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

Research Report RR-12-273

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

August 6th, 2012

Iraklis Leontiadis, Refik Molva, and Melek Onen

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

Iraklis Leontiadis, Refik Molva, and Melek Onen
Eurecom, Biot, France
{leontiad, molva, onen}@eurecom.fr

***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 and the way this is disseminated in outsourced storage. The scheme is based on a peer-to-peer architecture whereby a randomly selected set of peers assure usage control enforcement for each data segment. Usage control is achieved based on the assumption that at least t out of any set of n randomly chosen 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 the 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.

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 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. All users having adopted the P2P network for any operation, including ensures usage control enforcement, violations outside this network would not significantly affect the system. Furthermore, even in such a confined environment, an adversary may try to alter some policies by slightly modifying the content and re-introducing it in order to gain the ownership of the data. The proposed enforcement mechanism allows peers to detect similarities between any upcoming data and existing ones, 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 examined and in section 5 we give a detailed description of the scheme. Before closing with

our conclusion and future work in section 8, we examine the security of the proposed mechanism in section 6 and in section 7 we evaluate the correctness of the *error tolerant hash function*.

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, that 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.

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 u_i, imposed by its owner o_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 u_i'. 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 while preventing both **bypass** and **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 an initial solution does not protect the system from **re-injection** attacks. Since the control of each data is decentralized and distributed among a different subset of nodes, any attacker may slightly alter the data and submit it as a fresh new one to the system which will further assign it to different nodes and hence render the attack successful.

Such attacks are avoided thanks to the design of a dedicated data assignment algorithm which detects similarities between any new and already stored data. The algorithm 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 a hash function resilient to changes, the system should define a random generator to select these random peers whose main role is to apply the error-tolerant hash function for data assignment. 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 the previously introduced error tolerant hash function.

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 mainly. However, the use of such functions is not sufficient in order to fully ensure the control over data. The main building blocks of our scheme are described in section IV.

III. Related work

We next give a description of previous related work.

In [1] 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. Janicke *et al* [2] conducted research towards a formalization of usage control policy enforcement through a formal language definition with concrete classes of mechanisms. Usage control policy is enforced by $Executors$ and $Inhibitors$. $Delays$ are appended in content when usage control is forbidden. The mean of enforcement is not presented in this paper even though there is a well defined mathematical notation using the Z language for representation. In a similar direction Zhang *et al* [3] gave a different formalization of usage control using *petri nets*. In [4],[5] the authors proposed a policy based usage control language for usage control enforcement. Zhao *et al* [4] 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* [6] proposed an enforcement scheme which is most closer to the proposed solution in the sense of enforcement in a distributed environment. They depicted it in a form of statecharts and make use o formal language to analyze the dependencies of policies and these will affect the decision made from each $controller$. It differs from our construction as there is insufficient analysis of an inherent mechanism that is practical widely.

Recently a scheme proposed for secure data management over the cloud has been proposed by N. Santos *et al* [7]. Data are sealed based on a particular policy defined by the data owners and only cloud nodes that satisfy the policy requirements can retrieve the data. Our model takes into account the malicious nodes that may collude to the cloud and provides a more general adversary model. Another drawback is the existence of a single point of failure named $monitor$ which acts as solution to the barrier of multiple Trusted Platform Modules (TPM) that act as verifiers to cloud nodes integrity. Nonetheless once the data is released it cannot be controlled by a specific usage control policy.

In [8] the authors proposed a 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 [9], Kumari *et al* enforce the 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* [10] implemented a data flow control mechanism with system calls interposition by controlling the usage control to system calls with a monitoring mechanism. Even though the aforementioned practical usage control enforcement mechanisms are implemented in the different levels of a system we cope with a significant factor which can result in a misbehavior of the usage control on a resource, which is the **re-injection** attacks.

