Approach

When approaching my client-server model, I decided the easiest way to approach the problem would be to implement a second remote interface on my server, with methods for each stage of the DH key generation process for my client to call. This would allow easier scaling with the correct use of synchronized objects and locks, and save the hassle of sending several messages via sockets, and understanding these messages serverside (whether sent as strings, or serialized objects). A remote object would allow my client to negotiate a key almost autonomously, calling a method for each stage of the process as if the object was local.

I use randomly generated values for g and p, using primes of 256 and 128 bits respectively, and random a and b values between zero and 10,000. These values are then passed by the methods suggest and swap_public, requesting p and g values, the computing x and y values (public keys), which are then swapped between client and server when the client calls swap_ public, prompting both to compute a shared (secret) key. This key is cut down to 128 bits in length and used to encrypt all future messages between client and server, using the Java API's implementation of the AES algorithm. Note that the ciphertext is still sent to the client in its (doubly) encrypted form, as per the algorithm outlined in the spec. The send_message method is then used by the client to request the ciphertext (note that the implementation in my code is specific to the username being transmitted, this could easily be adapted to support several message types by parsing strings). The server then returns the still encrypted ciphertext to the client, where the client decrypts it.

The ECS Server requires an integer be passed in as its key, which would require generating an entirely new DH Key, at much shorter length. Hence the first two bytes of the 16-byte (128 bit) secret key are taken as a key to be passed to the ECS Server. Both the client and server take the first two bytes of their shared / secret key; hence both are aware of the key passed to the server, and it remains secret. The ciphertext is returned in its encrypted form back to the client, where the text is decrypted as per the substitution and transposition algorithm in the spec (having already been decrypted from the AES encryption between client and server; effectively encrypted, and decrypted twice). The client runs a reverse of the substitution twice, then the same for transposition, before stripping trailing Z's.

My approach to multiple clients is for the server to pass into the client a unique client identifier, then store all relevant information for each connection in unique instances of DH_Connection_Server. Each time the client calls a method on the server, this UUID is passed into the method, allowing the server to identify which client is connecting and use the appropriate keys. The DH_Connection_Server class largely contains a variety of getters and setters for each of the connections respective keys, as well as a getCipherText method; called when the client requests the ciphertext from the server. This allows unique connections to store their own data, without the need for synchronized locks. If I were to improve my implementation, I'd add checks to ensure that only one client per UUID is connected at any one time, and warn of multiple clients connecting with the same UUID, as well as investigating identifying clients without the need to pass in an integer every time; as while this isn't a substantial overhead, it is one that could probably be avoided.

Ciphertext:

LOOKINGVERYCALMVERYDIGNIFIEDWITHHISLEGSINTHEAIRCAMEEEYOREFROMBENEATHTHEBRIDGEITSEEYORECRIEDROOTERRIBLYEXCITE
DISTHATSOSAIDEEYOREGETTINGCAUGHTUPBYALITTLEEDDYANDTURNINGSLOWLYROUNDTHREETIMESIWONDEREDIDIDNTKNOWYOUWER
EPLAYINGSAIDROOIMNOTSAIDEEYOREEEYOREWHATAREYOUDOINGTHERESAIDRABBITILLGIVEYOUTHREEGUESSESRABBITDIGGINGHOLESI
NTHEGROUNDWRONGLEAPINGFROMBRANCHTOBRANCHOFAYOUNGOAKTREEWRONGWAITINGFORSOMEBODYTOHELPMEOUTOFTHERIV
ERRIGHTGIVERABBITTIMEANDHELLALWAYSGETTHEANSWERBUTEEYORESAIDPOOHINDISTRESSWHATCANWEIMEANHOWSHALLWEDOYOU
THINKIFWEYESSAIDEEYOREONEOFTHOSEWOULDBEJUSTTHETHINGTHANKYOUPOOHHESGOINGROUNDANDROUNDSAIDROOMUCHIMPRES
SEDANDWHYNOTSAIDEEYORECOLDLYICANSWIMTOOSAIDROOPROUDLYNOTROUNDANDROUNDSAIDEEYOREITSMUCHMOREDIFFICULTIDI
DNTWANTTOCOMESWIMMINGATALLTODAYHEWENTONREVOLVINGSLOWLYBUTIFWHENINIDECIDETOPRACTISEASLIGHTCIRCULARMOVEM
ENTFROMRIGHTTOLEFTORPERHAPSISHOULDSAYHEADDEDASHEGOTINTOANOTHEREDDYFROMLEFTTORIGHTJUSTASITHAPPENSTOOCCURT
OMEITISNOBODYSBUSINESSBUTMYOWNTHEREWASAMOMENTSSILENCEWHILEEVERYBODYTHOUGHTIVEGOTASORTOFIDEASAIDPOOHATL
ASTBUTIDONTSUPPOSEITSAVERYGOODONEIDONTSUPPOSEITISEITHERSAIDEEYOREGOONPOOHSAIDRABBITLETSHAVEITWELLIFWETHREWS
TONESANDTHINGSINTOTHERIVERONONESIDEOFEEYORETHESTONESWOULDMAKEWAVESANDTHEWAVESWOULDWASHHIMTOTHEOTHERS
IDETHATSAVERYGOODIDEASAIDRABBITANDPOOHLOOKEDHAPPYAGAINVERYSAIDEEYOREWHENIWANTTOBEWASHEDPOOHILLLETYOUKNO
W