Home

Jump to bottom

jim-b edited this page on 1 May 2015 · 115 revisions

ECCSI-SAKKE

#1. Project overview This work was a personal project to try and understand how the cryptography, often referred to as Mikey Sakke, described in RFCs 6507, 6508 and (parts of) 6509 worked. I had a few personal goals for this development:

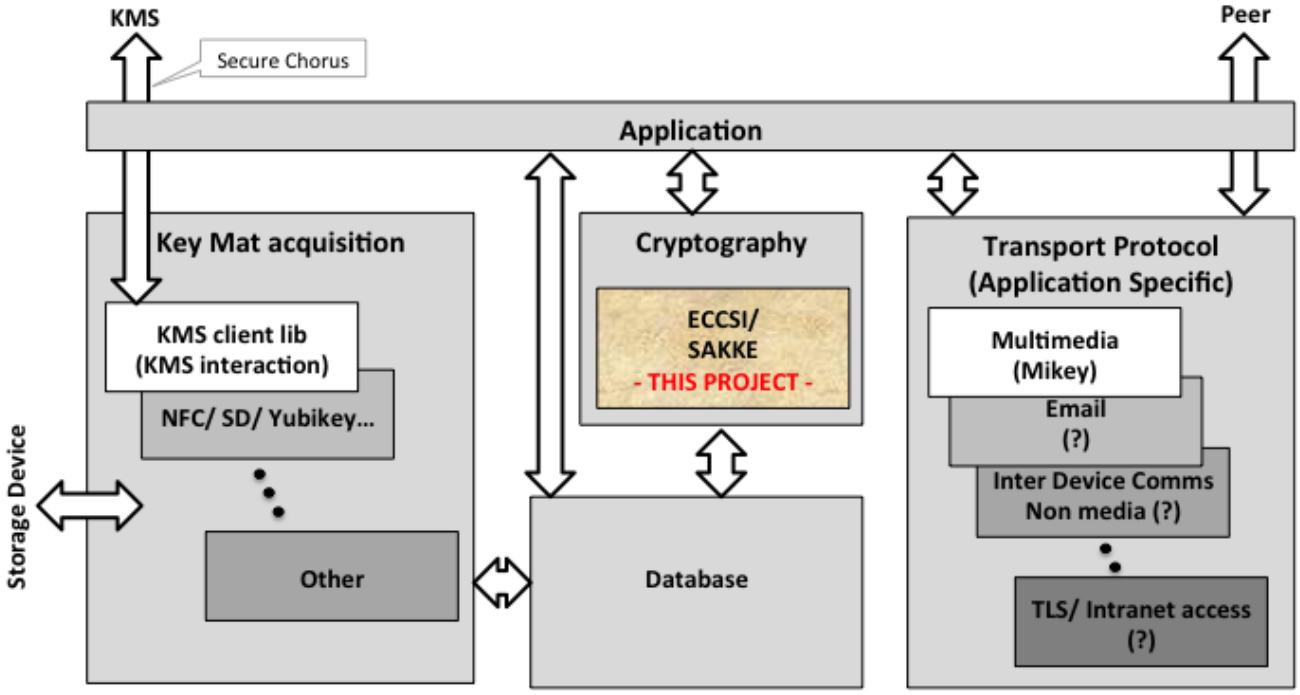
- Produce a simple (as possible), easy to build, 'C' implementation using OpenSSL for the maths.
- To make it as clear as possible, by reference out to the RFCs (6507-6509), where specific calculations were described and expected results were indicated, so that these could be checked i.e. A learning exercise/ tool, useful to others trying to understand what is going on internally.
- To only implement ECCSI and SAKKE. As shown in 'High Level System View" (below) my personal view is that there are many potential uses in various technologies and applications, not just for Multimedia where Mikey could be used for peer interaction. Isolating this part (the cryptography) from *Key distribution* and *Peer interaction* would allow others to build, not just Secure Multimedia solutions but also other applications using this core Crypto.

  As such, naming this project anything like **Mikey-Sakke-X** *could* be considered somewhat of a misnomer as Mikey is about Session Key (distribution) in real time environments. This does **not** in any way mean you can't take the output of this code and put it into I_Messages, as described in RFC 6509, I'm just not limiting you to that implementation (for Peer interaction) by bundling it all together.

