A P2P Content Authentication Protocol Based on Byzantine Agreement

Esther Palomar, Juan M. Estevez-Tapiador,
Julio C. Hernandez-Castro, and Arturo Ribagorda

Computer Science Department – Carlos III University of Madrid
Avda. Universidad 30 – 28911, Leganes, Madrid
{epalomar, jestevez, jcesar, arturo}@inf.uc3m.es

Abstract. One of the main advantages of peer-to-peer (P2P) systems is their capability to offer replicas of the same content at various locations. This allows to access contents even when some nodes are disconnected. However, this high degree of redundancy implies that it is necessary to apply some security mechanisms in order to avoid attacks based on non-authorized content modification. In this paper, we propose a content authentication protocol for pure P2P systems. Under certain restrictions, our scheme provides guarantees that a content is authentic, i.e. it has not been altered, even if it is a replica of the original and the source has lost control over it. Our proposal relies on a set of peers playing the role of a certification authority, for it is unrealistic to assume that appropriate trusted third parties can be deployed in such environments. Finally, we discuss some of its security properties through several attack scenarios.

1 Introduction

In a peer-to-peer (P2P) network, peers communicate directly with each other to exchange information. One particular example of this information exchange, that has been rather successful and has attracted considerable attention in the last years, is file sharing. This kind of systems are typically made up of millions of dynamic peers involved in the process of sharing and collaboration without relying in central authorities. P2P systems are characterized by being extremely decentralized and self-organized. These properties are essential in collaborative and ad-hoc environments, in which dynamic and transient population prevails.

The popularity of these systems has motivated inspiring research lines in the application of distributed P2P computing. New approaches have also been presented, such as scalability, robustness and fault tolerance, organization and coordination, adaptability, distributed storage, location and retrieval, reputation, and security. In particular, security advances have focused on anonymity, access control, integrity, and availability. New areas are being explored, such as fairness and authentication [3].

A significant part of the research on security in P2P systems intends to mitigate attacks against four main system properties: availability, authenticity, access control, and anonymity. Recent work primarily focus on addressing attacks against availability and authenticity [4]. For instance, some results already exist on the security of traditional Gnutella-like systems, in particular, concerning availability and authenticity [13]. Different authors have also studied how to use a P2P network to prevent DoS attacks on the Internet [8, 11]. On the other hand, works such as [2] study how to use P2P networks to provide user anonymity. Furthermore, current architectures for P2P networks are plagued with open and difficult issues in digital rights management and access control (e.g., [7] outlines some of the problems in this area).

Many of the network security services offered today rely in public key cryptography. One of the most important issues when dealing with public keys is ensuring their authenticity. In environments such as Internet, the classic solution relies on the existence of a Public Key Infrastructure (PKI). A hierarchy of Certificate Authorities (CA) can assure whether a public key belongs to someone or not. Nevertheless, it is not realistic to assume that trusted third parties (TTP) can be deployed in a P2P network, especially in the case of a mobile ad-hoc network, where there is a lack of fixed infrastructure [12].

A recent work addresses the issues related to this problem in P2P systems [14]. Authors introduce a scheme based on Byzantine agreement for authenticating public keys. The proposed mechanism is autonomous and does not require the existence of a trusted third party in charge of issuing certificates to ensure key authenticity. The key point is that the scheme works correctly if the number of honest peers in the network is above a certain threshold. Due to its relevance as an essential building block in the protocol proposed in this paper, we further elaborate on this proposal below.

An important issue in file sharing systems is to guarantee the authenticity of the shared resources, i.e. to ensure that distributed replicas have not been modified in a non-authorized way. A digital content is straightforwardly alterable; it can be manipulated, so that a binary stream looks like the original. If a public key can be securely associated to a party, the integrity of a content generated by her can be ensured as follows.

First, the source:

1. Computes a hash value from the content.
2. Encrypts the hash value with her private key, obtaining a digital signature.
3. The signature is enclosed together with the digital certificate which contains the user's public key.
4. A CA validates the sender's digital signature.

Then, the receiver:

1. Computes a hash value from the received content.
2. Decrypts the digital signature enclosed by using the public key certificate, thus generating a second hash value.
3. Compares both hash values to confirm the non-alteration of the content.

