file $\mathcal{F}$ and $Inv\mathcal{G}()$ is the *Inverse Content Extractor* that reconstructs the file. Finally $\mathcal{E}nc$ and $\mathcal{D}ec$ denote the encoding and decoding functions of the error correcting code respectively.

B. Storage

We assume the scenario where a producer UP_i wishes to store a file F_i with its predefined policy $\mathcal{P}_i$. The storage protocol is subdivided into the following three main phases:

- ***Initialization***: At this first phase, the producer UP_i sends the file F_i together with its policy $\mathcal{P}_i$ to a set of l *initiators* $\mathcal{I}_i$. These l *initiators* are randomly selected thanks to the use of a regular cryptographic hash function H. UP_i computes $H(F)$ and the output defines $\mathcal{I}_i$. The selection of l *initiators* is random because all nodes are not assumed to be legitimate however the collaboration of at least l nodes is assumed to produce a correct output. The parameter l is predefined and depends on the trust degree of the P2P level. In addition to defining the set of *caretakers* for a particular content, the first role of *initiators* is to extract the content from the file itself. Indeed, *initiators* first extract content D_i and construct its respective metadata MD_i from file F_i using the Extractor Ext described in IV-D. All further operations will be performed over the content D_i.
- ***Node assignment***: The main role of *initiators* is to define the set of *caretakers* that will store the relevant content. Of course, before allowing the storage of the data and in order to protect the

network from **re-injection** attacks, each initiator checks the similarity between files that are already inserted in the system and the candidate content. Therefore, initiators apply an error tolerant hash function $\mathcal{H}_{\mathrm{s}}$ as it is defined in IV-B.
Assume $\mathcal{H}_{\mathrm{s}}(D_j) = hs'_i$. Each initiator then computes the hamming distance between the candidate output hs'_i and the digests of all existing files which are stored in an index. If there exists hs_j in the index such as $HD(hs'_i, hs_j) \leq \sigma$ then initiators identify a re-injection attack and reject the storage request. On the other hand, if initiators agree on the novelty of the candidate content, then the output hs_i defines the unique set of *caretaker* nodes that are in charge of storing the data together with its policy. In order to assure the integrity of this result, a group signature is generated over the tuple $(filename_i || UP_i, \{CT_{i,j}\})$. This tuple is further added to the newly updated P2P filesystem index.
- ***Content and policy storage***: Once the non-similarity verification is successful and the new file references are added in the P2P filesystem index, the data is prepared to be sent to the corresponding *caretakers*. In order to first ensure data reliability, the error correction code described in section IV-C is applied over the data and the metadata separately. Therefore *initiators* generate the newly encoded data blocks $\{e_{i,1}, .., e_{i,n}\}$ and the encoded metadata blocks $\{e'_{i,1}, e'_{i,n}\}$. Initiators further sign each couple $(e_{i,j}, e'_{i,j})$ using a group signature again and send it to the corresponding caretaker node $CT_{i,j}$. Once these encoded blocks received, the *caretaker* $CT_{i,j}$ first verifies initiators' signature and further stores this couple together with its policy.

C. Retrieval

We assume *consumer* UC_v would like to retrieve a file F_i. As opposed to the storage protocol, the retrieval protocol does not use any error tolerant hash function and does not involve *initiators*. Only *caretakers* and *consumers* play a role in this protocol which is divided into the three following phases as in the case of the storage protocol:

- ***data lookup***: *Consumer* UC_v sends a regular P2P lookup request for the file F_i using the filename of F_i. Following the index, UC_v receives the set of *caretakers* that store the data corresponding to F_i.
- ***verification***: UC_v sends a retrieval request to at least k *caretaker* nodes together with its credentials. Each caretaker $CT_{i,j}$ verifies whether *consumer* UC_v's credentials are compliant with the policy P_i. If this verification is successful UC_v receives the corresponding couple $(e_{i,j}, e'_{i,j})$ from each $CT_{i,j}$.
- ***content retrieval and file reconstruction***: Once consumer UC_i receives at least k pairs of encoded blocks $(e_{i,j}, e'_{i,j})$, it applies the decoding function $\mathcal{D}$ over these encoded blocks in order to compute the original blocks and hence retrieve both data D_i and MD_i. Following the information in MD_i, UC_i reconstructs F_i using the inverse extractor $InvExt()$.

