



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

ALGORITHM:

1. Open the website – learnmeabitcoin.com/technical/cryptography/elliptic-curve/ecdsa
2. Read the concept of ECDSA – Understand how elliptic curves and digital signatures work.
3. Understand Private & Public Key generation – How a public key is derived from a private key.
4. Learn the Digital Signature process – How signing and verification of transactions works.
5. Note down key points – Full form of ECDSA, its role in Bitcoin, and the verification steps.
6. Write a summary – Create a short explanation in your own words.

* Softwares used

1. Brave Web Browser
2. Text Editor

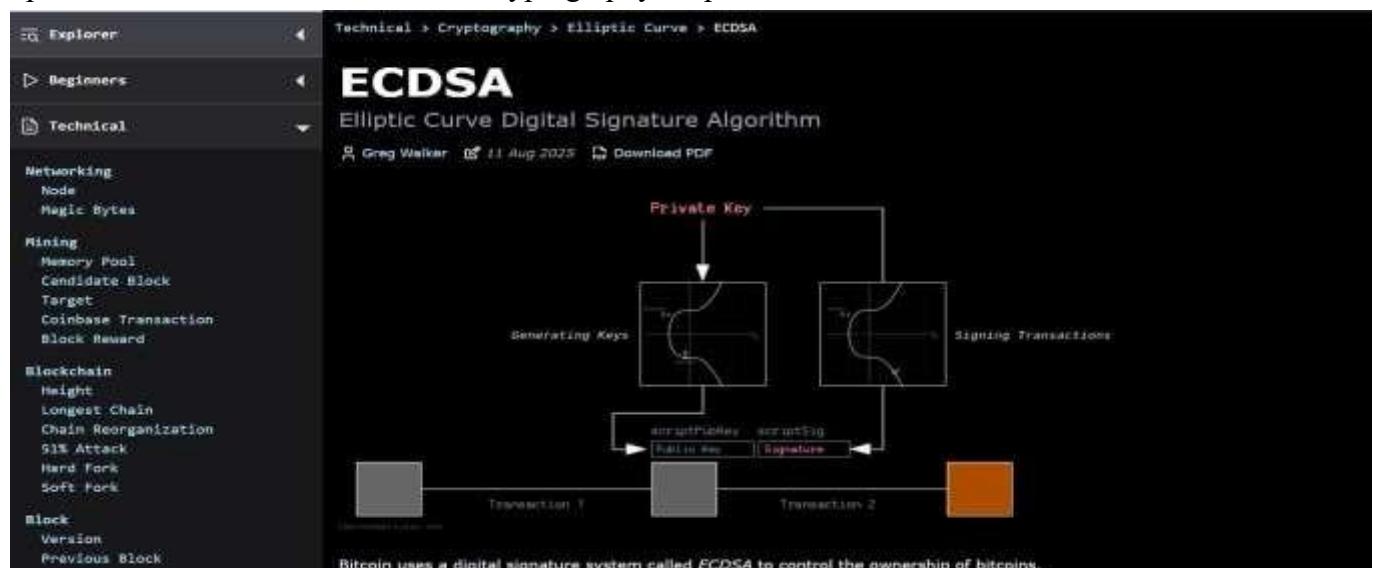
* Implementation Phase: Final Output (no error)

What is ECDSA:

