emSSH

Secure Shell

User Guide & Reference Manual

Document: UM20001
Software Version: 2.52.0
Revision: 0
Date: June 1, 2022

A product of SEGGER Microcontroller GmbH

www.segger.com

Disclaimer

The information written in this document is assumed to be accurate without guarantee. The information in this manual is subject to change for functional or performance improvements without notice. SEGGER Microcontroller GmbH (SEGGER) assumes no responsibility for any errors or omissions in this document. SEGGER disclaims any warranties or conditions, express, implied or statutory for the fitness of the product for a particular purpose. It is your sole responsibility to evaluate the fitness of the product for any specific use.

Copyright notice

Trademarks

Names mentioned in this manual may be trademarks of their respective companies.

Brand and product names are trademarks or registered trademarks of their respective holders.

Contact address

SEGGER Microcontroller GmbH

Ecolab-Allee 5
D-40789 Monheim am Rhein

Germany

Tel.	+49 2173-99312-0
Fax.	+49 2173-99312-28
E-mail:	support@segger.com[*]
Internet:	www.segger.com

---

[*]By sending us an email your (personal) data will automatically be processed. For further information please refer to our privacy policy which is available at https://www.segger.com/legal/privacy-policy/.

Manual versions

This manual describes the current software version. If you find an error in the manual or a problem in the software, please report it to us and we will try to assist you as soon as possible.

Print date: June 1, 2022

Software	Revision	Date	By	Description
2.54	0	220601	PC	Update to latest software version.
2.52	0	210422	PC	Update to latest software version.
2.50	0	190610	PC	Chapter "Secure copy (add-on)" • Added. Chapter "Core API reference" • Added "Data Types" section. • Added `SSH_WINDOW_CHANGE_ParseParas()`. Chapter "Configuring emSSH" • Added `curve448-sha512` key exchange. • Added `diffie-hellman-group15-sha512` key exchange. • Added `diffie-hellman-group17-sha512` key exchange.
2.48	0	190311	PC	Update to latest software version.
2.46	0	181010	PC	Chapter "Configuring cryptography" • Added iMX RT10xx DCP coprocessor.
2.44	0	180621	PC	Chapter "Configuring cryptography" • Added STM32 AES coprocessor. • Added STM32 HASH coprocessor.
2.42	0	171116	PC	Update to latest software version.
2.40	0	170823	PC	Chapter "Configuring emSSH" • Added Curve25519 key exchange. • Added RSA key exchanges. • Added DH-GEX key exchanges. • Added ChaCha20-Poly1305 encryption. Chapter "API reference" • Added `SSH_GetVersionText()`. • Added `SSH_GetCopyrightText()`. Chapter "Compatibility" • Updated compatibility matrices.
2.30	0	170728	PC	Chapter "Using emSSH" • Added window change request setup. Chapter "Configuring emSSH" • Added additional key exchanges. • Added additional ciphers. Chapter "API reference" • Added preprocessor symbols. Chapter "Compatibility" • Updated compatibility matrices. Index "Subject index" • Added.
2.20	0	170518	PC	Chapter "Configuring emSSH" • Updated for using shared emCrypt component. Chapter "Configuring cryptography" • Completely rewritten for emCrypt.
2.10a	0	160804	PC	Update to synchronize with emSSL 2.30 release.
2.10	0	160621	PC	Initial release.
1.00	0	150725	PC	Internal release.

4

About this document

Assumptions

This document assumes that you already have a solid knowledge of the following:

- The software tools used for building your application (assembler, linker, C compiler).
- The C programming language.
- The target processor.
- DOS command line.

If you feel that your knowledge of C is not sufficient, we recommend *C: A Reference Manual* by Harbison and Steele (ISBN 0--13--089592X). This book provides a complete description of the C language, the run-time libraries, and a style of C programming that emphasizes correctness, portability, and maintainability.

How to use this manual

This manual explains all the functions and macros that the product offers. It assumes you have a working knowledge of the C language. Knowledge of assembly programming is not required.

Typographic conventions for syntax

This manual uses the following typographic conventions:

Style	Used for
Body	Body text.
Parameter	Parameters in API functions.
Sample	Sample code in program examples.
Sample comment	Comments in program examples.
User Input	Text entered at the keyboard by a user in a session transcript.
Secret Input	Text entered at the keyboard by a user, but not echoed (e.g. password entry), in a session transcript.
Reference	Reference to chapters, sections, tables and figures.
Emphasis	Very important sections.
SEGGER home page	A hyperlink to an external document or web site.

6

Table of contents

Chapter 1

Introduction to emSSH

This section presents an overview of emSSH, its structure, and its capabilities.

1.1   What is emSSH?

emSSH is a software library that enables you to create secure connections between a client and a server, typically over the Internet using TCP/IP.

Although SSH is usually associated with secure connections to a server using TCP/IP, you can run an SSH session over any bidirectional channel, for instance a serial line or wireless link, and provide a secure connection.

emSSH is both hardware independent and transport independent, and integrates seamlessly with embOS/IP. emSSH supports SSH version 2 only.

1.2   Design goals

emSSH is designed with the following goals in mind:

- Highly modular such that unused features are never linked into the application.
- Be completely runtime configurable, adding each modular feature as needed.
- Present a simple user-level API that is easy to use without extensive setup.
- Easy to maintain both by SEGGER and anybody with access to the sources.
- Conform to all necessary standards and current best practices.
- Be efficient both in terms of resource usage and execution speed.
- Target 8-bit to 32-bit processors with limited resources as well as workstations.
- Use the the SEGGER Cryptographic Toolkit, the foundation of all SEGGER security products.

We believe all design goals are achieved by emSSH.

1.3 Features

emSSH is written in ANSI C and can be used on virtually any CPU. Here is a list of emSSH features:

- ISO/ANSI C source code.
- High performance.
- Small footprint.
- Runs "out-of-the-box".
- Highly compact implementation runs effortlessly on single-chip MCUs.
- Easy-to-understand and simple-to-use API.
- Simple configuration.
- Secure any open channel (e.g. serial line or wireless link).
- Wide range of encryption schemes, key exchange schemes, for interoperability with popular clients.
- Modular architecture links only what you need.
- Plug-in hardware acceleration.
- Broad support for confidentiality, integrity and authentication.
- Diffie-Hellman Ephemeral supporting forward secrecy.
- Elliptic curve key agreement for reduced key sizes.
- Royalty-free.

1.4   Package content

emSSH is provided in source code and contains everything required. The following table shows the content of the emSSH package:

Files	Description
`Application`	emSSH sample applications for bare metal and embOS.
`Config`	Configuration header files.
`CRYPTO`	Shared cryptographic toolkit source code.
`Doc`	emSSH documentation.
`Sample/Config`	Example emSSH user configuration.
`SEGGER`	SEGGER software component source code used in emSSH.
`SSH`	emSSH implementation source code.

Chapter 2

Exploring emSSH

This chapter describes how to try out emSSH on a PC and embedded hardware with minimal effort. We highly recommend that you try out a working version of emSSH, shipped by SEGGER, with a known-good setup, preferably on an emPower board, before attempting to add it to your own application.

2.1   Using a PC to try emSSH

emSSH is shipped with a precompiled example that demonstrate a simple SSH shell. You can run the example and connect to the local SSH server on port 22.

```
C:> SSH_Shell5.exe

emSSH V2.40 - Shell5 compiled Jun  5 2017 11:52:46
(c) 2015-2017 SEGGER Microcontroller GmbH    www.segger.com

Waiting for connection on port 22.
_
```

When you run this, Windows Firewall will present a dialog asking whether to grant network access to the shell application. Proceed and grant access otherwise you will not be able to log into the shell.

Once the shell is waiting for a connection, you can connect using your preferred client. In this example we will use Tera Term. Run Tera Term and select a new connection to the local PC:
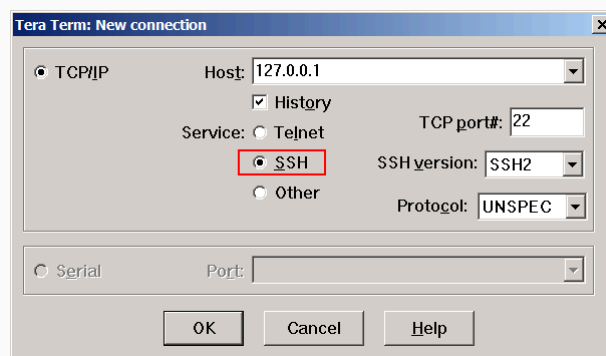
Tera Term New Connection dialog

Be sure to select SSH, not Telnet, as the protocol. You will notice that the server has accepted the connection and is starting the shell:

```
C:> SSH_Shell5.exe

emSSH V2.40 - Shell5 compiled Jun  5 2017 11:52:46
(c) 2015-2017 SEGGER Microcontroller GmbH    www.segger.com

Waiting for connection on port 22.
Connection accepted, starting shell.
_
```

The server requires authentication of the user, and Tera Term presents a user authentication dialog. Enter the user name "admin" with password "secret" to log on:
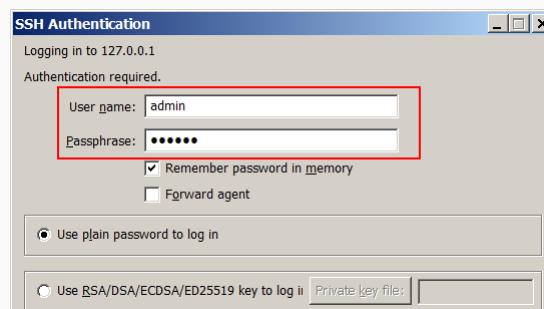
Tera Term Authentication dialog

Tera Term then opens up a terminal window:
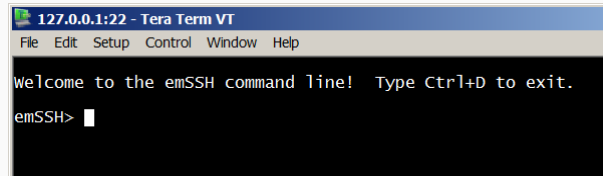
Tera Term terminal and shell

From here you can type commands into a virtual command line. There is nothing behind the command line, it only acts as a demonstration of emSSH. To exit the shell, type **Ctrl +D**, which also causes Tera Term to close.

At the server end, the connection is closed and it waits for a new connection. Type **Ctrl +C** to close the emSSH server.

```
C:> SSH_Shell5.exe

emSSH V2.40 - Shell5 compiled Jun  5 2017 11:52:46
(c) 2015-2017 SEGGER Microcontroller GmbH    www.segger.com

Waiting for connection on port 22.
Connection accepted, starting shell.
Connection closed.
Waiting for connection on port 22.
^C
C:> _
```

2.2   Moving to embedded hardware

When starting to run emSSH on embedded hardware, we recommend that you use one of the supplied "Start" projects for your target system to begin with and gain confidence with a working system before progressing to add emSSH to your own application.

The following sections describe this process using SEGGER Embedded Studio, but the principles are the same for any embedded development or workstation environment. The target hardware is an SEGGER emPower board which is supplied with Embedded Studio PRO or available separately from SEGGER and through authorized distributors.

Start Embedded Studio and load the SEGGER emPower start project:
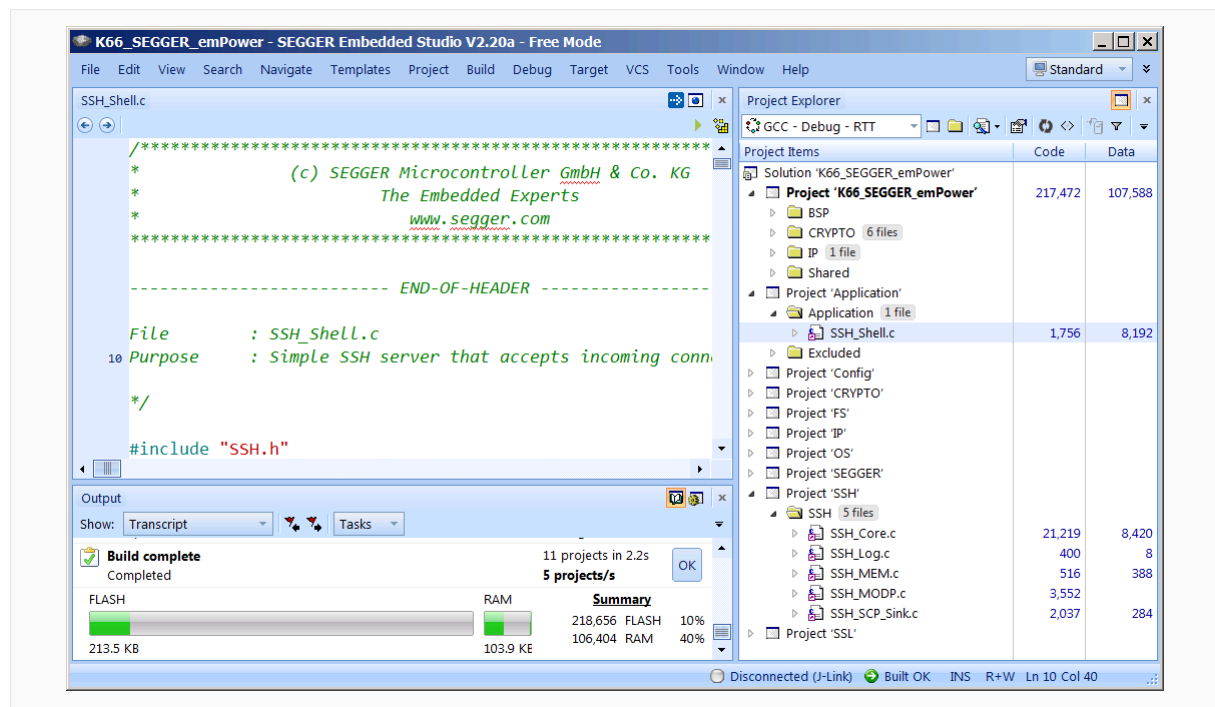
Start project in Embedded Studio

Once you have loaded your start project, you can test it out by choosing **Debug > Go** which flashes it into your target and starts running it under control of the debugger.

