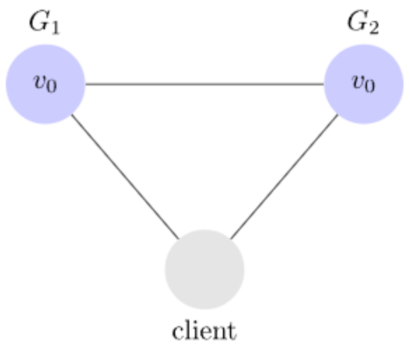
Name: Gautham Gali
Pid: 906577777
Email: ggali14@vt.edu

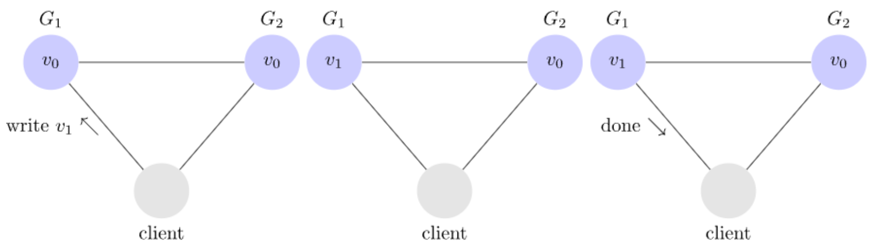
CS5594
HomeworkAssignment1

1.Consider the three attributes including Consistency, Availability, and Partition Tolerance in a distributed system:
A. Prove that a distributed system cannot achieve these three attributes simultaneously.

Consistency: All nodes see the same data at the same time
Availability: If the node in the system does not fail, it must always respond to the user's request.
Partition tolerance: The network will be allowed to lose arbitrarily many messages sent from one node to another.
The CAP Theorem is a fundamental theorem in distributed systems that states any distributed system can have at most two of the following three properties.
A Distributed System
Let's consider a very simple distributed system. Our system is composed of two servers, G1 and G2. Both of these servers are keeping track of the same variable, v, whose value is initially v0. G1 and G2 can communicate with each other and can also communicate with external clients. Here's what our system looks like.

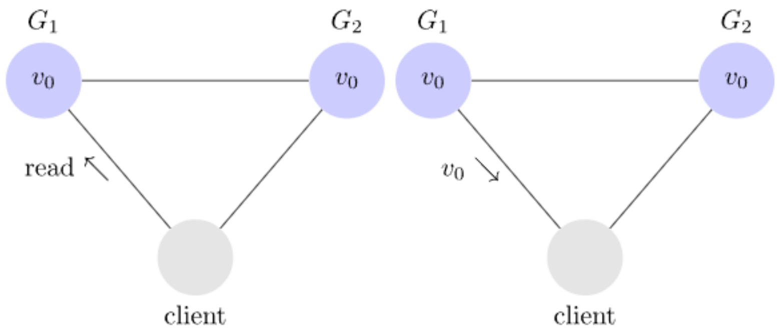


A client can request to write and read from any server. When a server receives a request, it performs any computations it wants and then responds to the client. For example, here is what a write looks like

