Automatically Learning a Model of the SSH2 Transport Layer

Author: Mirjam van Nahmen
3004120
m.vannahmen@student.ru.nl

Supervisor: Erik Poll, Fides Aarts, Joeri de Ruiter
Institute for Computing and Information Sciences
Radboud University Nijmegen

February 19, 2014
Abstract

This thesis is about experiments to automatically learn a finite state model of the SSH Transport Layer of an implementation of SSH. To be more specifically, the learned model is about the server side of the communication during the Transport Layer.

To apply the learning approach is a specific implementation needed, for this paper is for the OpenSSH implementation chosen. But it is just an example. With the resulting program is it possible to check every SSH server.

The description of the SSH protocol is not always precisely how some steps of the protocol should be implemented. The resulting model gives an overview which steps are possible and which answer the server really returns during the conversation. This can show the differences between the OpenSSH implementation and the formal description.

To learn the model the tool "LearnLib" and a modified SSH implementation JSch is used. LearnLib is a tool for automata learning and can automatically learn a model. With a modified SSH client LearnLib is able to send requests to the server and use the answers of the server to create a model.

The main work of this project was to modify the SSH client "JSch" in such a way that it can be connected with LearnLib. Some problems during this modification will also be shown. Because LearnLib acts as a client parallelism of SSH is left out of this project.
## Contents

1 Introduction ........................................... 3

2 Background ............................................. 5
   2.1 SSH ................................................. 5
      2.1.1 SSH Transport Layer Protocol ............... 5
   2.2 Programs/Tools .................................... 7
      2.2.1 Test harness .................................... 7
      2.2.2 LearnLib ....................................... 8

3 Implementing the test harness ......................... 9
   3.1 Harness ............................................. 9
      3.1.1 Problems ...................................... 9

A Appendix ............................................... 20
   A.1 SSHTestService.java ............................... 20
   A.2 Session.java ....................................... 26
   A.3 NewSession.java .................................... 61
1 Introduction

The SSH protocol is one of the oldest and most used secure communication protocols. There are also many different implementations of the SSH server and clients but it is not always possible to see if they are exactly implemented as in the original definitions (the RFC 4250-54\cite{5}, \cite{6}, \cite{7}, \cite{8} and \cite{9}) and sometimes these definitions are not very exactly. For instance sometimes it is necessary to view debug messages to troubleshoot any SSH connection issue, but in the documentation it is allowed to choose if the implementation ignores a message which is labelled with a SSH\_MSG\_DEBUG, or not.

An exact picture of one implementation of a SSH server can be given by a learned model. In order to build such a model LearnLib is a useful tool, which can use a given alphabet to give a 'System Under Test' (SUT) words as input and receives the response of this SUT. An alphabet contains all possible inputs of the SUT, e.g. connect, "SSH\_MSG\_KEXINIT", "SSH\_MSG\_KEXINIT\_DH", "SSH\_MSG\_NEWKYS".... . A word is then a sequence of the alphabet, e.g. ("SSH\_MSG\_KEXINIT", connect, "SSH\_MSG\_KEXINIT")

With this information is LearnLib able to create a model which is almost exactly the SUT. To create the words LearnLib uses the L*-algorithm.

In case of the project the SUT the SSH server OpenSSH\footnote{http://www.openssh.org/} and with the input alphabet LearnLib is able to control the SSH client which sends requests to the SSH server.

Figure 1: Project structure

Figure 1 shows the structure of this project. On one side is LearnLib which communicates with a Test Harness and the Test Harness communicates with the SSH server. The Test Harness is the modified SSH client. LearnLib uses the input alphabet to say which request should be sent to the server in which order.

Usually the client asks the server to begin a conversation with a connection request followed by a request to get the version number of the server, so that the client can compare the version of the server with the supported versions of the client. The next step of the client should be to ask to begin with a key exchange. But what happens if the client asks directly for a key exchange without a request for the version number?

LearnLib forms almost every possible order of the various requests in form of so called words of the input alphabet. A word can be seen as a name of a request. This word will be sent to the Test Harness. The function of the Test Harness is now to translate the names in the word to the function which sends the associated requests to the server. After sending a request the Test Harness should wait for the responses of the server translates the responses of the server into an understandable word for LearnLib and sends it to LearnLib.

Because this is a Bachelor thesis and to delimit this project, only the Transport Layer will be considered.
The Transport Layer is the bottom layer of the SSH protocol. Moreover SSH and the layer see section 2.1 or the very good German article [3].

Nothing further will be said about a model of the exactly architecture of other layers of SSH. This paper also deals only with the security or correctness of the tested implementation on the Transport Layer.

Chapter 2 gives some background information of SSH and the utilized programs. In the first section SSH is described and some detailed information about the Transport Layer is provide. The second section describes the used programs/tools. Chapter 3 describes which preparation needed for the testing. Chapter 4 will describe how the testing was done for this project. There were some problems to run the tests. And in the last two sections the results and the project will be evaluated and the conclusion of this thesis will be drawn.
2 Background

To understand the next steps in this project it is useful to describe first what SSH is, especially the architecture which is described in the [SSHARCH]. Here it will also be described what the result should look like and which programs/tools were used for this project.

2.1 SSH

SSH is a communication protocol that provides secure login, file transfer and TCP/IP connection through an untrusted network. For the authentication it uses a cryptographic authentication, automatic session encryption and integrity protection for transferred data [10].

SSH is specified in five RFCs: RFCs 4250-4524 [5-8]. In these RFCs the common notation is the overall architecture and three sub-protocols described. The sub-protocols are the transport[SSH-TRANS], authentication[SSH-AUTH] and connection[SSH-CONN] layer. The advantage of this is the modularity of the layers. With this the protocol is very flexible and can independently choose if the direction of the communication will negotiate the methods of the key exchange and the encryption. In the following section the layers will be described shortly. However the cryptographic algorithms won’t be described here, because these will be negotiated.

The following picture shows the protocols and their relation to each other which are used in a login session. The dashed line shows which component starts which other component. The thin solid lines show the communication for the authentication. The thick solid lines show the data and the lines with gray background are the protocols.

2.1.1 SSH Transport Layer Protocol

The RFC 4253 is the important RFC for this paper. It describes the first sub-protocol SSH Transport Layer Protocol. This sub-protocol begins a session, thus in this RFC is described how to set up a connection including creating the session keys, the authentication of the server and it finishes with initialization of the data exchange.

The format of the packets which are used during a session is, with some exceptions the beginning of the Transport Layer Protocol, the so called Binary Packet Protocol. This packet contains one byte for the message number, which determines the type of the message. For the Transport Layer the numbers 1-49
are reserved. The Transport Layer Protocol can also be divided into four levels:

1. the protocol identification phase - which version of SSH is run - SSH1 or SSH2
2. the algorithm negotiation phase - which key exchange algorithm is used
3. the key exchange - do the key exchange
4. the service request phase - initialization of the further sub-protocols.

The sequence of the sending and receiving of the packets can be shown as the following:

![Figure 3: SSH Transport Layer Protocol Packet Exchanges](image)

To establish the connection the client first connects to the server via the port 22. After this, both sends the version string "SSH-\{protocol version\}-\{softwareversion\}<CR><LF>", so that there are no misunderstandings by using different versions.

After the version string the data will be sent in the format of Binary Packets Protocol (BDP) which is specified in the [SSH-TRANS]. The format of the packets as described in [SSH-TRANS] is this:

```
unit32 packet_length
byte padding_length
byte[n1] payload; n1 = packet_length − padding_length − 1
byte[n2] randompadding; n2 = padding_length
byte[m] mac(Message Authentication Code − MAC); m = mac_length
```

The type of the packets are defined of the message numbers. This is defined in the [SSH-NUMBERS]. The important numbers for this project are:
For the key exchange two algorithms can be chosen. The "diffie-hellman-group1-sha1" and  "diffie-hellman-group14-sha1". With this algorithms the Transport Layer guarantees the central security objectives of SSH, the confidentiality and integrity. \[4\] If the encryption algorithm is found every packet will encrypt with it. \[3\]

To encrypt and sign, the protocol can use almost any public key format, encoding and algorithm. Currently defined are "ssh-dss - Raw DSS Key", "ssh-rsa - Raw RSA Key", "pgp-sign-rsa - OpenPGP certificates (RSA key)” and “pgp-sign-dss - OpenPGP certificates (DSS key)” \[5\]

### 2.2 Programs/Tools

#### 2.2.1 Test harness

The main part of the project is to build the Test Harness which has as main function to send requests to and receive the responds of the server. A Test Harness is a thus a SSH client with some special functions.

![Test Harness structure](image)

Figure 4: Test Harness structure

The conditions of such a client must be:

- The program code must be freely available (i.e. Open Source), otherwise it is not possible to adjust the program in the way, that I can determine with the test harness the contents of the packets.

- Because the SSH1 is not the actual used protocol, and had some disadvantages \[2\] the program must use at least SSH2 or both SSH1 and SSH2.

- Because LearnLib is written in Java it makes it easier to learn the model if the client and therefore the harness is also written in Java.

- And important for every client implementation is also that it adheres to the RFCs, which define the SSH. So it is representative of the SSH definition.

Anyone who searched for "SSH Java implementation" in google knows, that there are a lot of programs. Most of them are some small projects without good documentation and/or explicitly designed according
to the definitions in the RFCs. After checking in the descriptions and documentations of a lot of them, there were two favorites that fit the conditions. **MidpSSH** and **JSch**.

**MidpSSH**
MidpSSH is a simple SSH and Telnet client implementation in Java for mobile devices.\(^2\)
It is based on Floyd SSH and Telnet Floyd by Radek Polak.\(^3\)
The main features are:

- SSH1 and Telnet support, and now SSH2 support,
- Macros for entering frequently typed commands,
- Traffic usage reporting,
- Free and Open Source (GPL).

**JSch**
JSch is a pure Java implementation of the SSH client protocol.\(^4\) It is a known and often used implementation, which is also used in many IDEs (Eclipse, Netbeans...). The features of this implementation:

- standardized on IETF Secure Shell working group,
- SSH1 and SSH2 support,
- Free and Open Source (GPL).

In this project JSch is chosen as client implementation because it is an often used and very common library for the SSH communication in programs written in Java.

The following task was to modify the client to get the possibility to control the order of the requests of the client. This is done by splitting methods which control the communication and catch out the responding packets of the server. For more details see 3.1.

### 2.2.2 LearnLib

Finally a program is needed which makes a model out of the information which makes the test harness available. A common tool is LearnLib. This framework is made in a project group of the TU Dortmund for Automata learning.

With automata learning it is possible to construct a finite automate, which matches a goal-automate using observation. LearnLib tries to construct a model by trying out operations (in our case sending different types of SSH packets).

LearnLib gives a framework to use different learn algorithms and analyse their characteristics.\(^5\)

---

\(^2\) [http://www.midpssh.org](http://www.midpssh.org)

\(^3\) [http://phoenix.inf.upol.cz/~polakr/](http://phoenix.inf.upol.cz/~polakr/)

\(^4\) [http://jcraft.com/jsch/](http://jcraft.com/jsch/)

3 Implementing the test harness

3.1 Harness

As described above the test harness should be used to make it possible to send and receive the packets separately which are not in the normal order. It should also give an interface for the model based testing. This interface should contain the functions to send and receive the packets and also to change some settings of the packets.

So remember the figure 3, for every arrow is at least one function is needed. (For instance sendVersion(); and receiveVersion();)(See Harness.java??) But because JSch is a library which uses only the default packet sequence there are no functions to call for only one packet sending or receiving. After a longer and deeper look JSch seems to have two important classes, JSch.java and Session.java. The class Session.java controls the communication. This means that there will be decided which step is next and the needed functions will be called and the needed states will be set. So I made a new class which is similar to the original NewSession.java but with the difference that this class is based on the described interface,(see NewSession.javaA.3). To use this class and interface some changes were needed in the Session.java. First the different steps must be split, so that the sending of the packets can be separately called. This includes also the changing and setting of different states. Because the packets can be sent in a sequence which is not by default, some states must be set by hand to ensure the validity of the packets. Because if the client sends a packet and waits for the answer, the send and receive functions are always in one function call of the Session.java. The output of the functions of the harness should be the message numbers as integer. This is done by hand in the Session.java. Because the packet will be read by a so called socket and put in a buffer, it is very easy to extract the message number from this buffer.

Of course this is not the detailed description of the changes but for a detailed look what had be changed to make this work see (See the java class Session.javaA.2, Harness.java?? and NewSession.javaA.3)

3.1.1 Problems

Problems during the implementation:
In the beginning there were some problems, how to use the client implementation. This is a library and the small documentation is more intended for software developers who use only the already established methods. It can be seen more as a black box for using a SSH connection. But to implement the harness a deeper look is needed. To understand the inside of the library I had to read the code mostly without any documentation or JavaDoc. So it takes really a lot of time to read and understand the code and to get a detailed impression of the data flow.

For the project it was also interesting to look at how they built a packet. In the beginning of the Session.java stand all needed message numbers, so the first idea was to look at these numbers to find the data flow of the packets. But this was not the easiest way. They work not only with these numbers, often boolean or bytes are used to set a state and write a packet on an abstract level. Afterwards the best strategy to find the data flow and the building way was to look first at the boolean and bytes. Also to set the states on the right way was a little bit tricky. Sometimes informations from the old packet are needed, but if this packet is all corrupt, therefore does not contains the right values, is it impossible to get the needed information. It is not possible to compute the key after a corrupt SSH_MSG_KEXINIT, even if the states are set hard code with the values of a default session.
Results

Manual testing

To understand how to make a state machine or a model and to get a first look at the results, here a model is made by hand. Thus this makes it possible to call several steps in different order manually to simulate different situations and note the result of the states. Because it is almost impossible to check every possible combination of requests and this test is only geared to get a first impression only some different request sequences are tested. First to get a reference the common order of requests is sent (1.), after that it is tried to connect request after request (2.-5.), at last it is tried to send every request two times (6.-9.).

