Centurion University
Shaping Lives... Empowering Communities...

School: ..... Campus: .....

Academic Year: ..... Subject Name: ..... Subject Code: .....

Semester: ..... Program: ..... Branch: ..... Specialization: .....

Date: .....

Applied and Action Learning

(Learning by Doing and Discovery)

Name of the Experiement :

* Coding Phase: Pseudo Code / Flow Chart / Algorithm

ECDSA Workshop – Digital Signatures Demo:

The tool demonstrates Elliptic Curve Digital Signature Algorithm (ECDSA) - the same cryptography used by Bitcoin, Ethereum, and most blockchain networks. It shows how digital signatures provide authentication, integrity, and non-repudiation.

Key Components:

1. Elliptic Curve Cryptography (secp256k1)
   - Uses the same curve as Bitcoin/Ethereum (secp256k1)
   - Private key: A random 256-bit number (64 hex characters)
   - Public key: Derived from private key using elliptic curve multiplication
   - One-way function: Easy to generate public key from private key, but impossible to reverse
2. SHA-256 Hashing
   - Converts any message into a fixed 256-bit fingerprint
   - Ensures message integrity - any change in message changes the hash completely
3. Digital Signature Process
   - Signing: Private key + Message hash = Signature
   - Verification: Public key + Message hash + Signature = Valid/Invalid

Step-by-Step Workflow

Phase 1: Key Generation

User clicks "Generate New Key Pair"
↓
elliptic.ec('secp256k1').genKeyPair()
↓
Private Key: 64-character hex (e.g., "2f4...c3a")
↓
Public Key: 130-character hex (derived from private key)

Phase 2: Message Signing

Message: "Hello, Blockchain World!"
↓
SHA-256 Hash: "a591a6...b64b4" (64-character hex)
↓
Private Key signs the hash using ECDSA algorithm
↓
Digital Signature: "3044...022100...0220..." (DER format)

*As applicable according to the experiment.
Two sheets per experiment (10-20) to be used.

Coding Phase: Pseudo Code / Flow Chart / Algorithm

Phase 3: Signature Verification

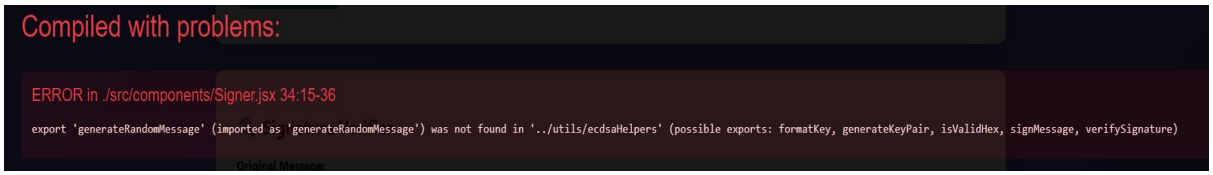
Original Message + Signature + Public Key
↓
SHA-256 Hash of original message
↓
elliptic.verify(messageHash, signature, publicKey)
↓
Result: VALID or INVALID

* Softwares used

- **VSCode**
- **Node.js**
- **React**
- **Elliptic.js, crypto.js**
- **Curve- secp256k1**
- **SHA256**
- **HTML, CSS, Javascript**

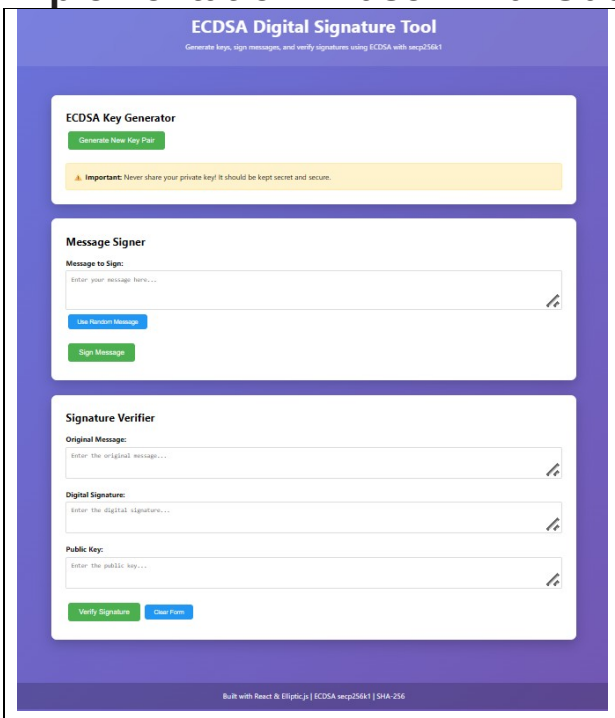
**As applicable according to the experiment.*
Two sheets per experiment (10-20) to be used.

