Azrieli
College of Engineering
Jerusalem

עזריאלי
מכללה אקדמית להנדסה
ירושלים

# אבטחת מידע ופרטיות במחשוב ענן

# Information Security and Privacy in cloud

## BetweenUs

## פרויקט גמר - הנדסת תוכנה

**מאת**

**לירן בן גידה**

**נדב לוצטו**

## תקציר:

חשיפת מידע פרטי במרשתת היא אחת מהבעיות הגדולות שאנו מתמודדים איתן כיום.
בזמן שהמידע שלנו "מוצפן" בשרת, אותה ישות שומרת לצד המידע המוצפן את מפתחות ההצפנה לפענוח המידע. המידע שלנו יכול להיחשף ללא שליטתנו כאשר ומנהל השרת הינו זדוני.
בפרויקט זה אנו מציעים יישום אלגוריתם להצפנה קבוצתית אשר מבוססת על אישורים.
אנו משתמשים במספר אלגוריתמים קריפטוגרפים כמו AES, RSA והאלגוריתם SHAMIR SECRET SHARING.

הפתרון שאנו מציעים ימנע מאותו מנהל שרת זדוני לפענח את המידע שנמצא בשרת וזאת על ידי שימור המידע המוצפן בשרת ללא המפתחות לפענוח המידע. בנוסף לכך, המפתחות נשמרים בצורה מבוזרת אצל הלקוחות שהמשתמש היוזם (יוצר המידע) סומך עליהם.
המידע אשר נשמר בצד השרת בצורה מוצפנת וניתן לפענוח על ידי הסכמה של K משתמשים אשר המשתמש היוזם הגדיר מראש.

הטכנולוגיות אשר בחרנו להשתמש בהם לצורך הקמת השרת, צד הלקוח ואחסון הנתונים הינן טכנולוגיות חדשות אשר העניקו לנו אפשרות לבצע יישום קל ונוח יותר לפתרון שהצענו.
הטכנולוגיות והשירותים בהם השתמשנו: Cloudant, Node.JS, React-Native

## Summary:

One of the issues faced today is the exposure of our data online. while our data may be encrypted on the cloud, the keys for said data is also held by another entity (mostly the service provider), which leaves the data exposed in case of a malicious database administrator, for example.

BetweenUs project proposes implementation of an algorithm / workflow for a secure confirmation-based multi-party secret sharing we developed.

Our proposed solution takes the ability to access the encrypted data that is located on the cloud from the service provider and moves it into the client's hand, and it achieves that by saving the encrypted data on the cloud, but instead of also storing the keys on the cloud, it distributes the keys for decryption among several trusted clients, and once enough clients has given their permission to decrypt the secret, only then can the key for decryption be reconstructed.

Several known crypto algorithms have been used to achieve that - RSA Public-key cryptography, AES Symmetric cryptography and Shamir Secret Sharing.
Frameworks and service providers used in the project are: Node.js, React Native, Cloudant (IBM's CouchDB DBaaS solution) and Bluemix (IBM) where the server is hosted.

**הצהרה:**

פרויקט גמר זה נעשה בהנחייתו ובהדרכתו של ד"ר ירון וינסברג עבור המחלקה להנדסת תוכנה, עזריאלי המכללה האקדמית להנדסה ירושלים.

חיבור זה מציג את עבודתנו הקבוצתית ומהווה חלק מדרישות המכללה לקבלת תואר ראשון בהנדסת תוכנה.

אנו, החתומים מטה מצהירים בזאת כי פרויקט הגמר וחיבור זה נעשה, נכתב ונערך על ידינו בלבד. כמו כן, פרויקט הגמר וספר הפרויקט נעשו על בסיס הנחייתו של המנחה האישי ד"ר ירון וינסברג.

לירן בן גידה: ____________

נדב לוצטו: ____________

## תודות:

- **תודה רבה לד"ר וינסברג** ירון על התמיכה, הליווי והנתינה הרבה לאורך השנה.

  ד"ר וינסברג ירון ליווה אותנו משלב הרעיון, הקדיש לנו מזמנו וחיבר אותנו עם גורמים רלבנטיים רבים אשר סייעו לנו מניסיונם ודעותיהם על מנת לממש את הרעיון.
- **תודה רבה לד"ר צור-דוד שמרית** על העזרה בתכנון אלגוריתמי ההצפנה
- **תודה רבה למר. נחמיאס נדב** על העזרה בפתרון בעיות ביישום ההצפנה הא-סימטרית
- **תודה רבה לד"ר יגל ראובן** על העזרה בליווי הפרויקט
- **תודה רבה לענת סיבוני** על העזרה בעריכה ובתרגום לאנגלית

# Table of content

## Definitions

- Shamir Secret Sharing: An algorithm that takes input of: a chunk of data (‘**secret’**), number of new ‘secret’ sized data to produce (‘**shares’**), and a threshold of shares needed to restore the original secret.

- Share: From Shamir Secret Sharing (‘**SSS’**) algorithm, a share is data produced by the algorithm. A share alone is meaningless data, but given enough shares (the threshold set when generating the shares), the original secret can be restored.

- RSA: Asymmetrical encryption algorithm (public-key cryptography). A key pair is generated, one key is used to encrypt the secret (private key), and only the other key generated can decrypt the secret (public key).