In this paper we concentrate in designing a secure content distribution protocol for P2P networks. For this, we rely on some results derived from well-studied problems such as those of reaching consensus in presence of traitors. Content authentication confirms non-alteration and source identification of the content, implemented through a digital content certificate. Our motivated scenarios are the implementation of these concepts in future P2P networks, in collaborative environments, and file sharing applications in order to make them more reliable and secure.

The rest of this paper is organized as follows. In Section 2, we briefly introduce the scheme suggested in Pathak and Iftode's paper [14] for public key authentication in P2P systems. Section 3 describes our proposal, while Section 4 is devoted to provide and informal security analysis through a number of attack scenarios. Finally, Section 5 concludes the paper and discuss some open issues and future work.

2 Public Key Cryptography for Pure P2P Systems

Since malicious attacks and dishonest peers can cause faulty nodes to exhibit Byzantine behavior, fault-tolerant algorithms are becoming increasingly important in many environments. The work described in [10] proposes a mechanism based on erasure coding for data durability, plus a Byzantine agreement protocol for consistency and update serialization. This technique is based on breaking the data into blocks and spreading them over many servers. The objects and their associated fragments are then named using a secure hash over the object contents, giving them globally unique identifiers. This provides data integrity by ensuring that a recovered file has not been corrupted (for a corrupted file would produce a different identifier). Blocks are dispersed with special care in order to avoid possible correlated failures, picking nodes in different geographic locations or administrative domains, or based on models of historical measurements.

Pathak and Iftode [14] apply the ideas presented in the Byzantine Generals Problem [9] for public key authentication in pure P2P systems. The Byzantine Generals Problem [9] consists in deciding if a group of distributed generals must "attack" or "retreat" the enemy. Each Byzantine army is camped around an enemy city. Each base communicates among each others sending conflict information, with the vulnerability of traitors and enemies who try to prevent the loyal generals from reaching a plan. Authors show that, using oral messages, this problem is solvable if and only if more than 2/3 of the generals are loyal, or it is the same, no solution with fewer than 3m+1 general can cope with m traitors; but with signed messages, the problem is solvable for any number of general and possible traitors.

In Pathak and Iftode's scheme, it is postulated that a correct authentication depends on an honest majority of a particular subgroup of the peers' community, labeled "trusted group". However, in P2P systems an authenticated peer could create multiple fake identities and act maliciously in the future (Sybil attack [6]).

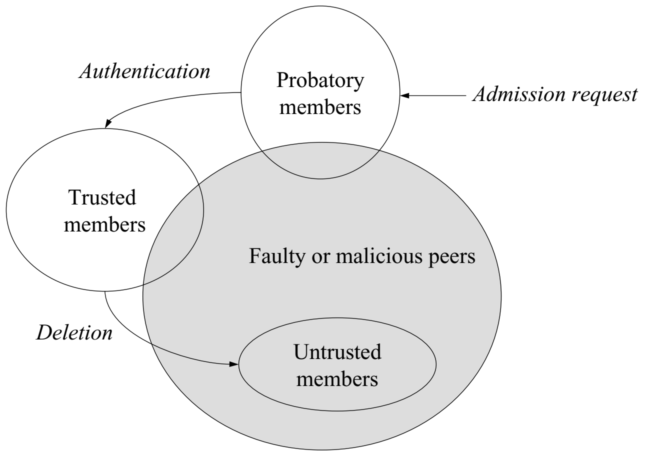

Fig. 1. Community structure according to the authentication state of each node

For this reason, the classification of the rest of the community maintained by each node (see Fig. 1) has to be proactive and should be periodically flushed. A periodic pruning of the trusted group will ensure honest majority. Thus, honest members from trusted groups are used to provide a functionality similar to that of the PKI through a consensus procedure.

The authentication protocol consists in the four phases that are briefly discussed below. Interested readers can find further details in [14].

1. **Admission request**. The protocol begins when B (Bob) run into a newly discovered peer A (Alice), which claims to be the owner of an unauthenticated public key K_A. Then, B asks to a subgroup of his trusted group for helping him in verifying the authenticity of K_A (Fig. 2, message 1). Finally, B sends K_A to those trusted peers that agree.
2. **Challenge response**. Each notified peer challenges Alice by sending a random nonce encrypted with Alice's supposed public key (Fig. 2, message 2). Alice is able to return each received nonce if and only if she holds the corresponding private key, K_A^{-1} (Fig. 2, messages 3). Each challenger checks if the received response is correct, thus obtaining a proof of possession of K_A.
3. **Distributed authentication**. Each peer helping Alice sends her proof of possession to Bob (Fig. 2, messages 4). If all peers are honest, then there will be a consensus, so Bob gets the authentication result: K_A belongs to Alice or not. However, some of the peers summoned by Bob could be malicious or faulty, which may result in Alice receiving different opinions about the authenticity of K_A. In this case, Bob must initiate the Byzantine agreement phase.
4. **Byzantine agreement**. First, Bob verifies if Alice is malicious by sending to her a proof request message. Alice must respond with all challenge messages received, and the respective responses sent by her (Fig. 2, message 5). If A

