

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

**Arno Wacker¹, Gregor Schiele²,
Sebastian Schuster¹, and Torben Weis¹**

¹University of Duisburg-Essen, Duisburg, Germany
{arno.wacker|sebastian.schuster|torben.weis}@uni-due.de

²University of Mannheim, Mannheim, Germany
gregor.schiele@uni-mannheim.de

1 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, as we have shown in Schiele et al. (2007). Otherwise, users may 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 describe two variants. First, we discuss a straight forward approach using a Certification Authority (CA). This approach builds on well known techniques. Second, 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.

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. We

analyse the security of our approach and describe our prototypical implementation. Finally, we offer a short conclusion and some thoughts about future work.

2 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.

2.1 Peer-to-Peer Network

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) as implemented in Chord by Stoica et al. (2001) or in Pastry by Rowstron & Druschel (2001). The P2P network must support four operations, discussed below. The operation `storeObject(pos, data)` (often called `put(key, object)` in the context of DHTs) 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 underlying P2P network. Different positions are often mapped to different peer sets, if possible, to achieve load balancing. 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 will be lost. We assume that the P2P network contains consistency between all replicates of a given data item. With `retrieveObject(pos)` (often called `get(key)` in the context of DHTs) 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 fulfils all properties discussed above. This network can be used to realize our approach. Its exact implementation is beyond the scope of this paper. For the rest of this paper we assume its existence.

We argue that these assumptions are sound because the P2P network must offer these properties to enable the state of the MMVE to be stored persistently. Thus, we only reuse already existing functionality.

2.2 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.

3 Requirements

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

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. Such a server would represent a single point of failure making the MMVE inaccessible during its downtime. In addition, someone would have to operate the server, making this approach unsuitable for open MMVEs.

Privacy With respect to privacy, the authentication must assure 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.

Availability Clearly, the availability of the authentication service is a crucial requirement for most MMVEs. 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.

4 Related Work

Security has been widely recognized as a major concern in MMVEs. However, most researchers focus on designing cheat-proof algorithms, e.g. GauthierDickey et al. (2004), Corman et al. (2006). All of these approaches make heavy use of cryptographic functions. They implicitly assume a public key infrastructure is available to authenticate the a priori unknown communication partners. For example Rieche et al. (2007) explicitly state that they intend to use an existing server for accounting in their P2P MMVE infrastructure, corresponding to a moderated approach.

Fully distributed key management infrastructures can be found in the area of P2P and ad-hoc networks. Threshold cryptography, as described in Desmedt & Frankel (1990), 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 where decisions are based 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 infeasible, due to the huge network size and dynamic changes at runtime.

Another approach to distribute authentication in an ad-hoc network is to form a web of trust as shown in Hubaux et al. (2001). This approach is similar to PGP, where 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 guarantee 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, some approaches in P2P systems are based on reputation measuring the trustworthiness of nodes e.g. Singh & Liu (2003). The 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.

5 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 6 we propose another variant for open MMVEs.

Our first approach consists of 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 by all potential users. 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 registers 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.

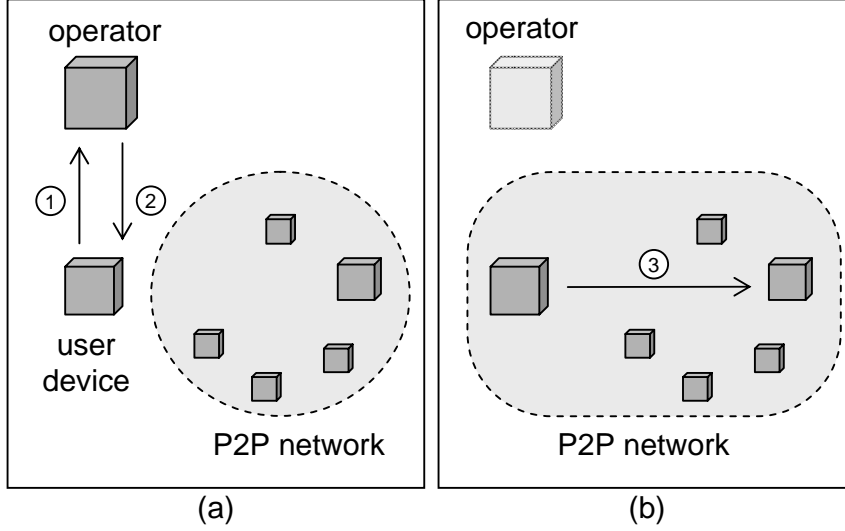


Figure 1 Authentication in a moderated P2P-based MMVE

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):

