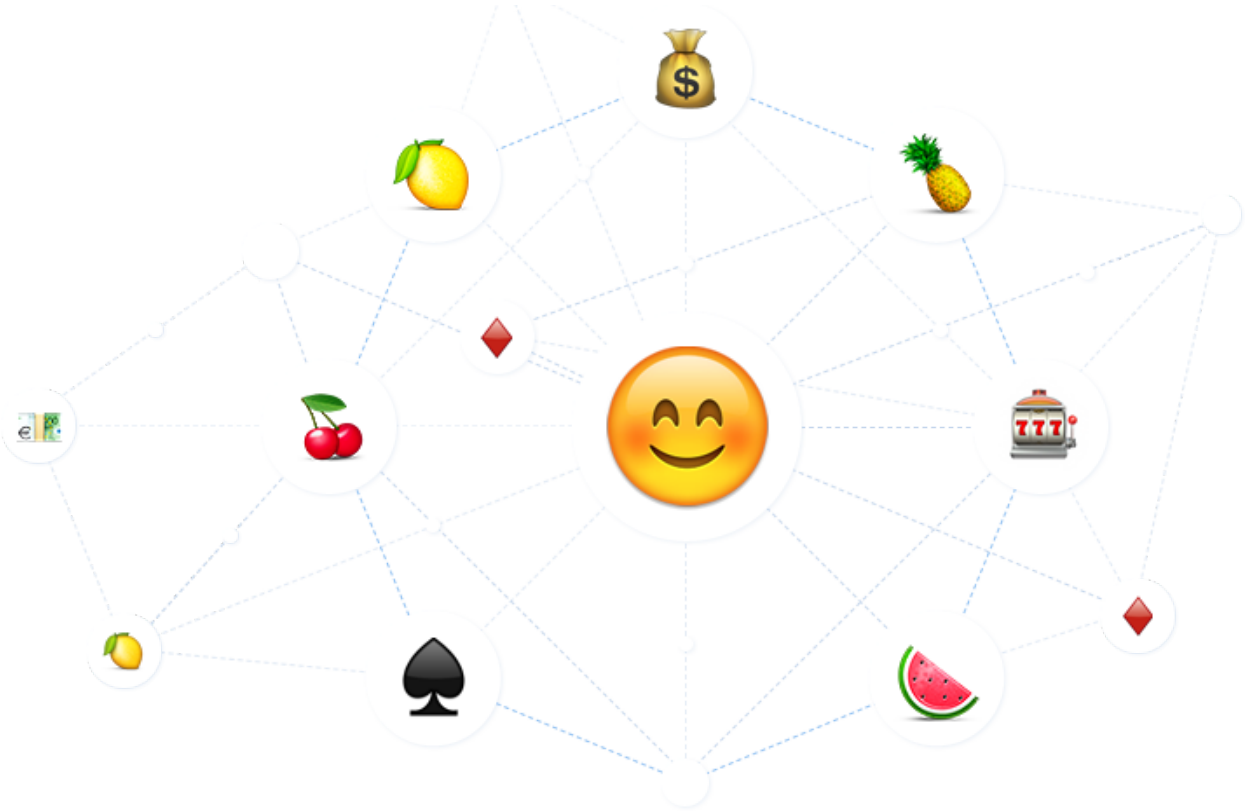
BotGaming



Technical Appendix

A blockchain powered gambling platform for messenger bots

Citizens and tax residents of the United States are not allowed to participate in the BotGaming token sale

System description

A user starts the game bot on a messenger using their account. Then they can create a game (single- or multi-user) by specifying available initial settings.

At this moment an association is initiated with the smart contract for this game (a game session is opened) where an open key required for pseudo random number generation algorithm is sent to. This key is kept in the contract and can be further used by the players to control the fair play.

