Towards an Authentication Service for Peer-to-Peer based Massively Multiuser Virtual Environments

Arno Wacker*, Gregor Schiele†, Sebastian Schuster*, and Torben Weis*

*University of Duisburg-Essen
Duisburg, Germany
{arno.wacker|sebastian.schuster|torben.weis}@uni-due.de

†University of Mannheim
Mannheim, Germany
gregor.schiele@uni-mannheim.de

***Abstract*— In this paper we propose a distributed authentication service for peer-to-peer (P2P) based massively multiuser virtual environments. Such a service is necessary to provide security, e.g. preventing a user's account being stolen or the user being impersonated. We describe two variants of our authentication service. The first uses certificates and a central certification authority to ensure the validity of user-generated public keys. These keys are then used to sign messages sent by the users' peers. The second variant distributes the users' public keys in the P2P network and uses quorums to verify them.**

I. Introduction

Massively multiuser virtual environments (MMVEs) allow a large number of users to participate in a shared virtual environment via the Internet. Security is a crucial requirement for such systems, to guarantee their smooth operation [1]. Otherwise, users may e.g. pose as somebody else or steal other users' data. This is especially true since the participants of a large scale system typically do not know each other and therefore cannot trust each other.

To provide security, we first must guarantee message authenticity, i.e. a user must be able to identify the sender of a message reliably. Building upon this, other goals can be realized, e.g. data confidentiality or integrity.

In this paper we discuss how to provide message authenticity in MMVEs. We restrict our discussion to peer-to-peer (P2P) based systems. In such systems, the virtual environment is provided and managed by the participating users' computers themselves, instead of a centralized server, as in a client/server-based MMVE.

Our approach is based on certificates and signed messages. We depict two variants. First we discuss a straight forward approach using a Certification Authority (CA). This approach builds on well known techniques. Secondly, we propose a novel approach that is based on replicating public keys in the P2P network. This approach is still work in progress. However, in our opinion it offers a lot of potential for future research activities.

The paper is structured as follows. First, we present our system model and introduce two types of MMVEs, which need to be distinguished when developing an authentication service. After that we discuss requirements for our authentication service. We then provide an overview of related work and present our approach. Finally, we offer a short conclusion and some thoughts about future work.

II. System Model

We define an MMVE as a persistent virtual environment that is shared by a large number of users worldwide. The number of users is a priori undetermined and may change dynamically. A P2P-based MMVE is a special kind of MMVE that is executed cooperatively by all users' computing devices, called peers. Each device is connected to a common communication network, e.g., the Internet. Using this network, the peers form a connected overlay network, the so-called P2P (overlay) network. To participate in the MMVE, a user activates his device and starts the preinstalled MMVE software. The software logs into the P2P network and the user can start operating in the MMVE. Each user is represented in the MMVE by a special character, called his avatar. Conceptually, our approach is able to handle multiple avatars per user. However, in this paper we restrict each user to one avatar and use both terms interchangeably. This makes the description of our approach easier to understand. To operate in the MMVE, the user directs his avatar to perform actions for him, e.g. moving or interacting with other users' avatars. Each activity is distributed to other peers and processed by them, e.g. by updating the state of the MMVE. After the user is done, he stops the software and deactivates the device.

Our approach assumes the existence of an underlying structured P2P network that can be used to store and retrieve data, e.g. a distributed hash table (DHT). The P2P network must support four operations, discussed below. The operation `storeObject(pos, data)` stores a data item at a given (logical) position in the P2P network. Using the position, the P2P network determines a set of peers and stores the data item at them. The specific algorithm to do so depends on the used P2P network. Different positions are mapped to different peer sets, if possible, to distribute the data evenly. Due to peer fluctuations, the peers responsible for a given position may change over time. We assume that the P2P network detects this and relocates the involved data items automatically. The reason for using a replicated storage is to ensure that data items are persistent. Otherwise, when a peer is lost unexpectedly, data may be lost. We assume that the

P2P network contains consistency between all replicates of a given data item. With `retrieveObject(pos)` we can retrieve a data item from the P2P network. The parameter `pos` specifies the logical position of this item in the P2P network. The network automatically resolves the position to a set of peers and retrieves the data item from them. To delete a data item stored at a given logical position, the operation `deleteObject(pos)` can be used. It determines all peers saving the data item and instructs them to delete it. These three operations realize a simple distributed data space. In addition, the P2P network allows peers to send data messages to each other, using the `send(peer, data)` operation. It sends a given data item reliably to the specified peer. In the peers@play project we are developing a P2P network that fulfills all properties discussed above. This network can be used to realize our approach. Its exact implementation is beyond the scope of this paper.

