Quantum Blockchain's „QKDBase" – the first QKD and QRNG based quantum blockchain model completed

Summary

In this communication we report on the completion of the second phase of the development of the technology that bridges domains of blockchain and of quantum cryptography. Theoretical base of this work was first proposed in a number of publications from the founders of Quantum Blockchains startup company. The most important among them is „Towards Quantum-Secured Permissioned Blockchain: Signature, Consensus, and Logic" work published in September 2019[1], where we have described, in purely theoretical terms, the possible algorithms for securitization, consensus and logical smart-contracts which could form a blockchain secured from the threats of emerging quantum computers. Then, during incubation of our startup, we have built an MVP class solution where quantum-secured links were simulated by classical channels (HTTP connections using classical TCP/IP network)[2].

Since our startup obtained its first funding[3] in October 2021, we started evolving our MVP code base into the model that can use real, commercially available QKD devices. As the QKD devices are very expensive and run on expensive infrastructure, we entered into partnership with one of the suppliers of the technology, QNU Labs, and secured a dedicated pair of QNU Labs' Armos QKD system for our disposal. To avoid the infrastructure creation costs, the pair was installed for us in QNU Labs laboratory in Bangalore, India, and the communication was provided by installing classically secured (by an SSL layer on top of HTTP protocol) proxy servers. Two such servers for the fictional cryptographic couple "Alice" and "Bob" have been made available via ETSI 014 QKD protocol[4]. The long-term tests using QNU labs Armos devices (more than 4 months) enabled us to replace the simulated quantum communication by the QKD real devices in a transparent remote mode.

At the very end of the development period we also were able to access a pair of ID Quantique devices made available for us at PSNC (Poznań Supercomputer and Networking Center). One of the QKD links between a pair of the nodes of our blockchain was set up to use that link via VPN secured access to PSNC resources. However as the number of rounds involving these links was too low, we could not treat these tests as definitive.

The second type of quantum device we used is QRNG – Quantum Random Number Generator. We used ID Quantique Quantis QRNG and the software layer (REST Web API) developed by our company[5].

[1] https://doi.org/10.3390/e21090887
[2] https://www.quantumblockchains.io/mvp/
[3] Grant of the Polish Agency for Enterprise Development - POPW.01.01.02-06-0031/21 *„Opracowanie i wdrożenie rynkowe innowacyjnych produktów i usług z zakresu kryptografii kwantowej związanych z koncepcją Kwantowego systemu Blockchain"*
[4] https://www.etsi.org/committee/1430-qkd
[5] https://www.quantumblockchains.io/current-services/qrng-numeric/

This communication describes the conceptual framework and the implementation of our system which represents the first real blockchain model running on the real, commercially available QKD devices.

Conceptual framework

QKDBase is based on the our previous theoretical model described in *"Towards Quantum-Secured Permissioned Blockchain: Signature, Consensus, and Logic"* scientific paper.

The most important elements of the QKDBase conceptual framework are:

1) Blockchain security:
    a. Toeplitz Hash Message Authentication Code
    b. Toeplitz Group Signature
2) Consensus algorithm:
    a. The QSYAC Protocol

All these elements are clearly described in the aforementioned paper.

From theoretical formulation to the design of the code

In more software developmental/algorithmic language we can describe the consensus protocol implementation in the following way:

- A transaction (an object serialized to JSON string representation) is sent by the client to all peers.
- The proposing peer establishes a Toeplitz matrix with all neighboring peers (QKD is used).
- The proposing peer establishes an One-Time-Pad as a random binary string with all neighboring peers (QKD is used).
- The proposing peer generates the Toeplitz Group Signature.

An example:

- The Toeplitz matrix is generated by generating a random binary string with 69 digits (representing the first row and first column) and populating diagonal values with the same values.
- The One-Time Pad is a random binary string with 35 digits.
- A simple transaction data of N characters is prepared which are then parsed to a binary string using UTF8.
- Toeplitz hash is calculated by multiplying Toeplitz matrix with transaction data. Then the result is used for modulo 2 calculation. The last step is calculating bitwise XOR between modulo result and the OTP.
- Toeplitz Group Signature is an array containing all calculated Toeplitz hashes, one for each connection between proposing peer and neighboring ones (in our scenario we have 3 Toeplitz hashes).
- The proposing peer generates Toeplitz hash for itself and adds it to the Toeplitz Group Signature. Without it, the proposal peer wouldn't be able to vote because it wouldn't have data to hash the transaction and send it with a vote request.

- The proposing peer creates a proposal block and sends it to all neighboring peers together with Toeplitz Group Signature (classical channel).

After receiving the proposal block the other peers do:

- Verify if Toeplitz Group Signature is correct: calculate Toeplitz hash using Toeplitz matrix, block proposal data and the OTP established before with proposing peer and check if Toeplitz Group Signature has the same hash as calculated Toeplitz hash.
- If Toeplitz Group Signature is correct they store block proposal.
- If Toeplitz Group Signature is correct they store Toeplitz Group Signature
- If Toeplitz Group Signature is correct they hash transaction (node hash + transaction + calculated Toeplitz hash) and store it (classical mode).
- Each peer generates a random array of all peers and sends a request to vote to the first one together with a hashed transaction (classical mode).

When peers get a request to vote they:

- Wait for the block proposal and Toeplitz Group Signature
- Hash transaction for each neighboring peer using transaction from block proposal, node hash and Toeplitz Group Signature.
- Check if the calculated hashed transaction is the same as one received from a voting request.
- If hashes are the same, they send to all peers request to increase their vote number by 1.
- If the number of the votes is equal or bigger than 12 (together with one just added), they send to all peers that they should add a proposal block to the blockchain (classical channel).
- If the number of the votes is less than 12 (together with one just added), they send a request to the next peer in the queue to vote together with a hashed transaction (classical channel).
- After adding the block to blockchain each peer clears Toeplitz Group Signature array, the One-Time Pads, transaction hash, block proposal and votes number.

Establishment of the Toeplitz value and one-time pad follows:

- The first peer sends a request to check if the second peer has Toeplitz value/one-time pad with the node hash of the first peer
- If yes, the first peer checks if it also has the same Toeplitz value/one-time pad and if everything is correct, the establishment is finished.
- If the second peer doesn't have corresponding Toeplitz value/one-time pad the first peer generates Toeplitz value/one-time pad and sends it to the second peer

The implementation

QKDBase was implemented using NodeJS programming and execution environment, and TypeScript, which is a syntactical superscript of JavaScript. The deployment was done using Docker platform-as-a-service virtualization software. Standard deployment mechanisms, where QKDBase nodes were run on separated machines were also tested.