- AES: Symmetrical encryption algorithm, a common key is established for both encryption and decryption of a secret.

- Cloud Storage: A remote server(s) where the data is stored at.

- RESTful API: REST is an architectural style consisting of a coordinated set of components, connectors, and data elements within a system, where the focus is on component roles and a specific set of interactions between data elements rather than implementation details. Web service APIs that adhere to the REST architectural constraints are called **RESTful APIs**.
  **RESTful API** is an interface based on standard HTTP methods (e.g., OPTIONS, GET, PUT, POST, and DELETE) to access system resources (via URLs).

- Plain-text: Non-encrypted content.

- Secret / Cipher-text: An encrypted plain-text is called cipher-text / secret.

# Abstract

This project focuses on the privacy and security issue with Cloud Computing.

Lately, we've been witnessing a significant amount of various applications and services that include sharing information between groups of individuals through the cloud:

- Chats
- Files sharing
- Emails

These social services store and retain the information passed by and through them in plain non-encrypted manner, in such way that a malicious administrator in the service can peek into the data, thus risking the user security and privacy.

The initial stage of the project was developing an algorithm based on Shamir Secret sharing, RSA and AES, to provide cryptographic infrastructure for a service that was developed in the second stage of the project.

The second stage of the project was developing a cloud-based service, that provide developers of applications and other services a RESTful API to the algorithm developed at the first stage, in order to ensure the privacy and security of their users.

Some possible use-cases for the service:

- Distributed signature
- Group file sharing
- Acknowledgements based systems

The service as a whole, will be generic and provide a REST API of which the app developers, both private and enterprise, could use to increase the information security and privacy of their users.

At the early stage, we developed the algorithm which our service would be based on, and made a rough prototype together, once we achieved a basic functionality of the service, we split it up into two areas of responsibility and divided them arbitrarily, *Liran* took on the client side and *Nadav* took on the server side.

# Problem Description

## Problem Statement

In an age where vast and sensitive information changes hands between users through cloud-based service, there is a growing demand for methods that provide increased security and privacy, insuring minimal exposure of data to unauthorized parties/individuals.

In most cloud-based services, developers, system administrators, database administrators have full access to the raw data passed between users, which leave the user private data exposed and vulnerable, for example - an exchange of an explicit photo between two lovers.

These kind of issues create the demand for reliable services, which provides a solution for those issues.

## Software Development challenge

Supplying a service that provides the option of creating a shared secret, such that, in order to restore the secret, we will need the confirmation of K out of N participants (K is predefined at the secret creation stage, N is the total amount of participants).
To provide such feature, we chose to use the Shamir Secret Sharing algorithm.
Using this algorithm, for sharing secret M with N participants will produce N replications of a data the size of the original secret M, resulting in a traffic of N*SizeOf(M) data, causing a major waste of bandwidth.

Additionally, this service needs to include an API generic enough to ensure compatibility with most of the existing cloud-based social services.
Not only does the API needs to be generic enough, it also has to be developer-friendly to make integration our service into an existing platform with a minor issue.

## Proposed Solution

In this project, we developed a cloud-based service, which provides an additional layer in the exchange of information between users and supply them with means of securely sharing data.
This service will provide users with additional functionality when sharing data, for example - sharing a file between N contacts, and dictating that the file will only be decrypted when K out of the N contacts has consented to viewing the file.

### System architecture

The system is composed of a cloud-based service, a database and a client, where the client is a person who is interested in sharing a secret and/or be a participant in a group share between other users of the service.

The cloud-based service will be a controller / intermediator that connects one client to the rest of the participants in the secret, using the database as a persistent storage.

### System States

The system includes three main states - Encryption, Decryption and Commit.

**Encryption** – When user creates new transaction, he generates a symmetric key and encrypt the data with it. He sets the threshold in order to decrypt the secret and gives a name to the transaction he made. The symmetric key he generated, goes into Shamir Secret Sharing ("SSS") algorithm. In order to encrypt each share to the participants of the group, he queries the server to get their public keys and encrypt each share respectively.
When submitting the shares to the server, it creates a share stash for each participant in the group.

**Decryption** - A participant is interested in decrypting the secret in a transaction, he queries the server with the transaction he is interested in decrypting, the server responds with his share stash and the information for the transaction, which is the threshold needed for reconstructing the secret.
If the user does not have enough shares in his stash (shares < threshold), then the secret cannot be restored.
If user has enough shares (shares >= threshold), then he can resolve the secret using his private key to decrypt the shares in his stash.
The decrypted shares are then combined using SSS into the secret – the symmetric key relevant to the transaction.

The user can now query the server for the encrypted data, and decrypt with the symmetric key.

**Commit** – Within a transaction, users can request other users for their shares.
When a user is being asked for his share, he can either accept or decline.
Declining notifies the requesting user that his request has been declined.
Accepting a share triggers the following flow:

- Query the server for the committing user share.
- Query the server for the requesting user public key.
- The committing user decrypts his own share with his private key, and encrypts it again using the requesting user public key.
- The committing user then commit the encrypted share to the server.
- The server places the committing user encrypted* share, in the requesting user stash**.

After the flow has ended, the requesting user will have the share he requested, encrypted with his own public key.