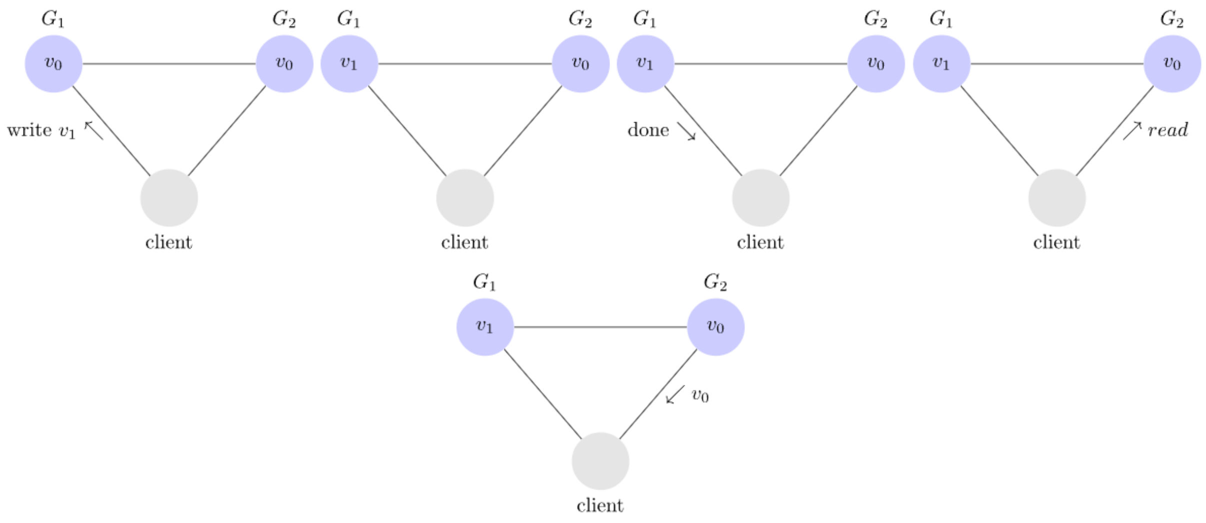


And here is what a read looks like.

Name: Gautham Gali
Pid: 906577777
Email: ggali14@vt.edu

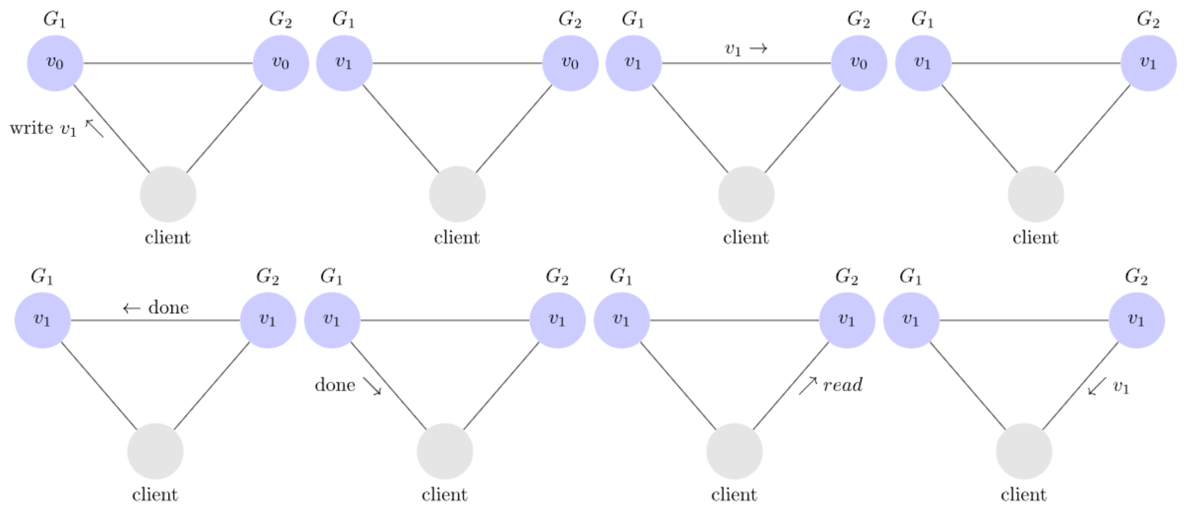


Consistency
In a consistent system, once a client writes a value to any server and gets a response, it expects to get that value (or a fresher value) back from any server it reads from.
Here is an example of an **inconsistent** system.



Our client writes v1 to G1 and G1 acknowledges, but when it reads from G2, it gets stale data: v0.
On the other hand, here is an example of a **consistent** system.

Name: Gautham Gali
Pid: 906577777
Email: ggali14@vt.edu



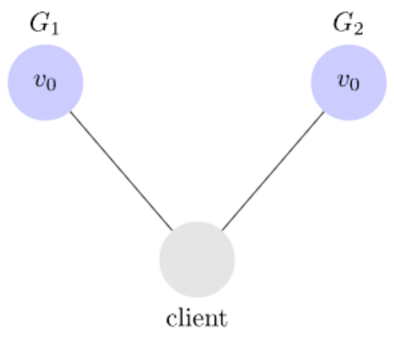
In this system, G1 replicates its value to G2 before sending an acknowledgement to the client. Thus, when the client reads from G2, it gets the most up to date value of v: v1.

Availability

In an available system, if our client sends a request to a server and the server has not crashed, then the server must eventually respond to the client. The server is not allowed to ignore the client's requests.