IV. Preliminaries

In order to introduce the proposed scheme, we 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 a usage policy on this particular file the owner of the file 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 can be defined 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 defined in usage control policies follow a protocol based on Distributed Hash Tables (DHT). 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 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 certain subset of peers. $Lookup$ 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 the pieces of data.

As opposed to cryptographic hash functions which given a slightly modified data 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 similar files x_1, x_2 with similarity degree σ then the hamming distance is less than σ. I.e: $HD(\mathcal{H}_{\mathrm{s}}(\mathrm{x}_1), \mathcal{H}_{\mathrm{s}}(\mathrm{x}_2)) \leq \sigma$.*
2) ***first pre-image resistance**: given the result of $\mathcal{H}_{\mathrm{s}}(\mathrm{x})$ it is computationally hard for a probabilistic polynomial time (PPT) adversary to reconstruct x.*

Similarity detection was the focus of several research activities [11], [12]. One of the most performant solution [13] which is nowadays widely used is Charikar's simhash algorithm [14]. 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 now hashed with a cryptographic hash function and represented as a l bit array digest.

Algorithm 1 Simhash algorithm

procedure SIMHASH(a, b) ▷ On input a message m compute its simhash digest
  $f_1, \ldots, f_n \leftarrow E(m)$.
  for all f_1 **do** $h_{i,1}||\ldots||h_{i,l} \leftarrow H(f_i)$
  end for
  for all $j \in \{1, \ldots, l\}$ **do** $T_j := \sum_{i=1}^{n} f_{i,j}$
  end for
  $F_j = \begin{cases} 0 & \text{if } T_j < 0 \\ 1 & \text{otherwise} \end{cases}$
  return $F = F_1||\ldots||F_k$
end procedure

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.

C. Error correcting code.

Since the proposed solution consists of a complete data management scheme that assumes usage control as well, this management scheme should of course ensure data reliability. 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 $\mathcal{E}nc : \{0,1\}^k \rightarrow \{0,1\}^n$ and the decoding function as $\mathcal{D}ec : \{0,1\}^n \rightarrow \{0,1\}^k$.

D. Content Generator.

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 generator Gen separates the *Data* D from the *metadata* M of a *file* F. We refer to this operation as content generation implemented by a function $\mathcal{G}$. Furthermore when the P2P network is asked to retrieve $content$ the $Inv\mathcal{G}()$ function reconstructs the file from both its $content$ and its $metadata$.

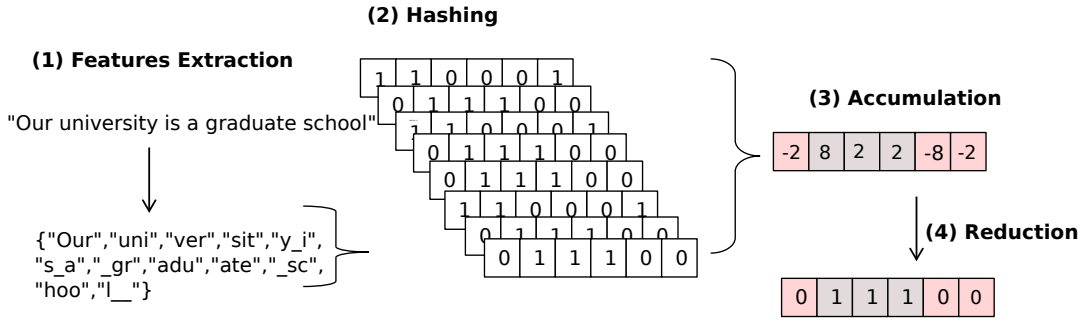

Fig. 1: *Simhash's* phases. In phase 1 the **features extraction** functionality extracts the features from the file given as input. Next the **hashing** procedure occurs whereby all the features are encrypted using cryptographic hash functions. Afterward in phase 3 the **accumulation** operation takes 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

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* 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* "*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* 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* define the set of *caretakers* that are responsible of a specific content upon reception of storage request.

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 1 . 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.