** using the requesting user public key.*
*** The stash that’s associated with the transaction.*

## Solution Description

To solve the bandwidth waste problem where we replicate the data N times, we will use the following algorithms:
Shamir Secret Sharing, Symmetric Cryptography, Asymmetric Cryptography (Public-key Cryptography).

The secret message will be encrypted with the symmetric key.
A new symmetric key is generated for each transaction and is split into N shares with a threshold of K using Shamir Secret Sharing ("SSS").
N is the number of participants of the group.
K is the number of shares needed to reconstruct the secret.

We avoid the bandwidth waste by using SSS on the symmetric key and not the actual data, thus reducing N*SizeOf (Data) to N*SizeOf (Symmetric Key).

All the shares produced by SSS will be encrypted using the group participants' public keys.

The purpose of introducing the asymmetric cryptography is encrypting the share for each participant before committing it to the server.
This ensures that the server cannot restore the symmetric key and peek at the encrypted message.

The motivation behind using private-public key-pairs:

- We ensure the server is technically incapable of decrypting the data.
- Each participant receives his own share, encrypted with his own public key.
- The data is on the cloud, encrypted with a symmetric key generated by the transaction initiator.

## Solution’s Tools and Technologies

The tools, algorithms and technologies we will use in our project:

Server\Client Side:

- Communication: RESTful API

Server Side:

- Languages: Javascript.
- Frameworks: Node.js.
- Cloud Service:
  - IBM’s Bluemix.
  - IBM’s Cloudant, DBaaS (CouchDB).

Client Side:

- Algorithms:
  - Asymmetric Cryptography (RSA).
  - Symmetric Encryption Cryptography (AES-256).
  - Shamir Secret Sharing.
- Languages: JavaScript.
- Framework: React-Native (Cross-platform mobile development platform)

## Tests

The projects essential algorithms are located on the client-side.
Since the client-side generates the symmetric key, encrypts the secret and split the symmetric key into shares and then encrypts it with the other participant's public key, we find it crucial to write a unit test to ensure our module integrity.
Our unit test examines every function written in our module, BetweenUs.JS and is written in JavaScript.

Shamir Secret Sharing unit tests:

- Split data into n = random(2 to 255) shares, k = random(2 to n-1), and try to reconstruct the secret with amount of shares that is smaller than the threshold k.
- Split data into n = random(2 to 255) shares, k = random(2 to n-1), and try to reconstruct the secret with amount of shares that equals to the threshold k.
- Split data into n = random(2 to 255) shares, k = random(2 to n-1), and try to reconstruct the secret with amount of shares that is larger than the threshold k.

Symmetric Key unit test:

- Generate a symmetric key, encrypt data with it, and see that when we decrypt it, we restore the original data.

Asymmetric Key unit tests:

- Generate a public-private key-pair.
- Encrypt data with public key, decrypt with private key, and check it is equal to the original data.

Server API unit tests:

- Construct a series of unit tests to the REST API of the server, and verify that we get the intended responses.

## Literature Comparison

This section contains comparisons with other secure end-to-end projects.

**OTR (Off-The-Record)** communication protocol is based on AES and SHA-1 hash function.
The protocols purpose is to provide deniable authentication between conversation participants.
Although the protocol supports mutual authentication between users, it does not support multi-user group chat (As of 2009).

Extension of the above algorithm named Multi-party OTR.

An example to a service that implements the OTR protocol is **Signals** chat.
Signals chat is a free and open-source encrypted voice-call and instant messaging application that provides end-to-end secured communication between two users.
The application based on the OTR protocol with some improvement such as deniability and forward secrecy to ensure the integrity of communication between the end-point users.

Our project purpose is to create a service that offers developers the option to use a multi-party secured communication between the end users using our algorithm.

## Project Management Tools

| # | System | Location |
|---|---|---|
| 1 | Code repository | https://github.com/liranbg/BetweenUs |
| 2 | Calendar | https://trello.com/b/nJPCPDXT/betweenus |
| 3 | Project management | https://waffle.io/liranbg/BetweenUs |
| 4 | Prototype demo video | https://vimeo.com/152884344<br>password: betweenus |

## Conclusions

- The design of the workflow could have been more simple. for example, there is no need for a group entity to accommodate transactions between clients.

- Picking a more mature language and framework for the server side, while Node.js is very easy to work with, but once the projects codebase became pretty big, some problems start to surface that are mainly caused by the lack of maturity of Node.js. For example, Node.js server is implemented in an event-driven manner, which causes deep nesting of functionality that makes the code and the flow difficult to read and debug (also known as 'Callback Hell').

- While we overcame this issue by introducing new methodology using 'Promises', we spent tremendous effort in converting the server side from using the old methodology of callbacks into using the Promises concept.

- Following the previous point, since Node.js and Javascript are at the bleeding edge of current internet technology and are being rapidly developed, we've run into some infrastructure providers that did not support some functionality our code used, since the functionality we used has been recently implemented in the language, and some service provider has not yet upgraded to the latest Javascript versions.

- On the client side – we chose React-Native over Xamarin\Native and more due to reason we wanted to use new technologies. The fact that React Native is still not mature enough caused us spending a lot of hours to find bugs in the system. As react-native releases new version each week, we had to upgrade in order to fix those bugs.