III. Open vs. Moderated MMVEs

We distinguish two types of MMVEs, moderated and open ones. A moderated MMVE is operated by a specific system operator. The operator is responsible for managing the MMVE and its participants. As an example, the operator decides which user may participate in the MMVE. To do so, users are normally expected to register with the operator before entering the MMVE. The operator may also decide to remove a user from the MMVE, e.g. in case of the user violating the license agreement. Clearly, all users have to trust the operator. A typical example for this type of MMVE is a commercial system, e.g. an online game.

An open MMVE works without a specific operator, i.e. the system is operated cooperatively by its participants. There is no single entity responsible for the system or trusted by all users. Access to the system is free for all. A typical example for such an MMVE is a non-commercial online social network.

IV. Requirements

In the following section we analyze the requirements that an authentication service (AS) for P2P-based MMVEs must fulfill.

1) Decentralized operation: The first requirement for authentication in P2P-based MMVEs is decentralization. The AS must allow users to login into the MMVE and to operate within it without accessing a centralized server. This server could become a bottleneck for the system performance and a single point of failure. In addition, someone would have to operate the server, making this approach unsuitable for open MMVEs.

2) Privacy: With respect to privacy, the authentication must make sure that other users of the MMVE are not able to derive knowledge about the identity of other users. Note that an MMVE developer/operator may decide to make the identity of its users available to others. This is an MMVE-specific design decision and does not influence the AS requirements.

3) Availability: Clearly, authentication is a crucial service for most MMVE. If the AS is not available, no user can log into the MMVE. This may lead to users getting frustrated with the MMVE and eventually abandoning it and must therefore be avoided.

V. Related Work

Security has been widely recognized as a major concern in MMVEs. However, most researchers concentrate on designing cheat-proof algorithms, e.g. [2], [3]. All of these approaches make heavy use of cryptographic functions. They implicitly assume a public key infrastructure to be present delivering means to authenticate the a priori unknown communication partners. For example Rieche et al. [4] explicitly state that they intend to use an existing server for accounting in their P2P MMVE infrastructure.

Fully distributed key management infrastructures can be found in the area of P2P and ad-hoc networks. Threshold cryptography [5] is a way to realize such a distributed CA. Out of a group of n nodes, at least k are necessary to authenticate other nodes. It shares some similarities with our approach basing decisions on trustworthiness of multiple nodes. However, there is no guarantee that k out of the n nodes are present at a specific time. Furthermore, it is not clear how to choose n and k and how to form subgroups. Choosing n as all nodes of the network to fully distribute the CA is unfeasible, due to the huge network size and its dynamic change at runtime.

Another approach to distribute authentication in an ad-hoc network is forming a web of trust [6], similar to PGP. Here, no single trusted authority exists. Instead, everybody can issue certificates showing that he believes in the identity of someone else. Thus, trust chains can be built to verify identities. Metrics like counting trust chains can be used to further strengthen the belief in one's identity. However there is no guarantees that a trust chain exists at all. At the same time, the verification of a nodes identity is mixed with its ability to authenticate other nodes. Consequently, misbehaving nodes can hurt the whole system by issuing lots of false certificates.

Finally, multiple approaches in P2P systems are based on reputation measuring the trustworthiness of nodes e.g. [7]. Reputation of a node changes according to other nodes' experiences with that node. This is a distributed voting mechanism, and could be used to build up trusted peer-based CAs. However, it relies on authentication of a node's identity and binding its identity to its reputation in the first place.

VI. Authentication in Moderated MMVEs

After discussing our requirements, we now present our approach for authentication in MMVEs. We start with our first variant, which is tailored towards moderated MMVEs. In Section VII we propose another variant for open MMVEs.

Our first approach is based on three parts: (1) peers signing their messages, (2) certificates to validate signatures, and (3) the MMVE operator issuing certificates to users at registration time.