is honest, she can provide a correct response and also demonstrate her good behavior by sending to B the challenges she received and the corresponding responses. If A cannot be proved to be malicious, then some of the peers must be. At this stage, B announces a Byzantine fault to the group (Fig. 2, message 6). Each group member sends an agreement message to others. At the end of this phase, the honest peers will be able to recognize malicious peers causing the split in authentication votes.

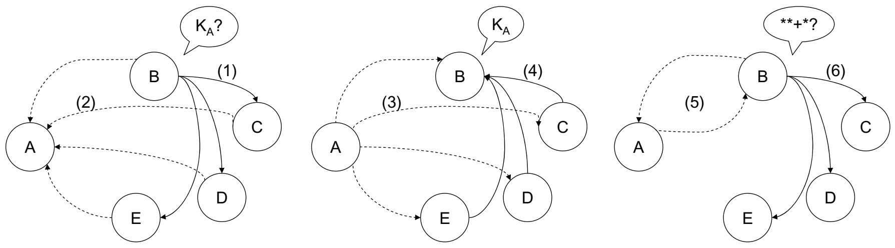

Fig. 2. Authentication protocol phases. Node A belongs to "others" group. Node B authenticates A using its trusted peers, and one of them turns malicious that tries to prevent authentication of A, C.

Successful authentication moves a peer to B's trusted group, while encountered malicious peers are moved to the untrusted group. Peers can be also deleted from trusted groups due to lack of liveliness and periodic pruning of the group.

The fundamental limit of this scheme is the following. Let N be the number of peers in the community; t the number of malicious or faulty peers; and ϕ a fraction of N, denoting that ϕN peers may not be reached during the protocol execution and another ϕN peers exhibit faulty behavior because the path between the source and them suffers a man-in-the-middle attack. Then, authors postulate that the community has honest majority if $t < \frac{1-6\phi}{3}N$. As the value of ϕ does not change the behavior of random selection, then we can consider that a group has honest majority with $3t + 1$ peers [14].

Summarizing, the previous protocol provides us with public key cryptography without relying in certification authorities. This can be viewed as an essential building block upon which more complex security schemes can be developed. As an example, Pathak and Iftode mention its application within an e-mail authentication system named SAM (Self Authenticating Mail).

3 A P2P Content Authentication Protocol

One of the most interesting aspects of P2P systems is the possibility of replicating the same content among different nodes. In many occasions, this task is not performed in a proactive way, but it is simply the result of the existence of a file sharing application. By using a search mechanism, users can locate a specific content and then download it from its source. Once a user gets the file, it is usual that a local copy will remain in the node, in such a way that future queries will identify the node as one of the various locations from which the content can be obtained.

This scheme presents some interesting properties. Faced with different locations of the same content, an application can grant priority to that which offers a less expensive path (e.g. in terms of bandwidth). To some extent, replication also guarantees some sort of fault tolerance, since information can be available even if some parts of the network are temporarily disconnected.

In a collaborative environment, previous features are highly desirable. However, it is unrealistic to assume that every integrating node will exhibit a honest behavior, even if they have always behaved correctly in the past. Once a content is replicated through different locations, the originator loses control over it. A malicious party can modify the replica according to several purposes:

- To claim ownership over the content.
- To insert malicious software into a highly demanded content. Not in vain, P2P networks are becoming an important medium to propagate recently developed viruses, spyware, etc.
- To boycott the system by offering fake contents. Eventually, this can generate distrust and bad reputation in the community.

