Greentree – Pseudonymous Transaction Management Design Document

Revision History

Date	Revision	Author	Description
05.16.18	**1.0**	**David Dietrich**	**Genesis**
05.23.18	**2.0**	**David Dietrich**	**Updated Use Cases List and Fully Dressed Use Cases**
05.27.18	**3.0**	**David Dietrich**	**Added "Use Case Diagrams" section**
06.03.18	**4.0**	**David Dietrich**	**Updated all 4.1 Use Case Diagrams**
06.05.18	**5.0**	**David Dietrich**	**Updated 4.1.1 Sequence Diagram**
06.10.18	**5.1**	**David Dietrich**	**Updated 4.1.2 Service & Business Layers Class Diagrams**
06.17.18	**5.2**	**David Dietrich**	**Updated 4.1.2 and 4.1.1 diagrams**
06.24.18	**6.0**	**David Dietrich**	**Updated fully dressed used case 3.3 and class diagrams; separated class diagram by Service-Business layers**
06.30.18	**7.0**	**David Dietrich**	**Redrew class diagrams using PlantUML, adding domain classes**
07.08.18	**8.0**	**David Dietrich**	**Converted document to .odt and .pdf formats**

Contents

1 INTRODUCTION

1.1 Intent

This document describes the design and requirements of a new pseudonymous transaction management system called Greentree that enables secure, verifiable, and pseudonymous transactions between mutually-unknown parties on local or TLS-enabled systems.

1.2 High Level Requirements

Personally-identifiable information is deeply embedded in modern transactional systems. Ebay, Amazon, Steam, banking, technical support, social media and similar transactional platforms gather personal information from parties involved in completing transactions, store it, and use it to provide services and drive revenue.

This paradigm is coming under increased scrutiny from both consumers and regulators alike, who view the escalating scale and frequency of high-profile data breaches as signals of the need to achieve superior data privacy. Enforcing standards such as General Data Protection Regulation (GDPR) is one way to do this. Greentree seeks a grass-roots solution.

Greentree is a distributed, pseudonymous transaction system facilitating secure, private transactions between mutually-unknown parties via token-based identities on a token-controlled audit log that provides both privacy and trust to system users. The initial iteration of the system should be installed and run locally, but later iterations should also scale out over TLS-encrypted web APIs.

1.2.1 Token Provisioning

Users loading Greentree initially should be prompted to log a private and unique identifying token with Greentree which will be used to identify the contact securely and anonymously through all future interactions with Greentree. The token prompt must use a simple and intuitive form that generates and stores the token in the Greentree system over a securely encrypted channel.

1.2.2 Privacy Regulation

Greentree data must be collected in compliance with General Data Protection Regulation (GDPR). This means users must log consent in the Greentree ledger before logging their tokens and associated transactions. Users must be informed how their data is transferred and stored by Greentree before providing consent. Privacy regulation also means users must have the option to erase their data from the ledger at any time they choose. Communicating this information to contacts is key both to comply with privacy regulations, and to build a trust relationship between Greentree and its users.

1.2.3 Access, Authorization, Auditing (AAA)

Access to data stored in Greentree requires secure authentication that never exposes credentials in plaintext, either in transit or at rest. Authorization to access identity data must be explicitly granted by Greentree, and

this authorization must be logged in an immutable format not subject to tampering. Every access to the data of each user in the system must be audited in an immutable log that is accessible to the owner.

1.2.4 Data Integration

Identity data should be integrated with a simple messaging system stored in Greentree which facilitates secure, pseudonymous communication between Greentree users. The initial prototype should implement this integration in local storage, but future iterations should enable TLS-encrypted RESTful API integration with services over the public and private TCP networks.

1.2.5 Pseudonymity

Pseudonymity in Greentree means that each user will have an identifiable collection of messages and logs associated with their token, but this does not include any personally identifying information; instead, tokens will be the only means of identification. These will be used to track and control ownership of, and access to, Greentree data. Each user with unlock their token by authenticating with Greentree using a private key file and passphrase set at the time of registration.

