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 Experiment: ECDSA Workshop – Digital Signatures Demo

*Coding Phase: Pseudo Code / Flow Chart / Algorithm

1. Select the elliptic curve parameters (e.g., secp256k1).
2. Generate a private key as a large random number.
3. Compute the corresponding public key using the elliptic curve base point.
4. Prepare a message or transaction to be digitally signed.
5. Hash the message using SHA-256.
6. Generate a random value (nonce).
7. Use elliptic curve operations to compute the signature pair (r, s).
8. Send the message along with the signature to the verifier.
9. The verifier hashes the message again.
10. The verifier uses the public key to check if the computed value matches the signature.
11. If it matches, the signature is valid; otherwise, it is rejected.

*Software used :

- Web Browser (Microsoft Edge / Brave Browser)
- Text Editor
- Block chain implementation insights -https://learnmeabitcoin.com

* Testing Phase: Compilation of Code (error detection)

NO ERROR

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

* Implementation Phase: Final Output (no error)

1. What is ECDSA ?

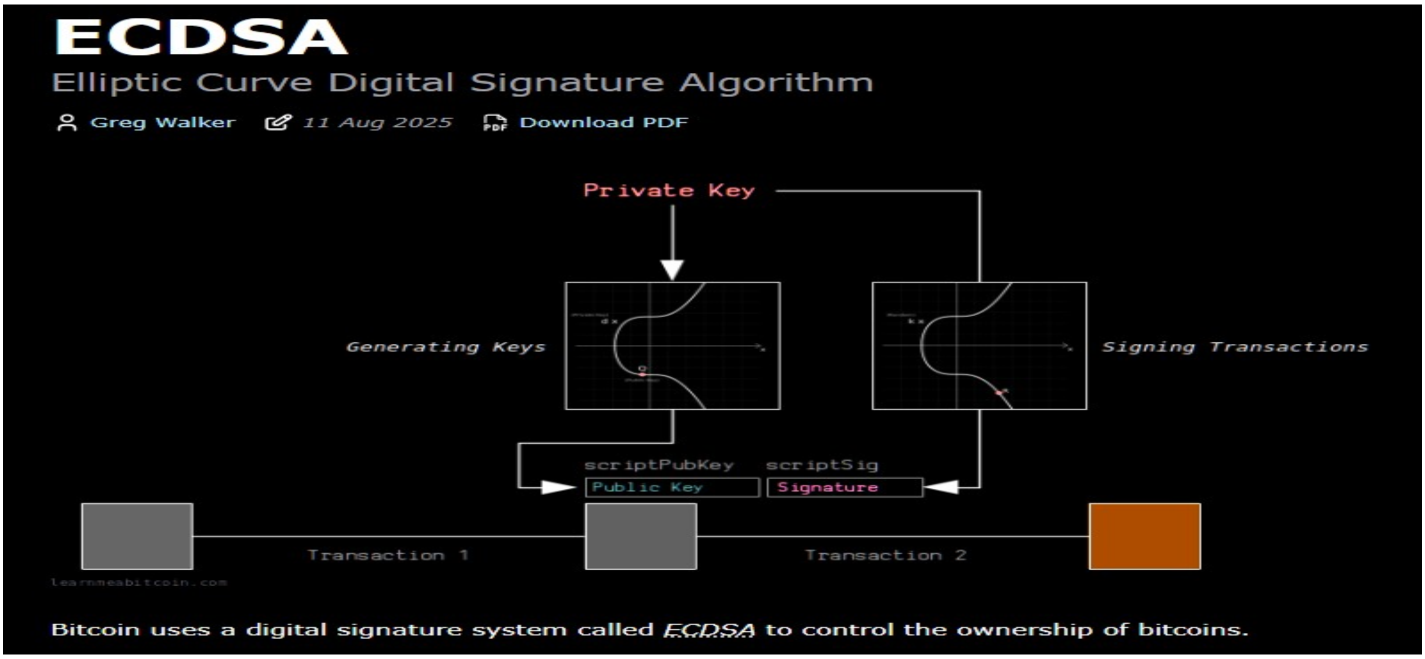
ECDSA (Elliptic Curve Digital Signature Algorithm) is a cryptographic technique used in blockchain technology to ensure authenticity and integrity of transactions. It relies on elliptic curve cryptography (ECC) to generate public and private key pairs. This report demonstrates the process of key generation, signing, and verification as implemented through an online tool, without using any programming language.