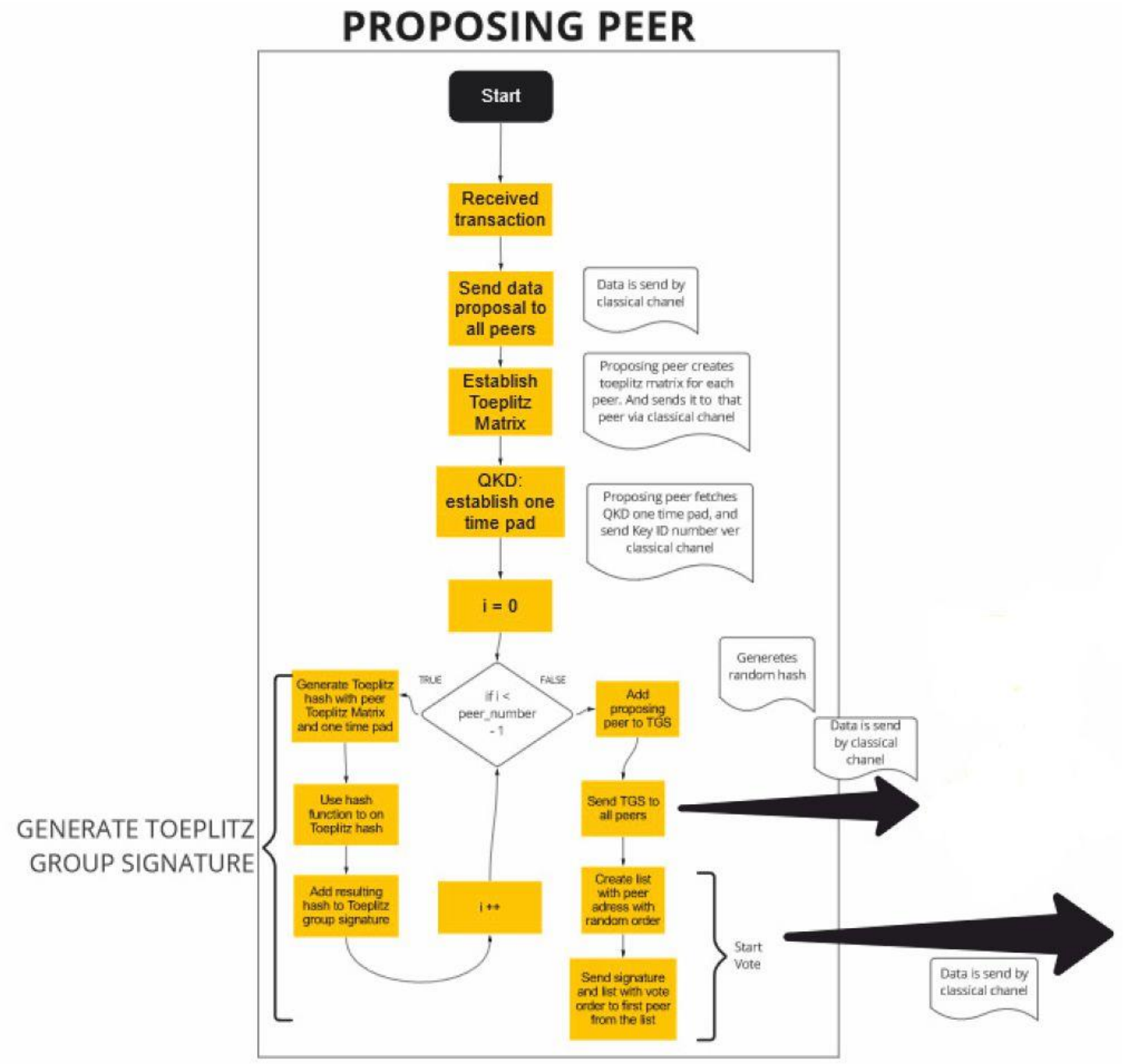
Transaction processing was initiated by Python scripts which invoked HTTP calls to send the transaction to the selected blockchain node.

Relation of the implementation to the conceptual/theoretical framework

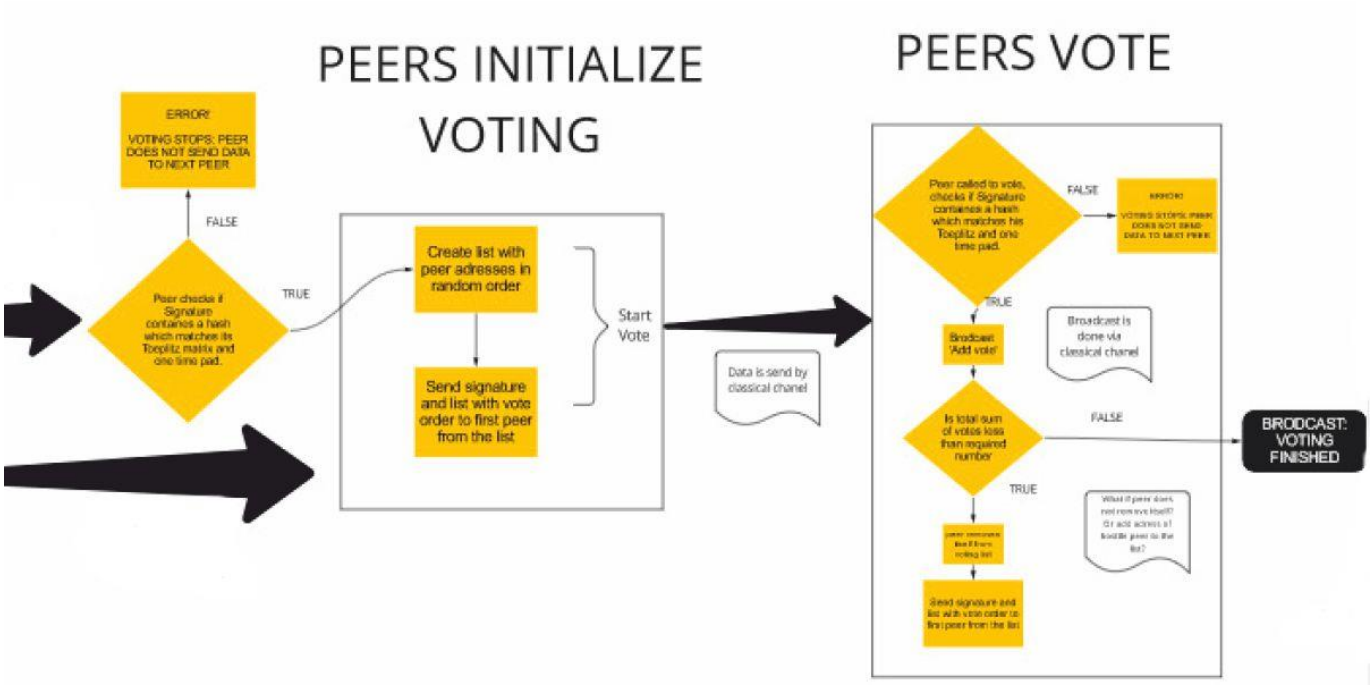
During the implementation of the QKDBase quantum blockchain we had to make some adjustments and modification to the original design described in the "conceptual framework" section above. To illustrate the final flow in the actually implemented blockchain, we present the flowchart of the main operations:

- Transaction proposal
- Consensus mechanism (voting)

The transaction proposal



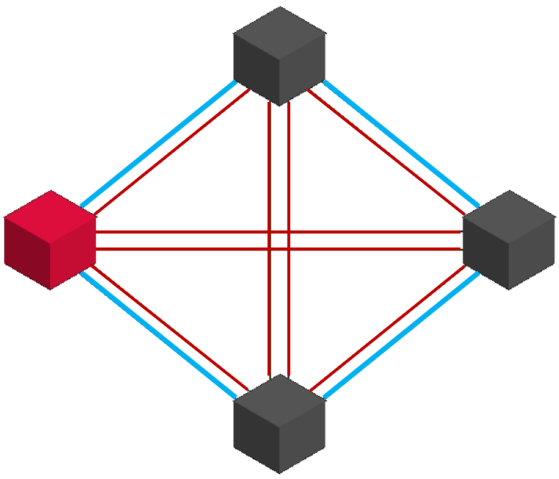
Voting (achieving consensus)



The QKDBase architecture

QKDBase uses hybercube network 2nd order topology with 4 nodes. In the future we will use other dimensions of the hypercube networks for the quantum blockchain.