The **Debug Terminal** will show the configured IP address of the emPower board and you will be able to use Tera Term to log into the SSH server in the same way as the PC application above. If you do not see the **Debug Terminal**, choose **View > Debug Terminal**. The terminal output will look something similar to this:
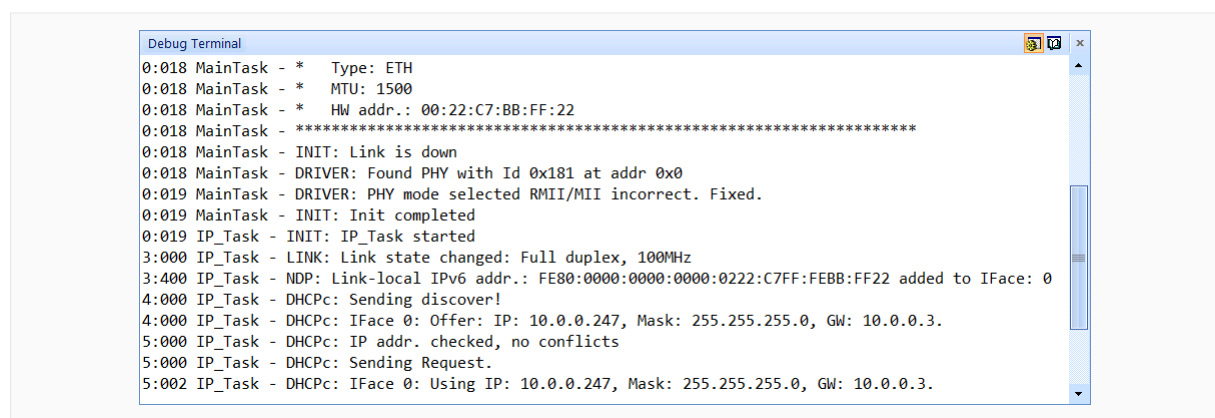
Log output in Debug Terminal

emSSH will then display its configuration and indicate that it's waiting for a connection:

```
Debug Terminal                                                          ×
5:021 MainTask - *   hmac-sha2-256-etm@openssh.com                        ▲
5:021 MainTask - *   hmac-sha2-512-etm@openssh.com
5:021 MainTask - *   hmac-ripemd160-etm@openssh.com
5:021 MainTask - *   hmac-sha224@ssh.com
5:021 MainTask - *   hmac-sha256-2@ssh.com
5:021 MainTask - *   hmac-sha384@ssh.com
5:021 MainTask - *   hmac-sha512@ssh.com
5:021 MainTask - *
5:022 MainTask - * Compression algorithms:
5:022 MainTask - *   none
5:022 MainTask - *
5:022 MainTask - ******************************************************************

(c) 2015-2016 SEGGER Microcontroller GmbH & Co. KG     www.segger.com
emSSH Simple Secure Shell V2.01 compiled Jun  5 2016 13:16:56

Waiting for connection on port 22.                                       ▼
```

emSSH initialized and waiting for a connection

At this point you will be able to use Tera Term, or your selected SSH client, to log into the emPower board.

Chapter 3

Using emSSH

This section describes how to configure emSSH for use and set up a shell connection using a sample project. The sample project can be customized and integrated into your application.

In this section we assume that you have a fully-functioning embOS/IP project that is able to connect to the network and all that is required is to add emSSH to the project.

3.1   Sample applications

emSSH ships with a number of sample applications that demonstrate how to integrate shell capability into your application. Each sample application adds an additional capability to the shell and is a small incremental step from the previous version.

The sample applications are:

Application	Description
SSH_Shell1.c	Minimal shell application.
SSH_Shell2.c	Adds a functional command line.
SSH_Shell3.c	Adds user authentication by password.
SSH_Shell4.c	Strengthens password storage.
SSH_Shell5.c	Displays a warning banner before login.
SSH_Shell6.c	Uses an RTOS to service multiple shells.

3.1.1   A note on the samples

Each sample that we present in this section is written in a style that makes it easy to describe and that fits comfortably within the margins of printed paper. Therefore, it may well be that you would rewrite the sample to have a slightly different structure that fits better, but please keep in mind that these examples are written with clarity as the prime objective, and to that end we sacrifice some brevity and efficiency.

3.1.2   Where to find the sample code

All samples are included in the Application directory of the emSSH distribution.

3.2   A minimal shell

The first application, `SSH_Shell1.c`, provides the capability for a user to log into the shell with no authentication. The terminal on the end of the shell does nothing more than echo incoming data from the terminal.

For a complete listing of this application, see *SSH_Shell1.c complete listing* on page 34.

3.2.1   Application entry

The main application task is responsible for setting up the environment ready to accept incoming SSH requests. This is simply boilerplate code that has no configuration:

```
void MainTask(void) {
  SSH_SESSION * pSession;
  int          BoundSocket;
  int          Socket;
  int          Status;
  //
  SEGGER_SYS_Init();    ❶
  SEGGER_SYS_IP_Init();
  SSH_Init();    ❷
  //
  // Allow "none" user authentication.
  //
  SSH_SERVICE_Add(&SSH_SERVICE_USERAUTH, 0);    ❸
  SSH_USERAUTH_METHOD_Add(&SSH_USERAUTH_METHOD_NONE, _UserauthRequestNone);
  //
  // Add support for interactive shells.
  //
  SSH_CHANNEL_REQUEST_Add(&SSH_CHANNEL_REQUEST_SHELL,         NULL);    ❹
  SSH_CHANNEL_REQUEST_Add(&SSH_CHANNEL_REQUEST_ENV,           NULL);
  SSH_CHANNEL_REQUEST_Add(&SSH_CHANNEL_REQUEST_PTYREQ,        _TerminalRequest);
  SSH_CHANNEL_REQUEST_Add(&SSH_CHANNEL_REQUEST_WINDOW_CHANGE, NULL);
```

❶   **Initialize system components**

The calls to `SEGGER_SYS_Init()` and `SEGGER_SYS_IP_Init()` use the SEGGER system abstraction layer to initialize services to the application.

❷   **Initialize SSH component**

This initializes the SSH component. The exact details of setup are described separately.

❸   **Configure user authentication**

SSH always requires user authentication even in the case where no password is required.

The first call to `SSH_SERVICE_Add()` prepares emSSH to handle the `ssh-userauth` service. This service provides user authentication, as you would expect. The second parameter is a callback that is activated when such a service is requested. We don't need to hook this particular capability now, so we pass in a null function pointer.

The second call to `SSH_USERAUTH_METHOD_Add()` adds the "none" method to the set of user authentication methods. The `none` method requires no authentication for a user, that is, no password, no key, nothing: when a user logs in, he is allowed through the front door without question.

We do, however, provide a callback in this case. This at least authenticates that the user name is valid, rather than accepting any user name. The implementation of this callback is described in *User authentication* on page 28.

❹   **Configure shell support**

These three function calls to `SSH_CHANNEL_REQUEST_Add()` set up SSH to accept shell requests. Each SSH session is capable of multiplexing several channels, each carrying different services, over a secure connection. In this case, we wish to accept shell requests, environment setup requests, and terminal requests. If any of these are missing, an SSH shell client will not connect to your embedded host.

The final request setup asks for a callback to be activated when a pseudo-terminal request is received and when a window change event occurs. The implementation of this callback is described in *Terminal requests* on page 30.

3.2.2   Accepting shell sessions

Once emSSH is correctly configured, the application in responsible for accepting connections and setting up any shell session:

```
//
// Bind SSH port.
//
BoundSocket = SEGGER_SYS_IP_Bind(22);   ❶
if (BoundSocket < 0) {
  _Exit("Cannot bind port 22!");
}
//
for (;;) {
  //
  // Wait for an incoming connection.
  //
  do {  ❷
    Socket = SEGGER_SYS_IP_Accept(BoundSocket);
  } while (Socket < 0);
  //
  SSH_SESSION_Alloc(&pSession);   ❸
  if (pSession == 0) {
    _Exit("No available session!");
  }
  //
  SSH_SESSION_Init(pSession, Socket, &_IP_Transport);   ❹
  SSH_SESSION_ConfBuffers(pSession,
                          _aRxTxBuffer, sizeof(_aRxTxBuffer),
                          _aRxTxBuffer, sizeof(_aRxTxBuffer));
  do {  ❺
    Status = SSH_SESSION_Process(pSession);
  } while (Status >= 0);
}
```

❶   **Bind the SSH port**

When used over TCP/IP, the server normally listens for connections on port 22. This port number has been registered with IANA, and has been officially assigned for SSH. You can, of course, use a different port number but then you would need to specify that port number when using a client: it's much easier to simply go with the standard port number.

The call to `SEGGER_SYS_IP_Bind()` uses the SEGGER abstraction layer to bind port 22 and return a socket corresponding to that binding. If the port is already bound and cannot accept incoming connections, the application terminates.

❷   **Accept an incoming connection**

Once the port is bound, we listen for incoming connections. The call to `SEGGER_SYS_IP_Accept()` waits for an incoming connection and creates a socket for that connection.

❸   **Allocate an SSH session for the connection**

The call to `SSH_SESSION_Alloc()` allocates an SSH session and assigns that session to its argument; if no session can be allocated, as they are all in use, a null pointer is assigned. For this simple example we only deal with a single session and therefore we don't expect allocation to fail, but if it does we exit the application.

❹   **Initialize and configure the session**

The calls to `SSH_SESSION_Init()` and `SSH_SESSION_ConfBuffers()` configure the session. `SSH_SESSION_Init()` sets up how the session communicates over the socket and `SSH_SESSION_ConfBuffers()` sets up the transmission and reception buffers that the SSH connection will use.

`SSH_SESSION_Init()` is provided a set of function pointers, in a structure, that vector to the appropriate send, receive, and close functions for a socket. In this example, we use the SEGGER abstraction layer to provide socket services:

```
static const SSH_TRANSPORT_API _IP_Transport = {
  SEGGER_SYS_IP_Send,
  SEGGER_SYS_IP_Recv,
  SEGGER_SYS_IP_Close,
};
```

These are very thin "shims" to the underlying embOS/IP or Windows socket functions, with the shim providing a consistent function prototype that adapts between the various implementations available.

`SSH_SESSION_ConfBuffers()` configures the buffers that emSSH will use when receiving and transmitting protocol packets. It's possible to specify separate reception and transmission buffers but, in this case, we use a shared buffer for both. With care, only a single buffer is required when transporting the protocol and using emSSH, but for more complex situations you may require separate buffers.

Although SSH is typically used over TCP/IP, it is not necessarily the only medium for SSH communications. For example, CANopen specifies a "shell" port that runs a user-defined protocol that could, quite literally, be SSH. In this case, your application could use emSSH to service TCP/IP connections *and* CAN connections using the same code but with different transport APIs: one for TCP/IP and one for CAN. And, if you wish to secure a serial connection, you could add functions that read and write over (one or more) serial connections.

❺   Run the SSH state machine

Once the connection is established and configured, you must call `SSH_SESSION_Process()` to run the SSH state machine that handles the SSH protocol including user authentication and connections. This is called in a loop, processing incoming messages and replying to them, until the session is closed.

The session may close gracefully or abruptly, and when `SSH_SESSION_Process()` returns with an error, the session is fully closed from an API perspective, and no further calls should be made to the API using that closed channel: doing so leads to undefined behavior.

Once the session is closed, it is returned for reuse. In this example, the code loops back to service additional incoming SSH connections.

3.2.3   User authentication

We now return to user authentication. Recall that user authentication setup provided a callback function:

```
//
// Allow "none" user authentication.
//
SSH_SERVICE_Add(&SSH_SERVICE_USERAUTH, 0);
SSH_USERAUTH_METHOD_Add(&SSH_USERAUTH_METHOD_NONE, _UserauthRequestNone);
```

A call is made to the function `_UserauthRequestNone` when the "none" user authentication method is requested by the client. emSSH calls any associated callback function to further process the authentication request. This is our implementation:

```
static int _UserauthRequestNone(SSH_SESSION               * pSession,  ❶
                                SSH_USERAUTH_REQUEST_PARAS * pReqParas) {
  SSH_USERAUTH_NONE_PARAS NoneParas;
  int                     Status;
  //
  SSH_USE_PARA(pSession);
  //
  Status = SSH_USERAUTH_NONE_ParseParas(pReqParas, &NoneParas);  ❷
  if (Status < 0) {
    Status = SSH_ERROR_USERAUTH_FAIL;
  } else if (pReqParas->UserNameLen == 4 &&  ❸
             SSH_MEMCMP(pReqParas->pUserName, "anon", 4) == 0) {
    Status = 0;
  } else {
    Status = SSH_ERROR_USERAUTH_FAIL;
  }
  //
  return Status;  ❹
}
```

❶ **Receive authentication parameters**

The callback function is provided with the session on which authentication is requested and the user authentication parameters that are available so far. We do not make use of the session in this case, so ignore it and silence any warning from compilers and source analysis tools using `SSH_USE_PARA`.