The experiment was implemented conceptually by studying how ECDSA functions within blockchain systems like Ethereum. The following flow illustrates the process (based on online diagrams and resources):

ECDSA Flow in Blockchain:

- Message Creation → Hashing → Signing → Verification
- Message / Transaction → SHA-256 → ECDSA Signature → Public Key Verification → Valid / Invalid Signature

A user creates a blockchain transaction which is hashed and signed using their private key. The signature is shared along with the message. Any blockchain node can verify the transaction using the sender's public key. Any modification in the transaction results in verification failure



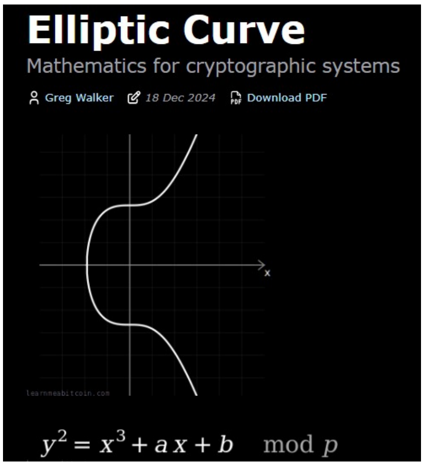
As applicable according to the experiment.
Two sheets per experiment (10-20) to be used.

* Implementation Phase: Final Output (no error)

2. Elliptic Curve Concept :

Elliptic curves form the mathematical foundation of ECDSA. Points on the curve represent public and private keys. The curve equation $y^2 = x^3 + ax + b \mod p$ defines how keys are derived and used.

An elliptic curve is used as the basis for some cryptographic systems.
The structure of the elliptic curve allows you to perform a mathematical function ("multiply") to move around the points on the curve in one direction, without being able to travel in the reverse direction. This is known as a "trapdoor function", and it's the key feature of elliptic curves that makes them ideal for use in public key cryptography.
So in short, elliptic curves have mathematical properties that make them useful for cryptography, and they're part of the digital signature system used in Bitcoin (ECDSA).



3. Secp256k1 parameters :

The Secp256k1 curve is used in Bitcoin's ECDSA implementation. It defines constants like the prime field, curve coefficients, and base point (G). These parameters are globally recognized for secure key generation.

Parameters (Secp256k1)

Satoshi chose the **secp256k1** curve for use with ECDSA, which has the following parameters:

Ruby

```ruby
# y² = x³ + ax + b
$a = 0
$b = 7

# prime field
$p = 115792089237316195423570985008687907853269984665640564039457584007908834671663 #=> 2**256 - 2**32 - 2**9 - 2**8 - 2**7 - 2**6 - 2**4 -

# number of points on the curve we can hit ("order")
$n = 115792089237316195423570985008687907852837564279074904382605163141518161494337

# generator point (the starting point on the curve used for all calculations)
$G = {
  x: 55066263022277343669578718895168534326250603453777594175500187360389116729240,
  y: 32670510020758816978083085130507043184471273380659243275938904335757337482424,
}
```