The current architecture is depicted in the following diagram:



QKD Links are depicted in blue, classical links are depicted in red.

As we had access to lower number of physical QKD links than logical node-to-node links, we have used multiplexing technique, where a single physical link was used for many logical links.

Also, we have used two types of distribution of the nodes in our lab: physical (two bare-metal servers with two nodes each) and simulated (four independent Docker containers on a single bare-metal server)

The physical QKD links used by QKDBase

The majority of QKDBase runs were performed using QNU Labs ARMOS QNLX 210 QKD pair of devices[6] connected by 60km of the fiber (SMF-28e) setup in Bangalore, India (see Appendix A for more data about the Armos QKD devices).

The remote access was provided by two proxy servers that were delivering ETSI 014 protocol (see below) payload over encrypted public Internet connections and authenticated through standard PKI certificates.

In some of the tests, we have used ID Quantique Clavis3 QKD platform[7] – the pair of devices was set up inside PSNC Lab in Poznań, Poland. The access to these devices was provided to us via highly secured VPN connection. We then used them using ETSI 014 protocol. See Appendix B for more information about the devices.

However, while the ID Quantique devices worked properly with our blockchain, the number of tests was far too low for the inclusion of the results into our report.

The ETSI protocol

To communicate with QKD devices we have used ETSI protocol described in ETSI Group Specification: "Quantum Key Distribution (QKD) - Protocol and data format of REST-based key delivery API"[8], known as ETSI 014 (ETSI stands for European Telecommunications Standards Institute and is one of the European Standards Organizations – ESO).

ETSI 014 is a communication protocol and data format for a quantum key distribution (QKD) network to supply highly secured cryptographic keys to any application that uses quantum cryptography. It provides interoperability between devices from different vendors. The protocol is implemented as a REST (REpresentational State Transfer) WebAPI. The REST-based WebAPI specifies the format of the URI calls, the use of communication protocols (HTTPS), and data format for encoding of parameters and for responses, including cryptographic key material using JSON (JavaScript Object Notation) data serialization format.

We refer the reader of this report to the specification itself**Error! Bookmark not defined.** for all details of the protocol we used, and, for more general information about ETSI Quantum Communication standard – to the Industry Specification Group (ISG) on Quantum Key Distribution[9].

[6] https://www.qnulabs.com/armos-quantum-key-distribution/
[7] https://www.idquantique.com/quantum-safe-security/products/clavis3-qkd-platform-rd/
[8] https://www.etsi.org/deliver/etsi_gs/QKD/001_099/014/01.01.01_60/gs_qkd014v010101p.pdf
[9] https://www.etsi.org/committee/qkd

To shortly present the protocol let us present the typical sequence of calls we used to communicate between two blockchain nodes:

STATUS call – used to check status of the device (<A_IP> is the IP endpoint of "Alice" of the exchange, <B_IP> is the endpoint of the "Bob" of the exchange)

```
https://<A_IP>/api/v1/keys/quantumblockchains/status
```


```
{
    "Source_KME_ID": "KME1",
    "Target_KME_ID": "KME2",
    "Master_SAE_ID": "qnulabs",
    "Slave_SAE_ID": "quantumblockchains",
    "key_size": 1024,
    "stored_key_count": 12207,
    "Max_key_count": 100,
    "Max_key_per_request": 10,
    "Max_key_size": 1024,
    "Min_key_size": 4,
    "Max_SAE_ID_count": 1
}
```


KEY encoding – used to generate a key and to initiate its transmission to the other endpoint over the quantum channel.

In the response the call returns the cryptographic key for "Alice" and its identification (key_ID) that the other end ("Bob") will use to retrieve the key.

```
https:// <A_IP>/api/v1/keys/quantumblockchains/enc_keys
```


```
{
    "keys": [
        {
            "key_ID": "fe13b9bd-c0b8-42a5-8e15-d8625b472b3b",
            "key":
"2171a9482574999e6ac898cdd453db13d91de79e71c07fbe027bb2b9cebac2a5186c43ae64
3136939c91b5a6f941955a29072471134992ca8cdaf8b7e283b8256b791e498ea302c3528d4
67cfbad5d0a7df5a0487e03f667974926bbd903208aae5f3b0359fc1da51f76bb05d36bf435
c5599c268205c282b08a70895d920be166e21d4a28e65ab5d4f6c2ce5edf42caa969430a567
797482b982e011e3be38b6d9fdd2a73ffdf351028f4ff985004b80f8d464e2bf6ec7289839b
12413fa5fc637ec0f2704836ce039a99f891cef61fe771a16e40b83769a676852fb336d9764
5be0ff026205d018a0d6c87238c3682dc7d930c3bf9894c4d28438ad4e3ea26ed4a3fb345ed
d9cccf240aa7218c7690eb4a6ca6562abd29c46a0dd43645baf0309ef04d1692b51ae0e085e
5e7f0187d0d4200f47a7b681dc05b70f3bd9c91c94cd87beba13eb2076bd968c07149122139
b6e106b515a41c0de9d0c2571c91840a9eb9af7698a14c3424eb0d51dc5776df9a973b301b1
5405c7b19ae913fefaf81f39976e5e2acda12892c028b73739b1dfeabb63cd95fe8d2bb1d0d
41b64328dcbfa4efa010209dac4a91e0f6bc9ff1a723fb1ae4c2b0873945868a93611346340
1e72cf4d6d20a42a09fa1979db43e592b69971068be996bf6d7b8116f342f6bb02cc5cf44ba
c24acccc7d98b8b765f0063d6cffa9aeecd2e03fa76ce6d64a"
        }
    ]
}
```

KEY decoding – used to retrieve the key send by "Alice" on the other ("Bob") side using the key_ID provided.

```
https://<B_IP>/api/v1/keys/quantumblockchains/dec_keys?key_ID=fe13b9bd-
c0b8-42a5-8e15-d8625b472b3b
```


```
{
    "key_ID": "fe13b9bd-c0b8-42a5-8e15-d8625b472b3b",
    "key":
"2171a9482574999e6ac898cdd453db13d91de79e71c07fbe027bb2b9cebac2a5186c43ae64
3136939c91b5a6f941955a29072471134992ca8cdaf8b7e283b8256b791e498ea302c3528d4
67cfbad5d0a7df5a0487e03f667974926bbd903208aae5f3b0359fc1da51f76bb05d36bf435
c5599c268205c282b08a70895d920be166e21d4a28e65ab5d4f6c2ce5edf42caa969430a567
797482b982e011e3be38b6d9fdd2a73ffdf351028f4ff985004b80f8d464e2bf6ec7289839b
12413fa5fc637ec0f2704836ce039a99f891cef61fe771a16e40b83769a676852fb336d9764
5be0ff026205d018a0d6c87238c3682dc7d930c3bf9894c4d28438ad4e3ea26ed4a3fb345ed
d9cccf240aa7218c7690eb4a6ca6562abd29c46a0dd43645baf0309ef04d1692b51ae0e085e
5e7f0187d0d4200f47a7b681dc05b70f3bd9c91c94cd87beba13eb2076bd968c07149122139
b6e106b515a41c0de9d0c2571c91840a9eb9af7698a14c3424eb0d51dc5776df9a973b301b1
5405c7b19ae913fefaf81f39976e5e2acda12892c028b73739b1dfeabb63cd95fe8d2bb1d0d
41b64328dcbfa4efa010209dac4a91e0f6bc9ff1a723fb1ae4c2b0873945868a93611346340
1e72cf4d6d20a42a09fa1979db43e592b69971068be996bf6d7b8116f342f6bb02cc5cf44ba
c24acccc7d98b8b765f0063d6cffa9aeecd2e03fa76ce6d64a"
}
```


The project source code

Source code of QKDBase containing the QKDbase is available in the repository:

https://github.com/quantumblockchains/QKDBase

Use of QRNG in the QKDBase code

QKDBase software calls QRNG (Quantum Random Number Generator) hardware for the generation of a random array of peers. The quantum device we used for QRNG hardware is ID Quantique Quantis device[10] and the software layer (REST Web API) was developed by our company.