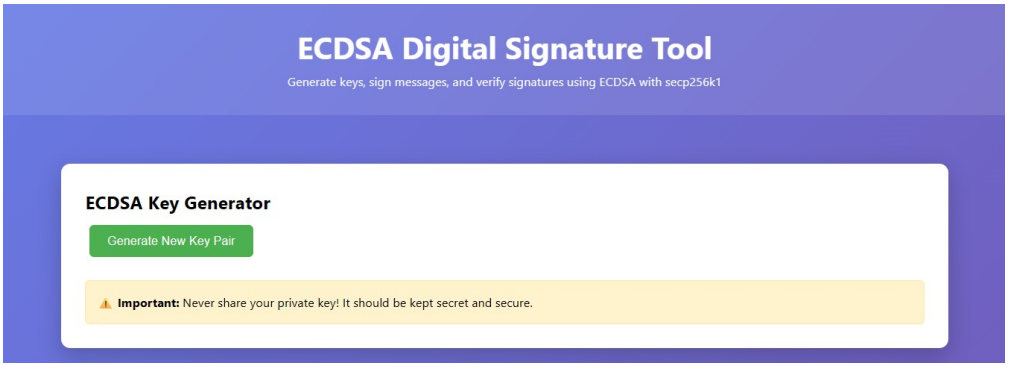
* Testing Phase: Compilation of Code (error detection)

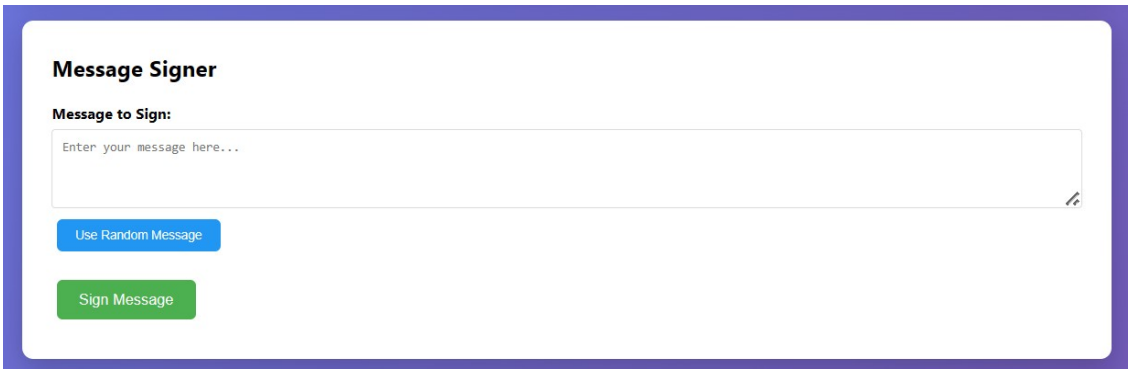


*As applicable according to the experiment.
Two sheets per experiment (10-20) to be used.

* Implementation Phase: Final Output (no error)