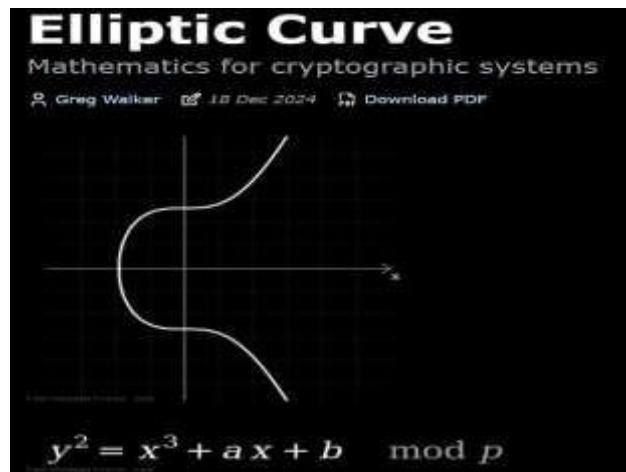
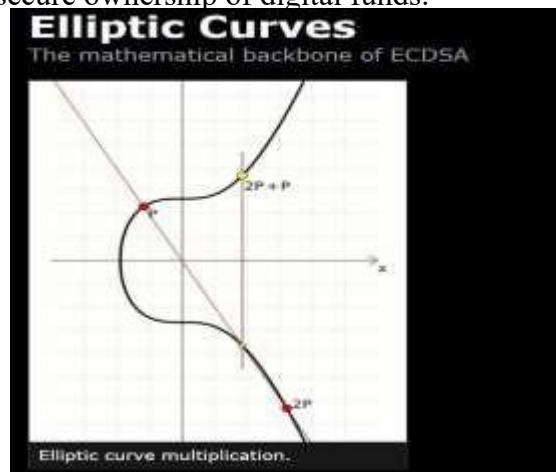
ECDSA (Elliptic Curve Digital Signature Algorithm) is a cryptographic algorithm used for creating and verifying digital signatures based on elliptic curve mathematics. It provides a secure way to prove ownership without revealing the private key. In Bitcoin and other cryptocurrencies, ECDSA is essential for verifying transactions: a user generates a private key (a secret number) and derives a public key from it using elliptic curve multiplication. When a transaction is signed with the private key, the network uses the corresponding public key to verify the signature's validity, ensuring that only the rightful owner can authorize spending of funds.

Visit this website to explore how we generate public and private key and our digital signature:-

<https://learnmeabitcoin.com/technical/cryptography/elliptic-curve/ecdsa/>



Bitcoin uses a digital signature system called ECDSA (Elliptic Curve Digital Signature Algorithm) to manage the ownership of bitcoins. This system works by allowing users to create a pair of keys: a private key, which must be kept secret, and a public key, which can be shared openly. The private key is used to generate digital signatures that prove ownership of the corresponding public key without revealing the private key itself. This mechanism ensures that users can securely receive and send bitcoins through transactions. Anyone can generate their own key pair, and others can send bitcoins to them by associating the funds with their public key. However, only the person who holds the matching private key can create valid signatures to unlock and spend those bitcoins. This prevents unauthorized access and guarantees secure ownership of digital funds.



* Implementation Phase: Final Output (no error)

ECDSA uses elliptic curve cryptography as the foundation for its digital signature system. In simple terms, public keys and signatures are points on an elliptic curve. If both the public key and the signature are generated from the same private key (a large secret number), there exists a geometric relationship between these points that proves ownership. The main mathematical operation involved is elliptic curve multiplication, which means repeatedly adding a point on the curve to itself a certain number of times to reach a new point. This operation is easy to perform in one direction but practically impossible to reverse, making it highly secure. Because of this property, elliptic curves are widely used in cryptography for generating keys and digital signatures.

```
Secp256k1 Parameters [hide]
Ruby

#  $y^2 = x^3 + ax + b$ 
$A = 9
$B = 7

# prime field
$p = 11579208923731619542357098500068790785326998466564056403945758400790834671663 #=> 2**256 - 2**32 - 2**9 - 2**8 - 2**7 - 2**6

# number of points on the curve we can hit ("order")
$N = 115792089237316195423570985000687907852837564279874904382605163141518161494337

# generator point (the starting point on the curve used for all calculations)
$G = {
  x: 55066263022277343669578718895168534326250603453777594175500187360389116729240,
  y: 32670510020758816978003085130507843184471273380659243275938904335757337482424,
}


```

Key Generation:

Key Generation

✉ Private Key [↑]

EC Multiply

Multiply a point on the secp256k1 elliptic curve.

Point 1

x:	ed 115453491402472396292791239385516169732385382997990096508954336517192233241886
y:	ed 45911817369703433493522827989910973041610803449732699181013639391692967496578

Multiplier

ed 103643957297735854668312638519945174297468491926922357240257949688253151834515	+1	Random
---	----	--------

Point 1 x Multiplier

x: ed 988009992411485317652839971088414027508124426434613272716711781535064826849296
y: ed 65884930941482873979258938892141063566762232266292846534114238765618011886858