The WebAPI designed by our engineers and available as a product at:
https://www.quantumblockchains.io/current-services/

The QRNG call occurs at the beginning of each voting phase – method startVoting calls generateRandomArrayOfNodes which returns a random array of peers.

- generateRandomArrayOfNodes : `\QuantumBlockchains\peer\services\qrng.service.ts`

[10] https://www.idquantique.com/random-number-generation/products/quantis-qrng-chip/

Use of QKD in the QKDBase code

QKDBase software calls QKD service run by the QKD hardware using ETSI compliant webservices.

The QKD calls are used for securing communication between peers while establishing the Toeplitz matrix and the one-time pad. Before the start of the voting phase, the proposing peer establishes with each peer a unique Toeplitz matrix and a unique one time pad. The proposing peer calls the method **establishToeplitzMatrixWithQKD** which allows for the generation of the Toeplitz matrix from the received QKD key. The QKD's key ID is then sent to the other peer which calls the method **fetchAndStoreToeplitzMatrix**. This method allows her/him to create the same Toeplitz matrix from the given key identified by the key ID.

The same process is repeated for the one-time pad: proposing peer calls **establishOneTimePadWithQKD** to get a one-time pad from QKD key, and sends the key ID to the other peer. The other peer calls **fetchAndStoreQKDKey** to receive the same one time pad, as proposing peer.

Location of the methods in the QKDBase code:

- startVoting: `\QuantumBlockchains\peer\services\api.service.ts`
- establishToeplitzMatrixWithQKD, fetchAndStoreToeplitzMatrix: `\QuantumBlockchains\peer\services\toeplitzQKD.service.ts`
- establishOneTimePadWithQKD, fetchAndStoreQKDKey: `\QuantumBlockchains\peer\services\oneTimePadQKD.service.ts`

The deployment of QKDBase

QKDBase was deployed in two modes:

1) Under Docker container mechanism with all nodes running on a single machine
2) Directly on the operating system level with two nodes running on one server and two other nodes running on the another one.

The typical multiterminal output of the first mode is presented in the following picture:

```
14:14:44:262 - Sending block proposal to http://peer_2:3016
14:14:44:266 - Sending block proposal to http://peer_3:3016
14:14:44:269 - Sending block proposal to http://peer_4:3016
14:14:44:282 - Establishing one time pad with peers using QKD
14:14:44:282 - Sending get QKD key request
14:14:45:076 - QKD key received. Key ID: d82ee5a3-cc53-4da1-b23a-a988abd0cfaf
14:14:45:077 - Adding established one time pad
14:14:45:077 - Sending QKD key id to http://peer_2:3016
14:14:45:974 - Sending get QKD key request
14:14:46:691 - QKD key received. Key ID: 371f5c02-b1ec-4d2b-910c-c4a3fa4ba6f1
14:14:46:692 - Adding established one time pad
14:14:46:692 - Sending QKD key id to http://peer_3:3016
14:14:47:279 - Sending get QKD key request
14:14:47:997 - QKD key received. Key ID: 8374deda-3ce1-4da6-a48a-0bfaf227414c
14:14:47:997 - Adding established one time pad
14:14:47:997 - Sending QKD key id to http://peer_4:3016
14:14:48:899 - Establishing toeplitz matrix with peers using QKD
14:14:48:899 - Sending get QKD key request
14:14:49:911 - QKD key received. Key ID: 53deddef-66dc-4f92-94ac-04bb0aacc95c
14:14:49:924 - Adding established Toeplitz matrix
14:14:50:820 - Sending get QKD key request
14:14:51:544 - QKD key received. Key ID: a43f128c-476d-4edd-b4f8-57bd9573bd9b
14:14:51:555 - Adding established Toeplitz matrix
14:14:52:111 - Sending get QKD key request
14:14:52:802 - QKD key received. Key ID: 3bbac03a-22d0-4a96-98aa-6de4b0d7dbb2
14:14:52:812 - Adding established Toeplitz matrix
```

```
14:14:28:314 - Peer listening
14:14:28:671 - Adding new node
14:14:44:268 - Received block proposal
14:14:44:268 - Storing block proposal
14:14:46:696 - Received QKD key id
14:14:46:696 - Fetching QKD key and storing as one time pad key
14:14:46:696 - Fetching QKD key
14:14:46:696 - Sending get QKD key by id request
14:14:47:277 - QKD key received. Key ID: 371f5c02-b1ec-4d2b-910c-c4a3fa4ba6f1
14:14:47:278 - Adding established one time pad
14:14:51:558 - Received toeplitz vector id
14:14:51:558 - Fetching toeplitz vector and storing as toeplitz matrix
14:14:51:559 - Fetching toeplitz vector
14:14:51:559 - Sending get QKD key by id request
14:14:52:096 - QKD key received. Key ID: a43f128c-476d-4edd-b4f8-57bd9573bd9b
14:14:52:108 - Adding established Toeplitz matrix
14:14:53:477 - Received Toeplitz Group Signature
14:14:53:477 - Calculating Toeplitz Hash
14:14:53:495 - Verifying data proposal signature
14:14:53:495 - Storing Toeplitz Group Signature
14:14:53:495 - Storing my hashed transaction
14:14:53:495 - Starting voting, creating peer queue
14:14:53:495 - Generating random peer array using QRNG
14:14:53:496 - Sending get QRNG random array request
https://qrng.qbck.io/9627fecf-f335-41b3-9560-82875c80afe9/qbck/block/short?size=15&min=0&max=3
14:14:53:579 - Received add vote request
```

