Supporting GDPR Requirements and Integrity in Distributed Ledger Systems

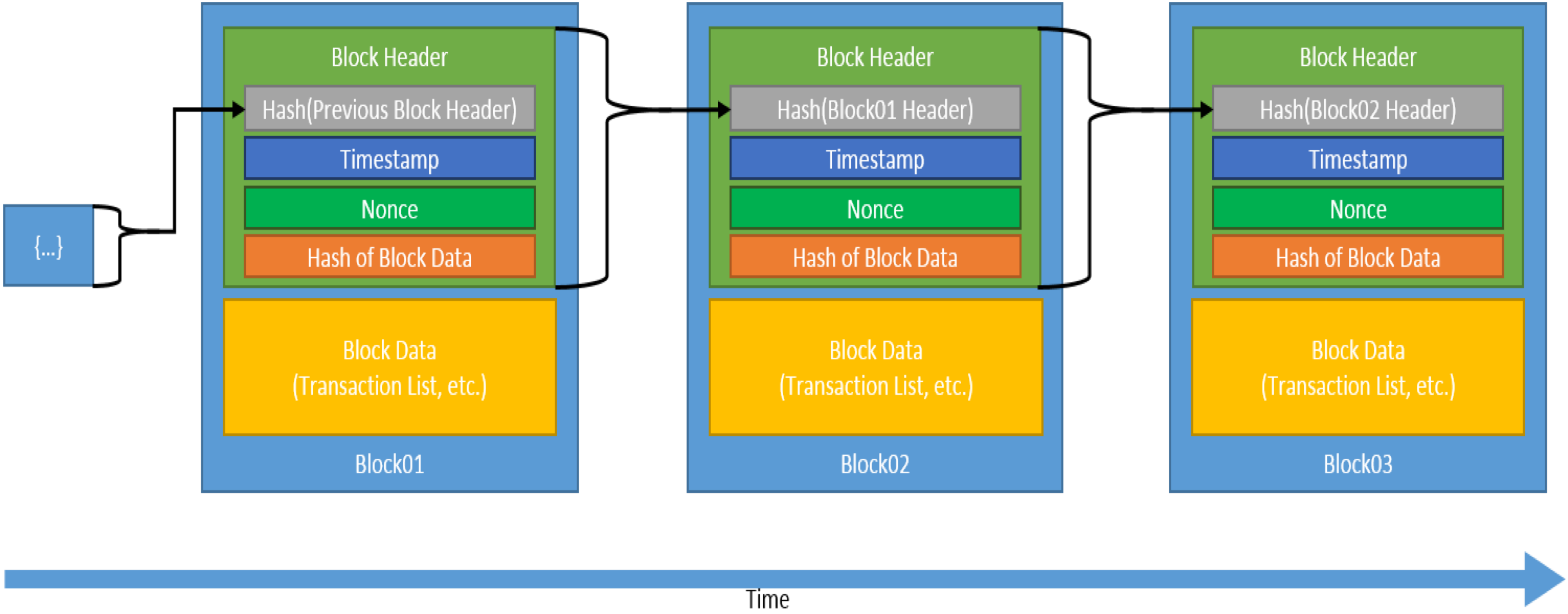
Rick Kuhn

NIST Computer Security Division

What is the problem?

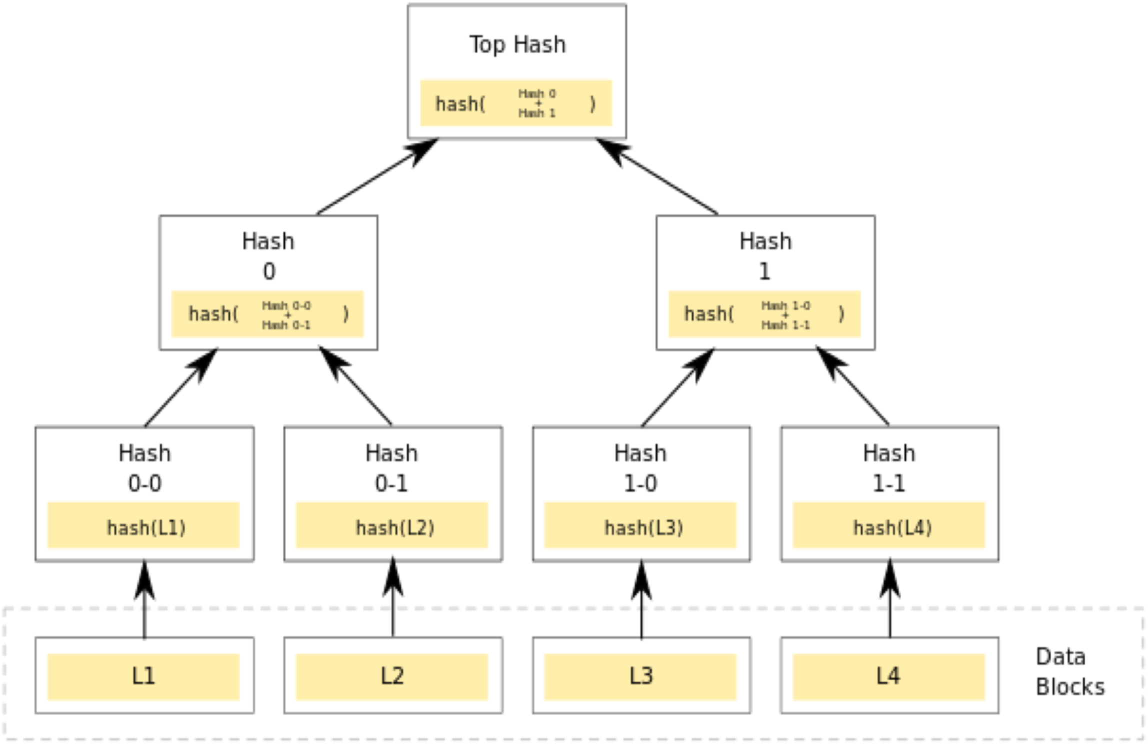
- Blockchain has been defined as "an open, distributed ledger that can record transactions between two parties efficiently and in a verifiable and permanent way".
- The permanence/immutability property that makes blockchain technology useful also leads to difficulty in supporting privacy requirements
- European Union General Data Protection Regulation (GDPR) requires that all information related to a particular person can be deleted at that person's request
  - *personal* data, defined as "any information concerning an identified or identifiable natural person" - data for which blockchains are designed to be used
  - "Personal data which have undergone pseudonymisation, which could be attributed to a natural person by the use of additional information should be considered to be information on an identifiable natural person."

Structure of a Traditional Blockchain



Why is GDPR deletion requirement a problem for blockchains?

- Conventional distributed ledger blockchain – change to one block changes hashes of all; provides integrity protection
- Hashes provide assurance that information in every other block is unchanged if one block is modified
- If we had to delete a block, hash values for others are no longer valid
- Don’t want to create a new chain



What are ways of dealing with this problem?

