# Step-by-Step Guide: Blockchain-Based Autonomous Notarization System

## Step 1: Plan and Define Requirements

1. Understand the Key Features:

- Register documents on the blockchain with a digital signature.

- Use Ethereum smart contracts to validate the signature.

- Generate a transaction receipt as proof of notarization.

2. Decide Technology Stack:

- Blockchain: Ethereum (testnets: Goerli, Sepolia, or Mainnet for production).

- Programming: Solidity (for smart contracts) and JavaScript (using Web3.js).

- Middleware: Infura (to connect to the Ethereum network).

- Frontend: React or another framework for the user interface.

- Hardware: eID card reader.

## Step 2: Set Up Your Environment

1. Install Node.js and npm:

- Install Node.js (includes npm) to use blockchain libraries.

```bash

sudo apt install nodejs npm # Linux

brew install node # macOS

```

2. Install Development Tools:

- Truffle: Ethereum development framework.

```bash

npm install -g truffle

```

- Ganache: Local blockchain for testing.

```bash

npm install -g ganache-cli

```

3. Create an Infura Account:

- Sign up at [Infura](https://infura.io/) and create a new project.

- Copy the Project ID and API endpoint URL.

4. Install Wallet:

- Use MetaMask to manage your Ethereum account.

## Step 3: Develop the Smart Contracts

1. Create a Truffle Project:

```bash

mkdir notarization-system

cd notarization-system

truffle init

```

2. Write the Smart Contracts:

Example for Signature Verification:

```solidity  
// SPDX-License-Identifier: MIT  
pragma solidity ^0.8.0;  
  
contract SignatureVerification {  
 struct Document {  
 bytes32 hash;  
 address signer;  
 uint256 timestamp;  
 }  
  
 mapping(bytes32 => Document) public documents;  
  
 event DocumentRegistered(bytes32 indexed hash, address indexed signer, uint256 timestamp);  
  
 function registerDocument(bytes32 \_hash) public {  
 require(documents[\_hash].signer == address(0), "Document already registered.");  
 documents[\_hash] = Document(\_hash, msg.sender, block.timestamp);  
 emit DocumentRegistered(\_hash, msg.sender, block.timestamp);  
 }  
  
 function verifyDocument(bytes32 \_hash) public view returns (bool, address, uint256) {  
 Document memory doc = documents[\_hash];  
 if (doc.signer != address(0)) {  
 return (true, doc.signer, doc.timestamp);  
 }  
 return (false, address(0), 0);  
 }  
}  
```

3. Compile and Migrate:

- Compile the contract:

```bash

truffle compile

```

- Deploy locally (use Ganache or a testnet later):

```bash

truffle migrate --network development

```

## Step 4: Deploy to Ethereum Using Infura

1. Configure Deployment in Truffle:

Edit truffle-config.js:

```javascript  
const HDWalletProvider = require('@truffle/hdwallet-provider');  
  
const mnemonic = 'your metamask seed phrase here'; // Replace with your wallet's mnemonic  
const infuraKey = 'your-infura-project-id'; // Replace with your Infura Project ID  
  
module.exports = {  
 networks: {  
 goerli: {  
 provider: () => new HDWalletProvider(mnemonic, `https://goerli.infura.io/v3/${infuraKey}`),  
 network\_id: 5, // Goerli's id  
 gas: 5500000, // Gas limit  
 confirmations: 2, // # of confs to wait between deployments  
 timeoutBlocks: 200, // # of blocks before a deployment times out  
 skipDryRun: true // Skip dry run before migrations  
 }  
 },  
 compilers: {  
 solc: {  
 version: "^0.8.0"  
 }  
 }  
};  
```

2. Deploy on Goerli Testnet:

```bash

truffle migrate --network goerli

```