**אבטחת מידע ופרטיות במחשוב ענן**

**Information Security and Privacy in cloud**

**BetweenUs**

**פרויקט גמר - הנדסת תוכנה**

**מאת**

**לירן בן גידה**

**נדב לוצטו**

**להכין**

**קיים**

**לשפר**

**תקציר**

**Summary**

**הצהרה -** העבודה נעשתה בהנחיית דר׳ ... במחלקה להנדסת תוכנה, עזריאלי - המכללה האקדמית להנדסה ירושלים ובחברת...  
החיבור מציג את עבודתי האישית ומהווה חלק מהדרישות לקבלת תואר ראשון בהנדסה.

**תודות**

**תוכן עניינים**

**מילון מונחים -** שימת מונחים, סימונים, קיצורים וראשי תיבות המופיעים בהמשך המסמך. ממוינת לפי א״ב - עברית ואח״כ אנגלית

**תיאור מסגרת הפרויקט**

**תיאור הבעיה כולל מבחינת הנדסת תוכנה**

**תיאור הפתרון**

**תיאור המערכת שמומשה**

**בדיקות**

**השוואה לספרות**

**מסקנות מימוש הפרויקט**

**סיכום**

**ספרות**

**קישורים למערכות ניהול הפרויקט ובקרת תצורה**

**נספחים (תרשמים, טבלאות מפורטות, דוח"ות בדיקה, דו"ח מדעי וכו')**

**תקציר:**

התקציר מהווה החלק הנקרא ביותר בספר ויש להקדיש לכתיבתו מאמץ רב וחזרות ובדיקות רבות. יש לוודא שהתקציר כתוב בשפה רהוטה וללא שגיאות כתיב. את התקציר יש להשלים בשלב האחרון כאשר כל העבודה גמורה וברורה. התקציר – כשמו כן הוא: 'ספר קצר'. כל הספר מתומצת לכדי דף אחד או דף וחצי לכל היותר. על התקציר לכלול פיסקה אחת מכל פרק בספר. פיסקה מתומצתת המכילה את העיקר של אותו פרק. החל במבוא, כולל את מטרת העבודה, תכנון בנייה, ניסויים ובדיקות ועיקר המסקנות. כך מתקבלת תמונה שלמה של הנעשה בפרויקט אך ללא הפרטים. לאחר קריאת התקציר הקורא יכול להחליט אם יש לו עניין להמשיך ולקרוא את כל הספר או חלקים נבחרים.

**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 6

Abstract 6

Problem Description 6

Problem Statement 6

Software Development’s challenge 6

Proposed Solution 6

System architecture 6

System States 6

Solution Description 6

Solution’s Tools and Technologies 6

Tests 6

Literature Comparison 6

Project Management Tools 6

Conclusions 6

Appendix 6

Bibliography 6

Data Flow and Diagrams 6

Database diagrams 6

Project Planning 6

Risks Table 6

User Requirement Document 6

# 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

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

The second stage of the project is developing a cloud-based service, that will 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.

# 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’s 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, 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 a minor issue.

# Proposed Solution

In this project, we will develop a cloud-based service, which will provide an additional layer in the exchange of information between uses 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 technologies:

Shamir Secret Sharing, Symmetric Crypto, Asymmetric Crypto.

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 being the number of participants of the group.

K being 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: REST API

Server Side:

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

Client Side:

* Algorithms:
  + Asymmetric (RSA).
  + Symmetric Encryption (AES-256).
  + Shamir Secret Sharing.
* Languages: JavaScript

# 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.

**Unit test will be done using the Node.js unit test framework, Node-unit.**

# 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/join> |
| 4 | Prototype demo video | <https://vimeo.com/152884344> password: betweenus |