  Additionally, this goal set sensible bounds for this project; As an individual I could not possibly hope to implement this Cryptography and a secure media application (possibly on several platforms) and a Key Management mechanism (e.g. KMS) within the timescale available to me whilst I took a break (~2-3 months).

NOTE! To generate Mikey-Sakke Key Material please refer to my other project KMS and it's wiki.

##1.1. High Level System View The diagram below shows how the ECCSI/ SAKKE Crypto library that has been produced *might* be used in multiple applications. It expresses a personal opinion, I do not know for sure that any further uses, other than Secure Multimedia, have been, or are being, considered.



1.2. Restrictions on use

None that I know of; You can use this code for free following the Apache 2.0 License, including the *kudos clause* provided by the NOTICE text file.

I'm not of entirely up to speed on licensing issues, if something's wrong or I've stepped on anyone else's toes, let me know and I'll put it right.

1.3. Other implementations

There was an existing PoC (Proof of Concept) implementations which was invaluable to me in developing this project, for cross checking expected values. I have **a great deal** of respect for the people who worked on this it was fine work. If you get the chance to check out that version, you can find it here Secollab's Mikey-Sakke implementation.

I have recently heard of another implementation Secure Chorus. I do not know for sure if this is the product of the Secure Chorus Group *per se*. Also, I have not checked it out to have a look as I've been to busy doing this.

If you're investigating Mikey Sakke you might want to have a look at either or both of these two as well (or others, if you can find them) and decide, with respect to what you are trying to achieve, what works for you. The other two look to me like they are trying to provide a more complete solution, whereas this project solely concerns itself with the core ECCSI/ SAKKE crypto i.e. no peer interaction with Mikey, or, KMS interaction using the Secure Chorus specifications. Pick what's right for you.

If you know of any other freely implementations please let me know and I'll add them here (see Contact Details below).

2. Mikey Sakke Overview

As well as the RFCs detailed above, there are some wiki Pages:

- ID based cryptography
- ID based encryption
- Sakke

Is that all too confusing? Has your brain seeped out through your ears in a desperate attempt to escape from the maths? Fear not, most of it is probably only understandable to a few people, and I'm not one of them either... instead I give you a brief Bluffers Guide To IBE explaining why this is actually *interesting*, and with no maths.

3. Architecture

3.1. Design Decisions

As you can see in the Software Structure section below I decided to abstract out a couple of pieces of the functionality that really should be written by whomever is looking to use this project as part of a solution. This is because there may be certain requirements, policies, or, procedures which the finished product must comply with.

- An abstraction from the core ECCSI/ SAKKE cryptography of PRNG (Pseudo Random Number Generator). Different organizations may have different requirements and policies with respect to PRNG. Abstracting this functionality, so that values are passed into the core ECCSI/SAKKE crypto allows developers to implement their own schemes, but leave the ECCSI/ SAKKE core untouched. Additionally, it makes easier to plug in 'known values' to test expected results (like those in the RFCs).

- An abstraction of the key material data storage. Different implementations may have different requirements for the Database technologies to be used (e.g. SQL, LDAP etc). Additionally, key material distribution is (currently) presumed to come from a KMS, but this is **not set in stone** and *may* not be suitable for some future uses. There could be a variety of mechanisms e.g. NFC, SD card, Yubikey or some other secure key storage mechanism. Implementing these storage mechanisms in line with a clear API (such as that provided) would mean no changes are required to the ECCSI/SAKKE core code, it remains untouched, and works independently of the back end storage mechanism implemented.

You will find **demo** code implementing this functionality in the appropriately named **4u2change** directory.