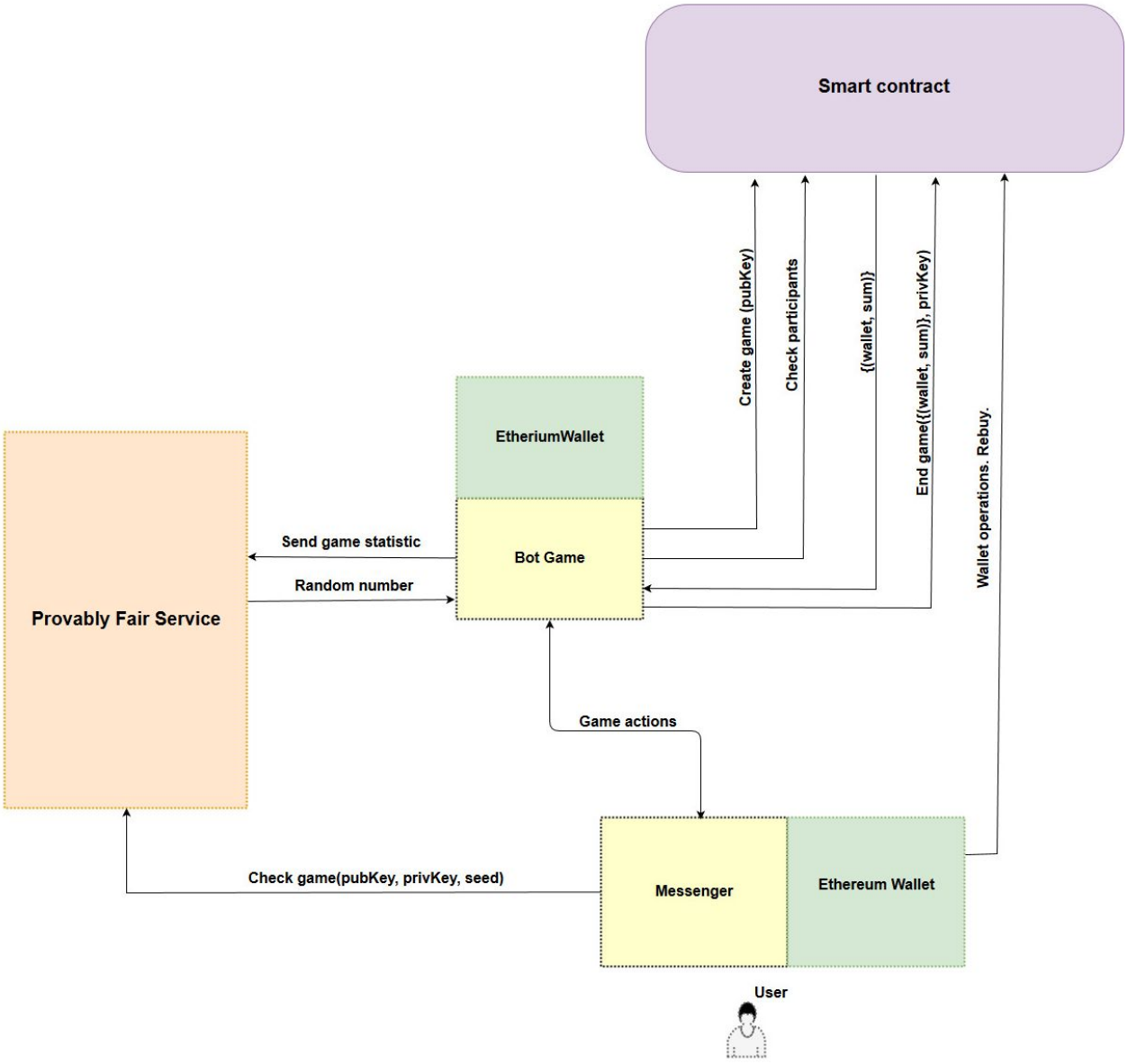
The user gets the address of the contract where tokens must be deposited in order to play. The user enters their wallet addresses which they will use for replenishment of the game bot, and then they transfer tokens to play the game from the stated wallets to the issued address of the contract.

The wallet information is required by the bot to determine that the link between the player and the game (game session) was opened by this user. The game bot then determines that the contract has been replenished and informs the player about it.

Next, in case of a multi-user game, an invitation to participate is published in the bot. Other players see the invitation, and those wishing to participate enter the number of the wallet from which the tokens will be transferred to the game session contract (similar to the initiator of the game). Next, they transfer the required tokens from their wallets to the contract, thereby confirming participation in the game. If a user transfers more or less than the agreed amount to the contract, the transaction will be rejected and the user will receive their tokens back.

When the maximum number of participants is reached, the game is initiated.

