BlockMatrix User Manual

National Institute of Standards and Technology

# Description

This java package serves as implementation of a transaction system implemented similarly to a blockchain, but which uses a “blockmatrix” data structure to allow deletion. It allows local use of a blockchain which uses a blockmatrix.

Users can:

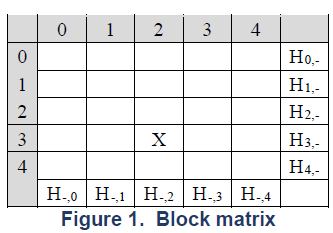
* Create wallets
* Pass funds along with a message to other user’s wallets
* Modify messages passed along with transactions after blocks have been published

## Structure and Concept

Each transaction consists of 4 key parts: A sender, a recipient, a value, and a message. The transaction is stored in a block in the blockmatrix. For each row and column in the blockmatrix, the blockmatrix stores a hash code made from the hashes of each block in that row or column.

The hash in a block is generated from the hash of the transactions in that block using a merkleroot.

In a traditional blockchain, we could not modify any of the info in a transaction/any info that has been placed in a block without compromising the integrity of the entire blockchain. This is because traditional blockchains maintain integrity by storing the hash of the contents of the previous block, so if any info in any block changes, the hashes of all subsequent blocks will be invalidated. Using a blockmatrix prevents this from occurring, as each block has 2 hashes ensuring its integrity. For example, below, the block marked X is represented in both hashes H-,2  and H3, - .



So, if the data in block in X was modified, as long as only the hashes H-,2 and H3, - were modified, we could be assured that every other block would have at least one row or column hash unchanged, meaning that block has not been modified. This allows us to modify the data in one block without being worried that all the info that has been placed in blocks since that block has been put at risk.

Only the message passed along with transactions can be modified, however, due to the nature of transactions in blockchains. Security is assured by not using a system where wallets have a traditional basic “balance.” Instead, users can only pass assets from one wallet to another by “proving” they own an asset they are passing by providing a reference to the transaction in which they received this asset. Just as in traditional blockchains, it is impossible to modify or delete the sender, recipient, or value of a transaction, because if these are modified or deleted, and someone needs to reference that transaction in a transaction they are trying to complete, it would prevent them from doing so.

Nevertheless, this provides use for transaction systems where providing messages is promoted or required (such as Venmo, a mobile application where users can pay each other but must include a message as to what the transaction is for.) It also provides the high level of security and the lack of need for trust that blockchains do, with the additional benefit of being able to modify messages passed along with transactions after they have been sent.

The block matrix is based off the data structure outlined in this NIST Cybersecurity White Paper:

<https://csrc.nist.gov/CSRC/media/Publications/white-paper/2018/05/31/data-structure-for-integrity-protection-with-erasure-capability/draft/documents/data-structure-for-integrity-with-erasure-draft.pdf>

# Setup

## Installation

To install the BlockMatrix package, navigate to out/artifacts/blockmatrix\_jar/ and download the JAR file blockmatrix.jar. Include this jar file as an external library for your project.

### Dependencies

You will need to import Bouncy Castle:

* bouncy castle: <https://www.bouncycastle.org/download/bcprov-jdk15on-159.jar>

Once you have done this, import the package with

**import** blockmatrix.\*;

at the top of your Java file to be able to use the program.

## First Steps

1. Create a BlockMatrix blockchain, initializing it with the constructor:

**public** BlockMatrix(**int** dimension)

The **dimension** parameter is where you can specify the size of the BlockMatrix you would like. A **dimension** of N will create an NxN BlockMatrix that will be able to hold NxN – N blocks total (due to the empty diagonal). This value cannot be changed, so it is recommended to make sure the BlockMatrix has more than enough space when you initialize it.

1. Set up the security provider. For the rest of this setup, let us suppose the name of the BlockMatrix object we initialized is bm. Set up the security provider like so:

bm.setUpSecurity();

This will add Bouncy Castle as a provider to the Java Security API we will be using, allowing us to create Elliptic-Curve KeyPairs for our wallets for the wallets Public and Private keys.

1. Create your initial wallet, initializing it with the constructor:

**public** Wallet();

1. Start up our blockchain by creating the genesis transaction which will give our initial wallet a specified amount of funds.

**public void** generate(Wallet wallet, **float** value)

The **wallet** parameter is the wallet we want to be the recipient of our genesis transaction to be. This wallet will receive **value**, a prespecified amount of the asset (or “coin”) you want this wallet to start with.

Once this is done, all the initial steps of setting up our blockmatrix blockchain are done! You are free to create wallets, create transactions, create blocks, add transactions to blocks, and add blocks to the block matrix at your own discretion.

//Unfinished