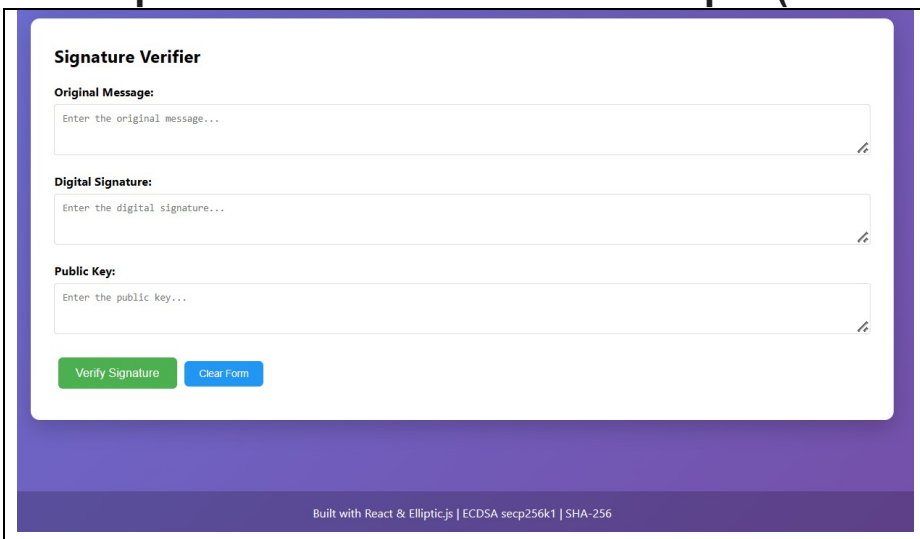


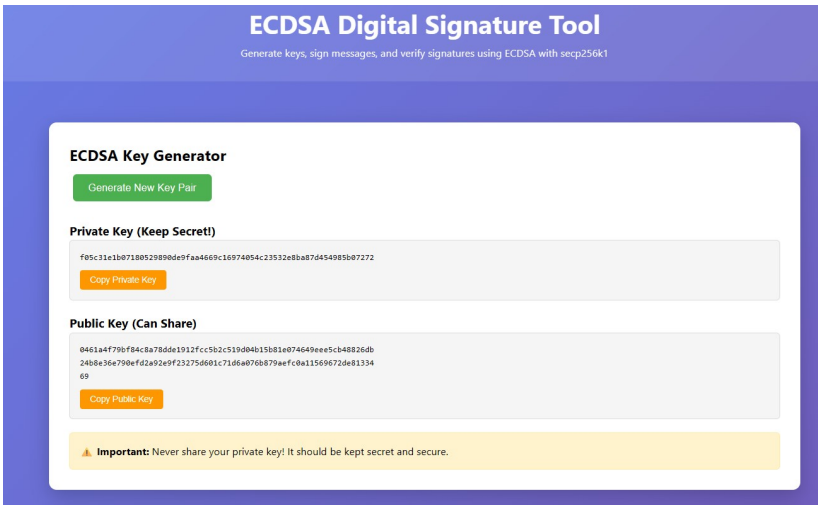


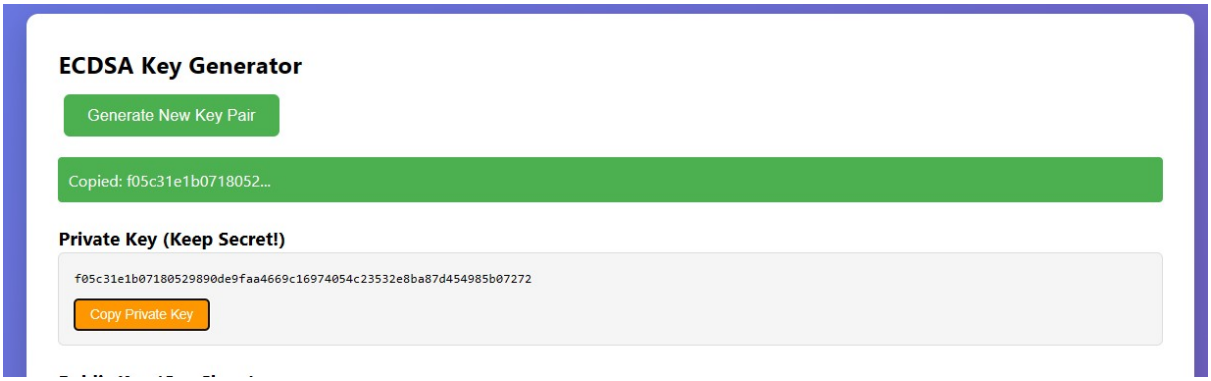


*As applicable according to the experiment.
Two sheets per experiment (10-20) to be used.