2: Terminal

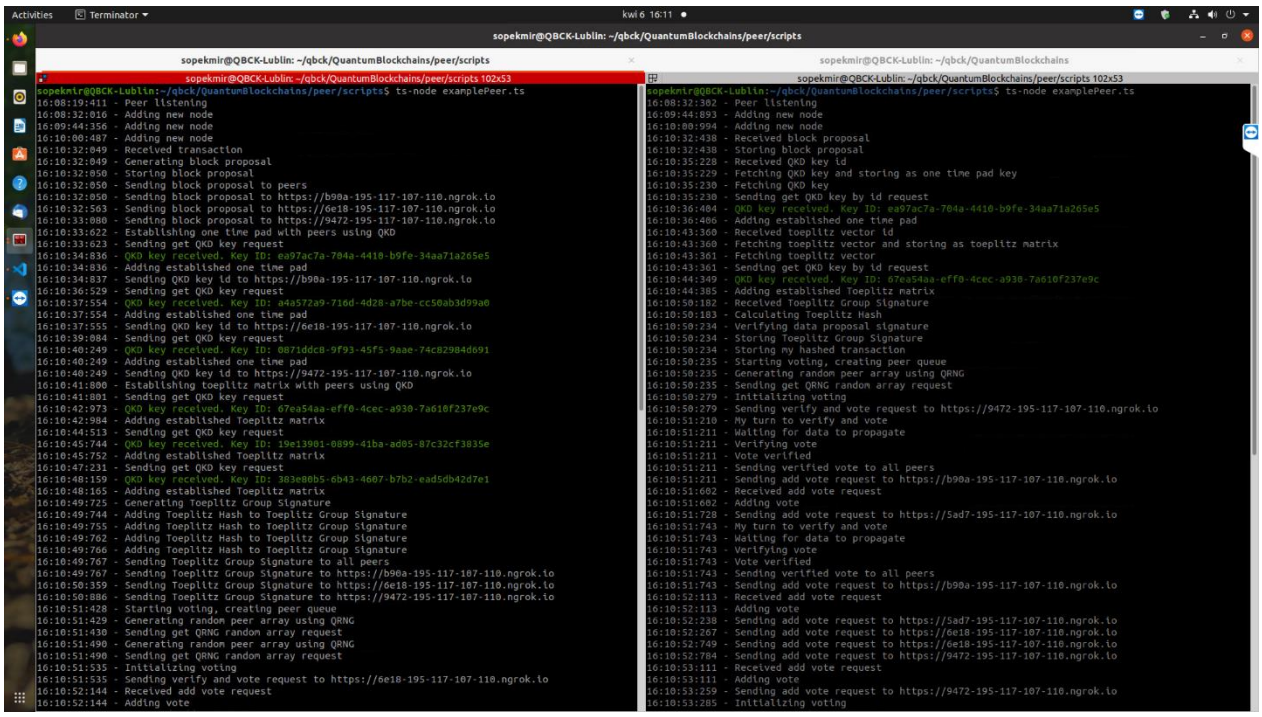
```
14:14:27:926 - Peer listening
14:14:28:303 - Adding new node
14:14:28:661 - Adding new node
14:14:44:264 - Received block proposal
14:14:44:265 - Storing block proposal
14:14:45:080 - Received QKD key id
14:14:45:080 - Fetching QKD key and storing as one time pad key
14:14:45:080 - Fetching QKD key
14:14:45:080 - Sending get QKD key by id request
14:14:45:970 - QKD key received. Key ID: d82ee5a3-cc53-4da1-b23a-a988abd0cfaf
14:14:45:971 - Adding established one time pad
14:14:49:929 - Received toeplitz vector id
14:14:49:929 - Fetching toeplitz vector and storing as toeplitz matrix
14:14:49:929 - Fetching toeplitz vector
14:14:49:929 - Sending get QKD key by id request
14:14:50:883 - QKD key received. Key ID: 53deddef-66dc-4f92-94ac-04bb0aacc95c
14:14:50:817 - Adding established Toeplitz matrix
14:14:53:454 - Received Toeplitz Group Signature
14:14:53:454 - Calculating Toeplitz Hash
14:14:53:473 - Verifying data proposal signature
14:14:53:473 - Storing Toeplitz Group Signature
14:14:53:473 - Storing my hashed transaction
14:14:53:473 - Starting voting, creating peer queue
14:14:53:473 - Generating random peer array using QRNG
14:14:53:473 - Sending get QRNG random array request
https://qrng.qbck.io/9627fecf-f335-41b3-9560-82875c80afe9/qbck/block/short?size=15&min=0&max=3
```

4: Terminal

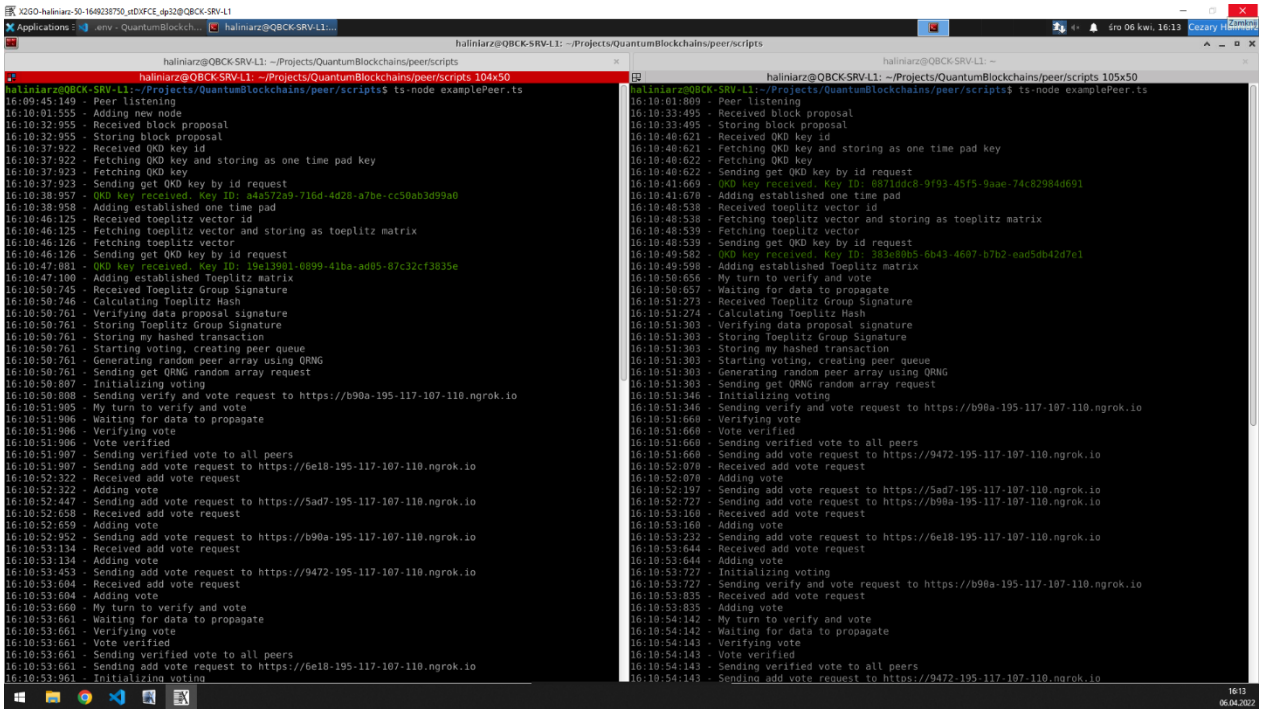
```
14:14:53:624 - Waiting for data to propagate
14:14:53:624 - Verifying vote
14:14:53:624 - Vote verified
14:14:53:624 - Sending verified vote to all peers
14:14:53:624 - Sending add vote request to http://peer_4:3016
14:14:53:625 - Sending add vote request to http://peer_3:3016
14:14:53:626 - My turn to verify and vote
14:14:53:626 - Waiting for data to propagate
14:14:53:626 - Verifying vote
14:14:53:626 - Vote verified
14:14:53:626 - Sending verified vote to all peers
14:14:53:626 - Sending add vote request to http://peer_4:3016
14:14:53:628 - Received add vote request
14:14:53:628 - Adding vote
14:14:53:629 - Initializing voting
14:14:53:629 - Sending verify and vote request to http://peer_1:3016
14:14:53:631 - Sending add vote request to http://peer_3:3016
14:14:53:632 - Received add vote request
14:14:53:632 - Adding vote
14:14:53:634 - Sending add vote request to http://peer_1:3016
14:14:53:636 - Sending add vote request to http://peer_1:3016
CONSENSUS ACHIEVED
14:14:53:639 - Sending request to add block to chain to all peers
14:14:53:640 - Sending add block to chain request to http://peer_1:3016
14:14:53:641 - Sending add vote request to http://peer_2:3016
14:14:53:642 - Sending add vote request to http://peer_2:3016
```

The output illustrates (the green lines) the generation of keys and their transfer between nodes using QNU Labs QKD devices.