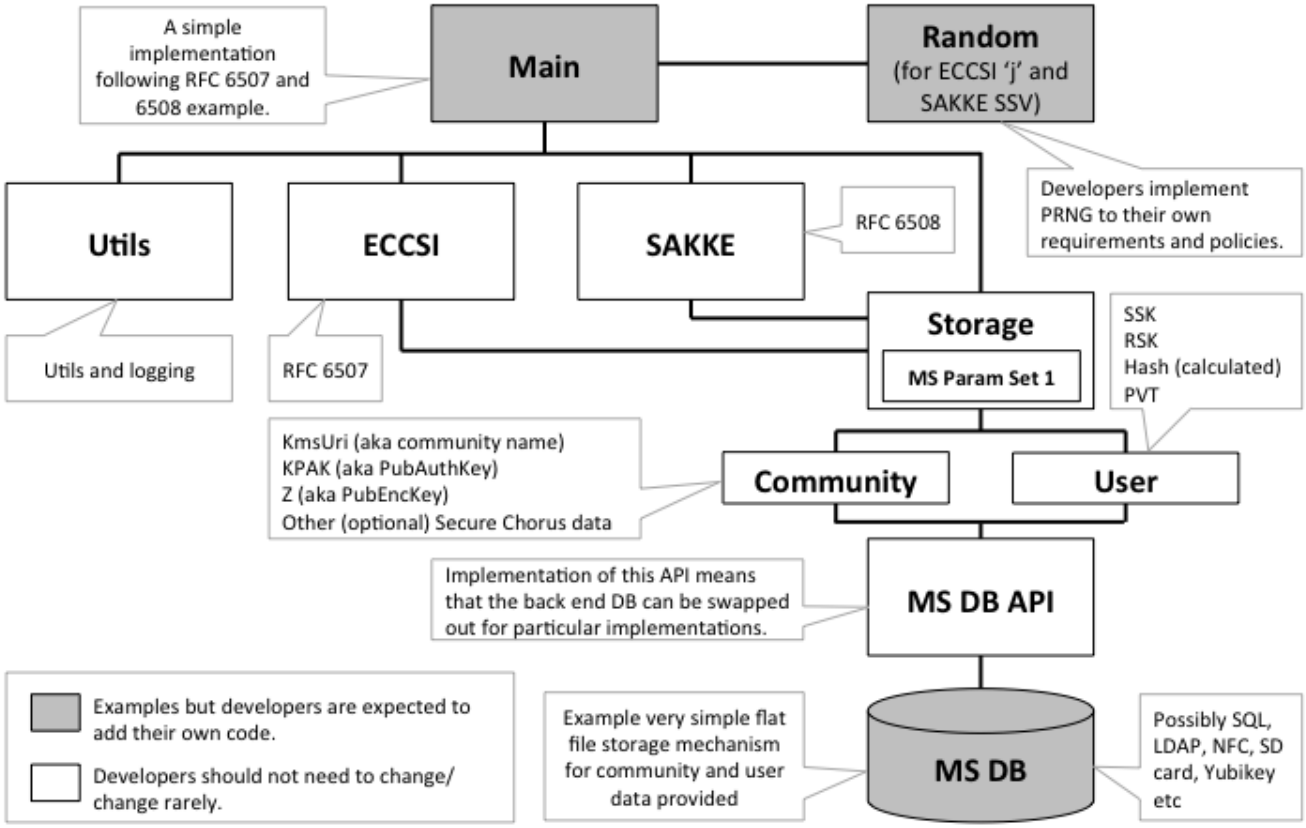
3.2. Software structure

The diagram below show structure of the ECCSI/ SAKKE Crypto library code constructed using the white boxes. The dark boxes indicate code that application developers would implement themselves, even though example/ demo code is provided.

The **Main box** is the demo *Application*, this would be rewritten for whatever application is being implemented, it may (as an example) include additional code for creating/ sending and receiving/ handling, Mikey messages for Secure Multimedia session.

The **Random box** implements a simple *PRNG* (Pseudo Random Number Generator), application developers may (and probably would) have their own requirements, policies and procedures for producing PRN (Pseudo Random Number) values.

The **MS DB box** implements a *Database* as a simple flat file storage mechanism for Mikey-Sakke Key Mat. Application developers may have preferences for any one of a plethora of back-end database implementations. As long as these comply with the specification of **MS DB API** it *should* all still work.