- Don’t put personal information on blockchain
  - Pseudo-anonymized data are still considered personal
  - Even if not directly tied to a person – dynamic IP address can be considered personal if it can be indirectly tied to an individual
- Encrypt data and destroy key to delete
  - Data must be secure for decades
  - Cannot be sure that future developments in crypto will not reveal it – e.g. quantum computing puts current public key systems at risk

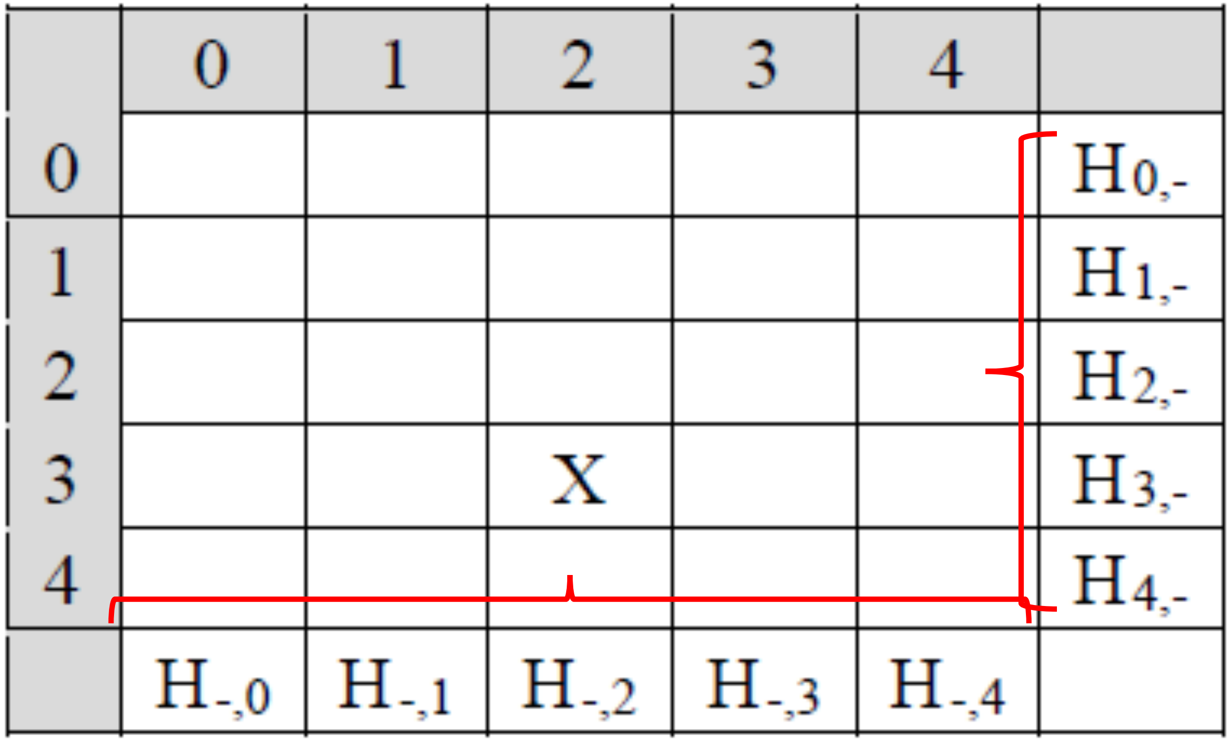
What are the constraints and assumptions?

- Hash integrity protection must not be disrupted for blocks not deleted
- Deletions will be relatively rare
- Ensure auditability and accountability
- Application to permissioned/private distributed ledger systems

New data structure solution: a datablock matrix

- A data structure that provides integrity assurance using hash-linked records while also allowing the deletion of records

- Stores hashes of each row and column
- => each block within the matrix is protected by two hashes
- Suggested use for private/permissioned distributed ledger systems



	0	1	2	3	4	
0						$H_{0,-}$
1						$H_{1,-}$
2						$H_{2,-}$
3			X			$H_{3,-}$
4						$H_{4,-}$
	$H_{-,0}$	$H_{-,1}$	$H_{-,2}$	$H_{-,3}$	$H_{-,4}$	

Figure 1. Block matrix

How does this work?

- Suppose we want to delete block 12

- disrupts the hash values of $H_{3,-}$ for row 3 and $H_{-,2}$ and column 2

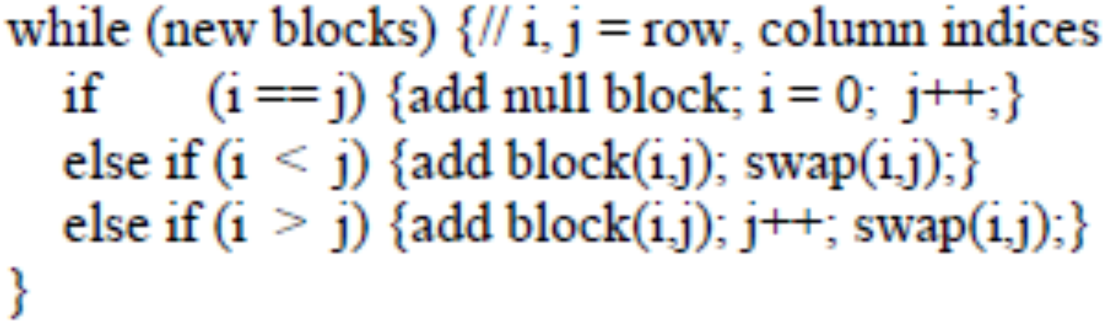
- blocks of row 3 are included in the hashes for columns 0, 1, 3, and 4

- blocks of column 2 are included in the hashes for rows 0, 1, 2, and 4

	0	1	2	3	4	
0	●	1	3	7	13	$H_{0,-}$
1	2	●	5	9	15	$H_{1,-}$
2	4	6	●	11	17	$H_{2,-}$
3	8	10	12	●	19	$H_{3,-}$
4	14	16	18	20	●	$H_{4,-}$
	$H_{-,0}$	$H_{-,1}$	$H_{-,2}$	$H_{-,3}$	$H_{-,4}$	etc.

Datablock Matrix Population Algorithm

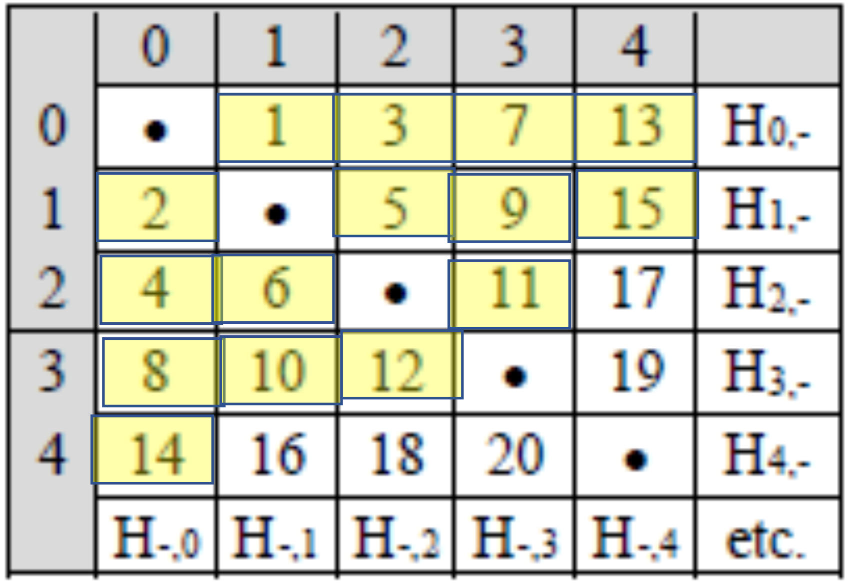
- Algorithm



```
while (new blocks) {// i, j = row, column indices
    if      (i == j) {add null block; i = 0;  j++;}
    else if (i  <  j) {add block(i,j); swap(i,j);}
    else if (i  >  j) {add block(i,j); j++; swap(i,j);}
}
```

- Basic algorithm is simple, many variations possible
- Implemented as Java code
- Github project