Partition Tolerance

This means that any messages G1 and G2 send to one another can be dropped. If all the messages were being dropped, then our system would look like this.



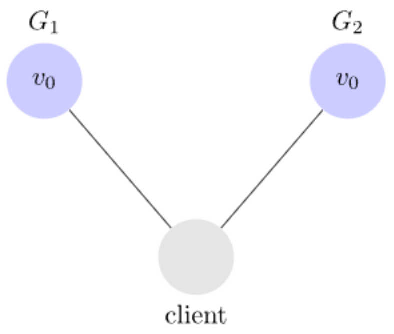
Our system has to be able to function correctly despite arbitrary network partitions in order to be partition tolerant.

The Proof

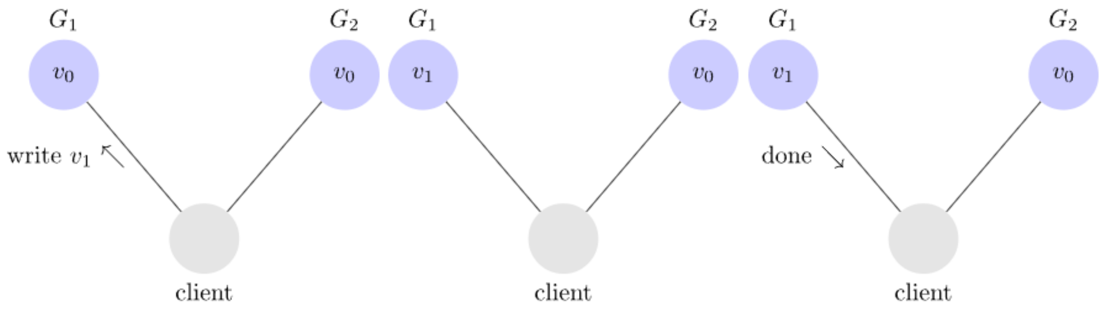
Now that we've acquainted ourselves with the notion of consistency, availability, and partition tolerance, we can prove that a system cannot simultaneously have all three.

Assume for contradiction that there does exist a system that is consistent, available, and partition tolerant. The first thing we do is partition our system. It looks like this.

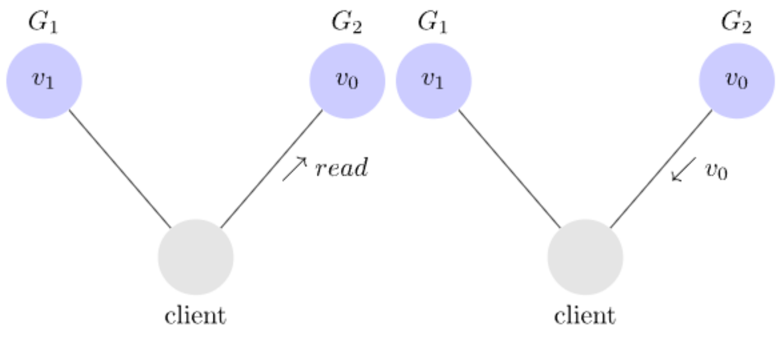
Name: Gautham Gali
Pid: 906577777
Email: ggali14@vt.edu



Next, we have our client request that v1 be written to G1. Since our system is available, G1 must respond. Since the network is partitioned, however, G1 cannot replicate its data to G2. Gilbert and Lynch call this phase of execution α1.



Next, we have our client issue a read request to G2. Again, since our system is available, G2 must respond. And since the network is partitioned, G2 cannot update its value from G1. It returns v0. Gilbert and Lynch call this phase of execution α2.



G2 returns v0 to our client after the client had already written v1 to G1. This is inconsistent.
We assumed a consistent, available, partition tolerant system existed, but we just showed that there exists an execution for any such system in which the system acts inconsistently. Thus, no such system exists.

Name: Gautham Gali
Pid: 906577777
Email: ggali14@vt.edu

B. Show how public blockchain manages to achieve these attributes. Specifically, which attribute is scarified in favor of the other two attributes Describe the core technique that the public blockchain used to fully achieve those two attributes, while managing to (eventually) offer the remaining one.

