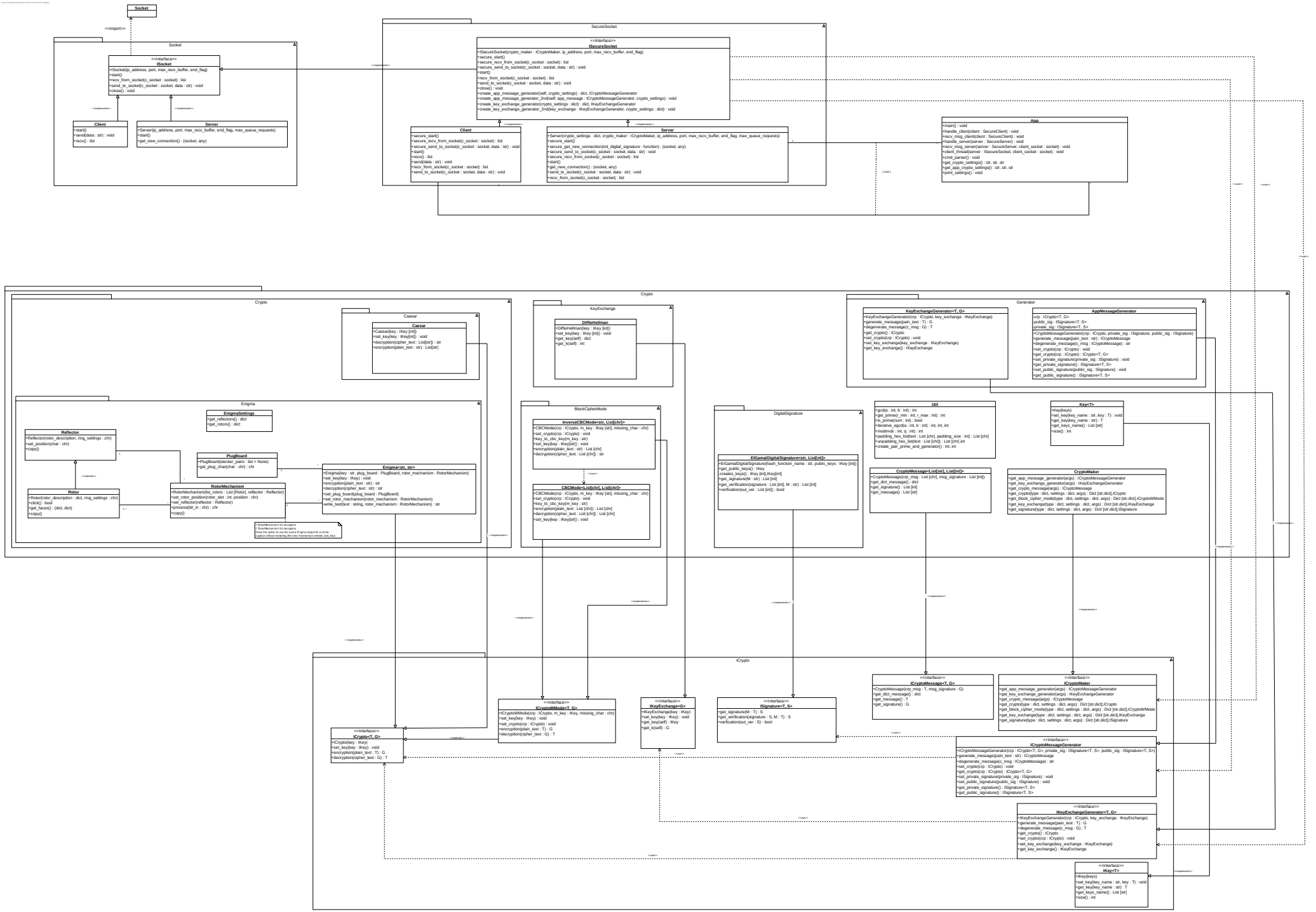
Socket
Socket
<<Interface>>
ISocket
+ISocket(ip_address, port, max_recv_buffer, end_flag)
+start()
+recv_from_socket(c_socket : socket) : list
+send_to_socket(c_socket : socket, data : str) : void
+close() : void
Client
+start()
+send(data : str) : void
+recv() : list
Server
+Server(ip_address, port, max_recv_buffer, end_flag, max_queue_requests)
+start()
+get_new_connection() : (socket, any)
SecureSocket
<<Interface>>
ISecureSocket
+ISecureSocket(crypto_maker : ICryptoMaker, ip_address, port, max_recv_buffer, end_flag)
+secure_start()
+secure_recv_from_socket(c_socket : socket) : list
+secure_send_to_socket(c_socket : socket, data : str) : void
+start()
+recv_from_socket(c_socket : socket) : list
+send_to_socket(c_socket : socket, data : str) : void
+close() : void
+create_app_message_generator(self, crypto_settings) : dict, ICryptoMessageGenerator
+create_app_message_generator_2nd(self, app_message : ICryptoMessageGenerator, crypto_settings) : void
+create_key_exchange_generator(crypto_settings : dict) : dict, IKeyExchangeGenerator
+create_key_exchange_generator_2nd(key_exchange : IKeyExchangeGenerator, crypto_settings : dict) : void
Client
+secure_start()
+secure_recv_from_socket(c_socket : socket) : list
+secure_send_to_socket(c_socket : socket, data : str) : void
+start()
+recv() : list
+send(data : str) : void
+recv_from_socket(c_socket : socket) : list
+send_to_socket(c_socket : socket, data : str) : void
Server
+Server(crypto_settings : dict, crypto_maker : ICryptoMaker, ip_address, port, max_recv_buffer, end_flag, max_queue_requests)
+secure_start()
+secure_get_new_connection(init_digital_signature : function) : (socket, any)
+secure_send_to_socket(c_socket : socket, data : str) : void
+secure_recv_from_socket(c_socket : socket) : list
+start()
+get_new_connection() : (socket, any)
+send_to_socket(c_socket : socket, data : str) : void
+recv_from_socket(c_socket : socket) : list
App
+main() : void
+handle_client(client : SecureClient) : void
+recv_msg_client(client : SecureClient) : void
+handle_server(server : SecureServer) : void
+recv_msg_server(server : SecureServer, client_socket : socket) : void
+client_thread(server : ISecureSocket, client_socket : socket) : void
+cmd_parse() : void
+get_crypto_settings() : str, str, str
+get_app_crypto_settings() : str, str, str