Before any user is able to access the MMVE, the MMVE operator sets up a CA that is accessible for all potential users, e.g. using the WWW. A CA is a trusted system service for generating certificates. The operator can use any existing CA software for this. He creates an asymmetric key pair for the CA, i.e. a public (K^+_{CA}) and a private key (K^-_{CA}). The public key is then embedded into the MMVE software and distributed with it. Thus, the CA's public key is well known to every peer in the system. When a user wants to register for the MMVE, he contacts the operator's CA and registers himself for the MMVE. During this registration, the user creates an MMVE identifier (ID_U) and generates an asymmetric key pair (K^+_U, K^-_U) for it. The identifier can be any arbitrary and sufficiently long string or number, which cannot be correlated with the real identity of the user. One way to create it is to hash the email address of the user with a secure hash function, i.e. $ID_U = Hash(user@domain.com)$. Other users in the P2P network will only see ID_U and never the real identity of the user. Hence, with this step the privacy requirement is fulfilled.

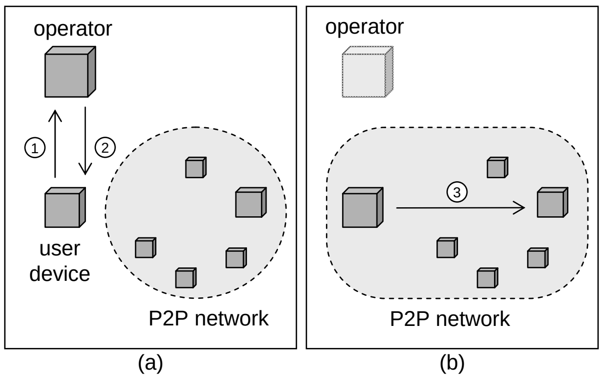

Fig. 1. Authentication in a moderated P2P-based MMVE

After the generation of these values the user finishes his registration by sending a signature request to the CA, i.e. message (1) in Fig. 1(a):

$$Message1 : U \rightarrow CA : \{ID_U, K^+_U\}_{K^+_{CA}} \quad (1)$$

Clearly, the system operator (i.e. his CA) can choose any appropriate means to identify the user. If the system operator needs to guarantee the identity of the user, a valid external certificate on the email address of the user could be used. When the user fulfills the registration requirements (e.g. payment was received) the operator issues a certificate for the new MMVE identity, i.e. message (2) in Fig. 1(a):

$$Message2 : CA \rightarrow U : \{ID_U, K^+_U, t_s, t_v\}_{sign(CA)} \quad (2)$$

where $\{m\}_{sign(X)} = m|\{Hash(m)\}_{K^-_X}$ for any message m. This signed message includes the MMVE identity (ID_U), the corresponding public key (K^+_U) and a period of validity given with the two time marks t_s and t_v. Clearly, the CA could include any additional data in the certificate – e.g. a serial number – if this is required by the MMVE.

After that, the user can log into the MMVE. To do so, no additional communication is needed. Instead, the user's peer signs all its messages with the user's private key, i.e. message (3) in Fig. 1(b):

$$Message3 : U \rightarrow peer : \{m\}_{sign(U)} \quad (3)$$

Upon receiving such a message, each peer checks if it knows the sender's certificate. Otherwise, it requests the certificate at the sending peer. Once the receiver knows the certificate, it validates the message signature using the sender's public key. If the signature is valid, the peer accepts the message. Otherwise the message is simply ignored, denying that peer the participation in the network.

Note that this approach resembles a classical certificate-based approach very closely. The main differences are that the MMVE operator is used to provide the CA instead of an external CA, e.g. VeriSign. In addition, each certificate is also a ticket, i.e. it allows the owner of the certificate to enter the network and participate in the MMVE. Our certificates have a restricted lifetime, similar to most other certification-based approaches. The duration of the certificate lifetime depends on the provider's payment model. If users have to pay to enter the MMVE, e.g. using a monthly fee, the certificate should be valid for the paid duration. After that, a new certificate must be issued. Clearly, this should be transparent for the user, i.e. the certification renewal must be done automatically.