- Block ordering provides desirable properties

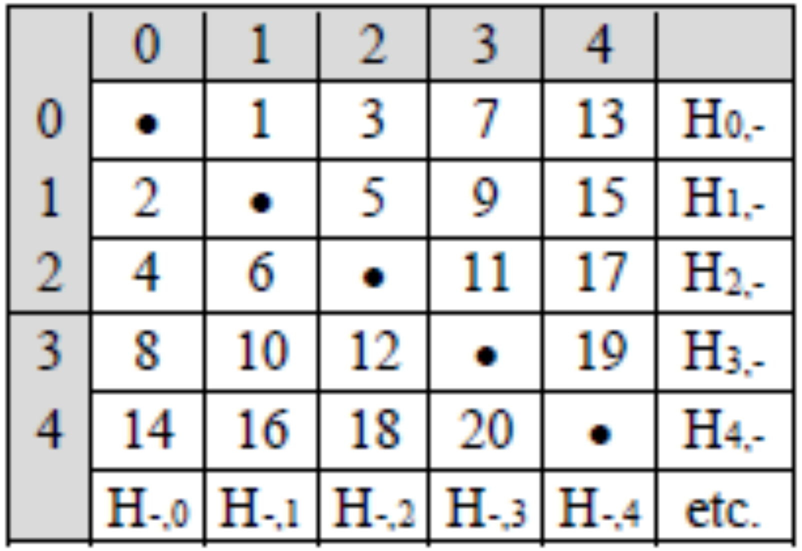


	0	1	2	3	4	
0	●	1	3	7	13	$H_{0,-}$
1	2	●	5	9	15	$H_{1,-}$
2	4	6	●	11	17	$H_{2,-}$
3	8	10	12	●	19	$H_{3,-}$
4	14	16	18	20	●	$H_{4,-}$
	$H_{-,0}$	$H_{-,1}$	$H_{-,2}$	$H_{-,3}$	$H_{-,4}$	etc.

Figure 2. Block matrix with numbered cells

Data Structure Properties

- *Balance*: upper half (above diagonal) contains at most one additional cell more than the lower half.
- *Hash sequence length*: number of blocks in a row or column hash proportional to $\sqrt{N}$ for a matrix with *N* blocks, by the balance property.
- *Number of blocks*: The total number of data blocks in the matrix is $N^2 - N$ since the diagonal is null.
- *Block dispersal*: No consecutive blocks appear in the same row or column

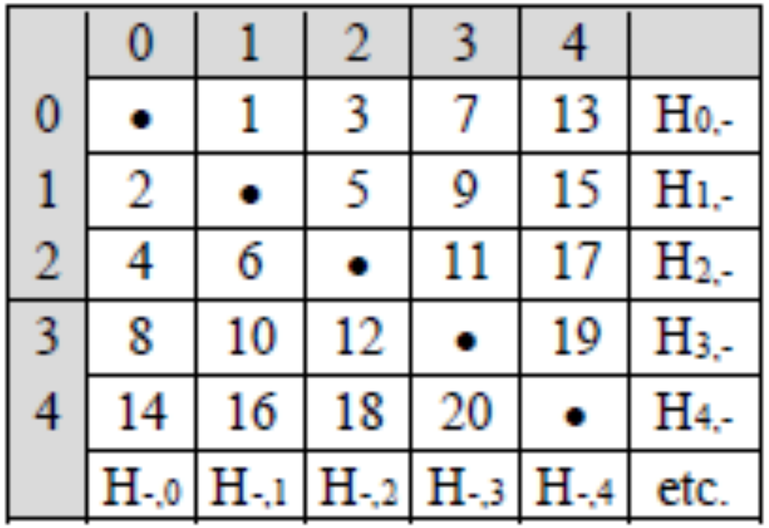


	0	1	2	3	4	
0	•	1	3	7	13	$H_{0,-}$
1	2	•	5	9	15	$H_{1,-}$
2	4	6	•	11	17	$H_{2,-}$
3	8	10	12	•	19	$H_{3,-}$
4	14	16	18	20	•	$H_{4,-}$
	$H_{-,0}$	$H_{-,1}$	$H_{-,2}$	$H_{-,3}$	$H_{-,4}$	etc.

Figure 2. Block matrix with numbered cells

Consecutive block deletion

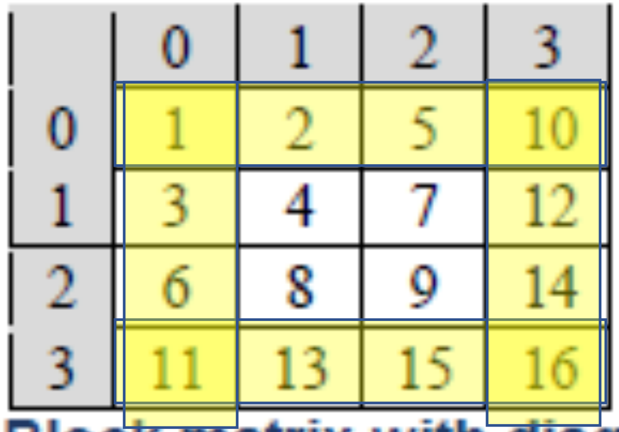
- Algorithm keeps main diagonal null
- Allows deletion of two consecutive blocks without disrupting hashes
- Example – deleting blocks 7 and 8 without null diagonal would lose hash integrity protection for blocks 4 and 9



	0	1	2	3	4	
0	•	1	3	7	13	$H_{0,-}$
1	2	•	5	9	15	$H_{1,-}$
2	4	6	•	11	17	$H_{2,-}$
3	8	10	12	•	19	$H_{3,-}$
4	14	16	18	20	•	$H_{4,-}$
	$H_{-,0}$	$H_{-,1}$	$H_{-,2}$	$H_{-,3}$	$H_{-,4}$	etc.

Figure 2. Block matrix with numbered cells

Vs.



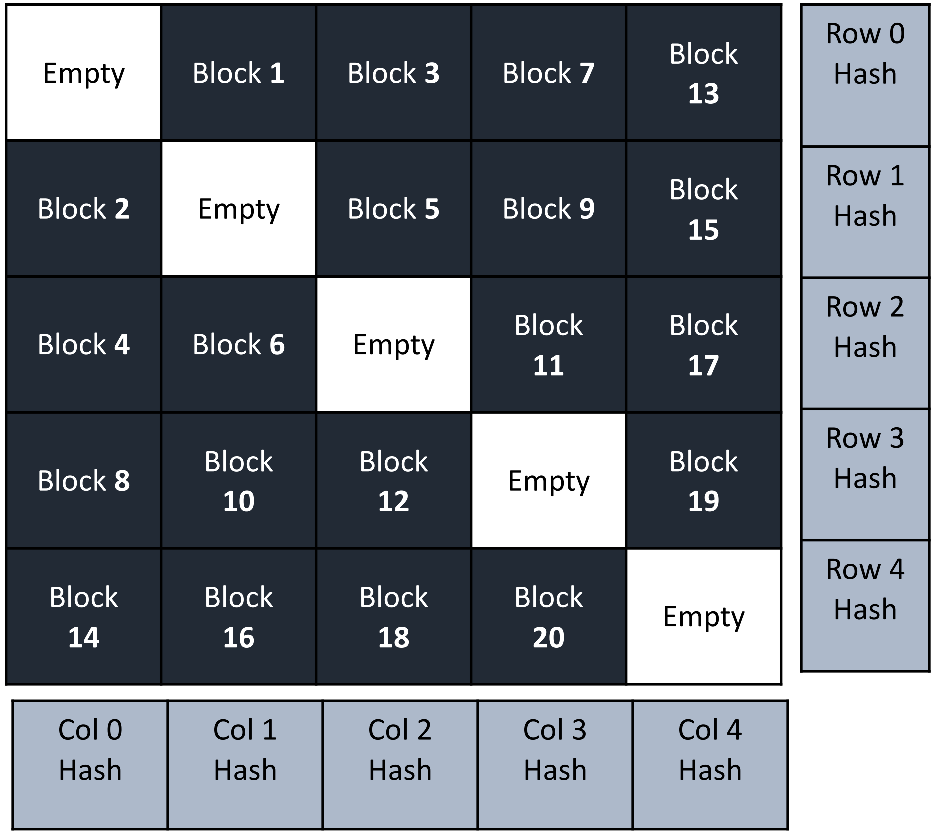
	0	1	2	3
0	1	2	5	10
1	3	4	7	12
2	6	8	9	14
3	11	13	15	16