Notations	
$\mathcal{F}$	File
$\mathcal{D}$	Data
MD	Metadata
$\mathcal{P}$	Policy
$\mathcal{I}$	Initiator
UP	Producer
UC	Consumer
CT	Caretaker
$\mathcal{H}_{\mathrm{s}}$	Error tolerant hash function
H	Cryptographic hash function
Gen	Content Generator
$Inv\mathcal{G}()$	Inverse Content Generator
$\mathcal{E}nc$	Encode function
$\mathcal{D}ec$	Decode function

TABLE I: Protocol's notations

The summary of the notations is presented in table 1. $\mathcal{F}$, $\mathcal{D}$, MD and $\mathcal{P}$ respectively denote the file that is to be saved in the system, the content of the file, the metadata that act as representation configurations of the file and. $\mathcal{I}$, UP, UC, CT act for $initiator$, $producer$, $consumer$ and $caretaker$ subsequently. $\mathcal{H}_{\mathrm{s}}$ is 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 Generator* which separates the data $\mathcal{D}$ from the metadata MD of a file $\mathcal{F}$ and $Inv\mathcal{G}()$ is the *Inverse Content Generator* that

is responsible for the reconstruction of the file. Finally $\mathcal{E}nc$ denotes the encoding function of the error correcting code and $\mathcal{D}ec$ is the decoding operation.

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 random selection of l *initiators* are mandatory because all nodes are not assumed to be legitimate but 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 define 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 generator Gen 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 implements a error tolerant hash function $\mathcal{H}_\text{s}$ as it is defined in 1.
  Assume $\mathcal{H}_\text{s}(D_j) = hs'_i$. Each initiator then computes the hamming distance between the candidate output hs'_i and the output of H_s on each file already existing in the index. If there exists a value hs_j in the index whereas $HD(hs'_i, hs_j) \leq \sigma$ 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 policy and over the metadata separately. Therefore *initiators* generate the newly encoded data blocks $\{e_{i,1}, .., e_{i,n+k}\}$ and the encoded metadata blocks $\{e'_{i,1}, e'_{i,n+k}\}$. 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_k 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_k sends a regular P2P lookup request for the file F_i using the filename of F_i. Following the index, UC_k receives the set of *caretakers* that store the data corresponding to F_i.
- ***verification***: UC_k sends a retrieval request to at least n *caretaker* nodes together with its credentials. Each caretaker $CT_{i,j}$ verifies whether *consumer* UC_k's credentials are compliant with the policy P_i. If this verification is successful UC_k receives the corresponding couple $(e_{i,j}, e'_{i,j})$ from each $CT_{i,j}$.
- ***content retrieval and file reconstruction***: Once consumer UC_k receives at least n pairs of encoded blocks $(e_{i,j}, e'_{i,j})$ at least, 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_k reconstructs F_i using the inverse generator $Inv\mathcal{G}()$.

Storage and retrieval protocols are described in figure 2 and figure 3 respectively.

VI. Security analysis

The correctness and security of the proposed mechanism on the one hand relies on the correct behavior of t peers out of any set of n peers as mentioned several times in previous sections; on the other hand, usage control is also achieved thanks to the use of cryptographic hash functions for the selection of initiators and the error tolerant hash function which assigns the same