VI. Security analysis

In this section we prove the first pre-image resistance property of the Simhash algorithm that is required to prevent re-engineering attacks through which an adversary can derive from a collection of colluding nodes data segments that would be managed by those nodes.

Thanks to the existence of collision resistance cryptographic hash functions [3], the set of nodes that selects the initiators, which in turn define the caretakers for a specific file, through the execution of the Simhash algorithm, are defined in a random manner. Resiliency to changes and collision resistance that are required for the security and the correctness of the usage control policy enforcement scheme are demonstrated through experimental evaluation in section VII.

Theorem 1: $\mathcal{H}_{\mathrm{s}}$ is first pre-image resistant, ie. there is no polynomial adversary A that can reconstruct the content of a file F given the output of the Simhash with probability no better than negligible.

Proof: In order to show that $\mathcal{H}_{\mathrm{s}}$ is first pre-image resistant, we first model the algorithm as a set of three transitions corresponding to the last three phases of the Simhash algorithm, namely, **hashing**, **accumulation**, **reduction** and a set of four states s_1, s_2, s_3 and s_4 where s_2, s_3 and s_4 respectively represent the outputs of each phase and s_1 denotes the input of the Simhash algorithm. The model can be summarized by the following states:

- s_1 : The file is a set of plaintext features.
- s_2 : Fingerprints are hashed.
- s_3 : Each feature is represented as a k long vector after the accumulation phase.
- s_4 : In the end a final fingerprint is available for similarity checking.

Therefore, the proof of Theorem 1 consists of proving that it is hard to find s_1 given s_4. We now sequentially analyze the probability of finding the state before a transition given the state after its execution. We therefore start to evaluate the probability of finding s_3 given s_4. Each number in s_3 is mapped to a bit (0,1) based on its sign. Since the accumulation phase consists of a simple

```
Input: A producer $UP_i$ wants to store file $F_i$ under policy $P_i$
  • $UP_i$: Compute $H(F_i)$. derive the list of $initiators\{I_1,I_2,\ldots, I_l\}$, send $F_i$ and $filename_i$ to each $I_i$
  • $Initiator\ I_i$: Extract $\{D_i, M_i\}$ from $F$, Compute and$\longrightarrow \{CT_{i,1}, CT_{i,2}, \ldots, CT_{i,n}\}$,
    $if\ \forall j\ HD(hs_j, hs'_i) > \sigma$ then:
      1) Store ($filename|UP_i$, $\{CT_{i,1}, CT_{i,2}, \ldots, CT_{i,3}\}$) as index.
      2) Group sign ($\{CT_{i,1}, CT_{i,2}, \ldots, CT_{i,n}\}$)
      3) Encode data:$\mathcal{E}nc(D_i) \longrightarrow \{d_i, r_i\} = e_{i,j}$, $\mathcal{E}nc(M_i) \longrightarrow \{m'_i, r''_i\} = e'_{i,j}$
      4) Send $(e_{i,j}, e'_{i,j})$ to the $CT_{i,j}$ signed with a group signature scheme.
  • $Caretaker\ CT_{i,j}$: $if$ $\mathsf{Verify}(e_{i,j}, e'_{i,j}) := success$
      1) Store $\{(e_{i,j}, e'_{i,j}, P_i)\}$
```

Fig. 2: Storage

```
Input: A $consumer\ UC_v$ seeks to obtain file $F$ under policy $P$ with credentials $C$
  1) $UC_v$ asks for $filename|U_{id}$ gets the list of nodes $\{n_1, n_2, n_3, \ldots, n_l\}$
  2) $UC_v$ is asking for $D_i$ from every participant of the $\{n_1, n_2, n_3, \ldots, n_l\}$ list.
  3) Each $CT_{i,j}$ evaluates credentials $C_i$ for data $D_i$ that she owns
  4) $if$ $\mathsf{Evaluate}(UC_v, C_i, D_i, P_i) = Success$ :
      a) Each $CT_{i,j}$ sends $\{(e_{i,j}, e'_{i,j})\}$ to $UC_v$.
      b) $UC_v$ decodes: $\mathcal{D}ec(e_{i,j}) \longrightarrow \{D_i, P_i\}$ , $\mathcal{D}ec(e'_{i,j}) \longrightarrow \{M_i\}$. and reconstructs the file:
         $InvExt(D_i, M_i) \longrightarrow F$
```