$$(1) \quad M1 : U \rightarrow CA : \{ID_U, K_U^+\}_{K_{CA}^+}$$

Without further precautions, message M1 could be created by anyone and an additional mechanism is needed to make sure that it was indeed created by the correct user. To guarantee this during registration is the responsibility of the system operator (i.e. his CA). He can choose any appropriate means to do so, e.g. by using external certificates for users and letting them sign the message. As this is only necessary during registration with the operator, it does not violate our privacy requirement. When the user fulfils 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):

$$(2) \quad M2 : CA \rightarrow U : \{ID_U, K_U^+, t_s, t_v\}_{sign(CA)}$$

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 timestamps 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):

$$(3) \quad M3 : U \rightarrow peer : \{m\}_{sign(U)}$$

Upon receiving such a message, each peer checks if it knows the sender's certificate. Otherwise, it requests the certificate from 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 and the peer is denied participation in the network. This approach guarantees message authenticity, i.e. a peer receiving a message can identify the original creator of the message. Further security goals, e.g. security against replay (so-called entity authentication) and message confidentiality, can be achieved using standard security mechanisms like encryption, counters, timestamps, sessions, in addition to our approach. An example on how to do this is given in Section 7.2.

Note that our 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 sent 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 2).

This approach fulfils the requirements given in Section 3. 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 fulfil our third requirement of 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 sent to other peers and only the MMVE provider can link ID_U to the user's identity.

6 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, a different 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 additional costs.

We avoid 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.

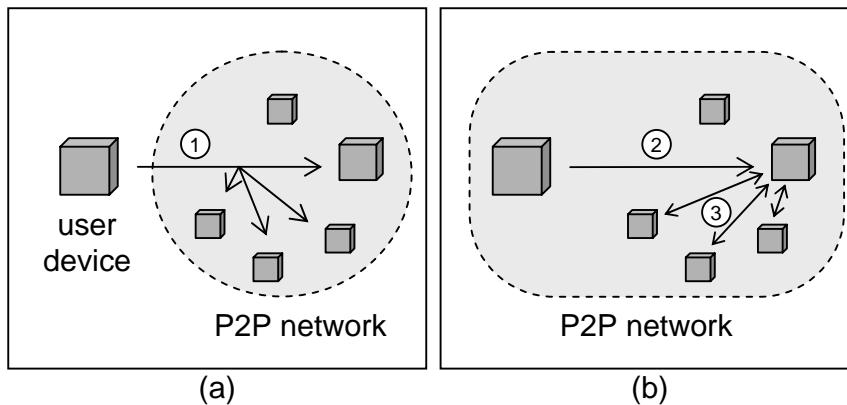


Figure 2 Authentication in an open P2P-based MMVE

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 sections we describe our approach in more detail and provide some pseudo code for it.

6.1 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:  $ID_U = Hash(user@domain.com)$ ; // Generate new MMVE ID
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:    $storeObject(pos_i, (ID_U, K_U^+))$ ; // store it
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 assume that the P2P network to distribute positions evenly among the available peers. Therefore, the probability for such a situation decreases with the number of peers in the network.

As discussed in our system model (see Section 2), once a key is stored at a position, we assume the P2P network to guarantee that this entry is never lost, regardless of network churn. When a node containing entries leaves the network, the P2P network automatically relocates the entries to another peer.

6.2 Key Retrieval

When a peer sends a signed message to another peer (see Equ. 1), 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.

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.