# Appendix

## Bibliography

- “How to share a secret” by Adi Shamir 1979 - link
- Shamir Secret Sharing by Amos Beimel - link
- Lagrange Interpolation Polynomial – the explanation behind Shamir’s SS algorithm – link

## Data Flow and Diagrams

### Terminology

SSS – Shamir Secret Sharing

Shares – SSS output components needed to restore the SSS input data.

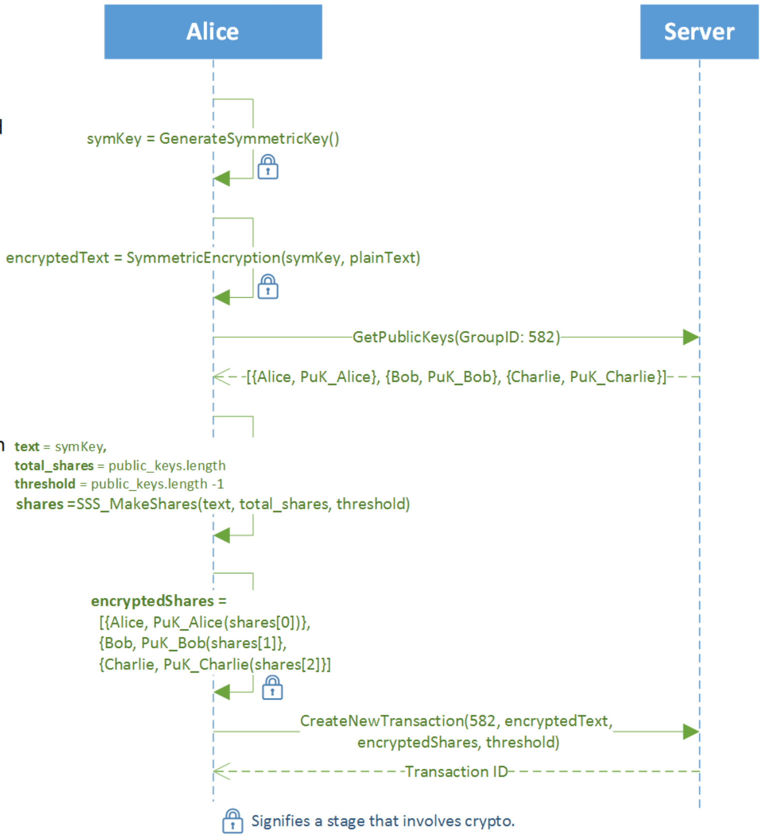
PuK_Alice – Asymmetrical public key of participant Alice.

PrK_Alice – Asymmetrical private key of participant Alice.

Transaction – An exchange of a secret message within an established group.

Group – One or more participants known to the main server grouped together.

### Encryption Sequence Diagram


Alice
Server
symKey = GenerateSymmetricKey()
encryptedText = SymmetricEncryption(symKey, plainText)
GetPublicKeys(GroupID: 582)
[{Alice, PuK_Alice}, {Bob, PuK_Bob}, {Charlie, PuK_Charlie}]
text = symKey,
total_shares = public_keys.length
threshold = public_keys.length -1
shares =SSS_MakeShares(text, total_shares, threshold)
encryptedShares =
[{Alice, PuK_Alice(shares[0])},
{Bob, PuK_Bob(shares[1]},
{Charlie, PuK_Charlie(shares[2]}]
CreateNewTransaction(582, encryptedText,
encryptedShares, threshold)
Transaction ID
Signifies a stage that involves crypto.

## Terminology

SSS – Shamir Secret Sharing
Shares – SSS output components needed to restore the SSS input data.

Sym_X – Symmetrical Key X

PuK_Alice – Asymmetrical public key of participant Alice.

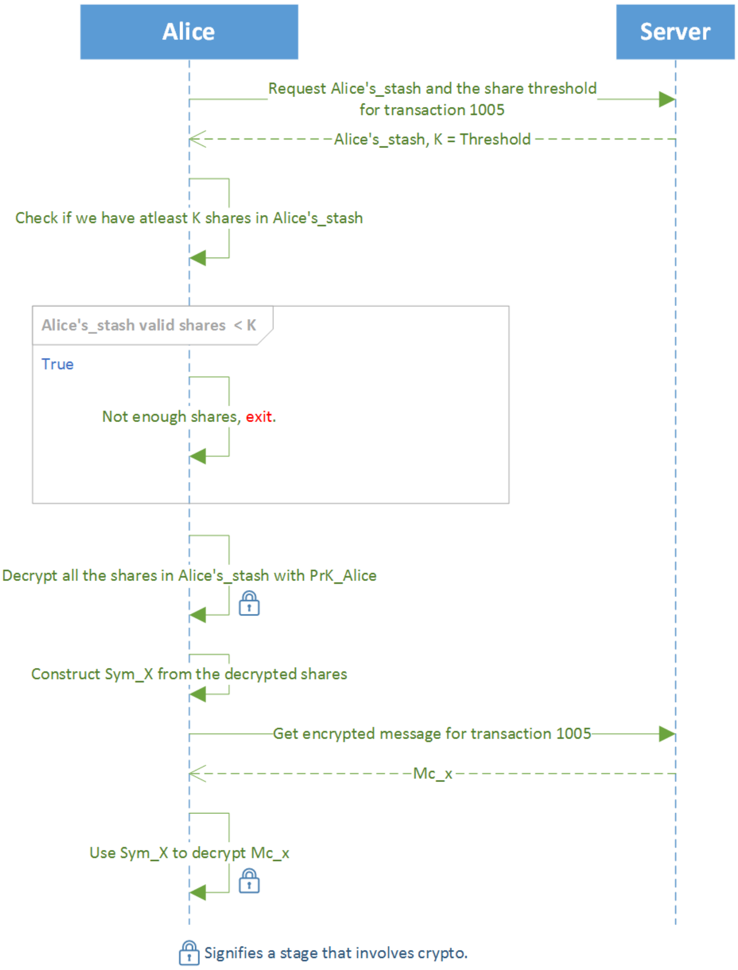
PrK_Alice – Asymmetrical private key of participant Alice.