Fig. 3: Retrieval

addition operation of l numbers which are set to either -1 or 1, the resulting sum for each element of the array is an integer between [-l, l]. Hence the probability of finding one element of s_3 is $1/(2l+1)$. Since s_4 is k-bit large, the probability to find s_3 given s_4 is:

$$Pr[s_3 \leftarrow s_4] = (\frac{1}{2l+1})^k$$

We further analyze the hardness of finding s_2 based on s_3. The state s_3 consists of an array T of size k, where each element is a number of size in the range $[-l, l]$ and is the result of the **accumulation** phase of Simhash algorithm as described in IV-B. We compute the probability of an adversary to successfully guess the set of all k numbers such that when summing them accordingly with the description of the **accumulation** phase of Simhash algorithm she can reconstruct the state s_3. Such a probability basically depends on the size l and differs if l is even or odd. We now analyze the probability Pr of finding l numbers whose sum is equal to T_i with respect to the nature of l.

Even: If l is even then $l = 2 \cdot k$ and there are $l+1$ possibilities for the sum. These are:

$$-l, -l+2, -l+4, \ldots, 0, 2, 4, \ldots, l$$

When $T_i = -l$ all the numbers should be equal to 0 as 0 indicates a transformation into -1, hence $Pr[T_i] = 1$.

For each subsequent case where $T_i = -l + 2j$ with $j = 1, .., l/2$, j numbers among l should be equal to ”1” and this probability is $\frac{1}{\binom{l}{j}}$. Therefore, the probability of guessing the l arrays of size k which defines the probability of guessing s_2 given s_3 is:

$$P_r[s_2 \leftarrow s_3] = (\frac{1}{1+l} \cdot \sum_{i=0}^{l} \frac{1}{\binom{l}{i}})^k$$

Odd: Similarly when l is odd, there are only l possible values for T_i; hence, the probability of finding the l values whose is T_i for all k bits is:

$$P_r[s_2 \leftarrow s_3] = (\frac{1}{l} \cdot \sum_{i=0}^{l} \frac{1}{\binom{l}{i}})^k$$

We analyze the hardness of finding s_1 given s_2. This step corresponds to the hashing phase that implements cryptographic hash functions which by their very definition are first pre-image resistant. Finding s_1 from s_2 is as hard as breaking the first pre-image resistance property of a cryptographic hash function.

To conclude, given s_4, we proved that the probability of a polynomial time adversary that can reconstruct s_1 depends on the security of the underlying cryptographic hash function, hence $\mathcal{H}_{\mathrm{s}}$ is first pre-image resistant. ■

VII. Experimental Evaluation

In this section we evaluate the correctness of the resiliency to changes property that is required in our $ETHF$. This property is of significant importance in our scheme as it allows similar data to be mapped to the same set of peers and as such it renders re-injection attacks impossible to occur. The second property that is demonstrated through the evaluation procedure is the collision resistance for two non-similar files given as input an $ETHF$. Indeed an attacker who manages to compute an identical Simhash digest for two non-similar files is be able to produce abnormal behavior, as the usage control policy enforcement scheme will raise an alert in two similar files that are different; thus introducing a false positive into the scheme. These two properties are demonstrated through an intensive experimental evaluation whereby different similarity degrees are assigned to different sets of files. It is very difficult to theoretically prove these two properties since there is no formal definition for similarity given the huge diversity among the representations of content.

A. Experimental Setup

In order to evaluate the accuracy of the underlying Simhash algorithm with respect to different modification operations (either minor or major), we analyze its performance over a large set of files. First, 180 files have been generated using the sci-gen random scientific paper generator [1]. Each file contains 700 words on average. Since the performance may vary with respect to the size of the files, another set of 180 small files has been created by simply extracting 20 consecutive words from each file of the previous set. The original files are further modified following six different scenarios. The modification operations consist in either adding or removing words to/from the original file at either the beginning, the end or from a random position in the file. The significance or impact of the modification depends on the number of words used in each scenario: in the case with small files, this number varies from 3 to 10 whereas for large files either 20 or 60 words are added/removed to/from files.