Input: A producer UP_i wants to save file F under policy P

1) UP_i hashes the file F and gets the list of $initiators$. $\longrightarrow H(F)$ $\{t_1$,t_2,t_3, …, $t_l\}$
2) UP_i sends the file F along with an identification parameter UP_i to the $initiators$.
3) The $initiators$ separate content from metadata using generator $\{D_i, M_i\} \leftarrow \mathcal{G}(F)$
4) The selected nodes compute the list of $caretakers$ $\mathcal{H}_s(D) \longrightarrow \{n_1, n_2, \ldots, n_k\} \longrightarrow = hs'_i$
5) The list of $\{n_1, n_2, \ldots, n_3\}_{signed}$ nodes is signed with a group signature operation by the caretakers.
6) if for all $HD(hs_j, hs'_i) > \sigma$ then:
   a) $filename|U_{id}$ is used as a key to store $\{n_1, n_2, \ldots, n_3\}$.
   b) The $initiators$ encode content,policy and metadata. $\mathcal{E}nc(D_i, P_i) \longrightarrow \{d_i, r_i\} = e_{i,j}$, $\mathcal{E}nc(M_i) \longrightarrow \{m'_i, r''_i\} = e'_{i,j}$
   c) They send $(e_{i,j}, e'_{i,j})$ to the $caretakers$ signed with a group signature scheme.
   d) if $\mathsf{Verify}(e_{i,j}, e'_{i,j}) := success$
      i) $CT_{i,j}$ stores $\{(e_{i,j}, e'_{i,j})\}$
   e) $else$: Reject operation.
7) $else$: Reject operation.

Fig. 2: Storage

Input: A $consumer$ U seeks to obtain file F under policy P with credentials C

1) UC_k asks for $filename|U_{id}$ gets the list of nodes $\{n_1$,n_2,n_3, …, $n_l\}$
2) UC_k is asking for D_i from every participant of the $\{n_1$,n_2,n_3, …, $n_l\}$ list.
3) Each $CT_{i,j}$ evaluates credentials C_i for data D_i that she owns
4) if $\mathsf{Evaluate}(UC_k, C_i, D_i, P_i) = Success$:
   a) Each $CT_{i,j}$ sends $\{(e_{i,j}, e'_{i,j})\}$ to UC_k.
   b) UC_k decodes: $\mathcal{D}ec(e_{i,j}) \longrightarrow \{D_i, P_i\}$, $\mathcal{D}ec(e'_{i,j}) \longrightarrow \{M_i\}$.
   c) UC_k applies the inverse generator function $Inv\mathcal{G}()$ that regenerates content. $Inv\mathcal{G}(D_i, M_i) \longrightarrow F$
5) $else$ Reject operation

Fig. 3: Retrieval

caretakers to similar files to detect re-injection attacks. In this decentralized architecture, since any node can have a malicious behavior, a malicious producer should not be able to assign the caretakers for a given file. Furthermore, it should neither be able to choose the set of initiators to perform this operation because of collusion risks between initiators and himself. Thanks to the use of cryptographic hash functions a different randomly selected set of initiators is defined for each different file.

Furthermore, the assignment operation itself assures that similar files are attributed to the same set of caretakers thanks to the use of error tolerant hash functions. As defined in definition IV-B, an adversary should not be able to retrieve a file given the output of an $ETHF$. In this section, we prove that $simhash$ is first-preimage resistant.

Theorem 1: $\mathcal{H}_s$ 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 $\mathcal{H}_s$ 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 **hashing** phase. 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.

and the following transitions:

- t_1 corresponds to the hashing phase and transforms s_1 to s_2.

- t_2: Each hashed feature is accumulated based on index with all the other hashed features to construct a single vector.
- t_3: The accumulated values are mapped into a binary vector: depending on the sign of the numerical value of the array that was constructed from the previous step, for each negative or 0 a 0 is placed, otherwise a 1.

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, starting from t_3 until t_1. We therefore start to evaluate the probability of finding s_3 given s_4. In t_3, each number in s_3 is mapped to a bit (0,1) based on its sign. Since the accumulation phase consists of a simple 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:

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

The previous equation shows that unfortunately, it is not hard to retrieve s_3 from s_4. 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 l and is the result of the **accumulation** phase of $simhash$ algorithm as described in IV-B. We compute the probabilities 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_2. There are two possible cases. Such a probability basically depends on the number l and differs if l is even or odd. We now analyze the probability $Pr[T_i]$ of finding the 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 of the sum. These are:

