Implementation of an Instant Messaging System with Focus on Protection of User Presence

Karsten Loesing, Maximilian Röglinger, Christian Wilms, and Guido Wirtz

University of Bamberg
Distributed and Mobile Systems Group
Feldkirchenstr. 21, 96047 Bamberg, GERMANY
{karsten.loesing, guido.wirtz}@wiai.uni-bamberg.de

***Abstract*— Instant Messaging (IM) systems provide its users with the information of which of their contacts are currently online. This presence information supplements text communication of IM systems and is an additional value compared to other synchronous communication media. Unauthorized users could illegally generate online logs of users by exploiting their presence information. Public IM systems lack the reliable means to protect user presence and force the user to trust in the central registry server.**

In this paper, we propose an IM system, which is explicitly designed to protect user presence without the need of a trusted central registry. We present a Java implementation based on the anonymous communication network Tor [1], the cryptographic suite Bouncy Castle [2], and the distributed hash table OpenDHT [3].

I. Motivation for protecting user presence

Instant Messaging (IM) systems enable their users not only to instantly send text messages, but also inform them of which of their contacts are currently online and available for interactive communication. This does not require explicit communication compared to other synchronous communication media e.g. telephone. Nardi et al. [4] observed in an ethnographic study the additional value of IM to negotiate the presence of contacts prior to interacting with them. They call this property "outeraction" which supplements the actual interaction with the communication partner.

The downside of providing presence information is its potential misuse. Presence information is inherently user-specific, and knowledge of it might reveal the user's habits, which should be private to a certain extent. Patil et al. [5] emphasized privacy issues when providing presence information to colleagues in collaborative working environments. We argue that these issues increase when considering not only authorized contacts, but also unauthorized users. User transparency allows unauthorized users to generate an online log, i.e. a history of logins, and evaluate personal behavior.

Although all well-known IM protocols (ICQ [6], AIM [7], and MSN [8]) allow limitation of presence information to authorized contacts, this property could be broken. These protocols rely on a central registry server, which authenticates users and assists in transmitting presence information and text messages between users. The registry and the connections to it could be the first point of attack for hackers. Apart from that, the central server could be untrustworthy and misuse the connection data from logged on users for its own personal analysis.

In this paper, we propose a cryptographically secure IM system to conceal user presence from unauthorized users and from the central registry. It relies on the anonymous communication network Tor [1] to conceal IP addresses of users, which could reveal a user's presence when observing IP communication now or in the future. The outcome of using our protocol is that users can be sure that only currently authorized contacts are aware of their presence without the need to trust a third party.

Currently, there are no comparable approaches to build a system which addresses the threat of revealing user presence. We believe that the protection of presence information will become even more important in the future, because users tend to stay online whenever they are at their computers and other communication media are equipped with awareness features (e.g. the VoIP application Skype [9]).

The rest of this paper is organized as follows: In the next section, we discuss the basic functions of an IM system and derive two typical system architectures to accomplish these functions. We consider rather conceptual system architectures than those of deployed systems to keep our investigations as general as possible. Section III investigates in how far those architectures are vulnerable to unauthorized uncovering of user presence. Then we propose a new cryptographic protocol in section IV which is not exposed to the identified threats. Section V describes the system architecture of our implementation and discusses the most important implementation issues. Section VI relates our approach to prior work and section VII concludes the paper and contains some aspects of future work.

II. Existing IM system architectures

RFC 2778 [10] defines an abstract model for IM services which can be considered as an abstraction of deployed IM systems and as a template for designing new IM systems. There, the services of an IM system are subdivided into two distinct services: The *presence* and the *instant messaging service*. The presence service "allows users to subscribe to each other and be notified of changes in state". Although subscription is not necessarily restricted to authenticated users

for all IM systems, we only consider such systems which require explicit authentication for subscriptions. Otherwise, protection of presence information is useless, as the set of subscribed users cannot be controlled by the user himself. The instant messaging service can be used "to send each other instant messages" (cf. [10]).

From a technical point of view, both presence and instant messaging service require a bi-directional connection between two users. Depending on the protocol, these messages are either exchanged directly between two users or stored on a central registry and forwarded to another user either directly or on demand. The presence service requires the possibility to *send* presence messages containing the user's current state when entering the system and whenever there are changes in his state including when leaving the system. Additionally, a user should be able to *send* a request to learn about his contacts' states when entering the system. On the other hand, the presence service requires *receiving* messages, either as a response on his requests for his contacts' states, or unsolicited whenever the state of one of his contacts changes. Unsurprisingly, the instant messaging service also requires the possibility to *send* instant messages to contacts and *receive* instant messages without being solicited.