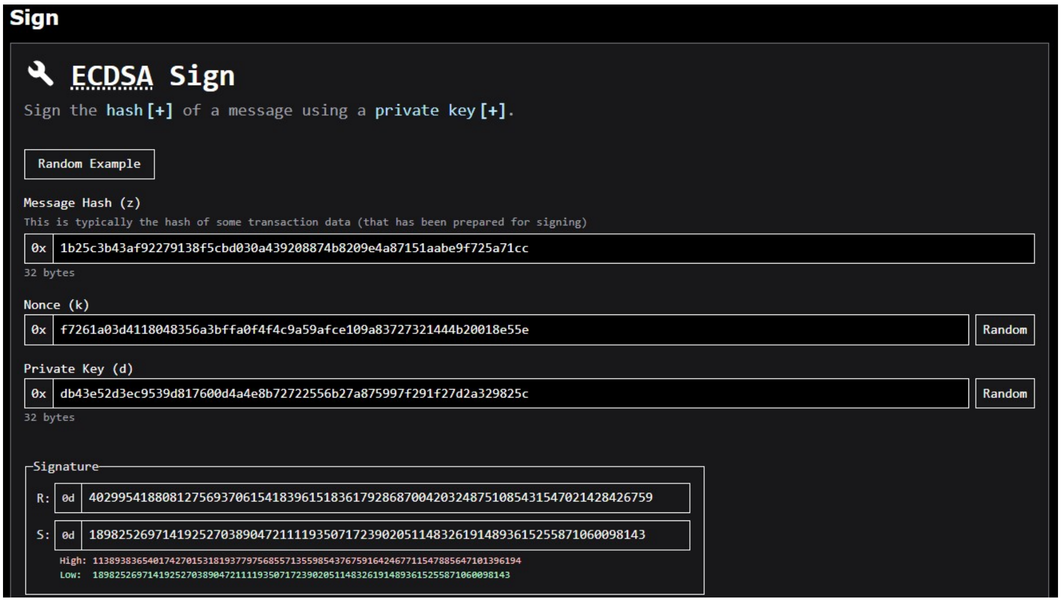

** As applicable according to the experiment.*
Two sheets per experiment (10-20) to be used.

* Implementation Phase: Final Output (no error)

- a, b – An elliptic curve is a set of points described by the equation $y^2 = x^3 + ax + b$, so this is where the a and b variables come from. Different curves will have different values for these coefficients, and a=0 and b=7 are the ones specific to secp256k1.
- p – This is the prime modulus. It's a number that keeps all of the numbers within a specific range when performing mathematical calculations (again it's specific to secp256k1). The fact that it's a prime number is a key ingredient for the cryptography to work.
- n – This is the order. It's the number of points on the curve that we can reach. It's less than p, and it's influenced by the chosen generator point (see below).
- G – This is the generator point. This is the starting point on the curve used when performing most mathematical operations. The exact origin for the choice of this point is unknown, but it's usually because it provides a high order (see above) and has shown to not have any inherent cryptographic weaknesses.

4. Key Generation :

In this step, a private key (a random number) is generated. Using elliptic curve multiplication, the public key is derived. The EC Multiply operation shown here demonstrates how each new key pair is mathematically related.

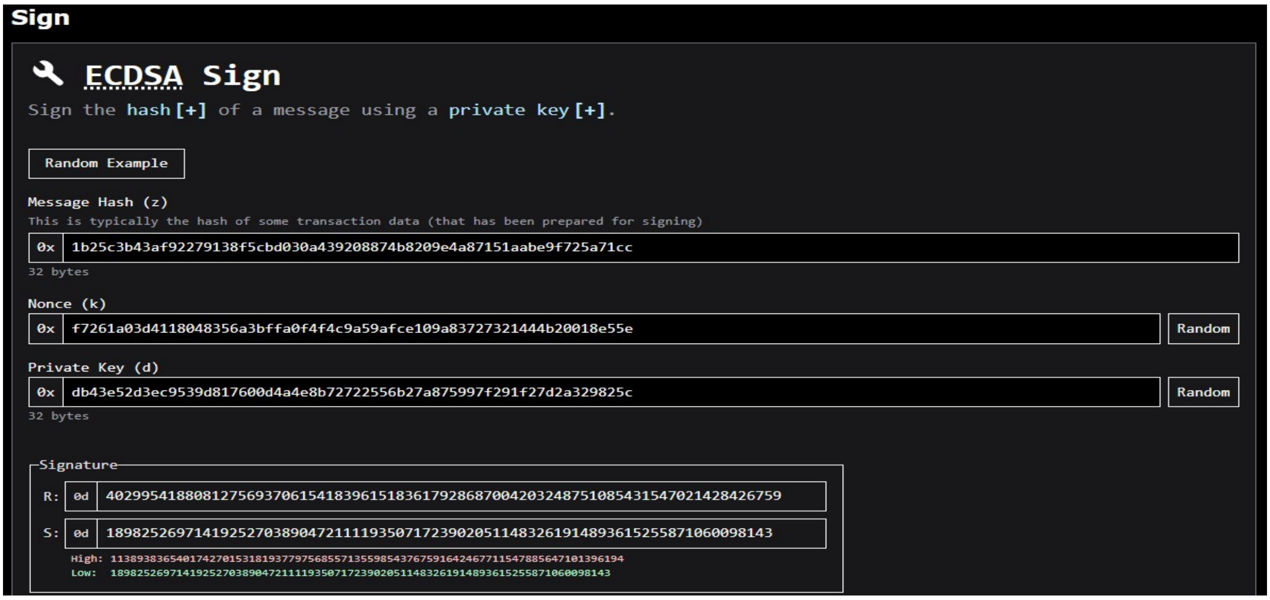


** As applicable according to the experiment.*
Two sheets per experiment (10-20) to be used.

* Implementation Phase: Final Output (no error)