In classic networking paradigms, guarantees of authenticity and integrity can be provided by digital signatures. If an authenticated user, A, wishes to offer a content m, she can rely on a CA to generate and sign an associated certificate, C_m, which can be checked by the rest of the community and also ensures that m has not been modified. Even tough the previous approach has been successfully applied in several domains (i.e. for public-key authentication), it requires the existence of trusted third parties. The reasons why A cannot sign her own certificates are simple. First, because she can misbehave, offering something different of what she announces. Furthermore, her signature alone does not prevent from manipulation. Suppose that A offers m in the form of a pair $< m, s_A(m) >$, being $s_A(m) = encryption(K_A^{-1}, h(m))$ the signature of A over m, and $h(m)$ any appropriate hash function. Once B obtains m, she can modify it and generate a new signature over the altered content. Moreover, even if B does not modify m, she can just remove $s_A(m)$ and add her own signature. As a result, several –and probably different– copies of m claimed by various parties may be circulating through the network.

As explained before, the key point is that assuming the existence of trusted third parties is an unrealistic hypothesis in P2P systems. Basically, the approach described in this paper relies on a honest majority of the nodes playing the role of a trusted CA. The owner of the content is responsible of generating a certificate containing the most important features of m, while a selected subset of the community signs it. Even though several signers do not constitute by themselves a proper “trusted third party”, some security properties can be ensured if the group has honest majority.

3.1 Assumptions and Notation

Before presenting the details of this proposal, we assume the following five working hypotheses:

1. Assured transactions without rejections. The absence of a message can be detected. This can be provided by using a scheme based on timeouts.
2. Identification of all participants is required through a unique pseudonym, the IP address, a network name, etc. Anonymity is not desired by now.
3. Identification of contents is also required. A unique name, which is also used for searching the content, is associated with the content.
4. Digital signatures cannot be forged unless the attacker get access to private keys.
5. Anyone can verify the authenticity of a node's signature by applying the Byzantine fault tolerant public key authentication protocol presented in [14].

Even though some terms have already been introduced, we summarize the notation that will be used throughout the paper:

- N is the number of network nodes.
- Each node is denoted by n_i. Eventually, specific nodes will be designated by capital letters: A, B, ...
- Each node n_i has a pair of public and private keys, denoted by K_i and K_i^{-1}, respectively.
- m denotes the content that nodes wish to publish.
- $h(x)$ represents a hash function on x.
- $s_i(x) = \mathit{encryption}(K_A^{-1}, h(x))$ is the signature of n_i over x.

3.2 Content Certificates

We are interested in avoiding non-authorized content alteration. For this, previously to its diffusion, an entity A generates a certificate C_m for content m, containing:

- The identity of the originator, which ultimately establishes who has generated the content and is its legitimate owner.
- The identity of the contents.
- A hash, $h(m)$, of m, assuring its integrity.
- An ordered list of signers (OLS) of the certificate. It contains the identity of $k + 1$ network nodes, denoted by $n_0, n_1, \ldots, n_k$, being $n_0 = A$ the content originator. The nodes are selected among A's trusted group.
- Finally, the previous items are recursively signed by the nodes listed in the OLS. First, A signs the certificate. The resulting signature is subsequently checked and signed by n_1, and so on.

The structure of the certificate is summarized in Fig. 3(a). Of course, this is just a functional description of the key elements contained in the certificate. In a real application, it would be necessary to include additional fields, such as

Content certificate $\mathbf{C_m}$

`Certificate` C:
`Originator:` A
`ID:` I_m
`Contents:` $h(m)$
`OLS:` $A, n_1, \ldots, n_k$
`Signatures:`
$s_{n_k}(\cdots(s_{n_1}(s_A(C))))$

(a)

Table of signed certificates $\mathbf{T_i}$

Date	Certificate received	Signature $\mathbf{s_{n_i}}$
$Time_1$	C_{m_1}	s_{i1}
$\vdots$	$\vdots$	$\vdots$
$Time_n$	C_{m_n}	s_{in}

(b)

Fig. 3. (a) Content certificate; (b) local database maintained by each certification node

a code indicating the hash function which has been used, a timestamp and an expiration date, etc.

As it will be justified below, each node must maintain a local register with the certificates it has previously signed. Fig. 3(b) shows the fields that this table should store: a timestamp, the received certificate (including the signatures contained), and the signature generated by the node.

3.3 Certificate Generation

Before C_m can be used to ensure content authenticity and integrity, it must be progressively signed by the nodes included in the OLS. At each stage, the next node in the OLS adds its signature to the previous ones. Due to the structure of the chain of signatures, this task cannot be carried out in parallel. We will denote by $C_0, C_1, \ldots, C_k = C_m$ the successive versions of the certificate as it passes through the list of nodes.