Personal comment

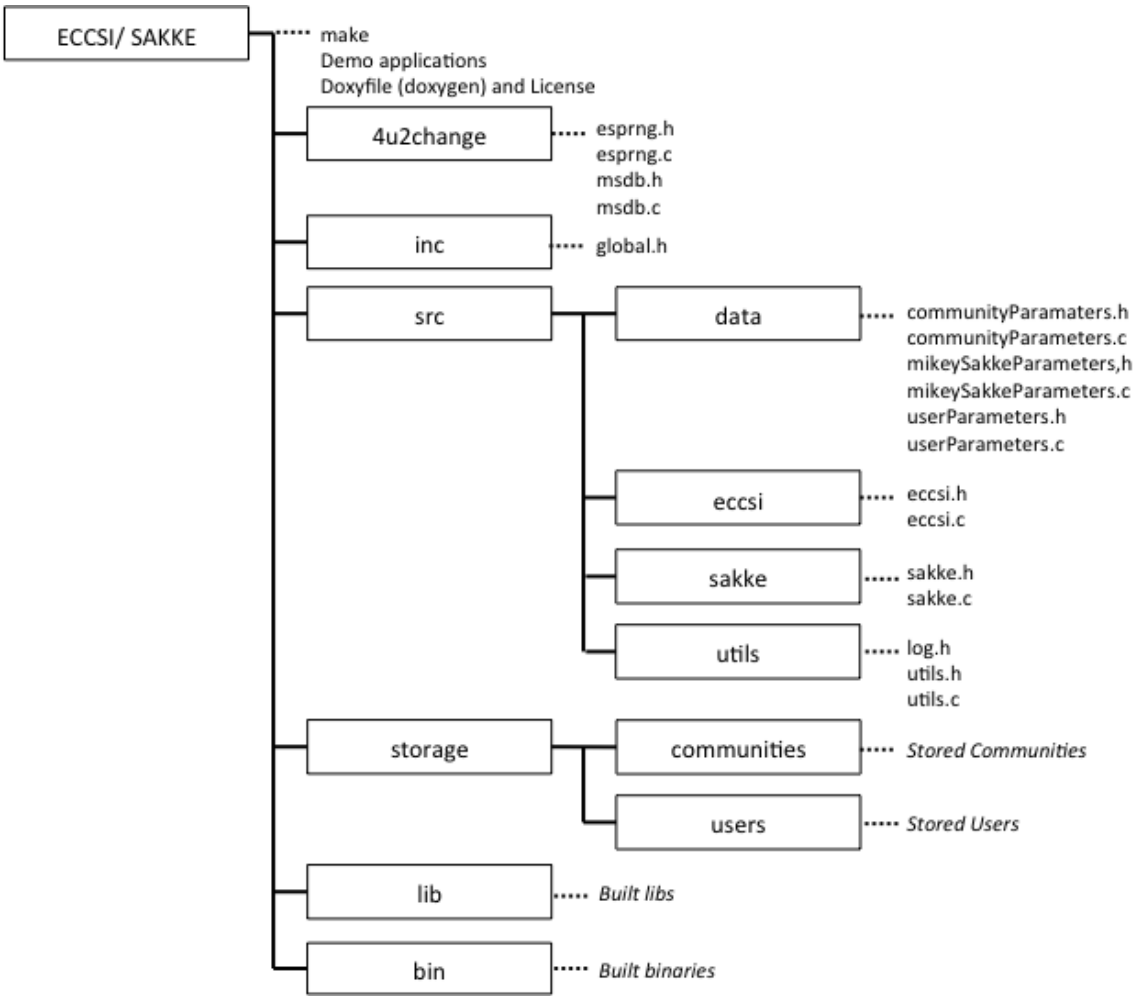
It would be **nice** if people writing these Applications, *PRNGs* or *Databases* gave some consideration to giving something back to the community as *plugins* e.g.

- Here's an SQL or LDAP implementation of the ECCSI/ SAKKE *MS DB* compliant to the ECCSI/SAKKE library MS DB API.
  or,
- Here's PRNG that conforms to standard **X** for use with the ECCSI/ SAKKE crypto library.*
  or,
- Here's an implementation to handle transmission and receipt of ECCSI/ SAKKE data provided by *this* project in Mikey (I_Messages as per RFC 6509), or, whatever protocol will be used for other applications, like email etc.

This would allow people to plug various aspects of a system together as required and maybe lead to faster take up/ adoption and a richer set of inter-operating services based on Mikey-Sakke.

*Obviously, there may be some difficulty with specific PRNG implementations, but just asking that you consider it.

3.3. Directory/ File Layout



4. Building the project

Building on Linux. Development machine was CentOS 6.

After cloning the repository, the make script needs to be executable, so:

```
chmod 775 make
```

Then to make, run the simple make script:

```
./make
```

In order to run the demo program you will need to make the libraries location accessible. The easiest way to do this on linux systems is to add the libraries location to LD_LIBRARY_PATH as follows:

```
export LD_LIBRARY_PATH=$LD_LIBRARY_PATH:./lib
```

To run the program:

```
./es-demo-1
```

4.1. Display options

The demo program by default displays lots of DEBUG information, that links out to the relevant RFCs. If you want to turn this debug off, you should comment out the following line:

```
#define ES_OUTPUT_DEBUG
```

from:

```
src/utils/log.h
```

Then rebuild.