+print_settings() : void
Crypto
Crypto
Caesar
Caesar
+Caesar(key : IKey [int])
+set_key(key : IKey[int]) : void
+encryption(cipher_text : List[str]) : str
+encryption(plain_text : str) : List[str]
KeyExchange
DiffieHellman
+DiffieHellman(key : IKey [int])
+set_key(key : IKey [int]) : void
+get_key(self) : dict
+get_k(self) : int
Generator
KeyExchangeGenerator<T, G>
+KeyExchangeGenerator(crp : ICrypto, key_exchange : IKeyExchange)
+generate_message(pain_text : T) : G
+degenerate_message(c_msg : G) : T
+get_crypto() : ICrypto
+set_crypto(crp : ICrypto) : void
+set_key_exchange(key_exchange : IKeyExchange)
+get_key_exchange() : IKeyExchange
AppMessageGenerator
-crp : ICrypto<T, G>
-public_sig : ISignature<T, S>
-private_sig : ISignature<T, S>
+CryptoMessageGenerator(crp : ICrypto, private_sig : ISignature, public_sig : ISignature)
+generate_message(pain_text : str) : ICryptoMessage
+degenerate_message(c_msg : ICryptoMessage) : str
+set_crypto(crp : ICrypto) : void
+get_crypto(crp : ICrypto) : ICrypto<T, G>
+set_private_signature(private_sig : ISignature) : void
+get_private_signature() : ISignature<T, S>
+set_public_signature(public_sig : ISignature) : void
+get_public_signature() : ISignature<T, S>
Enigma
EnigmaSettings
+get_reflectors() : dict
+get_rotors() : dict
Reflector
+Reflector(rotor_description, ring_settings : chr)
+set_position(char : chr)
+copy()
PlugBoard
+PlugBoard(stacker_pairs : list = None)
+get_plug_char(char : chr) : chr
Rotor
+Rotor(rotor_description : dict, ring_settings : chr)
+click() : bool
+get_faces() : (dict, dict)
+copy()
RotorMechanism
+RotorMechanism(list_rotors : List [Rotor], reflector : Reflector)
+set_rotor_position(rotor_slot : int, position : chr)
+set_reflector(reflector : Reflector)
+process(bit_in : chr) : chr
+copy()
Enigma<str, str>
+Enigma(key : str, plug_board : PlugBoard, rotor_mechanism : RotorMechanism)
+set_key(key : IKey) : void
+encryption(plain_text : str) : str
+decryption(cipher_text : str) : str
+set_plug_board(plug_board : PlugBoard)
+set_rotor_mechanism(rotor_mechanism : RotorMechanism)
+write_text(text : string, rotor_mechanism : RotorMechanism) : str
1 : RotorMechanism for encryption
1 RotorMechanism for decryption
Does the option to use the same Enigma object for a circle
Enigma without rewiring the rotor mechanism wheels (set_key).