M_a – Plain message a.

Mc_a – Encrypted message a.

Stash_Alice – Participant Alice stash that consists of all the PuK_Alice encrypted shares that has been submitted to the group.

## Decryption Sequence Diagram


Alice
Server
Request Alice's_stash and the share threshold for transaction 1005
Alice's_stash, K = Threshold
Check if we have atleast K shares in Alice's_stash
Alice's_stash valid shares < K
True
Not enough shares, exit.
Decrypt all the shares in Alice's_stash with PrK_Alice
Construct Sym_X from the decrypted shares
Get encrypted message for transaction 1005
Mc_x
Use Sym_X to decrypt Mc_x
Signifies a stage that involves crypto.

### Terminology

SSS – Shamir Secret Sharing
Shares – SSS output components needed to restore the SSS input data.

Sym_X – Symmetrical Key X

PuK_Alice – Asymmetrical public key of participant Alice.

PrK_Alice – Asymmetrical private key of participant Alice.

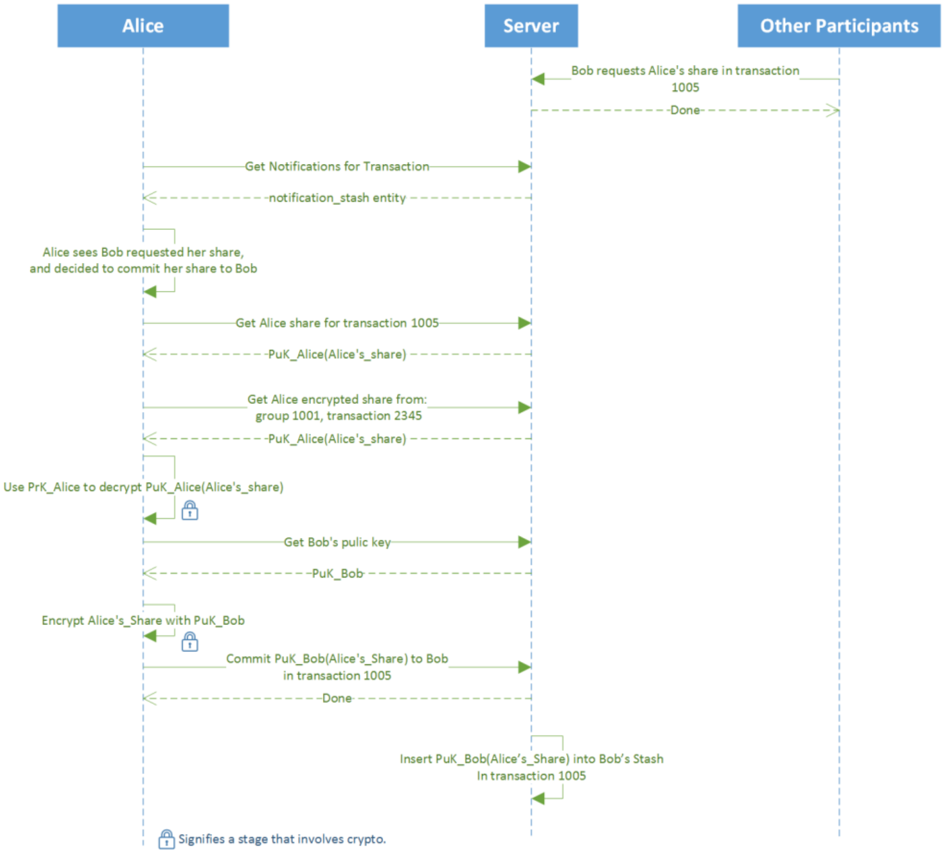
M_a – Plain message a.

Mc_a – Encrypted message a.

