



BlockStamp

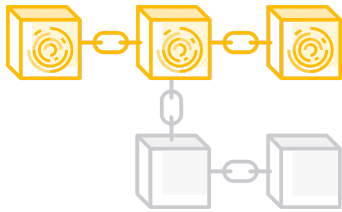
Whitepaper

Summary	3
What is BlockStamp	3
Document and signature storage	3
BlockStamp Games	4
DNS	5
Future applications	5
BlockStamp investment	5
BlockStamp Roadmap	8
Technology	9
BST Blockchain characteristics	9
Document and signature storage	10
Usage	10
Gaming center	12
Random Number Generation	12
Jackpot concept	13
Roulette betting model	13
Lottery betting model	15
Mechanisms for your own game	16
Bet verification	17
Block hash/number	17
Transaction hash	22
Wallet address	23
Mining and poles	26
Installation	26
Mining constraints	26
The approach to casino's profitability	27
DNS	27
Description	27
Usage	28
Atomic transaction	30
Workflow example	36
Desktop application & Wallet	38
Value transfer	38
Gaming	39
Data	39
Appendix 1 - bet types, syntax, and numbers	40

Summary

This documents describes the current status of the BlockStamp project, its functionalities and potential. The document will be constantly updated as the BlockStamp will be evolving. The document is maintained by the BlockStamp and its latest version will always be available on the project's [website](#).

1. What is BlockStamp



BlockStamp (BST) is a new digital currency intended to store user data in blockchain. It is based on Bitcoins peer-to-peer technology to operate with no central authority. The goal was to create a new fast and effective blockchain that could be used for trusted timestamping of documents. Currently, BlockStamp blockchain is used as the core of an online gaming platform. Future uses will cover timestamping dedicated for banking purposes, DNS and more.



BlockStamp Games is an example of an implementation of randomized transactions provided by the BlockStamp blockchain. The blockchain offers the possibility to obtain fair, untempered randomness that can be used in many applications such as e-gaming.

1.1. Document and signature storage



Sensitivity of banking transactions requires a very solid, hacking-proof, and secure system to store the data. BlockStamp blockchain is designed for this very purpose and can offer tamper-proof timestamping of banking transactions.

This can include but is not limited to:

- Money transfers
- Stock exchange prices
- Bonds
- Debentures
- Bank loans
- Notes payable
- Other debts units

This functionality is under development currently and will be made available shortly.

1.2. BlockStamp Games



The whole concept was to facilitate e-gaming in a completely new form, where there is no casino side, no house edge, and all the transactions can be verified by the players.

There were many attempts in order to create such a platform, however most of them failed during implementation, while some are still suffering from long implementation processes. The majority of those were ICOs, which after the initial fundraising, failed to produce the actual product. We took a completely different approach and decided to build our own currency as part of the project instead of raising funds for building online e-gaming sites.

Our goal was to build the whole new platform which would be able to host players and to provide randomness with 0% edges and minimal fees at the same time providing transparency of all the transactions. Hence, we decided to develop a new currency which blockchain would be used for the fairness purposes.

BlockStamp blockchain levels all issues with provably fairness, transparency, or any kind of tampering with the game results.

Current cryptocurrency casino market is still in its infancy, while its older brother does not provide any good example. The most common issues are:

- Lack of transparency of transactions
- High probability of tampering with game results (centralised infrastructure with no access from the outside of the casino)
- High fees related with betting and winning
- Extended withdrawal times and high withdrawal fees.
- Progressive jackpot
- No possibility to host one's own game
- Overall casinos' profit orientation

BlockStamp Games gets rid of all that and aims at educating people about gambling. The goal is to fight with gambling misconceptions by providing a platform that demonstrates that chance can not be controlled. The player can test different strategies and learn that the chance to win significant amounts is small even in a casino that does not take any fees. The player can run the test with demo coins that can be restored indefinitely and with crypto coins that can represent some value. In the second case the player can assess how an increase in the value of risked tokens affects the emotions during the game. When playing with demo coins the player is warned about potential risk of a gambling disorder when the balance drops below 90% of the starting balance.

1.3. DNS

The concept is based on the fact that more and more domains are being watched and blocked, against the will of the users. Peer-to-peer DNS prevents this kind of situations. This functionality is under development and will be made available shortly.

1.4. Future applications

Given that BlockStamp can be perceived as a solid timestamping device, its applications can only be limited by its users' imagination and requirements, but not by its functionality. A set of flexible procedures will be issued which will make purpose oriented timestamping possible. This document will be updated accordingly.

1.5. BlockStamp investment

BlockStamp currency is deemed to be a profitable investment option due to the following aspects:

- The gambling market was estimated at 44.16 billion in 2016 and is expected to reach USD 81.71 billion by 2022 according to Business Wire (<https://www.businesswire.com/news/home/20170928005702/en/Global-Online-Gambling-Market-2017-2022---Research>). Those numbers take under account only the official gambling market, while the illegal is deemed to be somewhere between 80 and 380 billion USD. The popularity of online gambling is growing and so will the BST price.
- According to the [CoinMarketCap](#), currencies' prices are in general higher and much more stable compared with tokens. BST is bitcoin fork and is expected to be stable and well priced. The currency is already being mined and used, hence it will not join the [Dead coin](#) list.
- The wide applicability of the BlockStamp functionality will hook it on the market and provide a solid base for cooperation with well recognised brands, e.g. on the financial market.

Although all investments have some degree of risk incorporated into them, the risk related with investing in BST has been diminished and is as low as possible on the crypto (and some fiat) investment market. Further information can be provided on a request basis, if required.

1.6. Blockstamp Roadmap

Blockstamp team has a long term plan for project development. We focus on creating useful tools to protect all types of users data in secure blockchain ecosystem. Blockstamp is a working platform which is available to use right now. Nevertheless, we have the exact plan and resources to implement new practical solutions for our community.



Cryptocurrency transfer

April 2018

Near-tamper proof blockchain technology, hosting BlockStamp currency is now combined with 1 minute block time to ensure everything happens quickly and efficiently.



BlockStamp online wallet - fully secure way to hold your BST

February 2019

BlockStamp online wallet with major security improvements such as PBKDF2 to protect your password from brute force attack is a convenient and user-friendly way to protect and manage your funds.



Blockchain based e-gaming and education platform

February 2019

Players can test different gambling strategies in a 0 house fee environment. Players can learn that chance can not be controlled. Check it out here: <https://blockstamp.games/>



Privacy network participants earn in BST

February 2019

Hprox is a BlockStamp partner project allowing users to surf the internet 100% anonymously. Members can rent out their IP addresses to privacy seekers in exchange for BST.

Check it out here: <https://hprox.com/>



Coinpaprika crypto market cap site lists the BST

February 2019

You can track all the most important data about the BST crypto coin such as its current price, exchange volume, and more.

Check it out here: <https://coinpaprika.com/coin/bst-blockstamp/>

BST mining pool

March 2019

You can join the BlockStamp mining pool via <http://bsod.pw/>.

The GPU and CPU miners' software is available at our github.



Listing on CoinMarketCap

April 2019

BST lists on the world's favorite cryptocurrency market cap ranking site.

Check it out here: <https://coinmarketcap.com/currencies/blockstamp/>

BlockStamp OpenBazaar Listing Explorer

August 2019

The BlockStamp listing explorer offers a full view of OpenBazaar's decentralized, commission-free marketplace. Sellers can boost their products' search rankings by burning BST.



Rooms at BlockStamp Games

September 2019

BlockStamp Gamers can create and administer their own casino rooms for friends.



Blockchain data sealing via online wallet

October 2019

Users can easily secure and authenticate their valuable data directly through the BlockStamp online wallet.



BlockStamp Chat

November 2019

A decentralized mobile and desktop messaging application focused on security and privacy.

2. Technology

2.1. BST Blockchain characteristics

BST is hard fork from BTC blockchain and as such it share almost all characteristics of the original blockchain. The main difference of the BST chain being the hashing, as hashes of the blocks start with the most significant bit set to one (0x80000000....) instead of the original zero (0x00000000...). In terms of transactions recording, the BST blockchain is almost identical with BTC blockchain. As such, it will be used for timestamping of transaction documents. However, to make the timestamping faster and more efficient, the transaction size was increased to 1MB and block time was reduced to 1 minute.

BST is minable currency with the number of BST increasing by an average of 500 000 BST each year. The rate of block creation is adjusted by Difficulty Adjustment Algorithm, which is similar to Bitcoin Cash - BCH. The number of BST generated per block equals 1 BST and is constant all over the time.

Chaining the blocks, like with BTC blockchain, makes it impossible to modify each block without modifying all the subsequent blocks. This is due to the hash of previous block being included in the following block. Given that a block's hash is generated based on its content, the block's content modifications would alter the hash and all subsequent blocks either. This characteristics of a blockchain is used for ensuring that no transactions, (as well as wins and loses in the gaming) would be tampered with. This is by far the most foolproof approach given the current technology.