The certificate is initialized by the originator, A, who selects an appropriate value for the number k of signing nodes and their identities (see discussion below). Next, A generates C_0 by providing the first signature and passes it to the next node in the OLS.

Local Verification. Each node n_i should perform a *local verification* stage when it receives certificate C_{i-1}. The purpose of this phase is to ensure the correctness of both the certificate and the previously added signatures. This consists in the following three steps:

- *Certificate verification.* Each peer verifies the correctness of the information received from the previous peer in the list of signers. This includes obtaining A's public key, computing $h(m)$ and comparing it with the value contained in the received certificate. The node must also check T_i and verify that no entries exist corresponding to the same content.
- *Signatures verification.* Each peer verifies the signatures contained in the received certificate according to the list order. If any public key is unknown, it can be acquired with an instance of Pathak and Iftode's public key authentication protocol.

- *Local management.* If previous verifications succeed, the node adds its signatures to C_{i-1}, thus creating C_i. Each peer stores separately C_{i-1} and the generated signature, $s_{n_i}(C_{i-1})$, in her local database T_i.

Signing the Certificate. We have identified two different ways in which the certificate can be signed by the nodes included in the OLS. Fig. 4 graphically illustrates both alternatives. The main differences are the following:

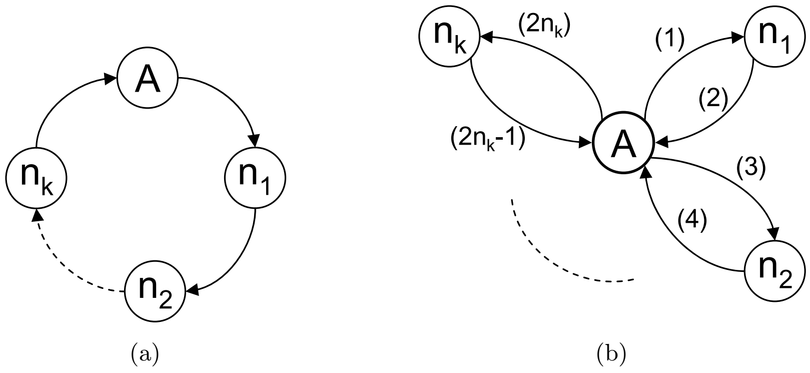

Fig. 4. Alternative procedures for certificate generation

- In the first one, illustrated in Fig. 4(a), each node is responsible of sending the signed certificate to the next node in the list. This way, A simply sends C_0 to n_1 and waits until C_m arrives. Although it is not explicitly pointed out in the figure, we assume that each peer must send a notification message to A when it passes the certificate to the next node. This, together with appropriate timeouts, allows A to be aware of the current state of the process.
- In the second alternative (Fig. 4(b)), A is responsible of sending C_{i-1} to each node and receiving C_i. Now, A can check whether the received certificate has been properly signed or not, thus having a higher level of control over the process. However, note that each node still has to perform its local verification stage in order to ensure that it is not being cheated on.

A more precise description for both alternatives is provided in Figs. 5 and 6.

3.4 Certificate Verification

Once obtained m, the certificate should be checked to ensure the authenticity and integrity of m. For this, a node B performs the following steps:

- *Step 1.* B computes $h(m)$ from m and compares the result with that included in the certificate. If both values differ, then either m has been altered or C_m is not an authentic certificate for m.

- *Step 2.* B obtains the public keys associated to each of the peers listed in the OLS. If any public key is unknown to B, it can be acquired by executing Pathak and Iftode's public key authentication protocol.
- *Step 3.* B verifies the chain of signatures by recursively encrypting C with the ordered list of public keys.