❷ **Parse method parameters**

Each user authentication method provides a different set of parameters for authentication. In this case we know that we are processing a `none` authentication request and, to do so, call `SSH_USERAUTH_NONE_ParseParas()` passing in the user authentication request. If there is an error in the incoming packet syntax, `SSH_USERAUTH_NONE_ParseParas()` returns a negative status indicating an error, and we set our running status to a "user authentication fail" error code.

❸ **Authenticate the user**

The user authentication request contains a reference to the user name that is the subject of this authentication request. That user name is pointed to by the `pUserName` member of the `SSH_USERAUTH_REQUEST_PARAS` structure, and the user name length is provided by `UserNameLen` in the same structure.

Rather than accept any user name, we accept only one, "`anon`," that can log in. The function tests to see whether user names match and, if they do, sets the running status to "success" which is defined as zero. If the user name does not match user authentication fails and the running status is set accordingly.

❹   **Return authentication status**

When user authentication is complete, the callback must return a status to emSSH indicating whether the user is authenticated or not. A zero or positive return value indicates that the user is authenticated and emSSH can continue with the session and send the peer a "success" message. A negative return value indicates that the user is not authenticated and emSSH sends the peer a "failure" message. At this point neither the session nor the channel are closed: the peer will handle the failure at its end and may decide to close the channel or session as it sees fit.

This completes the "`none`" user authentication scheme.

3.2.4   Terminal requests

We now return to pseudo-terminal requests. Recall that we provided a a callback to activate when a pseudo-terminal request is received:

```
//
// Add support for interactive shells.
//
SSH_CHANNEL_REQUEST_Add(&SSH_CHANNEL_REQUEST_SHELL,          NULL);
SSH_CHANNEL_REQUEST_Add(&SSH_CHANNEL_REQUEST_ENV,            NULL);
SSH_CHANNEL_REQUEST_Add(&SSH_CHANNEL_REQUEST_PTYREQ,         _TerminalRequest);
SSH_CHANNEL_REQUEST_Add(&SSH_CHANNEL_REQUEST_WINDOW_CHANGE, NULL);
```

A call is made to the function `_TerminalRequest` when the client requests a pseudo-terminal. Typical embedded targets do not support pseudo-terminals in the classic Unix model, but nevertheless we can emulate them such that SSH runs on embedded hosts. This is our implementation:

```
static int _TerminalRequest(SSH_SESSION              * pSession,   ❶
                            unsigned                   Channel,
                            SSH_CHANNEL_REQUEST_PARAS * pParas) {
  int Status;
  //
  SSH_CHANNEL_Config(pSession, Channel, 128, &_TerminalAPI, 0);   ❷
  if (pParas->WantReply) {   ❸
    Status = SSH_CHANNEL_SendSuccess(pSession, Channel);
  } else {
    Status = 0;
  }
  //
  return Status;   ❹
}
```

This is a very basic implementation of the terminal setup request and does not attempt to do anything more than configure a channel for terminal data.

❶ **Receive terminal request parameters**

The callback function is provided with the session on which the request is made (`pSession`), the channel number of that request (`Channel`), and specific per-request parameters (`pParas`). All three incoming parameters are relevant when setting up a terminal request.

❷ **Configure the channel**

Each terminal request is associated with a specific channel. The channel number is a small integer and identifies the *local* channel that is being addressed. We do not need to be concerned with the local channel number and how it is encoded or what it relates to—this is managed within the core of emSSH.

Each channel is bidirectional: it can receive data and data can be sent to it. In the case of a pseudo-terminal, we are transporting input and output that is human-readable.

The channel is configured by calling `SSH_CHANNEL_Config()`. The parameters are:

- A pointer to the session that carries the channel.
- The channel that is to be configured.
- A data capacity that limits incoming data and bandwidth consumed.
- Callbacks for handling events that occur on the channel.
- A user context to be associated with the channel.

The first two parameters passed to `SSH_CHANNEL_Config()` are the incoming parameters in our callback: they identify the channel to configure.

The next parameter is the *window size* of the SSH channel for receiving data form the client. This window size is rather like the window size of TCP/IP connections and it enables

connections to optimize resource use and bandwidth at a particular end of the bidirectional connection. (Each end is responsible for setting up its own window, so resource use at each end is individually configurable and independent of the other.)

For low-bandwidth connections, such as channels that carry interactive data from terminals, there is no real advantage to specifying a large window: the window is unlikely to become full (and therefore closed to any data transmission). For high bandwidth connections, such as secure copy or secure file transfer, a large window size is preferred to optimize efficient bulk data transfer over the connection.

In this case we have set a small window size of 128 bytes. This means that no more than 128 bytes of data will be presented to the client at any one time. This has some very real advantages for embedded systems and is covered more deeply in following sections.

The fourth parameter is a pointer to an API structure that contains functions to call when events occur on the channel. In our case we pass in a pointer to the following filled-in structure:

```
static const SSH_CHANNEL_API _TerminalAPI = {
  _TerminalChannelData,
  0,
  0
};
```

The specifics of channel data are described in details in the sections that follow, but briefly the first member of this structure is the function to call (`_TerminalChannelData`) when data is received on the channel.

The final parameter is a user context to associate with the channel. A client can associate any pointer with the context and the user context of a channel can be retrieved by `SSH_CHANNEL_QueryUserContext()`.

❸ **Acknowledge configuration**

The final responsibility of the callback is to acknowledge whether the channel was correctly configured or not. emSSH does not do this for you automatically on return because a callback may wish to acknowledge that a channel is correctly configured and then, immediately, perform some action on that channel (such as send data).

Whether the peer has requested an acknowledgment or not is indicated by the `WantReply` member of the incoming channel request parameters, a pointer to a `SSH_CHANNEL_REQUEST_PARAS` structure. If the peer has requested a reply, the callback must either call `SSH_CHANNEL_SendSuccess()` or `SSH_CHANNEL_SendFailure()` to send the peer the appropriate completion status.

If the peer does not wish to receive an acknowledgment, the callback need not send any reply—but, it is also not precluded from sending a failure *even if the peer requested no explicit reply*. If there is no need to send a reply, the running status is set to "success".

❹ **Return configuration status**

When configuration is complete, the callback must return a status to emSSH indicating whether the channel is correctly configured or not. If the channel is not correctly configured, the session is closed with the error propagated to the peer, and to the client by returning an error through `SSH_SESSION_Process()`.

3.2.5   Channel callbacks

As part of channel configuration the shell application provided a set of callbacks to handle events on a channel:

```
SSH_CHANNEL_Config(pSession, Channel, 128, &_TerminalAPI, 0);
```

We now turn our attention to the callback that deals with data reception on a channel:

```
static int _TerminalChannelData(        SSH_SESSION * pSession,    ❶
                                        unsigned      Channel,
                              const U8              * pData,
                                        unsigned      DataLen) {
  int Status;
  //
  Status = SSH_CHANNEL_SendData(pSession, Channel, pData, DataLen);    ❷
  //
  return Status;    ❸
}
```

Whenever the peer sends data to a channel, that data is presented to the client using the pfOnChannelData callback of the channel API.

❶   **Receive data callback parameters**

The parameters provided to the channel data callback are:

*   A pointer to the session that carries the channel.
*   The channel that has received data.
*   A pointer to the data that has been received.
*   The number of bytes that have been received.

❷   **Process incoming data**

In this simple application, we do no more than echo the received data back to the peer using SSH_CHANNEL_SendData(). The result of calling SSH_CHANNEL_SendData() is nonnegative on success and negative on failure.

What does this mean for a client that connects to a server running this code? It means that anything the user types is simply echoed back: it shows that a secure connection is established between the two peers and that the data can go through a round trip without corruption.

❸   **Return reception status**

When reception is complete, the callback must return a status to emSSH indicating whether the received data were handled correctly or not: nonnegative for success and negative for failure.

If there was an error processing the received data (in this case, if there was an error echoing back to the channel), the session is closed with the error propagated to the peer, and to the client by returning an error through SSH_SESSION_Process().

3.2.6   Testing the application

Now that this application is assembled, it's possible to test it out using an SSH client. In this case we choose to use Mac OS X on a Macintosh and select the standard `ssh` that comes from the OpenSSH project. The OpenSSH command line to log into a server is:

`ssh` *username@ip-address-or-domain-name*

We configured this particular application to only authenticate a single user successfully, the user name "`anon`". Trying to gain access using some other user name should fail:

```
MacBook:~ paul$ ssh somebody@10.0.0.247
Permission denied (none).
MacBook:~ paul$ _
```

This is correct behavior: when logging in with the user name `somebody`, we were denied access.

Now let's try our good user name:

```
MacBook:~ paul$ ssh anon@10.0.0.247
_
```

We're in! Typing some text echoes it to the terminal, as we expect:

```
MacBook:~ paul$ ssh anon@10.0.0.247
Hello!_
```

To close the connection from the client requires a specific key sequence. Press the **Return** key, followed by twiddle (`~`) followed by period (`.`). If you don't hit the keys in this order, the connection will remain open.

```
MacBook:~ paul$ ssh anon@10.0.0.247
Hello!
Connection to 10.0.0.247 closed.
MacBook:~ paul$ _
```

This concludes the simple presentation of emSSH; the following sections will elaborate on this and provide additional capabilities.