As is indicated in the previous Directory/ File Layout section, *community* and *user* information is stored in a directory name *storage*. If you want to have these files stored in another directory, you should modify:

```
STORAGE_ROOT
```

in

```
inc/globals.h
```

Then rebuild.

5. Doxygen

There is some limited doxygen documentation in the project. To access this you need to install doxygen.

On Linux, you need to install the following repositories (using yum, apt, whatever), or their equivalents. The following assumes CentOS 6:

- yum install graphviz
- yum install boost-graph
- yum install texlive
- yum install texlive-utils
- yum install doxygen

Then (from inside the top level directory, where you cloned this project):

```
doxygen Doxyfile
```

Next, open your web browser (on the same machine) and open file:

```
file://<path-to-this-dir>/doxygen_output/html/index.html
```

6. APIs

The software version can be retrieved using a call to:

```
char *softwareVersion(void)
```

There are three groups of data access APIs:

- ECCSI API
  ECCSI functions for Signing/ Verifying
- SAKKE API
  SAKKE functions for Encryption/ Decryption

- MS Database API
  MS Database Functions for storing/ retrieving Mikey-Sakke, community and user data.

6.1. ECCSI API

- 6.1.1. ECCSI Sign
- 6.1.2. ECCSI Verify
- 6.1.3. Validate SSK

###6.1.1. ECCSI Sign Create an ECCSI signature for the specified message.

```
uint8_t eccsi_sign(
    const uint8_t  *message,
    const size_t    message_len,
    const uint8_t  *user_id,
    const size_t    user_id_len,
    const uint8_t  *community,
    const uint8_t  *j_random,
    const size_t    j_random_len,
    uint8_t       **signature,
    size_t         *signature_len);
```

###6.1.2. ECCSI Verify Verify a signature following the actions described in section 5.2.2 of RFC 6507.

```
uint8_t eccsi_verify(
    const uint8_t  *message,
    const size_t    message_len,
    const uint8_t  *signature,
    const size_t    signature_len,
    const uint8_t  *userId,
    const size_t    userIdLength,
    const uint8_t  *community);
```

###6.1.3. Validate SSK Validate the SSK (Secret Signing Key) provided by the KMS (Key Management Server) for use by this user. See Section 6.1.2 (para 2) of RFC6508.

```
uint8_t eccsi_validateSSK(
    const uint8_t  *user_id,
    const size_t    user_id_len,
    const uint8_t  *community,
    const uint8_t  *SSK,
    const size_t    SSK_len,
    const uint8_t  *KPAK,
```

```
    const size_t    KPAK_len,
    const uint8_t  *PVT,
    const size_t    PVT_len,
    uint8_t       **hash,
    size_t         *hash_len);
```

##6.2. SAKKE API

- 6.2.1. Create SED
- 6.2.2. Extract SSV
- 6.2.3. Validate SSK

###6.2.1. Create SED Create SED (Sakke Encapsulated Data) that includes SSV (Shared Secret Value). Described in RFC6508 Section 6.2.1.

```
uint8_t sakke_generateSakkeEncapsulatedData(
    uint8_t       **encapsulated_data,
    size_t         *encapsulated_data_len,
    const uint8_t  *user_id,
    const size_t    user_id_len,
    const uint8_t  *community,
    const uint8_t  *ssv,
    const size_t    ssv_len);
```

###6.2.2. Extract SSV Extract SSV (Shared Secret Value) from SED (Sakke Encapsulated Data). Described in Section 6.2.2 of RFC 6508.

```
uint8_t sakke_extractSharedSecret(
    const uint8_t  *SED,
    const size_t    SEDLength,
    const uint8_t  *userId,
    const size_t    userIdLength,
    const uint8_t  *community,
    uint8_t        **ssv,
    size_t          *ssvLength);
```

###6.2.3. Validate RSK Validate the RSK (Receiver Secret Key) provided by the KMS (Key Management Server) for use by this user. See Section 6.1.2 (para 2) of RFC6508.

```
uint8_t sakke_validateRSK(
    const uint8_t *user_id,
    const size_t   user_id_len,
    const uint8_t *community,
    const uint8_t *RSK,
    const size_t   RSK_len);
```

##6.3. MS Database API These are the functions you will need to reproduce if you write your own Database storage mechanism.

