# Dummy Implementation of Ethereum, Infura, and Smart Contracts

## Introduction

This document explains how to implement a dummy communication process that mimics Ethereum, Infura, and Smart Contracts. The process includes public/private key generation, smart contract simulation, and Infura API behavior using dummy backend routes. The actual file upload and key generation processes are implemented in real-time.

## Key Steps in the Dummy Blockchain Process

1. Public/Private Key Generation:  
- Use a cryptographic library to generate public and private keys.  
- These keys mimic Ethereum wallet keys.  
  
2. Transaction Simulation:  
- Represent a smart contract interaction with a simple JSON object.  
- Implement the smart contract logic (e.g., key verification) on the backend.  
  
3. Infura Simulation:  
- Use backend routes to simulate Infura API behavior.  
- Mock request-response processes.  
  
4. Smart Contract Simulation:  
- Simulate storing and interacting with data (contract state) on the backend.

## Dummy Communication Flow

1. User Uploads a File:  
- Public and private keys are generated in real-time.  
  
2. File Access Request:  
- The user submits their public key.  
- The backend checks the public key (simulating smart contract verification).  
  
3. Dummy Smart Contract Transaction:  
- Generate a fake transaction hash.  
- Validate the public key against stored dummy records.  
  
4. Simulated Infura Communication:  
- Use dummy endpoints on the backend to simulate Infura API calls.

## Backend Implementation

### A. Public/Private Key Generation

Use the crypto library to mimic Ethereum wallet key generation.  
  
Code Example:  
const crypto = require('crypto');  
  
// Generate Public and Private Keys  
function generateKeys() {  
 const privateKey = crypto.randomBytes(32).toString('hex');  
 const publicKey = crypto.createHash('sha256').update(privateKey).digest('hex');  
 return { publicKey, privateKey };  
}  
  
// Example Usage  
const keys = generateKeys();  
console.log('Public Key:', keys.publicKey);  
console.log('Private Key:', keys.privateKey);

### B. Smart Contract Simulation

Here, the backend acts as the smart contract logic:  
- Verifies public keys.  
- Simulates storing and interacting with data (dummy contract state).  
  
Code Example:  
const express = require('express');  
const app = express();  
app.use(express.json());  
  
let userContracts = {}; // Dummy "smart contract" store { publicKey: metadata }  
  
// Smart Contract Simulation: Register Public Key  
app.post('/register-key', (req, res) => {  
 const { publicKey } = req.body;  
 if (userContracts[publicKey]) {  
 return res.status(400).send({ message: "Key already registered!" });  
 }  
 userContracts[publicKey] = { balance: 0, fileAccess: true };  
 res.send({ message: "Public key registered successfully!" });  
});  
  
// Verify Key  
app.post('/verify-key', (req, res) => {  
 const { publicKey } = req.body;  
 if (userContracts[publicKey]) {  
 return res.send({ message: "Public key verified!", status: "Success" });  
 }  
 res.status(400).send({ message: "Invalid public key!" });  
});

### C. Infura Communication Simulation

Instead of connecting to Infura, the backend mimics its API behavior:  
  
Code Example:  
// Simulate Infura API  
app.post('/infura/send-tx', (req, res) => {  
 const txHash = crypto.randomBytes(16).toString('hex');  
 res.send({ message: "Transaction sent to Infura!", txHash });  
});  
  
app.post('/infura/deploy-contract', (req, res) => {  
 const contractAddress = '0x' + crypto.randomBytes(20).toString('hex');  
 res.send({ message: "Smart contract deployed!", address: contractAddress });  
});

## Workflow

1. User Logs In:  
- OTP is sent and verified.  
- Public/Private keys are generated.  
  
2. Register Public Key:  
- User's public key is registered using the backend route.  
  
3. Upload Document:  
- File is uploaded to cloud storage.  
- File metadata is linked to the public key.  
  
4. Request File Access:  
- User submits their public key for verification.  
  
5. Simulate Infura Communication:  
- Dummy endpoints simulate sending transactions or smart contract deployment.

## Output Example

1. Key Generation:  
{  
 "publicKey": "a1b2c3d4e5f6g7h8i9j0",  
 "privateKey": "b2c3d4e5f6g7h8i9j0a1"  
}  
  
2. Register Key:  
{ "message": "Public key registered successfully!" }  
  
3. Transaction Simulation:  
{ "message": "Transaction successful!", "txHash": "abcdef1234567890" }  
  
4. Access Verification:  
{ "message": "Public key verified!", "status": "Success" }