3.2.7   SSH_Shell1.c complete listing

```c
/**********************************************************************
*                   (c) SEGGER Microcontroller GmbH                  *
*                       The Embedded Experts                         *
*                          www.segger.com                            *
**********************************************************************


----------------------- END-OF-HEADER -----------------------------

File        : SSH_Shell1.c
Purpose     : Simplest SSH server that accepts incoming connections.

*/

/**********************************************************************
*
*       #include Section
*
**********************************************************************
*/

#include "SSH.h"
#include "SEGGER_SYS.h"
#include <string.h>

/**********************************************************************
*
*       Prototypes
*
**********************************************************************
*/

void MainTask(void);
static int _TerminalChannelData(      SSH_SESSION * pSession,
                                      unsigned      Channel,
                                const U8          * pData,
                                      unsigned      DataLen);

/**********************************************************************
*
*       Static const data
*
**********************************************************************
*/

static const SSH_TRANSPORT_API _IP_Transport = {
  SEGGER_SYS_IP_Send,
  SEGGER_SYS_IP_Recv,
  SEGGER_SYS_IP_Close,
};

static const SSH_CHANNEL_API _TerminalAPI = {
  _TerminalChannelData,
  NULL,
  NULL,
  NULL
};

/**********************************************************************
*
*       Static data
*
**********************************************************************
*/

static U8 _aRxTxBuffer[8192];
```

```
/**********************************************************************
*
*       Static code
*
***********************************************************************
*/

/**********************************************************************
*
*       _AppExit()
*
*  Function description
*    Exit the application with an error.
*
*  Parameters
*    sReason - Reason for exit, displayed for the user.
*/
static void _AppExit(const char *sReason) {
  SEGGER_SYS_IO_Printf(sReason);
  SEGGER_SYS_OS_Halt(100);
}

/**********************************************************************
*
*       _TerminalChannelData()
*
*  Function description
*    Handle data received from peer.
*
*  Parameters
*    pSession - Pointer to session.
*    Channel  - Local channel receiving the data.
*    pData    - Pointer to object that contains the data.
*    DataLen  - Octet length of the object that contains the data.
*
*  Return value
*    >= 0 - Success.
*    <  0 - Error.
*
*  Additional information
*    Simply echo received data.
*/
static int _TerminalChannelData(       SSH_SESSION * pSession,
                                       unsigned      Channel,
                                 const U8          * pData,
                                       unsigned      DataLen) {
  int Status;
  //
  Status = SSH_CHANNEL_SendData(pSession, Channel, pData, DataLen);
  //
  return Status;
}

/**********************************************************************
*
*       _TerminalRequest()
*
*  Function description
*    Request a terminal.
*
*  Parameters
*    pSession - Pointer to session.
*    Channel  - Local channel requesting the terminal.
*    pParas   - Pointer to channel request parameters.
*
*  Return value
*    >= 0 - Success.
```

```c
*   <  0 - Error.
*/
static int _TerminalRequest(SSH_SESSION                * pSession,
                            unsigned                      Channel,
                            SSH_CHANNEL_REQUEST_PARAS * pParas) {
  int Status;
  //
  SSH_CHANNEL_Config(pSession, Channel, 128, &_TerminalAPI, NULL);
  if (pParas->WantReply) {
    Status = SSH_CHANNEL_SendSuccess(pSession, Channel);
  } else {
    Status = 0;
  }
  //
  return Status;
}

/**********************************************************************
*
*       _UserauthRequestNone()
*
*  Function description
*    Request authentication of user with method "none".
*
*  Parameters
*    pSession  - Pointer to session.
*    pReqParas - Pointer to user authentication request parameters.
*
*  Return value
*    >= 0 - Success.
*    <  0 - Error.
*/
static int _UserauthRequestNone(SSH_SESSION                 * pSession,
                                SSH_USERAUTH_REQUEST_PARAS * pReqParas) {
  SSH_USERAUTH_NONE_PARAS NoneParas;
  int                     Status;
  //
  SSH_USE_PARA(pSession);
  //
  Status = SSH_USERAUTH_NONE_ParseParas(pReqParas, &NoneParas);
  if (Status < 0) {
    Status = SSH_ERROR_USERAUTH_FAIL;
  } else if (pReqParas->UserNameLen == 4 &&
             SSH_MEMCMP(pReqParas->pUserName, "anon", 4) == 0) {
    Status = 0;
  } else {
    Status = SSH_ERROR_USERAUTH_FAIL;
  }
  //
  return Status;
}

/**********************************************************************
*
*       Public code
*
***********************************************************************
*/

/**********************************************************************
*
*       MainTask()
*
*  Function description
*    Application entry point.
*/
void MainTask(void) {
  SSH_SESSION * pSession;
```

```c
  int          BoundSocket;
  int          Socket;
  int          Status;
  //
  SEGGER_SYS_Init();
  SEGGER_SYS_IP_Init();
  SSH_Init();
  //
  SEGGER_SYS_IO_Printf("\nemSSH V%s - Shell1 compiled " __DATE__ "
" __TIME__ "\n",
                       SSH_GetVersionText());
  SEGGER_SYS_IO_Printf("%s    www.segger.com\n\n",
                       SSH_GetCopyrightText());
  //
  // Allow "none" user authentication.
  //
  SSH_SERVICE_Add(&SSH_SERVICE_USERAUTH, NULL);
  SSH_USERAUTH_METHOD_Add(&SSH_USERAUTH_METHOD_NONE, _UserauthRequestNone);
  //
  // Add support for interactive shells.
  //
  SSH_CHANNEL_REQUEST_Add(&SSH_CHANNEL_REQUEST_SHELL,          NULL);
  SSH_CHANNEL_REQUEST_Add(&SSH_CHANNEL_REQUEST_ENV,           NULL);
  SSH_CHANNEL_REQUEST_Add(&SSH_CHANNEL_REQUEST_PTYREQ,         _TerminalRequest);
  SSH_CHANNEL_REQUEST_Add(&SSH_CHANNEL_REQUEST_WINDOW_CHANGE, NULL);
  //
  // Bind SSH port.
  //
  BoundSocket = SEGGER_SYS_IP_Bind(22);
  if (BoundSocket < 0) {
    _AppExit("Cannot bind port 22!");
  }
  //
  for (;;) {
    //
    // Wait for an incoming connection.
    //
    do {
      Socket = SEGGER_SYS_IP_Accept(BoundSocket);
    } while (Socket < 0);
    //
    SSH_SESSION_Alloc(&pSession);
    if (pSession == 0) {
      _AppExit("No available session!");
    }
    //
    SSH_SESSION_Init(pSession, Socket, &_IP_Transport);
    SSH_SESSION_ConfBuffers(pSession,
                            _aRxTxBuffer, sizeof(_aRxTxBuffer),
                            _aRxTxBuffer, sizeof(_aRxTxBuffer));
    do {
      Status = SSH_SESSION_Process(pSession);
    } while (Status >= 0);
  }
}

/*************************** End of file ***************************/
```

3.3   Adding a command line

The first application, `SSH_Shell1.c`, provides a framework that we will now extend with a functional command line. The command line can has no processing behind it, it just gathers input from the user and, when complete, echoes the command line back.

For a complete listing of this application, see *SSH_Shell2.c complete listing* on page 43.

3.3.1   Supplying a sign-on

When we log into a system using SSH one of the first things to be presented is a welcome message. We can provide our own welcome message by hooking the shell request and, when we receive it, write a welcome message. And all systems supply a command prompt, so we define the sign-on message and command prompt we will use:

```c
#define PROMPT                                                      \
  "emSSH> "

#define SIGNON                                                      \
  "\r\n"                                                            \
  "Welcome to the emSSH command line!  Type Ctrl+D to exit.\r\n"    \
  "\r\n"                                                            \
  PROMPT
```

Note that the carriage return and line feed are made explicit in these strings: emSSH does not expand the single C newline character '\n' into a carriage-return linefeed pair on transmission.

We modify the startup code to hook the shell request and direct it to our callback:

```c
//
// Add support for interactive shells.
//
SSH_CHANNEL_REQUEST_Add(&SSH_CHANNEL_REQUEST_SHELL,          _ShellRequest);
SSH_CHANNEL_REQUEST_Add(&SSH_CHANNEL_REQUEST_ENV,            NULL);
SSH_CHANNEL_REQUEST_Add(&SSH_CHANNEL_REQUEST_PTYREQ,         _TerminalRequest);
SSH_CHANNEL_REQUEST_Add(&SSH_CHANNEL_REQUEST_WINDOW_CHANGE, NULL);
```

With all that in place, we proceed to code the callback function:

```c
static int _ShellRequest(SSH_SESSION              * pSession,  ❶
                         unsigned                   Channel,
                         SSH_CHANNEL_REQUEST_PARAS * pParas) {
  int Status;
  //
  Status = SSH_CHANNEL_SendData(pSession, Channel,
                                SIGNON, strlen(SIGNON));  ❷
  if (Status < 0) {  ❸
    Status = SSH_CHANNEL_SendFailure(pSession, Channel);
  } else if (pParas->WantReply) {  ❹
    Status = SSH_CHANNEL_SendSuccess(pSession, Channel);
  }
  //
  return Status;  ❺
}
```

This request framework follows the same form as the pseudo-terminal request we've seen before.

❶   **Receive shell request parameters**

The callback function is provided with the session on which the request is made (`pSession`), the channel number of that request (`Channel`), and specific per-request parameters (`pParas`).

❷   **Send sign-on**

Immediately the shell request is received, the callback sends the sign-on message and initial prompt to the peer using `SSH_CHANNEL_SendData()`. If there's an error sending the data to the peer, it's reflected in the value returned by `SSH_CHANNEL_SendData()`: negative for failure, nonnegative for success.

❸   **Send failure completion status**

The final responsibility of the callback is to acknowledge whether the channel request completed successfully or not, as with the terminal request. If sending the channel data failed, it's likely that sending a failure response will also fail, but we try sending it anyway.

❹   **Send optional completion status**

If there is no error sending the channel data, we send the optional success response if the peer has requested it; we maintain the transmission status of the success response so that we can return it from the callback.

❺   **Return shell request status**

When the shell request is complete, the callback must return a status to emSSH indicating whether it executed correctly or not. If there was an error processing the shell request, the session is closed with the error propagated to the peer, and to the client by returning an error through `SSH_SESSION_Process()`.

3.3.2   Processing incoming data

We are going to write the command line processor within the received data callback. As you will see later, this is not the recommended way of writing a command line processor, but it is expedient if you need only a simple command line processor.

We declare some static variables at file scope to manage the command line:

```
static U8       _aCommandLine[70];
static unsigned _Cursor;
```

This is sufficient because we support only one shell session so far.

The code to implement this single command line is:

```
static int _TerminalChannelData(      SSH_SESSION * pSession,  ❶
                                      unsigned      Channel,
                               const U8           * pData,
                                      unsigned      DataLen) {
  unsigned i;
  U8       Ch;
  int      Status;
  //
  Status = 0;
  //
  for (i = 0; Status >= 0 && i < DataLen; ++i) {  ❷
    Ch = pData[i];
    if (0x20 <= Ch && Ch <= 0x7E) {  ❸
      if (_Cursor < sizeof(_aCommandLine)) {
        _aCommandLine[_Cursor++] = Ch;
        Status = SSH_CHANNEL_SendData(pSession, Channel, &Ch, 1);
      }
    } else if (Ch == 0x08 || Ch == 0x7F) {  ❹
      if (_Cursor > 0) {
        --_Cursor;
        Status = SSH_CHANNEL_SendData(pSession, Channel, "\b \b", 3);
      }
    } else if (Ch == '\r') {  ❺
      SSH_CHANNEL_SendData(pSession, Channel, "\r\n...", 5);
      SSH_CHANNEL_SendData(pSession, Channel, _aCommandLine, _Cursor);
      SSH_CHANNEL_SendData(pSession, Channel, "\r\n", 3);
      Status = SSH_CHANNEL_SendData(pSession, Channel, PROMPT, strlen(PROMPT));
      _Cursor = 0;
    } else if (Ch == 0x04) {  ❻
      SSH_CHANNEL_SendData(pSession, Channel, "\r\n\r\nBye!\r\n\r\n\r\n", 12);
      SSH_CHANNEL_Close(pSession, Channel);
      break;
    }
  }
  //
  return Status;  ❼
}
```

The callback is more complex now, but we can break it down into its constituent parts:

❶   **Receive data callback parameters**

This is identical to the previous example.

❷   **Process incoming data**

The loop iterates over all data presented to the callback, but exits if any iteration raises an error condition.

❸   Process printable characters

On reception of a printable character, the character is added to the command line buffer if there is space and echoed to the user. If there is no space, the character is simply dropped and not echoed which provides a hard-stop indication to the user that there is no more space.

An alternative is to echo back an alert, or bell when the buffer is full, which can be coded like this:

```
if (_Cursor < sizeof(_aCommandLine)) {
  _aCommandLine[_Cursor++] = Ch;
} else {
  Ch = '\a';
}
Status = SSH_CHANNEL_SendData(pSession, Channel, &Ch, 1);
```

❹   Process backspace and delete

Processing a backspace is straightforward: only backspace if not at the start of the buffer, and erase the character before the cursor on the terminal by backspacing, replacing with a space, and backspacing again.

❺   Process enter

When an enter character is found, the command processor echoes the received command to the terminal, prints a new command prompt, and resets the incoming buffer cursor to receive a new command.

One or more of the first three calls to `SSH_CHANNEL_SendData()` may well fail, but if one fails then all subsequent data write requests to that channel will fail, so it is only necessary to store the result of the final data write request.

When a data write request fails, *the session is not closed by emSSH* and remains open. In this case, when a data write fails, the loop terminates and the error status is returned to emSSH which then proceeds to close the session.

❻   Process log out

For those familiar with any Unix-type system, the universal Ctrl+D "end of file" entered in a shell causes the shell to terminate and log out. In this example, when Ctrl+D is seen we start to close the connection and exit the loop, but as a nicety the command processor sends a final log-out message before closing down.

When closing the connection like this, it doesn't matter what status code is returned by the callback, the session is already dead and buried.

❼   Return reception status

This should now be a familiar idiom: we return whether we processed all channel data successfully or not.

3.3.3   Try out the new command processor

Testing the application is the same as the previous example:

```
MacBook:~ paul$ ssh anon@10.0.0.247

Welcome to the emSSH command line!  Type Ctrl+D to exit.

emSSH> fsck /
...fsck /
emSSH> cd Work
...cd Work
emSSH>

Bye!

Connection to 10.0.0.247 closed.
MacBook:~ paul$ _
```

This concludes the section on adding a simple command line processor to emSSH.