These tests are done by hard coding a sequence of requests in the main.java.

The outputs in the console look like this:

Result with the default sequence:
begin
Start...
Connected...
SSH-2.0-OpenSSH_5.9p1 Debian-5ubuntu1.1
Sever Version: 2.0
Good Server version(0) otherwise(-1): 0
KexInit1: SSH_MSG_KEXINIT = 20
KexInitDH: SSH_MSG_KEXDH_REPLY = 31
NewKeys: SSH_MSG_NEWKEYS = 21
SSH_MSG_SERVICE_REQUEST_accepted: SSH_MSG_SERVICE_ACCEPT = 6

This program calls every step in the right sequence so that it simulates a normal beginning of a session. But it is not really easy to read. A commonly used notation to describe the communication between a server and a client is to use arrows. Because we are looking from the client side and speak to the server we use → if we send something to the server and ← if we get a message from the server. The important information in every message is the message number, this is the information which is notated here.

The translated results of my handmade testing:

1. Result with the default sequence:
   → connect
   → VERSION.RECEIVED
   ← SSH_MSG_KEXINIT
   ← SSH_MSG_KEXINIT
   → SSH_MSG_KEXDH_INIT
   ← Kex_DH.Reply
   → SSH_MSG_NEWKEYS
   ← SSH_MSG_NEWKEYS
   → SSH_MSG_SERVICE_REQUEST
   ← SSH_MSG_SERVICE_ACCEPT

2. Result with two times connect after recieving msg number:
   → connect
   → connect
   → VERSION.RECEIVED
   ← SSH_MSG_KEXINIT
3. Results with connect after Kexinit:

→ connect
→ SEND_VERSION
← VERSION RECEIVED
→ SSH_MSG_KEXINIT
← SSH_MSG_KEXINIT
→ connect
→ SEND_VERSION
← VERSION RECEIVED
→ SSH_MSG_KEXINIT
← SSH_MSG_KEXINIT
← SSH_MSG_KEXDH_INIT
← Kex_DHReply
→ SSH_MSG_NEWKEYS
← SSH_MSG_NEWKEYS
→ SSH_MSG_SERVICE_REQUEST
← SSH_MSG_SERVICE_ACCEPT

4. Results with connect after KexInit_DH:

→ connect
→ SEND_VERSION
← VERSION RECEIVED
→ SSH_MSG_KEXINIT
← SSH_MSG_KEXINIT
← SSH_MSG_KEXINIT
← SSH_MSG_KEXDH_INIT
← Kex_DHReply
→ connect
→ SEND_VERSION
← VERSION RECEIVED
→ SSH_MSG_KEXINIT
← SSH_MSG_KEXINIT
← SSH_MSG_KEXDH_INIT
← Kex_DHReply
→ SSH_MSG_NEWKEYS
← SSH_MSG_NEWKEYS
→ SSH_MSG_SERVICE_REQUEST
← SSH_MSG_SERVICE_ACCEPT

5. Results connect after NEW_KEYS:
→ connect
→ SEND_VERSION
← VERSION_RECEIVED
→ SSH_MSG_KEXINIT
← SSH_MSG_KEXINIT
→ SSH_MSG_KEXDH_INIT
← Kex_DH_Reply
→ SSH_MSG_NEWKEYS
← SSH_MSG_NEWKEYS
→ connect
→ SEND_VERSION
← Packet corrupt

6. Result after 2 times kexInit:
→ connect
→ SEND_VERSION
← VERSION_RECEIVED
→ SSH_MSG_KEXINIT
← SSH_MSG_KEXINIT
→ SSH_MSG_KEXINIT
← Packet corrupt

7. Result after 2 times kexinit_DH:
→ connect
→ SEND_VERSION
← VERSION_RECEIVED
→ SSH_MSG_KEXINIT
← SSH_MSG_KEXINIT
→ SSH_MSG_KEXDH_INIT
← Kex_DH_Reply
→ SSH_MSG_KEXDH_INIT
← Packet corrupt

8. Result after 2 times Newkeys:
→ connect
→ SEND_VERSION
← VERSION_RECEIVED
→ SSH_MSG_KEXINIT
← SSH_MSG_KEXINIT
→ SSH_MSG_KEXDH_INIT
← Kex_DH_Reply
→ SSH_MSG_NEWKEYS
← SSH_MSG_NEWKEYS
← SSH_MSG_NEWKEYS
← Packet corrupt

9. Result after 2 times request:
→ connect
→ SEND_VERSION
← VERSION_RECEIVED
→ SSH_MSG_KEXINIT
← SSH_MSG_KEXINIT
→ SSH_MSG_KEXDH_INIT
← Kex_DH_Reply
→ SSH_MSG_NEWKEYS
← SSH_MSG_NEWKEYS
→ SSH_MSG_SERVICE_REQUEST
← SSH_MSG_SERVICE_ACCEPT
→ SSH_MSG_SERVICE_REQUEST
← SSH_MSG_SERVICE_ACCEPT

Now it is possible to set the output manually to a model. It is possible to do this manually via every sequence and set the message numbers or strings as state and the function name as step.

Set together manually it gets this result:

![Diagram of the SSH protocol with manual testing results]

Figure 5: The partial model obtained by manual testing

In some steps the send and received values are together because, there is only the one sequence order possible, otherwise the states are no longer valid. As notation of sent requests of the client the request a has ‘?’; which is comparable with an → in the notation of the results of the manual testing. The ‘!’ is for the other way around which is like the ← above. As one can see here the OpenSSH server goes through the protocol and if a packet from the client is not as expected sends the server a disconnect and closes the session. Only the reconnect, in order to start a new session, is possible at every state, but this is a secure step. And of course the client is allowed to send a SERVICE-REQUEST as often as he wants. But then the keys are defined and negotiated.

This is the hand made model, which allows for the fact that it is not complete. This is a short overview over the standard sequence and what happens if some steps are sent twice. To make this model it took at least three to four hours, thus you need time for this and it contains not every possible step.
Another method could be to test the program with model based testing. During model based testing a system will test if it fulfills the conditions which have been given by a model. The model of the manual testing is not complete enough. It shows only the conditions of the tests cases and there are still a multitude of possible test cases. A better solution is to make the model automatically use a tool which can test all possible test cases. One tool to learn state machines is LearnLib. LearnLib uses a very straight strategy of two steps. First learn a hypothetical model and then test if this model is correct. If it finds a counter example it uses the counter example to learn a new hypothetical model and test it again. This continues until it has the maximum of test done without finding a counter model.

Figure 6: Model made of the RFCs "SSH2 doing Diffie-Hellman key exchange, allowing for guessing of the key exchange algorithm (i.e. KEXDH INIT may be sent before KEXINIT S) and an erroneous initial key exchange packet by the other party (the optional KEX WRONG GUESS S that is ignored)."

Figure 6 shows a model which is made by Erik Poll and Aleksy Schubert of the definition of the SSH, the RFC 4250-4254. Comparing the two models shows, that the default path is correct. This means that a communications in which the usual order of requests work is used, it is the same in both figure 5 and figure 6. Further shows that some steps are not tested at this time. E.g. the step KEXINIT from the state after NEWKEYS. To check every possible state an automatic way should be better.
Automatically Testing

As said in the introduction, LearnLib is instead to sends all possible combinations of steps in the form of an alphabet to the Harness and uses the results to make a model.

LearnLib

LearnLib is a tool to automatically make a model in form of a finite state machine. As mentioned in the end of the section about the hand made model, it uses a two step strategy. First learn a hypothetical model and then test if it is correct.

LearnLib communicates with a so called alphabet, these are words which can be sent to the system under test. But the Test Harness understands only function calls. Thus the alphabet of LearnLib must be translated to the method calls of the Test Harness. This is the task of the Wrapper. The Wrapper receives the words of LearnLib and calls the right methods. The Wrapper also receives the responses of the Test Harness and translates them to understandable words for LearnLib.

Wrapper

The wrapper forms the connection between the system under test, the SSH communication, and LearnLib. Therefore it connects to the communication socket and translates the input from LearnLib. LearnLib always gets an alphabet as input. The alphabet contains all possible names for the requests, e.g. "SSH_MSG_KEXINIT", "SSH_MSG_KEXINIT_DH", "SSH_MSG_NEWKYS",... With this alphabet LearnLib forms "words" for the system under test understands. "Words" mean a sequence of inputs. In our case the system gets only an order of function calls, e.g. ("SSH_MSG_KEXINIT", connect, "SSH_MSG_KEXINIT"). LearnLib understands only Strings, so for every request is the name of the request chosen. After some test it seems that LearnLib has some problems with underscores '_', therefore it is chosen to write the names without them. So one of the tasks of the wrapper is to connect the names to the function calls.

Results

The resulting model of LearnLib looks like:

![Diagram of connection between LearnLib and the SSH client](image)
s0 kexinit/NoConnection kexinitDH/NoConnection newKey/NoConnection version/NoConnection
connect/connected
s1 kexinit/recieveVersionnr kexinitDH/InvalidProtocol newKey/InvalidProtocol version/NoConnection
connect/connected
s2 version/VersionOK kexinitDH/recieveVersionnr
s3 connect/connected
s4 kexinit/KEXINIT
s5 newKey/KEXINIT kexinit/recieveVersionnr kexinitDH/recieveVersionnr newKey/InvalidProtocol version/VersionOK
connect/connected
s6 kexinit/NEWKEYS kexinitDH/NEWKEYS version/ConnectionClosed
connect/connected
newKey/NEWKEYS
The learned model shows that the default path works fine.

\[
\begin{align*}
s_0 & \rightarrow \text{connect/connected} \\
s_1 & \rightarrow \text{version/versionOK} \\
s_2 & \rightarrow \text{kexinit/KEXINIT} \\
s_4 & \rightarrow \text{kexinitDH/KEXDHREPLY} \\
s_6 & \rightarrow \text{newkeys/NEWKEYS} \\
s_5 & \\
\end{align*}
\]

At every state every possible request is tested. This is an indication that the model is fully tested. Here fully means that only the requests of the transport layer are in the alphabet used. But it is not only possible to reach the end state \( s_5 \) via the default path; it can also be reached as follows:

\[
\begin{align*}
s_0 & \rightarrow \text{connect/connected} \\
s_1 & \rightarrow \text{version/versionOK} \\
s_2 & \rightarrow \text{newkeys/KEXINIT} \\
s_5 & \\
\end{align*}
\]

This happens because only the SSH Transport Layer is tested and not the entire SSH. A manual test showed, that it is not possible to go further than to state \( s_5 \).

Other than that there is nothing very surprising in the model. If a connect is sent after every state it is possible to continue with a version number otherwise, if a wrong request is sent it goes directly to the begin state or first to an error state and then to the begin state.

During the automatically testing also some problems arose. The main problem was that OpenSSH is seen as BlackBox, which means that there were some unknown behaviour. E.g. after more than 20 connection requests the server blocks the requests. This happens because OpenSSH disconnects after \( N \) time requests sending or don’t send a request in \( M \) seconds. LearnLib has a big problem with this behaviour. Because the responds of the server becomes non-deterministic and for LearnLib it is not possible to make a non-deterministic automata. A solution was to restarts the server during every reset of LearnLib.
Conclusion

The results show that the Transport layer of the OpenSSH server works well. This is not a really surprising result, but good to know it for sure. Now it is clear that the server works at that part precisely on the RFCs which are the definitions of the SSH.

Evaluation OpenSSH

But still were the tests only on the transport layer protocol, thus they say nothing about the other two sub protocols or the hole communication. Also is parallelism not considered. In the result model can be seen that the OpenSSH server goes through the protocol from state s0 to state s1 to the next state s2 and to state s4 and then to s5 and at the end to s6. But if the client sends during this communication a packet which is not as expected, sends the server an error request and close the session. This means, that the client must again begin in state s1. Only the reconnect, thus to start a new session, is at every state possible, but this is a secure step, because it leaves the encrypted state and go through an plain state. And of course the client is allowed to send a SERVICE-REQUEST as many as he wants. But then are the keys defined and negotiated and the data are already encrypted. But we can based on results in the section before conclude, that the OpenSSH Transport Layer Protocol is pretty correct through the definitions in the [SSH-TANS].

Evaluation Project

For this project the client implementation JSch was a bad choice. The implementation of the session and how the packets were made is not really useful to make the Harness that I wanted. Also the documentation was really not meant for further programming/reprogramming. Because it contains almost no JavaDoc and the other documentation is only for users of the library. A better choice had been to write an own program from scratch, which contains the needed functions of a client. This approach would perhaps need only less than the half of the code which is now used for the client. Also cost it not so much time to understand the packet building, sending and receiving. Which would reduce the needed time also a lot.