###6.3.1. Community Management

- 6.3.1.1. Community Add
- 6.3.1.2. Community Exists
- 6.3.1.3. Community Delete
- 6.3.1.4. Community Purge
- 6.3.1.5. Community List
- 6.3.1.6. Community Count

####6.3.1.1. Community Add Add KMS certificate data for a new KMS (community). If the kms_uri (community) name exists the storage is deleted first.

```
short msdb_communityAdd(
    const uint8_t *version,
    const uint8_t *cert_uri,
    const uint8_t *kms_uri, /* AKA community */
    const uint8_t *issuer,
    const uint8_t *valid_from,
    const uint8_t *valid_to,
    const short    revoked,
    const uint8_t *user_id_format, /* Optional */
    const uint8_t *pub_enc_key,
    const size_t   pub_enc_key_len,
    const uint8_t *pub_auth_key,
    const size_t   pub_auth_key_len,
    const uint8_t *kms_domain_list);
```

####6.3.1.2. Community Exists Check whether the specified community exists.

```
short    msdb_communityExists(
    const uint8_t *community);
```

####6.3.1.3. Community Delete Delete specified community.

```
short    msdb_communityDelete(
    const uint8_t *community);
```

####6.3.1.4. Community Purge Delete all (purge) stored communities.

```
short    msdb_communityPurge();
```

####6.3.1.5. Community List Get a CSV (Comma Separated Value) list of stored communities.

```
uint8_t *msdb_communityList();
```

####6.3.1.6. Community Count The number of stored communities.

```
uint16_t msdb_communityCount();
```

###6.3.2. Community Get Functions Retrieve stored Secure Chorus attributes.

- 6.3.2.1. Community Get Version
- 6.3.2.2. Community Get Cert URI
- 6.3.2.3. Community Get KMS URI
- 6.3.2.4. Community Get Issuer
- 6.3.2.5. Community Get Valid From
- 6.3.2.6. Community Get Valid To
- 6.3.2.7. Community Get Revoked
- 6.3.2.8. Community Get User ID Format
- 6.3.2.9. Community Get Pub Auth Enc Key
- 6.3.2.10. Community Get Pub Auth Key
- 6.3.2.11. Community Get KMS Domain List

####6.3.2.1. Community Get Version Get the stored version for the specified community.

```
short msdb_communityGetVersion(
    const uint8_t *community,
    uint8_t       *version);
```

####6.3.2.2. Community Get Cert URI Get the stored CertUri for the specified community.

```
short msdb_communityGetCertUri(
    const uint8_t *community,
    uint8_t       *cert_uri);
```

####6.3.2.3. Community Get KMS URI Get the stored KmUri for the specified community.

```
short msdb_communityGetKmsUri(
    const uint8_t *community,
    uint8_t       *kms_uri);
```

####6.3.2.4. Community Get Issuer Get the stored Issuer for the specified community.

```
short msdb_communityGetIssuer(
    const uint8_t *community,
    uint8_t       *issuer);
```

####6.3.2.5. Community Get Valid From Get the stored ValidFrom for the specified community.

```
short msdb_communityGetValidFrom(
    const uint8_t *community,
    uint8_t       *valid_from);
```

####6.3.2.6. Community Get Valid To Get the stored ValidTo for the specified community.

```
short msdb_communityGetValidTo(
    const uint8_t *community,
    uint8_t       *valid_to);
```

####6.3.2.7. Community Get Revoked Get the stored Revoked indicator for the specified community.

```
short msdb_communityGetRevoked(
    const uint8_t *community,
```

```
    short         *revoked);
```

####6.3.2.8. Community Get User ID Format Get the stored UserIdFormat for the specified community.

```
short msdb_communityGetUserIDFormat(
    const uint8_t *community,
    uint8_t       *user_id_format);
```

####6.3.2.9. Community Get Pub Enc Key Get the stored PubEncKey (Z) for the specified community.

```
short msdb_communityGetPubEncKey(
    const uint8_t *community,
    uint8_t       *Z);
```

####6.3.2.10. Community Get Pub Auth Key Get the stored PubAuthKey (KPAK) for the specified community.

```
short msdb_communityGetPubAuthKey(
    const uint8_t *community,
    uint8_t       *KPAK);
```