In certain cases it may be necessary to remove a user from the MMVE while his certificate is still valid. This may be the case if the user violates the usage terms or if his account is stolen. To remove a user, his certificate is revoked. A possible approach for revocation is to let each peer check with the CA if the certificate is still valid. However, this would put additional load on the CA and would require it to be available to perform the check. Therefore we propose the usage of *revocation list messages*, containing the current revocation list. The system operator creates this list and signs it with his private key. Then the list is send to the P2P network e.g. via (authenticated) flooding. Each (non-compromised) peer needs to store the revocation list locally. To identify updates a simple version counter could be used. With this kind of revocation list we just need to ensure, that the list is never lost, i.e. it needs to be replicated consistently. This is done automatically by our assumed underlying P2P network (see Section II).

This approach fulfills the requirements given in Section IV. It is decentralized since peers can authenticate themselves against other peers at runtime without contacting a centralized system component. To do so, they only have to sign their messages. The CA is only needed at registration time to create a new certificate. If the CA fails, no new users can register for the MMVE. However, already registered users can keep logging into the MMVE and use it. As long as two peers are able to communicate with each other, they can authenticate against each other. Thus, the approach does also fulfill our third requirement, availability. Finally, privacy is provided by using a secure hash function to create the user's identifier ID_U. For authentication, only ID_U is send to other peers and only the MMVE provider can link ID_U to the user's identity.

VII. Authentication in Open MMVEs

Our approach for authentication in moderated MMVEs is based on the system operator providing a trusted CA to validate public keys. In an open MMVE, no system operator exists. Therefore, another approach is needed. An easy modification is to use an external globally available CA, e.g. VeriSign, to provide the needed certificates. In the future, many users may have such a certificate anyway, e.g. for electronic commerce. However, such CAs usually include the user's identity into the certificate. This violates our privacy requirement. In addition, all users would need a certificate issued by the external CA, resulting in addition costs.

We omit using a central CA and propose a different approach to validate public keys. The basic idea is to adapt the registration process such that instead of a certificate being created, the public key is stored at a number of different peers in the P2P network.

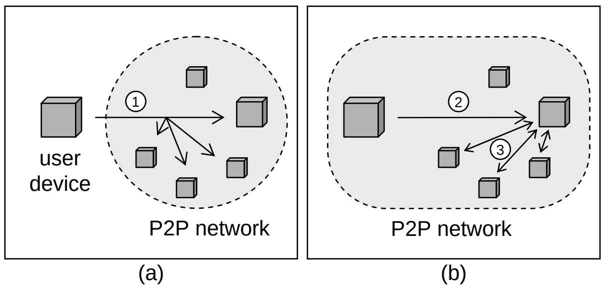

Fig. 2. Authentication in an open P2P-based MMVE

Similar to the registration in moderated MMVEs, a new user first has to create an MMVE identifier ID_U and generate an asymmetric key pair (K_U^+, K_U^-). The public key K_U^+ must be stored in a way such that (1) it cannot be tampered with and (2) all other peers can retrieve it when needed. We cannot assume that any single peer is secure. Therefore, we propose to store K_U^+ on multiple peers in the P2P network. When we later want to retrieve the public key, we use a majority voting mechanism to identify modified entries. We introduce the security level s to denote the number of manipulated entries that are tolerated by our approach. To tolerate s such manipulated entries, we need to store the data on at least $(2s+1)$ different peers.

An overview of this approach is given in Fig. 2. Initially, the new peer stores its user's public key at $(2s+1)$ different positions (see (1) in Fig. 2(a)). This registers the user in the MMVE. After that, the user can log into the MMVE and operate in it. Whenever his peer has to send a message to another peer, it signs the message with the user's private key (see (2) in Fig. 2(b)). To check the signature, the receiver contacts the P2P network and retrieves the corresponding public key from all $(2s+1)$ different positions in the network (see (3) in Fig. 2(b)).

In the following we describe our approach in more detail and provide some pseudo code for it.

A. Registration

The algorithm for registering a new user is given in Algorithm 1. It must be executed before the user logs into the MMVE for the first time.