Future work

In this project it is only the transport layer protocol tested. A further work would be to test what happens if also requests of the other layers would be tried during the transport layer session. The RFCs over the other layers of the SSH protocol are also not always very clear. It could also be interesting to test these layers with the same sort of model based tests, to show how the server react at every state if he receives not the expected packet.
Maybe it could also be interesting to look what happens if packets will be to the server send, which have not the expected structure. E.g. a packet with a request number for a data packet, but after this it sends only a version string and not the expected data length and data.
The resulting program (Harness and wrapper) can also used to test other SSH server. With this it can be possible to find fingerprints of different servers and thus it is a possibility to identify the server implementation. Maybe some implementations uses some solutions which has no unimplemented parts and thus answers never with an unimplemented code. To find features like this can be used to identify implementations of SSH. This can be helpful to find server with an implementation, which has a known bug or error, to find targets for a hacker attack.
References


A Appendix

A.1 SSHTestService.java

Wrapper for automatic testing

```java
package com.jcraft.jsch;

/**
 * Wrapper for automatic testing
 * @author Mirjam van Nahmen <m.vannahmen @ student.ru.nl>
 * @version 1.0
 * @since 2014-01-31
 */
import static com.jcraft.jsch.NewSession.jsch;
import static com.jcraft.jsch.NewSession.kexinitDH;
import java.io.BufferedReader;
import java.io.IOException;
import java.io.InputStream;
import java.io.InputStreamReader;
import java.io.OutputStream;
import java.util.logging.Level;
import java.util.logging.Logger;
import java.net.Socket;
import java.util.*;
import sun.security.ssl.*;

@SuppressWarnings("deprecation")
public class SSHTestService {

    JSch jsch;
    Session session;
    //SSH-Login information
    String hostname = "localhost";
    String UserName = "Miri";
    String Password = "Hallo";
    String HostKeyAlias = "localhost";
    String Host = "127.0.0.1";

    //Booleans to check the previous state
    boolean connected = false;
    boolean versioned = false;
    boolean kexinited = false;
    boolean kexinitedDH = false;
    boolean newKeyed = false;

    //Cache for first-state
    private Session initState;
    private Buffer initBuf;
    private JSch initJSch;

    //Cache for second-state
    private Session versionedState;
    private Buffer versionBuf;
    private JSch versionJSch;

    //Cache for third-state
    private Session kexInitState;
    private Buffer kexInitBuf;
    private JSch kexInitJSch;

    //Cache for fourth-state
    private Session kexInitDHState;
    private Buffer kexInitDHBuf;
    private JSch kexInitDHJSch;

    private KeyExchange kexInitDHkex; //Key information

    //Cache for fifth-state
    private Session newKeyState;
    private Buffer newKeyBuf;
    private JSch newKeyJSch;
```
```
public static SSHTestService createSSHTestService() throws Exception {
    SSHTestService service = new SSHTestService();
    return service;
}

// Set up connection and set all caches
public SSHTestService() throws Exception {
    initConnection(); // set up connection
    initJSch = jsch;
    initState = session;
    session.connect();
    byte[] V_C = Util.str2byte("SSH-2.0-JSCH-"+JSch.VERSION);
    session.version(session.i, session.j, V_C);
    versionJSch = jsch;
    versionedState = session;
    versionBuf = session.buf;
    reset();
}

// set up connection
public void initConnection() throws JSchException {
    jsch = new JSch();
    session = new Session(jsch);
    session.setUserName(UserName);
    session.setPassword(Password);
    session.setHostKeyAlias(Password);
    session.setHost(host);
    java.util.Properties config = new java.util.Properties();
    config.put("StrictHostKeyChecking", "no");
    session.setConfig(config);
    System.out.println("Start...");
}

// Reset server and set new caches and init connection
public void reset() throws Exception {
    // ______Restart server_____
    Process p = Runtime.getRuntime().exec("sudo /home/miri/test.sh");
    p.waitFor();
    Thread.sleep(100);
    BufferedReader reader = new BufferedReader(new InputStreamReader(p.getErrorStream()));
    String line = reader.readLine();
    System.out.println("restart Server:"+line);
    // ___________________________
    initConnection();
    // set caches
    initJSch = jsch;
    initState = session;
    initBuf = session.buf;
    session.connect();
    byte[] V_C = Util.str2byte("SSH-2.0-JSCH-"+JSch.VERSION);
    session.version(session.i, session.j, V_C);
    versionJSch = jsch;
    versionedState = session;
    versionBuf = session.buf;
    session.kexinit(0, 0, 0, 0);
    kexInitJSch = jsch;
    kexInitState = session;
    kexInitBuf = session.buf;
    session.kexDH_init();
}
kexInitDHJSch = jsch;
kexInitDHState = session;
kexInitDHBuf = session.buf;
kexInitDHkex = session.tmp_kex;

// set actual state back to initial state
jsch = initJSch;
session = initState;
session.buf = initBuf;

// reset all booleans
connected = false;
versioned = false;
kexinited = false;
kexinitedDH = false;
}

// send connect request
public void connect() throws Exception {
    session.connect();
    connected = true;
}

/*
* Check the input of LearnLib which request should send
* @param input: input of LearnLib
* @return: the name of the request number which where response of the server
*/
public String processSymbol(String input) throws Exception {
    String inAction = input;
    System.out.println(inAction);

    // If LearnLib sends the request to send the versionnumber
    if (inAction.equals("version")) {
        String outAction;
        if (connected == false) { // If the server is not connected it is not possible to send a request
            kexinited = false;
            return "NoConnection";
        }
        byte[] V_C = Util.str2byte("SSH-2.0-JSCH-" + JSch.VERSION);
        // send request and store response
        outAction = session.version(session.i, session.j, V_C);
        // check response
        if (outAction == "ConnectionClosed") {
            connected = false;
        } else if (outAction == "VersionOK") {
            versioned = true;
            versionJSch = jsch;
            versionedState = session;
            versionBuf = session.buf;
            kexinited = false;
        }
        return outAction.toString();
    } else if (inAction.equals("kexinit")) {
        if (connected == false) {
            return "NoConnection"; // If the server is not connected it is not possible to send a request
        }
        if (versioned == false) {
            // If the previous state was not the send version, put the actual state of the client in the right state
            if (versioned == false)
```java
jsch = versionJSch;
session = versionedState;
session.buf = versionBuf;
}
//send request and store response
String outAction;
outAction = MSGToString(session.kexinit(0,0,0,0));
//prepare the state according to the response
if (outAction == "ConnectionClosed") {
    connected=false;
}
if(outAction == "KEXINIT")
{
    kexInitJSch = jsch;
kexInitState = session;
kexInitBuf = session.buf;
kexinited = true;
}
if (outAction == "DISCONNECT")
{
    connected = false;
}
if(outAction == "receiveVersionnr")
{
    reset();
}
if(outAction == "NEWKEYS")
{
    reset();
    versioned = false;
kexinited = false;
}
System.out.println(outAction);
return outAction;

//If LearnLib sends the request to send the KEXINIT_DH
else if (inAction.equals("kexinitDH")) {
    if(connected == false){
        return "NoConnection"; //If the server is not connected it is not possible to
        send a request
    }
    //If the previous state was not the KEXINIT, put the actual state of the client in
    the right state
    if(kexinited == false)
    {
        jsch = kexInitJSch;
session = kexInitState;
session.buf = kexInitBuf;
    }
    //send request and store response
    String outAction;
    outAction = MSGToString(session.kexDH_init());
    //prepare the state according to the response
    if (outAction == "ConnectionClosed") {
        session = new Session(jsch);
session.setUserName(UserName);
session.setPassword(Password);
session.setHostKeyAlias(Password);
session.setHost(host);
java.util.Properties config = new java.util.Properties();
cfg.put("StrictHostKeyChecking", "no");
session.setConfig(config);
connect();
byte[] V_C=Util.str2byte("SSH-2.0-JSCH-"+JSch.VERSION);
session.version(session.i, session.j,V_C);
session.kexinit(0,0,0,0);
```
if (outAction == "DISCONNECT") {
  connected = false;
}
if (outAction == "recieveVersionnr" || outAction == "InvalidProtocol") {
  reset();
}
if (outAction == "KEXDHREPLY") {
  kexinitedDH = true;
  kexinited = false;
  kexInitDHJSch = jsch;
  kexInitDHState = session;
  kexInitDHBuf = session.buf;
  kexInitDHkex = session.tmp_kex;
}
if (outAction == "NEWKEYS") {
  reset();
  versioned = false;
  kexinited = false;
}
System.out.println(outAction);
return outAction;
}
// If LearnLib sends the request to send the NEWKEY
else if (inAction.equals("newKey")) {
  if (connected == false) {
    return "NoConnection"; // If the server is not connected it is not possible to send a request
  }
  // If the previous state was not the KEXINIT_DH, put the actual state of the client in the right state
  if (kexinitedDH == false) {
    jsch = kexInitDHJSch;
    session = kexInitDHState;
    session.buf = kexInitDHBuf;
    session.tmp_kex = kexInitDHkex;
  }
  // send the request and store the response of the server
  String outAction;
  outAction = MSGToString(session.newKey());
  String outAction;
  // prepare the state according to the response
  if (outAction == "ConnectionClosed") {
    session = new Session(jsch);
    session.setUserName(UserName);
    session.setPassword(Password);
    session.setHostKeyAlias(Password);
    session.setHost(host);
    java.util.Properties config = new java.util.Properties();
    config.put("StrictHostKeyChecking", "no");
    session.setConfig(config);
    connect();
    byte[] V_C=Util.str2byte("SSH-2.0-JSCH-"+JSch.VERSION);
    session.version(session.i, session.j, V_C);
    session.kexinit(0,0,0,0);
    connected = false;
    versioned = false;
if (outAction == "DISCONNECT")
{
    connected = false;
}

if(outAction == "recieveVersionnr" || outAction == "InvalidProtocol")
{
    reset();
}

if(outAction == "NEWKEYS")
{
    newKeyed = true;
    kexinitedDH = false;
    kexinited = false;
    newKeyJSch = jsch;
    newKeyState = session;
    newKeyBuf = session.buf;
}

System.out.println(outAction);
return outAction;

else {
    System.out.println("Unknown input symbol ...");
    System.exit(0);
}

return null;

/**
 * Transelate the output of the server (the int’s which the Session class sends) into the
 * names for LearnLib
 * @param msg: the output of the server send of the session.class
 * @return name of request(String) without underscores, because LearnLib has problems with
 * it.
 * public static String MSGToString(int msg)
 * switch(msg){
 *    case 1: return "DISCONNECT";
 *    case 2: return "IGNORE";
 *    case 3: return "UNIMPLEMENTED";
 *    case 4: return "DEBUG";
 *    case 5: return "SERVICEREQUEST";
 *    case 6: return "SERVICEACCEPT";
 *    case 20: return "KEXINIT";
 *    case 21: return "NEWKEYS";
 *    case 30: return "KEXDHINIT";
 *    case 31: return "KEXDHREPLY";
 *    case -1: return "ConnectionClosed";
 *    case -5: return "recieveVersionnr";
 *    case -67: return "KEYEXCHANGEFAILED";
 *    case 46: return "InvalidProtocol";
 *    default: return ("somthing else" + msg);
 * }
 * public static void main(String[] args) throws Exception {
 * SSHTestService ssh = new SSHTestService();
 * ssh.reset();
 * }
 * */
A.2 Session.java

This version of the session.class of the JSch contains splitted function for every request.

```java
package com.jcraft.jsch;

/**
 * @modified from Mirjam van Nahmen <m.vannahmen@student.ru.nl>
 * @version 1.0
 * @since 2014-01-31
 * This version of the session.class of the JSch contains splitted function for every request.
 */

import java.io.*;
import java.net.*;

public class Session implements Runnable{

// Message numbers of the SSH
static final int SSH_MSG_DISCONNECT= 1;
static final int SSH_MSG_IGNORE= 2;
static final int SSH_MSG_UNIMPLEMENTED= 3;
static final int SSH_MSG_DEBUG= 4;
static final int SSH_MSG_SERVICE_REQUEST= 5;
static final int SSH_MSG_SERVICE_ACCEPT= 6;
static final int SSH_MSG_KEXINIT= 20;
static final int SSH_MSG_NEWKEYS= 21;
static final int SSH_MSG_KEXDH_INIT= 30;
static final int SSH_MSG_KEXDH_REPLY= 31;
static final int SSH_MSG_KEX_DH_GEX_GROUP= 32;
static final int SSH_MSG_KEX_DH_GEX_INIT= 33;
static final int SSH_MSG_KEX_DH_GEX_REPLY= 34;//End TPL
static final int SSH_MSG_GLOBAL_REQUEST= 80;
static final int SSH_MSG_REQUEST_SUCCESS= 81;
static final int SSH_MSG_REQUEST_FAILURE= 82;
static final int SSH_MSG_CHANNEL_OPEN= 90;
static final int SSH_MSG_CHANNEL_OPEN_CONFIRMATION= 91;
static final int SSH_MSG_CHANNEL_OPEN_FAILURE= 92;
static final int SSH_MSG_CHANNEL_WINDOW_ADJUST= 93;
static final int SSH_MSG_CHANNEL_DATA= 94;
static final int SSH_MSG_CHANNEL_EXTENDED_DATA= 95;
static final int SSH_MSG_CHANNEL_EOF= 96;
static final int SSH_MSG_CHANNEL_CLOSE= 97;
static final int SSH_MSG_CHANNEL_REQUEST= 98;
static final int SSH_MSG_CHANNEL_SUCCESS= 99;
static final int SSH_MSG_CHANNEL_FAILURE= 100;

private static final int PACKET_MAX_SIZE = 256 * 1024;
// Message numbers of the SSH
private static final int PACKET_MAX_SIZE = 256 * 1024;
// MI: if there was an error MSG
private boolean isDisconnect=false; // if there was a disconnect msg during the buffer reading
private boolean isDisconnected=false; // if there was a disconnect msg during the buffer reading
private boolean isUnimplemented=false; // if there was an unimplemented msg during the buffer reading
private boolean isIgnored=false; // if there was an ignored msg during the buffer reading