Basically, there are two possible architectures available to realize an IM system: Either the users exchange presence and instant messages directly, or they use a central node to store and forward all messages to their destination. Regardless of the choice, a central node is required.

a) Hybrid architecture: If users communicate directly with each other, a central node is necessary to resolve the users' current physical addresses. The reason is that in practice the physical address of a user's node might be a dynamic IP address or might change due to a change of the user's location. In that case, the contacts have to learn about the user's new physical address in order to directly contact him. Therefore, a central node is applied to act as name service on behalf of the users.

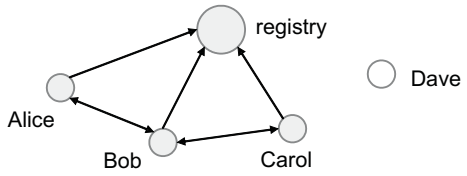

Fig. 1. IM system architecture with a registry merely acting as a name service.

Figure 1 shows the system architecture with a registry acting only as a name service and the users exchanging presence and instant messages directly. Every user entering the system connects to the registry to inform it about his current IP address and to request the mappings for his contacts. Only authorized contacts are allowed to learn about a user's presence. Therefore, the registry may only return the physical addresses of those users who have agreed to it. This is accomplished by storing a list of authorized contacts in the registry. In the next step, the user establishes connections to his contacts to exchange presence and text messages. In this step, there should be a mutual authentication of both users.

b) Centralized architecture: The second system architecture applies a central node to store messages at a central registry on behalf of a user and forward them to the user's contacts either directly or on demand. While instant messages are likely to be passed directly to their recipients, a user's state information can be stored in the registry to answer future requests by his contacts. In contrast to the first system architecture, this architecture does not imply direct interaction between two users.

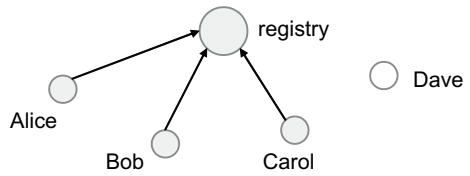

Fig. 2. IM system architecture with a registry that stores and forwards all messages in the system.

Figure 2 illustrates the second system architecture by an example. A user connects to the registry and authenticates himself to it. The registry forwards the user's initial presence to his contacts and replies to the user which of his contacts are likewise currently connected. Therefore, the registry needs to know about the user's contacts as in the first system architecture. The registry also mediates text messages sent by one user to another.

The well-known systems rely mostly on a central registry to store and forward messages. The reason is that users are not required to accept incoming connections which might be a problem due to firewalls or network address translation (NAT) boxes. Obviously, the drawback of the centralized architecture is that most work is accomplished by the central registry, so that this approach is less scalable than the more decentralized, hybrid approach. Some systems also implement a mixture of both architectures by forwarding presence and text messages using the registry and handing out the user's IP addresses for direct file transfers.

III. Security analysis

Our focus is an IM system that protects its users' presence information. While this information shall be transmitted to a user's authorized contacts, it must not be known to unauthorized users. Although this kind of protection is meant to be provided by all currently deployed IM systems, it can be circumvented. We will point out some weaknesses in order to determine necessary system requirements to protect presence information. For a more elaborate discussion, see [11].

The first potential weakness of deployed IM systems is the fact that the presence of all system users is stored in the registry. In a hybrid architecture, every user connects once to the registry when entering the system thus revealing his presence from that point of time up to an uncertain time in the future. In a centralized architecture, every user stays

connected to the registry during his entire online session, thus revealing his presence to the registry completely. This knowledge could be exploited by an untrustworthy system provider or could be exposed to an attacker by breaking into the registry. As a counter measure, no information that is passed to the registry may reveal any user's presence in the system. This requirement is the most difficult to realize and is therefore the central part of our contribution.

Another threat is the linkability of messages to a certain user, thus revealing that user's presence. Depending on the chosen architecture, this applies either to messages exchanged between a user and the registry, or to messages sent from one user to another. In order to prevent this threat, all messages must be encrypted, so that neither content nor sender or receiver are disclosed. Some deployed systems offer link encryption to prevent this threat.

The last threat we consider is that the mapping between a user and his current IP address is revealed. Being aware of this mapping, an attacker could monitor communication originating from or being sent to that IP address and thus conclude presence of the associated user. If a user's IP address is static, this attack can be performed at any time in the future. In a hybrid architecture, a user's IP address must be known at least to his contacts, so that they can establish a connection to him. Although the user's current contacts may be aware of the user's IP address, they are not allowed to know about it after the user has revoked their authorization by removing them from his contact list. Unfortunately, knowledge about this mapping does not disappear at former contacts, so that no user should learn about the mapping at all. In a centralized architecture, the user's IP address is known to the registry which passes messages to the user. Hence, a user's IP address is either known to unauthorized users or to the registry which we do not consider to be trustworthy. One possible way to hide the mapping between a user and his IP address is to apply an anonymous communication network.