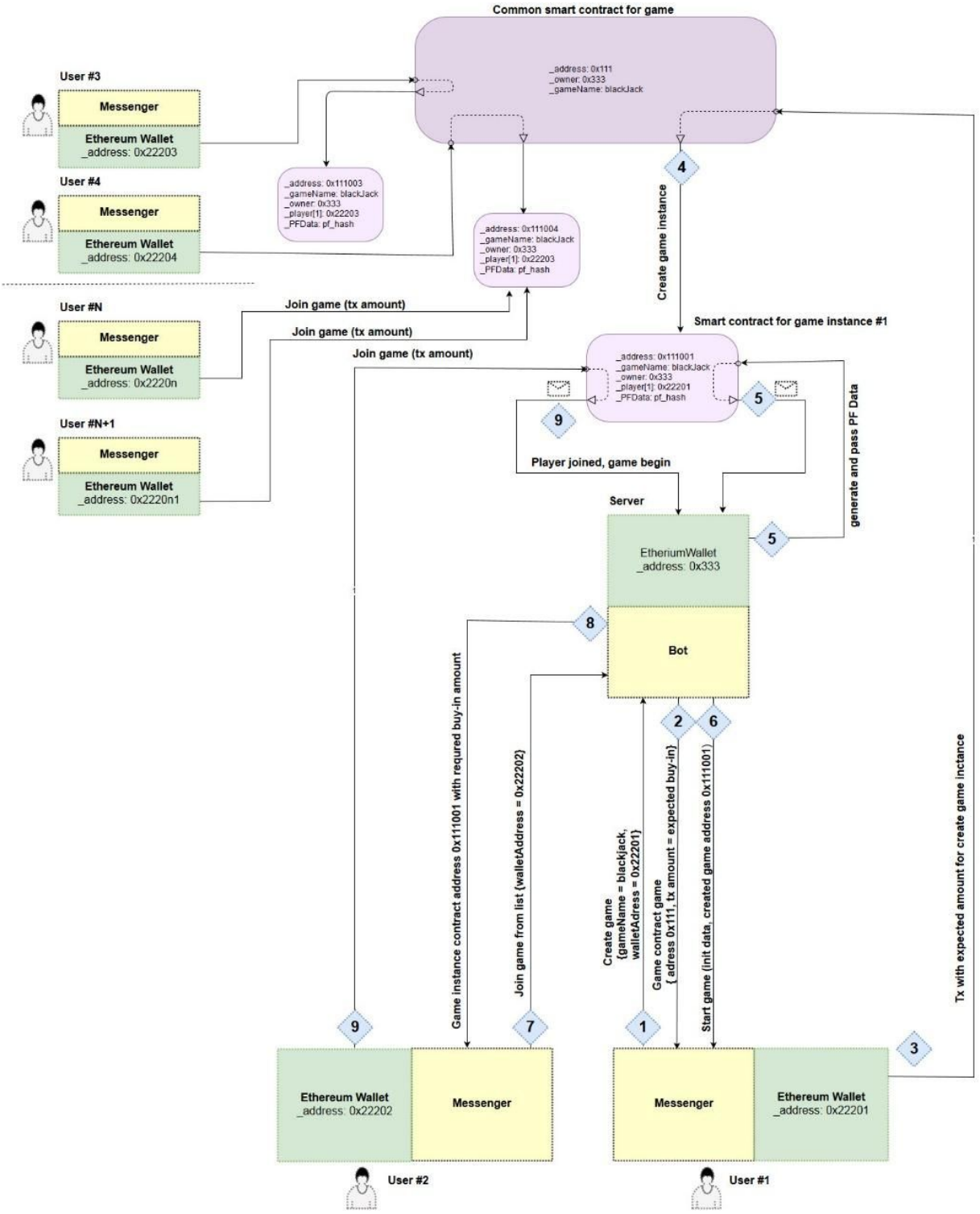
A simplified scheme of the system



Further payments from new players are not accepted to this game session, which is checked at the smart contract level. For a single-user game, it is initiated immediately once the tokens are delivered to the contract from the user who created the game.

After this, the game process is performed with numerous internal operations, but with no interaction with the main network. The history of operations is stored in a special service (**ServiceProvablyFair**). At the end of the game, a session completion command with players' balance information and private key is sent to the corresponding contract. This key and the key sent at the game session opening, together with the biases (PRNG initialization parameters) received from the players, can be further used by them in order to control integrity in the Provably Fair service or manually through the open PRNG algorithm and the source code located at GitHub. Next, the contract distributes the tokens among the players, and the changes are sent to the main network and are fixed in the blockchain.

Scheme of interaction of the system elements



Provably Fair Service

Initial requirements are imposed by the market to pseudo random number generators [(P)RNG]:

- It needs to generate a "high quality" sequence of random numbers. I.e. pass all standard tests for a uniform distribution, diehard tests, etc.
  https://en.wikipedia.org/wiki/Randomness_tests
  https://en.wikipedia.org/wiki/Diehard_tests

- It needs to be "unpredictable" in order to eliminate the possibility of an unscrupulous player to cheat on the game initiator and other players.

- It needs to support mechanisms of integrity control. For example, the technology is "provably fair." This simple and open technology enables a player to check the result of each draw (spin, hand) and to ensure that they were not deceived by the game initiator.
  https://en.wikipedia.org/wiki/Provably_fair
  https://dicesites.com/provably-fair

- It needs to work quickly and cheaply. To do this, it is necessary for RNG to work (generate random numbers) not by conducting on-chain transactions, but inside a special game session channel, i.e. off-chain. Herewith, the information for implementation of the Provably Fair goes to the smart contract of the game (blockchain) only 2 times: at the beginning of the game, when the game channel is opened, the public part of the information is recorded; and at the end of the game, when the channel is closed and the rewards are distributed, at which point the private information is disclosed, enabling the players to check the results of each drawing.