1.2.6 Token Standards

Tokens will facilitate distributing, validating, and controlling claims by Greentree users. Claims will track who issued each token, for how long, to whom, and for what audience. Users seeking access to data in Greentree must provide a token with claims suitable to the data requested, and this token must include a signature that is successfully validated by Greentree before access is granted. Tokens and data must be protected with public key cryptography, and users should be expected to store their private keys outside the Greentree system.

1.3.0 Iterative Improvement

The initial prototype of the identity management system should be desktop-based. The tool should be installed and operated locally, allowing users to manage their tokens, access data, and communicate with one another asynchronously on the local system. Later iterations should involve adding interfaces for SSH, HTTP, and RESTful web API access over TLS-encrypted TCP sockets that will allow the identity system to fully integrate with systems on IP networks. Early iterations of Greentree may use simple filesystem storage, but mature iterations should implement distributed blockchain storage.

2 Use Cases List

2.1 Intent

This section lists the use cases describing user interaction with the identity management system.

2.2 Customer Interacting Use Cases

Following are the customer interacting use cases by priority:

Provision a token and view its audit log

Post an encrypted message and grant another token access to it

Decommission a token, deleting associated messages and audit logs

Post an encrypted file and grant another token access to it

Mine a distributed transaction ledger to unlock more storage

2.2.1 Use Case Diagram

The diagram below depicts the use cases described above. The Greentree user is the main actor.

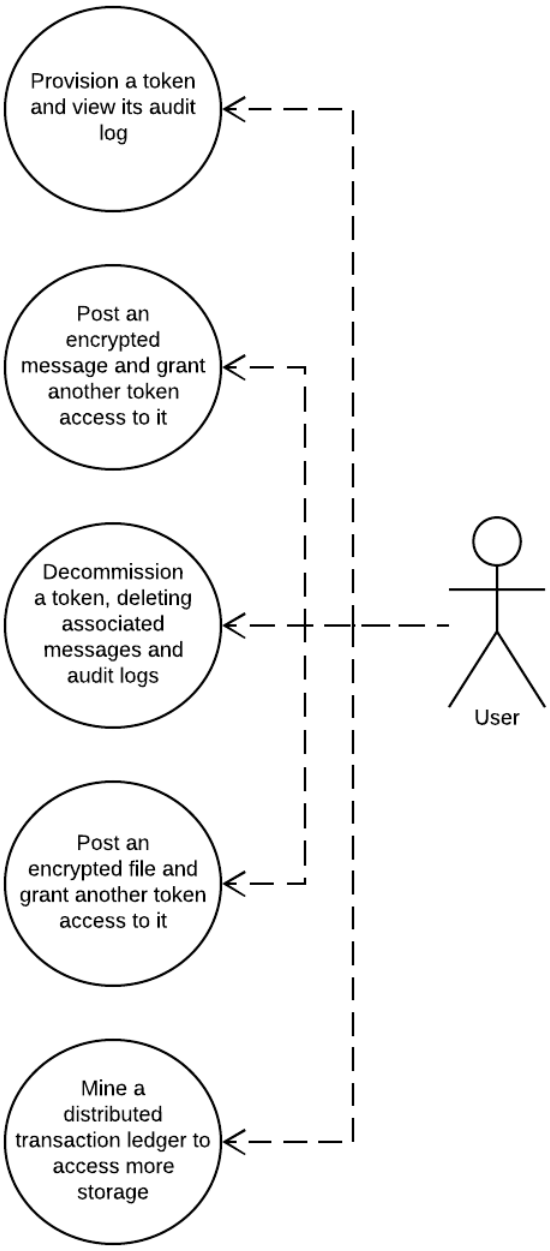

Figure 2-1 GreenTree Contact Token Use Cases

3 Fully Dressed Use Cases

3.1 Intent

This section documents fully dressed use cases for the GreenTree project.