IV. Overall protocol design

After discussing the possible threats to IM systems with regard to protection of user presence, we present a new IM system design which is not exposed to these threats.

The basic decision in designing the new IM system is whether to apply a hybrid or a centralized architecture. As we demonstrated above, both architectures can exhibit threats which lead to presence exposure. These threats can be counteracted in both architectures with more or less similar measures. We decided to *apply a hybrid architecture*, because of reasons which are beyond our attempt to protect user presence: The main reason is that the registry in a hybrid architecture simply has to provide a storage mechanism instead of a rather sophisticated application logic of a storing and forwarding registry. This allows us to make use of deployed storage systems, e.g. a distributed hash table (DHT), instead of deploying a registry ourselves. However, this is not a conceptual issue. It should also be possible to redesign our protocol to match a centralized architecture, if required.

Our design consists of two main principles that refer to the different technical levels of security threats:

A We attempt to hide the mapping between a user and his IP address by using an anonymous communication network, so that disclosure of IP communication does not reveal the presence of a certain user.

B We apply a cryptographic protocol between two users and the central registry. This protocol allows the two users to establish a connection to share presence information and text messages and ensures that no other participant in the system learns about the presence of either user.

The protocol consists of three possible communication links as the following scenario illustrates. The contents of exchanged messages are subject to the subsequent discussion.

When connecting to the system, Bob publishes his address at the registry which is simply acknowledged by the registry:

$$B \leftrightarrow R \tag{1}$$

Alice, after doing the same operation, retrieves Bob's address from the registry:

$$A \leftrightarrow R \tag{2}$$

If Alice could retrieve Bob's address, she establishes a connection to Bob to exchange presence and text messages with him:

$$A \leftrightarrow B \tag{3}$$

A. Anonymous communication network

The purpose of the anonymous communication network is to hide the IM user's IP address in all communication links. Connections to the registry require anonymity of the user as initiator, but not necessarily of the registry as recipient. The reason is that the IP address of the registry may be known to anyone. For connections between two users, both IP addresses must be hidden.

Initiator anonymity is provided by most anonymous communication networks, because it is used for typical client-server settings, e.g. anonymous Web browsing. In contrast to that, recipient anonymity is only required for offering an anonymous service or for peer-to-peer settings and is not provided by all anonymous communication networks. Tor [1] offers both types of anonymity. Using sender anonymity is accomplished by accessing a remote address via the Tor network which hides the sender's IP address. In order to make use of recipient anonymity, the user has to register a so-called *hidden service* which can then be accessed using a unique logical address via the Tor network. Tor allows hiding both, the initiator's location and the location of the hidden service.

We apply Tor to our system for incoming and outgoing communication. A user provides a hidden service that can be contacted by his contacts to exchange presence and text messages. In our design, the hidden service address must be changed from time to time in order to prevent unauthorized users from identifying a certain user by observing the users hidden service address.

B. Cryptographic protocol

While the anonymous communication network hides the users' IP addresses, the purpose of the cryptographic protocol is to hide all user-specific information of the message content from unauthorized users and from the registry. This applies to all three communication links that have been identified above.

1) Publication in the registry: The first communication takes place when Bob enters the system and deposits his address in the registry, so that Alice and his other contacts may contact him when they enter the system, too. According to our requirements, Bob has to encrypt his address, so that only his contacts may decrypt it. This can be achieved with public key encryption by encrypting the address for all intended recipients using their public keys. Therefore, we assume that all users in the system have generated a key pair for an asymmetric encryption scheme, e.g. ElGamal [12]. Further we expect that every user has created a public key certificate for publication on a global key server and has informed his contacts about the key identifier, so that they can download the user's certificate. It is important to notice that we do not consider communication to a global key server as a threat to our attempt to protect the user's presence. Such a connection must only be performed once and without a direct chronological correlation for using the IM system. An alternative to using a global key server would be the transmission of public key certificates using another communication media.

Bob has to include a signature to the message before encrypting it. In doing so, Alice can verify that the address really belongs to Bob. Otherwise, an attacker could generate such a message and make Alice connect to his address, thus revealing her presence. We require another key pair for an asymmetric signature scheme, e.g. RSA [13]. The public key can also be published with the public key certificate of a user.

Although the content of Bob's address is now protected from unauthorized users including the registry, Bob has to provide a means for Alice and his other contacts to find it. If he did not, Alice would have to try to decrypt all addresses that are stored in the registry. This would ensure that only authorized users can learn about Bob's address, but obviously, this solution does not scale. In an ordinary name service, Bob could use his user name as a key for storing his address. Unfortunately, this would immediately reveal his presence in the system. Bob could also store his address separately for every contact using his contact's user name as key. However, this would reveal the contact's presence when accessing the addresses that have been deposited for him.