In the second mode, the first server output of the first two nodes is displayed here:



And the second server output is shown here:



It is clearly visible when the quantum communication over QKD is used.

Results of tests

Executing blockchain transactions

QKDBase was used for tests for about 2 months. In a typical session tens to several thousand of transactions were performed. To represent some arbitrary data, the body of each transaction contained a piece of poetry and some random streams of data.

The chains, represented by a sequence of JSON objects were produced:


```
{"index":1,"previousBlockHash":"eac5cc20452c3814f6ff38a474f86de2dbc008f81d296b353a318e6546dba2a5","data":"Litwo, Ojczyzno moja! ty jestes jak zdrowie","timestamp":1649859284261,"hash":"dd234d58eecd0523fe40c67670cbc2516020db679b719549f9bc449a4c5bcb03"}

{"index":2,"previousBlockHash":"dd234d58eecd0523fe40c67670cbc2516020db679b719549f9bc449a4c5bcb03","data":"Litwo, Ojczyzno moja! ty jestes jak zdrowie","timestamp":1649859403357,"hash":"0a10823be1bedbaca7a35199be762c537c83909c6c5dad025b5dd52d038de044"}

{"index":3,"previousBlockHash":"0a10823be1bedbaca7a35199be762c537c83909c6c5dad025b5dd52d038de044","data":"Litwo, Ojczyzno moja! ty jestes jak zdrowie","timestamp":1649859423005,"hash":"5e4bc435aac948f92357533b341a11cd8fc25aa114d81c1d4a4f976b1fca9127"}

{"index":4,"previousBlockHash":"5e4bc435aac948f92357533b341a11cd8fc25aa114d81c1d4a4f976b1fca9127","data":"Litwo, Ojczyzno moja! ty jestes jak zdrowie","timestamp":1649874291477,"hash":"71c5f02cd5e6befb5a7d606ac9371208f7369b95c8d5774b0b9a89ecef98ceef"}
```

{"index":5,"previousBlockHash":"71c5f02cd5e6befb5a7d606ac9371208f7369b95c8d5774b0b9a89ecef98ceef","data":"Litwo, Ojczyzno moja! ty jestes jak zdrowie","timestamp":1649881739791,"hash":"daea76a2bb1788f1f4694188c0b8d3ad41a2a887225e1c0064a3a664fc19450d"}

{"index":6,"previousBlockHash":"daea76a2bb1788f1f4694188c0b8d3ad41a2a887225e1c0064a3a664fc19450d","data":"Litwo, Ojczyzno moja! ty jestes jak zdrowie","timestamp":1649881751756,"hash":"575fb41b3869f760b0b36bc777cc0a811e816606cfef356d63dd452aa0adaa9a"}

{"index":7,"previousBlockHash":"575fb41b3869f760b0b36bc777cc0a811e816606cfef356d63dd452aa0adaa9a","data":"Litwo, Ojczyzno moja! ty jestes jak zdrowie","timestamp":1649881773386,"hash":"fc9f52ea44f5c51a9fa70ebf0d932b53ff6565371702f010d2e66d43da0c48d9"}

{"index":8,"previousBlockHash":"fc9f52ea44f5c51a9fa70ebf0d932b53ff6565371702f010d2e66d43da0c48d9","data":"Litwo, Ojczyzno moja! ty jestes jak zdrowie","timestamp":1649911086569,"hash":"0adfde42b1b1124c678b5b3a5be5efa4771ac0ce4feda171493a32afa55bce0c"}

{"index":9,"previousBlockHash":"0adfde42b1b1124c678b5b3a5be5efa4771ac0ce4feda171493a32afa55bce0c","data":"Litwo, Ojczyzno moja! ty jestes jak zdrowie","timestamp":1649911098064,"hash":"aadf39119fd2f8cd0e48eca3ad4671946cb15e16d7b137f64226d7efd7852174"}

...

{"index":873,"previousBlockHash":"b3e12bbe1cc3ef4f88efdd47637ba60a405fdaab7e4344c489cab8c8ba08f37d","data":"Nie na tym niebie ani gwiazda nie ta - RgECPVXXhV - 872","timestamp":1650043723247,"hash":"b4f23b1f162ee89c3725d6ab776f7eaddd3e7fa2ddc664885a3e25b77eb4c6ca"}

{"index":873,"previousBlockHash":"b3e12bbe1cc3ef4f88efdd47637ba60a405fdaab7e4344c489cab8c8ba08f37d","data":"Nie na tym niebie ani gwiazda nie ta - RgECPVXXhV - 872","timestamp":1650043723247,"hash":"b4f23b1f162ee89c3725d6ab776f7eaddd3e7fa2ddc664885a3e25b77eb4c6ca"}

{"index":871,"previousBlockHash":"2d68f21f3108fefd98cded56699aae8b976831385c2b2b616e6768b79ea0f39a","data":"Nie na tym niebie ani gwiazda nie ta - amiFJIfnmN - 870","timestamp":1650043643190,"hash":"917e8399d19ebe181c9701cb95502dd2333e768a9ec59c1c85bba0e61fe69789"}

{"index":881,"previousBlockHash":"f5e0cd47a67b887fa77d063d88e1789008328c3c3b79ae6eed3081d0fd483c4a","data":"Nie na tym niebie ani gwiazda nie ta - ivakuKDhxj - 880","timestamp":1650043954700,"hash":"692eb7c93b02da83bd9f6329f17dddc2d5cf7c4c8592df52be1b5c49bdd6b816"}

{"index":880,"previousBlockHash":"3fc0e3c1480d64525d93620c73567791f2d086328e95afd620c93d09f9b5b773","data":"Nie na tym niebie ani gwiazda nie ta - aWVhRpLzKz - 879","timestamp":1650043910307,"hash":"f5e0cd47a67b887fa77d063d88e1789008328c3c3b79ae6eed3081d0fd483c4a"}