2.2. Document and signature storage

2.2.1. Usage

To store and retrieve data from the blockchain following RPC commands should be used:

- I. `bst-cli storemessage`
which accepts data string as an argument and returns block hash, e.g.
`bst-cli storemessage "user data string"`

RETURN:

```
c04878b7bf26def16ee689863943da91f9e7dcce77250e1ca8b63
90549356006
```

- II. `bst-cli retrievemessage` which retrieves message from a particular block (accepts block hash as an argument and returns the data string contained by it, e.g.
`bst-cli retrievemessage`
`c04878b7bf26def16ee689863943da91f9e7dcce77250e1ca8b63`
`90549356006`

RETURN: "user data string"

- III. `bst-cli storesignature` which can store user's digital signature into a transaction, retrieved from a file; it accepts path to the file as an argument and returns block hash, i.e.
`bst-cli storesignature /path/to/file/myfile`

RETURN:

```
0eec311afa6e8253c984d9bd57dadedd72848065160a093dc66a7
```

76388cfda13

- IV. `bst-cli storedata` which takes any data from a file and stores it in a block; it accepts path to the file as an argument and returns block hash, i.e.

```
bst-cli storedata /path/to/file/myfile
```

RETURN:

```
ad1e1c0736f366fdf6b2e9b63048c02a0aef83fdf4e705e78f89c
3e654fa3323
```

- V. `bst-cli retrievedata` which retrieves data from a block and returns it as a string; it accepts block hash as an argument and returns the data string contained by it, e.g.:

```
bst-cli retrievedata
ad1e1c0736f366fdf6b2e9b63048c02a0aef83fdf4e705e78f89c
3e654fa3323
```

RETURN: "content of a myfile as a string"

- VI. `bst-cli retrievedata` Same as above, only with two arguments (block hash and path to a file to which the command is suppose to store the data), it returns no output, e.g.:

```
bst-cli retrievedata
ad1e1c0736f366fdf6b2e9b63048c02a0aef83fdf4e705e78f89c
3e654fa3323 /path/to/file/outfile
```

RETURN: (none)

To check if data stored in blockchain match user data, the following RPC commands should be used:

- I. `bst-cli checkdata` which retrieves data from a block, compares with the file provided as second argument and returns PASS or FAIL depending on the comparison result, e.g.

```
bst-cli checkdata
e.g.:ad1e1c0736f366fdf6b2e9b63048c02a0aef83fdf4e705e7
8f89c3e654fa3323 /path/to/file/myfile
```

RETURN: PASS

- II. `bst-cli checkmessage` which retrieves data from a block, compares with the string provided as second argument and returns PASS or FAIL depending on the comparison result, e.g.

```
bst-cli checkmessage  
c04878b7bf26def16ee689863943da91f9e7dcce77250e1ca8b63  
90549356006 "user data string"
```

RETURN: PASS

- III. `bst-cli checksignature` which retrieves signature from a block, compares with the file provided as second argument and returns PASS or FAIL depending on the comparison result, e.g.

```
bst-cli checksignature  
0eec311afa6e8253c984d9bd57dadedd72848065160a093dc66a7  
76388cfda13 /path/to/file/myfile
```

RETURN: PASS

- IV. `bst-cli listtransactions` which retrieves transactions list from a block and returns it. In addition to the list, it also returns "datasize" field informing about the size of the data stored in the blockchain, e.g.:

```
bst-cli listtransactions  
0eec311afa6e8253c984d9bd57dadedd72848065160a093dc66a7  
76388cfda13
```

2.3. Gaming center

The idea behind the gaming center was to create a platform anyone could use. The design allows us to get rid of house (and house edge). The platform is an open source project, and is available to everyone on [GitHub](#) for view and for use. Everyone who wishes so can download the code, and build one's own node, which can then be used for mining and gaming.

2.3.1. Random Number Generation

There are different approaches to the topic of random number generation, and the topic is also one of the most sensitive aspects of gaming. Everyone knows that randomness is the key to fair gaming. Old Fashioned central server online casinos were very often accused (not without a reason of course) of tampering with the random number generators or of having no randomness implemented at all. It is not very surprising, given the profit oriented approach of most, if not all, of the casinos. Having a solid random system is key to building a trustworthy online casino.

As said, there are different approaches to the RNG, with Oracles being the most popular for ETH smart contract. Most of the random number generation approaches are deemed slow and ineffective. There is no truly random system

in computing science, nevertheless some degree of randomness can be obtained.

BlochStamp casino's approach is to use the hash of the block containing bet transaction. This approach guarantees true randomness as no user can predict the hash of the block his/her transaction will be put into.

When accepted by pool, a bet is put into a block. The last 4 bytes word of block hash is divided modulo by the modulo argument (the actual bet) and the result is increased by 1. The result is then taken and compared with the bet, and the bet is a winning one if the numbers are equal.

For example, let's imagine that a user bets on roulette result to be 12. The bet is put into a transaction and the transaction is part of a freshly mined block. The hash of the block is then truncated to obtain only the last 4 bytes, decimalised, and divided modulo 36. The result of the equation is increased by 1. If the increased value is equal 12 then the bet is a winning one.

2.3.2. Jackpot concept

By design, the jackpot MAX_PAYOFF is limited to 1024×1024 BST, i.e. 1 048 576 BST. The jackpot is set to this amount from the beginning. No losing bets are required to fill it in. It is also designed so that a win of the maximum amount does not reduce future rewards. The reward coins will simply be mined as part of the block containing the winning transaction.

The design was possible due to probability of a win of such amount and some constraints on the actual betting. There are two rules around how transactions are chosen for blocks. The sum of input amounts (all bets) included in a given block will not be greater than 0.9 of a current block subsidy (i.e. 50BST at the moment), and the sum of payoffs/rewards for for a block will not be greater than the sum of the inputs and the subsidy. This prevents huge bets with lowest modulo (i.e. 2) that can cause the system to collapse.

2.3.3. Roulette betting model



The BlockStamp roulette was designed to have 0% house edge, as there is no House betting with the players. Understandably, there is no 0 or 00 on the roulette. The numbers available for betting are 1-36.

This adds up to the fairness of the gaming model, as the win on a single number is true 35:1.

The transaction to make a bet and play is `makebet`.

In order to make a bet, a user needs to choose a number or numbers and the sum of a bet. There are 13 types of bets:

- Straight (single number) with 35:1 ratio
- Split (two numbers) with 17:1 ratio
- Street (three numbers) with 11:1 ratio
- Corner (four numbers) with 9:1 ratio
- Line (six numbers) with 6:1 ratio
- Column (twelve numbers) with 3:1 ratio
- Dozen (twelve numbers) with 3:1 ratio
- Low (eighteen numbers) with 1:1 ratio
- High (eighteen numbers) with 1:1 ratio
- Even (eighteen numbers) with 1:1 ratio
- Odd (eighteen numbers) with 1:1 ratio
- Red (eighteen numbers) with 1:1 ratio
- Black (eighteen numbers) with 1:1 ratio

Details of the actual bet types are available in [Appendix 1](#) of this document.

There can be more bets in one transaction than one. For example:

```
makebet straight_3@0.1+street_5@0.05+high@0.7
```

Which is a compilation of three bets:

- No 3 with 0.1BST bet amount
- No 7,8,9 with 0.05BST bet amount
- High numbers with 0.7BST bet amount

The details of the bet will be stored in `op_return` field in the transaction. In this case, it would look as follows:

```
Op_return:
```

```
00000024_straight_3@100000+street_5@500000+high@7000000
```

To make a bet a user should have total transaction input amount equal to sum of bet amounts plus fee. In this case: $0.1+0.05+0.7+0.000 \times \text{BST}$. The fee is computed automatically inside `makebet` RPC and is the mirror of current BTC fee algorithm. This fee is a mining fee only, there are no other fees related with betting or winning.

Calling `makebet` RPC returns transaction ids.

Making bets has following limitation, by design:

- Maximum modulo argument (reward ratio) is $1024 \times 1024 \times 1024$
- Maximum payoff is 1024×1024 BST
- Minimum bet is 0.00000001BST

Winnings are redeemed automatically. A block following the block containing the winning transaction, will contain a payout transaction. The winning will be transferred to the wallet used in the `makebet` transaction.

Before block 224940, the implementation included a `getbet` transaction that was designed to verify if a transaction was a winning or a lose and redeem the reward to a wallet provided as the transaction's parameter. Calling `makebet` RPC returns transaction id. This transaction ID was used then as an input to `getbet` RPC, which accepted two arguments - transaction ID and an address where the reward should be sent. For example:

```
getbet
123d6c76257605431b644b43472ee3666c4f27cc665ec8fc48c2551a88
f9906e 36TARZ3BhxUYaJcZ2EF5FCT32RnQPHSxYB
```

`Getbet` returned a transaction ID for a successfully redeemed wins or errors in case of loses. To properly redeem the win amount, Pay-To-Public-Key-Hash transaction keys must have matched.

Please note that the transaction is obsolete and it no longer exists. It was replaced by the automated redeem system introduced by the hardfork in block 224940. This applies also to the lottery and the mechanism for user's own game described below.