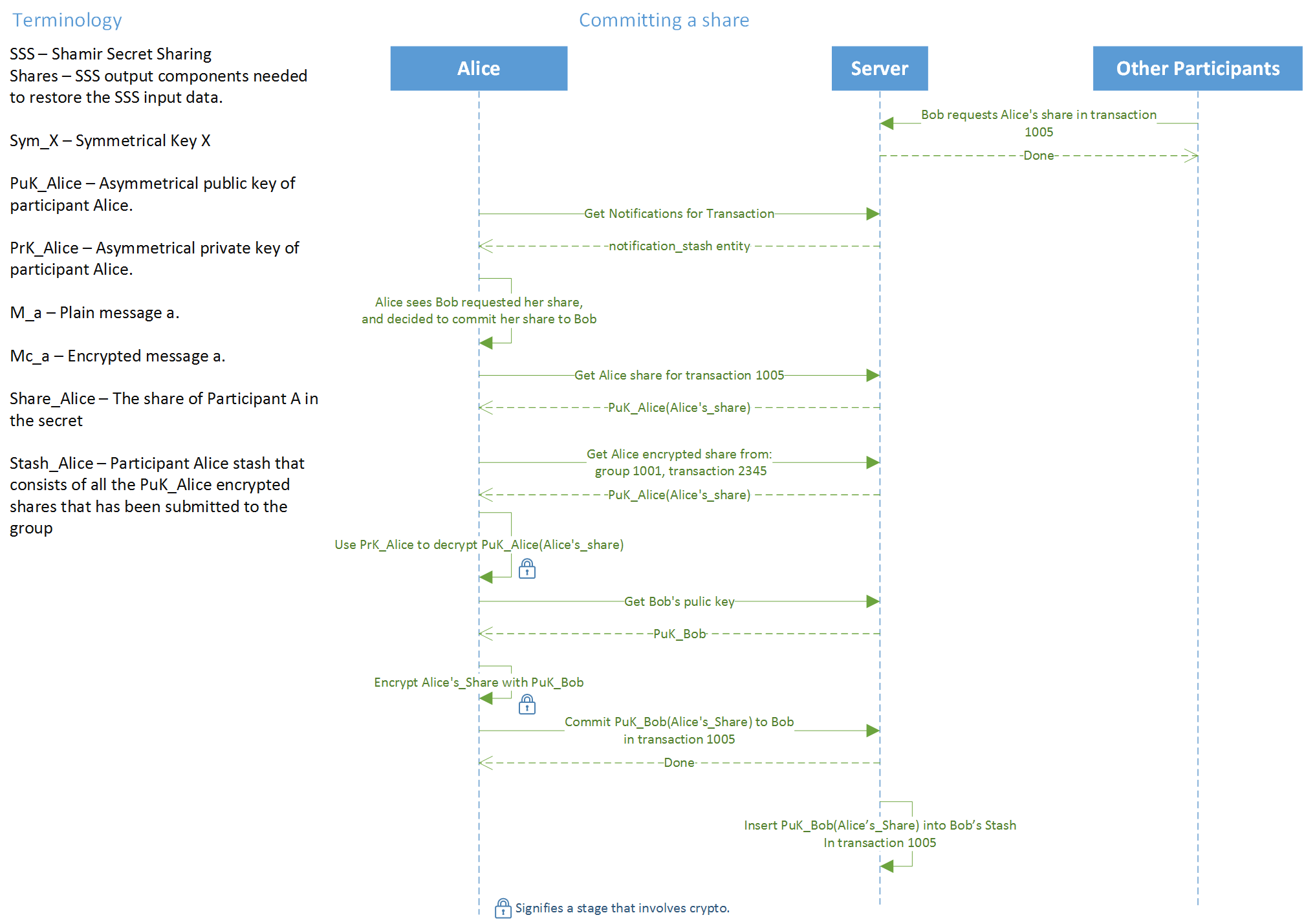
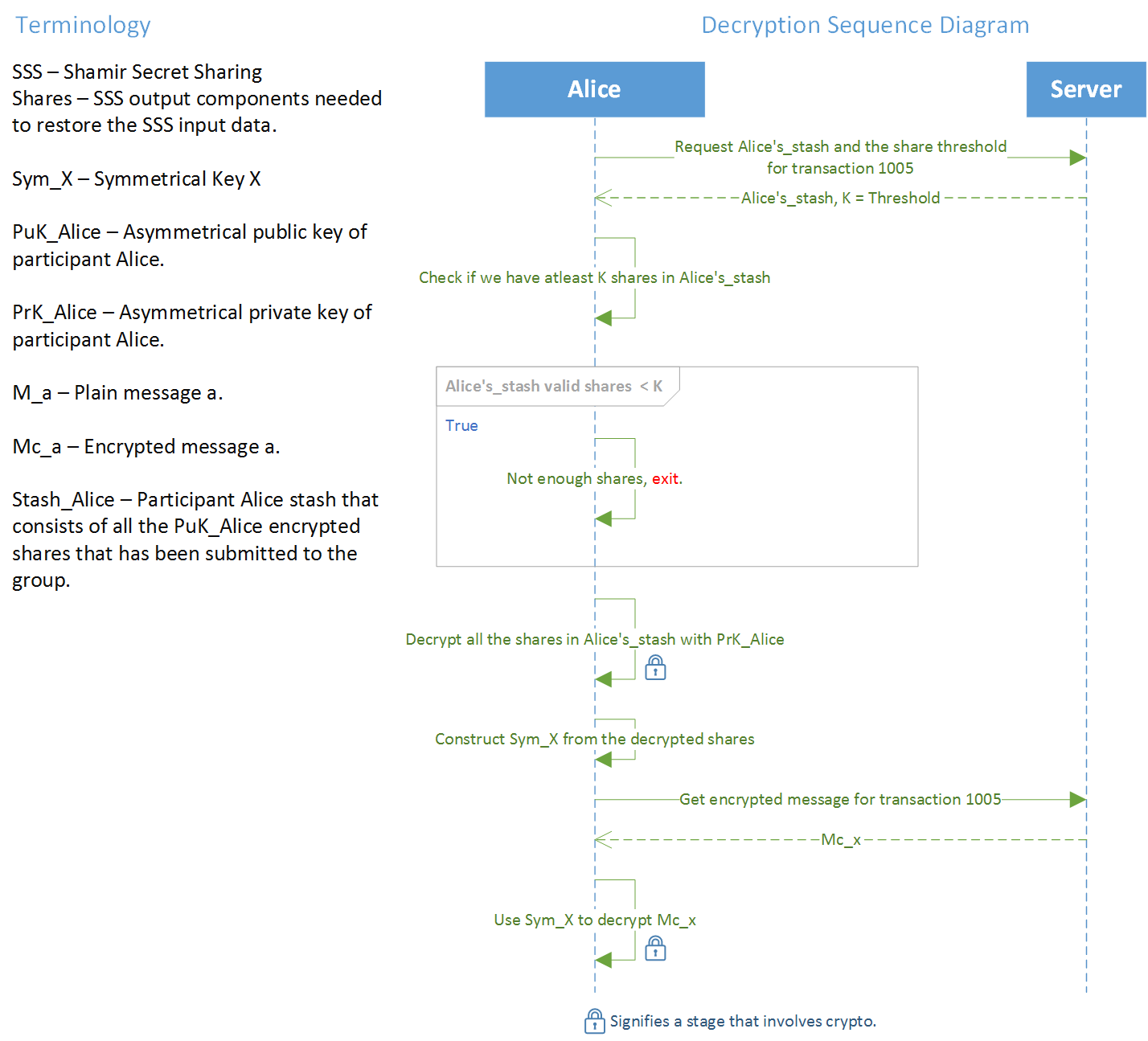