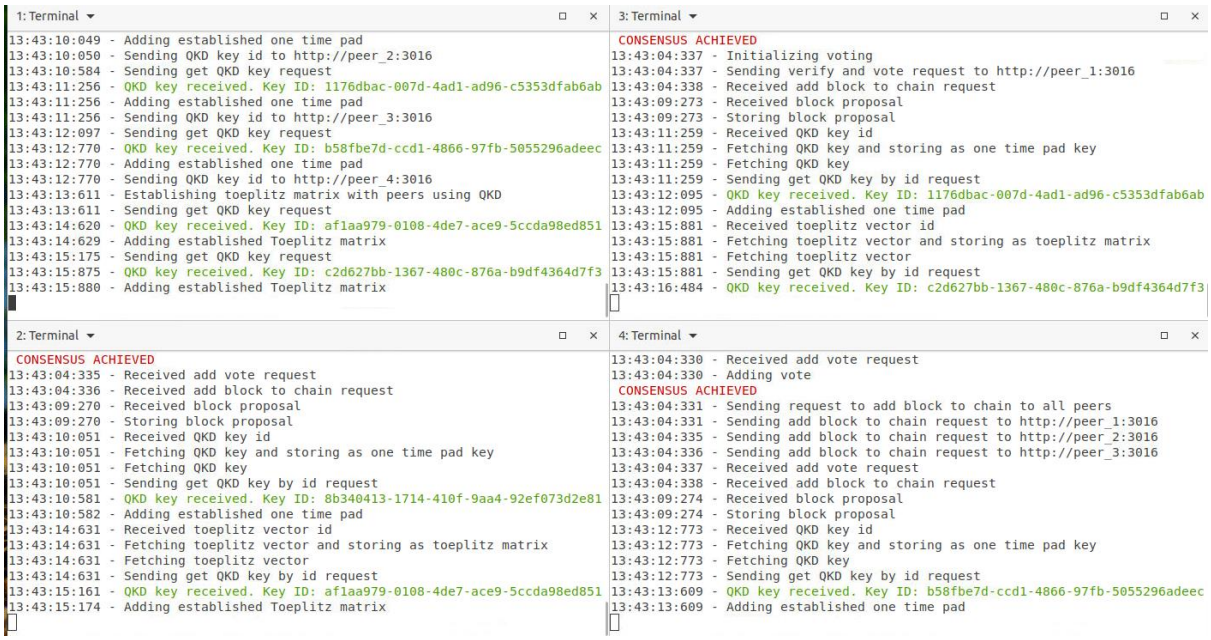
{"index":878,"previousBlockHash":"7c81eee0e901d5b9c7691120422ab7b09e2e650469f10415970e4505461a18ab","data":"Nie na tym niebie ani gwiazda nie ta - uTZMtNMsjo - 877","timestamp":1650043852755,"hash":"ba79b5cf00ee440329093156b807a8c3c8d65e4402752c18a2e933b135b6e985"}

{"index":877,"previousBlockHash":"7eb0c8f24dcba62a070b1543b6dba3e98feaf3b294db2316e6126aec7a2bf7dd","data":"Nie na tym niebie ani gwiazda nie ta - vYilBMSRzP - 876","timestamp":1650043828322,"hash":"7c81eee0e901d5b9c7691120422ab7b09e2e650469f10415970e4505461a18ab"}

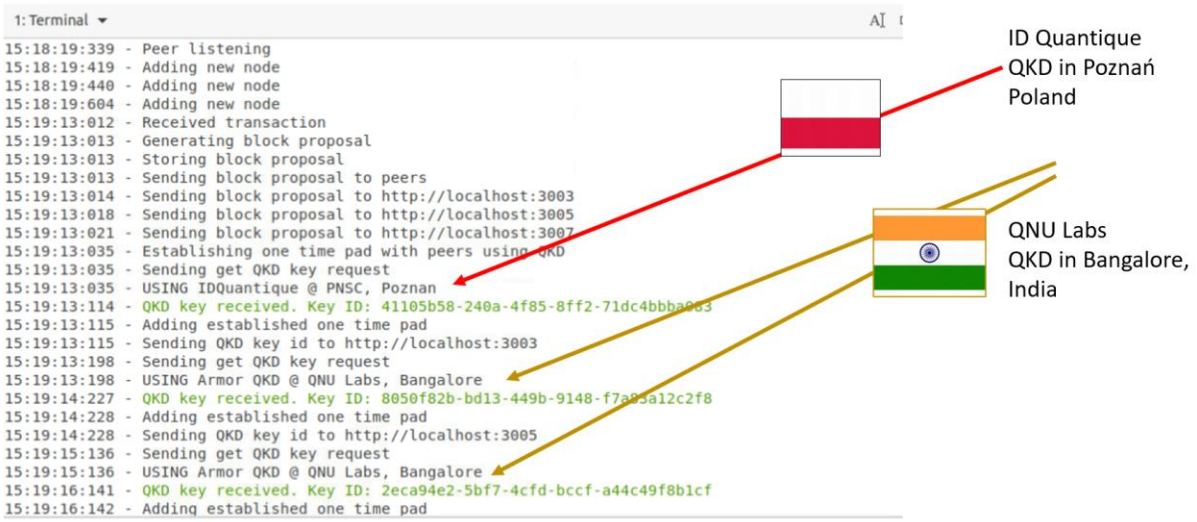
{"index":876,"previousBlockHash":"b2876bc3b0501eb59d9ec43cb9cf7649982c2d6ec782bc38da7b4b8fe715d49b","data":"Nie na tym niebie ani gwiazda nie ta - eoYtslDmnu - 875","timestamp":1650043803022,"hash":"7eb0c8f24dcba62a070b1543b6dba3e98feaf3b294db2316e6126aec7a2bf7dd"}

{"index":875,"previousBlockHash":"af9e3f1ccfe763b20c3c8c2de5a19e6ffded953d1193e551e8a16e20a34f3d78","data":"Nie na tym niebie ani gwiazda nie ta - HMCkELJXVJ - 874","timestamp":1650043779960,"hash":"b2876bc3b0501eb59d9ec43cb9cf7649982c2d6ec782bc38da7b4b8fe715d49b"}

In a typical run the terminals of the four nodes reported the following flow of operations:



We also performed several runs with the second OKD link. In the following terminal screen shot we see the use of both QKD links: from QNU Labs in Bangalore, India and from PSNC in Poznań:



The runs visualization

A short recording showing 1 minute of the QKDBase run with use of the QKD device in Bangalore, India (QNU Labs' Armos) is available here:

https://youtu.be/954QFce8s0I

The QKDBase Performance

A typical transaction time for exemplary transactions of length up to 128 bytes was in range of about 30 seconds including all the communication delays related to the remote use of the QKD links.

The following table shows typical results:

Running nodes on separated servers:

Transaction text	Start time	End time	Execution time
Litwo! Ojczyzno moja! ty jesteś jak zdrowie	15:54:08:607	15:54:40:121	31,514
Choć Sędzia z dokumentów przekonywał o tem,	16:29:57:233	16:30:28:671	31,438
Że architekt był majstrem z Wilna, nie zaś Gotem.	16:47:39:441	16:48:06:817	27,376
Dość, że Hrabia chciał zamku. Właśnie i Sędziemu	16:53:17:592	16:53:43:683	26,091
Przyszła nagle taż chętka, nie wiadomo czemu.	16:10:32:049	16:10:59:879	27,83

Running nodes under docker on a single server:

```
Start
{"transaction":"Nie na tym niebie ani gwiazda nie ta - lxQLKkpuXm - 20"}
End  20  in: 23.712284340988845
Start
{"transaction":"Nie na tym niebie ani gwiazda nie ta - PPMskbhhYo - 21"}
End  21  in: 23.620207378990017
Start
{"transaction":"Nie na tym niebie ani gwiazda nie ta - ksjkrAQhDH - 22"}
End  22  in: 24.350749919947702
Start
{"transaction":"Nie na tym niebie ani gwiazda nie ta - SzyJTSqKQW - 23"}
End  23  in: 39.933265700004995
Start
{"transaction":"Nie na tym niebie ani gwiazda nie ta - XABDokCspF - 24"}
End  24  in: 39.20870982698398
Start
{"transaction":"Nie na tym niebie ani gwiazda nie ta - yHsEdHUJYU - 25"}
End  25  in: 24.493852388986852
Start
{"transaction":"Nie na tym niebie ani gwiazda nie ta - KwEwwVkJLh - 26"}
```

Very long runs have been performed in this mode.