3.3.4   SSH_Shell2.c complete listing

```c
/*********************************************************************
*                  (c) SEGGER Microcontroller GmbH                   *
*                      The Embedded Experts                          *
*                         www.segger.com                             *
**********************************************************************

----------------------- END-OF-HEADER ----------------------------

File       : SSH_Shell2.c
Purpose    : SSH server that supports a command line.

*/

/*********************************************************************
*
*       #include Section
*
**********************************************************************
*/

#include "SSH.h"
#include "SEGGER_SYS.h"
#include <string.h>

/*********************************************************************
*
*       Defines, configurable
*
**********************************************************************
*/

#define PROMPT                                               \
  "emSSH> "

#define SIGNON                                               \
  "\r\n"                                                     \
  "Welcome to the emSSH command line!  Type Ctrl+D to exit.\r\n" \
  "\r\n"                                                     \
  PROMPT

/*********************************************************************
*
*       Prototypes
*
**********************************************************************
*/

void MainTask(void);
static int _TerminalChannelData(       SSH_SESSION * pSession,
                                       unsigned      Channel,
                                 const U8          * pData,
                                       unsigned      DataLen);

/*********************************************************************
*
*       Static const data
*
**********************************************************************
*/

static const SSH_TRANSPORT_API _IP_Transport = {
  SEGGER_SYS_IP_Send,
  SEGGER_SYS_IP_Recv,
  SEGGER_SYS_IP_Close,
};
```

```c
static const SSH_CHANNEL_API _TerminalAPI = {
  _TerminalChannelData,
  NULL,
  NULL,
  NULL
};

/*********************************************************************
*
*       Static data
*
**********************************************************************
*/

static U8        _aRxTxBuffer[8192];
static U8        _aCommandLine[70];
static unsigned _Cursor;

/*********************************************************************
*
*       Static code
*
**********************************************************************
*/

/*********************************************************************
*
*       _AppExit()
*
*  Function description
*    Exit the application with an error.
*
*  Parameters
*    sReason - Reason for exit, displayed for the user.
*/
static void _AppExit(const char *sReason) {
  SEGGER_SYS_IO_Printf(sReason);
  SEGGER_SYS_OS_Halt(100);
}

/*********************************************************************
*
*       _ShellRequest()
*
*  Function description
*    Handle a shell channel request.
*
*  Parameters
*    pSession - Pointer to session.
*    Channel  - Local channel receiving the data.
*    pParas   - Pointer to channel request parameters.
*
*  Return value
*    >= 0 - Success.
*    <  0 - Error.
*/
static int _ShellRequest(SSH_SESSION              * pSession,
                         unsigned                   Channel,
                         SSH_CHANNEL_REQUEST_PARAS * pParas) {
  int Status;
  //
  Status = SSH_CHANNEL_SendData(pSession, Channel,
                                SIGNON, (unsigned)strlen(SIGNON));
  if (Status < 0) {
    Status = SSH_CHANNEL_SendFailure(pSession, Channel);
  } else if (pParas->WantReply) {
    Status = SSH_CHANNEL_SendSuccess(pSession, Channel);
```

```c
  }
  //
  return Status;
}

/**********************************************************************
*
*       _TerminalChannelData()
*
*  Function description
*    Handle data received from peer.
*
*  Parameters
*    pSession - Pointer to session.
*    Channel  - Local channel receiving the data.
*    pData    - Pointer to object that contains the data.
*    DataLen  - Octet length of the object that contains the data.
*
*  Return value
*    >= 0 - Success.
*    <  0 - Error.
*
*  Additional information
*    Provide in-callback handling of a command line processor.
*    This sample supports only one connection at a time.
*/
static int _TerminalChannelData(      SSH_SESSION * pSession,
                                      unsigned      Channel,
                                const U8          * pData,
                                      unsigned      DataLen) {
  unsigned i;
  U8       Ch;
  int      Status;
  //
  Status = 0;
  //
  for (i = 0; Status >= 0 && i < DataLen; ++i) {
    Ch = pData[i];
    if (0x20 <= Ch && Ch <= 0x7E) {
      if (_Cursor < sizeof(_aCommandLine)) {
        _aCommandLine[_Cursor++] = Ch;
        Status = SSH_CHANNEL_SendData(pSession, Channel, &Ch, 1);
      }
    } else if (Ch == 0x08 || Ch == 0x7F) {
      if (_Cursor > 0) {
        --_Cursor;
        Status = SSH_CHANNEL_SendData(pSession, Channel, "\b \b", 3);
      }
    } else if (Ch == '\r') {
      SSH_CHANNEL_SendData(pSession, Channel, "\r\n...", 5);
      SSH_CHANNEL_SendData(pSession, Channel, _aCommandLine, _Cursor);
      SSH_CHANNEL_SendData(pSession, Channel, "\r\n", 3);

  Status = SSH_CHANNEL_SendData(pSession, Channel, PROMPT, (unsigned)strlen(PROMPT));
      _Cursor = 0;
    } else if (Ch == 0x04) {
      SSH_CHANNEL_SendData(pSession, Channel, "\r\n\r\nBye!\r\n\r\n\n", 12);
      SSH_CHANNEL_Close(pSession, Channel);
      break;
    }
  }
  //
  return Status;
}

/**********************************************************************
*
*       _TerminalRequest()
```

```
*
*   Function description
*      Request a terminal.
*
*   Parameters
*      pSession - Pointer to session.
*      Channel  - Local channel requesting the terminal.
*      pParas   - Pointer to channel request parameters.
*
*   Return value
*    >= 0 - Success.
*    <  0 - Error.
*/
static int _TerminalRequest(SSH_SESSION               * pSession,
                            unsigned                    Channel,
                            SSH_CHANNEL_REQUEST_PARAS * pParas) {
  int Status;
  //
  SSH_CHANNEL_Config(pSession, Channel, 128, &_TerminalAPI, 0);
  if (pParas->WantReply) {
    Status = SSH_CHANNEL_SendSuccess(pSession, Channel);
  } else {
    Status = 0;
  }
  //
  return Status;
}

/**********************************************************************
*
*       _UserauthRequestNone()
*
*   Function description
*      Request authentication of user with method "none".
*
*   Parameters
*      pSession  - Pointer to session.
*      pReqParas - Pointer to user authentication request parameters.
*
*   Return value
*    >= 0 - Success.
*    <  0 - Error.
*/
static int _UserauthRequestNone(SSH_SESSION               * pSession,
                                SSH_USERAUTH_REQUEST_PARAS * pReqParas) {
  SSH_USERAUTH_NONE_PARAS NoneParas;
  int                     Status;
  //
  SSH_USE_PARA(pSession);
  //
  Status = SSH_USERAUTH_NONE_ParseParas(pReqParas, &NoneParas);
  if (Status < 0) {
    Status = SSH_ERROR_USERAUTH_FAIL;
  } else if (pReqParas->UserNameLen == 4 &&
             SSH_MEMCMP(pReqParas->pUserName, "anon", 4) == 0) {
    Status = 0;
  } else {
    Status = SSH_ERROR_USERAUTH_FAIL;
  }
  //
  return Status;
}

/**********************************************************************
*
*       Public code
*
**********************************************************************
```

```c
*/
/*********************************************************************
*
*       MainTask()
*
*  Function description
*    Application entry point.
*/
void MainTask(void) {
  SSH_SESSION * pSession;
  int           BoundSocket;
  int           Socket;
  int           Status;
  //
  SEGGER_SYS_Init();
  SEGGER_SYS_IP_Init();
  SSH_Init();
  //
  SEGGER_SYS_IO_Printf("\nemSSH V%s - Shell2 compiled " __DATE__ "
 " __TIME__ "\n",
                       SSH_GetVersionText());
  SEGGER_SYS_IO_Printf("%s    www.segger.com\n\n",
                       SSH_GetCopyrightText());
  //
  // Allow "none" user authentication.
  //
  SSH_SERVICE_Add(&SSH_SERVICE_USERAUTH, NULL);
  SSH_USERAUTH_METHOD_Add(&SSH_USERAUTH_METHOD_NONE, _UserauthRequestNone);
  //
  // Add support for interactive shells.
  //
  SSH_CHANNEL_REQUEST_Add(&SSH_CHANNEL_REQUEST_SHELL,          _ShellRequest);
  SSH_CHANNEL_REQUEST_Add(&SSH_CHANNEL_REQUEST_ENV,            NULL);
  SSH_CHANNEL_REQUEST_Add(&SSH_CHANNEL_REQUEST_PTYREQ,         _TerminalRequest);
  SSH_CHANNEL_REQUEST_Add(&SSH_CHANNEL_REQUEST_WINDOW_CHANGE, NULL);
  //
  // Bind SSH port.
  //
  BoundSocket = SEGGER_SYS_IP_Bind(22);
  if (BoundSocket < 0) {
    _AppExit("Cannot bind port 22!");
  }
  //
  for (;;) {
    //
    // Wait for an incoming connection.
    //
    do {
      Socket = SEGGER_SYS_IP_Accept(BoundSocket);
    } while (Socket < 0);
    //
    SSH_SESSION_Alloc(&pSession);
    if (pSession == 0) {
      _AppExit("No available session!");
    }
    //
    SSH_SESSION_Init(pSession, Socket, &_IP_Transport);
    SSH_SESSION_ConfBuffers(pSession,
                            _aRxTxBuffer, sizeof(_aRxTxBuffer),
                            _aRxTxBuffer, sizeof(_aRxTxBuffer));
    do {
      Status = SSH_SESSION_Process(pSession);
    } while (Status >= 0);
  }
}
```

```
/************************** End of file **************************/
```

3.4   Users with passwords

Our security for log in is very weak: supplying only the correct user name allows a login. In this section the security is enhanced by supporting *password authentication* where a user must supply the correct password to log in.

For a complete listing of this application, see *SSH_Shell3.c complete listing* on page 52.

3.4.1   Adding a second user authentication method

Previous samples have added only one user authentication method to emSSH, the `none` method. This method is useful for guest access, if you need it, where the services offered through the guest login is restricted.

To enhance this with a login protocol that requires user name and password, a second protocol is added when the application is entered:

```
//
// Allow "none" and "password" user authentication.
//
SSH_SERVICE_Add(&SSH_SERVICE_USERAUTH, 0);
SSH_USERAUTH_METHOD_Add(&SSH_USERAUTH_METHOD_NONE,     _UserauthRequestNone);
SSH_USERAUTH_METHOD_Add(&SSH_USERAUTH_METHOD_PASSWORD, _UserauthRequestPassword);
```

The second user authentication method uses a different callback that processes login requests with a password. That callback implementation is:

```
static int _UserauthRequestPassword(SSH_SESSION            * pSession,  ❶
                                    SSH_USERAUTH_REQUEST_PARAS * pReqParas) {
  SSH_USERAUTH_PASSWORD_PARAS PasswordParas;
  int                         Status;
  //
  SSH_USE_PARA(pSession);
  //
  Status = SSH_USERAUTH_PASSWORD_ParseParas(pReqParas, &PasswordParas);  ❷
  if (Status < 0) {
    Status = SSH_ERROR_USERAUTH_FAIL;
  } else if (pReqParas->UserNameLen == 5 &&  ❸
             memcmp(pReqParas->pUserName, "admin", 5) == 0) {
    if (PasswordParas.PasswordLen == 6 &&  ❹
        memcmp(PasswordParas.pPassword, "secret", 6) == 0) {
      Status = 0;
    } else {
      Status = SSH_ERROR_USERAUTH_FAIL;
    }
  } else {
    Status = SSH_ERROR_USERAUTH_FAIL;  ❺
  }
  //
  return Status;
}
```

This implementation adds one password-authenticated user.

❶   **Recieve user authentication parameters**

This is the same as the `none` user authentication method.

❷   **Parse the password parameters**

As each user authentication method has a different packet format containing format-specific parameters, the callback is responsible for decoding the authentication parameters with an appropriate decoding function. For password authentication, the user name is provided

in the `SSH_USERAUTH_REQUEST_PARAS` structure pointed to by `pReqParas` just as the `none` method, but the password is stored in (as-yet) undecoded parameters that need parsing.

For password authentication, the `SSH_USERAUTH_PASSWORD_ParseParas()` function parses the parameters and ensures that the packet format is correct. If the packet format is correct, user authentication can proceed.

❸   Check the user name

The user name is checked in the same manner as `none` authentication. We accept only one password-authenticated user, the user "`admin`".

❹   Check the user's password

If the user name matches `admin`, the password parsed from the user authentication password parameters is checked for a match with the store password, "`secret`". If there is a match, the user is authenticated and the running status is set to zero to reflect a successful authentication, and if not, it's set to an authentication failure status.

This is a particularly weak implementation of password authentication as all user names and passwords are held in the clear and will be readable in the clear in the flash of the target microprocessor or—worse still—in the PC application if emSSH is compiled for a PC. There are better ways of storing passwords, but this sample shows only the mechanics of verifying a user's password in the context of the SSH password authentication method, not in the context of "best practice." We show how this can be achieved in the next sample.

❺   Reject other users

The if-else-if structure can be extended to allow additional users, of course, but after all user names are exhausted, authentication by password must fail.

3.4.2   Testing password log in

Testing the application now requires that we use the user name `admin` rather than `anon`.