One solution that reveals neither the publisher's nor the retriever's presence is to previously agree on a secret key for publication. As long as an attacker cannot relate this secret key to either publisher or intended retriever, the observation of a publication or a retrieval using this key does not reveal presence of a certain user.

Although Bob could agree upon the same secret key with all of his contacts, this would make removal of a contact more difficult. A removed contact would not automatically forget the secret key, so that he could still determine the user's presence from observing a publication under that key. Thus, Bob would have to agree on a new secret key with his remaining contacts whenever he removes a contact, which is quite cumbersome. Instead, Bob agrees on separate secret keys with all of his contacts and publishes his address for them individually. Thus, he can remove an existing contact by simply stopping to publish his address for that contact.

The agreement on a secret key does not permit communication within the system, because communication is not possible until the address of the communication partner has been retrieved. This makes ordinary key agreement using a Diffie-Hellman key agreement schema [14] inapplicable, because it requires mutual exchange of two public keys based on the same key parameters. We can however modify this schema to use fixed parameters for all users in the system and use pre-computed Diffie-Hellman public keys. Such a schema is e.g. described by Menezes et al. as "Diffie-Hellman with fixed exponentials" [15]. These public keys are added to a user's public key certificate. Every two users may then combine their own private key with their contact's public key to calculate their shared secret key. This approach reduces the communication to the mutual exchange of public key certificates.

In order to prevent from generating patterns of publication keys, we want to change them on a regular basis. Thereby an unauthorized user cannot derive information from observing the publication of addresses using the same key or a combination of keys. Therefore, we have to ensure that an unauthorized user cannot conclude that two distinct keys that have been generated at different times belong to the same pair of users. To achieve this, we append the number of days since 1970 of the local system time (GMT) to the secret key and apply a cryptographically secure hash function, e.g. SHA-1 [16], to the resulting value. The result is a key that can only be constructed by the two users holding the Diffie-Hellman private keys and that changes every day. We use this value for publishing and retrieving the addresses of the two users.

In summary, the first message consists of the following values: Bob (B) sends to the registry (R) two values: The first value is the concatenation of his shared secret key with Alice K_{AB} and the current day D on which the secure hash function H is applied. The second value consists of the address of his hidden service HS_B and a signature of it created with his signature function S_B. Both values are encrypted for Alice as recipient using her encryption function E_A.

$$B \rightarrow R : H(K_{AB} + D), E_A(HS_B, S_B(HS_B)) \qquad (4)$$

2) Retrieval from the registry: The second communication takes place between Alice and the registry. It is quite similar to the first communication with the difference, that she requests the encrypted messages using the same key as Bob used for publication and that the registry replies with the encrypted address:

$$A \rightarrow R : H(K_{AB} + D) \quad (5)$$

$$R \rightarrow A : E_A(HS_B, S_B(HS_B)) \quad (6)$$

3) Message exchange between two users: The third communication occurs when Alice tries to establish a connection to Bob using the address that she has received from the registry. This communication needs to be encrypted, so that neither Alice's nor Bob's presence can be derived from their exchanged messages. We apply a hybrid encryption schema starting with an asymmetric encryption using the asymmetric encryption key that was already used above and a symmetric encryption, e.g. Triple-DES [17], using a session key that can be agreed on at the beginning of the communication. In addition to encryption, Alice has to authenticate herself to Bob, because in principle, anyone could initiate a communication to Bob's hidden service, not only Alice or one of Bob's other contacts. Authentication can again be performed by adding a signature that is created using Alice's signature function.

The resulting messages for establishing a communication and for subsequent messages contain the following values: Alice sends a fresh session key K_S and a signature of it using her signature function S_A, both encrypted using Bob's encryption function E_B to Bob. Succeeding messages M are then encrypted using the symmetric encryption function E_S using the shared session key. Alice can be sure that she is really talking to Bob, because only Bob can decrypt the session key and use it for encrypting messages addressed to her.

$$A \rightarrow B : E_B(K_S, S_A(K_S)) \quad (7)$$

$$B \leftrightarrow A : E_S(M) \quad (8)$$

This cryptographic protocol is not exposed to the threats we described above. The registry does not learn any user-specific information, but merely acts as a storage for encrypted key-value pairs. All messages are encrypted and do not reveal any information about sender or recipient. Finally, IP addresses are hidden using the anonymous communication network.

V. Implementation

After describing the underlying protocol, as well as how it addresses the threats discussed in III, we provided a sketch of the entire system being implemented and the subsystems on which it is based on.