2.3.4. Lottery betting model



Lottery also utilises `makebet` transaction with the same syntax. What makes lottery different is that the only bet type available is straight bet. Each `makebet` can define one number or more as separate bets from 1 up to 2^{30} with modulo set to maximum 2^{30} . `Makebet` RPC can accept more than one straight bet, up to 10 bets. The modulo

operator defined is tantamount to the number of options we draw from. For example:

```
makebet 2@0.1+3@0.05 72
```

would put into a transaction a bet for No 2 and No 3 with different incentives (0.1BST and 0.05BST accordingly). Having modulo set to 72 means that the 4 bytes of hash of the block in which the transaction will be put, shall be divided modulo 72 (as if we drawn from 72 numbers from 1 to 72). If the result of the hash modulo 72 operation plus one is equal to any of the numbers user betten on, the transaction will be a winning one. All bets in one `makebet` RPC are subject to the same rules and same modulo defined in the `makebet`.

In this case reward ratio is equal modulo argument and the op_return field contains only reward ratio and consecutive straight bets, excluding bet name prefix (e.g. "straight"). For example, in case of the above bet op_return bet description would look as follows:

```
op_return: 00000048_2@10000000+3@100000000
```

If a bet is a losing one, all coins in the bet are sent to the op_return and 'burnt'. Otherwise, the rewards are sent automatically to the bet wallet in the following block.

2.3.5. Mechanisms for your own game



The BlockStamp platform can be treated as a hosting platform for user's own game. Makebet transaction is available for this option and works as it does in the lottery game. User's own game rules determine the bets and modulo, where modulo is the reward ratio. For example, if dice is to be implemented, the bets should accept numbers from

1 to 6 and modulo should be set to 6. If two dices are to be rolled together, numbers would be from 2 to 12 and modulo should be set to 11. To avoid 0, the result of hash modulo operation is plused one and only after that it is compared with the number user betted on. So the makebet could look as follows:

```
makebet 2@0.1+11@0.05 11
```

This means that a user bets on double dice rolling 2 (1+1) with 0.1BST and 11 (5+6) with 0.05BST. Modulo is set to 11, and the block of the hash which would contain the bet would be divided by 11 and then added 1.

As for the actual transaction syntax, it looks as follows. Makebet transaction is signaled by 30'th bit in the transaction version field. Value 1 at this position (0x40000000) sets transaction type to makebet transaction. Bets outputs occupies outputs beginning from index 0. Next to bets output, there is an op_return output containing the description of the bets included in the transaction. The op_return string begins with hexadecimal number defining a modulo argument (e.g. 0x24 for roulette, 0x48 for our lottery example, and 0xB for our double dice example), followed with user bets. The modulo argument is separated by underscore, bets are separated by plus. The last output field is reserved for change purpose, where change is transaction input minus bets amount minus fee, i.e.:


```
prev_txs_outs_amount-Bet1_amount-Bet2_amount-Bet3_amount-.  
..-fee
```

In case then the transaction input is 50BST (which is the standard block reward), and 3 bets for 0.1BST, 0.05BST, and 0.7BST, and fee=0.0004 BST the change field would contain 49.1496 BST ($50 - 0.1 - 0.05 - 0.7 - 0.0004 = 49.1496$ BST).

If a bet is a losing one, all coins in the bet are sent to the op_return and 'burnt'. Otherwise, the rewards are transferred automatically, as in roulette and lottery

2.3.6. Bet verification

What one needs to know before doing the calculations is that block hash in which a transaction is put is our random number which is used for the actual draw (or roulette spin). Details are available on our [github wiki](#) and in this very document. It contains all the details on how it is ensured that there is no way anyone can predict the random numbers before making a bet.

The procedure is pretty simple as all the necessary details can be found via [BST explorer](#).

Data needed for the verification is either of the following:

- 1) Block hash/height
- 2) Transaction hash (your bet)
- 3) Wallet address

3.2.6.1. Block hash/number

Block hash or block height can be put into the explorer search which can be found on the top left corner. A result of the search is a block with all the details. For example:

<https://explorer.blockstamp.info/block/8000000000023e663e322af14e8febecfa5047bf28158ac3a13e4a06db69b212>

1 BST / 0.00546360 ETH
-4.21%
1 BST / 0.00009267 BTC
-0.72%
Buy at
PicoStocks.com
Follow us at:
M
Q
B
T
+

BlockStamp
explorer

BlockStamp block: 56237
8000000000023e663e322af14e8febefa5047bf28158ac3a13e4a06db69b212
Previous block
Next block

- Time2018-09-24 20:44:05
- Total transacted221.50705208 BST
- Total fees0.00240905 BST

Hash	8000000000023e663e322af14e8febefa5047bf28158ac3a13e4a06db69b212	Time	2018-09-24 20:44:05
Height	56237	Value	221.50705208
Size	5309	Version	536870912

Advanced details

Transactions

Transaction
56e7fd1519680f500ee36ac2f9a25795230a2f197da90fcbc02e7b1fe9bbad5c
CONFIRMED

12.27473596 BST
366bz4ReCUdvg6QxyKyMDhkDZUh81LPuMW/

2.75463612 BST
3N49zu97ePjpeQCxZTjyBKS1nKx4RYE1G

9.52000000 BST
3JHfSjcBB4rZGUgXWkBV1xdL4NVaVAm8q

Value transacted:12.27463612 BST

Transaction
2f968532ae05cd3ff9b54d1e825fe88af411ca1067ff87173ae9026f994b22b
CONFIRMED

What can be found there are all block details, like block size and time, fees, amount transacted. Also the list of all transactions that were part of that block is available. The required bet transaction should be there too. It will contain all the details like wallet address, type of game (roulette, lottery), the bet details like bet type for roulette (e.g. a single number, corner, line) and a number plus pool/modulo for lottery. Pool/modulo is the number of options used to draw from, e.g. for a draw of 1 number out of 6 - 6 will be the modulo.

Transaction
afdf7bc44629413d5cef0550ade4fa02834c1222fd650cb45a6e1cc2d080a1594
CONFIRMED

3.08000000 BST
3jfmkFomBi37FHTPZVME64eNEWH4i84hBQ
0.08789660 BST
3NHdRQm54KE61kLmxUKc6nDHLG7sBLCwB4

3.02764820 BST
3NHdRQm54KE61kLmxUKc6nDHLG7sBLCwB4
0.00000000 BST
Address unparsed
0.14000000 BST
37qxd5e5gXdDVv47t3JdyteB7PL8H9Q4
BLOCKSTAMP GAMES 0.14000000 BST
Lottery: 185 x1374 WON 192.36000000 BST

Value transacted:
3.16764820 BST

For modulo operation, we recommend this calculator:

<https://www.miniwebtool.com/modulo-calculator> The result of the operation is **184**.

- All results are increased by one in order to avoid zeros. So the actual result of the lottery game is **185**.
- Now you need to compare it with the number you betted on. In our example, they are equal.

Transaction
afdf7bc44629413d5cef0550ade4fa02834c1222fd650cb45a6e1cc2d080a1594
CONFIRMED

3.08000000 BST
3jfmkFomBi37FHTPZVME64eNEWH4i84hBQ
0.08789660 BST
3NHdRQm54KE61kLmxUKc6nDHLG7sBLCwB4

3.02764820 BST
3NHdRQm54KE61kLmxUKc6nDHLG7sBLCwB4
0.00000000 BST
Address unparsed
0.14000000 BST
37qxd5e5gXdDVv47t3JdyteB7PL8H9Q4
BLOCKSTAMP GAMES 0.14000000 BST
Lottery: 185 x1374 WON 192.36000000 BST

Value transacted:
3.16764820 BST

A roulette example would only differ at the point where the modulo operation comes in. To picture that, the following winning transaction and the block containing will be used:

<https://explorer.blockstamp.info/block/800000000000effc8960ec3254ed66668a9a193c3b7d2ee859538bfc311d73245>

<https://explorer.blockstamp.info/tx/5080f81f77a8bf01834696de71a8d05f46856c6db48b324a6b7dce89e4bbd043>

Details

Transaction
5080f81f77a8bf01834696de71a8d05f46856c6db48b324a6b7dce89e4bbd043

CONFIRMED

0.41100000 BST
3Q2QnQLXeR5hSuyRjrSBPIXLfrjcnFA9W

0.40763140 BST
32tAf6b9RDBLtpfDU8mSDuFo2a16VK7AGn
0.00000000 BST
Address unparsed
0.00329000 BST
36rAbZ54vW3wedgHb5A9XDkkGBprCFZMye

BLOCKSTAMP GAMES

0.00329000 BST
Roulette: CORNER (7, 8, 10, 11) x9

WON 0.02961000 BST

Value transacted:
0.41092140 BST

The calculations are as follows:

- The last 8 characters (**11d73245**)