Correct password

```
MacBook:~ paul$ ssh admin@10.0.0.247

admin@10.0.0.247's password: secret

Welcome to the emSSH command line!  Type Ctrl+D to exit.

emSSH> ls -l
...ls -l
emSSH>

Bye!

Connection to 10.0.0.247 closed.
MacBook:~ paul$ _
```

Incorrect password

```
MacBook:~ paul$ ssh admin@10.0.0.247

admin@10.0.0.247's password: 123456
Permission denied, please try again.
admin@10.0.0.247's password: password
Permission denied, please try again.
admin@10.0.0.247's password: qwerty
Permission denied (none,password).
MacBook:~ paul$ _
```

Unknown user

```
MacBook:~ paul$ ssh starbuck@10.0.0.247

starbuck@10.0.0.247's password: kara
Permission denied, please try again.
starbuck@10.0.0.247's password: galactica
Permission denied, please try again.
starbuck@10.0.0.247's password: adama
Permission denied (none,password).
MacBook:~ paul$ _
```

3.4.3   Restricting guest access

This sample has both the `none` user authentication method and the `password` user authentication method installed. If all users are required to hold a password, and guest access without password must be denied, modify the code to add only the password authentication method:

```
//
// Allow only "password" user authentication.
//
SSH_SERVICE_Add(&SSH_SERVICE_USERAUTH, 0);
SSH_USERAUTH_METHOD_Add(&SSH_USERAUTH_METHOD_PASSWORD, _UserauthRequestPassword);
```

3.4.4   SSH_Shell3.c complete listing

```c
/**********************************************************************
*                (c) SEGGER Microcontroller GmbH                    *
*                    The Embedded Experts                           *
*                       www.segger.com                              *
**********************************************************************

----------------------- END-OF-HEADER ----------------------------

File       : SSH_Shell3.c
Purpose    : SSH server that adds password user authentication.

*/

/**********************************************************************
*
*       #include Section
*
**********************************************************************
*/

#include "SSH.h"
#include "SEGGER_SYS.h"
#include <string.h>

/**********************************************************************
*
*       Defines, configurable
*
**********************************************************************
*/

#define PROMPT                                                 \
  "emSSH> "

#define SIGNON                                                 \
  "\r\n"                                                       \
  "Welcome to the emSSH command line!  Type Ctrl+D to exit.\r\n" \
  "\r\n"                                                       \
  PROMPT

/**********************************************************************
*
*       Prototypes
*
**********************************************************************
*/

void MainTask(void);
static int _TerminalChannelData(       SSH_SESSION * pSession,
                                       unsigned      Channel,
                                 const U8          * pData,
                                       unsigned      DataLen);

/**********************************************************************
*
*       Static const data
*
**********************************************************************
*/

static const SSH_TRANSPORT_API _IP_Transport = {
  SEGGER_SYS_IP_Send,
  SEGGER_SYS_IP_Recv,
  SEGGER_SYS_IP_Close,
};
```

```c
static const SSH_CHANNEL_API _TerminalAPI = {
  _TerminalChannelData,
  NULL,
  NULL,
  NULL
};

/*********************************************************************
*
*       Static data
*
**********************************************************************
*/

static U8        _aRxTxBuffer[8192];
static U8        _aCommandLine[70];
static unsigned _Cursor;

/*********************************************************************
*
*       Static code
*
**********************************************************************
*/

/*********************************************************************
*
*       _AppExit()
*
*  Function description
*    Exit the application with an error.
*
*  Parameters
*    sReason - Reason for exit, displayed for the user.
*/
static void _AppExit(const char *sReason) {
  SEGGER_SYS_IO_Printf(sReason);
  SEGGER_SYS_OS_Halt(100);
}

/*********************************************************************
*
*       _ShellRequest()
*
*  Function description
*    Handle a shell channel request.
*
*  Parameters
*    pSession - Pointer to session.
*    Channel  - Local channel receiving the data.
*    pParas   - Pointer to channel request parameters.
*
*  Return value
*    >= 0 - Success.
*    <  0 - Error.
*/
static int _ShellRequest(SSH_SESSION                 * pSession,
                         unsigned                      Channel,
                         SSH_CHANNEL_REQUEST_PARAS * pParas) {
  int Status;
  //
  Status = SSH_CHANNEL_SendData(pSession, Channel,
                                SIGNON, (unsigned)strlen(SIGNON));
  if (Status < 0) {
    Status = SSH_CHANNEL_SendFailure(pSession, Channel);
  } else if (pParas->WantReply) {
    Status = SSH_CHANNEL_SendSuccess(pSession, Channel);
```

```
    }
    //
    return Status;
}

/**********************************************************************
*
*       _TerminalChannelData()
*
*  Function description
*    Handle data received from peer.
*
*  Parameters
*    pSession - Pointer to session.
*    Channel  - Local channel receiving the data.
*    pData    - Pointer to object that contains the data.
*    DataLen  - Octet length of the object that contains the data.
*
*  Return value
*    >= 0 - Success.
*    <  0 - Error.
*
*  Additional information
*    Provide in-callback handling of a command line processor.
*    This sample supports only one connection at a time.
*/
static int _TerminalChannelData(       SSH_SESSION * pSession,
                                       unsigned      Channel,
                                 const U8          * pData,
                                       unsigned      DataLen) {
  unsigned i;
  U8       Ch;
  int      Status;
  //
  Status = 0;
  //
  for (i = 0; Status >= 0 && i < DataLen; ++i) {
    Ch = pData[i];
    if (0x20 <= Ch && Ch <= 0x7E) {
      if (_Cursor < sizeof(_aCommandLine)) {
        _aCommandLine[_Cursor++] = Ch;
        Status = SSH_CHANNEL_SendData(pSession, Channel, &Ch, 1);
      }
    } else if (Ch == 0x08 || Ch == 0x7F) {
      if (_Cursor > 0) {
        --_Cursor;
        Status = SSH_CHANNEL_SendData(pSession, Channel, "\b \b", 3);
      }
    } else if (Ch == '\r') {
      SSH_CHANNEL_SendData(pSession, Channel, "\r\n...", 5);
      SSH_CHANNEL_SendData(pSession, Channel, _aCommandLine, _Cursor);
      SSH_CHANNEL_SendData(pSession, Channel, "\r\n", 3);

  Status = SSH_CHANNEL_SendData(pSession, Channel, PROMPT, (unsigned)strlen(PROMPT));
      _Cursor = 0;
    } else if (Ch == 0x04) {
      SSH_CHANNEL_SendData(pSession, Channel, "\r\n\r\nBye!\r\n\r\n\n", 12);
      SSH_CHANNEL_Close(pSession, Channel);
      break;
    }
  }
  //
  return Status;
}

/**********************************************************************
*
*       _TerminalRequest()
```

```
 *
 *  Function description
 *    Request a terminal.
 *
 *  Parameters
 *    pSession - Pointer to session.
 *    Channel  - Local channel requesting the terminal.
 *    pParas   - Pointer to channel request parameters.
 *
 *  Return value
 *    >= 0 - Success.
 *    <  0 - Error.
 */
static int _TerminalRequest(SSH_SESSION               * pSession,
                            unsigned                    Channel,
                            SSH_CHANNEL_REQUEST_PARAS * pParas) {
  int Status;
  //
  SSH_CHANNEL_Config(pSession, Channel, 128, &_TerminalAPI, NULL);
  if (pParas->WantReply) {
    Status = SSH_CHANNEL_SendSuccess(pSession, Channel);
  } else {
    Status = 0;
  }
  //
  return Status;
}

/*********************************************************************
 *
 *       _UserauthRequestNone()
 *
 *  Function description
 *    Request authentication of user with method "none".
 *
 *  Parameters
 *    pSession  - Pointer to session.
 *    pReqParas - Pointer to user authentication request parameters.
 *
 *  Return value
 *    >= 0 - Success.
 *    <  0 - Error.
 */
static int _UserauthRequestNone(SSH_SESSION               * pSession,
                                SSH_USERAUTH_REQUEST_PARAS * pReqParas) {
  SSH_USERAUTH_NONE_PARAS NoneParas;
  int                     Status;
  //
  SSH_USE_PARA(pSession);
  //
  Status = SSH_USERAUTH_NONE_ParseParas(pReqParas, &NoneParas);
  if (Status < 0) {
    Status = SSH_ERROR_USERAUTH_FAIL;
  } else if (pReqParas->UserNameLen == 4 &&
             SSH_MEMCMP(pReqParas->pUserName, "anon", 4) == 0) {
    Status = 0;
  } else {
    Status = SSH_ERROR_USERAUTH_FAIL;
  }
  //
  return Status;
}

/*********************************************************************
 *
 *       _UserauthRequestPassword()
 *
 *  Function description
```

```c
*    Request authentication of user with method "password".
*
*  Parameters
*    pSession  - Pointer to session.
*    pReqParas - Pointer to user authentication request parameters.
*
*  Return value
*    >= 0 - Success.
*    <  0 - Error.
*/
static int _UserauthRequestPassword(SSH_SESSION                 * pSession,
                                    SSH_USERAUTH_REQUEST_PARAS * pReqParas) {
  SSH_USERAUTH_PASSWORD_PARAS PasswordParas;
  int                         Status;
  //
  SSH_USE_PARA(pSession);
  //
  Status = SSH_USERAUTH_PASSWORD_ParseParas(pReqParas, &PasswordParas);
  if (Status < 0) {
    Status = SSH_ERROR_USERAUTH_FAIL;
  } else if (pReqParas->UserNameLen == 5 &&
             memcmp(pReqParas->pUserName, "admin", 5) == 0) {
    if (PasswordParas.PasswordLen == 6 &&
        memcmp(PasswordParas.pPassword, "secret", 6) == 0) {
      Status = 0;
    } else {
      Status = SSH_ERROR_USERAUTH_FAIL;
    }
  } else {
    Status = SSH_ERROR_USERAUTH_FAIL;
  }
  //
  return Status;
}

/**********************************************************************
*
*             Public code
*
***********************************************************************
*/

/**********************************************************************
*
*       MainTask()
*
*  Function description
*    Application entry point.
*/
void MainTask(void) {
  SSH_SESSION * pSession;
  int           BoundSocket;
  int           Socket;
  int           Status;
  //
  SEGGER_SYS_Init();
  SEGGER_SYS_IP_Init();
  SSH_Init();
  //
  SEGGER_SYS_IO_Printf("\nemSSH V%s - Shell3 compiled " __DATE__ "
 " __TIME__ "\n",
                       SSH_GetVersionText());
  SEGGER_SYS_IO_Printf("%s    www.segger.com\n\n",
                       SSH_GetCopyrightText());
  //
  // Allow "none" and "password" user authentication.
  //
  SSH_SERVICE_Add(&SSH_SERVICE_USERAUTH, NULL);
```

```c
    SSH_USERAUTH_METHOD_Add(&SSH_USERAUTH_METHOD_NONE,      _UserauthRequestNone);
    SSH_USERAUTH_METHOD_Add(&SSH_USERAUTH_METHOD_PASSWORD, _UserauthRequestPassword);
    //
    // Add support for interactive shells.
    //
    SSH_CHANNEL_REQUEST_Add(&SSH_CHANNEL_REQUEST_SHELL,         _ShellRequest);
    SSH_CHANNEL_REQUEST_Add(&SSH_CHANNEL_REQUEST_ENV,           NULL);
    SSH_CHANNEL_REQUEST_Add(&SSH_CHANNEL_REQUEST_PTYREQ,        _TerminalRequest);
    SSH_CHANNEL_REQUEST_Add(&SSH_CHANNEL_REQUEST_WINDOW_CHANGE, NULL);
    //
    // Bind SSH port.
    //
    BoundSocket = SEGGER_SYS_IP_Bind(22);
    if (BoundSocket < 0) {
      _AppExit("Cannot bind port 22!");
    }
    //
    for (;;) {
      //
      // Wait for an incoming connection.
      //
      do {
        Socket = SEGGER_SYS_IP_Accept(BoundSocket);
      } while (Socket < 0);
      //
      SSH_SESSION_Alloc(&pSession);
      if (pSession == 0) {
        _AppExit("No available session!");
      }
      //
      SSH_SESSION_Init(pSession, Socket, &_IP_Transport);
      SSH_SESSION_ConfBuffers(pSession,
                              _aRxTxBuffer, sizeof(_aRxTxBuffer),
                              _aRxTxBuffer, sizeof(_aRxTxBuffer));
      do {
        Status = SSH_SESSION_Process(pSession);
      } while (Status >= 0);
    }
}

/*************************** End of file ***************************/
```

3.5   Better password storage

Rather than storing passwords in the clear, it's better to use a one-way function and, what's more, a computationally-expensive one-way function to prevent using brute force to crack the key. Even better still is to use a *salted password* that prevents dictionary lookup of the one-way function result.

For a complete listing of this application, see *SSH_Shell4.c complete listing* on page 60.

3.5.1   Securing passwords

A secure way of storing passwords is to use the PBKDF2 algorithm combined with an HMAC algorithm, for example PBKDF2-HMAC-SHA-256. If we take the password `secret` and prepend 16 bytes of random data, the *salt*, and run it through PBKDF2-HMAC-SHA-256, and ask for 32 bytes of output, the result is the block:

```
C8 52 33 15 9D 6C 74 E3 04 97 BE 95 54 CE DB 6A
37 DD 07 EE AA 7A 99 01 F1 1E 57 6F 64 0A 99 2F
```

It is infeasible to find a password combined with the salt which hashes to the same output (this is technically called a *collision*). We no longer store the password in the clear and, what is more, we're confident that the password cannot be cracked by brute force.

