How to Integrate

```
Integrating $TSOTCHKE QRNG into your project on Solana mainnet:
import {
 Connection,
 Keypair,
 PublicKey,
 Transaction,
 sendAndConfirmTransaction,
 ComputeBudgetProgram,
} from '@solana/web3.js';
import {
 getOrCreateAssociatedTokenAccount,
 getAccount,
 TOKEN_PROGRAM_ID,
} from '@solana/spl-token';
// Connect to Solana mainnet
const connection = new Connection('https://api.mainnet-beta.solana.com', 'confirmed');
// Program and token information
const PROGRAM_ID = new PublicKey('F7E268Uek6YJvYCNaeamnxLy1umzPPTfrK2TiATxffQg');
const TSOTCHKE_TOKEN_MINT = new PublicKey('4mbdysBik3jmzD7mt6FGPDsMxnYcxExSQRFjPucdpump');
const TREASURY_ADDRESS = new PublicKey('3vuKcjqows8T19z7amN2XMkSdVcicqBuf5vVa8ZjaVfc');
// Instruction types for different random number formats
const GENERATE_RANDOM_U64 = 1;  // For uint64 random number
const GENERATE_RANDOM_DOUBLE = 2; // For double between 0-1
const GENERATE RANDOM BOOLEAN = 3; // For boolean
async function requestRandomNumber(payer, userTokenAccount) {
 // Find the program config PDA
 const [configAddress, ] = PublicKey.findProgramAddressSync(
   [Buffer.from('token_qrng_config')],
   PROGRAM ID
 );
 // Create instruction data - use the appropriate type
 const instructionData = Buffer.from([GENERATE_RANDOM_U64]);
 // Optional: Set compute budget to optimize costs
 const computeBudgetIx = ComputeBudgetProgram.setComputeUnitLimit({
   units: 200_000
 });
 // Find the clock sysvar
 // Create transaction
 const transaction = new Transaction()
   .add(computeBudgetIx)
   .add({
     keys: [
       { pubkey: payer.publicKey, isSigner: true, isWritable: true },
```

```
{ pubkey: userTokenAccount.address, isSigner: false, isWritable: true },
        { pubkey: TREASURY_ADDRESS, isSigner: false, isWritable: true },
        { pubkey: TOKEN PROGRAM ID, isSigner: false, isWritable: false },
        { pubkey: configAddress, isSigner: false, isWritable: false },
        { pubkey: SYSVAR_CLOCK_PUBKEY, isSigner: false, isWritable: false },
     ],
      programId: PROGRAM ID,
      data: instructionData,
   });
  // Send transaction
  const signature = await sendAndConfirmTransaction(
    connection,
   transaction,
    [payer],
    { commitment: 'confirmed' }
  );
 return signature;
// Extract the random number from transaction
async function getRandomNumber(signature) {
  const txInfo = await connection.getTransaction(signature, {
   commitment: 'confirmed',
   {\tt maxSupportedTransactionVersion: 0}
  });
  if (txInfo?.meta?.returnData) {
    const [encoded, encoding] = txInfo.meta.returnData;
    if (encoding === 'base64' && encoded) {
      const data = Buffer.from(encoded, 'base64');
      // Extract a uint64 random number
      const randomU64BigInt = data.readBigUInt64LE(0);
      return BigInt(randomU64BigInt);
   }
  }
 throw new Error('Failed to extract random number from transaction');
}
```

For more detailed integration instructions, visit our GitHub repository.