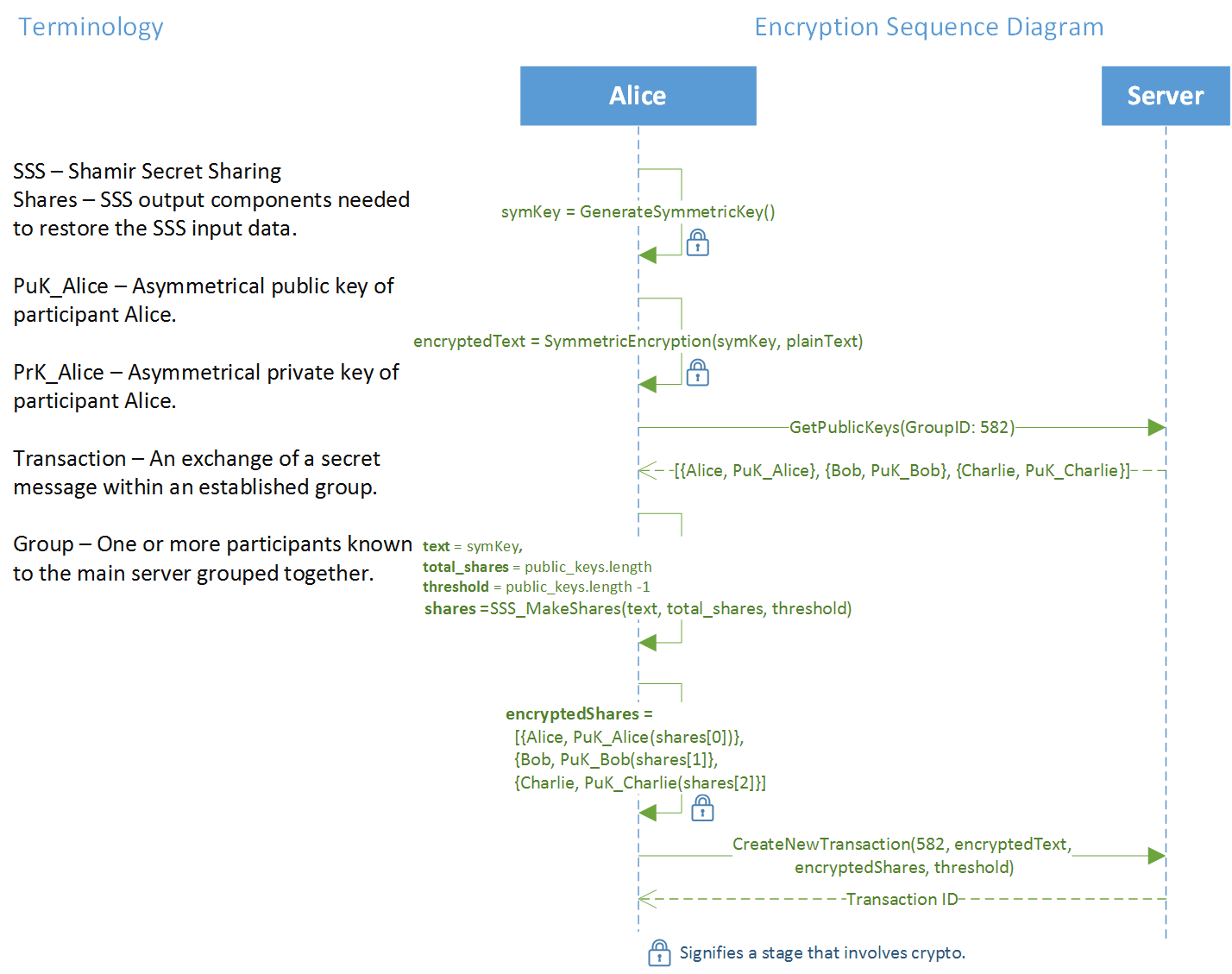
# Conclusions