Figure 3. Block matrix with diagonal used

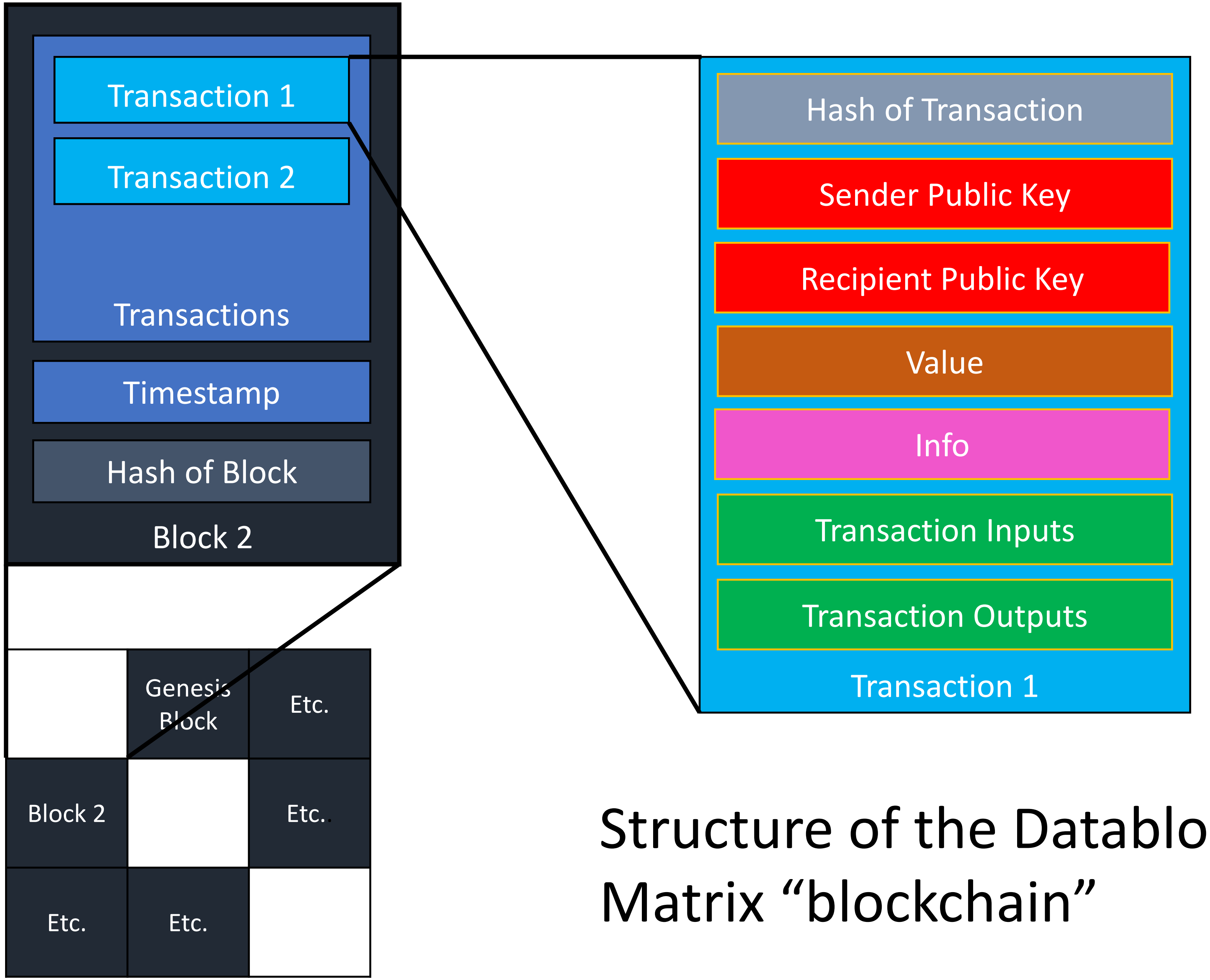
Applying Block Matrices to Blockchains

- Similar structure and security as a blockchain
- capability of deleting or modifying certain parts of a transaction or block
- Same transaction model, same cryptographic key/address model
- Implemented in open source code

Implementation by Arsen Klyuev, Johns Hopkins Univ



Empty	Block 1	Block 3	Block 7	Block 13	Row 0 Hash
Block 2	Empty	Block 5	Block 9	Block 15	Row 1 Hash
Block 4	Block 6	Empty	Block 11	Block 17	Row 2 Hash
Block 8	Block 10	Block 12	Empty	Block 19	Row 3 Hash
Block 14	Block 16	Block 18	Block 20	Empty	Row 4 Hash
Col 0 Hash	Col 1 Hash	Col 2 Hash	Col 3 Hash	Col 4 Hash	



Transaction 1

Transaction 2

Transactions

Timestamp

Hash of Block

Block 2

	Genesis Block	Etc.
Block 2		Etc.
Etc.	Etc.	

Hash of Transaction

Sender Public Key

Recipient Public Key

Value

Info

Transaction Inputs

Transaction Outputs

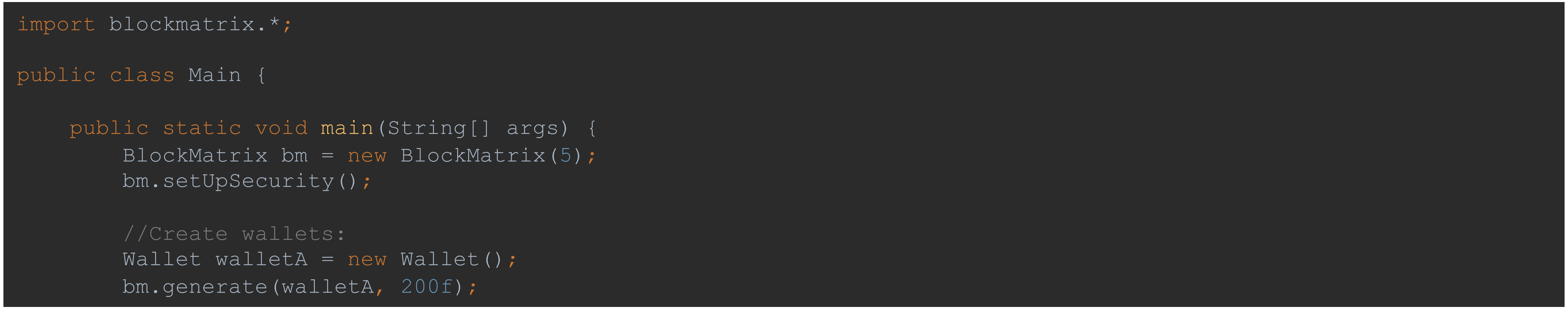
Transaction 1

Structure of the Datablock Matrix “blockchain”