BlockStamp block: **89149**

800000000000effc8960ec3254ed66668a9a193c3b7d2ee859538bfc3

11d73245

7 17:15:48	<ul style="list-style-type: none"> Total transacted 60.99451623 BST	<ul style="list-style-type: none"> Total 1 0.004
<div>Advanced details</div>		

decimalised would make **299315781**

- **299315781** modulo **36** (the default roulette modulo) is **9**
- **9** plus 1 is **10** so the actual result of the roulette round is **10**.
- Now, comparing it with the bet which was CORNER (7, 8, 10, 11) we can see that **10** is one of the betted numbers. Hence, the transaction is a win

Alternatively, having block hash, one can check the following block in order to see all reward transaction details. For example, block 229353

(<https://explorer.blockstamp.info/block/8000000000001a2b1cd5ba03ef55c31c50f2cb5e62c0353234bf421297cdd859>) contains:

Transaction
757f789342bcc3a997a2abe7647be11f617839fc90815a292576fa9d6e9e1737
CONFIRMED

69.91397242 BST
3C4HTVXek4jEMvH29kF6rdCdDpcVQuyGXB

69.73994417 BST
34iFjHpKXU8kXajVpZjdceF8wU4mDAyr4C
0.17400000 BST
Address unparsed

BLOCKSTAMP
GAMES

0.17400000 BST
Roulette:
x2
WON 0.34800000 BST

Value transacted:
69.91394417 BST

And the following block 229354

(<https://explorer.blockstamp.info/block/800000000000020398b7a3b0d948662d4f520b821951c53f0fd31a2d8fba6906>) contains the following transaction:

Transaction
563c5fafc0bf23c9b7104b2e148a1285a3dee35df4c3ea5a782ca0d2e97671d8
CONFIRMED

0.17400000 BST
Address unparsed
Show winning transaction >

0.34800000 BST
129czu6Ag19AAj8wznpjHKjjkMjaPC7tnuj

BLOCKSTAMP
GAMES

0.17400000 BST
Roulette:
x2
WON 0.34800000 BST

0.48300000 BST
Address unparsed
Show winning transaction >

1.44900000 BST
12N2vpEDJCEpC2ictPgM6eLF5RDAB11xn3

BLOCKSTAMP
GAMES


0.48300000 BST
Roulette:
2nd COLUMN x3
WON 1.44900000 BST

Value transacted:
1.79700000 BST

3.2.6.2. Transaction hash

If only transaction hash is available, the verification instructions are as follows:

- Go to the [explorer](#) and search for the transaction hash. You will be shown the transaction details.
- Find the block details is to click on the advanced properties of a transaction and check which block was it part of. The link there will open the block details for you.



BlockStamp block: **89149**
 80000000000effc8960ec3254ed66668a9a193c3b7d2ee859538bfc311d73245

Time
 2018-10-17 17:15:48

Total transacted
 60.99451623 BST

Total fees
 0.00440958 BST

Hash	80000000000effc8960ec3254ed66668a9a193c3b7d2ee859538bfc311d73245	Time	2018-10-17 17:15:48
Height	89149	Value	60.99451623
Size	17401	Version	536870912

Advanced details ^

- Having the block’s hash, you can start the calculations. Just follow the above instruction and you will verify your bet.

3.2.6.3. Wallet address

If all info available is the wallet address, the verification is also possible. The above lottery win example will be used to show that. In the lottery winning transaction, the wallet involved in the bet was <https://explorer.blockstamp.info/address/37qxd5e5gXdDVv47t3JDyjteB7PL8H9Q4>

The link will show all transactions that went to and from that wallet.

The screenshot displays the BlockStamp explorer website. At the top, there's a navigation bar with the BlockStamp logo and 'explorer' text. Below this, a yellow banner shows the 'BlockStamp address:' as `37qqxd5e5gXdDVv47t3JDyte87PL8H9Q4`. The main content area is divided into two columns: 'Received' and 'Sent', both showing a balance of 0.14000000 BST. A 'Balance' section shows 0.00000000 BST. To the right of these sections is a QR code. Below the balance sections is a 'Transactions' section. It lists two transactions. The first transaction, with ID `31af26fc7fc6147df15a8cd461cf2d73cbe3abf0c98c3800ea7db3d8889c267b`, is marked 'CONFIRMED'. It shows an 'OUT' transaction of 0.14000000 BST to an unparsed address, and a 'WINNING TRANSACTION' of 0.14000000 BST from 'BLOCKSTAMP GAMES' Lottery, with a multiplier of x1374 and a win amount of 192.35985300 BST. The second transaction, with ID `afd7bc44629413d5cef0550ade4fa02834c1222fd650cb45a6e1cc2d080a1594`, is also 'CONFIRMED'. It shows an 'IN' transaction of 3.08000000 BST from an unparsed address, and a 'WINNING TRANSACTION' of 0.14000000 BST from 'BLOCKSTAMP GAMES' Lottery, with a multiplier of x1374 and a win amount of 192.35985300 BST. The value transacted for the second transaction is 3.16764820 BST.

To find the transaction details, one needs to open the bet transaction by following the link under the winning transaction or by clicking on the link below TXID:

Transactions

Transaction
31af26fc7fc6147df15a8cd461cf2d73cbe3abf0c98c3800ea7db3d8889c267b

CONFIRMED

0.14000000 BST
Address unparsed
[Show winning transaction >](#)

OUT

192.35985300 BST
3NHdRQm54KE61kLmxUKc6nDHLG7sBLCwB4

 0.14000000 BST
Lottery: 185 x1374 WON 192.36000000 BST

Value transacted:
192.35985300 BST

Transaction
afd7bc44629413d5cef0550ade4fa02834c1222fd650cb45a6e1cc2d080a1594

CONFIRMED

3.08000000 BST
3JfmkFomBi37FHTPZVME64eNEWH4i84hBQ

IN

3.02764820 BST
3NHdRQm54KE61kLmxUKc6nDHLG7sBLCwB4

0.08789660 BST
3NHdRQm54KE61kLmxUKc6nDHLG7sBLCwB4

0.00000000 BST
Address unparsed

0.14000000 BST
37qqxd5e5gXdDvV47t3JDyiteB7PL8H9Q4

 0.14000000 BST
Lottery: 185 x1374 WON 192.36000000 BST

Value transacted:
3.16764820 BST

The transaction will show block height in its advanced properties (please refer to the [transaction hash](#) example). Having the hash and bet details, one needs to follow the calculations instruction to get the result.

Similarly for a lost bet:

<https://explorer.blockstamp.info/address/37tePnymtSM33egBCpveSqcAsNXJGPYdRe>, the TXID link needs to be followed to obtain transaction data:

Transaction

31bea2793d4d803acc078449ebceb2ec6cbc705a3cd322d47e816a9a444a90f

CONFIRMED

0.32600000 BST

37tePnymtSM33egBCpveSqcAsNXJGPYdRe

OUT

0.32573611 BST

32tAf6b9RDBLtpfDU8mSDuFo2a16VK7AGn

0.00000000 BST

Address unparsed

0.00018500 BST

38eUiDc1SkakX34Tvxv1Q3XX9JYkG3sKk5

BLOCKSTAMP GAMES

0.00018500 BST

Lottery: 101600091 x695731076 LOST

Value transacted: 0.32592111 BST

After that, the calculation instructions needs to be followed to verify the bet.

2.3.7. Mining and poles

2.3.7.1. Installation

BlockStamp project's code is available on our [GitHub project site](#). To start using BlockStamp one should build the project and run a node with option -txindex to enable blockchain transaction queries. The node default configuration lets you connect to our working nodes. When connected to the existing node, the new node should start downloading blocks. After downloading process is completed, one can start working with BlockStamp.

Default settings for BST are following:

- working directory: ~/.bst
- config file: bst.conf
- RPC port: 8445
- peer-to-peer network port: 8446
- executable names: bst*

2.3.7.2. Mining constraints

Coinbase transaction in each of the blocks contains 50 BST, that are unspendable for the following 1000 blocks.

A mining policy exists preventing blocks from being mined, if the rules are not followed. The rules are:

- The sum of input amounts for winning makebet transactions included in a given block is not greater than 0.9 of a current block subsidy

(currently 50BST) and the sum of payoffs for for this block is not greater than the sum of inputs and the subsidy.

- The sum of payoffs for winning `makebet` transactions included in a given block is not greater than maximum reward (1024*1024 BST, i.e. 1 048 576 BST).

Miners can select the transactions they prefer to put into a block, based on the pools rules (e.g. fee, amount of a bet, potential reward).

2.3.8. The approach to casino's profitability

The project is maintained and developed by BlockStamp which is a non-profit organisation. Hence, the BlockStamp Games is not designed to be a profitable project. There are no hidden costs, fees, or edges as BlockStamp is not a side in the gaming.

The project is open source and can be used by anyone who wishes so. The currency as well as the platform will be improved as much as possible by BlockStamp and its cooperators.

2.4. DNS

2.4.1. Description

DNS service is build on the BlockStamp blockchain base. Details of domain names are stored in blocks. Transactions are used to:

- Create name
- Modify name details (change owner info, i.e. the wallet, modify expiration date which is 36000 blocks by default which is a year time)
- Search for a name (in order to verify if it is already taken or to check details of a specific name, like owner, expiry date, availability for sale)
- Show name history (this can be used to show historical data, like ownership, expiry, etc). It lists all entries related with the name provided as parameter.
- List all names that match a pattern (regex is used to define the pattern). No pattern means that 500 (or any other defined number) names will be listed starting from the defined block.
- List all unconfirmed operations related with DNS. This pan also be parameterized and a list of all operations related with a specific name can be obtained.