Protocol for distributed content certificate generation
(Note: $A = n_0$)

```
1. $n_0$ generates $C$ and signs: $C_0 =< C, s_{n_0}(C) >$
2. $n_0$ sends $(m, C_0)$ to $n_1$
3. For $i = 1$ to $k$
        (a) $n_i$ performs the local verification stage on $C_{i-1}$
        (b) $n_i$ adds its signature and generates $C_i$
        (c) $n_i$ updates $T_i$ with the tuple $< timestamp, C_{i-1}, s_{n_i}(C_{i-1}) >$
        (d) $n_i$ sends $(m, C_i)$ to $n_{i+1(mod\ k)}$
        (e) $n_i$ sends a notification message to $n_0$
```

Fig. 5. Distributed content certificate generation

Protocol for centralized content certificate generation
(Note: $A = n_0$)

```
1. $n_0$ generates $C$ and signs: $C_0 =< C, s_{n_0}(C) >$
2. For $i = 1$ to $k$
        (a) $n_0$ sends $(m, C_{i-1})$ to $n_i$
        (b) $n_i$ performs the local verification stage on $C_{i-1}$
        (c) $n_i$ adds its signature: $C_i =< C, s_{n_i}(C_{i-1}) >$
        (d) $n_i$ sends $C_i$ to $n_0$
        (e) $n_i$ updates $T_i$ with the tuple $< timestamp, C_{i-1}, s_{n_i}(C_{i-1}) >$
        (f) $n_0$ verifies the correctness of the signature of $n_i$
```

Fig. 6. Centralized content certificate generation

4 Security Analysis

In this section, we provide an informal analysis about the security of the proposed scheme. For this purpose, we discuss several attack scenarios and forms of malicious behavior which can occur during each phase of the protocol execution.

4.1 Content Authenticity

The protocol is initiated by n_0 and relies on her honesty. In case of n_0 being honest, we can assume that the original copy of m is authentic, the hash function is not manipulated, and the OLS contains A's trusted members. Therefore, the initial content certificate, C_0, is correct.

The originator might try to exhibit a malicious behavior. Any modification of the certificate fields will be eventually detected by, at least, one node in the OLS, since it is assumed honest majority, n_0 cannot collude with a sufficient number of traitors. This also prevents a group of malicious nodes to collude during the signing process in order to forge a false certificate. Suppose that a signer B, who belongs to the OLS, gets the pair $< m, C_m >$ and tries to generate a fake certificate, C'_m, as follows:

`Certificate` C':
    `Originator:` B
    `ID:` I_m
    `Contents:` $h(m)$
    `OLS:` $B, n_{1'}, \ldots, n_{k'}$

`Signatures:` $s_{n_{k'}}(\cdots(s_{n_{1'}}(s_B(C'))))$

This behavior will be detected by a subset of nodes in the OLS during the local verification stage, as at least one of them has previously signed C_m and will refuse to cooperate.

Note that this scheme does not prevent from B modifying m into m', changing the identifier I_m into I'_m, and subsequently executing another protocol instance with the aim of obtaining a different content certificate, C'_m. This kind of manipulation cannot be avoided exclusively by external means, but by inserting appropriate mechanisms *inside* m.

4.2 Local Verification

This phase can be attacked in a number of ways. *Assumption 5* assures that messages exchanged among trusted peers cannot be spoofed and man-in-the-middle attacks cannot be mounted, since they are signed by authenticated public keys. However, Pathak and Iftode's protocol can fail due to the impossibility of getting an honest majority. In this case, an adversary could convince a honest peer that the public key of a node is K_i when it is not. This kind of man-in-the-middle attack is detected if at least one peer (apart from A) is honest.

4.3 Incorrect Sending and Non-cooperation

A dishonest signer can delay the content authentication by not signing the certificate, not sending any message to the next node in the OLS, and/or not notifying anything to the originator A. Surely, this behavior will produce a kind of DoS attack [11], so A must check nodes' availability when this situation appears.

Each intermediate peer must ignore any messages that do not have the proper form of content and a signed certificate. Besides, peers know that the originator A is malicious if her signature is incorrect.

Furthermore, a dishonest signer could send its participation randomly or maliciously, instead of a properly constructed signature. Even though this fact can be trivially detected, in a sense it is a point of failure, since cooperation is required among the $k+1$ nodes to achieve a successful execution of the protocol. In other words, the protocol allows to *detect* these forms of misbehavior and no cooperation, but it cannot enforce nodes to behave properly.

5 Conclusions and Future Work

Due to the very nature of P2P systems, it is usual that several copies of the same content exist at different network locations. Despite the advantages derived from replication, in general it is unrealistic to assume that every node will behave correctly. In this way, once a malicious node gets access to a content, its integrity can be compromised in several ways.

In this paper, we have proposed a content authentication protocol based on Byzantine agreement, especially oriented to pure P2P systems. This scheme allows for a secure content replication among peers, which is able to detect if non-authorized alterations have been made on the published contents. Furthermore, our proposal does not rely on the use of a trusted third party, an assumption that would be totally impractical for decentralized P2P environments.