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:    $items[i] = retrieveObject(pos_i)$ ; // get it
6: end for
7:  $item = majority(s, items)$ ;
8: return  $item.K_U^+$ ;

```

6.3 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:  $\{m\}_{sign(X)} = m \parallel \{Hash(m)\}_{K_X^-}$ 
5: // Note:  $m.sig(X) = \{Hash(m)\}_{K_X^-}$ 
6: if  $localStore[pos] \neq null$  then
7:    $k_{pub} = localStore[pos].K_U^+$ ;
8:   if  $(Hash(ID_U, K_U^+) == \{m.sig(U)\}_{k_{pub}})$  then
9:     if  $K_U^+ \neq null$  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.

6.4 Discussion

Our approach for authentication in open MMVEs fulfils 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. The handling of peer fluctuations and the resulting secure transfer of stored keys between peers is the responsibility of the underlying P2P network. Thus, the AS is always available, fulfilling our third and last requirement.

Our approach is independent of the number of peers in the P2P network. Instead, it depends on the security level s . To first register and later validate a public key in the P2P network a quorum of $s + 1$ peers is used. The number of accesses to the DHT needed for this, grows linearly with s . However, s is independent of the total number of peers present in the system. It is set according to the actual security requirements of the MMVE.

For our approach to work, a peer must be able to contact directly 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, see Wacker et al. (2004), and are planning to transfer these results to MMVEs.

7 Evaluation

In this section we provide a brief security analysis of our proposed approaches. After that we present an overview of our current prototypical implementation.

7.1 Security Analysis

In our security analysis, we describe possible attacks against our approaches and discuss whether these attacks can be successful or not. The main goal of an attacker against an authentication service is to impersonate other users. If the attacker is able to do so, he can send messages on behalf of them. He may even be able to access the users' accounts. Our proposed approaches prevents a malicious user from impersonating others and taking over their accounts. In the following we first discuss attacks against a moderated MMVE. After that we discuss attacks against open MMVEs. In general, an attacker can try to intercept messages, drop or modify them, create new messages or replay previously intercepted messages.

For a moderated MMVE, a user E trying to impersonate another – already registered – user A , would need to sign her messages with the correct private key of A (M3). Assuming that the private key stays secret, this is not possible. Thus, the attacker cannot create forged messages or modify existing ones. However, the attacker could intercept messages and either drop them or record and replay them later. Clearly, dropping messages could seriously interfere with the MMVE operation. However, it does not allow the attacker to impersonate another user. Our approach does not protect against such denial-of-service attacks. Replay attacks can be prevented by guaranteeing message freshness on top of message authenticity. In our prototypical implementation, as described in the next section, we realize this by replacing M3 with three messages performing a 3-way handshake between the two communication partners. With this handshake we additionally establish a symmetric session key for any further communication, thus optimizing the computational overhead. With this extension message replay is no longer possible and additionally any communication is also confidential. Another way for an attacker to impersonate another user is during registration. At registration time a MMVE ID is linked with a public key by the operator's signature. Clearly, if the attacker can register as another user our approach cannot work. Therefore, the operator must ensure (1) that there are no double registrations with the same ID and (2) identify the actual entity requesting the registration by any means necessary. Both additional requirements can be met by an operator and are outside the scope of our approach.

In an open MMVE we do not assume the presence of a trusted operator. Therefore, because of our privacy requirement, it is in general not possible to make sure that a given user identity is correct. To do so would require other peers to check the user's identity by some external means. Therefore, the primary goal of an authentication service in an open MMVE is to prevent user account theft. Once a user created a new account, it must not be possible for any other user to use this account as it's own. However, any user is able to create a new account, possibly using arbitrary personal information.

As for moderated MMVEs, to impersonate a user A an attacker E must sign her messages with the private key of A . Again this is only possible by knowing A 's private key, e.g. by compromising A 's computer, and beyond the scope of this work. As long as the private key stays secret, messages cannot be forged or modified.