2.4.2. Usage

- I. `bst-cli name_show` which looks up the current data for the given name and returns name, data, wallet information, update date, and expiry information if possible e.g.

```
bst-cli name_show mydomain.com
```

```
RETURN: {
  "name": "mydomain.com",
  "name_encoding": "ascii",
  "value": "new-test-value",
  "value_encoding": "ascii",
  "txid":
"d839cd97b2339fa3f204b5d38517480aa89d9b1cd5a6c6e4e6bf
09b611558967",
  "vout": 1,
  "address": "1FLF43YBB1JaKqLQFpoBkjsg8HTnWfFBwe",
  "height": 116181,
  "expires_in": 525596,
  "expired": false,
  "ismine": true
}
```

- II. `bst-cli name_history` which looks up the current and all past data for the given name and returns name, wallet information, update date, and expiry information for all the entries found for that specific name, e.g.

```
bst-cli name_history mydomain.com
```

- III. `bst-cli name_scan` which all names found in the database. It also accepts parameters (name to start from and number of blocks to show, which by default is 500) e.g.

```
bst-cli name_scan mydomain.com
```

```
{
  "name": "mydomain.com",
  "name_encoding": "ascii",
  "value": "for sale at 1BST",
  "value_encoding": "ascii",
  "txid":
"d839cd97b2339fa3f204b5d38517480aa89d9b1cd5a6c6e4e6bf
09b611558967",
  "vout": 1,
```

```

        "address":
        "1FLF43YBB1JaKqLQFpoBkjs8HTnWfFBwe",
        "height": 116181,
        "expires_in": 525595,
        "expired": false,
        "ismine": true
    }

```

IV. `bst-cli name_filter` which shows all names found in the database that match the regular expression provided as parameter. It also accepts the following optional parameters:

- A. Maximal age (No of blocks to check back to)
- B. Height of the block to check from
- C. No of results to throw
- D. Stats - whether statistics is to be thrown instead of domain names

Usage example:

```
bst-cli name_filter [a-z]8[1-9]1
```

```

RETURN: {
    "name": "mydomain1.com",
    "name_encoding": "ascii",
    "value": "not for sale",
    "value_encoding": "ascii",
    "txid":
    "d839cd97b2339fa3f204b5d38517480aa89d9b1cd5a6c6e4e6bf
    09b611558967",
    "vout": 1,
    "address":
    "1FLF43YBB1JaKqLQFpoBkjs8HTnWfFBwe",
    "height": 116182,
    "expires_in": 525595,
    "expired": false,
    "ismine": true
}
{
    "name": "mydomain2.com",
    "name_encoding": "ascii",
    "value": "not for sale",
    "value_encoding": "ascii",
    "txid":
    "4b5d38517480aa89d9b1cd5a6c6e4e6bf09b611558967d839cd9
    7b2339fa3f20",
    "vout": 1,

```

```

        "address":
        "1JaKqLQFpoFLF43YBB1Bkjs8HTnWfFBwe",
        "height": 108181,
        "expires_in": 525595,
        "expired": false,
        "ismine": true
    }

```

- V. `bst-cli name_pending` which shows all name related unconfirmed operations in the pool. It can be limited to a specific name only, if parametrised (name of the domain should be provided as parameter)

Usage example:

```
bst-cli name_pending
```

RETURN:

```

{
    "name": "mydomain.com",
    "name_encoding": "ascii",
    "value": "not for sale",
    "value_encoding": "ascii",
    "txid":
    "d7afa2230eec0a22fc9abfe0185fd507ce57a209d3eef6340641
    c1e022a50cc0",
    "vout": 0,
    "address":
    "1PRqgLKvSwhqcoQgGt3CAD34M4ndMNEt2Z",
    "ismine": true,
    "op": "name_firstupdate"
}

```

- VI. `bst-cli name_checkdb` which verifies the name db consistency.

2.4.3. Atomic transaction

The idea of atomic transaction is to enable trustless transactions without the need of escrow. Atomic transaction is a transaction that ensures that either both elements of it happen simultaneously or none does.

In case of DNS, this means that atomic transaction covers both aspects - domain name transfer and payment for the transfer, and ensures that they are treated as one logical operation.

The below procedures are used to prepare an atomic transaction which will cover the whole sales/registration process.

- I. `bst-cli namerawtransaction` which adds the name transaction to the new raw transaction. To create a new transaction, use `Createrawtransaction` procedure. It requires the following 3 parameters:
- The transaction hex string
 - The vout of the desired name output
 - Json object for name operation, where `name_op` can be any of the following:
 1. `Name_new` (to create a new name entry in the database)
 2. `Name_firstupdate` (to update a newly created name, and add ownership and expiry, or description/data)
 3. `Name_update` (to update a name that already exists and is in use)

Usage example:

```
bst-cli namerawtransaction
8a592b73a8beb79cd0d73fc9bfb7c55de825ca981b5e35654cd22
28703694357582528d8ab01fffffffff03ca9a3b1976a9143c4b7d
4b93bc6194087bbbc422fd6cd1a40f820e88ac60ed3877161976a
914ab96be7f12bd38dd25b62be02b88ac40420f3f5310642f6d79
2d636f6f6c2d646f6d61696e117468616e6b732066726f6d20536
16c6c796d7576a914ad4a0929e9c7c95910534b93ec0727058a27
f2b988ac7102aabfac0ee6277ca45bccca1f98453101143993970
19ea12c8e243eb8839fed12fffffffff 1 {"op":"name_new",
"name":"mydomain.com"}
```

RETURN:

```
25b62be02b88ac40420f3f5310642f6d792d636f6f6c2d646f6d6
1696e117468616eb79cd0d73fc9bfb7c55de825ca981b5e35654c
d228703694357582528d8ab01ffffe6b732066726f6d2053616c
6c796d7576a914ad77ca45bccca1f9845310114399397019ea12c
8e243eb8839fed12fff4a09298fffffa592b73a8bfffff03ca9a3b
1976a9143c4b7d4b93bc6194087bbbc422fd6cd1a40f820e88ac6
0ed3877161976a914ab96be7f12bd38dde9c7c95910534b93ec07
27058a27f2b988ac7102aabfac0ee62
```

- II. `bst-cli Createrawtransaction` which creates a transaction spending the given inputs (array containing txid and vout) and creating new outputs. Outputs can be an array of addresses or data. It returns hex-encoded raw transaction.

Note that the transaction's inputs are not signed, and it is not stored in the wallet or transmitted to the network. In order to sign a raw transaction, please use `Signrawtransactionwithkey`, in order to send the transaction to the network, use `Sendrawtransaction`.

Usage example:

```
bst-cli Createrawtransaction [{"txid":
"460ff04e500afd4c6164d70f0421b44e8d4979dfc11f33bf7f7c
f3ad45333bb5", "vout":0},
{"34FYZErJ3CzdtQTjzxGyTPVrYBZAzeKKw4":1.00,
"3G8upiyMDTaA6YYiHUoUyikuBewoLGsuGt":3.20}]
```

RETURN:

```
c9bfb7c55de825be7f12bd38dd25b62be02b88ac40420f3f53106
42f6d792d636f6f6c2d646f6d61696e117468616e6b732066726f
6d2053616c6c796d7576a914ad4a0929e9c7c95910534b93ec072
7058a27f2b988ac7102aabfac0ee6277ca45bccca1f9845310114
399397019ea12c8e243eb8839fed12fffffffff8a592b73a8beb79
cd0d73fca981b5e35654cd2228703694357582528d8ab01ffffff
ff03ca9a3b1976a9143c4b7d4b93bc6194087bbbc422fd6cd1a40
f820e88ac60ed3877161976a914ab96
```

- III. `bst-cli Decoderawtransaction` which decodes hex-encoded transaction, e.g. the product of `Createrawtransaction`. It returns a JSON object representing the serialized, hex-encoded transaction.