The Simhash algorithm is implemented using Python 2.7 on a commodity machine with 3.30 GHz Intel Core i5 2500, 8GB memory, 6MB cache which runs Fedora OS. The accuracy of this algorithm is evaluated by applying it over a pair of files and further computing the ratio of the cardinality of the same bits set to the total number of bits of the resulting Simhash digest.

We evaluate the resiliency to changes property by running the Simhash algorithm on every file and its altered version. The similarity degree of two files is computed as the ratio of the cardinality of the same bits set to the total number of bits of the Simhash digest in a percentage form. The result is a collection of 5040 Simhash digests. The sensitivity of the algorithm on different number of words is analyzed as well by running Simhash several times on a file while gradually increasing the number of words that alter the file according to the aforementioned scenarios. The sensitivity shows how unexpectedly Simhash behaves at each scenario. Next we compute the Simhash digests for each possible pair of files from each category (small and large files) and evaluate the collision resistant property for different files.

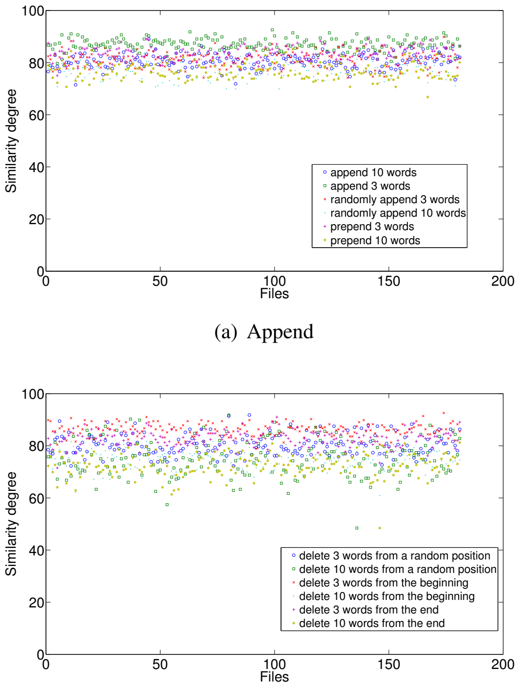
B. Resiliency to changes

As already mentioned, for each scenario presented in the previous section, we apply two different modifications for each original file: in the case with small files, either 3 or 10 words are added/removed/replaced. Figures 4 and 5 show the similarity degree between an original file and an altered one computed for each of the seven modification scenarios and Table I depicts their mean values. The most visual result is that in all scenarios, the average similarity degree is approximately 80%. In the case with small files, there is a significant difference between the two modification operations since the modification which uses only 3 words results in an average similarity degree of 85% whereas the one with 10 words ends up with a degree of 76%. The main reason of this strong difference is the fact that 10 words correspond to the 50% of the size of the content. Hence, this can be considered as a significant modification. Additionally, in order to evaluate the sensitivity of the underlying Simhash algorithm, we also computed the similarity degree with respect to the number of words varying from 1 to 20 for small files. From Figure 6a, we observe that Simhash reacts as expected to the gradual increase of the modification parameter. The analysis with large files (figure 5) basically shows the same behavior as small files with respect to the decrease of the similarity degree with the increase of the modification parameter. Furthermore, we also notice a very large similarity degree for the first two scenarios where words are added (either in the beginning or at the end). To conclude, the proposed error-tolerant hash function outputs a large similarity degree ranging from 72.36% to 97.38% and therefore can be considered as being resilient to minor modifications.