The architecture of our IM system is based on four other systems:

1) The low-latency anonymous communication network Tor [1]: We only use Tor as a communication layer for outgoing and incoming connections. A so-called *Tor proxy* runs on the local client machine and connects to the so-called *Tor routers* which are distributed over the Internet.
2) The Java implementation of the OpenPGP crypto system [18] Bouncy Castle [2]: This is a cryptographic API which provides cryptographic primitives for symmetric and asymmetric encryption as well as secure hash functions. PGP public key servers provide storage of OpenPGP-compatible public key certificates.
3) The distributed hash table OpenDHT [3] which we use as registry: OpenDHT is a deployed distributed hash table that can be used for testing new peer-to-peer applications. Its nodes can be accessed using a simple protocol based on XML-RPC [19]. A current list of OpenDHT nodes can be downloaded from the OpenDHT web server via HTTP.
4) The Jabber-based [20] Wildfire server [21], which we adapt for presenting the IM service to the local user: The original purpose of this server is to connect Jabber clients to the Jabber network. Nevertheless, we adapted it to connect Jabber clients to our own IM network. The primary purpose for our implementation is to provide users with a comfortable interface by allowing them to use a potentially more user-friendly Jabber client.

Our implementation is located between the Wildfire server as the presentation layer and the Tor proxy as the communication layer. It is implemented in Java 1.5 and is running in the same Java virtual machine as the Wildfire server. It performs the tasks that are described in the protocol design above. It maintains its own hidden service for incoming connection requests and periodically publishes its address in the DHT. It also attempts to retrieve addresses from its contacts. Whenever it receives a hidden service address, it tries to establish a connection to the contact and starts exchanging presence and text messages. On disconnecting, our implementation sends a last presence message and closes the existing connections. Figure 3 shows an overview of the architecture.

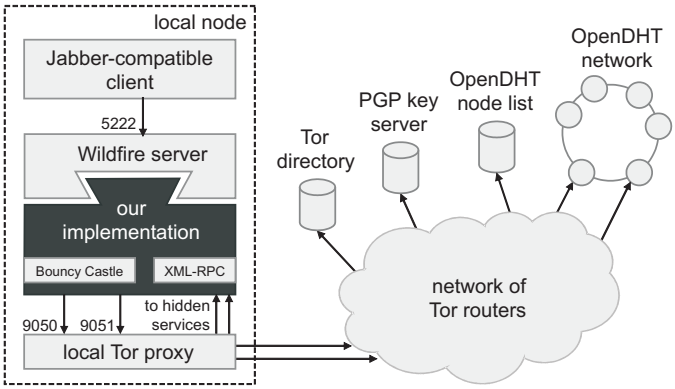

Fig. 3. Overview of the implemented system architecture.

In the following text, we go into the details as to how these subsystems had to be adopted in order to provide the proposed protocol.

A. Tor

The Tor network consists of approximately 800 Tor routers (as of August 2006 [22]) that are distributed over the Internet and create an overlay network on top of TCP/IP. The Tor protocol has a binary format. Its specification is available as open source. The Tor directory is a central network component

that is used e.g. for managing the network topology. It is currently replicated among 4 servers. In order to use Tor for anonymizing connections, users access it using a local Tor proxy. Although this is rather the exception, it is also possible to access Tor using a local Tor router instead of a Tor proxy. The implementations of all Tor components are written in C and are available as open source.

The architecture of our IM system incorporates a local Tor proxy that is running as a separate process. The reason for this is that the rest of the implementation is written in Java and that there does not exist a Java implementation of the Tor proxy yet. Our implementation connects to the local Tor proxy using a couple of TCP sockets: The Tor proxy provides a SOCKS port running on TCP port 9050 that can be used for outgoing connections. Furthermore, it offers a control port on TCP port 9051 that can be used to change the configuration of the Tor proxy and send commands to it. In addition to that, our implementation opens one TCP port for every offered hidden service to which the local Tor proxy connects. Finally, the Tor proxy opens quite a lot of outgoing connections to remote Tor routers. However, it does not require the local node to accept incoming connection requests.

Tor hidden services are usually configured using the configuration file `torrc` before starting the Tor proxy and remain the same during execution time. As already stated above, we have to change the hidden service address from time to time. This is accomplished by deleting the hidden service configuration and creating a new one which typically requires restarting the Tor proxy. Unfortunately, this closes all running connections which we would rather avoid. That is why we access the Tor proxy via a separate data connection. This can be accomplished with a specific Tor controller API which is available in Java from the Tor project. Using the controller, we can configure new hidden services and reconfigure existing ones during runtime.

After registering a new hidden service, we have to find out and publish its address, so that other users can access it via the Tor network. Every hidden service has its own data directory for storing its private key and the file `hostname`. The latter contains the hidden service address formatted as base32 string [23] of 16 characters (80 bit length) plus the extension ".onion". This address can be passed to a remote Tor proxy to establish a connection to this hidden service.

B. Bouncy Castle and OpenPGP key servers

According to our design, we require implementations of cryptographic primitives for symmetric and asymmetric encryption, a secure hash function, and key agreement. OpenPGP specifies algorithms for all of these cryptographic primitives, e.g. RSA, ElGamal, Triple-DES, SHA-1, and DH. It also specifies a format for public key certificates. Due to the widespread employment of email encryption, a couple of OpenPGP implementation and a good infrastructure of key servers exist.