3.2 Provision a token and view its audit log

Primary Actor: Greentree User

Assumptions:
- User has installed the Greentree

Stakeholders:
- User provisioning the Greentree token

Pre-conditions:
- The user has no token

Post-conditions (Success Guarantee):
- The user has been issued an identifying token from Greentree
- The user has a copy of their passphrase for storage outside Greentree
- Greentree has saved the user token to persistent storage
- Greentree has logged the provisioning event
- Greentree has displayed the token log events to the user

Flow of Events:

1. Contact loads the Greentree UI.

2. Greentree prompts the user to create a new Greentree token or authenticate with an existing key and passphrase.

   Prompt Requirements

   a. The prompt includes a brief notice stating that accessing Greentree creates audit logs which are accessible to users bearing authorized tokens.

   b. The prompt links to fuller documentation on Greentree token storage, privacy, and security practices.

c. The prompt states that access to Greentree requires the user to securely store their key and passphrase outside Greentree, and that losing this key will invalidate access to all data linked with the token for that key.

3. The user picks the option to generate a token, and enters a secure passphrase.

   a. **ALT FLOW 3.2.1: Contact cancels the token prompt.** The contact will be denied access to Greentree until they provide or provision a valid token.

   b. **ALT FLOW 3.2.2: Contact submits a valid key and passphrase.** Greentree validates the passphrase with the key, then loads the main Greentree interface and logs the event in the audit log of the token.

   c. **ALT FLOW 3.2.3: Contact submits an invalid key or passphrase.** Greentree detects the invalid key or passphrase, denies the user access with an error message.

4. Greentree generates and stores a new token with the key and passphrase given by the user and logs an audit message noting the provision event.

   Token Processing Requirements

   a. A new audit log must be instantiated, linked with the token, and loaded with an entry logging the provision event.

   b. Each audit log entry must store the datetime, type, token id, and source address for the event.

   c. Each token issued must be signed by the private key of the Greentree server instance and the token payload must include the validity duration and domain of the token.

   d. The token should be stored in local storage of the Greentree host, and the user must be notified of this.

   e. Subsequent authentication events prompt Greentree to update an audit log linked to the token.

5. The user views the audit log in Greentree, noting the date and time of the token generation along with the successful authentication.

6. The user logs back out of the system, enters their key and passphrase to log back in, and is presented with an updated log showing the new authentication event.

3.3 Post an encrypted message and grant another token access to it

Primary Actor: Greentree User

Assumptions:

- Greentree User "A" has an message they wish to post in Greentree

- Greentree User "B" would like to view the message posted by User A

Stakeholders:
- User entering a new message
- User receiving new message

Pre-conditions:
- User A has generated a Greentree token and logged into Greentree.
- User A has the address of the token for User B

Post-conditions (Success Guarantee):
- User A has stored message in Greentree
- User A has granted User B access to read the message
- User B has accessed the message through GreenTree
- GreenTree has logged the activity of both users

Flow of Events:

1. User A logs into Greentree by entering their password and key.

2. Greentree logs the authentication attempt, validates the password and token using the key, grants the user access, and logs the successful attempt in the audit log of the token.

    a. **ALT FLOW 3.3.1: Greentree denies access.** See Alt Flow 3.2.2.

3. Greentree loads a home screen which includes the option to post a new message.

    Home Screen Requirements

    a. Display an option to access a central messaging and event log with token audit events, access requests, and messages ordered chronologically.

    b. Display options to provision new tokens, revoke existing tokens, or delete data affiliated with tokens, revoked or otherwise.

    c. Display options to log new messages and grant access to these messages for other users.