Steps [hide]

```
point      = x:115453491402472396292791239385516169732385382997990096508954336517192233241886, y:45911817369703433493522827989910973041610803449732699181013639391692967496578
multiplier = 1110010100100110011010011011001110010010101010111000101100011100111001100001101000101010101011110011110001001010111101

double and add = x:85692596131553575982208396689972389292148901579505846788103261224003280916330, y:789465478477091736275960798916999185279994663
double and add = x:97556985993298008716601983816592243059608655932712174953198286128988978814623, y:1839955262278208634741128025933613940006375172
double      = x:96284045483778277323873951212118291492041295996937842955113441753881726473, y:860148278455257779427356484482821628206990881
```

* Implementation Phase: Final Output (no error)

In this step, I used the Random button in the EC Multiply tool to generate a random multiplier, which represents a private key. After clicking Generator Point and then Random, the tool displayed a random number as the multiplier and calculated the corresponding point (x, y) on the elliptic curve. This new point acts as the public key. The private key (multiplier) is a large random number, and the public key is derived from it using elliptic curve multiplication. This process is secure because it is practically impossible to reverse the operation to find the private key from the public key. **Sign:**

ECDSA Sign
Sign the hash [+] of a message using a private key [+].

Random Example

Message Hash (z)
This is typically the hash of some transaction data (that has been prepared for signing)
Hex: fdaee2d483638b8c4f91965bb3eae49014e6fa8b6395537b27bd7f298291c5c9
32 bytes

Nonce (k)
Hex: ee5539b451222e8bc535dbf3e12163799a489c9225f9ac422987db9a54729cb7
Random

Private Key (d)
Hex: 508646ea65a7910769471aac8e65008a2207ce3e7cb5b54c94ffc3cc8e4a841d
32 bytes
Random

Signature
R: 6d 63903189497355943988544874666345530471730716903173559002510300679977108678548
S: ed 5206873888640688785899715197051813146770536037595522050548188359657823871205
R: 120585215348075506637671269113610094706067028261479382332056974781861137621132
S: 5206873888640688785899715197053813146770556037595522050548188359657823871205

In this step, I performed the ECDSA signing process using the online tool. The signing process requires three main inputs: a message hash (z), a private key (d), and a random nonce (k). After entering these values and clicking the Sign button, the tool generated two numbers, r and s, which together form the digital signature. The value of r is calculated from an elliptic curve point derived using the nonce, while s is computed using the message hash, private key, and r. This signature proves ownership of the private key without exposing it and can later be verified with the corresponding public key.

Verify:

In the verification step, I explored how the network or a verifier checks whether a digital signature is valid without needing access to the private key. The process uses three main elements: the original message hash (z), the public key, and the signature values (r and s). When these are provided to the verification tool, it performs several elliptic curve calculations to confirm if the signature corresponds to the given public key. The core idea is that the verifier calculates a specific point on the curve using r, s, and the generator point and then checks if it matches the relationship established during signing. If the calculated point matches the expected value, the signature is considered valid, proving that the signer owns the private key linked to the public key. This step ensures authenticity and integrity in Bitcoin transactions because only the correct private key holder can produce a valid signature, but anyone can verify it using the public key.

* Implementation Phase: Final Output (no error)

ECDSA Verify

Verify a signature [+] using the hash [+] of a message and a public key [+].

Random Example

Message Hash (z)
Ex: 55dae55db5b3759783e37577f833fb92fb2b0d0348544a26af6516edd74b8886
32 bytes

Signature
R: ed 14243687729357601461140652441301064652192805594426970143236533999153723439982
S: ed 41083357558913467665439182163715262038857508745664402753172030598584848353889

Public Key (Q)
Ex: 03df6b25a5044be368545cef129f7bff566aaff04f520a2191eb2113f5a0a16fd7
33 bytes

Signature Verification
x: ed 14243687729357601461140652441301064652192805594426970143236533999153723439982
y: ed 93906148398725603245958626109312510213704041434536951760373227724546048520282

Signature is valid!

* Observations

1. ECDSA uses elliptic curve cryptography to generate a public key from a private key using a one-way multiplication function, making it secure.
2. The signing process creates a digital signature (r, s) that proves ownership without revealing the private key.
3. Verification only requires the public key, message hash, and signature, ensuring the private key remains confidential.
4. Reusing the same nonce during signing can compromise security by exposing the private key, so randomness is critical.

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:

Signature of the Faculty:

Name :

Regn. No. :

Page No.....

* As applicable according to the experiment.

Two sheets per experiment (10-20) to be used.