Bouncy Castle is an open-source Java implementation of both, the Java Cryptography Extension (JCE) service provider interface and the OpenPGP standard. The cryptographic primitives can be accessed via the JCE API which is included in Java by default. In order to use Bouncy Castle with realistic key sizes, one has to install the so-called Unlimited Strength Jurisdiction Policy Files which merely means to exchange two files in the JRE installation folder. The OpenPGP implementation provides access to cryptographic keys and messages in the OpenPGP format, which is also required for message exchange with OpenPGP key servers.

We had to perform one small, but important change to the way Bouncy Castle encrypts messages asynchronously. The OpenPGP standard adds a so-called *key hint* to every encrypted message. This hint contains the identifier of the key which must be used for decryption. It relieves the recipient from trying several keys until finding the right key. Unfortunately, this key hint uniquely identifies an IM user who retrieves an address that has been encrypted for him and reveals his presence. Therefore, we had to change the implementation of Bouncy Castle, so that the key hint is overwritten with a random identifier after performing the encryption.

OpenPGP does not include a specification for pre-computed Diffie-Hellman keys as we need them for agreement of a secret key without direct communication. Hence, we generate an ElGamal key pair based on system-wide identical parameters and add its public key to the public key certificate of a user. The calculation of the secret key shared by two users is obtained by converting these ElGamal keys into DH keys and performing the DH key agreement.

Public key certificates obtain unique identifiers by means of the main key identifier which is the RSA signature key. This identifier consists of the lowest 32 bits of the 160-bit hash value of the key. We use this identifier as a pseudo-unique user name and for retrieving a user's public key certificate from the key server.

Although Bouncy Castle also provides symmetric ciphers, we do not need them. The reason is that hidden service connections are always end-to-end encrypted between the two Tor proxies. Therefore, we do not need further symmetric encryption, but only authentication of the initiator of a hidden service connection to the hidden service provider. This is accomplished by replacing the session key in the protocol above by a *nonce* and attaching a signature of this nonce instead of the signature of the session key.

C. OpenDHT

Our protocol assumes a central registry for exchanging encrypted Tor hidden service addresses. The requirements to this registry are rather simple. It merely has to support two operations: Storing given key-value pairs and returning the values belonging to a given key.

Although we could have implemented a new registry providing this task, we wanted to rely on a deployed infrastructure. One possibility would have been to store these entries in the Tor directory which provides a similar task e.g. for router descriptors and hidden service descriptors. According to the Tor developers, the Tor directory is already a performance

bottleneck [24]. Hence, we refrained from it, because we do not want to decrease performance. Apart from that, storing a new type of entries would have required changing the implementation and redeploying the Tor directory servers.

We selected a publicly available distributed hash table (DHT) as a logic central registry. It guarantees, that a key-value pair that is successfully stored at one DHT node can be retrieved from any other DHT node at least for a given time-to-live. We chose OpenDHT [3] to use as registry, because it is a deployed and freely available DHT service. Its purpose is to allow developers to test DHT-based applications without the need of setting up an own DHT. OpenDHT restricts the key size to 160 bits and the value size to 1024 bits. The key length is sufficient as SHA-1 provides equally sized hash values. The restriction of values is sufficient for our application, because operations like attaching a signature, random padding, and asymmetric encryption do not let it increase beyond 600 bits. OpenDHT allows storing multiple values for the same key, which is important for our protocol imlementation, because two contacts both use the same DHT key for up to one day, but with possibly changing DHT values. OpenDHT can be accessed by a simple protocol based on XML-RPC [19].

D. Wildfire server

The purpose of the Wildfire server for our implementation is merely to provide a user-friendly interface. This is accomplished by connecting a Jabber-compatible messaging client to the locally running Wildfire server. The alternative would have been to implement an own user interface or extend an existing messaging client. We did not want to put too much effort into creating and maintaining a nice GUI, but rather focus on providing a correct and secure implementation.

Wildfire provides a sophisticated mechanism for creating plug-ins. It allows registration of event interceptors for adding or removing local users, starting or stopping local user sessions, changing presence state, and sending text messages. Presence updates and text messages can easily be sent to attached clients. The server contains a database that can be extended by user-defined tables. It also provides an own logging mechanism. The user roster containing user and contact data can be accessed from the plug-in. Installation of a plug-in is accomplished by copying an appropriate `jar` file to the plug-in directory and starting the server.

E. Preliminary protocol performance evaluation

The important issues regarding the protocol implementation have already been described in section IV. The detailed technical issues are of minor interest here.