Implementation of Provably Fair in BotGaming

The service is based on the use of data from two sources, the server and the client.

Terms:

ServerSeed is a secret unique random sequence of symbols which is generated by the server at the very beginning of the game session, before any players have have made their bets yet. The ServerSeed is not transferred to the client and can not be disclosed during the game.

ServerHash is a sequence of symbols which are hash received from **ServerSeed** by hashing an irreversible function (e.g., SHA-265).

ClientSeed is a sequence of symbols which is provided by a player together with their bet.

Address is a player's unique public account address.

Nonce is a number that is increased by 1 for each random number which is required to calculate the result of a drawing, starting with 0 (or 1) for each game session.

Description of the process of a random number generation:

At the beginning of a game session, the server generates a private sequence **ServerSeed** and a public sequence **ServerHash** based on **ServerSeed**.

Then it records the public sequence in the Smart contract of the game (in blockchain), and thus **ServerHash** becomes available for the player for further integrity control of the generated random number.

The player provides their own unique sequence of symbols (**ClientSeed**) to the server and their Address.

The server computes a unique **Nonce** for each random number.

The server computes the random number from a defined range by using hash (for example, SHA-512) of the sequence of symbols are generated by concatenation of **ServerSeed, ClientSeed, Address** and **Nonce**.

Basic policies of data exchange during the random number generation :

The game initiator must not know the **ClientSeed** before they bet.

The player knows only the **ServerHash** before they bet. **ServerSeed** remains unknown (secret) before the result comes (or a sequence of results for the games with several rounds within one game session).

Compliance with these rules ensures that:

- The game initiator cannot find the result of the drawing to cheat a player, because they don't know the **ClientSeed**.
- The player is given the **ServerHash** before the computation of the result of the drawing so that they can check that the game initiator hasn't changed the **ServerSeed**, and the result computed later is truly random.
- The **ServerSeed** used to calculate the result remains unknown to the player before the end of the game, so the player cannot deceive the game initiator calculating the result of the drawing in advance.

For example:

ServerSeed: 6b0cfcf2e408a2487f8c435cc78f1db29ebc76019fb912468398dc9ed1b5feae

ServerHash: dd853cba44372af1bd9982eea9a163a1c8c45959499333626905b64a51f122af

ClientSeed: ClientSeedRandomExample

Address: 0xa27dd7a52Ad241802E0F4d24195c7552170fEDfA

Nonce: 1

Where ServerHash = sha256(ServerSeed)

We calculate the hash of the concatenated data

sha512(ServerSeed + ClientSeed + Address + Nonce)

sha512('6b0cfcf2e408a2487f8c435cc78f1db29ebc76019fb912468398dc9ed1b5feaeClientSeedRandomExample0xa27dd7a52Ad241802E0F4d24195c7552170fEDfA1')

We come at

Da93036c5059b80ff63a57ca04f1d9b3b85280b9928873764a2fb78c4072bb6367887dad534a965766df3a3a1a777d29224d080c98992a55beb10c86183f75ee

Next, we get the result of the drawing; for this, we convert the computed sequence from base 16 (hexadecimal) to base 10 (decimal) and bring the resulting number to the required range by division with remainder by the number of possible results (depending on the game rules; for example, by division with remainder by 52 for card games, by 36 for roulette, etc.)

The result (the random number) check:

Once the game session is completed, the player gets the access to the **ServerSeed** (recorded into the blockchain after completion of the game session) and can check the result of the drawing.

They can check the authenticity of the **ServerSeed** by calculating the hash by available means and comparing it with the **ServerHash** they have.

ServerHash = sha256(ServerSeed)

They can also reproduce the generation algorithm of a random number by available means for hash calculation, converting from hexadecimal format to decimal and dividing by modulus, and verify its correctness.

HexToDec(sha512(ServerSeed + ClientSeed + Address + Nonce)) % 52

In order to facilitate the verification, we will provide the player with an open service for automated verification of the results of drawings.

Implementation features of PRNG-algorithm in gambling

The implementation data works well for single-player games, i.e. slot-machines, roulettes, lotteries, etc.
For multi-player card games, such as Blackjack, Baccarat or classic poker, an extended version of the algorithm is applicable. When generating random numbers, a joint **MultiClientSeed** will be used – a concatenation of **ClientSeeds** of all players in a defined order.

MultiClientSeed = ClientSeed1 + ClientSeed2 + ...

Alternatively, a concatenation of hashes of the players' **ClientSeeds** can be used for the generation.

MultiClientSeed = sha256(ClientSeed1) + sha256(ClientSeed2) + ...