Algorithm 1 Registering in open MMVEs

```
1: $ID_U = Hash(user@domain.com)$;
2: ($K_U^+$, $K_U^-$) = new Key(); // Generate a new key-pair
3: for $i = 1$ to $(2s+1)$ do
4:     $pos_i = Hash(ID_U|i)$; // Calculate DHT position
5:     storeObject($pos_i$,($ID_U, K_U^+$));
6: end for
```

Lines 1–2 show the creation of a new MMVE identity and the generation of a new asymmetric key pair. After that, the for-loop (see Line 3–6) stores the public key at $(2s+1)$ different positions in the P2P network. Each individual position is calculated by hashing the MMVE identity ID_U concatenated with a unique number (see Line 4). After that, we store the public key K_U^+ and the corresponding ID_U using the operation `storeObject` (Line 5).

Note that if the P2P network stores different positions pos_i on the same peer, the security of our approach decreases. By compromising this peer, an attacker can gain control over multiple copies of the public key. In our system model, we assumed the P2P network to distribute positions evenly on the available peers. Therefore, the probability for such a situation to occur decreases with the number of peers in the network.

B. Key Retrieval

When a peer sends a signed message to another peer , the receiver has to check the message signature. To do so, it first checks if it already knows the sender's public key. If not, the receiver executes Algorithm 2.

Algorithm 2 Retrieving a public key

```
1: // Retrieves the public key for a given ID
2: items[] = new array[$2s+1$];
3: for $i = 1$ to $(2s+1)$ do
4:     $pos_i = Hash(ID_U|i)$; // Calculate DHT position
5:     $items[i]$ = retrieveObject($pos_i$);
6: end for
7: $item$ = majority($s$,$items$);
8: return $item.K_U^+$;
```

Since we cannot rely on a single peer to provide the valid public key of a user, we have to collect the public key from all $(2s+1)$ positions and perform a majority voting. With the for-loop (Line 3–6) we retrieve each copy of the previously stored public key and store it in the local array $items[]$ (Line 5). In case there are different answers, a simple majority is used (Line 7), i.e. at least s keys need to be equal in order to return the public key (Line 8). If no majority of s can be achieved, we cannot decide which public key is correct. In this case, the received message is discarded.

C. Key Updates

In certain cases, a user might want to update his public key, e.g. because he suspects that it might be compromised. In addition, the MMVE might use relatively short keys to achieve higher encryption efficiency. In this case, the keys should be exchanged regularly. Finally, if the user decides that he will not use the MMVE anymore, he should be able to remove his registration from the P2P network. To update a registered key, Algorithm 3 can be used.

Algorithm 3 Updating a public key item

```
1: // Received a message $m = \{ID_U, K'^{+}_U\}_{sign(U)}$
2: // $K'^{+}_U$: new (updated) public key
3: // To be stored at position $pos$
4: // Note: $\{m\}_{sign(X)} = m|\{Hash(m)\}_{K^{-}_X}$
5: // Note: $m.sig(X) = \{Hash(m)\}_{K^{-}_X}$
6: if localStore[$pos$] != null) then
7:     $kpub$ = localStore[$pos$].$K^{+}_U$;
8:     if $(Hash(ID_U, K'^{+}_U) == \{m.sig(U)\}_{kpub})$ then
9:         if $K'^{+}_U$ != null then
10:            localStore[$pos$] = $(ID_U, K'^{+}_U)$;
11:        else
12:            localStore[$pos$] = null;
13:        end if
14:    end if
15: end if
```

When the user wants to update his public key, he prepares an update message and sends it to all peers holding a copy of his public key. These peers are determined similarly to the original registration (see Algorithm 1). An update message contains the user's MMVE identifier ID_U and the new public key K^{+}_U. It is signed using the old public key. Upon receiving an update message, each peer first checks if it has an entry at the specified position (Line 6). If this is the case, the stored public key is used to validate the signature of the message (Line 8). If the signature is valid, the peer updates the entry (Line 10). Otherwise, the update is denied. To allow deleting entries, the peer checks if the provided new key equals null (Line 9). In that case, instead of updating the key, the peer removes the entry from its local storage (Line 12).

Note that while this algorithm is executed, peers trying to retrieve the public key may not be able to constitute a valid quorum, if some copies are compromised. This is resolved once the algorithm terminates for all copies.

D. Discussion

Our approach for authentication in open MMVEs fulfills all three requirements. It is completely decentralized, both at registration time and during the MMVE execution. Similar to our approach for moderated MMVEs, privacy is provided by using only hashed user identities. Hence, the identity of other users is kept hidden. Since the public key is stored in the P2P network, it is always accessible. Thus, the AS is always available, fulfilling our third and last requirement.