Instead, we present some early performance measurements of the factors, which have an impact on our protocol implementation. The purpose of these measurements is not to compare our approach with existing IM systems (which do not have our privacy properties) or to find reasons for the found results. We are rather interested whether our approach performs in a way that is acceptable for an "instant" messaging system, or not. We measured the delays of requests to OpenDHT over Tor (and whether Tor or OpenDHT is responsible for these delays) and of communication to hidden services of other IM users (connection establishment and subsequent messages). In detail, we measured the following data:

1) The round-trip time for an initiator-anonymous request to a public DHT server. Requests are malformed on purpose, so that they can be answered by the DHT node directly and do not require further requests to other DHT nodes.
2) The time to answer a DHT request (put operation with random key and value), but using a direct IP connection instead of Tor.
3) The time to establish a connection to a running hidden service
4) The round-trip time for messages to a hidden service using an established connection.

For (1) we found that Tor features a round-trip time of 1.97 $\pm$ 0.06 seconds (all values are given as mean value $\pm$ standard deviation). Only 2 outliers out of 64,905 measured values took 7.53 and 7.41 seconds, respectively, with the next smaller value of 2.33 seconds. The surprisingly low standard deviation indicates that Tor establishes connections in advance of actual requests. Most of the time seems to be used for encryption and decryption on the Tor routers which usually takes a constant amount of time.

The response times of OpenDHT that we measured in (2) are 5.89 $\pm$ 12.5 seconds. The high variance is characteristic for a DHT, in which response times increase logarithmically depending on the number of nodes.

The time to establishment a connection to a hidden service in Tor (3) is 5.39 $\pm$ 12.4 seconds. This high standard deviation is a sign that Tor has to establish new circuits in order to connect to a hidden service or that there are other problems in Tor.

After establishing a connection to a hidden service, message round-trip times (4) are 2.32 $\pm$ 1.66 seconds with a maximum value of 9.99 seconds. This comparatively low variance seems to result from encryption and decryption, rather than from establishment of new Tor circuits.

The total time for establishing an initial connection to an IM contact can be calculated by adding twice the request time to a DHT node over Tor (1) and performing a DHT request (2), i.e. one request for a put operation and one for a get operation, plus the time to establish a connection to a hidden service (3). The calculated mean time for this is 21.1 seconds. Afterwards, exchange of text messages and presence information takes 2.32 seconds in the mean (4).

We also performed some basic measurements of OpenDHT concerning its reliability. We found that the DHT keeps all stored values for the specified time-to-live.

In our opinion, the measured values indicate that our approach is acceptable for a typical usage of IM. However, these measurements should only be considered as first step towards a complete performance evaluation. So far, we have not deployed our implementation on a large scale.

VI. Related work

Most related work on security in IM systems focuses on message confidentiality and authentication of users. Cerulean Studios [25] have developed SecureIM, an encryption protocol for instant messages on top of AIM and ICQ that provides message integrity and confidentiality. Neither deals with authentication of communication partners nor confidentiality of presence information. VeriSign offers a personal certificate [26] for signing and encrypting text messages within the AIM network. It provides authentication, integrity, and confidentiality of instant messages, but no privacy of presence information. The wija project [27] provides an IM application based on the Jabber protocol which allows users to encrypt and sign instant messages and to sign presence information, but does not conceal presence from other users. ScatterChat [28] is an extension of the open-source multi-messenger Gaim [29] which allows encryption of instant messages and anonymity by accessing the public IM server via Tor—unfortunately, this still reveals the user's presence to the registry.

Prior to this paper, we have published an earlier version of our protocol [11] which underwent some important design changes in the course of its implementation.

Currently, there are no comparable approaches that consider the threat of revelation of presence by an untrustworthy system provider or an attack on the central registry. However, we do think that such attacks are possible and that the value of a user's presence information is worth protecting.

VII. Conclusion and future work

We have proposed an IM system that is explicitly designed to protect user presence without the need of a trusted, central registry. We presented a Java implementation based on the anonymous communication network Tor, the cryptographic suite Bouncy Castle, and the distributed hash table OpenDHT.

The next step is a comprehensive and more detailed threat analysis of our cryptographic protocol. We intend to perform a formal verification of its security properties. A performance evaluation of the deployed system is up to future work, too.

There might also be some benefit from generalizing our solution from IM systems to peer-to-peer systems in general. It could for example, be applied to friend-to-friend networks [30], i.e. peer-to-peer networks, which only allow connections between mutually authorized nodes. Therefore, we subdivided out implementation into an IM-specific and a more general part, which provides a lightweight, socket-like API.

Acknowledgement

The authors would like to thank Jens Bruhn, Sven Kaffille, and Andreas Schönberger for numerous fruitful discussions.