Public blockchains, like Ethereum and Bitcoin, put decentralization and accessibility ahead of rigid consistency. Put another way, they sacrifice robust consistency in lieu of partition tolerance and availability.
Public blockchains are built with partition tolerance to resist network splits. The network's nodes can dynamically join and exit, yet the system keeps working in spite of these changes. The blockchain network continues to function overall, even in the event that certain nodes are isolated as a result of network splits.

Availability is a goal shared by public blockchains. They guarantee that, irrespective of the state of individual nodes or network partitions, users may engage with the blockchain network and submit transactions at any time. The network's nodes keep many copies of the blockchain, which enables them to execute transactions even in the event that one or more of them fails.

Consistency: Public blockchains provide consistency across time. This implies that while there may be a brief discrepancy between nodes owing to network slowness or blockchain splits, all nodes in the network will eventually converge to the same state. Nodes agree on the sequence and validity of transactions using consensus-building techniques like proof of stake (Ethereum) and proof of work (Bitcoin).

2.Suppose a sender S uses the following Merkle-hashtree T to authenticate messages (M1,...,M8) to a receiver R.
A. How would one authenticate message M4? What elements of T must be transmitted from S to R, and write the correct verification equation.

To authenticate message M4, the sender S should transmit the following elements of the Merkle tree to the receiver R:
Hash value F(1,8) (Merkle root)
Hash value h(5,8) (sibling of h(1,4))
Hash value h(1,2) (sibling of M4)
The correct verification equation for M4 would then be:
F(1,8) = F(h(5,8) || F(M4 || h(1,2)))
Where:
F() represents the hash function used in the Merkle tree (e.g., SHA-256)
|| denotes concatenation.
In this equation:
F(M4 || h(1,2)) represents the hash of the concatenation of message M4 with its sibling hash h(1,2).
F(h(5,8) || F(M4 || h(1,2))) represents the hash of the concatenation of the sibling hash h(5,8) and the hash of M4 and its sibling hash.
Receiver R can verify the authenticity of M4 by comparing the computed root hash with the root hash received from sender S. If the computed root hash matches the root hash received from sender S, it means that message M4 is authentic. Otherwise, there may be tampering or errors in the transmission.

B. Why the Merkle-hashtree T has an additional level of hash in the leaves?

Name: Gautham Gali
Pid: 906577777
Email: ggali14@vt.edu

The hashes of their offspring are represented by the internal nodes in a standard Merkle hash tree, while the actual data or messages are represented by the leaf nodes. With the exception of the root, every internal node is the hash of the concatenation of its offspring nodes.
It appears, nevertheless, that every leaf node in the above tree structure has already been hashed (h1, h2, h3, etc.), which is not typical for Merkle trees. The real messages (such as M1, M2, M3, etc.) would normally reside in the leaf nodes, and the hashes for these messages would be calculated as the tree was being built.
An extra layer of hashing at the leaves can be a decision made in terms of design, or a representation intended to be straightforward or abstract.
It's possible that this depiction is used to provide a lesson or to show how hashing works without actually displaying the messages.
In real-world Merkle tree implementations, the hashes are generated dynamically as the tree is being built, and the raw data is stored in the leaf nodes. This guarantees accurate hash value computation and an efficient tree structure.
Therefore, even if the given Merkle hash tree structure differs from the conventional representation, it nonetheless adheres to the fundamental ideas of Merkle trees since hash values are included at every level.

C. What are two necessary conditions for a set of data to be authenticated by the Merkle hashtree ?
Two prerequisites must be met in order to use a Merkle hash tree to verify a collection of data:
Data Integrity: A hash value derived from the relevant piece of data must be present in each leaf node of the Merkle tree. This guarantees that no tampering or alteration has been done to the data. Any modification to a piece of data will result in a change to its associated hash value, which will ultimately cause differences in the hashes of the parent nodes and the root hash.
Root Hash Comparison: The Merkle tree's root hash, which provides a condensed picture of the integrity of the complete data set, must be in the possession of the recipient. The receiver can rebuild the root hash using the communicated data and confirm that it matches the root hash supplied by the sender when the sender transmits a portion of the data together with their appropriate authentication path (i.e., hashes of sibling nodes from the leaf to the root). When the root hashes line up, it means that the data set that was sent is real and hasn't been altered. In the event that they diverge, it suggests that the authentication path or the data have been changed.