The main difference between our approaches for moderated and open MMVEs is the fact that the public key of a correctly registered user A cannot be faked by any attacker, because the operator has signed this key. In open MMVEs, this is not the case. Instead, the public key is stored at $(2s + 1)$ different peers in a DHT and verified using a majority voting approach. If the attacker E can get access to more than s copies of A 's public key, E can exchange A 's public key with her own public key and therefore impersonate A . However, this is only possible if one of two cases happen: (1) E is able to get access to more than s peers which store the public key of A , or (2) multiple copies are stored at the same peer. The first case can be made arbitrarily hard by increasing the parameter s . In addition, the DHT must prevent the attacker to deliberately introduce new peers that will be chosen by the DHT to store a given key. To do so, the DHT can derive the IDs of its participating peers from their IP addresses and make sure that copies are stored on peers that are far away from each other (with respect to their IP addresses). Clearly, this does not

prevent an attacker with enough resources from setting up appropriate peers, i.e. mounting a Sybil attack as described by Douceur (2002). However, together with increasing s , it allows to make attacks increasingly costly. For the second case – multiple copies on one peer – we must first ensure that the used hash function of the DHT distributes all keys evenly among all peers. If this is the case, then the probability that multiple copies of A 's public key are stored on E 's peers decreases inversely with the number of available peers. In large networks with many peers, the probability is very small and can be neglected in practice. However, in small networks this is not the case. Hence we cannot guarantee anymore that A 's key cannot be tampered with. The only – trivial – solution to this problem is the requirement that at any time there must be much more than $(2s + 1)$ peers online. If the number of available peers falls below a given threshold, the users could be notified of the potential security risk.

Once a key for A is stored in the DHT, it can only be modified by A . It is not possible for E to overwrite it with a new key. This is the case since the peers that store the public key check if a key update message is valid by comparing its signature with the stored key.

While E cannot impersonate A , it is possible for E to launch a kind of denial-of-service attack by registering itself for as many as possible IDs. By doing so, A could not register anymore later because its ID is already used. Our current approach does not prevent this attack. In case a user's ID is derived from his email address (see Section 5), a possible approach for doing so is to accept a new registration only after additionally testing the new user's email.

Clearly, the proposed approach for open MMVEs also only offers message authentication – for a secure communication at least freshness, confidentiality and integrity need to be added after the authentication part. This can be achieved with a similar handshake as we proposed for our moderated approach and as described in the next section.

7.2 Prototypical Implementation

We developed a prototypical implementation of our approach for moderated MMVEs. We use RSA for asymmetric and AES for symmetric operations. For the message authentication codes we use HMAC based on SHA-1.

To reduce computational overhead on each peer, we use the approach described in Section 5 only for establishing a symmetric session key which is then used for further communication. In detail, whenever a peer A establishes communication with another peer B in our network, it first performs the registration part as per Equ. 1 and Equ. 2. After that, each peer is in possession of its own certificate (signed by the operator). To establish a secure and authenticated communication between two peers, a 3-way handshake is performed. First the initiating peer A sends a communication request message $M3b$ to peer B as shown in Equ. 4:

$$(4) \quad M3b : A \rightarrow B : \{\{ID_A, K_A^+, t_s, t_v\}_{sign(CA)}, r_A\}_{sign(A)}$$

The communication request contains the signed public key of peer A and a random number r_A . The random number is used to ensure the freshness of the message, preventing message replay. The receiving peer B can check the authenticity of the

sender by validating the message signature with the public key from the certificate sent. Clearly this cannot be forged, since it is signed with the operators private key. If the message is authenticated, peer B responds with a communication response message M4 in turn, signaling A that B is willing to communicate with it as shown in Equ. 5:

$$(5) M4 : B \rightarrow A : \{\{ID_B, K_B^+, t_s, t_v\}_{sign(CA)}, \{k_{AB}^E, k_{AB}^H, r_A + 1, r_B\}_{K_A^+}\}_{sign(B)}$$

When peer A receives this communication response from peer B it can validate B 's identity. The communication response includes two proposed session keys and some random number linking this message to the previous communication request. The session key k_{AB}^E is the symmetric key which is going to be used for communication encryption and the session key k_{AB}^H is going to be used for generating HMAC-based message authentication codes. The session key and the random number are encrypted with A 's public key, hence this part of the message can only be decrypted by peer A . With the aid of the first random number r_A any replay of message M3b or M4 can be detected by verifying $r_A + 1$. After that step both communicating peers can be sure that they really communicate with the intended communication partner and are therefore fully authenticated against each other.