4. User A clicks a "new message" option, enters a message, grants User B access to the message, and posts it.

5. Greentree logs the activity, then stores the data in an encrypted blockchain.

6. User B logs into Greentree with their passphrase and key, then views the message posted by User A.

4 DIAGRAMS

4.1 Sequence Diagram for Token Provisioning

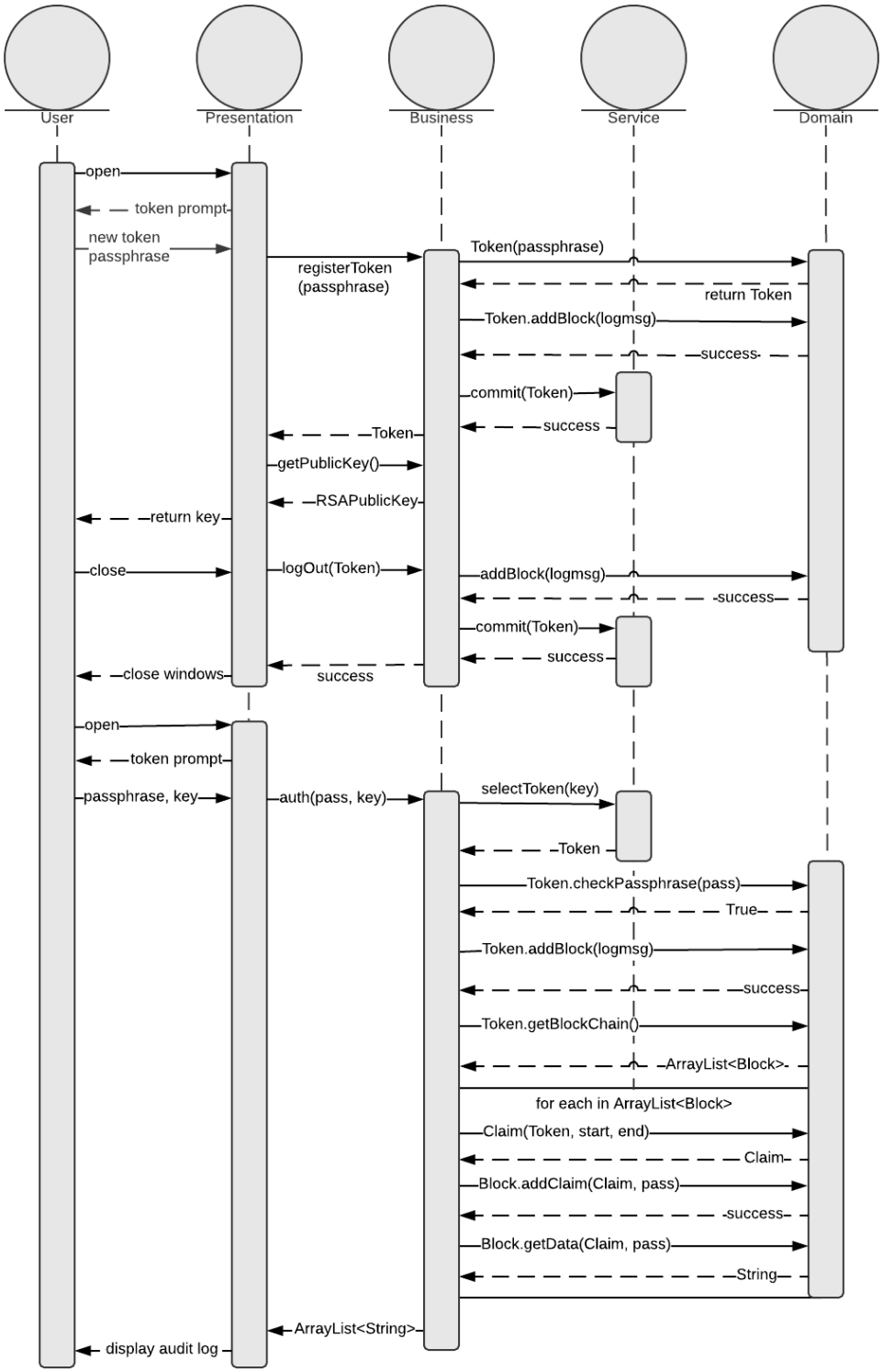

Figure 4-1

4.2 Service Layer Class Diagram

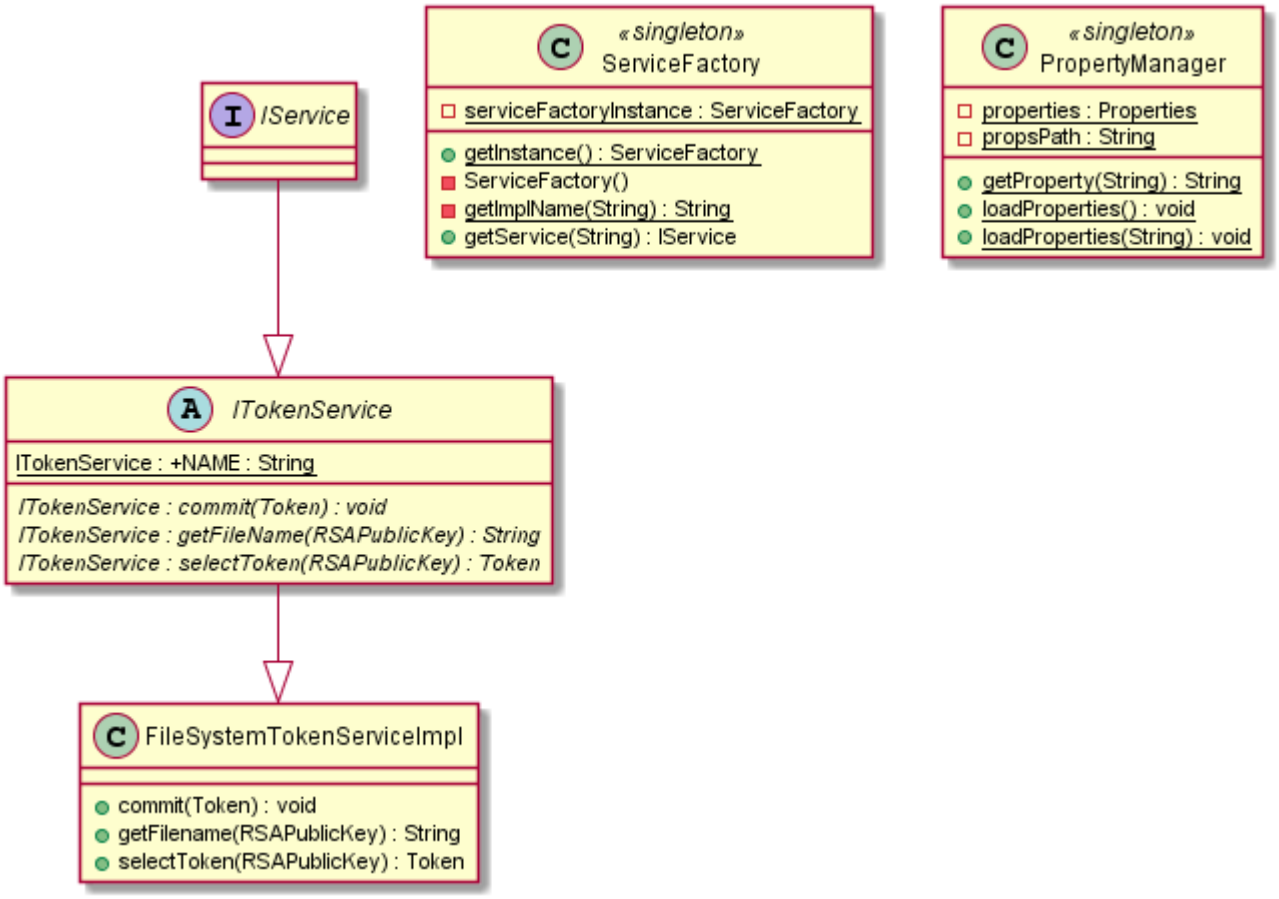

Figure 4-2

4.3 Business Layer Class Diagram

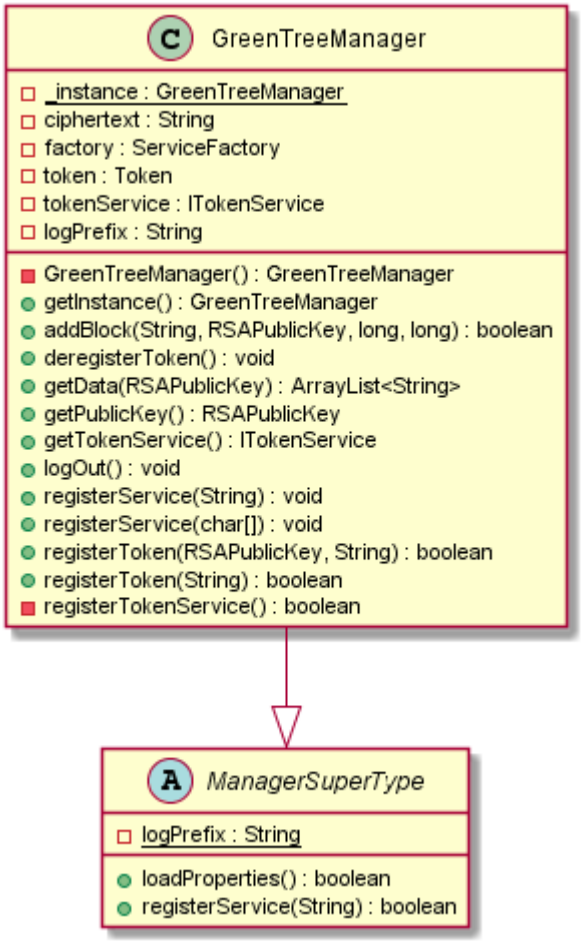

Figure 4.3

4.4 Domain Layer Class Diagram

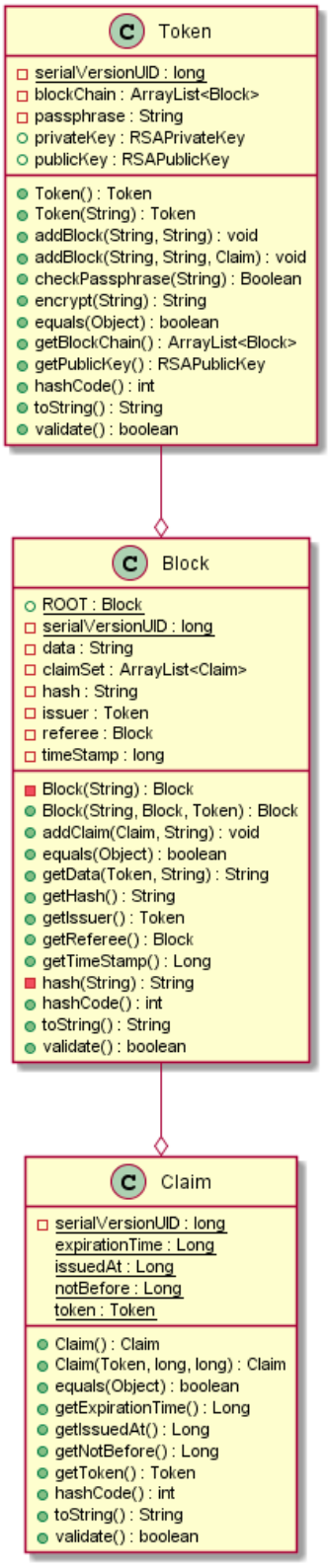
Token
serialVersionUID : long