The salt and password hash can be stored in plain sight, without any need for secrecy, that's the beauty of this method, because we rely on the hard one-way function of the salted password provide the security.

```
static const U8 _aAdminPasswordSalt[16] = {
  0xBC, 0x4B, 0xCD, 0x8C, 0xC9, 0x99, 0x74, 0xC6,
  0xC5, 0xFE, 0x5E, 0x98, 0x20, 0xD8, 0x6D, 0x03
};

static const U8 _aAdminHashedPassword[32] = {
  0xC8, 0x52, 0x33, 0x15, 0x9D, 0x6C, 0x74, 0xE3,
  0x04, 0x97, 0xBE, 0x95, 0x54, 0xCE, 0xDB, 0x6A,
  0x37, 0xDD, 0x07, 0xEE, 0xAA, 0x7A, 0x99, 0x01,
  0xF1, 0x1E, 0x57, 0x6F, 0x64, 0x0A, 0x99, 0x2F
};
```

3.5.2   Modifying the password authenticator

This is how we modify the user authentication callback:

```
static int _UserauthRequestPassword(SSH_SESSION                * pSession,
                                    SSH_USERAUTH_REQUEST_PARAS * pReqParas) {
  SSH_USERAUTH_PASSWORD_PARAS PasswordParas;
  int                         Status;
  U8                          aHash[32];
  //
  SSH_USE_PARA(pSession);
  //
  Status = SSH_USERAUTH_PASSWORD_ParseParas(pReqParas, &PasswordParas);
  if (Status < 0) {
    Status = SSH_ERROR_USERAUTH_FAIL;
  } else if (pReqParas->UserNameLen == 5 &&
             memcmp(pReqParas->pUserName, "admin", 5) == 0) {
    //
    CRYPTO_PBKDF2_HMAC_SHA256_Calc(PasswordParas.pPassword,  ❶
                                   PasswordParas.PasswordLen,
                                   _aAdminPasswordSalt,
                                   sizeof(_aAdminPasswordSalt),
                                   10000,
                                   aHash,
                                   sizeof(aHash));
```

```
    if (CRYPTO_MEMDIF(aHash, _aAdminHashedPassword, sizeof(aHash)) == 0) {   ❷
      Status = 0;
    } else {
      Status = SSH_ERROR_USERAUTH_FAIL;
    }
  } else {
    Status = SSH_ERROR_USERAUTH_FAIL;
  }
  //
  return Status;
}
```

The direct compare of the entered password with the user's correct password is replaced with a calculation.

❶   **Compute the hash of the salted password**

This uses the emCrypt implementation of PBKDF2-HMAC-SHA-256 to compute the hash of the the salted password, selecting 10,000 iterations and 32 bytes of output. The value 32 happens to match the MAC size of the HMAC-SHA-256 algorithm, but any length can be requested—longer lengths require more computation and are more secure.

❷   **Compare the expected and calculated password hashes**

The expected and calculated password hashes are compared for equality: only if they match is the password correct. This particular implementation uses a constant-time comparison function, `CRYPTO_MEMDIF`, but it's not absolutely necessary. From here on it's standard form to return the result of user authentication.

3.5.3   What's the cost?

This implementation uses 10,000 iterations of PBKDF2-HMAC-SHA-256 and therefore it takes slightly longer to log in to a shell than with plaintext password comparisons. On the Cortex-M4 emPower board running at 168 MHz with hardware acceleration of SHA-256, it takes approximately 1.4 seconds to verify the password using 10,000 iterations.

There is no fixed password hashing scheme, this is but one example. You can tune the implementation to your particular requirements.

3.5.4   SSH_Shell4.c complete listing

```c
/*********************************************************************
*                  (c) SEGGER Microcontroller GmbH                  *
*                      The Embedded Experts                         *
*                         www.segger.com                            *
*********************************************************************


----------------------- END-OF-HEADER -----------------------------

File       : SSH_Shell4.c
Purpose    : SSH server that adds secure passwords.

*/

/*********************************************************************
*
*       #include Section
*
*********************************************************************
*/

#include "SSH.h"
#include "SEGGER_SYS.h"
#include <string.h>

/*********************************************************************
*
*       Defines, configurable
*
*********************************************************************
*/

#define PROMPT                                                  \
  "emSSH> "

#define SIGNON                                                  \
  "\r\n"                                                        \
  "Welcome to the emSSH command line!  Type Ctrl+D to exit.\r\n" \
  "\r\n"                                                        \
  PROMPT

/*********************************************************************
*
*       Prototypes
*
*********************************************************************
*/

void MainTask(void);
static int _TerminalChannelData(      SSH_SESSION * pSession,
                                      unsigned      Channel,
                                const U8          * pData,
                                      unsigned      DataLen);

/*********************************************************************
*
*       Static const data
*
*********************************************************************
*/

static const SSH_TRANSPORT_API _IP_Transport = {
  SEGGER_SYS_IP_Send,
  SEGGER_SYS_IP_Recv,
  SEGGER_SYS_IP_Close,
};
```

```
static const SSH_CHANNEL_API _TerminalAPI = {
  _TerminalChannelData,
  NULL,
  NULL,
  NULL
};

static const U8 _aAdminPasswordSalt[16] = {
  0xBC, 0x4B, 0xCD, 0x8C, 0xC9, 0x99, 0x74, 0xC6,
  0xC5, 0xFE, 0x5E, 0x98, 0x20, 0xD8, 0x6D, 0x03
};

static const U8 _aAdminHashedPassword[32] = {
  0xC8, 0x52, 0x33, 0x15, 0x9D, 0x6C, 0x74, 0xE3,
  0x04, 0x97, 0xBE, 0x95, 0x54, 0xCE, 0xDB, 0x6A,
  0x37, 0xDD, 0x07, 0xEE, 0xAA, 0x7A, 0x99, 0x01,
  0xF1, 0x1E, 0x57, 0x6F, 0x64, 0x0A, 0x99, 0x2F
};

/**********************************************************************
*
*       Static data
*
***********************************************************************
*/

static U8       _aRxTxBuffer[8192];
static U8       _aCommandLine[70];
static unsigned _Cursor;

/**********************************************************************
*
*       Static code
*
***********************************************************************
*/

/**********************************************************************
*
*       _AppExit()
*
*  Function description
*    Exit the application with an error.
*
*  Parameters
*    sReason - Reason for exit, displayed for the user.
*/
static void _AppExit(const char *sReason) {
  SEGGER_SYS_IO_Printf(sReason);
  SEGGER_SYS_OS_Halt(100);
}

/**********************************************************************
*
*       _ShellRequest()
*
*  Function description
*    Handle a shell channel request.
*
*  Parameters
*    pSession - Pointer to session.
*    Channel  - Local channel receiving the data.
*    pParas   - Pointer to channel request parameters.
*
*  Return value
*    >= 0 - Success.
*    <  0 - Error.
```

```c
*/
static int _ShellRequest(SSH_SESSION               * pSession,
                         unsigned                    Channel,
                         SSH_CHANNEL_REQUEST_PARAS * pParas) {
  int Status;
  //
  Status = SSH_CHANNEL_SendData(pSession, Channel,
                                SIGNON, (unsigned)strlen(SIGNON));
  if (Status < 0) {
    Status = SSH_CHANNEL_SendFailure(pSession, Channel);
  } else if (pParas->WantReply) {
    Status = SSH_CHANNEL_SendSuccess(pSession, Channel);
  }
  //
  return Status;
}

/*********************************************************************
*
*       _TerminalChannelData()
*
*  Function description
*    Handle data received from peer.
*
*  Parameters
*    pSession - Pointer to session.
*    Channel  - Local channel receiving the data.
*    pData    - Pointer to object that contains the data.
*    DataLen  - Octet length of the object that contains the data.
*
*  Return value
*    >= 0 - Success.
*    <  0 - Error.
*
*  Additional information
*    Provide in-callback handling of a command line processor.
*    This sample supports only one connection at a time.
*/
static int _TerminalChannelData(      SSH_SESSION * pSession,
                                      unsigned      Channel,
                                const U8          * pData,
                                      unsigned      DataLen) {
  unsigned i;
  U8       Ch;
  int      Status;
  //
  Status = 0;
  //
  for (i = 0; Status >= 0 && i < DataLen; ++i) {
    Ch = pData[i];
    if (0x20 <= Ch && Ch <= 0x7E) {
      if (_Cursor < sizeof(_aCommandLine)) {
        _aCommandLine[_Cursor++] = Ch;
        Status = SSH_CHANNEL_SendData(pSession, Channel, &Ch, 1);
      }
    } else if (Ch == 0x08 || Ch == 0x7F) {
      if (_Cursor > 0) {
        --_Cursor;
        Status = SSH_CHANNEL_SendData(pSession, Channel, "\b \b", 3);
      }
    } else if (Ch == '\r') {
      SSH_CHANNEL_SendData(pSession, Channel, "\r\n...", 5);
      SSH_CHANNEL_SendData(pSession, Channel, _aCommandLine, _Cursor);
      SSH_CHANNEL_SendData(pSession, Channel, "\r\n", 3);

  Status = SSH_CHANNEL_SendData(pSession, Channel, PROMPT, (unsigned)strlen(PROMPT));
      _Cursor = 0;
    } else if (Ch == 0x04) {
```

```c
        SSH_CHANNEL_SendData(pSession, Channel, "\r\n\r\n\nBye!\r\n\r\n\r\n", 12);
        SSH_CHANNEL_Close(pSession, Channel);
        break;
      }
    }
  }
  //
  return Status;
}

/*********************************************************************
*
*       _TerminalRequest()
*
*  Function description
*    Request a terminal.
*
*  Parameters
*    pSession - Pointer to session.
*    Channel  - Local channel requesting the terminal.
*    pParas   - Pointer to channel request parameters.
*
*  Return value
*    >= 0 - Success.
*    <  0 - Error.
*/
static int _TerminalRequest(SSH_SESSION                 * pSession,
                            unsigned                      Channel,
                            SSH_CHANNEL_REQUEST_PARAS * pParas) {
  int Status;
  //
  SSH_CHANNEL_Config(pSession, Channel, 128, &_TerminalAPI, 0);
  if (pParas->WantReply) {
    Status = SSH_CHANNEL_SendSuccess(pSession, Channel);
  } else {
    Status = 0;
  }
  //
  return Status;
}

/*********************************************************************
*
*       _UserauthRequestNone()
*
*  Function description
*    Request authentication of user with method "none".
*
*  Parameters
*    pSession  - Pointer to session.
*    pReqParas - Pointer to user authentication request parameters.
*
*  Return value
*    >= 0 - Success.
*    <  0 - Error.
*/
static int _UserauthRequestNone(SSH_SESSION                 * pSession,
                                SSH_USERAUTH_REQUEST_PARAS * pReqParas) {
  SSH_USERAUTH_NONE_PARAS NoneParas;
  int                     Status;
  //
  SSH_USE_PARA(pSession);
  //
  Status = SSH_USERAUTH_NONE_ParseParas(pReqParas, &NoneParas);
  if (Status < 0) {
    Status = SSH_ERROR_USERAUTH_FAIL;
  } else if (pReqParas->UserNameLen == 4 &&
             SSH_MEMCMP(pReqParas->pUserName, "anon", 4) == 0) {
    Status = 0;
```

```c
    } else {
      Status = SSH_ERROR_USERAUTH_FAIL;
    }
    //
    return Status;
  }

/*********************************************************************
*
*       _UserauthRequestPassword()
*
*  Function description
*    Request authentication of user with method "password".
*
*  Parameters
*    pSession  - Pointer to session.
*    pReqParas - Pointer to user authentication request parameters.
*
*  Return value
*    >= 0 - Success.
*    <  0 - Error.
*/
static int _UserauthRequestPassword(SSH_SESSION               * pSession,
                                    SSH_USERAUTH_REQUEST_PARAS * pReqParas) {
  SSH_USERAUTH_PASSWORD_PARAS PasswordParas;
  int                         Status;
  U8                          aHash[CRYPTO_SHA256_DIGEST_BYTE_COUNT];
  //
  SSH_USE_PARA(pSession);
  //
  Status = SSH_USERAUTH_PASSWORD_ParseParas(pReqParas, &PasswordParas);
  if (Status < 0) {
    Status = SSH_ERROR_USERAUTH_FAIL;
  } else if (pReqParas->UserNameLen == 5 &&
             memcmp(pReqParas->pUserName, "admin", 5) == 0) {
    //
    CRYPTO_PBKDF2_HMAC_SHA256_Calc(PasswordParas.pPassword,
                                   PasswordParas.PasswordLen,
                                   _aAdminPasswordSalt,
                                   sizeof(_aAdminPasswordSalt),
                                   10000,
                                   aHash,
                                   sizeof(aHash));
    if (CRYPTO_MEMDIF(aHash, _aAdminHashedPassword, sizeof(aHash)) == 0) {
      Status = 0;
    } else {
      Status = SSH_ERROR_USERAUTH_FAIL;
    }
  } else {
    Status = SSH_ERROR_USERAUTH_FAIL;
  }
  //
  return Status;
}

/*********************************************************************
*
*       Public code
*
**********************************************************************
*/

/*********************************************************************
*
*       MainTask()
*
*  Function description
*    Application entry point.
```

```c
*/
void MainTask(void) {
  SSH_SESSION * pSession;
  int          BoundSocket;
  int          Socket;
  int          Status;
  //
  SEGGER_SYS_Init();
  SEGGER_SYS_IP_Init();
  SSH_Init();
  //
  SEGGER_SYS_IO_Printf("\nemSSH V%s - Shell4 compiled " __DATE__ "
 " __TIME__ "\n",
                       SSH_GetVersionText());
  SEGGER_SYS_IO_Printf("%s    www.segger.com\n\n",
                       SSH_GetCopyrightText());
  //
  // Allow "none" and "password" user authentication.
  //
  SSH_SERVICE_Add(&SSH_SERVICE_USERAUTH, NULL);
  SSH_USERAUTH_METHOD_Add(&SSH_USERAUTH_METHOD_NONE,     _UserauthRequestNone);
  SSH_USERAUTH_METHOD_Add(&SSH_USERAUTH_METHOD_PASSWORD, _UserauthRequestPassword);
  //
  // Add support for interactive shells.
  //
  SSH_CHANNEL_REQUEST_Add(&SSH_CHANNEL_REQUEST_SHELL,         _ShellRequest);
  SSH_CHANNEL_REQUEST_Add(&SSH_CHANNEL_REQUEST_ENV,          NULL);
  SSH_CHANNEL_REQUEST_Add(&SSH_CHANNEL_REQUEST_PTYREQ,        _TerminalRequest);
  SSH_CHANNEL_REQUEST_Add(&SSH_CHANNEL_REQUEST_WINDOW_CHANGE, NULL);
  //
  // Bind SSH port.
  //
  BoundSocket = SEGGER_SYS_IP_Bind(22);
  if (BoundSocket < 0) {
    _AppExit("Cannot bind port 22!");
  }
  //
  for (;;) {
    //
    // Wait for an incoming connection.
    //
    do {
      Socket = SEGGER_SYS_IP_Accept(BoundSocket);
    } while (Socket < 0);
    //
    SSH_SESSION_Alloc(&pSession);
    if (pSession == 0) {
      _AppExit("No available session!");
    }
    //
    SSH_SESSION_Init(pSession, Socket, &_IP_Transport);
    SSH_SESSION_ConfBuffers(pSession,
                            _aRxTxBuffer, sizeof(_aRxTxBuffer),
                            _aRxTxBuffer, sizeof(_aRxTxBuffer));
    do {
      Status = SSH_SESSION_Process(pSession);
    } while (Status >= 0);
  }
}

/************************** End of file **************************/
```

3.6   Displaying a warning banner

In some jurisdictions it may be desirable, or even necessary, to display a warning or legal notice before logging in. The SSH protocol supports this and emSSH provides the mechanism to implement this.