Usage example:

```
bst-cli Decoderawtransaction
c9bfb7c55de825be7f12bd38dd25b62be02b88ac40420f3f53106
42f6d792d636f6f6c2d646f6d61696e117468616e6b732066726f
6d2053616c6c796d7576a914ad4a0929e9c7c95910534b93ec072
7058a27f2b988ac7102aabfac0ee6277ca45bccca1f9845310114
399397019ea12c8e243eb8839fed12fffffffff8a592b73a8beb79
cd0d73fca981b5e35654cd2228703694357582528d8ab01ffffff
ff03ca9a3b1976a9143c4b7d4b93bc6194087bbbc422fd6cd1a40
f820e88ac60ed3877161976a914ab96
```

RETURN:

```
{
  "txid":
  "ad22e94a210df8ca340c208cdb09d1e42775a03bb32b4db113e
  76fb374b6495",
  "hash":
  "e5ec499bd7cec80a502252c25fca1df8de6876682b149e7ee4b6
  da447c0833b7",
  "version": 2,
  "size": 388,
  "vsize": 225,
  "weight": 898,
  "locktime": 37932,
  "vin": [
    {
      "txid":
```



```

"139c224c6ea460c7fb081c6e404a22e23d8afe666d742b001021
9f53ccbace18",
  "vout": 0,
  "scriptSig": {
    "asm":
"0014b0b8ce35b26784d89431e15cecf025eed121d07c",
    "hex":
"160014b0b8ce35b26784d89431e15cecf025eed121d07c"
  },
  "txinwitness": [

"3045022100b9a53ac49bd2bef1c0fd07f34588d536aee166f5bf
c047f75f17bc687699b5f4022064c6471d58a7705196bfd9272b0
4ca1b9ea05ff51cafffa85d252c308680fd7201",

"03d14ee6f062e0cfcb7c5a9502a357ace8f7e2c07e96ec28d154
6633c673a8970e"
  ],
  "Value": 1
  "sequence": 4294967293
},
{
  "txid":
"3ff20f34d7790e4ba90f3914df511a86fc47ccbbb36ad154a612
0931f57a901a",
  "vout": 0,
  "scriptSig": {
    "asm":
"0014b0b8ce35b26784d89431e15cecf025eed121d07c",
    "hex":
"160014b0b8ce35b26784d89431e15cecf025eed121d07c"
  },
  "txinwitness": [

"3045022100aeb015980ee237b8322af2e7f74b37c96b163fbffa
7099e8947a285765c596e302206cf805e54d5d5c3d89c3602e150
b2319d1749cbe982c99caf35bdda46babbd8701",

"03d14ee6f062e0cfcb7c5a9502a357ace8f7e2c07e96ec28d154
6633c673a8970e"
  ],
  "sequence": 4294967293
}

```

```

],
"vout": [
  {
    "value": 1,
    "n": 0,
    "scriptPubKey": {
      "asm": "OP_HASH160
4244f06d6f6f29a69f259235bba944caca4edc11 OP_EQUAL",
      "hex":
"a9144244f06d6f6f29a69f259235bba944caca4edc1187",
      "reqSigs": 1,
      "type": "scripthash",
      "addresses": [
        "37jR7BWh6DhofUftXEBCMHD53FGwG4ckJK"
      ]
    }
  },
  {
    "value" : 0.01000000,
    "n" : 2,
    "scriptPubKey" : {
      "nameOp" : {
        "op" : "name_update",
        "name" : "mydomain.com",
        "value" : "as agreed"
      },
      "asm" : "NAME_OPERATION OP_DUP OP_HASH160
ad4a0929e9c7c95910534b93ec0727058a27f2b9 OP_EQUAL",
      "hex" :
"5310642f6d792d636f6f6c2d646f6d61696e117468616e6b7320
66726f6d2053616c6c796d7576a914ad4a0929e9c7c95910534b9
3ec0727058a27f2b988ac",
      "reqSigs" : 1,
      "type" : "pubkeyhash",
      "addresses" : [

"0014b0b8ce35b26784d89431e15cecf025eed121d07c"
      ]
    }
  }
]
}

```

- IV. `bst-cli Combinerawtransaction` which combines multiple partially signed transactions into one transaction. The combined transaction may be another partially signed transaction or a fully signed transaction. It accepts an array of hex-encoded transactions (strings) and returns a hex-encoded **signed** transaction.

Usage example:

```
bst-cli Combinerawtransaction
e9c7c95910534b93ec0727058a27f2b988ac7102aabfac0ee6277
ca45bccca1f9845310114399397019ea12c8e243eb8839fed12ff
ffffff8a592b73a8beb79cd0d73fc9bfb7c55de825ca981b5e356
54cd2228703694357582528d8ab01fffffffff03ca9a3b1976a914
3c4b7d4b93bc6194087bbbc422fd6cd1a40f820e88ac60ed38771
61976a914ab96be7f12bd38dd25b62be02b88ac40420f3f531064
2f6d792d636f6f6c2d646f6d61696e117468616e6b732066726f6
d2053616c6c796d7576a914ad4a0929,
C9bfb7c55de825be7f12bd38dd25b62be02b88ac40420f3f53106
42f6d792d636f6f6c2d646f6d61696e117468616e6b732066726f
6d2053616c6c796d7576a914ad4a0929e9c7c95910534b93ec072
7058a27f2b988ac7102aabfac0ee6277ca45bccca1f9845310114
399397019ea12c8e243eb8839fed12fffffffff8a592b73a8beb79
cd0d73fca981b5e35654cd2228703694357582528d8ab01fffff
ff03ca9a3b1976a9143c4b7d4b93bc6194087bbbc422fd6cd1a40
f820e88ac60ed3877161976a914ab96
```

RETURN:

```
8a592b73a8beb79cd0d73fc9bfb7c55de825ca981b5e35654cd22
28703694357582528d8ab01fffffffff03ca9a3b1976a9143c4b7d
4b93bc6194087bbbc422fd6cd1a40f820e88ac60ed3877161976a
914ab96be7f12bd38dd25b62be02b88ac40420f3f5310642f6d79
2d636f6f6c2d646f6d61696e117468616e6b732066726f6d20536
16c6c796d7576a914ad4a0929e9c7c95910534b93ec0727058a27
f2b988ac7102aabfac0ee6277ca45bccca1f98453101143993970
19ea12c8e243eb8839fed12fffffffff
```

- V. `bst-cli Sendrawtransaction` which submits raw transaction (serialized, hex-encoded) to local node and network. It accepts hex of a signed raw transaction as input, and returns transaction hash.

Usage example:

```
bst-cli Sendrawtransaction
8a592b73a8beb79cd0d73fc9bfb7c55de825ca981b5e35654cd22
28703694357582528d8ab01fffffffff03ca9a3b1976a9143c4b7d
4b93bc6194087bbbc422fd6cd1a40f820e88ac60ed3877161976a
914ab96be7f12bd38dd25b62be02b88ac40420f3f5310642f6d79
2d636f6f6c2d646f6d61696e117468616e6b732066726f6d20536
16c6c796d7576a914ad4a0929e9c7c95910534b93ec0727058a27
```

```
f2b988ac7102aabfac0ee6277ca45bccca1f98453101143993970
19ea12c8e243eb8839fed12fffffffff
```

RETURN:

```
460ff04e500afd4c6164d70f0421b44e8d4979dfc11f33bf7f7cf
3ad45333bb5
```

- VI. `bst-cli signrawtransactionwithkey` which signs raw transaction (serialized, hex-encoded). The second argument is an array of base58-encoded private keys that will be the only keys used to sign the transaction. The third optional argument (may be null) is an array of previous transaction outputs that this transaction depends on but may not yet be in the blockchain. It returns signed transaction hex, completion status (true/false) with errors in case of any.

```
bst-cli signrawtransactionwithkey
```

```
8a592b73a8beb79cd0d73fc9bfb7c55de825ca981b5e35654cd22
28703694357582528d8ab01fffffffff03ca9a3b1976a9143c4b7d
4b93bc6194087bbbc422fd6cd1a40f820e88ac60ed3877161976a
914ab96be7f12bd38dd25b62be02b88ac40420f3f5310642f6d79
2d636f6f6c2d646f6d61696e117468616e6b732066726f6d20536
16c6c796d7576a914ad4a0929e9c7c95910534b93ec0727058a27
f2b988ac7102aabfac0ee6277ca45bccca1f98453101143993970
19ea12c8e243eb8839fed12fffffffff
[ { 5Kb8kLf9zgWQnogidDA76MzPL6TsZZY36hWXMssSzNydYXYB9KF
} ]
```

RETURN:

```
D6cd1a40f820e88ac60ed3877161976a914ab96be7f12bd38dd25
b62be02b88ac40420f3f53128d8ab01ffff0642f6d792d636f6f6
c2d646f6d61696e117468616e6b732066726f6d2053616c6c796d
7576a914ad4a0929e9c7c95910534b93ec0727058a27f2b988ac7
102aabfac0ee6277ca45bccca1f9845310114399397019ea12c8e
243eb8839fed12fffffffff8a592b73a8beb79cd0d73fc9bfb7c55
de825ca981b5e35654cd2287036943575825ffff03ca9a3b1976
a9143c4b7d4b93bc6194087bbbc422f, true
```

2.4.4. Workflow example

In a theoretical scenario, John owns a domain `mydomain.com` which Mary is interested in. Mary can use `bst-cli name_show mydomain.com` to find details about who owns the domain and how the domain can be obtained. The command returns the following info:

```
mydomain.com,
"current price 1BST paid to
34FYZErJ3CzdtQTjzxGyTPVrYBZAzeKKw4"
3G8upiyMDTaA6YYiHUoUyikuBewoLGsuGt
```

Which means that mydomain.com is for sale at 1BST price. The money needs to be transferred to the wallet address provided in the data info. The wallet containing the domain name is the bottom one.

If Mary wants the domain for 1BST, she needs to prepare an atomic transaction that will contain both the transfer and the payment.