Java BlockMatrix Package

Implementation by Arsen Klyuev, JHU

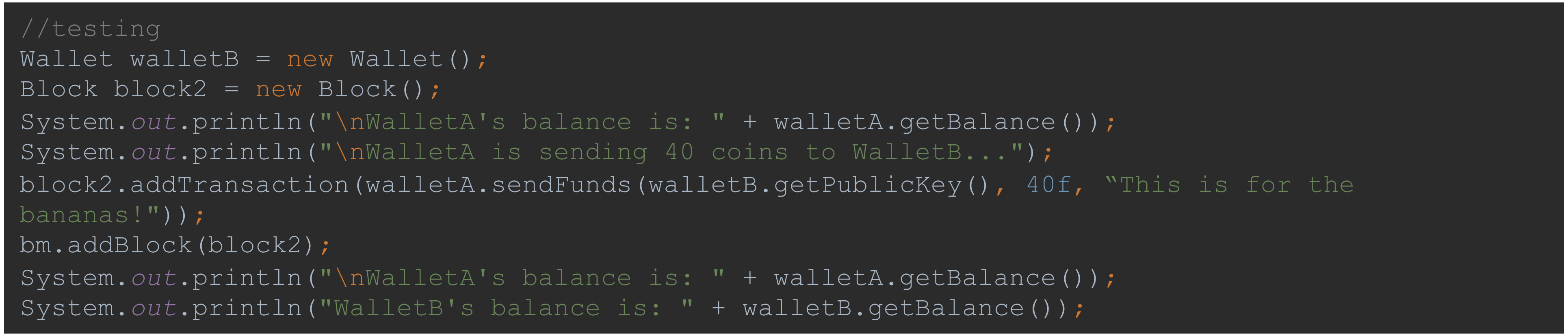
- Basic proof-of-concept Java package for incorporation into other code
- Not a full working peer-to-peer blockchain
- SHA-256 hashing
- Elliptic-Curve Key pairs



```
import blockmatrix.*;

public class Main {

    public static void main(String[] args) {
        BlockMatrix bm = new BlockMatrix(5);
        bm.setUpSecurity();

        //Create wallets:
        Wallet walletA = new Wallet();
        bm.generate(walletA, 200f);
```

An example of use

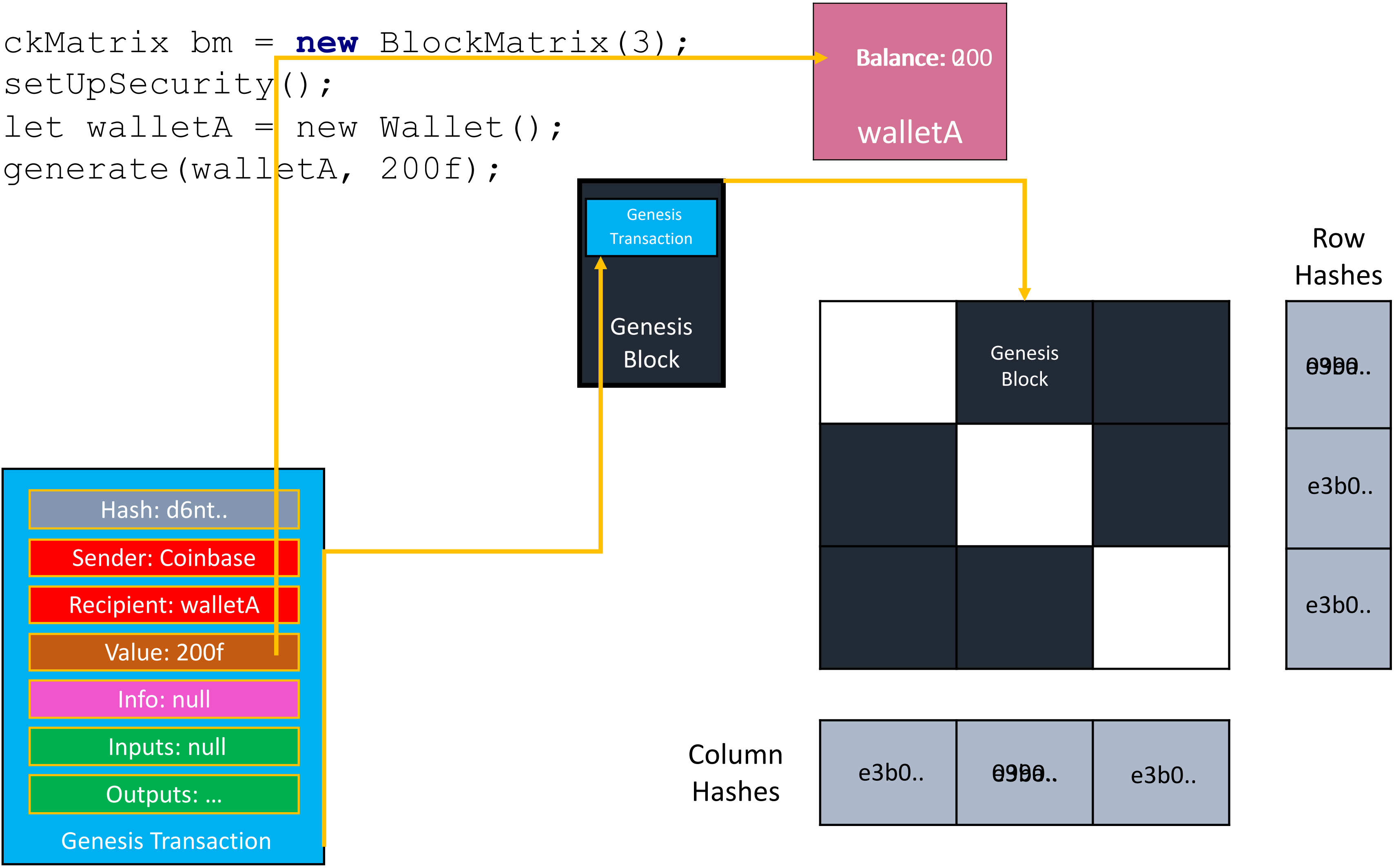
- Create wallets: `Wallet walletB = new Wallet();`
- Create Blocks: `Block block2 = new Block();`
- Create transactions
  - `Transaction tr = walletA.sendFunds(walletB.getPublicKey(), 40f, “This is for the bananas!");`
- Add the transactions to blocks: `block2.addTransaction(tr);`
- Add the blocks to the block matrix `bm.addBlock(block2);`