References

[1] R. Dingledine, N. Mathewson, and P. Syverson, "Tor: The second-generation onion router," in *Proceedings of the 13th USENIX Security Symposium*, 2004, pp. 303–320. [Online]. Available: http://www.usenix.org/events/sec04/tech/full_papers/dingledine/dingledine.pdf

[2] Bouncy Castle project homepage. [Online]. Available: http://www.bouncycastle.org/

[3] S. Rhea, B. Godfrey, B. Karp, J. Kubiatowicz, S. Ratnasamy, S. Shenker, I. Stoica, and H. Yu, "OpenDHT: A Public DHT Service and Its Uses," in *SIGCOMM '05: Proceedings of the 2005 conference on Applications, technologies, architectures, and protocols for computer communications*. New York, NY, USA: ACM Press, 2005, pp. 73–84.

[4] B. A. Nardi, S. Whittaker, and E. Bradner, "Interaction and outeraction: instant messaging in action," in *Proceedings of the 2000 ACM conference on Computer supported cooperative work*, December 2000.

[5] S. Patil and A. Kobsa, "Preserving Privacy in Awareness Systems," in *Wissen in Aktion*, 2004, pp. 119–130. [Online]. Available: http://www.ics.uci.edu/~kobsa/papers/2004-FSKuhlen-kobsa.pdf

[6] ICQ homepage. [Online]. Available: http://www.icq.com

[7] AOL Instant Messenger homepage. [Online]. Available: http://www.aim.com

[8] MSN Messenger homepage. [Online]. Available: http://messenger.msn.com

[9] Skype homepage. [Online]. Available: http://www.skype.com

[10] M. Day, J. Rosenberg, and H. Sugano, "RFC 2778 – A Model for Presence and Instant Messaging," February 2000. [Online]. Available: http://www.ietf.org/rfc/rfc2778.txt

[11] K. Loesing, M. Dorsch, M. Grote, K. Hildebrandt, M. Röglinger, M. Sehr, C. Wilms, and G. Wirtz, "Privacy-aware Presence Management in Instant Messaging Systems," in *20th IEEE International Parallel and Distributed Processing Symposium*, April 2006.

[12] T. ElGamal, "A public key cryptosystem and a signature scheme based on discrete logarithms," in *Proceedings of CRYPTO 84 on Advances in cryptology*. New York, NY, USA: Springer-Verlag New York, Inc., 1985, pp. 10–18.

[13] R. L. Rivest, A. Shamir, and L. Adleman, "A method for obtaining digital signatures and public-key cryptosystems," *Commun. ACM*, vol. 21, no. 2, pp. 120–126, 1978.

[14] W. Diffie and M. E. Hellman, "New Directions in Cryptography," *IEEE Transactions on Information Theory*, vol. IT-22, no. 6, pp. 644–654, November 1976.

[15] A. Menezes, P. van Oorschot, and S. Vanstone, *Handbook of Applied Cryptography*. CRC Press Inc., U.S., December 1996.

[16] "Proposed Federal Information Processing Standard for Secure Hash Standard," *Federal Register*, vol. 57, no. 21, pp. 3747–3749, January 1992.

[17] H. Kummert, "RFC 2420 – The PPP Triple-DES Encryption Protocol (3DESE)," September 1998. [Online]. Available: http://www.ietf.org/rfc/rfc2420.txt

[18] J. Callas, L. Donnerhacke, H. Finney, and R. Thayer, "RFC 2440 – OpenPGP Message Format," November 1998. [Online]. Available: http://www.ietf.org/rfc/rfc2440.txt

[19] Apache XML-RPC project homepage. [Online]. Available: http://ws.apache.org/xmlrpc/

[20] Jabber project homepage. [Online]. Available: http://www.jabber.org/

[21] Wildfire server project homepage. [Online]. Available: http://www.jivesoftware.org/wildfire/

[22] Number of running Tor routers. [Online]. Available: http://www.noreply.org/tor-running-routers/

[23] S. Josefsson, "RFC 3548 – The Base16, Base32, and Base64 Data Encodings," July 2003. [Online]. Available: http://www.ietf.org/rfc/rfc3548.txt

[24] R. Dingledine, N. Mathewson, and P. Syverson, "Challenges in deploying low-latency anonymity (DRAFT)," 2005. [Online]. Available: http://tor.eff.org/cvs/tor/doc/design-paper/challenges.pdf

[25] Cerulean Studios homepage. [Online]. Available: http://www.ceruleanstudios.com/

[26] Enterprise-Level Security and Management for Instant Messaging. [Online]. Available: http://www.verisign.com/stellent/groups/public/documents/white_paper/005324.pdf

[27] Wija project homepage. [Online]. Available: http://www.media-art-online.org/wija/

[28] ScatterChat project homepage. [Online]. Available: http://www.scatterchat.com/

[29] Gaim project homepage. [Online]. Available: http://gaim.sourceforge.net/

[30] D. Bricklin, "Friend-to-Friend Networks." [Online]. Available: http://www.bricklin.com/f2f.htm