# Appendix

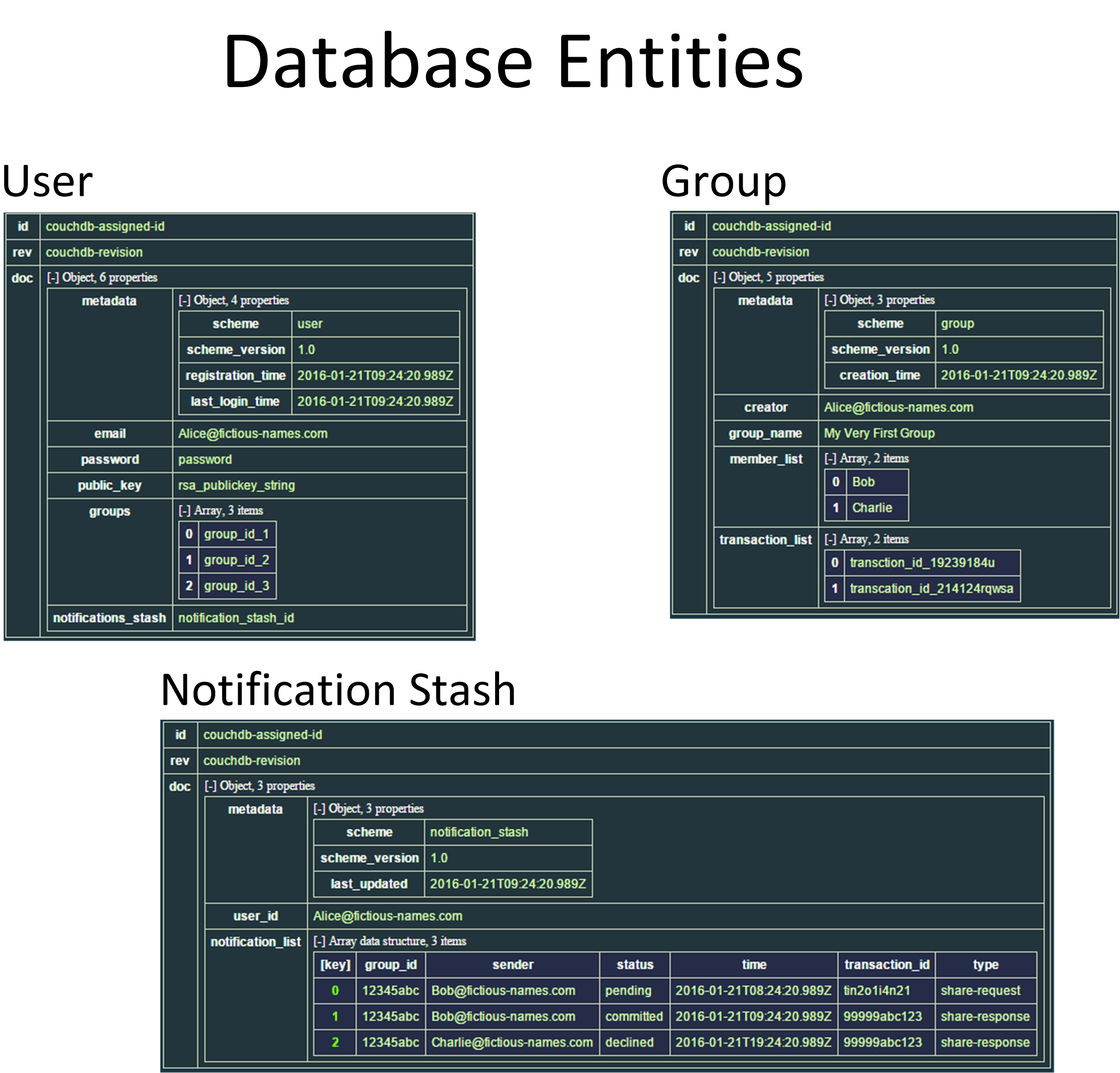