The average transaction time is now dependent only on the networked access to the QKD device. We reported it to be ~ 8.66 seconds (minimum was 7.24, maximum 32.4) and the average of the absolute deviations of transaction time from its mean value was 0.497408334.

Conclusions

We have reported here the implementation, deployment and testing of a minimal blockchain model equipped with the real QKD devices available commercially. Most of the tests were performed using QNU Labs Armos QKD devices, and some of them includes ID Quantique Clavis3 QKD platform.

The results obtained here proved that it is feasible and realistic to build a functioning quantum blockchain based on QKD devices and QRNG hardware for increased security of the blockchain systems.

The work reported here represents the first step of the delivery of the blockchain code of the new kind. In the next step we will apply all the experiences we have accumulated in this stop to deliver more sophisticated quantum blockchain using both QKD and QRNG capable to realize our planned use cases.

Appendix A – QNU Labs Armos QKD devices

(based on vendor materials)

Armos, QKD (Quantum Key Distribution) is a state-of-the-art appliance which provides unconditional security for your critical data by leveraging the principles of quantum physics.

Armos is used to securely generate and distribute encryption keys between two ends of a Symmetric Key Encryption system without ever sharing the actual keys on any links. The basic principle is to exploit the peculiarities of quantum mechanics by utilizing encoded photons or “Qubits" from one end (Alice) to the other end (Bob) over a single fiber core called the quantum channel.

ARMOS QNLX-210 specification

MODEL	ARMOS QNLX 210	
PHYSICAL	Dimensions	560mmx424mmx85mm (Alice) 560mmx424mmx85mm (Bob)
	Enclosure	2U 19" rack mountable
	Weight	14.5 kgs
OPERATING CONDITIONS	Operating temperature range	15 deg C - 25 deg C (ambient), 60% RH
WEAK COHERENT SOURCE	Source	DWDM DFB Laser with VOA
QKD PROTOCOL	Protocol	Distributed phase reference
QUANTUM CHANNEL	Fiber type	SMF-28
	Fiber transmission loss (typical)	0.24 db/km
	Transmission loss acceptable (typical)	12/14/16 dB
	Maximum transmission loss acceptable	24 dB
	Acceptable length of quantum channel	Up to 100km
	Secret key rates (typical)	Up to 40kb/s
	Connector type	ST
CLOCK SYNCHRONIZATION	Fiber type	SMF-28
	Connector	ST/UPC
DETECTOR	Type	Integrated, QNu proprietary
	Operation	Gated or continous, based on setup
RNG	TRNG	Integrated
INTERFACES	Key interface for external applications	Ethernet (RJ45), RESTful API
	Interface for key reconciliation	Ethernet (RJ45), encrypted
	Time synchronization channel	DWDM, C band channel 34
	Host computer interface	GUI, browser based
QKD SOFTWARE	Authentication Error correction Privacy amplification Reconciliation	QNu proprietary algorithms based on universal 2 optimized for execution on high-performance FPGA and co-processor compute engine
SECURITY	Provably secure key distribution and instantaneous intrusion detection	
MANAGEMENT & MONITORING FUNCTIONS	Diagnostics and operational status reporting along with system configuration & recovery Auto-calibration Programmable key parameters: VOA, Temperature, QBER and IP adderess	
UI	Ethernet connected to PC	View all processes with associated information
POWER	Input voltage	Auto ranging 100-240VAC @ 50/60Hz
	Connector	EAC 309, power entry
STATUS INDICATORS	Important indicators like Power, System and Channel Health, QBER, Clock Sync	
ADDITIONAL ADVANTAGES	Integrated QRNG Pre-integrated with Cisco ISR and ASR series router	

Appendix B – ID Quantique Clavis^3 QKD Platform

(based on vendor materials)

The Clavis3 Quantum Key Distribution Platform – Clavis is the Latin word for key – was developed by ID Quantique to serve as a versatile research tool for both academic and technology evaluation labs. The user can therefore experiment different parameter set-up and configurations, in both automated and manual modes.

The Clavis3 platform comprises two stations, the transmitter unit, Clavis3-A and the receiver unit, Clavis3-B. Each station consists of an optical and electronic platform controlled by an external computer which is linked to the station through an Ethernet connector.

The Clavis 3-A and Clavis 3-B units are linked by the quantum channel, used for the key transmission. In addition, a Service Channel is used for synchronisation between the two units.

It is made of a couple of optical fibre strands, connected to the units with SFP transceivers with LC/UPC connectors. The two fibre strands can be reduced to a single one with SFP transceivers supporting bidirectional transmissions.

Secure key exchange is possible over fibres with a maximum loss of 12 dB to 18 dB (typ. up to one hundred kilometres), as well as over a single core using WDM. The optical platform is well documented in scientific publications and has been extensively tested and characterized.

The Clavis3 also integrates a key management system that manage key requests and key transfers between QKD optical systems and external encryptors. Key distribution to encryptors or any key consumer is performed over secured QKD ETSI REST API or proprietary interfaces developed in partnership with major vendors. The Clavis3 receiver, Clavis3-B, can use external single-photon detectors, which can be provided either by ID Quantique, or by the end-user himself.

CLAVIS^3 specification

Model	Clavis^3
GENERAL INFORMATION	
Parameters	
Dimensions (L x W x H)	424 x 402 x 144 mm To be installed on a plate in a 19'' rack
Weight (QKDS-A)	10 kg
Weight (QKDS-B)	10 kg
Operating conditions:	
Temperature	20 to 30 °C
Max relative humidity (@ 30 °C)	80%
Non-operating conditions:	
Temperature	-10 to +60 °C
Max relative humidity (@ 40 °C)	90%
Recommended computer specifications	
Ethernet connexion	✓
RAM	4 GB
Hard Disk	A minimum of 100 MB of free space for software suite installation, additional space is needed when running the applications
Processor	Minimum Intel Core Duo

TECHNICAL SPECIFICATIONS	
Hardware	
Optical platform	✓
Proprietary digital signal generation and data acquisition electronics	✓
Random number generation	One Quantis QRNG OEM component in each station
Power supply	100-240 VAC @ 50/60 Hz
Interfaces and Inputs/Outputs	
Optical connectors (front panel):	
Quantum channel Connector type: Optical fibre type:	 FC/APC SMF-28
Service channel Two SFP modules, with LC/UPC connectors (for two-fibre configuration) Or one bidirectional SFP module (for single-fibre configuration)	
Computer interface (back panel):	Ethernet
Front Panel Indicators	
Power LED indicator (red: on)	
Quantum Link LED indicator (green: quantum channel active)	
Data LED indicator (green: raw key exchange in progress)	
Quantum Link LED indicator	
Key Exchange Characteristics	
Maximum transmission loss acceptable (typ.)	12 dB Standard
	14/18/16/18 dB Premium
Maximum length of quantum channel (typ. @ 0.24 dB/km)	50 km / 58 km / 66 km /75 km
Secret key rate (typ.)	1.4 kb/s (12 dB)
Sifting and Key Distillation	Fully automated