private byte[] V_S; // MI: server version
private byte[] V_C=Util.str2byte("SSH-2.0-JSCH-"+JSch.VERSION); // MI: client version
private byte[] I_C; // the payload of the client's SSH_MSG_KEXINIT
private byte[] I_S; // the payload of the server's SSH_MSG_KEXINIT
private byte[] K_S; // the host key
private byte[] session_id;
```

private byte[] MACs2c;
private int seqi=0;
private int seqo=0;
String[] guess=null;
private Cipher s2ccipher;
private Cipher c2scipher;
private MAC s2cmac;
private MAC c2smac;
private Compression deflater;
private Compression inflater;
private IO io;
Socket socket;
private int timeout=0;
int i, j;
volatile boolean isConnected=false;
private boolean isAuthed=false;
private Thread connectThread=null;
private Object lock=new Object();
boolean x11_forwarding=false;
boolean agent_forwarding=false;
InputStream in=null;
OutputStream out=null;
static Random random;
Buffer buf;
Packet packet;
SocketFactory socket_factory=null;
static final int buffer_margin = 32 + // maximum padding length
20 + // maximum mac length
32; // margin for deflater; deflater may inflate data
private java.util.Hashtable config=null;
Proxy proxy=null;
private UserInfo userinfo;
private String hostKeyAlias=null;
private int serverAliveInterval=0;
private int serverAliveCountMax=1;
private IdentityRepository identityRepository = null;
protected boolean daemon_thread=false;
private long kex_start_time=0L;
int max_auth_tries = 6;
int auth_failures = 0;
String host="127.0.0.1";
int port=22;
String username="Miri";
byte[] password= Util.str2byte("Hallo");
JSch jsch;

Session(JSch jsch) throws JSchException{
    super();
    this.jsch=jsch;
    buf=new Buffer();
    packet=new Packet(buf);
}

public String connect() throws JSchException, IOException, Exception{
    return connect(timeout);
}

int connectTimeout;

public String connect(int connectTimeout) throws JSchException, IOException, Exception{
    this.connectTimeout=connectTimeout;
    /*MI: if(isConnected){
        throw new JSchException("session is already connected");
    }*/
    io=new IO();
    if(random==null){
        try{
            Class c=Class.forName(getConfig("random"));
            random=(Random)c.newInstance();
        }
        catch(Exception e){
            throw new JSchException(e.toString(), e);
        }
    }
    Packet.setRandom(random);
    if(JSch.getLogger().isEnabled(Logger.INFO)){
        JSch.getLogger().log(Logger.INFO,"Connecting to "+host+" port "+port);
    }
    try {
        if(proxy==null){
            InputStream in;
            OutputStream out;
            if(socket_factory==null){
                socket=Util.createSocket(host, port, connectTimeout);
                in=socket.getInputStream();
                out=socket.getOutputStream();
            } else{
                socket=socket_factory.createSocket(host, port);
                in=socket_factory.getInputStream(socket);
                out=socket_factory.getOutputStream(socket);
            }
            //if(timeout>0){ socket.setSoTimeout(timeout); }
            socket.setTcpNoDelay(true);
            io.setInputStream(in);
            io.setOutputStream(out);
        } else{
            synchronized(proxy){
                proxy.connect(socket_factory, host, port, connectTimeout);
                io.setInputStream(proxy.getInputStream());
                io.setOutputStream(proxy.getOutputStream());
            }
        }
    }
if(connectTimeout>0 && socket!=null){
    socket.setSoTimeout(connectTimeout);
}

isConnected=true;

if(JSch.getLogger().isEnabled(Logger.INFO)){
    JSch.getLogger().log(Logger.INFO,
            "Connection established");
}

) catch(Exception e){}

if(isConnected)
    return ":1";
    return ":2";

//MI: V_C set the version outside of the class
public String version(int i, int j,byte[] V_C) throws IOException, JSchException
{
    jsch.addSession(this);
    
    // Some Cisco devices will miss to read ' \n' if it is sent separately.
    byte[] foo=new byte[V_C.length+1];
    System.arraycopy(V_C, 0, foo, 0, V_C.length);
    foo[foo.length-1]=(byte)'\n';
    io.put(foo, 0, foo.length);

    while(true){
        i=0;
        j=0;
        while(i<buf.buffer.length){
            try{
                j=io.getByte();
            }catch(SocketException e)
            {
                return "ConnectionClosed";
            }
            // System.out.print((char)j);
            if(j<0)break;
            buf.buffer[i]=(byte)j; i++;
            if(j==10)break;
        }
        if(j<0){
            return "ConnectionClosed";//new String(new byte[] {buf.getCommand()});
            // MI: throw new JSchException("connection is closed by foreign host");
        }
        if(buf.buffer[i-1]==10){ // 0x0a
            i--;
            if(i>0 && buf.buffer[i-1]==13){ // 0x0d
                i--;
            }
        }
        if(i<=3 ||
            (i==buf.buffer.length) &&
            (buf.buffer[0]=="S"||buf.buffer[1]=="S"||
            // It must not start with 'SSH-'
            // System.err.println(new String(buf.buffer, 0, i));
            }
continue;
}
;
if((i==buf.buffer.length) ||
i<7) // SSH-1.99 or SSH-2.0
   (buf.buffer[4]=='1' && buf.buffer[6]!='9') // SSH-1.5
   ){ System.out.println("old Server version ");/+(char)buf.buffer[4]+"."+(char)buf.buffer[6]);
   return "ConnectionClosed";
} //MI: throw new JSchException("invalid server's version string");
break;
}
V_S=new byte[i]; System.arraycopy(buf.buffer, 0, V_S, 0, i);
//System.err.println("V_S: ("+i+'") ['+new String(V_S)+'"]");
if(JSch.getLogger().isEnabled(Logger.INFO)){
   JSch.getLogger().log(Logger.INFO,
   "Remote version string: "+Util.byte2str(V_S));
   JSch.getLogger().log(Logger.INFO,
   "Local version string: "+Util.byte2str(V_C));
}
return "VersionOK";

/**
 * send kexinit with manual settings
 * @param kex_contant: default=0, only the first=1, empty=2
 * @param server_host_key_algorithms_contant: default=0, empty=1
 * @param cipher_c2s_contant: default=0, only the first=1, empty=2
 * @param cipher_s2c_contant: default=0, only the first=1, empty=2
 * @return the message number of the server
 * @throws Exception
 */
int kexinit(int kex_contant, int server_host_key_algorithms_contant, int cipher_c2s_contant, int cipher_s2c_contant) throws Exception
{
   send_kexinit( kex_contant, server_host_key_algorithms_contant, cipher_c2s_contant, cipher_s2c_contant);
   try{
      buf=read(buf);
   }catch(SocketException e)
   {
      return -1; //MI: Connection Closed
   }catch(IOException e)
   {
      return -1; //End of IOStream -> Connection closed
   }
   //MI: if not versioned server sends verionnr...abfangen und verwerten
   //System.out.println(strTmp);
   if(buf.tmpinput!=null)// == "SSH-2.0-OpenSSH_5.9p1 Debian-5ubuntu1.1")
      return -5;
   //MI: returns the MSG if there is one of the ERROR MSG
   if(isDisconnect)
      return 1; //new String(new byte[]){SSH_MSG_DISCONNECT});
   if(isDebuged)
      return 4; //new String(new byte[]){SSH_MSG_DEBUG});
   if(isUnimplemented)
      return 3; //new String(new byte[]){SSH_MSG_UNIMPLEMENTED});
   if(isIgnored)
      return 2; //new String(new byte[]){SSH_MSG_IGNORE});
}
byte tmp = buf.getCommand();
if(tmp!=SSH_MSG_KEXINIT){
    //MI: in_kex=false;
    System.out.println("no SSH_MSG_KEXINIT recieved");
    //MI: throw new JSchException("invalid protocol: "+buf.getCommand());
}
if(JSch.getLogger().isEnabled(Logger.INFO)){
    JSch.getLogger().log(Logger.INFO, "SSH_MSG_KEXINIT received");
}
return buf.getCommand(); //"SSH_MSG_KEXINIT";
}
KeyExchange tmp_kex;
public int kexDH_init() throws Exception
{
    //MI: get the algorithm information out of the packet
    KeyExchangeANDString ks=receive_kexinit(buf);
    KeyExchange kex=ks.kex;
    if(ks.tmp!=null)
    {
        System.err.println("kexDH_init "+ks.tmp);
        return -5;
    }
    else if(kex!=null){
        byte tmp;
        while(true){
            try{
                buf=read(buf);
            }catch(SocketException e) {
                return -1; //Connection closed
            }catch(IOException e) {
                return -1; //End of IOS stream -> Connection closed
            }
            tmp = buf.getCommand();
            if(kex.getState()==tmp){
                kex_start_time=System.currentTimeMillis();
                boolean result=kex.next(buf);
                if(!result){
                    System.out.println("verify: "+result);
                    in_kex=false;
                    return tmp;
                }
            }
            else{  
                in_kex=false;
                System.out.println("tmpinput: "+kex.tmpForVersionnr);
                System.out.println("invalid protocol(kex): "+buf.getCommand());
                return tmp;
            }
        }
    }
    else if(kex!=null){
        byte tmp;
        while(true){
            try{
                buf=read(buf);
            }catch(SocketException e) {
                return -1; //Connection closed
            }catch(IOException e) {
                return -1; //End of IOS stream -> Connection closed
            }
            tmp = buf.getCommand();
            if(kex.getState()==tmp){
                kex_start_time=System.currentTimeMillis();
                boolean result=kex.next(buf);
                if(!result){
                    System.out.println("verify: "+result);
                    in_kex=false;
                    return tmp;
                }
            }
            else{
                in_kex=false;
                System.out.println("tmpinput: "+kex.tmpForVersionnr);
                System.out.println("invalid protocol(kex): "+buf.getCommand());
                return tmp;
            }
            if(kex.getState()==KeyExchange.STATE_END){
                break;
            }
        }
    }
    tmp_kex=kex;
    return tmp;
}
public int newKey() throws JSchException, Exception
{
    KeyExchange kex = tmp_kex;
    return -10;
}
try{ checkHost(host, port, kex); }
catch(JSchException ee){
in_kex=false;
throw ee;
}

send_newkeys();

//MI: receive SSH_MSG_NEWKEYS(21)
try{
  buf=read(buf);
  }catch(SocketException e)
  {   
      return -1; //Connection closed
  }catch(IOException e)
  {
      return -1; //End of IOStream -> Connection closed
  }
}

public int service_accepted() throws JSchException, Exception{
  try{
    try{
      String s = getConfig("MaxAuthTries");
      if(s!=null){
        max_auth_tries = Integer.parseInt(s);
      }
    }catch(NumberFormatException e){
      throw new JSchException("MaxAuthTries: "+getConfig("MaxAuthTries"), e);
    }
    boolean auth=false;
    boolean auth_cancel=false;
    UserAuth ua=null;
    try{
      Class c=Class.forName(getConfig("userauth.none"));
      ua=(UserAuth)(c.newInstance());
    }catch(Exception e){
      throw new JSchException(e.toString(), e);
    }
    boolean auth=false;
    boolean auth_cancel=false;
    UserAuth ua=null;
    try{
      Class c=Class.forName(getConfig("userauth.none"));
      ua=(UserAuth)(c.newInstance());
    }catch(Exception e){
      throw new JSchException(e.toString(), e);
    }
    //MI: send SSH_MSG_SERVICE_REQUEST and receive SSH_MSG_SERVICE_ACCEPT(6)
    auth=ua.start(this);
return ua.service_recieved;
/* MI:
String cmethods=getConfig("PreferredAuthentications");
String[] cmethos=Util.split(cmethods, ",");
String[] smethods=null;
if(!auth){
  smethods= Util.split(((UserAuthNone)ua).getMethods();
  if(smethods!=null){
    smethods=smethods.toLowerCase();
  }
}
else{    
  // methods: publickey,password,keyboard-interactive
  //smethos="publickey,password,keyboard-interactive";
  smethods=cmethods;
}
String[] smethoda=Util.split(smethods, ",");
int methodi=0;
while(true){
  while(!auth &&
    cmethos!=null && methodi<cmethos.length){
    String method=cmethos[methodi++];
    boolean acceptable=false;
    for(int k=0; k<smethoda.length; k++){
      if(smethoda[k].equals(method)){
        acceptable=true;
        break;
      }
    }
    if(!acceptable){
      continue;
    }
    //System.err.println(" method: "+method);
    if(JSch.getLogger().isEnabled(Logger.INFO)){
      String str="Authentications that can continue: ";
      for(int k=methodi-1; k< cmethos.length; k++){
        str+=cmethos[k];
        if(k+1<cmethos.length)
          str+=",";
      }
      JSch.getLogger().log(Logger.INFO,
        str);
      JSch.getLogger().log(Logger.INFO,
        "Next authentication method: "+method);
    }
    ua=null;
    try{
      Class c=null;
      if(getConfig("userauth."+method)!=null){
        c=Class.forName(getConfig("userauth."+method));
        ua=(UserAuth)(c.newInstance());
      }
    }
    catch(Exception e){
      if(JSch.getLogger().isEnabled(Logger.WARN)){
        JSch.getLogger().log(Logger.WARN,
          "failed to load "+method+" method");
      }
if(ua!=null){
    auth_cancel=false;
    try{
        auth=ua.start(this);
        if(auth &
            JSch.getLogger().isEnabled(Logger.INFO)){
            JSch.getLogger().log(Logger.INFO,
                "Authentication succeeded ("+method+").");
        }
        catch(JSchAuthCancelException ee){
            auth_cancel=true;
        }
        catch(JSchPartialAuthException ee){
            String tmp = smethods;
            smethods=ee.getMethods();
            smethods=Util.split(smethods, ",");
            if(!tmp.equals(smethods)){
                methodi=0;
            }
            //System.err.println("PartialAuth: "+methods);
            auth_cancel=false;
            continue loop;
        }
        catch(RuntimeException ee){
            throw ee;
        }
        catch(JSchException ee){
            throw ee;
        }
        catch(Exception ee){
            //System.err.println("ee: "+ee); // SSH_MSG_DISCONNECT: 2 Too many authen-
            failures
            if(JSch.getLogger().isEnabled(Logger.WARN)){
                JSch.getLogger().log(Logger.WARN,
                    "an exception during authentication
                    "+ee.toString());
            }
            break loop;
        }
        break;
    }
    if(!auth){
        if(!auth_failures >= max_auth_tries){
            if(JSch.getLogger().isEnabled(Logger.INFO)){
                JSch.getLogger().log(Logger.INFO,
                    "Login trials exceeds "+max_auth_tries);
            }
        }
        if(auth_cancel)
            throw new JSchException("Auth cancel");
        throw new JSchException("Auth fail");
    }
    if(connectTimeout>0 || timeout>0){
        socket.setSoTimeout(timeout);
    }
    isAuthed=true;
    synchronized(lock){
        if(isConnected){
            connectThread=new Thread(this);
            connectThread.setName("Connect thread "+host+" session");
        }
    }
}
if(daemon_thread)
    connectThread.setDaemon(daemon_thread);
}
connectThread.start();
}
else{
    //MI: The session has been already down and
    //MI: we don’t have to start new thread.
}
} //MI: return (int) service_accepted;++
}
}
}

try{
    packet.reset();
    buf.putByte((byte)SSH_MSG_DISCONNECT);
    buf.putInt(3);
    buf.putString(Util.str2byte(e.toString()));
    buf.putString(Util.str2byte("en"));
    write(packet);
    disconnect();
    return SSH_MSG_DISCONNECT;
}
catch(Exception ee){
}

isConnected=false;
//printStackTrace();
if(e instanceof RuntimeException) throw (RuntimeException)e;
if(e instanceof JSchException) throw (JSchException)e;
throw new JSchException("Session.connect: "+e);
}
finally{
    Util.bzero(this.password);
    this.password=null;
}

@SuppressWarnings("empty-statement")
private KeyExchangeANDString receive_kexinit(Buffer buf) throws Exception {
    String tmpinput=new String(buf.buffer,"UTF-8");
    String tmpForVersionnr = null;
    if(tmpinput.contains("SSH-")) //"SSH-2.0-OpenSSH_5.9p1 Debian-5ubuntu1.1")
    {
        tmpForVersionnr = tmpinput;