[1] http://pdos.csail.mit.edu/scigen/

TABLE I: Average similarity degree

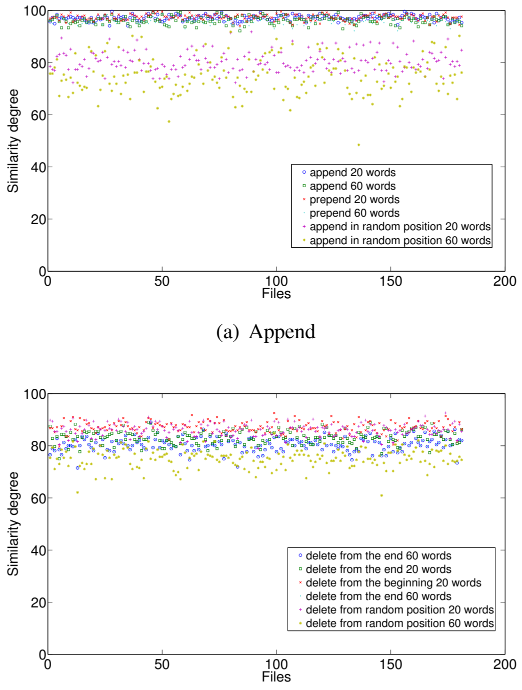
Scenarios	**Small files**		**Large Files**	
	3 words	10 words	20 words	60 words
Prepend	83.34	76.66	97.38	96.17
Append	87.11	80.61	97.16	96.23
Randomly Append	81.26	77.20	80.38	74.88
Pre-Delete	86.21	78.06	87.11	80.61
Delete	82.77	72.36	83.34	76.66
Randomly delete	80.35	74.84	86.21	78.06



(a) Append

(b) Delete

Fig. 4: Small files



(a) Append

(b) Delete

Fig. 5: Large files

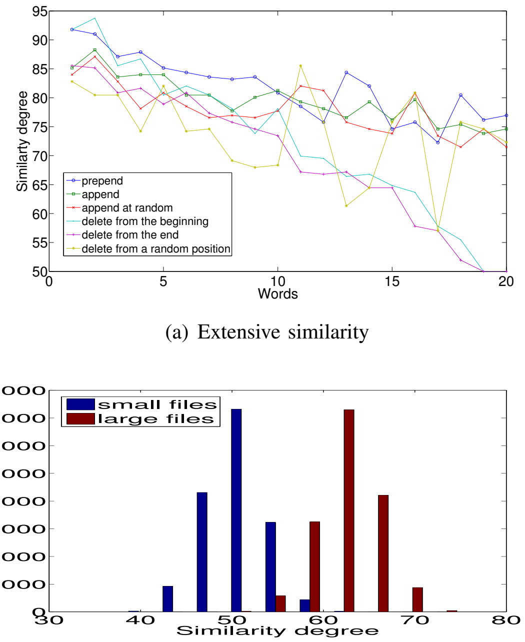
C. *Collision resistance*

Albeit an error-tolerant hash function should provide the same output given two similar files, it should also follow the collision resistance property of a conventional cryptographic hash function in the case where files are significantly different. Therefore, we also compute the similarity degree between all possible two files from the same category. Figure 6b depicts the results originating from the 180 files for each category: the average similarity degree is approximately 50% for small files and 60% for large files from a corpus of $\binom{180}{2} = 16000$ comparisons. The maximum detected similarity degree for large files is 78.39 and for small files 71.48. Even the largest similarity degree is much lower than the minimum similarity degree computed based on similar files. Therefore, we realize that Simhash can be considered as being collision resistant.

D. *Summary*

Via evaluating our scheme with respect to the $ETHF$ we conclude that:

- Similar files end up with a similarity degree that can be accurately defined by a threshold, as the plots in figures 4 and 5 show that the results of Simhash have a low "spread" around the mean value.
- Different files end up with a similarity degree that is lower than the degree defined for similar files according to our test scenarios.



(a) Extensive similarity

(b) Pairwise similarity on different files

Fig. 6

The previously analyzed properties, namely, resiliency to changes and collision resistance imply a random selection of peers to assign caretakers for data management in such a way that similar content is assigned to the same set of peers. It is therefore impossible for potential intruders to apply a re-injection attack by slightly altering the content of the file and gaining the ownership of it. Moreover it is guaranteed that different content will be assigned to a different set of peers.

VIII. Conclusion and Future Work

In this paper we presented a solution for the usage control policy enforcement problem. Namely, usage control defines a continuous validation of the policy imposed by the user during the entire lifetime of its data. The proposed solution assumes the legitimate behavior of t out of n peer nodes in a P2P network. Thanks to the employment of Simhash algorithm which is an $ETHF$, similar data segments are assigned to the same set of peers, therefore rendering potential re-injection attacks impossible to occur. Bypass attacks are not of significant impact in the scheme, since we assume that a copy of a file from a confined environment S and its usage outside this environment with the policy defined by the user doesn't have an intense impact because S has a global coverage.