To finally establish the new communication, peer A sends message M5 to peer B using the newly established keys as shown in Equ. 6:

$$(6) M5 : A \rightarrow B : \{r_B + 1\}_{k_{AB}^E}, HMAC_{k_{AB}^H}(r_B + 1)$$

Message M5 proves to B that A knows both session keys. To allow this, M5 contains the incremented random number r_B encrypted with k_{AB}^E , and the message authentication code $HMAC_{k_{AB}^H}(r_B + 1)$.

Now the session is established and can be used to exchange further data messages. Each data message is encrypted with the session key k_{AB}^E and contains a counter to prevent message replay. The random number r_B is used as the initial value for this counter. In addition each message is authenticated by attaching $HMAC_{k_{AB}^H}$.

The communication overhead of our approach for moderated MMVEs consists of three parts: first, the registration at the operator (M1,M2), secondly, the 3 handshake messages (M3b,M4,M5) per session establishment, and thirdly, the additional overhead for each data message that peer's exchange after session establishment. To encrypt data messages we use AES in CBC-mode. This may result in a small overhead because each message length is expanded to a multiple of the AES block size (128 bit). In addition, the attached message authentication code produces a communication overhead of 160 bits for each message.

Note that even though this approach is used in our prototypical implementation of the moderated case, a slightly adopted variant can be used for the open MMVE as well. In an open MMVE, message M3b can be omitted. In addition, M4 does not contain the certificate but only the encrypted proposed session key and the random number r_B . The certificates are not needed here since the validity of the public key is determined locally by requesting a quorum of other peers.

8 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 .

At the moment we are further refining our prototypical implementation of our approach for moderated MMVEs. As an example, our prototype so far does not include key revocation. This feature is currently being integrated. In addition, we will demonstrate our prototype at the P2P 2008, see Wacker et al. (2008).

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 develop techniques to achieve it.

References

- A. B. Corman, et al. (2006). 'A Secure Event Agreement (SEA) protocol for peer-to-peer games'. In *The First International Conference on Availability, Reliability and Security (ARES)*, pp. 34–41. IEEE Computer Society.
- Y. Desmedt & Y. Frankel (1990). 'Threshold Cryptosystems'. In *CRYPTO '89: Proceedings of the 9th Annual International Cryptology Conference on Advances in Cryptology*, pp. 307–315, London, UK. Springer-Verlag.
- J. R. Douceur (2002). 'The Sybil Attack'. In P. Druschel, M. F. Kaashoek, & A. I. T. Rowstron (eds.), *IPTPS '01: Revised Papers from the First International Workshop on Peer-to-Peer Systems*, vol. 2429 of *Lecture Notes in Computer Science*, pp. 251–260, London, UK. Springer.
- C. GauthierDickey, et al. (2004). '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*, pp. 134–139, New York, NY, USA. ACM.
- J.-P. Hubaux, et al. (2001). 'The Quest for Security in Mobile Ad Hoc Networks'. In *Proceeding of the ACM Symposium on Mobile Ad Hoc Networking and Computing (MobiHOC)*, pp. 146 – 155, Long Beach, CA, USA. ACM.

- S. Rieche, et al. (2007). ‘Peer-to-Peer-based Infrastructure Support for Massively Multiplayer Online Games’. In *4th Annual IEEE Consumer Communications and Networking Conference (CCNC 2007)*, pp. 763–767, Las Vegas. IEEE.
- A. Rowstron & P. Druschel (2001). ‘Pastry: Scalable, Decentralized Object Location, and Routing for Large-Scale Peer-to-Peer Systems’. *Lecture Notes in Computer Science* **2218**:329–350.
- G. Schiele, et al. (2007). ‘Requirements of Peer-to-Peer-based Massively Multiplayer Online Gaming’. In *Proceedings of the Seventh International Workshop on Global and Peer-to-Peer Computing*, pp. 773–782, Rio de Janeiro, Brazil. IEEE.
- A. Singh & L. Liu (2003). ‘TrustMe: Anonymous Management of Trust Relationships in Decentralized P2P Systems’. In N. Shahmehri, R. L. Graham, & G. Caronni (eds.), *Peer-to-Peer Computing*, pp. 142–149. IEEE Computer Society.
- I. Stoica, et al. (2001). ‘Chord: A Scalable Peer-To-Peer Lookup Service for Internet Applications’. In *Proceedings of the 2001 ACM SIGCOMM Conference*, pp. 149–160, San Diego, California, United States. ACM.
- A. Wacker, et al. (2004). ‘A Key-Distribution Scheme for Wireless Home Automation Networks’. In *Proceedings of IEEE CCNC 2004*, pp. 47–52, Las Vegas, Nevada, USA. IEEE Communications Society, IEEE.
- A. Wacker, et al. (2008). ‘A NAT Traversal Mechanism for Peer-To-Peer Networks’. In *P2P ’08: Proceedings of the Eighth IEEE International Conference on Peer-to-Peer Computing (P2P’08)*, pp. 81–83, Aachen, Germany. IEEE.

