# Blockchain-Based Autonomous Notarization System Using National eID Cards

## Introduction

This document outlines the step-by-step process for creating a Blockchain-Based Autonomous Notarization System using National eID Cards. The project focuses on fixed-date notarizations, leveraging blockchain technology and smart contracts to automate verification and ensure data integrity and transparency.

## Phase 1: Requirements Analysis and Planning

1. Understand the Scope:

- The project automates fixed-date notarizations using blockchain and national eID cards.

- System Components:

- Verifying document authenticity.

- Recording transaction receipts on a blockchain.

- Utilizing smart contracts for verification.

2. Technologies and Tools:

- Blockchain Platform: Ethereum or similar.

- Programming Language: Solidity (for smart contracts), Swift/JavaScript (for front-end/mobile app).

- Test Environment: Ethereum Testnets like Rinkeby or Goerli.

- Hardware: eID card reader (conforming to ISO standards).

- Middleware: Libraries for blockchain integration (e.g., Web3.js or Ethers.js).

3. Set Clear Objectives:

- Automate notarizations using national eID cards.

- Ensure data integrity and transparency via blockchain.

- Enable seamless user interaction through an intuitive interface.

## Phase 2: System Architecture Design

Design the system architecture with the following components:

1. Client-Side Application:

- Mobile/desktop app to interact with the eID card and hash the document.

- Interface to send transactions to the blockchain.

2. Smart Contracts:

- Signature Verification Contract: Verifies the document hash and digital signature.

- Public Key Management Contract: Manages keys associated with eID cards.

3. Blockchain: Public blockchain like Ethereum for storing notarization records.

4. Government Backend: A government agency service to manage eID card issuance and revocation.

## Phase 3: Development

1. Develop the eID Card Interface:

- Implement the client-side application for reading the eID card using NFC/contact card readers.

- Create digital signatures and hash document content (e.g., using SHA256).

2. Create the Smart Contracts:

- Signature Verification Contract: Validate digital signatures and store document hashes.

- Public Key Management Contract: Manage keys for eID card validation (add, revoke keys).

3. Integrate Blockchain: Use libraries like Web3.js or Ethers.js for interaction with blockchain.

4. Build the User Interface: A user-friendly app/website for document upload, hashing, and signing.

5. Test and Deploy: Test on Ethereum testnets (Rinkeby, Goerli). Optimize gas fees and ensure security.

## Phase 4: Security and Performance

1. Security:

- Use secure hashing algorithms (e.g., SHA256).

- Prevent impersonation and tampering with secure eID card management.

2. Performance:

- Optimize gas consumption for Ethereum transactions.

- Consider Layer 2 solutions (e.g., Optimistic Rollups) for scalability.

## Phase 5: Documentation and Deployment

1. Document the Project: Write clear technical and user documentation for the system.

2. Deploy the System: Release the smart contracts on the blockchain and launch the application.

## Tools and Resources

1. Smart Contract Development:

- Solidity documentation: https://soliditylang.org/

- Remix IDE for testing contracts: https://remix.ethereum.org/

2. Blockchain Libraries:

- Web3.js: https://web3js.readthedocs.io/

- Ethers.js: https://docs.ethers.org/

3. Cryptographic Libraries: Node.js crypto module for hashing and signing.

4. Ethereum Testnets: Rinkeby, Goerli for experimentation.