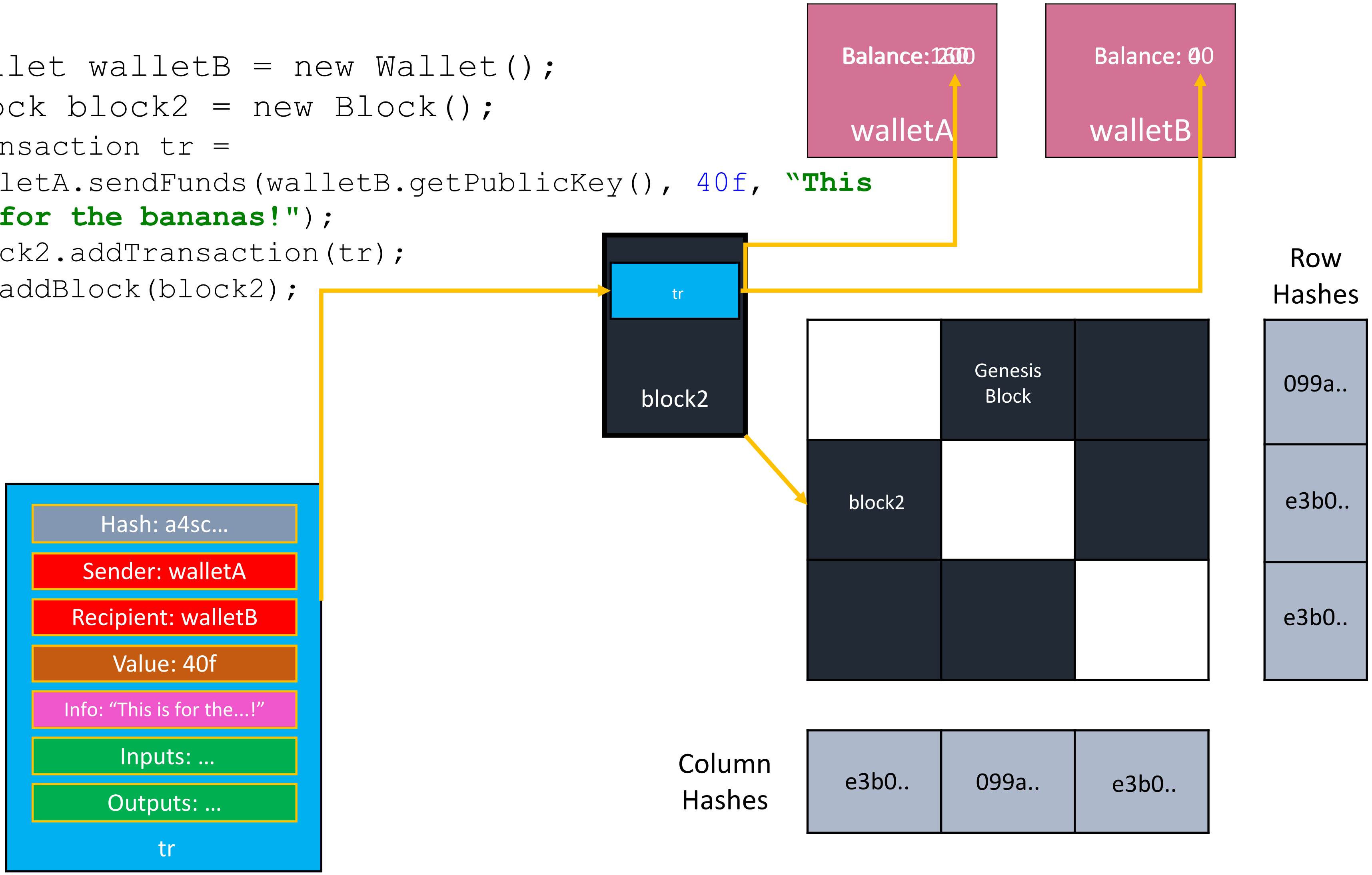
```
//testing
Wallet walletB = new Wallet();
Block block2 = new Block();
System.out.println("\nWalletA's balance is: " + walletA.getBalance());
System.out.println("\nWalletA is sending 40 coins to WalletB...");
block2.addTransaction(walletA.sendFunds(walletB.getPublicKey(), 40f, “This is for the
bananas!"));
bm.addBlock(block2);
System.out.println("\nWalletA's balance is: " + walletA.getBalance());
System.out.println("WalletB's balance is: " + walletB.getBalance());
```

- Clearing info in blocks: `bm.clearInfoInTransaction(2, 0);`

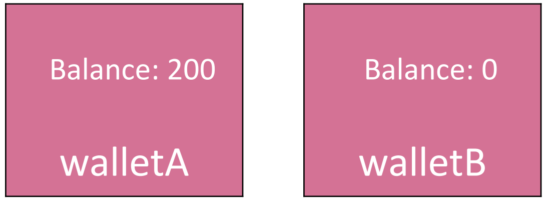
```
BlockMatrix bm = new BlockMatrix(3);
bm.setUpSecurity();
Wallet walletA = new Wallet();
bm.generate(walletA, 200f);
```



```
Wallet walletB = new Wallet();
Block block2 = new Block();
Transaction tr =
walletA.sendFunds(walletB.getPublicKey(), 40f, “This
is for the bananas!");
block2.addTransaction(tr);
Bm.addBlock(block2);
```



```
Wallet walletB = new Wallet();
Block block2 = new Block();
Transaction tr =
walletA.sendFunds(walletB.getPublicKey(), 40f, "This
is for the bananas!");
block2.addTransaction(tr);
bm.addBlock(block2);
bm.clearInfoInTransaction(2, 0);
```

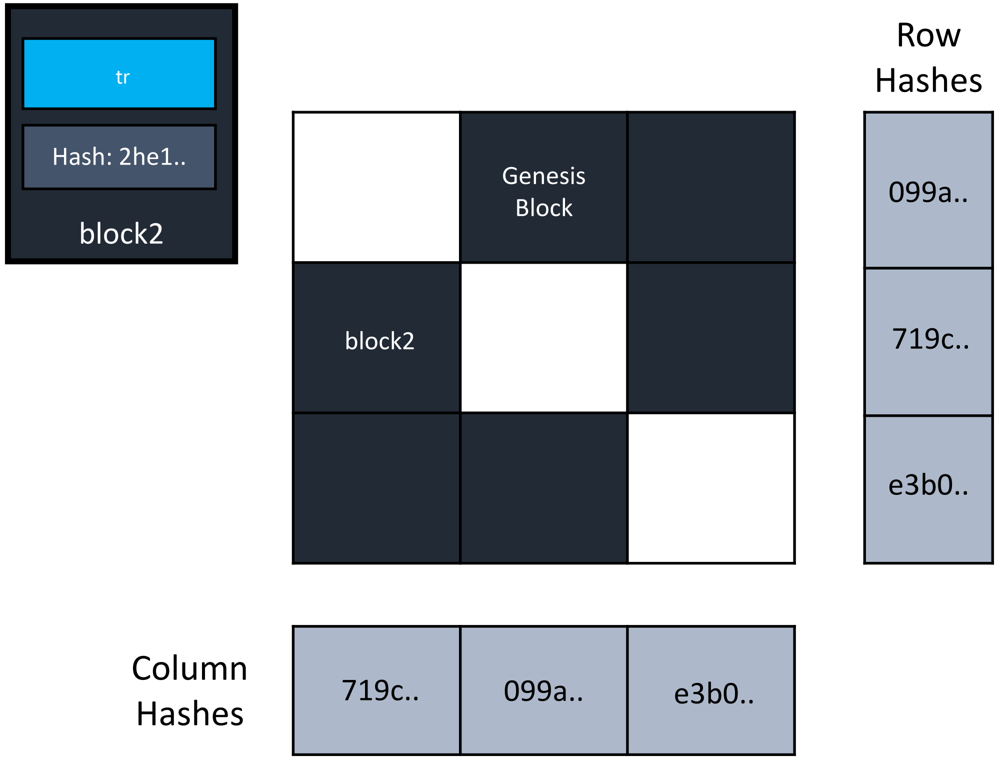


Balance: 200

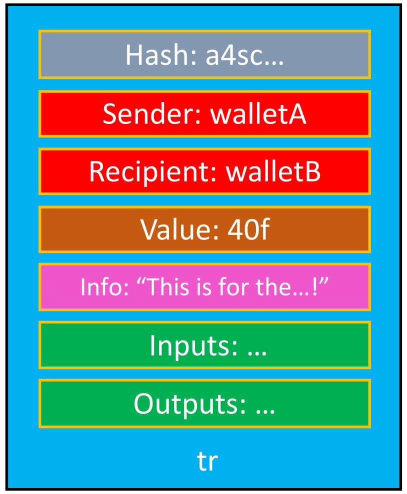
walletA

Balance: 0

walletB



				Row Hashes
		Genesis Block		099a..
	block2			719c..
				e3b0..
Column Hashes	719c..	099a..	e3b0..	



