AtSign.py:
The AtSign class provides a way to communicate with the @ protocol servers. The class includes various functions that can be used to authenticate, lookup, update and perform other operations on the @ protocol servers. Functions: authenticate(keys): This function authenticates the atSign with the @ protocol servers using the provided keys. It takes a dictionary of keys as a parameter, and it returns True if the authentication is successful. lookUp(key, location): This function looks up the specified key at the specified location on the @ protocol servers. It takes the key and location as parameters and returns the value associated with the key. plookUp(key, location): This function is similar to lookUp but is used to lookup the public key of the specified location. lLookUp(key): This function is used to lookup the specified key at the AtSign's own location. slookUp(keys, key, location): This function is used to lookup a key at a shared location. It takes a dictionary of keys, the key to lookup, and the shared location as parameters, and returns the decrypted value associated with the key. update(key, value, location): This function updates the specified key with the specified value at the specified location on the @ protocol servers. It takes the key, value, and location as parameters and returns True if the update is successful. publicKeyUpdate(keyShare, location, time): This function updates the public key of the specified location. It takes the key share, location, and time as parameters and returns True if the update is successful. sharedKeyUpdate(keyShare, location, time): This function updates the shared key of the specified location. It takes the key share, location, and time as parameters and returns True if the update is successful. sUpdate(keys, key, value, location): This function updates a key at a shared location. It takes a dictionary of keys, the key to update, the value to update it to, and the shared location as parameters and returns True if the update is successful.

- testAtSignAuthentication: This test case verifies the authentication process of an AtSign object. It loads the keys for a specific atSign, creates an atSign instance, authenticates it using the keys, and asserts that the authentication is successful.
- testMultipleAtSignAuthentication: This test case tests the authentication of multiple AtSign objects. It loads keys for two different atSigns, creates instances of AtSign, authenticates them using their respective keys, and asserts that both authentications are successful.
- testAtSignLocalUpdateAndLocalLookup: This test case checks the functionality of local update and local lookup operations of an AtSign object. It loads keys for an atSign, creates an AtSign instance, authenticates it, performs a local update operation, and verifies that the local lookup operation returns the expected result.
- testAtSignUpdateAndLookup: This test case tests the update and lookup operations between two AtSign objects. It loads keys for two different atSigns, creates instances of AtSign, authenticates them, performs an update operation on one atSign and a lookup operation on the other atSign, and asserts that the lookup result matches the expected value.
- testAtSignUpdateAndLookUpSecure: This test case verifies the secure update and lookup operations between two AtSign objects using secure keys. It loads secure keys for two

different atSigns, creates instances of atSign, authenticates them with their respective keys, performs a secure update operation on one atSign and a secure lookup operation on the other atSign, and asserts that the lookup result is as expected.

AtConnection.py:
The AtConnection class is an abstract base class that provides a common interface and functionality for connecting to and communicating with an @ protocol server. It defines the basic methods and attributes required for establishing a connection, sending and receiving data. Attributes: url: The URL of the connection. host: The hostname of the server. port: The port number to connect to. connected: A boolean indicating whether the connection is currently active. addrInfo: The address information for the server. context: The SSL context for secure connections. _socket: The underlying socket object. secureRootSocket: The secure socket for communication. verbose: A boolean indicating whether to print verbose output. Methods: write(data): Writes the given data to the connection. The data parameter is a string. read(): Reads the response from the connection and returns it as a string. isConnected(): Returns a boolean indicating whether the connection is currently active. connect(): Establishes a connection with the server. It wraps the socket with a secure socket and performs the handshake. disconnect(): Closes the connection. parseRawResponse(rawResponse): Abstract method to parse the raw response received from the server. Subclasses should implement this method. executeCommand(command, retryOnException=0, readTheResponse=True): Executes a command on the server and returns the response. The command parameter is the command to be executed as a string. retryOnException specifies whether to retry the command in case of an exception. readTheResponse indicates whether to read and return the response. Returns the parsed response if readTheResponse is True, otherwise returns an empty string. str(): Returns a string representation of the connection object. init(host, port, context, verbose=False): Initializes the connection with the provided host, port, SSL context, and verbosity setting. It sets up the socket, address information, and other necessary attributes.