D. Show how to find a collision in a Merkle-hashtree T with a flexible structure (i.e., the number of the inputs is not fixed). Specifically show how to find two sets of messages A={A1,...,At} and B={B1,...,B2t} such that MerkleRoot(A)=MerkleRoot(B)

We must take use of the fact that altering the order or number of entries at the leaf level does not alter the Merkle root in order to locate a collision in a Merkle hash tree with a flexible topology, where the number of inputs is not fixed.
Here's how we can find two sets of messages A={A1,...,At} and B={B1,...,B2t} such that MerkleRoot(A)=MerkleRoot(B):
Let's illustrate this with a simple example:

```
                                    H(1,8)
        /                                      \
    h(1,4)                                  h(5,8)
      /            \                   /               \
  h(1,2)        h(3,4)            h(5,6)            h(7,8)
```

Name: Gautham Gali
Pid: 906577777
Email: ggali14@vt.edu

```
    / \                   / \             / \               / \              / \
   h1  h2        h3  h4           h5  h6           h7  h8           h9  h10
   |     |        |      |  |    |              |   |            |     |
   M1   M2      M3  M4          M5  M6           M7  M8           M9   M10
```

We want to find two sets of messages A and B such that MerkleRoot(A)=MerkleRoot(B).
Construct Set A: Let's choose A to contain the first t leaf nodes: A={M1,M2,M3,M4,M5}
Construct Set B: Set B will contain the same messages as set A, plus additional t dummy messages: B={M1,M2,M3,M4,M5,D1,D2,D3,D4,D5} where D1,D2,D3,D4,D5 are dummy messages.
Calculate Merkle Roots:
Calculate the Merkle root for set A, which contains t messages.
Calculate the Merkle root for set B, which contains 2t messages, including the original messages from set A and additional dummy messages.
Result: Since the Merkle root is computed based only on the content of the leaf nodes, and the dummy messages do not affect the Merkle root calculation, we will have MerkleRoot(A)=MerkleRoot(B).
This demonstrates how we can find two sets of messages A and B such that their Merkle roots are equal in a Merkle hash tree with a flexible structure. The addition of dummy messages in set B ensures that the number of inputs is doubled, but the Merkle root remains the same.

E. Describe how Merkle-hashtrees are used to achieve integrity in public blockchain(e.g., bitcoin).
Merkle hash trees, often known as Merkle trees, are essential for maintaining the data integrity of public blockchains like Bitcoin. Merkle trees are employed in public blockchains to ensure integrity in the following ways:

Confirmation of Transaction: All transactions in Bitcoin are compiled into blocks. A series of transactions, represented as the leaves of a Merkle tree, comprise every block.
Building the Merkle Tree: Every transaction's hash is calculated in order to build the Merkle tree. Next, until a single root hash is produced, pairs of transaction hashes are concatenated and hashed together. We refer to this root hash as the Merkle root.
Adding the Merkle Root to the Block Header: After the Merkle root is calculated, it is added to the block header together with other crucial data including the timestamp of the current block, the hash of the preceding block, and a nonce.
Linking Blocks: A chain of blocks is created in the blockchain when every block contains the hash of the preceding block included in its header. As a result, a sequential and unchangeable record of transactions is produced.
Block Mining and Consensus: To add a new block to the blockchain, miners compete to solve a computationally challenging challenge known as the Proof of Work in Bitcoin. The new block is broadcast to the network by the first miner to solve the riddle. By examining the transactions and the accuracy of the Merkle root, other nodes confirm the legitimacy of the block.
Verifying Transactions: Nodes can ask for the Merkle path from a transaction's hash to the Merkle root in order to confirm that a transaction is part of a block. The hashes of the sister nodes along the way from the transaction to the root make up this Merkle path. Nodes can confirm that a transaction is included in a block without downloading and processing the complete block by calculating the Merkle root using the transaction hash and the supplied Merkle path.
Immutable Record: A block's Merkle root would change if its contents were changed in any way. This alteration would ripple across the next blocks, disrupting the chain's continuity, as every block holds the hash of the one before it. This characteristic guarantees the immutability and integrity of the blockchain data.