First, she needs to prepare the payment information, i.e. the transaction id of a transaction which unspent output can cover the transfer, and the vout of the transaction output to be used. For example, if Mary owns 4 BST from an unspent transaction, she needs to send 1BST to John and the rest to a wallet she creates for the purpose. Given that the vout to be used is the first on the list, its number is 0.

Based on that info, she can create a new raw transaction. See the example below:

```
bst-cli Createrawtransaction [{"txid":
"460ff04e500afd4c6164d70f0421b44e8d4979dfc11f33bf7f7c
f3ad45333bb5", "vout":0},
{"34FYZErJ3CzdtQTjzxGyTPVrYBZAzeKKw4":1.00,
"3G8upiyMDTaA6YYiHUoUyikuBewoLGsuGt":3.20}]
```

Next, Mary needs to create name_operation details and add them to the raw transaction by using namerawtransaction. For example:

```
bst-cli namerawtransaction
8a592b73a8beb79cd0d73fc9bfb7c55de825ca981b5e35654cd22
28703694357582528d8ab01fffffffff03ca9a3b1976a9143c4b7d
4b93bc6194087bbbc422fd6cd1a40f820e88ac60ed3877161976a
914ab96be7f12bd38dd25b62be02b88ac40420f3f5310642f6d79
2d636f6f6c2d646f6d61696e117468616e6b732066726f6d20536
16c6c796d7576a914ad4a0929e9c7c95910534b93ec0727058a27
f2b988ac7102aabfac0ee6277ca45bccca1f98453101143993970
19ea12c8e243eb8839fed12fffffffff 1
{"op":"name_update", "name":"mydomain.com"
"value":"not for sale"}
```

The transaction needs to be signed before it is sent for verification to John. For that purpose, Mary can use `bst-cli signrawtransactionwithkey` using the transaction hex obtained as a result of `namerawtransaction`. Mary can either use her private key stored locally, or an existing key that she stored in the blockchain.

The hex of the partially signed transaction can be sent to John, who can decode the transaction in order to verify it.

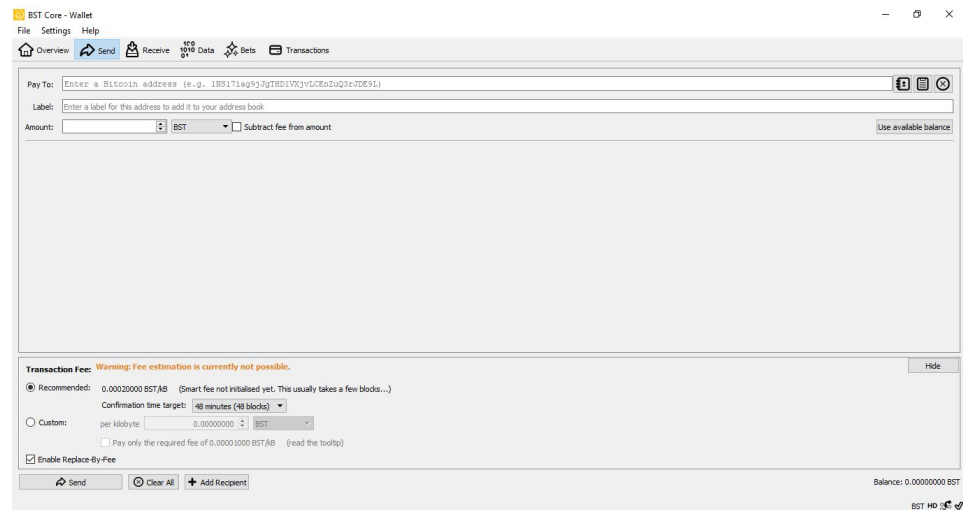
If John is happy with the transaction, he can sign it with a key and send it to the blockchain using `bst-cli Sendrawtransaction`. In return he will get a hash of the transaction. When the transaction is confirmed, the operations take place and Mary becomes the new owner of mydomain.com.

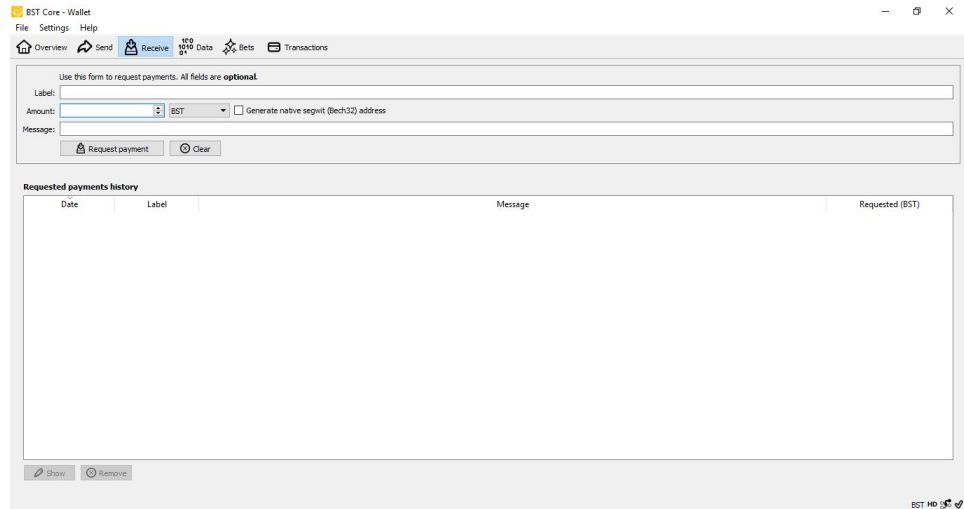
2.5. Desktop application & Wallet

The desktop application serves as a desktop interface to the blockchain. The application offers the below described functionalities. In addition to that, it also allows to encrypt the wallet, sign or verify message. The latter can be used in the above scenario, for DNS raw transaction signing.

2.5.1. Value transfer

Desktop application can be used to send and request/receive value transfers in BST.

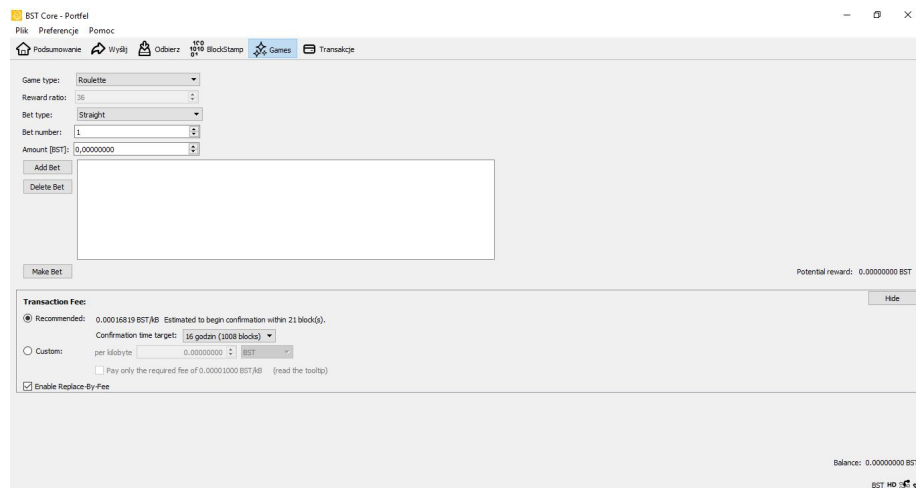




The transfer tab allows user to define the fee and choosing confirmation target time by selecting from the recommended fees or defining one's own.

2.5.2. Gaming

The application provides betting interface for lottery and roulette. It allows to define numbers and probability (modulo) for lottery or select bet type for roulette.

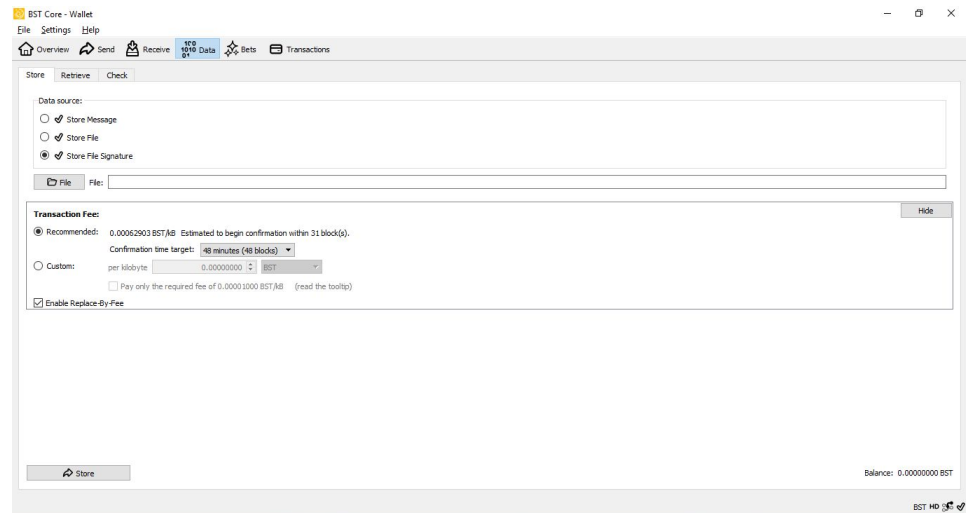


As in the value transfer, Bets tab also allows users to define the fee they want to pay, or even select 0.00 BST fee.