####6.3.2.11. Community Get KMS Domain List Get the stored KmsDomainList for the specified community.

```
short msdb_communityGetKmsDomainList(
    const uint8_t *community,
    uint8_t       *domain_list);
```

###6.3.3. User Management

- 6.3.3.1. User Add
- 6.3.3.2. User Exists
- 6.3.3.3. User Delete
- 6.3.3.4. User Purge
- 6.3.3.5. User List
- 6.3.3.6. User Count

####6.3.3.1. User Add Add a user to the user store. If the user exists already that data entry is deleted first.

```
uint8_t msdb_userAdd(
    const uint8_t *user_id,
    const size_t   user_id_len,
    const uint8_t *community,
    const uint8_t *ssk,
    const size_t   ssk_len,
    const uint8_t *rsk,
    const size_t   rsk_len,
    const uint8_t *hash,
    const size_t   hash_len,
    const uint8_t *pvt,
    const size_t   pvt_len);
```

####6.3.3.2. User Exists Check whether the specified user exists.

```
short    msdb_userExists(
    const uint8_t *user,
    const size_t   user_len,
    const uint8_t *community);
```

####6.3.3.3. User Delete Delete a specified user within a community.

```
short    msdb_userDelete(
    const uint8_t *user,
    const size_t   user_len,
    const uint8_t *community);
```

####6.3.3.4. User Purge Delete all (purge) stored users.

```
short    msdb_userPurge();
```

####6.3.3.5. User List Get a CSV (Comma Separated Value) list of stored users.

```
uint8_t *msdb_userList();
```

####6.3.3.6. User Count Get the number of stored users.

```
uint16_t msdb_userCount();
```

###6.3.4. User Get Functions

- 6.3.4.1. User Get SSK
- 6.3.4.2. User Get RSK
- 6.3.4.3. User Get Hash
- 6.3.4.4. User Get PVT

####6.3.4.1. User Get SSK Get a specified user's SSK (Secret Signing Key).

```
short    msdb_userGetSSK(
    const uint8_t *user_id,
    const size_t   user_id_len,
    const uint8_t *community,
    uint8_t       *ssk);
```

####6.3.4.2. User Get RSK Get a specified user's RSK (Receiver Secret Key).

```
short    msdb_userGetRSK(
    const uint8_t *user_id,
    const size_t   user_id_len,
    const uint8_t *community,
    uint8_t       *rsk);
```

####6.3.4.3. User Get Hash Get a specified user's Hash.

```
short    msdb_userGetHash(
    const uint8_t *user_id,
    const size_t   user_id_len,
    const uint8_t *community,
    uint8_t       *hash);
```

####6.3.4.4. User Get PVT Get a specified user's PVT (Public Validation Token).

```
short    msdb_userGetPVT(
    const uint8_t *user_id,
    const size_t   user_id_len,
    const uint8_t *community,
    uint8_t       *pvt);
```

#7. To do Pretty much there I think.

- Peer review and testing please, anyone?
- There's some magic numbers to quash.
- There's still a few return values to check.
- Add ec-demo-2 a demonstration not using RFC values and with Alice and Bob using different ID's. I know this works, as I've tested it. I just haven't checked in the code yet... **DONE!**
- Presently writing the Key Mat generator (as *might* exist in a **KMS**), so that users can generate their own Key Mat for new *Communities* and *Users* and use it in this code... **DONE!**

#8. Contact details jim<-AT->mikey-sakke.org

▸ **Pages** 2

Contents

1. Project overview
   1.1. High Level System View
   1.2. Restrictions on use
   1.3. Other Implementations
2. Mikey Sakke Overview
3. Architecture
   3.1. Design Decisions
   3.2. Software Structure
4. Building
   4.1. Display Options
5. Doxygen
6. APIs
   6.1. ECCSI API
      6.1.1. ECCSI Sign
      6.1.2. ECCSI Verify
      6.1.3. ECCSI Validate SSK
   6.2. SAKKE API
      6.2.1. SAKKE Create SED
      6.2.2. SAKKE Extract SSV
      6.2.3. SAKKE Validate RSK
   6.3. MS Database API
      6.3.1. Community Management
      6.3.2. Community Get Functions
      6.3.3. User Management
      6.3.3. User Get Functions
7. To Do

8. Contact details

Clone this wiki locally

```
https://github.com/jim-b/ECCSI-SAKKE.wiki.git
```