        System.out.println("receive_kexinit"+tmpinput);
    }
    else{
        int j=buf.getInt();
        if(j!=buf.getLength()){ // packet was compressed and
            buf.getBye(); // j is the size of deflated packet.
            I_S=new byte[buf.index-5];
        }
        else{
            I_S=new byte[j-1-buf.getBye()];
        }
        System.arraycopy(buf.buffer, buf.s, I_S, 0, I_S.length);
    }
    // if(!in_kex){ // We are in rekeying activated by the remote!
    // send_kexinit(0,0,0,0);
    // }
    //MI: guess=KeyExchange.guess(I_S, I_C);
    //MI: set guess allways on normal session state
    guess=new String[]{"diffie-hellman-group1-sha1","ssh-rsa","aes128-ctr","aes128-ctr
  ","hmac-md5","hmac-md5","none","none","",""};

35
System.out.println("guess ");
for(int k = 0; k< guess.length;k++)
System.out.print(""+guess[k]+"",");
if(guess==null){
throw new JSchException("Algorithm negotiation fail");
}
/*
if(!isAuthed &&
(guess[KeyExchange.PROPOSAL_ENC_ALGS_CTOS].equals("none") ||
(guess[KeyExchange.PROPOSAL_ENC_ALGS_STOC].equals("none"))){
throw new JSchException("NONE Cipher should not be chosen before authentification is succeeded.");
}
*/
KeyExchange kex=null;
if(tmpForVersionnr==null)
{
try{
Class c=Class.forName(getConfig(guess[KeyExchange.PROPOSAL_KEX_ALGS]));
kex=(KeyExchange)(c.newInstance());
}
catch(Exception e){
throw new JSchException(e.toString(), e);
}

kex.init(this, V_S, V_C, I_S, I_C);
}
// kex.init(this, V_S, V_C, I_S, I_C);
// kex.tmpForVersionnr=tmpForVersionnr;

KeyExchangeANDString ks = new KeyExchangeANDString(kex,tmpForVersionnr);
return ks;
}
/*
* make the kex_init_packet manually
* @param kex_contant: default=0,only the first=1,empty=2
* @param server_host_key_algorithms_contant: default=0,empty=1
* @param cipher_c2s_contant: default=0,only the first=1,empty=2
* @param cipher_s2c_contant
* @throws Exception
*/
void send_kexinit(int kex_contant,int server_host_key_algorithms_contant, int cipher_c2s_contant, int cipher_s2c_contant) throws Exception {
String ciphers2c, cipherc2s;
if(cipher_c2s_contant==0){
cipherc2s=getConfig("cipher.c2s");//"aes128-ctr,aes128-cbc,3des-ctr,3des-cbc,
blowfish-cbc,aes192-cbc,aes256-cbc"
}else if(cipher_c2s_contant==1){
cipherc2s = "aes128-ctr";
}
else{
cipherc2s="asdf";
}

if(cipher_s2c_contant==0){
ciphers2c=getConfig("cipher.s2c"); //"aes128-ctr,aes128-cbc,3des-ctr,3des-cbc,
blowfish-cbc,aes192-cbc,aes256-cbc"
}else if(cipher_s2c_contant==1){
ciphers2c = "aes128c-cbc";
}
else{
ciphers2c="asdf";
}
String[] not_available_ciphers = checkCiphers(getConfig("CheckCiphers")); // aes256-ctr, aes192-ctr, aes128-ctr, aes256-cbc, aes192-cbc, aes128-cbc, 3des-ctr, arcfour, arcfour128, arcfour256

if(cipher_c2s_contant == 0 && cipher_s2c_contant == 0 && not_available_ciphers != null && not_available_ciphers.length > 0) {
    cipherc2s = Util.diffString(cipherc2s, not_available_ciphers);
    ciphers2c = Util.diffString(ciphers2c, not_available_ciphers);
    if(cipherc2s == null || ciphers2c == null) {
        throw new JSchException("There are not any available ciphers.");
    }
}

//MI: kex_algorithms_contant: default: 0, only the first: 1, empty: 2
String kex;
if(kex_contant == 0) {
    kex = getConfig("kex"); // "diffie-hellman-group1-shal, diffie-hellman-group14-shal, diffie-hellman-group-exchange-shal"
    String[] not_available_kexes = checkKexes(getConfig("CheckKexes")); // "diffie-hellman-group14-shal"
    if(not_available_kexes != null && not_available_kexes.length > 0) {
        kex = Util.diffString(kex, not_available_kexes);
        if(kex == null) {
            throw new JSchException("There are not any available kexes.");
        }
    }
}
else if(kex_contant == 1) {
    kex = getConfig("diffie-hellman-group1-shal");
}
else {
    kex = " ";
}

//MI: server_host_key_algorithms_contant: default: 0, empty: 1
String server_host_key_algorithms;
if(server_host_key_algorithms_contant == 0)
    server_host_key_algorithms = getConfig("server_host_key");
else
    server_host_key_algorithms = " ";

in_kex = true;
kex_start_time = System.currentTimeMillis();

// byte SSH_MSG_KEXINIT(20)
// byte[16] cookie (random bytes)
// string kex_algorithms
// string server_host_key_algorithms
// string encryption_algorithms_client_to_server
// string encryption_algorithms_server_to_client
// string mac_algorithms_client_to_server
// string mac_algorithms_server_to_client
// string compression_algorithms_client_to_server
// string compression_algorithms_server_to_client
// string languages_client_to_server
// string languages_server_to_client
Buffer buf = new Buffer(); // send_kexinit may be invoked
Packet packet = new Packet(buf); // by user thread.
packet.reset();
buf.putByte((byte) SSH_MSG_KEXINIT);
buf.putString(Util.str2byte(kex));
buf.putString(Util.str2byte(server_host_key_algorithms));
buf.putString(Util.str2byte(cipherc2s));
// set normal state
cipherc2s = getConfig("cipher.c2s"); // "aes128-ctr, aes128-cbc, 3des-ctr, 3des-cbc, blowfish-cbc, aes192-cbc, aes256-cbc"
buf.putString(Util.str2byte(ciphers2c));

// set normal state
ciphers2c=getConfig("cipher.s2c");"aes128-ctr,aes128-cbc,3des-ctr,3des-cbc,
blowfish-cbc,aes192-cbc,aes256-cbc"
if(cipher_c2s_contant==0 && cipher_s2c_contant==0 && not_available_ciphers!=
null && not_available_ciphers.length>0){
ciphers2c=Util.diffString(ciphers2c, not_available_ciphers);
if(cipherc2s==null || ciphers2c==null){
    throw new JSchException("There are not any available ciphers.");
}
} else {
    buf.putString(Util.str2byte(getConfig("mac.c2s")));    
    buf.putString(Util.str2byte(getConfig("mac.s2c")));    
    buf.putString(Util.str2byte(getConfig("compression.c2s")));    
    buf.putString(Util.str2byte(getConfig("compression.s2c")));    
    buf.putString(Util.str2byte(getConfig("lang.c2s")));    
    buf.putString(Util.str2byte(getConfig("lang.s2c")));    
    buf.putByte((byte)0);    
    buf.putInt(0);
    buf.setOffset(5);
    I_C=new byte[buf.getLength()];
    buf.getByte(I_C);
    write(packet);
    if(JSch.getLogger().isEnabled(Logger.INFO)){
        JSch.getLogger().log(Logger.INFO,
            "SSH_MSG_KEXINIT sent");
    }
}

private void send_newkeys() throws Exception {
    // send SSH_MSG_NEWKEYS(21)
    packet.reset();
    buf.putByte((byte)SSH_MSG_NEWKEYS);
    write(packet);
    if(JSch.getLogger().isEnabled(Logger.INFO)){
        JSch.getLogger().log(Logger.INFO,
            "SSH_MSG_NEWKEYS sent");
    }
}

private void checkHost(String chost, int port, KeyExchange kex) throws JSchException {
    String shkc=getConfig("StrictHostKeyChecking");
    if(hostKeyAlias!=null){
        chost=hostKeyAlias;
    }
    //System.err.println("shkc: "+shkc);
    byte[] K_S=kex.getHostKey();
    String key_type=kex.getKeyType();
    String key_fprint=kex.getFingerPrint();
    if(hostKeyAlias==null && port!=22){
        chost="(["+chost+":]"+port+";
    }
    HostKeyRepository hkr=jsch.getHostKeyRepository();
    String hkh=getConfig("HashKnownHosts");
    if(hkh.equals("yes") && (hkr instanceof KnownHosts)){
        hostkey=((KnownHosts)hkr).createHashedHostKey(chost, K_S);
    }
}
else{
    hostkey=new HostKey(chost, K_S);
}

int i=0;
synchronized(hkr){
i=hkr.check(chost, K_S);
}
boolean insert=false;
if((shkc.equals("ask") || shkc.equals("yes")) &&
i==HostKeyRepository.CHANGED){
    String file=null;
synchronized(hkr){
        file=hkr.getKnownHostsRepositoryID();
    }
    if(file==null) file="known_hosts";
    boolean b=false;
    if(userinfo!=null){
        String message=
            "WARNING: REMOTE HOST IDENTIFICATION HAS CHANGED!\n"
            "IT IS POSSIBLE THAT SOMEONE IS DOING SOMETHING NASTY!\n"
            "Someone could be eavesdropping on you right now (man-in-the-middle attack)!\n"
            "It is also possible that the " + key_type + " host key has just been changed.\n"
            "The fingerprint for the " + key_type + " key sent by the remote host is\n"
            key_fprint + "\n"
            "Please contact your system administrator.\n"
            "Add correct host key in " + file + " to get rid of this message.\n"
            "Do you want to delete the old key and insert the new key?\n"
        
        b=userinfo.promptYesNo(message+
            
        }
    else{ // shkc.equals("yes")
        userinfo.showMessage(message);
    }
}
if(!b){
    throw new JSchException("HostKey has been changed: " + chost);
}
if(userinfo!=null){
    String message=
        
    } else{ // shkc.equals("yes")
        userinfo.showMessage(message);
    }
}

if(!foo){
    throw new JSchException("reject HostKey: " + host);
}
if((shkc.equals("ask") || shkc.equals("yes")) &&
    (!=HostKeyRepository.OK) && !insert){
    if(shkc.equals("yes")){
        throw new JSchException("reject HostKey: " + host);
    }
    //System.err.println("finger-print: " + key_fprint);
    if(userinfo!=null){
        boolean foo=userinfo.promptYesNo{
            "The authenticity of host '" + host + "' can’t be established.\n"
            key_type + " key fingerprint is " + key_fprint + ".\n"
            "Are you sure you want to continue connecting?"
            
        }
        if(!foo){
            throw new JSchException("reject HostKey: " + host);
        }
    }
insert=true;
}
else{
    if(i==HostKeyRepository.NOT_INCLUDED)
        throw new JSchException("UnknownHostKey: "+host+. "+key_type+" key fingerprint is "+key_fprint);
    else
        throw new JSchException("HostKey has been changed: "+host);
}

if(shkc.equals("no") &&
    HostKeyRepository.NOT_INCLUDED==i){
    insert=true;
}

if(i==HostKeyRepository.OK &&
    JSch.getLogger().isEnabled(Logger.INFO)){
    JSch.getLogger().log(Logger.INFO,
        "Host "+host+" is known and mathces the "+key_type+" host key ");
}

if(insert &&
    JSch.getLogger().isEnabled(Logger.WARN)){
    JSch.getLogger().log(Logger.WARN,
        "Permanently added "+host+" ("+key_type+") to the list of
        known hosts.");
}

if(insert){
    synchronized(hkr){
        hkr.add(hostkey, userinfo);
    }
}

//public void start(){ (new Thread(this)).start(); }

public Channel openChannel(String type) throws JSchException{
    if(!isConnected){
        throw new JSchException("session is down");
    }
    try{
        Channel channel=Channel.getChannel(type);
        addChannel(channel);
        channel.init();
        return channel;
    }
    catch(Exception e){
        //e.printStackTrace();
    }
    return null;
}

// encode will bin invoked in write with synchronization.
public void encode(Packet packet) throws Exception{
    //System.err.println("encode: "+packet.buffer.getCommand());
    //System.err.println(" "+packet.buffer.index);
    if(deflater!=null){
        compress_len[0]=packet.buffer.index;
        packet.buffer.buffer=deflater.compress(packet.buffer.buffer,
            5, compress_len);
        packet.buffer.index=compress_len[0];
    }
    if(c2scipher!=null){
        //
packet.padding(c2scipher.getIVSize());
packet.padding(c2scipher_size);
int pad=packet.buffer.buffer[4];
synchronized(random){
    random.fill(packet.buffer.buffer, packet.buffer.index-pad, pad);
}
else{
    packet.padding(8);
}
if(c2smac!=null){
c2smac.update(seqo);
c2smac.update(packet.buffer.buffer, 0, packet.buffer.index);
c2smac.doFinal(packet.buffer.buffer, packet.buffer.index);
} else{
    byte[] buf=packet.buffer.buffer;
c2scipher.update(buf, 0, packet.buffer.index, buf, 0);
if(c2smac!=null){
packet.buffer.skip(c2smac.getBlockSize());
}
}
int[] uncompress_len=new int[1];
int[] compress_len=new int[1];
private int s2ccipher_size=8;
private int c2scipher_size=8;
public Buffer read(Buffer buf) throws Exception{
j=0;
while(true){
io.getByte(buf.buffer, buf.index, s2ccipher_size);
buf.index+=s2ccipher_size;
if(s2ccipher!=null){
s2ccipher.update(buf.buffer, 0, s2ccipher_size, buf.buffer, 0);
}
j=((buf.buffer[0]<<24)&0xff000000) |
((buf.buffer[1]<<16)&0x00ff0000) |
((buf.buffer[2]<< 8)&0x0000ff00) |
((buf.buffer[3] )&0x000000ff);
buf.tmpinput=null;
//Mi: check if versionn is send(connect-> kexinit(without version)
tmpinput=new String(buf.buffer,"UTF-8");
if(tmpinput.contains("SSH-"))="/SSH-2.0-OpenSSH_5.9p1 Debian-Subuntul.1"){
    buf.tmpinput=tmpinput;
    break;
}
// System.out.println(new String(buf.buffer,"UTF-8");)
// RFC 4253 6.1. Maximum Packet Length
if(j<5 | j>PACKET_MAX_SIZE){
    start_discard(buf, s2ccipher, s2cmac, j, PACKET_MAX_SIZE);
} else{
    packet.padding(8);
}
System.out.println("index: "+buf.index+" j: "+j+" s2ccipher_size: "+s2ccipher_size);
if(need<0){
    throw new IOException("invalid data");
}
System.out.println("bufferbyte: "+ buf.buffer[0]);
if((buf.index+need)>buf.buffer.length){
    byte[] foo=new byte[buf.index+need];
    System.arraycopy(buf.buffer, 0, foo, 0, buf.index);
    buf.buffer=foo;
}
if((need%s2ccipher_size)!=0){
    String message="Bad packet length "+need;
    if(JSch.getLogger().isEnabled(Logger.FATAL)){
        JSch.getLogger().log(Logger.FATAL, message);
    }
    start_discard(buf, s2ccipher, s2cmac, j, PACKET_MAX_SIZE-s2ccipher_size);
}
if(need>0){
    io.getByte(buf.buffer, buf.index, need); buf.index+=(need);
    if(s2ccipher!=null){
        s2ccipher.update(buf.buffer, s2ccipher_size, need, buf.buffer, s2ccipher_size);
    }
    if(s2cmac!=null){
        s2cmac.update(seqi);
        s2cmac.update(buf.buffer, 0, buf.index);
        s2cmac.doFinal(s2cmac_result1, 0);
        io.getByte(s2cmac_result2, 0, s2cmac_result2.length);
        if(!java.util.Arrays.equals(s2cmac_result1, s2cmac_result2)){
            if(need > PACKET_MAX_SIZE){
                throw new IOException("MAC Error");
            }
            start_discard(buf, s2ccipher, s2cmac, j, PACKET_MAX_SIZE-need);
            continue;
        }
    }
}
seqi++;
if(inflater!=null){
    //inflater.uncompress(buf);
    int pad=buf.buffer[4];
    uncompress_len[0]=buf.index-5-pad;
    byte[] foo=inflater.uncompress(buf.buffer, 5, uncompress_len);
    if(foo!=null){
        buf.buffer=foo;
        buf.index=5+uncompress_len[0];
    }
    else{
        System.err.println("fail in inflater");
        break;
    }
}
int type=buf.getCommand()&0xff;
if(type==SSH_MSG_DISCONNECT){
    buf.rewind();
    buf.getInt();buf.getShort();
    int reason_code=buf.getInt();
    byte[] description=buf.getString();
    byte[] language_tag=buf.getString();
    //throw new JSchException("SSH_MSG_DISCONNECT: "+

isDisconnect=true;
break;
)
else if(type==SSH_MSG_IGNORE){
isIgnored=true;
}
else if(type==SSH_MSG_UNIMPLEMENTED){
buf.rewind();
buf.getInt();buf.getShort();
int reason_id=buf.getInt();
if(JSch.getLogger().isEnabled(Logger.INFO)){
JSch.getLogger().log(Logger.INFO,
"Received SSH_MSG_UNIMPLEMENTED for "+reason_id);
}
isUnimplemented=true;
}
else if(type==SSH_MSG_DEBUG){
buf.rewind();
buf.getInt();buf.getShort();
isDebuged=true;
*/
byte always_display=(byte)buf.getByte();
byte[] message=buf.getString();
byte[] language_tag=buf.getString();
System.err.println("SSH_MSG_DEBUG:"+
" "+Util.byte2str(message)+
" "+Util.byte2str(language_tag));
*/
else if(type==SSH_MSG_CHANNEL_WINDOW_ADJUST){
buf.rewind();
buf.getInt();buf.getShort();
Channel c=Channel.getChannel(buf.getInt(), this);
if(c==null){
}
else{
c.addRemoteWindowSize(buf.getInt());
}
}
else if(type==UserAuth.SSH_MSG_USERAUTH_SUCCESS){
isAuthed=true;
if(inflater==null && deflater==null){
String method;
method=gx[KeyExchange.PROPOSAL_COMP_ALGS_CTOS];
initDeflater(method);
method=gx[KeyExchange.PROPOSAL_COMP_ALGS_STOC];
initInflater(method);
}
break;
}
else{
break;
}
}
buf.rewind();

//Mi: check tmpinput

*/
if(tmpinput!=null)
{
buf = new Buffer();
byte[] temp=tmpinput.getBytes();
buf.buffer=temp;
}*/
return buf;
```java
private byte start_discard(Buffer buf, Cipher cipher, MAC mac,
    int packet_length, int discard) throws JSchException,
    IOException{
    MAC discard_mac = null;