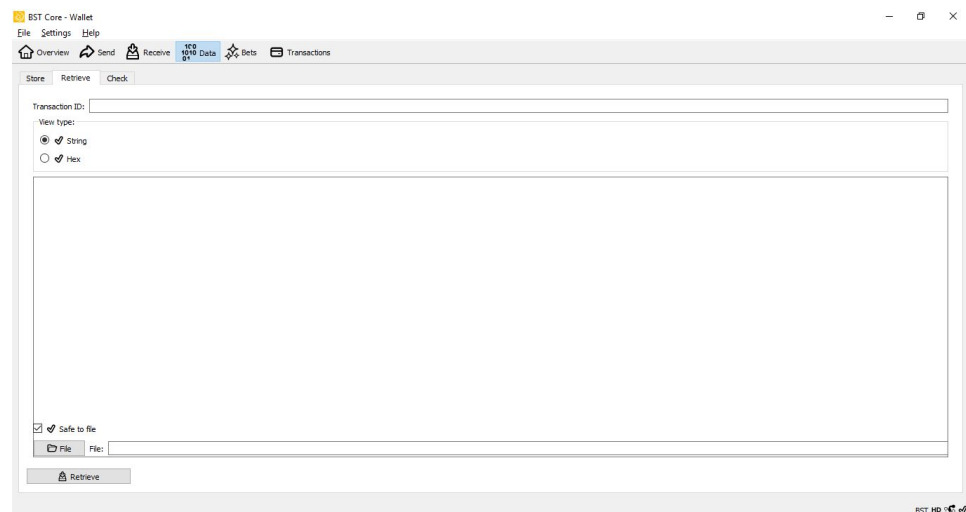
2.5.3. Data

Data tab provides three basic functionalities - Store, Retrieve, and Check.

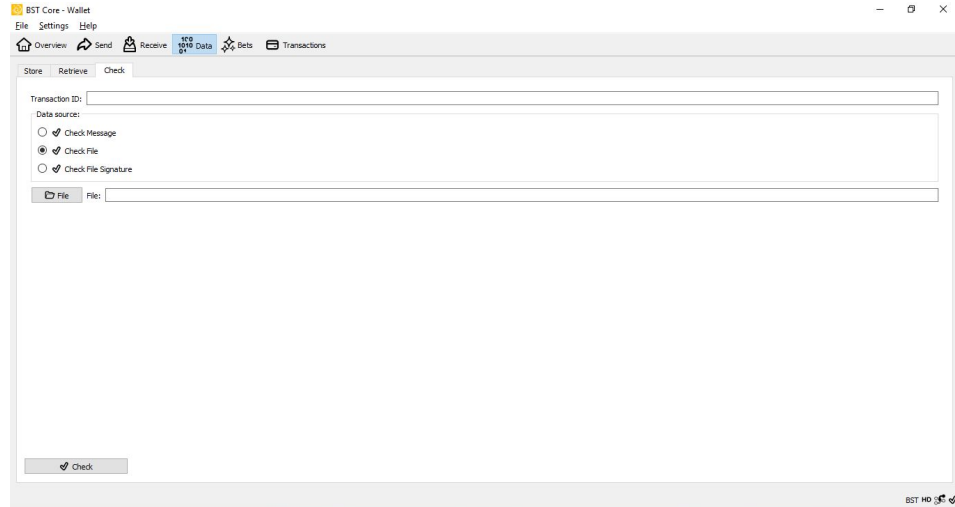
Store option allows to store a message, a file, or a file signature in the blockchain. Again, it allows to define the fee one wants to pay.



Retrieve option returns content of a transaction in String or Hex. It also allows to store the content in a file.



Check option allows to verify content of a transaction and compare it with data provided by user (in a file or message).



3. Appendix 1 - bet types, syntax, and numbers

1. straight: straight_1, straight_2, ...,straight_36.

2. split:

```
split_1={1, 4},
split_2={4, 7},
split_3={7, 10},
split_4={10, 13},
split_5={13, 16},
split_6={16, 19},
split_7={19, 22},
split_8={22, 25},
split_9={25, 28},
split_10={28, 31},
split_11={31, 34},
split_12={2, 5},
split_13={5, 8},
split_14={8, 11},
split_15={11, 14},
split_16={14, 17},
split_17={17, 20},
split_18={20, 23},
split_19={23, 26},
split_20={26, 29},
split_21={29, 32},
split_22={32, 35},
split_23={3, 6},
split_24={6, 9},
```

```
split_25={9, 12},
split_26={12, 15},
split_27={15, 18},
split_28={18, 21},
split_29={21, 24},
split_30={24, 27},
split_31={27, 30},
split_32={30, 33},
split_33={33, 36},
split_34={1, 2},
split_35={2, 3},
split_36={4, 5},
split_37={5, 6},
split_38={7, 8},
split_39={8, 9},
split_40={10, 11},
split_41={11, 12},
split_42={13, 14},
split_43={14, 15},
split_44={16, 17},
split_45={17, 18},
split_46={19, 20},
split_47={20, 21},
split_48={22, 23},
split_49={23, 24},
split_50={25, 26},
split_51={26, 27},
split_52={28, 29},
split_53={29, 30},
split_54={31, 32},
split_55={32, 33},
split_56={34, 35},
split_57={35, 36}
```

3. street:

```
street_1={1, 2, 3},
street_2={4, 5, 6},
street_3={7, 8, 9},
street_4={10, 11, 12},
street_5={13, 14, 15},
street_6={16, 17, 18},
street_7={19, 20, 21},
street_8={22, 23, 24},
street_9={25, 26, 27},
```

```
street_10={28, 29, 30},  
street_11={31, 32, 33},  
street_12={34, 35, 36}
```

4. corner:

```
corner_1={1, 2, 4, 5},  
corner_2={4, 5, 7, 8},  
corner_3={7, 8, 10, 11},  
corner_4={10, 11, 13, 14},  
corner_5={13, 14, 16, 17},  
corner_6={16, 17, 19, 20},  
corner_7={19, 20, 22, 23},  
corner_8={22, 23, 25, 26},  
corner_9={25, 26, 28, 29},  
corner_10={28, 29, 31, 32},  
corner_11={31, 32, 34, 35},  
corner_12={2, 3, 5, 6},  
corner_13={5, 6, 8, 9},  
corner_14={8, 9, 11, 12},  
corner_15={11, 12, 14, 15},  
corner_16={14, 15, 17, 18},  
corner_17={17, 18, 20, 21},  
corner_18={20, 21, 23, 24},  
corner_19={23, 24, 26, 27},  
corner_20={26, 27, 29, 30},  
corner_21={29, 30, 32, 33},  
corner_22={32, 33, 35, 36}
```

5. line:

```
line_1={1, 2, 3, 4, 5, 6},  
line_2={4, 5, 6, 7, 8, 9},  
line_3={7, 8, 9, 10, 11, 12},  
line_4={10, 11, 12, 13, 14, 15},  
line_5={13, 14, 15, 16, 17, 18},  
line_6={16, 17, 18, 19, 20, 21},  
line_7={19, 20, 21, 22, 23, 24},  
line_8={22, 23, 24, 25, 26, 27},  
line_9={25, 26, 27, 28, 29, 30},  
line_10={28, 29, 30, 31, 32, 33},  
line_11={31, 32, 33, 34, 35, 36}
```

6. column:

```
column_1={1, 4, 7, 10, 13, 16, 19, 22, 25, 28, 31, 34},  
column_2={2, 5, 8, 11, 14, 17, 20, 23, 26, 29, 32, 35},  
column_3={3, 6, 9, 12, 15, 18, 21, 24, 27, 30, 33, 36}
```

7. dozen:

```
dozen_1={1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12},  
dozen_2={13, 14, 15, 16, 17, 18, 19, 20, 21, 22, 23, 24},  
dozen_3={25, 26, 27, 28, 29, 30, 31, 32, 33, 34, 35, 36}
```

8. low:

```
low={1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18}
```

9. high:

```
high={19, 20, 21, 22, 23, 24, 25, 26, 27, 28, 29, 30, 31, 32, 33, 34, 35, 36}
```

10. even:

```
even={2, 4, 6, 8, 10, 12, 14, 16, 18, 20, 22, 24, 26, 28, 30, 32, 34, 36}
```

11. odd:

```
odd={1, 3, 5, 7, 9, 11, 13, 15, 17, 19, 21, 23, 25, 27, 29, 31, 33, 35}
```

12. black:

```
black={2, 4, 6, 8, 10, 11, 13, 15, 17, 20, 22, 24, 26, 28, 29, 31, 33, 35}
```

13. red:

```
red={1, 3, 5, 7, 9, 12, 14, 16, 18, 19, 21, 23, 25, 27, 30, 32, 34, 36}
```