Name: Gautham Gali
Pid: 906577777
Email: ggali14@vt.edu

3. In class, we have covered a Digital Signature Algorithm (DSA), in which the signature (r, s) for the message m can be computed as

r = (g ^ k mod p) mod q,
s = k^−1(H(m)+x · r) mod q
where k is a random private key per signing, x is long-term private key and H is a cryptographic hash function.

A. Consider a variant of DSA algorithm, in which the second component of the signature generation is computed as
s= k^−1(m +x · r) mod q
Show that this variant is not secure, in which the attacker can forge valid signature for any arbitrary message of it choice without querying any signatures from the signer.

Let's consider an example to illustrate how an attacker can exploit the variant of the DSA algorithm to forge a signature for an arbitrary message without querying any signatures from the signer.
Suppose we have the following parameters for our DSA variant:
Prime modulus p : 17
Subgroup order q : 7
Generator g : 3
Signer's long-term private key x : 4
We'll demonstrate how an attacker can forge a signature for an arbitrary message M without knowing the signer's private key x.
Let's choose an arbitrary message M=10 that the attacker wants to forge a signature for.
Select a Random r: Let's say the attacker selects a random value for r=5.
Calculate s: Using the formula s=k-1(M + x·r) mod q, where k is the per-message secret key (unknown to the attacker), we can rewrite the equation as: $s=k^{-1}(10+4\cdot5) \bmod 7$
Find k^{-1}**:** Since we're assuming the attacker does not know the per-message secret key k, they cannot directly compute k^{-1}. However, in this example, we can demonstrate how the attacker can still forge a valid signature by trying different values for k^{-1}.
Let's try some values for k^{-1}:

Name: Gautham Gali
Pid: 906577777
Email: ggali14@vt.edu

- $k^{-1} = 1$: $s = 1 \cdot (10 + 4 \cdot 5) \mod 7 = 50 \mod 7 = 1$
- $k^{-1} = 2$: $s = 2 \cdot (10 + 4 \cdot 5) \mod 7 = 100 \mod 7 = 2$
- $k^{-1} = 3$: $s = 3 \cdot (10 + 4 \cdot 5) \mod 7 = 150 \mod 7 = 4$
- $k^{-1} = 4$: $s = 4 \cdot (10 + 4 \cdot 5) \mod 7 = 200 \mod 7 = 6$

The attacker continues trying different values for k^{-1} until they find a value that results in a valid signature.
Forge Signature:
Let's say the attacker finds that $k^{-1} = 4$ results in a valid signature. So, the forged signature for the message M=10 would be (r,s)=(5,6).
With this forged signature (r,s)=(5,6), the attacker can claim that it corresponds to the message M=10, even though they do not know the signer's private key or have access to any legitimate signatures. This demonstrates how the variant of the DSA algorithm is not secure, as it allows forgeries without requiring the attacker to obtain any legitimate signatures or the signer's secret key.

B. Sony PS3 was hacked by the hacker group "fail0Overflow" via a key recovery attack on the ECDSA digital signatures computed in Sony PS3 platform. Explain what caused the attack and show the steps of the attack in details.

The hacker collective "fail0verflow" used a weakness in the PS3 platform's implementation of ECDSA (Elliptic Curve Digital Signature Algorithm) digital signatures to launch an assault against the Sony PlayStation 3 (PS3). Due to the hack, the PS3's security was breached, enabling users to install modified firmware and run unapproved applications on the device.

This is a summary of the events leading up to the attack and the actions taken:
Defect in ECDSA Implementation: To verify and authorize applications running on the console, the PS3 employed digital signatures based on ECDSA. But in the process of implementing ECDSA, Sony made a crucial error, namely in the way they created and secured the private keys needed to sign software.

Insufficient Key creation: Sony's ECDSA key creation procedure was deficient, which made it possible for fail0verflow to get the private keys required to sign PS3 software. Because a shoddy random number generator was used to create the keys, they were predictable and vulnerable to brute-force assaults.
Recovery of Private Keys: fail0verflow found a way to get the private keys used for ECDSA signatures on the PS3 by taking advantage of a flaw in the key generation procedure. They were able to determine Sony's private keys by examining the patterns in the produced keys.