blockChain : ArrayList<Block>
passphrase : String
privateKey : RSAPrivateKey
publicKey : RSAPublicKey
Token() : Token
Token(String) : Token
addBlock(String, String) : void
addBlock(String, String, Claim) : void
checkPassphrase(String) : Boolean
encrypt(String) : String
equals(Object) : boolean
getBlockChain() : ArrayList<Block>
getPublicKey() : RSAPublicKey
hashCode() : int
toString() : String
validate() : boolean
Block
ROOT : Block
serialVersionUID : long
data : String
claimSet : ArrayList<Claim>
hash : String
issuer : Token
referee : Block
timeStamp : long
Block(String) : Block
Block(String, Block, Token) : Block
addClaim(Claim, String) : void
equals(Object) : boolean
getData(Token, String) : String
getHash() : String
getIssuer() : Token
getReferee() : Block
getTimeStamp() : Long
hash(String) : String
hashCode() : int
toString() : String
validate() : boolean
Claim
serialVersionUID : long
expirationTime : Long
issuedAt : Long
notBefore : Long
token : Token
Claim() : Claim
Claim(Token, long, long) : Claim
equals(Object) : boolean
getExpirationTime() : Long
getIssuedAt() : Long
getNotBefore() : Long
getToken() : Token
hashCode() : int
toString() : String
validate() : boolean

Figure 4.4

5 SUMMARY

This document details the requirements, highlights key use cases and walks through two fully dressed use cases of the Greentree system.

- End -