Hash: a4sc...

Sender: walletA

Recipient: walletB

Value: 40f

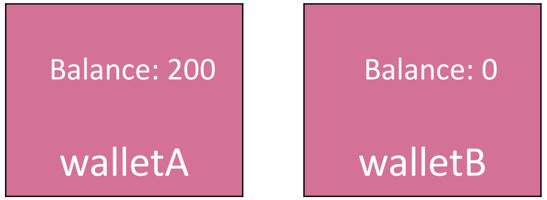
Info: "This is for the...!"

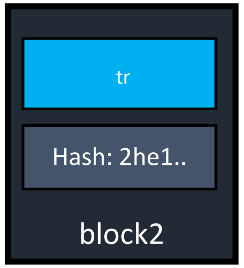
Inputs: ...

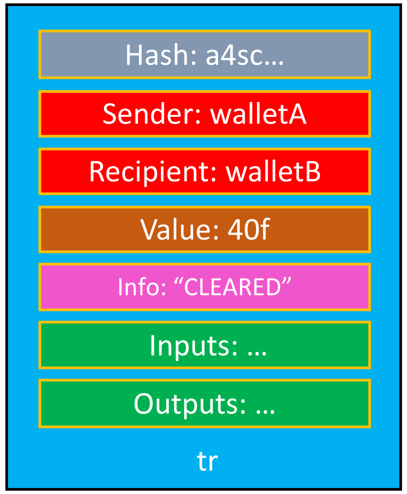
Outputs: ...

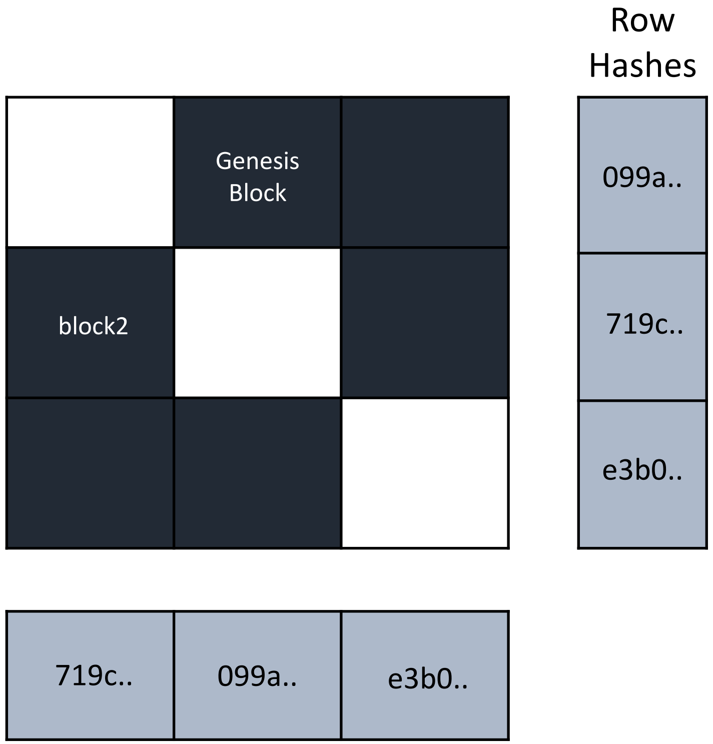
tr

```
Wallet walletB = new Wallet();
Block block2 = new Block();
Transaction tr =
walletA.sendFunds(walletB.getPublicKey(), 40f, “This
is for the bananas!");
block2.addTransaction(tr);
bm.addBlock(block2);
bm.clearInfoInTransaction(2, 0);
```






Hash: a4sc...
Sender: walletA
Recipient: walletB
Value: 40f
Info: “CLEARED”
Inputs: ...
Outputs: ...
tr


Row Hashes

	Genesis Block		099a..
block2			719c..
			e3b0..
719c..	099a..	e3b0..	

Column Hashes

```
Wallet walletB = new Wallet();
Block block2 = new Block();
Transaction tr =
walletA.sendFunds(walletB.getPublicKey(), 40f, "This
is for the bananas!");
block2.addTransaction(tr);
bm.addBlock(block2);
bm.clearInfoInTransaction(2, 0);
```

Balance: 200

walletA

Balance: 0

walletB

tr

Hash: 2he1..

block2

Hash: 34ds...

Sender: walletA

Recipient: walletB

Value: 40f

Info: “CLEARED”