    if(cipher==null||!cipher.isCBC()){
        return buf.getCommand();// throw new JSchException("Packet corrupt");
    }

    if(packet_length!=PACKET_MAX_SIZE && mac != null){
        discard_mac = mac;
    }

    discard -= buf.index;

    while(discard>0){
        buf.reset();
        int len = discard>buf.buffer.length ? buf.buffer.length : discard;
        io.getByte(buf.buffer, 0, len);
        if(discard_mac!=null){
            discard_mac.update(buf.buffer, 0, len);
        }
        discard -= len;
    }

    if(discard_mac!=null){
        discard_mac.doFinal(buf.buffer, 0);
    }

    throw new JSchException("Packet corrupt");
}

byte[] getLsid(){
    return session_id;
}

private void receive_newkeys(Buffer buf, KeyExchange kex) throws Exception {
    updateKeys(kex);
    in_ke=false;
}

private void updateKeys(KeyExchange kex) throws Exception{
    byte[] K=kex.getK();
    byte[] H=kex.getH();
    HASH hash=kex.getHash();

    if(session_id==null){
        session_id=new byte[H.length];
        System.arraycopy(H, 0, session_id, 0, H.length);
    }

    /*
    Initial IV client to server: HASH (K || H || "A" || session_id)
    Initial IV server to client: HASH (K || H || "B" || session_id)
    Encryption key client to server: HASH (K || H || "C" || session_id)
    Encryption key server to client: HASH (K || H || "D" || session_id)
    Integrity key client to server: HASH (K || H || "E" || session_id)
    Integrity key server to client: HASH (K || H || "F" || session_id)
    */
    buf.reset();
    buf.putMPInt(K);
    buf.putByte(H);
    buf.putByte((byte)0x41);
    buf.putByte(session_id);
    hash.update(buf.buffer, 0, buf.index);
    IVc2s=hash.digest();