Implementation Phase: Final Output (no error)

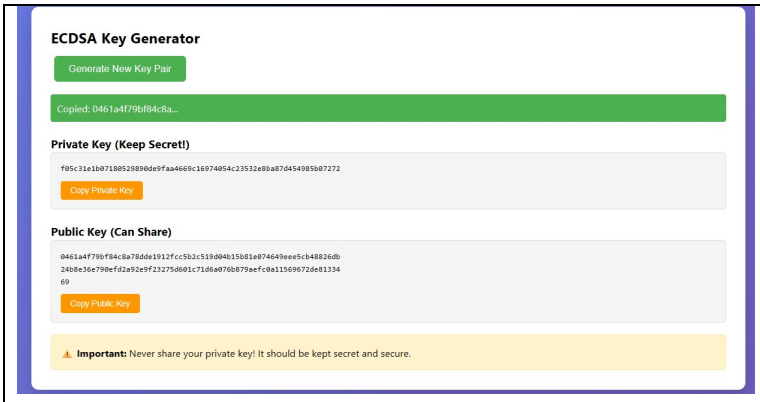


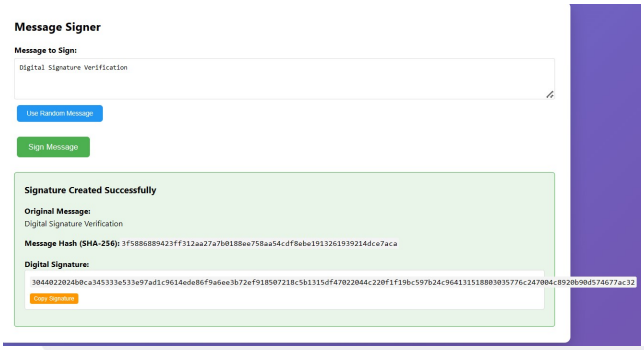


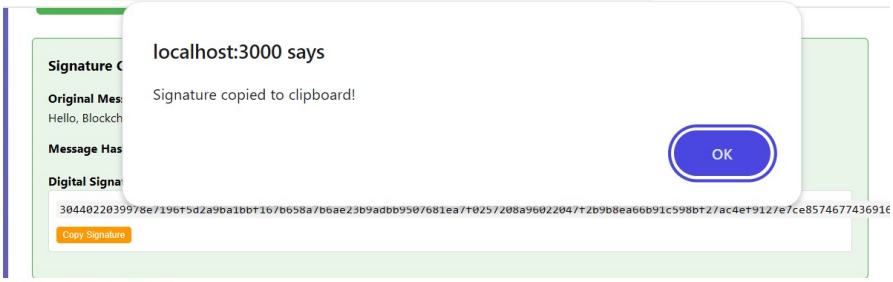


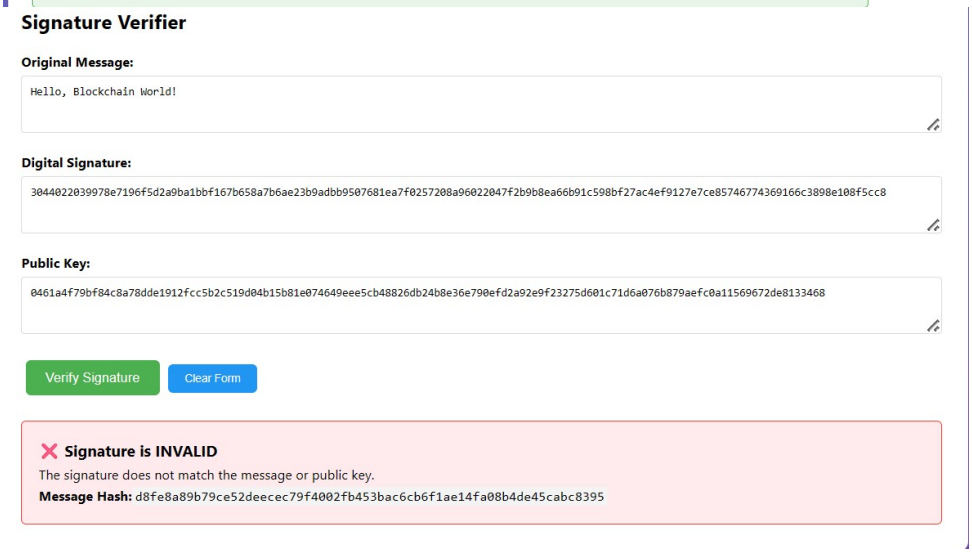
*As applicable according to the experiment.
Two sheets per experiment (10-20) to be used.