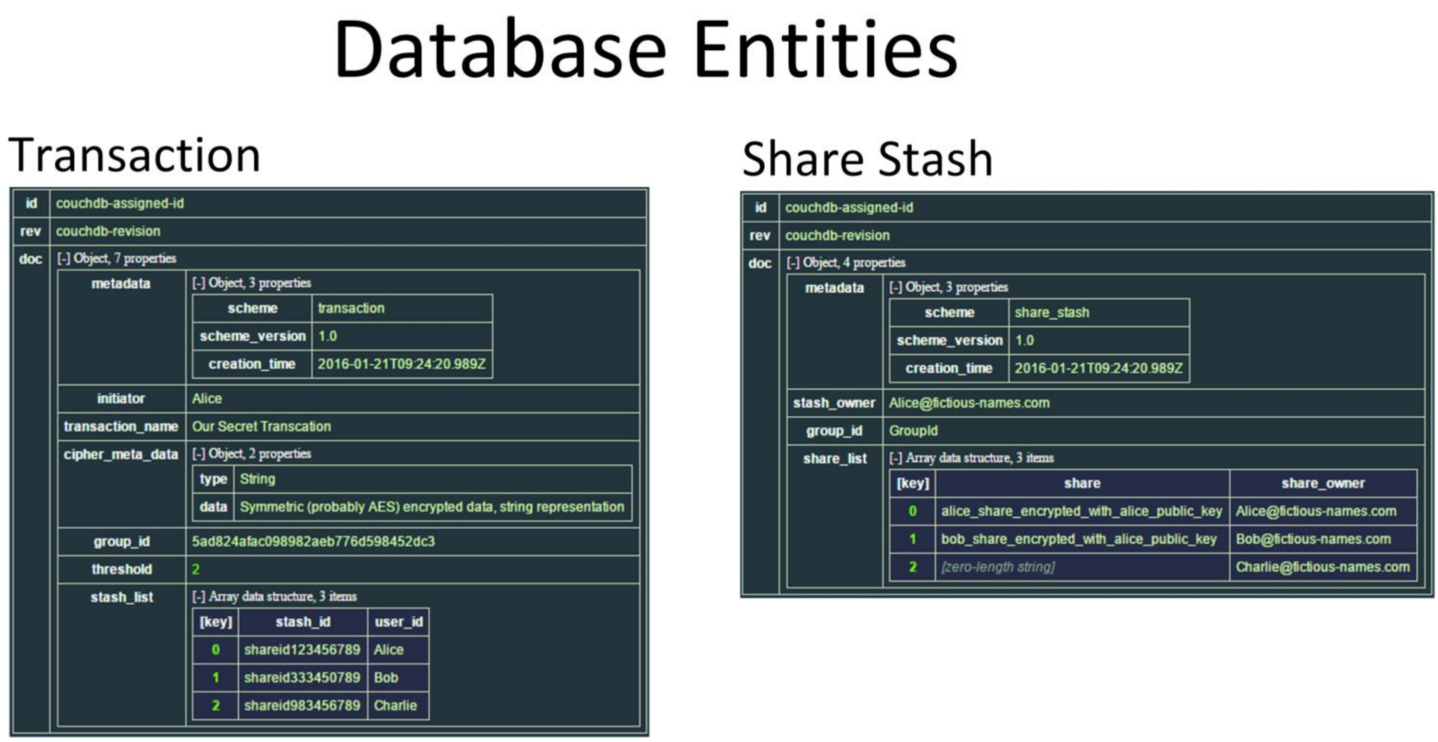
Share_Alice – The share of Participant A in the secret

Stash_Alice – Participant Alice stash that consists of all the PuK_Alice encrypted shares that has been submitted to the group

### Committing a share


Alice
Server
Other Participants
Bob requests Alice's share in transaction 1005
Done
Get Notifications for Transaction
notification_stash entity
Alice sees Bob requested her share, and decided to commit her share to Bob
Get Alice share for transaction 1005
PuK_Alice(Alice's_share)
Get Alice encrypted share from: group 1001, transaction 2345
PuK_Alice(Alice's_share)
Use PrK_Alice to decrypt PuK_Alice(Alice's_share)
Get Bob's pulic key
PuK_Bob
Encrypt Alice's_Share with PuK_Bob
Commit PuK_Bob(Alice's_Share) to Bob in transaction 1005
Done
Insert PuK_Bob(Alice's_Share) into Bob's Stash In transaction 1005
Signifies a stage that involves crypto.

## Database diagrams



# Database Entities

## Transaction

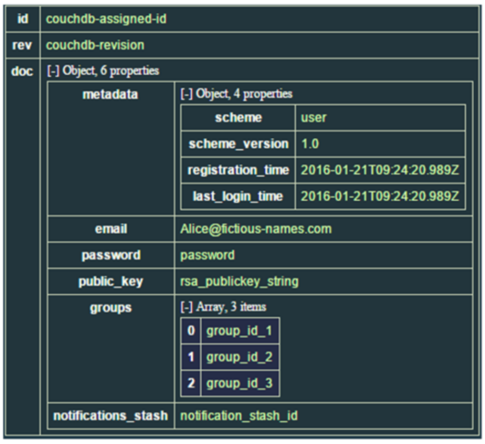
<table>
<tr><td>id</td><td>couchdb-assigned-id</td></tr>
<tr><td>rev</td><td>couchdb-revision</td></tr>
<tr><td>doc</td><td>[-] Object, 7 properties
<table>
<tr><td>metadata</td><td>[-] Object, 3 properties
<table>
<tr><td>scheme</td><td>transaction</td></tr>
<tr><td>scheme_version</td><td>1.0</td></tr>
<tr><td>creation_time</td><td>2016-01-21T09:24:20.989Z</td></tr>
</table></td></tr>
<tr><td>initiator</td><td>Alice</td></tr>
<tr><td>transaction_name</td><td>Our Secret Transcation</td></tr>
<tr><td>cipher_meta_data</td><td>[-] Object, 2 properties
<table>
<tr><td>type</td><td>String</td></tr>
<tr><td>data</td><td>Symmetric (probably AES) encrypted data, string representation</td></tr>
</table></td></tr>
<tr><td>group_id</td><td>5ad824afac098982aeb776d598452dc3</td></tr>
<tr><td>threshold</td><td>2</td></tr>
<tr><td>stash_list</td><td>[-] Array data structure, 3 items
<table>
<tr><th>[key]</th><th>stash_id</th><th>user_id</th></tr>
<tr><td>0</td><td>shareid123456789</td><td>Alice</td></tr>
<tr><td>1</td><td>shareid333450789</td><td>Bob</td></tr>
<tr><td>2</td><td>shareid983456789</td><td>Charlie</td></tr>
</table></td></tr>
</table></td></tr>
</table>