atRootConnection.py:
The AtRootConnection class is a subclass of AtConnection and represents a connection to the root server in the @ protocol. It provides methods to establish a connection, send commands, and parse responses specific to the root server. Attributes: __instance: Singleton instance of the AtRootConnection class. Methods: getInstance(host='root.atsign.org', port=64, context=ssl.create_default_context(), verbose=False): Static method that returns the singleton instance of the AtRootConnection class. If the instance does not exist, it creates a new instance with the given host, port, SSL context, and verbosity setting. init(host, port, context, verbose): Initializes the AtRootConnection object with the provided host, port, SSL context, and verbosity setting. It also checks for an existing instance to enforce the singleton pattern. connect(): Overrides the connect() method of the parent class to perform additional actions specific to the

root server connection. It calls the parent's connect() method and prints a success message if the connection is established successfully. parseRawResponse(rawResponse): Overrides the abstract parseRawResponse() method of the parent class to parse the raw response received from the root server. It handles responses that are either 'null' or in the format host:port. It removes any trailing "@" character and returns the parsed response as a string. findSecondary(atSign): Sends a command to the root server to find the secondary server for the given @sign. It returns the response, which is the secondary server address in the format host:port. If the response is 'null', it raises an exception indicating that the @sign was not found. lookupAtSign(atSign): Helper method that calls findSecondary(atSign) and returns the secondary server address as a string. Note: The AtRootConnection class is implemented as a singleton, meaning that only one instance of the class can exist. The getInstance() method is used to retrieve the singleton instance.

- testRootConnection: It creates an instance of AtRootConnection using the getInstance method and sets the verbose parameter to True. It calls the connect method of the connection object to establish a connection. It asserts that the connection is successfully established by checking the isConnected method, which should return True.
- testFindSecondary: It creates an instance of AtRootConnection with verbose set to True. It calls the findSecondary method of the connection object, passing an at sign (@27barracuda) as an argument. It asserts that the returned secondary address is not None, indicating a successful lookup.
- testFindSecondaryFailure: It creates an instance of AtRootConnection with verbose set to True. It attempts to call the findSecondary method with an incorrect at sign (@wrongAtSign). It expects an exception to be raised and asserts that the exception message matches the expected error message.
- testFindMultipleSecondaryAddresses: It creates an instance of AtRootConnection with verbose set to True. It calls the findSecondary method multiple times with different at signs (@27barracuda, @19total67, @wildgreen, @colin). It splits each returned secondary address by the colon (:) and asserts that none of the split parts are None, indicating successful lookup.

These tests help ensure that the AtRootConnection class and its methods are functioning correctly and producing the expected results.

atSecondaryConnection.py:

The AtSecondaryConnection class is a subclass of AtConnection and represents a connection to a secondary server in the @ protocol. It inherits methods from the parent class to establish a connection, send commands, and parse responses. Methods: init(host, port, context=ssl.create_default_context(), verbose=False): Initializes the AtSecondaryConnection object with the provided host, port, SSL context, and verbosity setting. It calls the parent class's constructor to set up the connection. connect(): Overrides the connect() method of the parent class to perform additional actions specific to the secondary server connection. It calls the

parent's connect() method and prints a success message if the connection is established successfully. parseRawResponse(rawResponse): Overrides the abstract parseRawResponse() method of the parent class to parse the raw response received from the secondary server. It removes any trailing "@" character and leading/trailing whitespace from the response and returns it as a string. Note: The AtSecondaryConnection class inherits the executeCommand(), isConnected(), disconnect(), write(), read(), and str() methods from the parent class AtConnection, which are used to execute commands, check the connection status, close the connection, send data, receive data, and get a string representation of the connection, respectively.

- testSecondaryConnection: This test case checks if a secondary connection can be established successfully. It creates an instance of AtSecondaryConnection, connects to the secondary address obtained from the AtRootConnection instance, and asserts that the connection is successfully established.
- testSecondaryConnectionFailure: This test case verifies the handling of connection failures. It attempts to connect to a non-existent address by modifying the secondary address, and expects an exception to be raised with a specific error message.
- testMultipleSecondaryConnections: This test case tests the ability to establish multiple secondary connections. It connects to multiple secondary addresses obtained from the AtRootConnection instance and asserts that each connection object is not None.

  The test cases ensure the correctness and robustness of the AtSecondaryConnection class by covering different scenarios and asserting expected outcomes. The unittest framework is used to run the test cases and report the results. It loads the test cases from the AtSecondaryConnectionTest class, runs them, and prints the test results with verbosity level 2.