Implementation Phase: Final Output (no error)



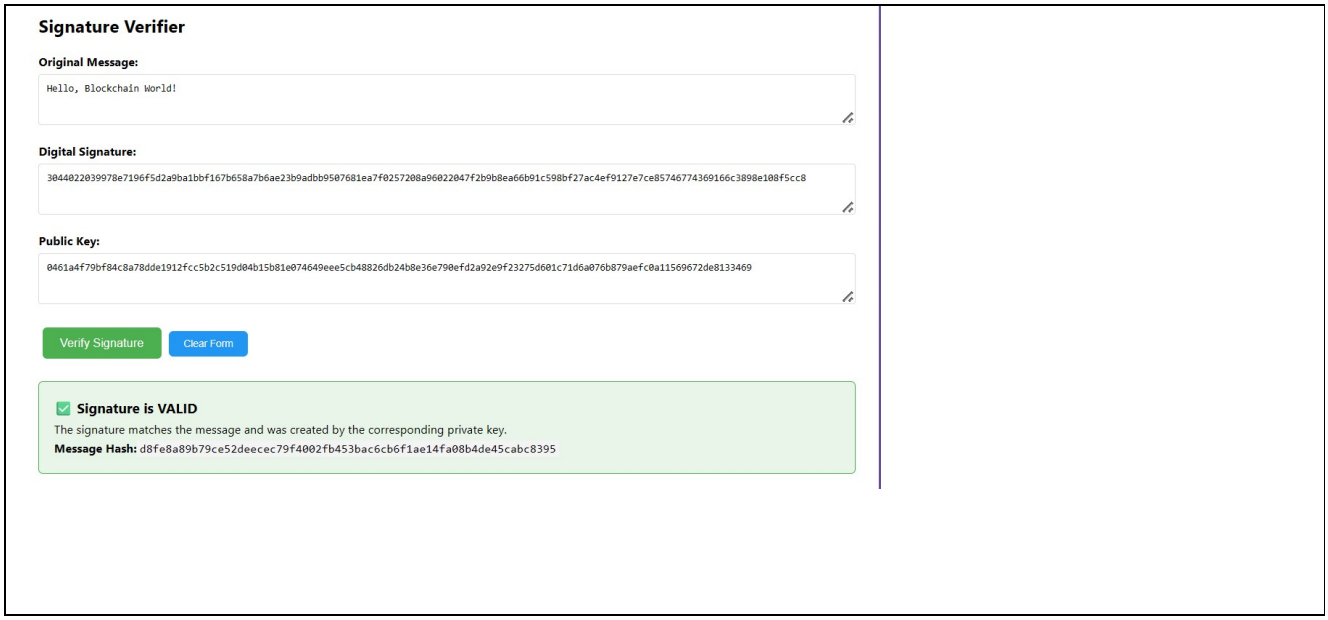






*As applicable according to the experiment.
Two sheets per experiment (10-20) to be used.

* Implementation Phase: Final Output (no error)



* Observations

1. Authentication
   - Only the holder of the private key can create a valid signature
   - Verifier can confirm the signer's identity using the public key
2. Integrity
   - Any change to the message invalidates the signature
   - SHA-256 ensures even a single character change produces completely different hash
3. Non-Repudiation
   - Signer cannot deny having signed the message
   - Signature is mathematically tied to their private key

ASSESMENT

Rubrics	Full Mark	Marks Obtained	Remarks
Concept	10		
Planning and Execution/ Practical Simulation/ Programming	10		
Result and Interpretation	10		
Record of Applied and Action Learning	10		
Viva	10		
Total	**50**		

Signature of the Student:

Name :

Regn. No. :

Signature of the Faculty:

**As applicable according to the experiment.*
Two sheets per experiment (10-20) to be used.