## Share Stash

<table>
<tr><td>id</td><td>couchdb-assigned-id</td></tr>
<tr><td>rev</td><td>couchdb-revision</td></tr>
<tr><td>doc</td><td>[-] Object, 4 properties
<table>
<tr><td>metadata</td><td>[-] Object, 3 properties
<table>
<tr><td>scheme</td><td>share_stash</td></tr>
<tr><td>scheme_version</td><td>1.0</td></tr>
<tr><td>creation_time</td><td>2016-01-21T09:24:20.989Z</td></tr>
</table></td></tr>
<tr><td>stash_owner</td><td>Alice@fictious-names.com</td></tr>
<tr><td>group_id</td><td>GroupId</td></tr>
<tr><td>share_list</td><td>[-] Array data structure, 3 items
<table>
<tr><th>[key]</th><th>share</th><th>share_owner</th></tr>
<tr><td>0</td><td>alice_share_encrypted_with_alice_public_key</td><td>Alice@fictious-names.com</td></tr>
<tr><td>1</td><td>bob_share_encrypted_with_alice_public_key</td><td>Bob@fictious-names.com</td></tr>
<tr><td>2</td><td>[zero-length string]</td><td>Charlie@fictious-names.com</td></tr>
</table></td></tr>
</table></td></tr>
</table>

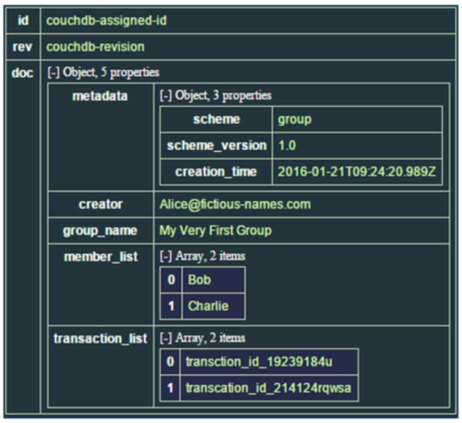
# Database Entities

## User



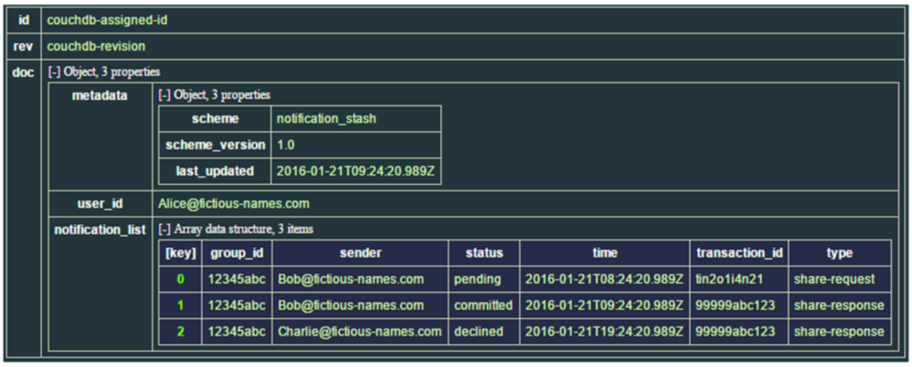
<table>
<tr><td>id</td><td>couchdb-assigned-id</td></tr>
<tr><td>rev</td><td>couchdb-revision</td></tr>
<tr><td>doc</td><td>[-] Object, 6 properties
<table>
<tr><td>metadata</td><td>[-] Object, 4 properties
<table>
<tr><td>scheme</td><td>user</td></tr>
<tr><td>scheme_version</td><td>1.0</td></tr>
<tr><td>registration_time</td><td>2016-01-21T09:24:20.989Z</td></tr>
<tr><td>last_login_time</td><td>2016-01-21T09:24:20.989Z</td></tr>
</table></td></tr>
<tr><td>email</td><td>Alice@fictious-names.com</td></tr>
<tr><td>password</td><td>password</td></tr>
<tr><td>public_key</td><td>rsa_publickey_string</td></tr>
<tr><td>groups</td><td>[-] Array, 3 items
<table>
<tr><td>0</td><td>group_id_1</td></tr>
<tr><td>1</td><td>group_id_2</td></tr>
<tr><td>2</td><td>group_id_3</td></tr>
</table></td></tr>
<tr><td>notifications_stash</td><td>notification_stash_id</td></tr>
</table></td></tr>
</table>