    int j=buf.index-session_id.length-1;
```

44
buf.buffer[j]++;  
hash.update(buf.buffer, 0, buf.index);  
IVs2c=hash.digest();  
buf.buffer[j]++;  
hash.update(buf.buffer, 0, buf.index);  
Ec2s=hash.digest();  
buf.buffer[j]++;  
hash.update(buf.buffer, 0, buf.index);  
MACc2s=hash.digest();  
buf.buffer[j]++;  
hash.update(buf.buffer, 0, buf.index);  
MACs2c=hash.digest();  

try{
  Class c;
  String method;
  method=guess[KeyExchange.PROPOSAL_ENC_ALGS_STOC];
  c=Class.forName(getConfig(method));
  s2ccipher=(Cipher)(c.newInstance());
  while(s2ccipher.getBlockSize()>Es2c.length){
    buf.reset();
    buf.putMPInt(K);
    buf.putByte(H);
    buf.putByte(Es2c);
    hash.update(buf.buffer, 0, buf.index);
    byte[] foo=hash.digest();
    byte[] bar=new byte[Es2c.length+foo.length];
    System.arraycopy(Es2c, 0, bar, 0, Es2c.length);
    System.arraycopy(foo, 0, bar, Es2c.length, foo.length);
    Es2c=bar;
  }

  s2ccipher.init(Cipher.DECRYPT_MODE, Es2c, IVs2c);
  s2ccipher_size=s2ccipher.getIVSize();

  method=guess[KeyExchange.PROPOSAL_MAC_ALGS_STOC];
  c=Class.forName(getConfig(method));
  s2cmac=(MAC)(c.newInstance());
  MACs2c = expandKey(buf, K, H, MACs2c, hash, s2cmac.getBlockSize());
  s2cmac.init(MACs2c);
  //mac_buf=new byte[s2cmac.getBlockSize()];
  s2cmac_result1=new byte[s2cmac.getBlockSize()];
  s2cmac_result2=new byte[s2cmac.getBlockSize()];

  method=guess[KeyExchange.PROPOSAL_ENC_ALGS_CTOS];
  c=Class.forName(getConfig(method));
  c2scipher=(Cipher)(c.newInstance()));
  while(c2scipher.getBlockSize()>Ec2s.length){
    buf.reset();
    buf.putMPInt(K);
    buf.putByte(H);
    buf.putByte(Ec2s);
    hash.update(buf.buffer, 0, buf.index);
    byte[] foo=hash.digest();
    byte[] bar=new byte[Ec2s.length+foo.length];
    System.arraycopy(Ec2s, 0, bar, 0, Ec2s.length);
    System.arraycopy(foo, 0, bar, Ec2s.length, foo.length);
    Ec2s=bar;
  }

  c2scipher.init(Cipher.ENCRYPT_MODE, Ec2s, IVc2s);
  c2scipher_size=c2scipher.getIVSize();
c=Class.forName(getConfig(method));
c2smac=(MAC)(c.newInstance());
MACc2s = expandKey(buf, K, H, MACc2s, hash, c2smac.getBlockSize());
c2smac.init(MACc2s);

method=guess[KeyExchange.PROPOSAL_COMP_ALGS_CTOS];
isทนDeflater(method);

method=guess[KeyExchange.PROPOSAL_COMP_ALGS_STOC];
isทนInflater(method);
}
catch(Exception e){
    if(e instanceof JSchException)
        throw e;
    throw new JSchException(e.toString(), e);
    //System.err.println("updatekeys: "+e);
}
}

private byte[] expandKey(Buffer buf, byte[] K, byte[] H, byte[] key, 
    HASH hash, int required_length) throws Exception {
    byte[] result = key;
    int size = hash.getBlockSize();
    while(result.length < required_length){
        buf.reset();
        buf.putMPInt(K);
        buf.putByte(H);
        buf.putByte(result);
        hash.update(buf.buffer, 0, buf.index);
        byte[] tmp = new byte[result.length+size];
        System.arraycopy(result, 0, tmp, 0, result.length);
        Util.bzero(result);
        result = tmp;
    }
    return result;
}

/*public*/ /*synchronized*/ void write(Packet packet, Channel c, int length) throws 
    Exception{
    long t = getTimeout();
    while(true){
        if(!in_kex){
            if(t>0L && (System.currentTimeMillis()-kex_start_time)>t){
                throw new JSchException("timeout in wating for rekeying process.");
            }
            try{Thread.sleep(10);}catch(java.lang.InterruptedException e){};
        }
        synchronized(c){
            }
        continue;
    }
if(c.rwsize<length){
    try{
        c.notifyme++;
        c.wait(100);
    }
    catch(java.lang.InterruptedException e){
        finally{
            c.notifyme--;
        }
    }
}

if(c.rwsize>=length){
    c.rwsize-=length;
    break;
}

if(c.close || !c.isConnected()){
    throw new IOException("channel is broken");
}

boolean sendit=false;
int s=0;
byte command=0;
int recipient=-1;
synchronized(c){
    if(c.rwsize>0){
        long len=c.rwsize;
        if(len>length){
            len=length;
        }
        if(len!=length){
            s=packet.shift((int)len,
                (c2scipher!=null ? c2scipher_size : 8),
                (c2smac!=null ? c2smac.getBlockSize() : 0));
        }
        command=packet.buffer.getCommand();
        recipient=c.getRecipient();
        length-=len;
        c.rwsize-=len;
        sendit=true;
    }
    if(sendit){
        _write(packet);
        if(length==0){
            return;
        }
        packet.unshift(command, recipient, s, length);
    }
}

if(in_kex){
    continue;
}

if(c.rwsize>=length){
    c.rwsize-=length;
    break;
    }

    _write(packet);
}

public void write(Packet packet) throws Exception{
    // System.err.println("in_kex="+in_kex++ "+(packet.buffer.getCommand())");
    long t = getTimeout();
    while(in_kex){
if(t>0L && (System.currentTimeMillis()-kex_start_time)>t){
    throw new JSchException("timeout in wating for rekeying process.");
}

//MI: without TPL
byte command=packet.buffer.getCommand();
//System.err.println("command: "+command);
if(command==SSH_MSG_KEXINIT ||
   command==SSH_MSG_NEWKEYS ||
   command==SSH_MSG_KEXDH_INIT ||
   command==SSH_MSG_KEXDH_REPLY ||
   command==SSH_MSG_KEX_DH_GEX_GROUP ||
   command==SSH_MSG_KEX_DH_GEX_INIT ||
   command==SSH_MSG_KEX_DH_GEX_REPLY ||
   command==SSH_MSG_KEX_DH_GEX_REQUEST ||
   command==SSH_MSG_DISCONNECT) {
    break;
}
try{Thread.sleep(10);}
catch(java.lang.InterruptedException e){};

_write(packet);

private void _write(Packet packet) throws Exception{
    synchronized(lock){
        encode(packet);
        if(io!=null){
            io.put(packet);
            seqo++;
        }
    }
}

Runnable thread;
public void run(){
    thread=this;
    byte[] foo;
    Buffer buf=new Buffer();
    Packet packet=new Packet(buf);
    int i=0;
    Channel channel;
    int[] start=new int[1];
    int[] length=new int[1];
    KeyExchange kex=null;
    int stimeout=0;
    try{
        while(isConnected &&
            thread!=null){
            try{
                buf=read(buf);
                stimeout=0;
            }catch(InterruptedIOException/*SocketTimeoutException*/ ee){
                if(!in_kex && stimeout<serverAliveCountMax){
                    sendKeepAliveMsg();
                    stimeout++;
                    continue;
                } else if(in_kex && stimeout<serverAliveCountMax){
                    stimeout++;
                    continue;
                }
                throw ee;
            }
            msgType=buf.getCommand()&0xff;
        }
    }
if(kex!=null & kex.getState()==msgType){
    kex_start_time=System.currentTimeMillis();
    boolean result=kex.next(buf);
    if(!result){
        throw new JSchException("verify: "+result);
    }
    continue;
}

switch(msgType){
    case SSH_MSG_KEXINIT:
        //System.err.println("KEXINIT");
        KeyExchangeANDString ks=receive_kexinit(buf);
        kex=ks.kex;
        break;
    case SSH_MSG_NEWKEYS:
        //System.err.println("NEWKEYS");
        send_newkeys();
        receive_newkeys(buf, kex);
        kex=null;
        break;
    case SSH_MSG_CHANNEL_DATA:
        buf.getInt();
        buf.getByte();
        i=buf.getInt();
        channel=Channel.getChannel(i, this);
        foo=buf.getString(start, length);
        if(channel==null){
            break;
        }
        if(length[0]==0){
            break;
        }
        try{
            channel.write(foo, start[0], length[0]);
        } catch(Exception e){
            //System.err.println(e);
            try{channel.disconnect();}catch(Exception ee){}
            break;
        }
        int len=length[0];
        channel.setLocalWindowSize(channel.lwsize-len);
        if(channel.lwsize<channel.lwsize_max/2){
            packet.reset();
            buf.putByte((byte)SSH_MSG_CHANNEL_WINDOW_ADJUST);
            buf.putInt(channel.getRecipient());
            buf.putInt(channel.lwsize_max-channel.lwsize);
            synchronized(channel){
                if(!channel.close)
                    write(packet);
            }
        }
        channel.setLocalWindowSize(channel.lwsize_max);
    break;
    case SSH_MSG_CHANNEL_EXTENDED_DATA:
        buf.getInt();
        buf.getShort();
        i=buf.getInt();
        channel=Channel.getChannel(i, this);
        buf.getInt(); // data_type_code == 1
        foo=buf.getString(start, length);
        //System.err.println("stderr: "+new String(foo,start[0],length[0]));
    break;
}
if(channel==null){
    break;
}

if(length[0]==0){
    break;
}

channel.write_ext(foo, start[0], length[0]);

len=length[0];
channel.setLocalWindowSize(channel.lwsize-len);
if(channel.lwsize<channel.lwsize_max/2){
    packet.reset();
    buf.putByte((byte)SSH_MSG_CHANNEL_WINDOW_ADJUST);
    buf.putInt(channel.getRecipient());
    buf.putInt(channel.lwsize_max-channel.lwsize);
    synchronized(channel){
        if(!channel.close)
            write(packet);
    }
    channel.setLocalWindowSize(channel.lwsize_max);
}

break;

case SSH_MSG_CHANNEL_WINDOW_ADJUST:
    buf.getInt();
    buf.getShort();
    i=buf.getInt();
    channel=Channel.getChannel(i, this);
    if(channel==null){
        break;
    }
    channel.addRemoteWindowSize(buf.getInt());
    break;

case SSH_MSG_CHANNEL_EOF:
    buf.getInt();
    buf.getShort();
    i=buf.getInt();
    channel=Channel.getChannel(i, this);
    if(channel!=null){
        //channel.eof_remote=true;
        //channel.eof();
        channel.eof_remote();
    }
    break;

case SSH_MSG_CHANNEL_CLOSE:
    buf.getInt();
    buf.getShort();
    i=buf.getInt();
    channel=Channel.getChannel(i, this);
    if(channel!=null){
        // channel.close();
        channel.disconnect();
    }
    /*
     * if(Channel.pool.size()==0){
     *     thread=null;
     *     */
    break;

case SSH_MSG_CHANNEL_OPEN_CONFIRMATION:
buf.getInt();
buf.getShort();
i=buf.getInt();
channel=Channel.getChannel(i, this);
if(channel==null){
    //break;
}
int r=buf.getInt();
long rws=buf.getUInt();
tvo*=buf.getInt();
channel.setRemoteWindowSize(rws);
channel.setRemotePacketSize(rps);
channel.open_confirmation=true;
channel.setRecipient(r);
bvreak;
case SSH_MSG_CHANNEL_OPEN_FAILURE:
buf.getInt();
buf.getShort();
i=buf.getInt();
channel=Channel.getChannel(i, this);
if(channel==null){
    //break;
    int reason_code=buf.getInt();
    //foo=buf.getString(); // additional textual information
    //foo=buf.getString(); // language tag
    channel.setExitStatus(reason_code);
    channel.close=true;
    channel.eof_remote=true;
    channel.setRecipient(0);
bvreak;
case SSH_MSG_CHANNEL_REQUEST:
buf.getInt();
buf.getShort();
foo=buf.getString();
boolean reply=(buf.getByte()!=0);
channel=Channel.getChannel(i, this);
if(channel!=null){
    byte reply_type=(byte)SSH_MSG_CHANNEL_FAILURE;
    if((Util.byte2str(foo)).equals("exit-status")
        i=buf.getInt();
        // exit-status
        channel.setExitStatus(i);
        reply_type=(byte)SSH_MSG_CHANNEL_SUCCESS;
    }
    if(reply){
        packet.reset();
        buf.putByte(reply_type);
        buf.putInt(channel.getRecipient());
        write(packet);
    }
} else{
    bbreak;
}
case SSH_MSG_CHANNEL_OPEN:
buf.getInt();
buf.getShort();
foo=buf.getString();
String ctyp=Util.byte2str(foo);
if("*forwarded-tcpip*.equals(ctyp) &&
   !"*x11*.equals(ctyp) && x11_forwarding) &&
   !("*auth-agent@openssh.com*.equals(ctyp) && agent_forwarding) ||
   //System.err.println("Session.run: CHANNEL OPEN *"+ctyp);
   //throw new IOException("Session.run: CHANNEL OPEN *"+ctyp);
   packet.reset();
   buf.putByte((byte)SSH_MSG_CHANNEL_OPEN_FAILURE);
   buf.putInt(buf.getInt());
buf.putInt(SSH.OPEN_ADMINISTRATIVELY_PROHIBITED);
buf.putString(Util.empty);
buf.putString(Util.empty);
write(packet);
}

else{
channel=Channel.getChannel(ctyp);
addChannel(channel);
channel.getData(buf);
channel.init();

Thread tmp=new Thread(channel);
tmp.setName("Channel "+ctyp+" *host");
if(daemon_thread){
tmp.setDaemon(daemon_thread);
}
tmp.start();
break;
}

case SSH_MSG_CHANNEL_SUCCESS:
buf.getInt();
buf.getShort();
i=buf.getInt();
channel=Channel.getChannel(i, this);
if(channel==null){
break;
}
channel.reply=1;
break;

case SSH_MSG_CHANNEL_FAILURE:
buf.getInt();
buf.getShort();
i=buf.getInt();
channel=Channel.getChannel(i, this);
if(channel==null){
break;
}
channel.reply=0;
break;
}
case SSH_MSG_GLOBAL_REQUEST:
buf.getInt();
buf.getShort();
foo=buf.getString(); // request name
reply=(buf.getByte()!=0);
if(reply){
packet.reset();
buf.putByte((byte)SSH.MSG_REQUEST_FAILURE);
write(packet);
}
break;

case SSH_MSG_REQUEST_FAILURE:
break;

case SSH_MSG_REQUEST_SUCCESS:
Thread t=grr.getThread();
if(t!=null){
grr.setReply(msgType==SSH.MSG_REQUEST_SUCCESS? 1 : 0);
t.interrupt();
}
break;

default:
//System.err.println("Session.run: unsupported type "+msgType);
throw new IOException("Unknown SSH message type "+msgType);
}
"Caught an exception, leaving main loop due to " + e
               .getMessage());
   }
  // System.err.println("# Session.run");
  // e.printStackTrace();
 }
 try{
   disconnect();
 }
 catch(NullPointerException e){
   // System.err.println("@1");
   // e.printStackTrace();
 }
 catch(Exception e){
   // System.err.println("@2");
   // e.printStackTrace();
 }
 isConnected=false;
}

public void disconnect(){
 if(!isConnected) return;
 // System.err.println(this+": disconnect");
 // Thread.dumpStack();
 if(JSch.getLogger().isEnabled(Logger.INFO)){
   JSch.getLogger().log(Logger.INFO,
      "Disconnecting from "+host+" port "+port);
    } /*
   for(int i=0; i<Channel.pool.size(); i++){
     try{
       Channel c=((Channel)(Channel.pool.elementAt(i)));
       if(c.session==this) c.eof();
     } catch(Exception e){
       }
   } */
   Channel.disconnect(this);
   isConnected=false;
   PortWatcher.delPort(this);
   ChannelForwardedTCP/IP.delPort(this);
   ChannelX11.removeFakedCookie(this);
   synchronized(lock){
     if(connectThread!=null){
       Thread.yield();
       connectThread.interrupt();
       connectThread=null;
     }
     thread=null;
   try{
     if(io!=null){
       if(io.in!=null) io.in.close();
       if(io.out!=null) io.out.close();
       if(io.out_ext!=null) io.out_ext.close();
     }
     if(proxy==null){
       if(socket!=null)
       socket.close();
     } else{
       synchronized(proxy){
         proxy.close();
       }
   }
proxy=null;

}  
// }  
// synchronized(jsch.pool)

jsch.removeSession(this);

// System.gc();

public int setPortForwardingL(int lport, String host, int rport) throws JSchException{
    return setPortForwardingL("127.0.0.1", lport, host, rport);
}

public int setPortForwardingL(String boundaddress, int lport, String host, int rport) throws JSchException{
    //ServerSocketFactory ssf) throws JSchException{
        //PortWatcher pw=PortWatcher.addPort(this, boundaddress, lport, host, rport, ssf);
        tmp.setName("PortWatcher Thread for "+host);
        if(daemon_thread){
            tmp.setDaemon(daemon_thread);
        }
        tmp.start();
        return pw.lport;
    }
}

public void delPortForwardingL(int lport) throws JSchException{
    delPortForwardingL("127.0.0.1", lport);
}

public void delPortForwardingL(String boundaddress, int lport) throws JSchException{
    PortWatcher.delPort(this, boundaddress, lport);
}

public String[] getPortForwardingL() throws JSchException{
    return PortWatcher.getPortForwarding(this);
}

public void setPortForwardingR(int rport, String host, int lport) throws JSchException{
    setPortForwardingR(null, rport, host, lport, (SocketFactory)null);
}

public void setPortForwardingR(String bind_address, int rport, String host, int lport) throws JSchException{
    setPortForwardingR(bind_address, rport, host, lport, (SocketFactory)null);
}

public void setPortForwardingR(int rport, String host, int lport, SocketFactory sf) throws JSchException{
    setPortForwardingR(null, rport, host, lport, sf);
}

public void setPortForwardingR(String bind_address, int rport, String host, int lport, SocketFactory sf) throws JSchException{
    ChannelForwardedTCPIP.addPort(this, bind_address, rport, host, lport, sf);
    setPortForwarding(bind_address, rport);
}

public void setPortForwardingR(int rport, String daemon) throws JSchException{
    setPortForwardingR(null, rport, daemon, null);
}

public void setPortForwardingR(int rport, String daemon, Object[] arg) throws JSchException{
    setPortForwardingR(null, rport, daemon, arg);
}
public void setPortForwardingR(String bind_address, int rport, String daemon, Object[] arg) throws JSchException{
    ChannelForwardedTCPIP.addPort(this, bind_address, rport, daemon, arg);
    setPortForwarding(bind_address, rport);
}

private class GlobalRequestReply{
    private Thread thread=null;
    private int reply=-1;
    void setThread(Thread thread){
        this.thread=thread;
        this.reply=-1;
    }
    Thread getThread(){ return thread; } 
    void setReply(int reply){ this.reply=reply; } 
    int getReply(){ return this.reply; } }

private GlobalRequestReply grr=new GlobalRequestReply();
private void setPortForwarding(String bind_address, int rport) throws JSchException{
    synchronized(grr){
        Buffer buf=new Buffer(100); // ??
        Packet packet=new Packet(buf);
        String address_to_bind=ChannelForwardedTCPIP.normalize(bind_address);
        grr.setThread(Thread.currentThread());
        try{
            // byte SSH_MSG_GLOBAL_REQUEST 80
            // string "tcpip-forward"
            // boolean want_reply
            // string address_to_bind
            // uint32 port number to bind
            packet.reset();
            buf.putByte((byte) SSH_MSG_GLOBAL_REQUEST);
            buf.putString(Util.str2byte("tcpip-forward"));
            buf.putByte((byte)1);
            buf.putString(Util.str2byte(address_to_bind));
            buf.putInt(rport);
            write(packet);
        } catch(Exception e){
            grr.setThread(null);
            if(e instanceof Throwable)
                throw new JSchException(e.toString(), (Throwable)e);
            throw new JSchException(e.toString());
        }
        int count = 0;
        int reply = grr.getReply();
        while(count < 10 && reply == -1){
            try{ Thread.sleep(1000); } 
            catch(Exception e){
                count++;
                reply = grr.getReply();
            }
            grr.setThread(null);
            if(reply != 1){
                throw new JSchException("remote port forwarding failed for listen port "+rport);
            }
        }
    }
}
public void delPortForwardingR(int rport) throws JSchException{
    ChannelForwardedTCPIP.delPort(this, rport);
}
private void initDeflater(String method) throws JSchException{
if(method.equals("none")){
deflater=null;
return;
}

String foo=getConfig(method);
if(foo!=null){
  if(method.equals("zlib") ||
    (isAuthed && method.equals("zlib@openssh.com"))){
    try{
      Class c=Class.forName(foo);
deflater=(Compression)(c.newInstance());
      int level=6;
      try{ level=Integer.parseInt(getConfig("compression_level"));}
      catch(Exception ee){ }
deflater.init(Compression.DEFLATER, level);
    } catch(NoClassDefFoundError ee){
      throw new JSchException(ee.toString(), ee);
    } catch(Exception ee){
      throw new JSchException(ee.toString(), ee);
    } //System.err.println(foo+" isn’t accessible.");
  }
}

private void initInflater(String method) throws JSchException{
  if(method.equals("none")){
    inflater=null;
    return;
  }
  String foo=getConfig(method);
  if(foo!=null){
    if(method.equals("zlib") ||
      (isAuthed && method.equals("zlib@openssh.com"))){
      try{
        Class c=Class.forName(foo);
        inflater=(Compression)(c.newInstance());
        inflater.init(Compression.INFLATER, 0);
      } catch(Exception ee){
        throw new JSchException(ee.toString(), ee);
      } //System.err.println(foo+" isn’t accessible.");
    }
  }
}

void addChannel(Channel channel){
  channel.setSession(this);
}

public void setProxy(Proxy proxy){ this.proxy=proxy; }
public void setHost(String host){ this.host=host; }
public void setPort(int port){ this.port=port; }
void setUserName(String username){ this.username=username; }
public void setUserInfo(UserInfo userinfo){ this.userinfo=userinfo; }
public UserInfo getUserInfo(){ return userinfo; }
public void setInputStream(InputStream in){ this.in=in; }
public void setInputStream(OutputStream out){ this.out=out; }
public void setX11Host(String host){ ChannelX11.setHost(host); }
public void setX11Port(int port){ ChannelX11.setPort(port); }
public void setX11Cookie(String cookie){ ChannelX11.setCookie(cookie); }
public void setPassword(String password){
  if(password!=null)
    this.password=Util.str2byte(password);
}
public void setPassword(byte[] password){
  if(password!=null){
    this.password=new byte[password.length];
  }
}
    System.arraycopy(password, 0, this.password, 0, password.length);
    }
    }
    }
    public void setConfig(java.util.Properties newconf){
      setConfig((java.util.Hashtable)newconf);
    }
    public void setConfig(java.util.Hashtable newconf){
      synchronized(lock){
        if(config==null)
          config=new java.util.Hashtable();
        for(java.util.Enumeration e=newconf.keys() ; e.hasMoreElements() ;)
          String key=(String)(e.nextElement());
        config.put(key, (String)(newconf.get(key)));
      }
    }
    public void setConfig(String key, String value){
      synchronized(lock){
        if(config==null){
          config=new java.util.Hashtable();
        }
        config.put(key, value);
      }
    }
    public String getConfig(String key){
      Object foo=null;
      if(config!=null){
        foo=config.get(key);
        if(foo instanceof String) return (String)foo;
      }
      foo=jsch.getConfig(key);
      if(foo instanceof String) return (String)foo;
      return null;
    }
    public void setSocketFactory(SocketFactory sfactory){
      socket_factory=sfactory;
    }
    public boolean isConnected(){ return isConnected; }
    public int getTimeout(){ return timeout; }
    public void setTimeout(int timeout) throws JSchException {
      if(socket==null){
        if(timeout<0){
          throw new JSchException("invalid timeout value");
        }
        this.timeout=timeout;
        return;
      }
      try{
        socket.setSoTimeout(timeout);
        this.timeout=timeout;
      }
      catch(Exception e){
        if(e instanceof Throwable)
          throw new JSchException(e.toString(), (Throwable)e);
        throw new JSchException(e.toString());
      }
      public String getServerVersion(){
        return Util.byte2str(V_S);
      }
      public String getClientVersion(){
        return Util.byte2str(V_C);
      }
    public void setClientVersion(String cv){
V_C = Util.str2byte(cv);
}

public void sendIgnore() throws Exception {
    Buffer buf = new Buffer();
    Packet packet = new Packet(buf);
    packet.reset();
    buf.putByte((byte) SSH_MSG_IGNORE);
    write(packet);
}

private static final byte[] keepalivemsg = Util.str2byte("keepalive@jcraft.com");

public void sendKeepAliveMsg() throws Exception {
    Buffer buf = new Buffer();
    Packet packet = new Packet(buf);
    packet.reset();
    buf.putByte((byte) SSH_MSG_GLOBAL_REQUEST);
    buf.putString(keepalivemsg);
    buf.putByte((byte) 1);
    write(packet);
}

private HostKey hostkey = null;

public HostKey getHostKey() { return hostkey; }

public String getHost() { return host; }

public String getUserName() { return username; }

public int getPort() { return port; }

public void setHostKeyAlias(String hostKeyAlias) {
    this.hostKeyAlias = hostKeyAlias;
}

public String getHostKeyAlias() {
    return hostKeyAlias;
}

/**
 * Sets the interval to send a keep-alive message. If zero is specified, any keep-alive message must not be sent. The default interval is zero.
 * @param interval the specified interval, in milliseconds.
 * @see #getServerAliveInterval()
 */
public void setServerAliveInterval(int interval) throws JSchException {
    setTimeout(interval);
    this.serverAliveInterval = interval;
}

/**
 * Returns setting for the interval to send a keep-alive message.
 * @see #setServerAliveInterval(int)
 */
public int getServerAliveInterval() {
    return this.serverAliveInterval;
}

/**
 * Sets the number of keep-alive messages which may be sent without receiving any messages back from the server. If this threshold is reached while keep-alive messages are being sent, the connection will be disconnected. The default value is one.
 * @param count the specified count
 * @see #getServerAliveCountMax()
 */
public void setServerAliveCountMax(int count) {
    this.serverAliveCountMax = count;
}

/**
 * Returns setting for the threshold to send keep-alive messages.
 * @see #getServerAliveCountMax(int)
 */
public int getServerAliveCountMax()
{
    return this.serverAliveCountMax;
}

public void setDaemonThread(boolean enable)
{
    this.daemon_thread=enable;
}

private String[] checkCiphers(String ciphers)
{
    if(ciphers==null || ciphers.length()==0)
        return null;

    if(JSch.getLogger().isEnabled(Logger.INFO))
    {
        JSch.getLogger().log(Logger.INFO,
            "CheckCiphers: "+ciphers);
    }

    java.util.Vector result=new java.util.Vector();
    String[] _ciphers=Util.split(ciphers, ",");
    for(int i=0; i<_ciphers.length; i++)
    {
        if(!checkCipher(getConfig(_ciphers[i])))
            result.addElement(_ciphers[i]);
    }

    if(result.size()==0)
        return null;

    String[] foo=new String[result.size()];
    System.arraycopy(result.toArray(), 0, foo, 0, result.size());

    if(JSch.getLogger().isEnabled(Logger.INFO))
    {
        for(int i=0; i<foo.length; i++)
        {
            JSch.getLogger().log(Logger.INFO,
                foo[i]+" is not available.");
        }
    }

    return foo;
}

static boolean checkCipher(String cipher)
{
    try
    {
        Class c=Class.forName(cipher);
        Cipher _c=(Cipher)c.newInstance();
        _c.init(Cipher.ENCRYPT_MODE,
            new byte[_c.getBlockSize()],
            new byte[_c.getIVSize()];
        return true;
    }
    catch(Exception e)
    {
        return false;
    }
}

private String[] checkKexes(String kexes)
{
    if(kexes==null || kexes.length()==0)
        return null;

    if(JSch.getLogger().isEnabled(Logger.INFO))
    {
        JSch.getLogger().log(Logger.INFO,
            "CheckKexes: "+kexes);
    }

    java.util.Vector result=new java.util.Vector();
    String[] _kexes=Util.split(kexes, ",");
    for(int i=0; i<_kexes.length; i++)
    {
        if(!checkKex(this, getConfig(_kexes[i])))
            result.addElement(_kexes[i]);
    }

    return foo;
}
if(result.size()==0)
    return null;
String[] foo=new String[result.size()];
System.arraycopy(result.toArray(), 0, foo, 0, result.size());
if(JSch.getLogger().isEnabled(Logger.INFO)){
    for(int i=0; i<foo.length; i++){
        JSch.getLogger().log(Logger.INFO,
            foo[i]+" is not available.");
    }
}
return foo;

static boolean checkKex(Session s, String kex){
    try{
        Class c=Class.forName(kex);
        KeyExchange _c=(KeyExchange)(c.newInstance());
        _c.init(s ,null, null, null, null);
        return true;
    }
    catch(Exception e){ return false; }
}

/**
 * Sets the identityRepository, which will be referred
 * in the public key authentication. The default value is null.
 *
 * @param identityRepository
 * @see #getIdentityRepository
 */
public void setIdentityRepository(IdentityRepository identityRepository){
    this.identityRepository = identityRepository;
}

/**
 * Gets the identityRepository. If this.identityRepository is null,
 * JSch#getIdentityRepository() will be invoked.
 *
 * @see JSch#getIdentityRepository()
 */
IdentityRepository getIdIdentityRepository(){
    if(identityRepository == null)
        return jsch.getIdentityRepository();
    return identityRepository;
}
A.3 NewSession.java

Test Wrapper for the manual testing