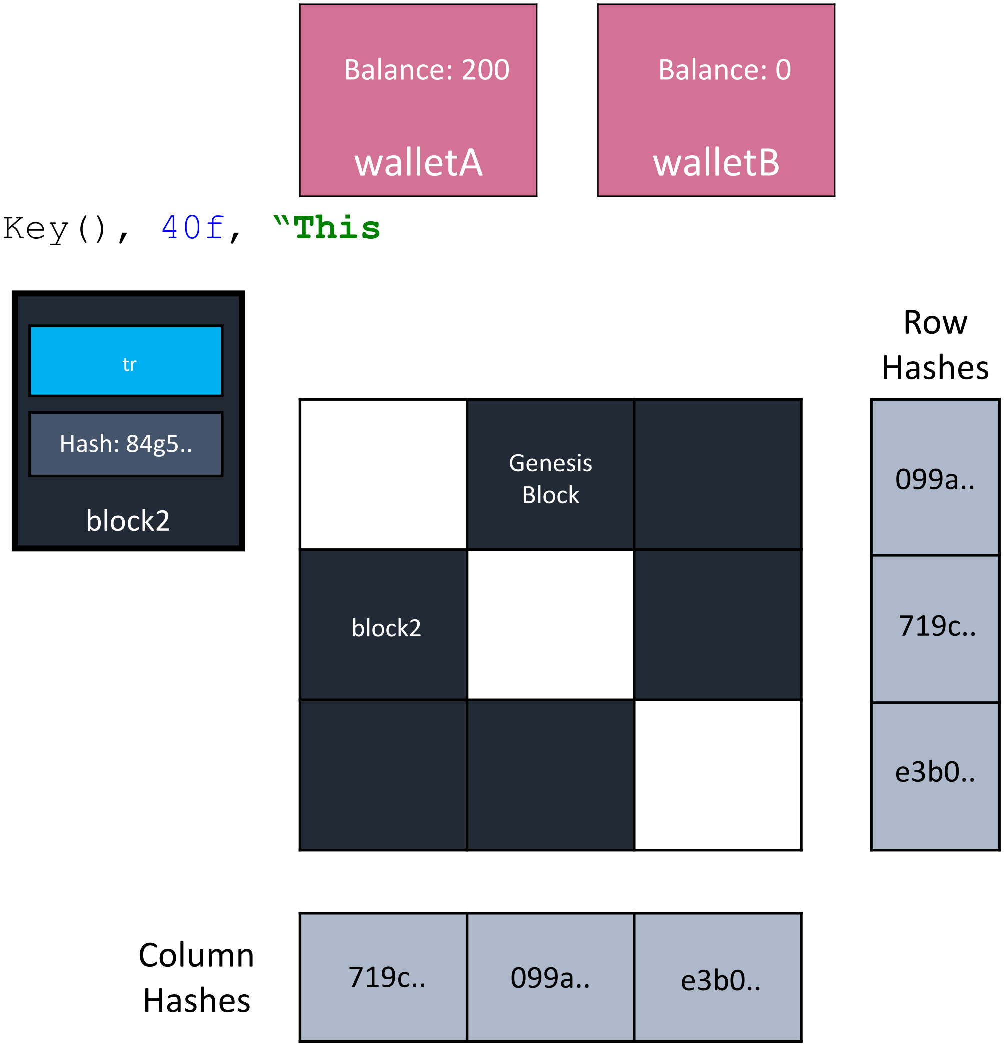
Inputs: …

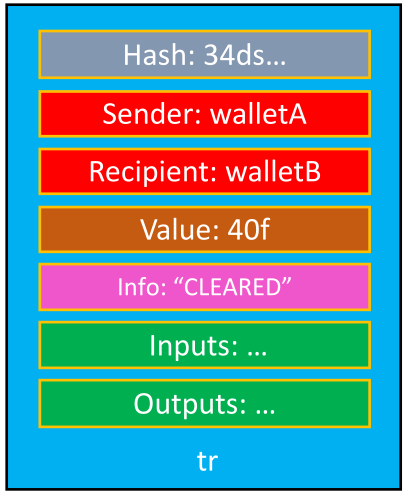
Outputs: …

tr

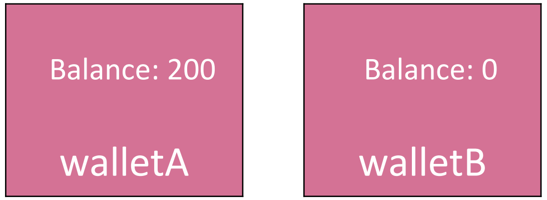
			Row Hashes
	Genesis Block		099a..
block2			719c..
			e3b0..
719c..	099a..	e3b0..	Column Hashes

```
Wallet walletB = new Wallet();
Block block2 = new Block();
Transaction tr =
walletA.sendFunds(walletB.getPublicKey(), 40f, “This
is for the bananas!");
block2.addTransaction(tr);
bm.addBlock(block2);
bm.clearInfoInTransaction(2, 0);
```





```
Wallet walletB = new Wallet();
Block block2 = new Block();
Transaction tr =
walletA.sendFunds(walletB.getPublicKey(), 40f, "This
is for the bananas!");
block2.addTransaction(tr);
bm.addBlock(block2);
bm.clearInfoInTransaction(2, 0);
```

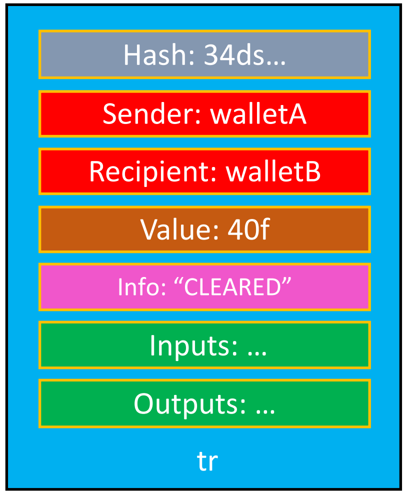


Balance: 200

walletA

Balance: 0

walletB



Hash: 34ds...

Sender: walletA

Recipient: walletB

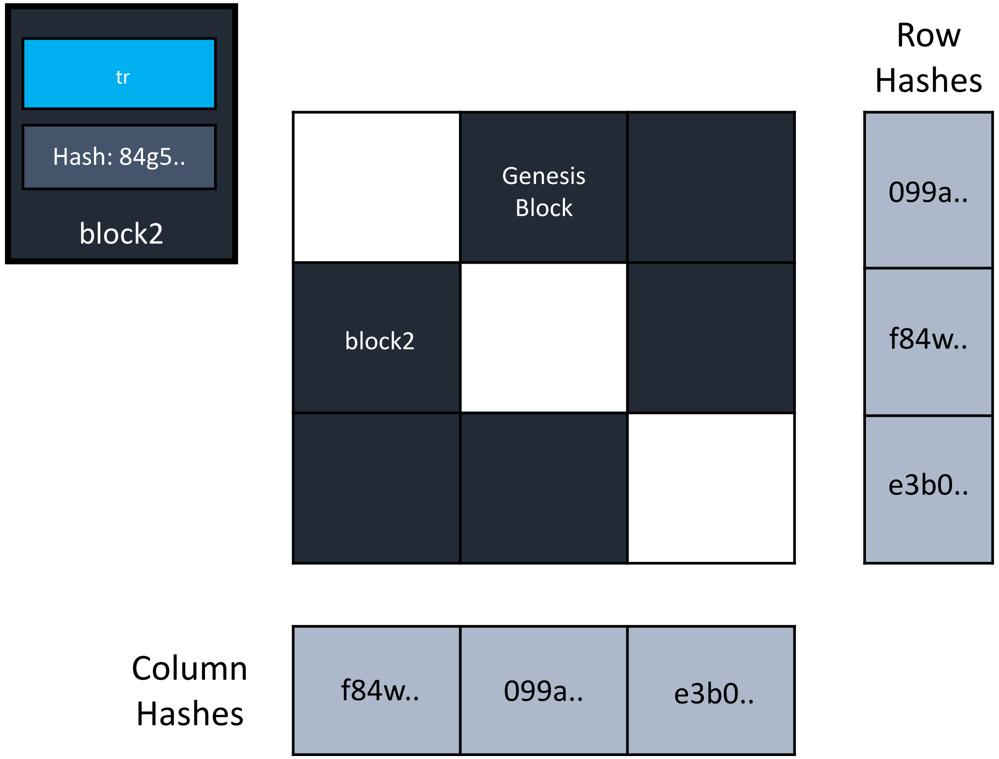
Value: 40f

Info: "CLEARED"

Inputs: ...

Outputs: ...

tr



tr

Hash: 84g5..

block2

Row Hashes

Column Hashes

	f84w..	099a..	e3b0..
099a..		Genesis Block	
f84w..	block2		
e3b0..			

Ensuring Matrix Validity

- isMatrixValid() method
    - Encompassing function which checks if blockmatrix is secure
- Features:
    - Checks every block and ensures its hash is what it should be
    - Checks every row and column and ensures its hash is what it should be
    - Checks every transaction in each block and makes sure that
        - The transactions signature is valid
        - Inputs are equal to outputs in the transaction
        - Etc.
    - Checks that all deletions/modifications of data changed only one row and column hash, and the rest are unchanged

Future of Datablock Matrix

- Consider proof of work or alternate consensus schemes
- Web tool to easily see structure of your DatablockMatrix
- Extension to peer-to-peer system
- Creation of generic DatablockMatrix data structure which can be used for any purpose
- Implementation in existing blockchains
  - Multichain
  - Hyperledger Fabric
- Consider higher dimension structures – can be done, but is there any value?

More information:

NIST publication

- Kuhn, D. R. (2018). A Data Structure for Integrity Protection with Erasure Capability
- https://csrc.nist.gov/publications/detail/white-paper/2018/05/31/data-structure-for-integrity-protection-with-erasure-capability/draft

Github project:

- https://github.com/usnistgov/blockmatrix

Acknowledgements

- Arsen Klyuev, Johns Hopkins Univ
- Dylan Yaga, NIST
- Gokhan Kocak, Asena, Inc.