For our approach to work, a peer must be connected to at least $(2s + 1)$ other peers in the P2P network. Otherwise, during the first connection of a new user, the neighboring peers could cheat about its identity, i.e. mount a so called man-in-the-middle attack. One way to guarantee this, is to organize the P2P network such that it forms a $(2s+1)$-connected graph. In such a graph, there are always $(2s + 1)$ paths between each peer, making attacks impossible if less than $(s + 1)$ peers are compromised. We already applied this approach successfully in wireless sensor networks [8] and are planning to transfer these results to MMVEs.

VIII. Conclusion and Future Work

In this paper we proposed an AS for P2P-based MMVEs. We provided two variants. The first is tailored towards moderated MMVEs. It relies on the MMVE operator offering a CA to issue certificates to users of the MMVE. At runtime, messages are signed with the users' public keys and validated using their certificates. This approach is relatively straight forward and can be applied with little effort. However, it is not applicable to open MMVEs, since they do not have an operator. Our second variant is able to operate without an operator. Instead of a CA issuing certificates, public keys are stored redundantly in the P2P network and checked using quorums. This approach is able to tolerate up to s compromised copies of the stored public key. If an attacker is able to gain control over more copies, he can modify the public key and impersonate another user. Therefore, the MMVE must use a sufficiently high security level s.

Clearly, the usage of asymmetric cryptography on each message is very resource intensive. The straight forward improvement is to establish a symmetric session key at the beginning of the communication. This key can then be used for creating message authentication codes for each message. With the same method also confidentiality can be achieved.

At the moment we are developing a prototypical implementation of our approach for moderated MMVEs. Our approach for open MMVEs is still work in progress and must be refined and analyzed in more detail with respect to different possible attacks. As an example, until now we assumed that the P2P network is able to relocate data items securely between peers, when the responsibility for a given logical position in the network changes. We plan to analyze this assumption and investigate adequate mechanisms to achieve it.

References

[1] G. Schiele, R. Sueselbeck, A. Wacker, J. Haehner, C. Becker, and T. Weis, "Requirements of peer-to-peer-based massively multiplayer online gaming," in *Proceedings of the Seventh International Workshop on Global and Peer-to-Peer Computing*, 2007.

[2] C. GauthierDickey, D. Zappala, V. Lo, and J. Marr, "Low latency and cheat-proof event ordering for peer-to-peer games," in *NOSSDAV '04: Proceedings of the 14th international workshop on Network and operating systems support for digital audio and video*. New York, NY, USA: ACM, 2004, pp. 134–139.

[3] A. B. Corman, S. Douglas, P. Schachte, and V. Teague, "A secure event agreement (SEA) protocol for peer-to-peer games," in *The First International Conference on Availability, Reliability and Security (ARES)*. IEEE Computer Society, 2006, pp. 34–41.

[4] S. Rieche, K. Wehrle, M. Fouquet, H. Niedermayer, L. Petrak, and G. Carle, “Peer-to-peer-based infrastructure support for massively multiplayer online games,” in *4th Annual IEEE Consumer Communications and Networking Conference (CCNC 2007)*, Las Vegas, Jan. 2007.

[5] Y. Desmedt and Y. Frankel, “Threshold cryptosystems,” in *CRYPTO ’89: Proceedings of the 9th Annual International Cryptology Conference on Advances in Cryptology*. London, UK: Springer-Verlag, 1990, pp. 307–315.

[6] J.-P. Hubaux, L. Buttyn, and S. Capkun, “The quest for security in mobile ad hoc networks,” in *Proceeding of the ACM Symposium on Mobile Ad Hoc Networking and Computing (MobiHOC)*, 2001.

[7] A. Singh and L. Liu, “Trustme: Anonymous management of trust relationships in decentralized P2P systems,” in *Peer-to-Peer Computing*, N. Shahmehri, R. L. Graham, and G. Caronni, Eds. IEEE Computer Society, 2003, pp. 142–149.

[8] A. Wacker, T. Heiber, and H. Cermann, “A key-distribution scheme for wireless home automation networks,” in *Proceedings of IEEE CCNC 2004*, IEEE Communications Society. Las Vegas, Nevada, USA: IEEE, January, 5-8 2004.