```java
package com.jcraft.jsch;

/**
 * Test Wrapper for the manual testing
 * @author Mirjam van Nahmen <m.vannahmen@student.ru.nl>
 * @version 1.0
 * @since 2014-01-31
 */
import java.io.IOException;
import java.util.logging.Level;
import java.util.logging.Logger;

public class NewSession {
    static JSch jsch;
    static Session session;

    public static void main(String[] args) {
        try {
            System.out.println("begin");
            jsch = new JSch();
            session = new Session(jsch);
            session.setUserName("Miri");
            session.setPassword("Hallo");
            session.setHostKeyAlias("localhost");
            session.setHost("127.0.0.1");
            java.util.Properties config = new java.util.Properties();
            config.put("StrictHostKeyChecking", "no");
            session.setConfig(config);
            System.out.println("Start...");

            System.out.println("Connected...");
            System.out.println("Good Server version(0) otherwise(-1): "+version());
            System.out.println("KexInit1: "+MSGToString(kexinit()));
            System.out.println("NewKeys: "+MSGToString(newKeys()));
            System.out.println("Service_Request_accepted: "+MSGToString(service_accepted()));
            session.disconnect();
        }
        catch (JSchException ex) {
            session.disconnect();
            Logger.getLogger(NewSession.class.getName()).log(Level.SEVERE, null, ex);
        }
    }
}
```
Connect to the server.
* Only with default values, otherwise no connection
* @return 0 if the connection is established
**/
public static String connect() throws JSchException, IOException, Exception{
    session.connect();
    return "Connected";
}
/**
 * send the version number
 * @return if the right version is send
 **/
public static String version() throws IOException, JSchException{
    byte[] V_C=Util.str2byte("SSH-2.0-JSCH-"+JSch.VERSION);
    return session.version(session.i, session.j,V_C);
}
/**
 * send kexinit with manual settings
 * @param kex_contant: default=0
 * @param server_host_key_algorithms_contant: default=0
 * @param cipher_c2s_contant: default=0
 * @param cipher_s2c_contant: default=0
 **/
public static int kexinit() throws Exception{
    return session.kexinit(0,0,0,0);
}
/**
 * send the KEXINIT_DH
 * @return the message number
 **/
public static int kexinitDH() throws Exception{
    return session.kexDH_init();
}
/**
 * send the SERVICE_ACCEPTED
 * @return the message number
 **/
private static int service_accepted() throws JSchException, Exception {
    return session.service_accepted();
}
/**
 * send the NEWKEY
 * @return the message number
 **/
private static int newKeys() throws JSchException, Exception {
    return session.newKey();
}
/**
 * convert the integer of the message number to the associated String
 * @param int msg
 * @return String SSH-MSG-XXX
 **/
public static String MSGToString(int msg){
    switch(msg){
    case 1: return "SSH_MSG_DISCONNECT = 1";
    case 2: return "SSH_MSG_IGNORE = 2";
    case 3: return "SSH_MSG_UNIMPLEMENTED = 3";
    case 4: return "SSH_MSG_DEBUG = 4";
    case 5: return "SSH_MSG_SERVICE_REQUEST = 5";
    case 6: return "SSH_MSG_SERVICE_ACCEPT = 6";
    case 20: return "SSH_MSG_KEXINIT = 20";
    case 21: return "SSH_MSG_NEWKEYS = 21";
    case 30: return "SSH_MSG_KEXDH_INIT = 30";
case 31: return "SSH_MSG_KEXDH_REPLY = 31";
case -5: return "sendVersion";
default: return "something else"+msg;
}