The security and the correctness of Simhash are demonstrated through an analytical and experimental evaluation respectively. The experimental evaluation shows that Simhash is resilient to changes, thus Simhash contributes to the mitigation of re-injection attacks through various scenarios of possible file manipulations. Furthermore, Simhash is collision resistant: thus it acts as a barrier in the abnormal behavior of our scheme without allowing different content to be assigned to the same set of caretakers. As part of future work we are planning to deploy the proposed solution in a P2P system in order to further evaluate it in terms of communication and computational overhead.

References

[1] M. S. Charikar. Similarity estimation techniques from rounding algorithms. *Proceeding STOC '02 Proceedings of the thiry-fourth annual ACM symposium on Theory of computing*, 2002. 4

[2] R. da Silva Villaca, L. B. de Paula, R. Pasquini, and M. F. Magalhaes. Hamming dht: Taming the similarity search. In *Consumer Communications and Networking Conference (CCNC), 2013 IEEE*, pages 7–12, 2013. 3

[3] S. Goldwasser and M.Bellare. Lecture notes on cryptography. In *Summers course on cryptography, MIT*, pages 136–140. 1996-2008. 6

[4] M. Harvan and A. Pretschner. State-based usage control enforcement with data flow tracking using system call interposition. *Network and System Security, International Conference on*, 0:373–380, 2009. 3

[5] M. Henzinger. Finding near-duplicate web pages: A large-scale evaluation of algorithms. *Proceeding SIGIR '06 Proceedings of the 29th annual international ACM SIGIR conference on Research and development in information retrieval*, 2006. 4

[6] M. Hilty, A. Pretschner, D. Basin, C. Schaefer, and T. Walter. A policy language for distributed usage control. *ESORICS*, pages 531–546, 2007. 3

[7] H. Janicke, A. Cau, F. Siewe, and H. Zedan. Concurrent enforcement of usage control policies. In *Proceedings of the 2008 IEEE Workshop on Policies for Distributed Systems and Networks*, POLICY '08, pages 111–118, Washington, DC, USA, 2008. IEEE Computer Society.

[8] H. Janicke, A. Cau, and H. Zedan. A note on the formalisation of ucon. In *Proceedings of the 12th ACM symposium on Access control models and technologies*, SACMAT '07, pages 163–168, New York, NY, USA, 2007. ACM. 3

[9] B. Katt, X. Zhang, and M. Hafner. Towards a usage control policy specification with petri nets. pages 905–912, 2009. 3

[10] P. Kumari, A. Pretschner, J. Peschla, and J.-M. Kuhn. Distributed data usage control for web applications: a social network implementation. In *Proceedings of the first ACM conference on Data and application security and privacy*, CODASPY '11, pages 85–96, New York, NY, USA, 2011. ACM. 3

[11] G. S. Manku, A. Jain, and A. D. Sarma. Detecting near-duplicates for web crawling. *Proceeding WWW '07 Proceedings of the 16th international conference on World Wide Web*, 2007. 4

[12] J. Park and R. Sandhu. The uconabc usage control model. *ACM Trans. Inf. Syst. Secur.*, 7(1):128–174, Feb. 2004. 3

[13] A. Pretschner, M. Büchler, M. Harvan, C. Schaefer, and T. Walter. Usage control enforcement with data flow tracking for x11. In *5th International Workshop on Security and Trust Management (STM 2009)*, 2009. 3

[14] H. Pucha, D. G. Andersen, and M. Kaminsky. Exploiting similarity for multi-source downloads using file handprints. In *Proceedings of the 4th USENIX conference on Networked systems design &#38; implementation*, NSDI'07, pages 2–2, Berkeley, CA, USA, 2007. USENIX Association. 4

[15] I. S. Reed and G. Solomon. Polynomial Codes Over Certain Finite Fields. *Journal of the Society for Industrial and Applied Mathematics*, 8(2):300–304, 1960. 4

[16] G. Urdaneta, G. Pierre, and M. V. Steen. A survey of dht security techniques. *ACM Comput. Surv.*, 43(2):8:1–8:49, Feb. 2011. 3

[17] B. Zhao, R. Sandhu, X. Zhang, and X. Qin. Towards a times-based usage control model. In *Proceedings of the 21st annual IFIP WG 11.3 working conference on Data and applications security*, pages 227–242, Berlin, Heidelberg, 2007. Springer-Verlag. 3