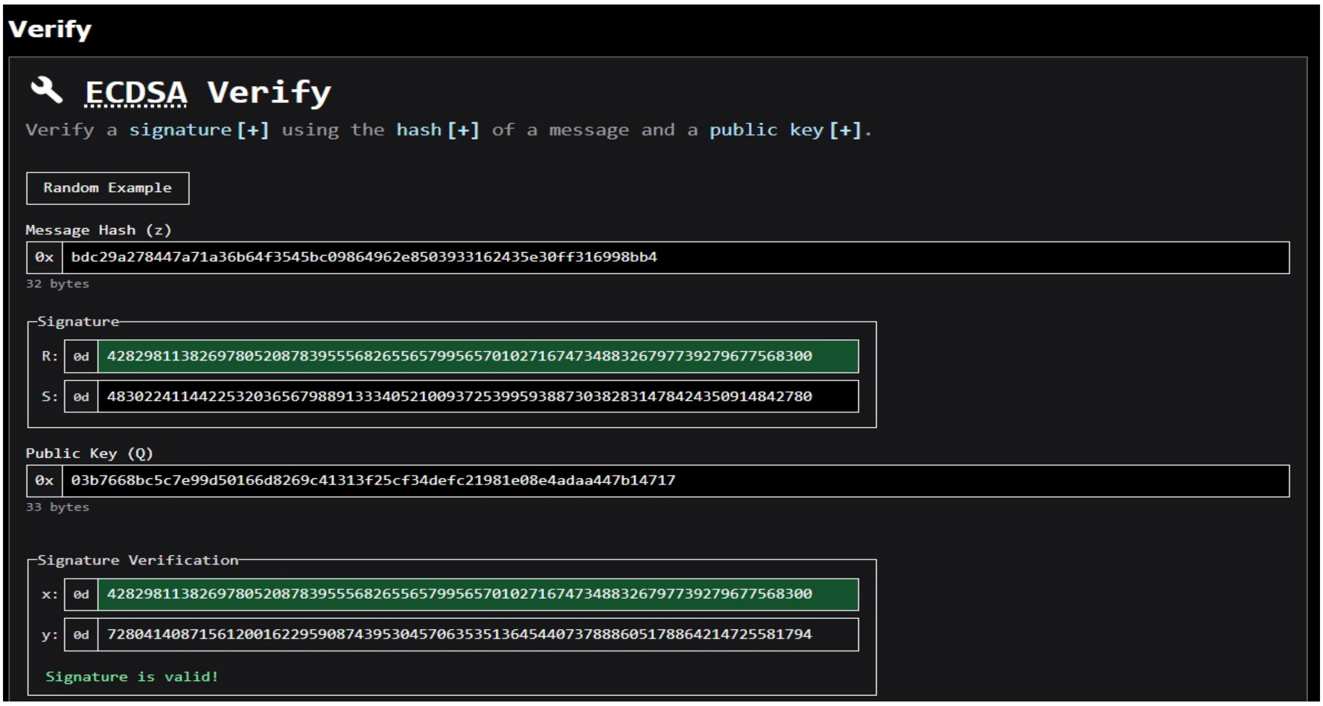
5. Signing Process:

The signing phase involves generating a unique signature using a private key and message hash. The result produces signature components (r, s), which verify the authenticity of the message.



6. Verification Process:

Verification ensures that the digital signature is valid and created by the correct private key holder. The verifier uses the public key, message hash, and signature to check mathematical consistency. If verified, the signature is declared valid.



*Observations

Key Generation:

ECDSA (Elliptic Curve Digital Signature Algorithm) uses elliptic curve cryptography to create a public key from a private key using a one-way mathematical function. This makes it highly secure.

Digital Signature Creation:

During signing, ECDSA generates a unique digital signature — made up of two values, (r, s) — that proves ownership of the private key without revealing it.

Verification Process:

To verify a signature, only the public key, message hash, and signature are needed. The private key stays secret and is never shared.

Importance of Randomness:

If the same random value (nonce) is reused during signing, the private key can be exposed. Therefore, using a new random nonce for every signature is crucial for security.

ASSESSMENT

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 :

Signature of the Faculty : ***Regn. No. :***

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