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

When $T_i = -l$ the solution is trivial and all the numbers should be equal to 0 as 0 indicates a transformation into -1, hence $Pr[T_i] = 1$. If $T_i = -l+2$ then one of the numbers that make this particular T_i should be 1. As such an adversary can reconstruct this number with probability:

$$Pr[T_i] = \frac{1}{\binom{l}{1}} = \frac{1}{l}$$

In the case where $T_i = -l+4$ there are 2 elements whose value is 1 and $l-2$ whose value is 0. As such $Pr[T_i] = \frac{1}{\binom{l}{2}}$ Following the previous equation, since there are $l+1$ possible values that T_i can have, the probability of correctly guessing the l numbers whose sum is T_i is defined in the following equation:

$$Pr[T_i] = 1 + \frac{1}{\binom{l}{1}} + \frac{1}{\binom{l}{2}} + \ldots + \frac{1}{\binom{l}{l-1}} + 1$$

Since array T is k in length the probability of successfully guessing all the numbers that construct each value of the array 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. t_1 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 is no better than negligible, hence $\mathcal{H}_s$ is first pre-image resistant. ■

VII. Evaluation

We implement the $\mathcal{H}_s$ simhash function and we run several times the algorithm given as input to it text files. The experiments are conducted in two steps. The accuracy of the newly proposed error-tolerant hash function is first evaluated by applying the simhash algorithm over small text files where the content consists of 140 characters. We extend previously implemented work in plagiarism detection competitions [16], [17], [18] by defining the following sophisticated attack scenarios that can be considered as being relevant for file similarity detection in the context of the protection against re-injection attacks.

Small files alterations

1) Append words in the beginning.
2) Append words at the end of the file.
3) Append words in the middle of the file.

We apply the simhash algorithm several times changing the number of shingles. The set of shingles is the

extracted features each of them defined by a sequential number of words. The size of each set is randomly chosen. The higher the similarity metric results to more similar the files.

shingles	Similarity
2	89.84
3	87.89
4	89.45
5	87.05

(a) Append in the beginning

shingles	Similarity
2	78.90
3	76.17
4	65.62
5	66.40

(b) Append in the end

shingles	Similarity
2	75.39
3	75.39
4	66.40
5	69.92

(c) Append in the middle

TABLE II: Small files comparisons.

As shown from the previous tables the similarity of altered files according to our different re-injections scenarios in what we consider small files can be accurately estimated by observing our results. By changing the number of each shingle that will constitute the set of extracted features we achieved almost the same results. By doing so it is straightforward that the algorithm is more resilient to the set of extracted features that in our case is the number of consequent words inside a document. Thus by taking the smallest value of the simhash output as a minimum bound of similar files we can identify possible re-injection attacks on small files.

In a second step, we apply the same algorithm over bigger files up to many pages as scientific papers. The similarity ratio under the previously defined three attacks always outputs 100% since such modification is not considered as a significant alteration in a big file. Therefore, we have defined a second set of attack scenarios which seems more appropriate for the case with big files:

Big files alterations

1) Replacement of words that present high entropy.
2) Replacement of the most used words.
3) Removal of a set of sequential words.
4) Addition of random text.

Following the same practice for big files (Table 2) we changed the number of shingles. We observed that by changing the words that present the highest entropy we achieve the highest similarity results. High similarity results are shown also to addition and removal of a set of sequential words . Thus we can estimate in big files with accuracy whether or not possible re-injection attacks have been conducted.

shingles	Similarity
2	97.26
3	96.87
4	95.70
5	97.65

(a) Replace words with high entropy

shingles	Similarity
2	78.51
3	76.17
4	76.17
5	69.53

(b) Replace the most used word

shingles	Similarity
2	94.92
3	89.45
4	84.37
5	80.46

(c) Remove a set of sequential words

shingles	Similarity
2	94.92
3	89.45
4	84.37
5	80.46

(d) Add a set of sequential words

TABLE III: Big files comparisons.

To summarize our results in the evaluation section we showed that the $\mathcal{H}_{\mathrm{s}}$ is resilient to changes according to the experiments presented above in various re-injection scenarios in big and small files.