## Group



<table>
<tr><td>id</td><td>couchdb-assigned-id</td></tr>
<tr><td>rev</td><td>couchdb-revision</td></tr>
<tr><td>doc</td><td>[-] Object, 5 properties
<table>
<tr><td>metadata</td><td>[-] Object, 3 properties
<table>
<tr><td>scheme</td><td>group</td></tr>
<tr><td>scheme_version</td><td>1.0</td></tr>
<tr><td>creation_time</td><td>2016-01-21T09:24:20.989Z</td></tr>
</table></td></tr>
<tr><td>creator</td><td>Alice@fictious-names.com</td></tr>
<tr><td>group_name</td><td>My Very First Group</td></tr>
<tr><td>member_list</td><td>[-] Array, 2 items
<table>
<tr><td>0</td><td>Bob</td></tr>
<tr><td>1</td><td>Charlie</td></tr>
</table></td></tr>
<tr><td>transaction_list</td><td>[-] Array, 2 items
<table>
<tr><td>0</td><td>transction_id_19239184u</td></tr>
<tr><td>1</td><td>transcation_id_214124rqwsa</td></tr>
</table></td></tr>
</table></td></tr>
</table>

## Notification Stash



<table>
<tr><td>id</td><td>couchdb-assigned-id</td></tr>
<tr><td>rev</td><td>couchdb-revision</td></tr>
<tr><td>doc</td><td>[-] Object, 3 properties
<table>
<tr><td>metadata</td><td>[-] Object, 3 properties
<table>
<tr><td>scheme</td><td>notification_stash</td></tr>
<tr><td>scheme_version</td><td>1.0</td></tr>
<tr><td>last_updated</td><td>2016-01-21T09:24:20.989Z</td></tr>
</table></td></tr>
<tr><td>user_id</td><td>Alice@fictious-names.com</td></tr>
<tr><td>notification_list</td><td>[-] Array data structure, 3 items
<table>
<tr><th>[key]</th><th>group_id</th><th>sender</th><th>status</th><th>time</th><th>transaction_id</th><th>type</th></tr>
<tr><td>0</td><td>12345abc</td><td>Bob@fictious-names.com</td><td>pending</td><td>2016-01-21T08:24:20.989Z</td><td>tin2o1i4n21</td><td>share-request</td></tr>
<tr><td>1</td><td>12345abc</td><td>Bob@fictious-names.com</td><td>committed</td><td>2016-01-21T09:24:20.989Z</td><td>99999abc123</td><td>share-response</td></tr>
<tr><td>2</td><td>12345abc</td><td>Charlie@fictious-names.com</td><td>declined</td><td>2016-01-21T19:24:20.989Z</td><td>99999abc123</td><td>share-response</td></tr>
</table></td></tr>
</table></td></tr>
</table>

## Project Planning

| Date | Description |
|---|---|
| **22.11.15** | Proposal Submission |
| **1.12.15** | Server\Client side – Build main infrastructure |
| **16.12.15** | Server\Client side – Build algorithms and Interfaces |
| **31.12.15** | Connect between Client and Server side (Make API) |
| **16.1.16** | Server side - Connect with a cloud service |
| **24.1.16** | Prototype submission |
| **28.2.16** | Client side – develop application |
| **30.3.16** | Server side – Error and exceptions handling |
| **5.5.16** | Construction submission |
| **19.6.16** | Delivery submission |
| **7.7.16** | Final submission |

## Risks Table

| Risks | The severity of the impact on the project | Probability that the risk will occur | What steps will performed lowering risk | If the risk is realized |
|---|---|---|---|---|
| MITM by a Malicious Server. | A malicious server might upon registration of each client create an additional pair of secret/public key and use it to perform a 'Man in the Middle' attack to the encrypted data. | **Highly** unlikely. | Clients should only use trusted servers. | **The symmetric key for the decryption of the secret will be exposed.** |
| Denial of Service by Injection of Malicious Shares. | A malicious client might commit a faulty share instead of his actual share, thus causing the participants of the group produce bad result when restoring the secret. | Unlikely. | The damage can't be mitigated, just minimized. Steps will be taken to identify bad secret resolution. | **The encrypted data might the group be lost - depends on the threshold of creation of the secret and the number malicious clients within the group.** |
| Vulnerabilities in the RSA Crypto. | If such scenario occurs and the RSA is rendered useless - a malicious client can obtain all the encrypted shares and the public keys of the participants, decrypt them, and construct the symmetrical key and decrypt the cipher text of the group. | **Highly** unlikely. | None. | **Find another crypto to use.** |
| Cloudant is permanently down | Since Cloudant is a Database As a Service and based on couchdb, we might counter the option that IBM will stop this service permanently. | **Highly** unlikely. | None. | **Set up a local couchdb database server and use it instead.** |

User Requirement Document

| # | Description |
|---|---|
| 1. | **An Android Device with Internet access** |
| 2. | **A registered account with our service** |
| 3. | **ID’s (aka email) of friends to share the secret\shares with** |