Currently, we are working on statistically measuring how serious the problem of false content distribution is in real environments (e.g. eMule and BitTorrent). We are evaluating the level of boycott, mistrust, and bad reputation generated due to the distribution of forged contents. For this, we are studying several variables as: the download effort (time and bandwidth metrics), the user satisfaction degree, the content quality (especially relevant for multimedia contents), etc.

Future works will try to improve our protocol's efficiency, since it has not being our main objective in its conception. The large number of needed signers and the communication overhead could probably be improved with the use of other approaches, notably those based in multisignature schemes [1]. Additionally, we plan to include in C_m, as in most current digital certificates, two time parameters to specify when the certificate was generated and its expiration date. These fields, together with some additional information, could be added into the content certificate, for example as an agreed timestamp imposed from the source, and accepted and stored by each intermediate peer.

Finally, we are also studying the application of proof-of-work techniques for ensuring access control, for instance by using puzzles [5]. The use of Threshold Cryptography in P2P systems for reaching consensus is also an interesting research line that will be tackled in future works.

References

1. C. Boyd. "Digital multisignatures". In H. Baker and F. Piper (Eds.), *Cryptography and Coding*, pp. 241–246. Clarendon Press, 1989.
2. M. Conti, E. Gregori, and G. Turi. "Towards Scalable P2P Computing for Mobile Ad-Hoc Networks". In *Proceedings of the Second IEEE Annual Conference on Pervasive Computing and Communications Workshops (PERCOMW '04)*, Orlando, USA, pp. 109–113. March, 2004.

3. E. Damiani, S. De Capitani, S. Paraboschi, P. Samarati, and F.Violante. "A Reputation-Based Approach for Choosing Reliable Resources in Peer-to-Peer Networks". In *Proceedings of the 9th ACM Conference on Computer and Communications Security (CCS'02)*, Washington, USA, pp. 207–216. November, 2002.
4. N. Daswani, H. Garcia-Molina and B. Yang. "Open Problems in Data-sharing Peer-to-peer Systems". In *Proceedings of 9th International Conference on Database Theory*, Italy. January, 2003.
5. D. Dean and A. Stubblefield. "Using client puzzles to protect TLS". In *Proceedings of the 10th USENIX Security Symposium.* August, 2001.
6. J. Douceur. "The Sybil attack". In *Proceedings of 1st International Workshop on Peer-to-Peer Systems (IPTPS02) Workshop*, Cambridge, USA, pp. 251–260. March, 2002.
7. G. Fox. "Peer-to-Peer Networks". In *Computing in Science & Engineering, vol. 3, no. 3.* May, 2001.
8. A. Juels and J. Brainard. "Client puzzles: A cryptographic countermeasure against connection depletion attacks". In *Proceedings of the Network and Distributed Security Systems Symposium*, California , USA, pp. 151–165. February, 1999.
9. L. Lamport, R. Shostak and M. Pease. "The Byzantine General Problem". *ACM Transactions on Programming Languages and Systems, Vol. 4, No. 3*, pp. 382–401. July, 1982.
10. W. K. Lin, D. M. Chiu, Y. B. Lee. "Erasure Code Replication Revisited". In *Proceeding of the 4th IEEE International Conference on Peer-to-Peer Computing.* August, 2004.
11. P. Maniatis, T.J. Giuli, M. Roussopoulos, D.S.H. Rosenthal, and M. Baker. "Impeding Attrition Attacks in P2P Systems". In *Proceedings of the 11th ACM SIGOPS European Workshop*, Leuven, Belgium. September, 2004.
12. M. Oguchi, Y. Nakatsuka and C. Tomizawa. "A Proposal of User Authentication and a Content Distribution Mechanism using P2P Connection over a Mobile Ad Hoc Network". In *Proceedings of the IASTED International Conference on Communication Systems and Networks*, Marbella, Spain, pp. 65-69. September, 2004.
13. A. Oram, ed.: "Peer-to-Peer: Harnessing the Benefits of a Disruptive Technology". O'Reilly, Sebastopol (CA), 2001.
14. V. Pathak and L. Iftode. "Byzantine Fault Tolerant Public Key Authentication in Peer-to-Peer Systems". *Computer Networks*, Vol. 50, No. 4, pp. 579–596, March 2006. Elsevier.