VIII. Conclusion and Future Work

In this paper we presented a scheme for secure usage control policy enforcement. The main building block of our scheme is an error-tolerant hash function that guarantees the unique assignment of storage nodes according to the $content$ of the $file$ while still assigning similar files to the same set of storage nodes. We evaluated the accuracy of the $simhash$ algorithm with some experiments run over different files with different attacking scenarios. The scheme is easy to implement leveraging existing P2P systems that underpin to the retrievability and the random participation-assignment. Our solution is based on the underlying hash functions and re-injection attacks are mitigated using an error tolerant hash function. We proved that $ETHF$ is first pre-image resistant.

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] J. Park and R. Sandhu, "The uconabc usage control model," *ACM Trans. Inf. Syst. Secur.*, vol. 7, no. 1, pp. 128–174, Feb. 2004. [Online]. Available: http://doi.acm.org/10.1145/984334.984339

[2] 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*, ser. POLICY '08. Washington, DC, USA: IEEE Computer Society, 2008, pp. 111–118. [Online]. Available: http://dx.doi.org/10.1109/POLICY.2008.44

[3] B. Katt, X. Zhang, and M. Hafner, "Towards a usage control policy specification with petri nets," pp. 905–912, 2009. [Online]. Available: http://dx.doi.org/10.1007/978-3-642-05151-7_11

[4] 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*. Berlin, Heidelberg: Springer-Verlag, 2007, pp. 227–242. [Online]. Available: http://dl.acm.org/citation.cfm?id=1770560.1770583

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

[6] 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*, ser. SACMAT '07. New York, NY, USA: ACM, 2007, pp. 163–168. [Online]. Available: http://doi.acm.org/10.1145/1266840.1266867

[7] N. Santos, R. Rodrigues, K. P. Gummadi, and S. Saroiu, "Policy-sealed data: A new abstraction for building trusted cloud services," in *USENIX Security Symposium*. USENIX Association, 2012.

[8] 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.

[9] 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*, ser. CODASPY '11. New York, NY, USA: ACM, 2011, pp. 85–96. [Online]. Available: http://doi.acm.org/10.1145/1943513.1943526

[10] 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*, vol. 0, pp. 373–380, 2009.

[11] 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 & implementation*, ser. NSDI'07. Berkeley, CA, USA: USENIX Association, 2007, pp. 2–2. [Online]. Available: http://dl.acm.org/citation.cfm?id=1973430.1973432

[12] 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.

[13] 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.

[14] 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.

[15] I. S. Reed and G. Solomon, "Polynomial Codes Over Certain Finite Fields," *Journal of the Society for Industrial and Applied Mathematics*, vol. 8, no. 2, pp. 300–304, 1960. [Online]. Available: http://dx.doi.org/10.1137/0108018

[16] M. Potthast, B. Stein, A. Eiselt, B. universitt Weimar, A. Barrn-cedeo, and P. Rosso, "P.: Overview of the 1st international competition on plagiarism detection," in *In: SEPLN 2009 Workshop on Uncovering Plagiarism, Authorship, and Social Software Misuse (PAN 09), CEUR-WS.org*, 2009, pp. 1–9.

[17] M. Potthast, A. Barrn-Cedeo, A. Eiselt, B. Stein, and P. Rosso, "Overview of the 2nd international competition on plagiarism detection." in *CLEF (Notebook Papers/LABs/Workshops)*, M. Braschler, D. Harman, and E. Pianta, Eds., 2010. [Online]. Available: http://dblp.uni-trier.de/db/conf/clef/clef2010w.html#PotthastBESR10

[18] M. Potthast, A. Eiselt, A. Barrón-Cedeño, B. Stein, and P. Rosso, "Overview of the 3rd international competition on plagiarism detection," in *CLEF (Notebook Papers/Labs/Workshop)*, V. Petras, P. Forner, and P. D. Clough, Eds., 2011.