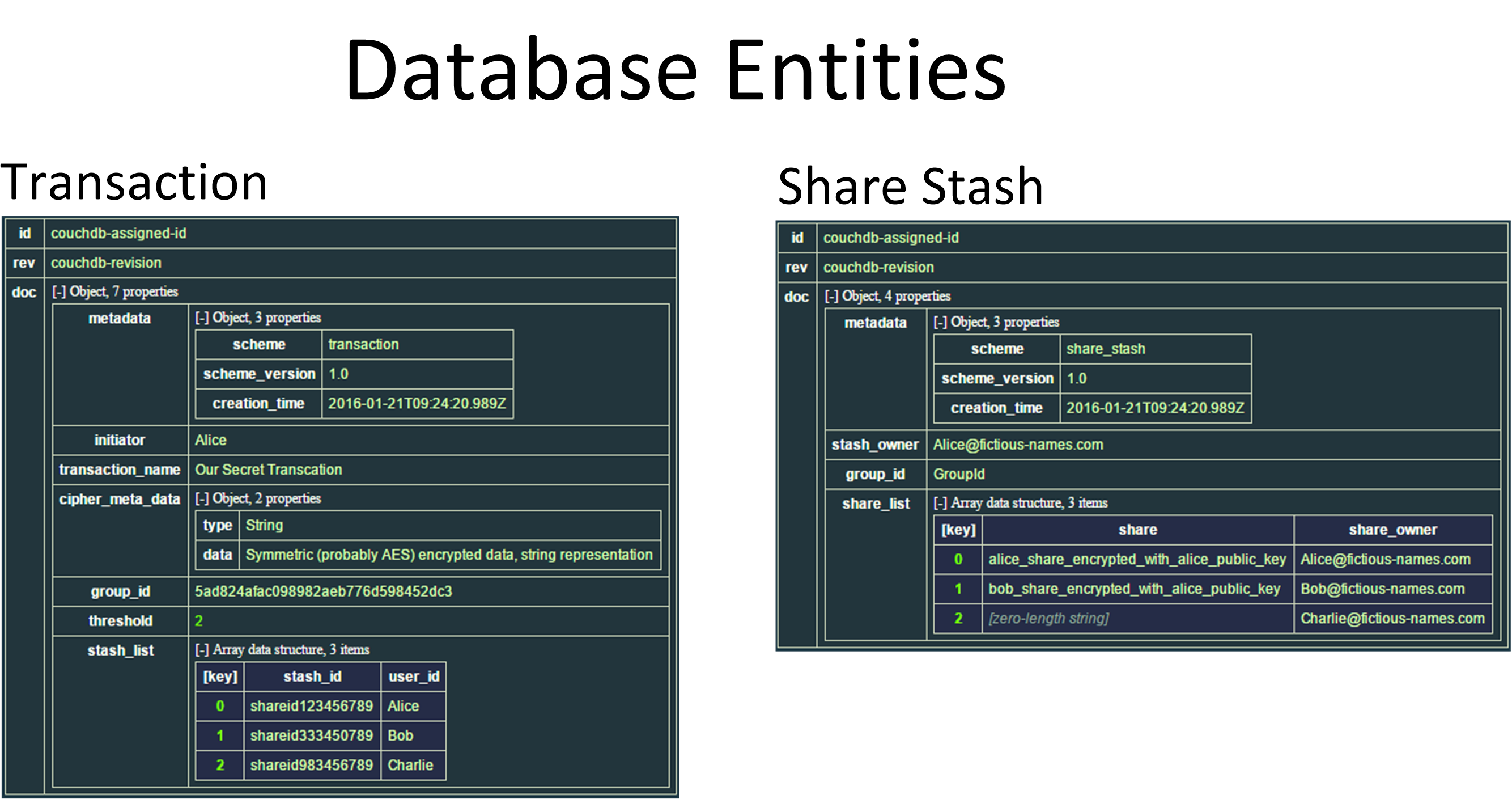
## Bibliography