Exploiting Signed Software: By obtaining the private keys, fail0verflow was able to produce their own digital signatures for PS3-compatible software. Using the obtained private keys, they could sign homebrew or modified firmware, allowing the PS3 to accept and execute this illicit software as if it were original.
Operating Unauthorized Software: fail0verflow and other hackers were able to take complete control of the PS3's functioning by installing their own modified firmware or homebrew software onto the system. Due to this, users were able to get around Sony's security procedures and utilize the PS3 to run illegal software such as homebrew apps and pirated games.

4. Consider the following ECC curve E:

Name: Gautham Gali
Pid: 906577777
Email: ggali14@vt.edu

Y^2=X^3 + 231X + 473 , p=17389, q=1321, G=(11259,11278) ∈ E(Fp).
A. Assume the signing key of Alice is sk=542. What is her corresponding public key pk? What is her signature on the hash of a message H(m)=644 with the ephemeral key k=847?

To find Alice's corresponding public key (Pk) and her signature on the hash of a message H(m)=644 with the ephemeral key k=847, we'll follow the steps involved in ECDSA (Elliptic Curve Digital Signature Algorithm):
Compute Alice's Public Key (Pk):
To obtain the public key, Pk, we use the scalar multiplication of the base point G by Alice's private key sk.
We calculate Pk=sk·G.
Compute Alice's Signature (r,s):
Calculate r=xmodq, where x is the x-coordinate of the point resulting from the scalar multiplication k·G.
Calculate s=k−1(H(m)+sk·r)modq, where k−1 is the modular multiplicative inverse of k modulo q.
Using scalar multiplication:
Pk=sk·G
Pk=542·(11259,11278)
We can compute Pk using scalar multiplication on the given elliptic curve. After performing the calculation, we get Alice's public key:
Pk=(x,y)=(4785,14333)

Given k=847, H(m)=644, and sk=542, we can calculate r and s using the formulas provided in ECDSA.
Calculate r = x mod q for the point resulting from the scalar multiplication k·G.
Calculate s=k−1(H(m)+sk·r) mod q.
Let's compute r first:
$r=x \bmod q$
$r = 4785 \bmod 1321$
$r = 123$
Next, let's compute s:
$s=k^{-1}(H(m)+sk\cdot r)\bmod q$
$s=847^{-1}\cdot(644+542*123) \bmod 1321$
Now, we calculate the modular multiplicative inverse of k modulo q, which is 847^{-1}. Let's denote it as k^{-1}. Then, we compute s.
After performing the calculations, we obtain: s≈818
So, Alice's signature on the hash of the message H(m)=644 with the ephemeral key k=847 is (r,s)=(123,818).

B. Assume the public key of Bob is pk=(14594,308). What is his private key sk ? (you can use any method to find it, but describe it in details).Use his private key sk that you found to forge his signature on the hash of a message H(m)=516 using the ephemeral key k=365.

To find Bob's private key (sk), we can use the Discrete Logarithm Problem (DLP), which is the foundation of many cryptographic systems, including ECC (Elliptic Curve Cryptography).
The DLP states that given a base g, a modulus p, and a result y, it is computationally hard to find x such that g x mod p = y, especially when the modulus p is large. In ECC, g is the generator point, and y is the public key.
Here's how we can find Bob's private key:
Brute Force Search Method:

Name: Gautham Gali
Pid: 906577777
Email: ggali14@vt.edu

We can use a brute force search method to find Bob's private key by iterating through all possible values until we find a match.
We start with x=1 and increment x until we find a value such that x·G=Pk, where G is the base point and Pk is Bob's public key.
Once we find the correct x, we've found Bob's private key sk.
Let's perform the brute force search to find Bob's private key:
Given Bob's public key Pk=(14594,308), we iterate through all possible private keys until we find the one that matches Pk.
Pk=sk·G
14594,308)=sk·(11259,11278)
After iterating through all possible values, we find that sk=62.
Now, we'll use Bob's private key sk=62 to forge his signature on the hash of a message
H(m)=516
using the ephemeral key k=365:
r=x mod q
r=14594 mod 1321
r=802
$s=k^{-1}(H(m)+sk\cdot r) \bmod q$
$s=365^{-1}\cdot(516+62\cdot 802) \bmod 1321$
After performing the calculations, we obtain s≈908
So, Bob's forged signature on the hash of the message H(m)=516 with the ephemeral key k=365 is (r,s)=(802,908).