BlockCipherMode
InverseCBCMode<str, List[chr]>
+CBCMode(crp : ICrypto, m_key : IKey [str], missing_char : chr)
+set_crypto(crp : ICrypto) : void
+key_to_cbc_key(m_key : str)
+set_key(key : IKey[str]) : void
+encryption(plain_text : str) : List [chr]
+decryption(cipher_text : List [chr]) : str
CBCMode<List[chr], List[chr]>
+CBCMode(crp : ICrypto, m_key : IKey [str], missing_char : chr)
+set_crypto(crp : ICrypto) : void
+key_to_cbc_key(m_key : str)
+encryption(plain_text : List [chr]) : List [chr]
+decryption(cipher_text : List [chr]) : List [chr]
+set_key(key : IKey[str]) : void
DigitalSignature
ElGamalDigitalSignature<str, List[int]>
+ElGamalDigitalSignature(hash_function_name : str, public_keys : IKey [int])
+get_public_keys() : IKey
-creates_keys() : IKey [int],IKey[int]
+get_signature(M : str) : List [int]
+get_verification(signature : List [int], M : str) : List [int]
+verification(out_ver : List [int]) : bool
Util
+gcd(a : int, b : int) : int
+get_primes(r_min : int, r_max : int) : int
+is_prime(num : int) : bool
+iterative_egcd(a : int, b : int) : int, int, int
+modinv(b : int, q : int) : int
+padding_hex_list(text : List [chr], padding_size : int) : List [chr]
+unpadding_hex_list(text : List [chr]) : List [chr],int
+create_pair_prime_and_generator() : int, int
Key<T>
+Key(keys)
+set_key(key_name : str, key : T) : void
+get_key(key_name : str) : T
+get_keys_name() : List [str]
+size() : int
CryptoMessage<List[str], List[int]>
+CryptoMessage(crp_msg : List [chr], msg_signature : List [int])
+get_dict_message() : dict
+get_signature() : List [int]
+get_message() : List [str]
CryptoMaker
+get_app_message_generator(args) : ICryptoMessageGenerator
+get_key_exchange_generator(args) : IKeyExchangeGenerator
+get_crypto_message(args) : ICryptoMessage
+get_crypto(type : dict, settings : dict, args) : Dict [str,dict],ICrypto
+get_block_cipher_mode(type : dict, settings : dict, args) : Dict [str,dict],ICryptoWMode
+get_key_exchange(type : dict, settings : dict, args) : Dict [str,dict],IKeyExchange
+get_signature(type : dict, settings : dict, args) : Dict [str,dict],ISignature
ICrypto
<<Interface>>
ICrypto<T, G>
+ICrypto(key : IKey)
+set_key(key : IKey) : void
+encryption(plain_text : T) : G
+decryption(cipher_text : G) : T
<<Interface>>
ICryptoWMode<T, G>
+ICryptoWMode(crp : ICrypto, m_key : IKey, missing_char : chr)
+set_key(key : IKey) : void
+set_crypto(crp : ICrypto) : void
+encryption(plain_text : T) : G
+decryption(cipher_text : G) : T
<<Interface>>
IKeyExchange<G>
+IKeyExchange(key : IKey)
+set_key(key : IKey) : void
+get_key(self) : IKey
+get_k(self) : G
<<Interface>>
ISignature<T, S>
+get_signature(M : T) : S
+get_verification(signature : S, M : T) : S
+verification(out_ver : S) : bool
<<Interface>>
ICryptoMessage<T, G>
+CryptoMessage(crp_msg : T, msg_signature : G)
+get_dict_message() : dict
+get_message() : T
+get_signature() : G
<<Interface>>
ICryptoMaker
+get_app_message_generator(args) : ICryptoMessageGenerator
+get_key_exchange_generator(args) : IKeyExchangeGenerator
+get_crypto_message(args) : ICryptoMessage
+get_crypto(type : dict, settings : dict, args) : Dict [str,dict],ICrypto
+get_block_cipher_mode(type : dict, settings : dict, args) : Dict [str,dict],ICryptoWMode
+get_key_exchange(type : dict, settings : dict, args) : Dict [str,dict],IKeyExchange
+get_signature(type : dict, settings : dict, args) : Dict [str,dict],ISignature
<<Interface>>
ICryptoMessageGenerator
+ICryptoMessageGenerator(crp : ICrypto<T, G>, private_sig : ISignature<T, S>, public_sig : ISignature<T, S>)
+generate_message(pain_text : str) : ICryptoMessage
+degenerate_message(c_msg : ICryptoMessage) : str
+set_crypto(crp : ICrypto) : void
+get_crypto(crp : ICrypto) : ICrypto<T, G>
+set_private_signature(private_sig : ISignature) : void
+set_public_signature(public_sig : ISignature) : void
+get_private_signature() : ISignature<T, S>
+get_public_signature() : ISignature<T, S>
<<Interface>>
IKeyExchangeGenerator<T, G>
+IKeyExchangeGenerator(crp : ICrypto, key_exchange : IKeyExchange)
+generate_message(pain_text : T) : G
+degenerate_message(c_msg : G) : T
+get_crypto() : ICrypto
+set_crypto(crp : ICrypto) : void
+set_key_exchange(key_exchange : IKeyExchange)
+get_key_exchange() : IKeyExchange
<<Interface>>
IKey<T>
+IKey(keys)
+set_key(key_name : str, key : T) : void
+get_key(key_name : str) : T
+get_keys_name() : List [str]
+size() : int
<<implements>>
<<create>>
[JSON]