* “How to share a secret” by Adi Shamir 1979 - [link](http://scholar.google.co.il/scholar_url?url=http://www.dtic.mil/cgi-bin/GetTRDoc%3FAD%3DADA069397&hl=iw&sa=X&scisig=AAGBfm1el3iYTNhzBZjYgpj3ynnNjTw1pw&nossl=1&oi=scholarr&ved=0CBoQgAMoADAAahUKEwiAurSZlP7IAhUF9HIKHaUMBzw)
* Shamir Secret Sharing by Amos Beimel - [link](http://www.cs.bgu.ac.il/~beimel/Papers/Survey.pdf)
* Lagrange Interpolation Polynomial – the explanation behind Shamir’s SS algorithm – [link](http://sam.nitk.ac.in/sites/default/Lagrange's%20Interpolating%20polynomial.pdf)

## Data Flow and Diagrams



## Database diagrams



## 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 |
| 30.4.16 | Create project’s landing page |
| 5.5.16 | Construction submission |
| 19.6.16 | Delivery submission |
| 7.7.16 | Final submission |

## Risks Table

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| If the risk is realized | What steps will performed lowering risk | Probability that the risk will occur | The severity of the impact on the project | Risks |
| The symmetric key for the decryption of the secret will be exposed. | Clients should only use trusted servers. | **Highly** unlikely. | 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. | MITM by a Malicious Server. |
| 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. | The damage can't be mitigated, just minimized. Steps will be taken to identify bad secret resolution. | Unlikely. | 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. | Denial of Service by Injection of Malicious Shares. |
| Find another crypto to use. | None. | **Highly** unlikely. | 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. | Vulnerabilities in the RSA Crypto. |
| Set up a local couchdb database server and use it instead. | None. | **Highly** unlikely. | Since Cloudant is a Database As a Service and based on couchdb, we might counter the option that IBM will stop this service permanently. | Cloudant is permanently down |

## User Requirement Document

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