For a complete listing of this application, see *SSH_Shell5.c complete listing* on page 68.

3.6.1   Hooking the user authentication service

Before a user can log in, the client requests the authentication service. During authentication and before password entry, there is a possibility to send the client a banner to display to the user that warns them that, for instance, all logins attempts are recorded.

This example isn't so menacing, and we start by defining the banner that we wish to display to the user:

```
#define BANNER \
  "\r\n"                                                              \
  "**************************************************************\r\n" \
  "* This server is powered by SEGGER emSSH.  It simply works! *\r\n" \
  "**************************************************************\r\n" \
  "\r\n"
```

We hook requests for the user authentication service and direct them to a callback function that will display the banner:

```
//
// Hook user authentication to display a banner.  Allow "none"
// and "password" user authentication.
//
SSH_SERVICE_Add(&SSH_SERVICE_USERAUTH, _UserauthServiceRequest);
SSH_USERAUTH_METHOD_Add(&SSH_USERAUTH_METHOD_NONE,     _UserauthRequestNone);
SSH_USERAUTH_METHOD_Add(&SSH_USERAUTH_METHOD_PASSWORD, _UserauthRequestPassword);
```

The user authentication service callback is:

```
static int _UserauthServiceRequest(      SSH_SESSION * pSession,
                                   const char        * sServiceName) {  ❶
  int Status;
  //
  Status = SSH_SESSION_SendServiceAccept(pSession, sServiceName);  ❷
  if (Status >= 0) {
    Status = SSH_SESSION_SendUserauthBanner(pSession, BANNER, "en");  ❸
  }
  //
  return Status;  ❹
}
```

The code is fairly straightfoward:

❶   **Receive user authentication service parameters**

The service name will always be "ssh-userauth" as this is the service that we hooked.

❷   **Accept the service**

As this callback replaces emSSH's default of accepting installed services, it's the callback's responsibility to accept the service by using SSH_SESSION_SendServiceAccept() to send a service accept message to the peer.

❸   **Ask the peer to display a banner**

With the service accepted, the server is cleared to send a user authentication banner to the client using `SSH_SESSION_SendUserauthBanner()`. The final parameter is the language tag that indicates to the client the language the banner appears in. It is unclear exactly how the client is supposed to handle the language tag—hard-coding this to "en" is good enough in most cases as English is widely understood.

❹   **Return the service accept status**

By now this should be a familiar idiom.

3.6.2   Seeing the banner

Testing the application is the same as previous examples:

```
MacBook:~ paul$ ssh admin@10.0.0.247

*************************************************************
* This server is powered by SEGGER emSSH.  It simply works! *
*************************************************************

admin@10.0.0.247's password: secret

Welcome to the emSSH command line!  Type Ctrl+D to exit.

emSSH> ls -l
...ls -l
emSSH>

Bye!

Connection to 10.0.0.247 closed.
MacBook:~ paul$ _
```

3.6.3   SSH_Shell5.c complete listing

```c
/*********************************************************************
*                  (c) SEGGER Microcontroller GmbH                   *
*                      The Embedded Experts                          *
*                         www.segger.com                             *
**********************************************************************

----------------------- END-OF-HEADER -----------------------------

File        : SSH_Shell5.c
Purpose     : SSH server that adds a warning banner.

*/

/*********************************************************************
*
*       #include Section
*
**********************************************************************
*/

#include "SSH.h"
#include "SEGGER_SYS.h"
#include <string.h>

/*********************************************************************
*
*       Defines, configurable
*
**********************************************************************
*/

#define PROMPT                                                      \
  "emSSH> "

#define SIGNON                                                      \
  "\r\n"                                                            \
  "Welcome to the emSSH command line!  Type Ctrl+D to exit.\r\n"    \
  "\r\n"                                                            \
  PROMPT

#define BANNER \
  "\r\n"                                                              \
  "****************************************************************\r\n" \
  "* This server is powered by SEGGER emSSH.  It simply works! *\r\n" \
  "****************************************************************\r\n" \
  "\r\n"

/*********************************************************************
*
*       Prototypes
*
**********************************************************************
*/

void MainTask(void);
static int _TerminalChannelData(       SSH_SESSION * pSession,
                                       unsigned       Channel,
                                 const U8           * pData,
                                       unsigned       DataLen);

/*********************************************************************
*
*       Static const data
*
**********************************************************************
```

```c
*/

static const SSH_TRANSPORT_API _IP_Transport = {
  SEGGER_SYS_IP_Send,
  SEGGER_SYS_IP_Recv,
  SEGGER_SYS_IP_Close,
};

static const SSH_CHANNEL_API _TerminalAPI = {
  _TerminalChannelData,
  NULL,
  NULL,
  NULL
};

/*********************************************************************
*
*       Static data
*
**********************************************************************
*/

static U8       _aRxTxBuffer[8192];
static U8       _aCommandLine[70];
static unsigned _Cursor;

/*********************************************************************
*
*       Static code
*
**********************************************************************
*/

/*********************************************************************
*
*       _AppExit()
*
*  Function description
*    Exit the application with an error.
*
*  Parameters
*    sReason - Reason for exit, displayed for the user.
*/
static void _AppExit(const char *sReason) {
  SEGGER_SYS_IO_Printf(sReason);
  SEGGER_SYS_OS_Halt(100);
}

/*********************************************************************
*
*       _ShellRequest()
*
*  Function description
*    Handle a shell channel request.
*
*  Parameters
*    pSession - Pointer to session.
*    Channel  - Local channel receiving the data.
*    pParas   - Pointer to channel request parameters.
*
*  Return value
*    >= 0 - Success.
*    <  0 - Error.
*/
static int _ShellRequest(SSH_SESSION               * pSession,
                         unsigned                    Channel,
                         SSH_CHANNEL_REQUEST_PARAS * pParas) {
  int Status;
```

```c
      //
      Status = SSH_CHANNEL_SendData(pSession, Channel,
                                    SIGNON, (unsigned)strlen(SIGNON));
      if (Status < 0) {
        Status = SSH_CHANNEL_SendFailure(pSession, Channel);
      } else if (pParas->WantReply) {
        Status = SSH_CHANNEL_SendSuccess(pSession, Channel);
      }
      //
      return Status;
}

/*********************************************************************
*
*       _TerminalChannelData()
*
*  Function description
*    Handle data received from peer.
*
*  Parameters
*    pSession - Pointer to session.
*    Channel  - Local channel receiving the data.
*    pData    - Pointer to object that contains the data.
*    DataLen  - Octet length of the object that contains the data.
*
*  Return value
*    >= 0 - Success.
*    <  0 - Error.
*
*  Additional information
*    Provide in-callback handling of a command line processor.
*    This sample supports only one connection at a time.
*/
static int _TerminalChannelData(      SSH_SESSION * pSession,
                                      unsigned      Channel,
                                const U8          * pData,
                                      unsigned      DataLen) {
  unsigned i;
  U8       Ch;
  int      Status;
  //
  Status = 0;
  //
  for (i = 0; Status >= 0 && i < DataLen; ++i) {
    Ch = pData[i];
    if (0x20 <= Ch && Ch <= 0x7E) {
      if (_Cursor < sizeof(_aCommandLine)) {
        _aCommandLine[_Cursor++] = Ch;
        Status = SSH_CHANNEL_SendData(pSession, Channel, &Ch, 1);
      }
    } else if (Ch == 0x08 || Ch == 0x7F) {
      if (_Cursor > 0) {
        --_Cursor;
        Status = SSH_CHANNEL_SendData(pSession, Channel, "\b \b", 3);
      }
    } else if (Ch == '\r') {
      SSH_CHANNEL_SendData(pSession, Channel, "\r\n...", 5);
      SSH_CHANNEL_SendData(pSession, Channel, _aCommandLine, _Cursor);
      SSH_CHANNEL_SendData(pSession, Channel, "\r\n", 3);

    Status = SSH_CHANNEL_SendData(pSession, Channel, PROMPT, (unsigned)strlen(PROMPT));
      _Cursor = 0;
    } else if (Ch == 0x04) {
      SSH_CHANNEL_SendData(pSession, Channel, "\r\n\r\nBye!\r\n\r\n", 12);
      SSH_CHANNEL_Close(pSession, Channel);
      break;
    }
  }
```

```c
  //
  return Status;
}

/**********************************************************************
*
*       _TerminalRequest()
*
*  Function description
*    Request a terminal.
*
*  Parameters
*    pSession - Pointer to session.
*    Channel  - Local channel requesting the terminal.
*    pParas   - Pointer to channel request parameters.
*
*  Return value
*    >= 0 - Success.
*    <  0 - Error.
*/
static int _TerminalRequest(SSH_SESSION              * pSession,
                            unsigned                   Channel,
                            SSH_CHANNEL_REQUEST_PARAS * pParas) {
  int Status;
  //
  SSH_CHANNEL_Config(pSession, Channel, 128, &_TerminalAPI, 0);
  if (pParas->WantReply) {
    Status = SSH_CHANNEL_SendSuccess(pSession, Channel);
  } else {
    Status = 0;
  }
  //
  return Status;
}

/**********************************************************************
*
*       _UserauthServiceRequest()
*
*  Function description
*    Request the user authentication service.
*
*  Parameters
*    pSession     - Pointer to session.
*    sServiceName - Service being requested.
*
*  Return value
*    >= 0 - Success.
*    <  0 - Error.
*
*  Additional information
*    Displays a banner before user authentication commences.
*/
static int _UserauthServiceRequest(SSH_SESSION *pSession, const char *sServiceName) {
  int Status;
  //
  Status = SSH_SESSION_SendServiceAccept(pSession, sServiceName);
  if (Status >= 0) {
    Status = SSH_SESSION_SendUserauthBanner(pSession, BANNER, "en");
  }
  //
  return Status;
}

/**********************************************************************
*
*       _UserauthRequestNone()
*
```

```
*  Function description
*    Request authentication of user with method "none".
*
*  Parameters
*    pSession  - Pointer to session.
*    pReqParas - Pointer to user authentication request parameters.
*
*  Return value
*    >= 0 - Success.
*    <  0 - Error.
*/
static int _UserauthRequestNone(SSH_SESSION                * pSession,
                                SSH_USERAUTH_REQUEST_PARAS * pReqParas) {
  SSH_USERAUTH_NONE_PARAS NoneParas;
  int                     Status;
  //
  SSH_USE_PARA(pSession);
  //
  Status = SSH_USERAUTH_NONE_ParseParas(pReqParas, &NoneParas);
  if (Status < 0) {
    Status = SSH_ERROR_USERAUTH_FAIL;
  } else if (pReqParas->UserNameLen == 4 &&
             SSH_MEMCMP(pReqParas->pUserName, "anon", 4) == 0) {
    Status = 0;
  } else {
    Status = SSH_ERROR_USERAUTH_FAIL;
  }
  //
  return Status;
}

/*********************************************************************
*
*       _UserauthRequestPassword()
*
*  Function description
*    Request authentication of user with method "password".
*
*  Parameters
*    pSession  - Pointer to session.
*    pReqParas - Pointer to user authentication request parameters.
*
*  Return value
*    >= 0 - Success.
*    <  0 - Error.
*/
static int _UserauthRequestPassword(SSH_SESSION                * pSession,
                                    SSH_USERAUTH_REQUEST_PARAS * pReqParas) {
  SSH_USERAUTH_PASSWORD_PARAS PasswordParas;
  int                         Status;
  //
  SSH_USE_PARA(pSession);
  //
  Status = SSH_USERAUTH_PASSWORD_ParseParas(pReqParas, &PasswordParas);
  if (Status < 0) {
    Status = SSH_ERROR_USERAUTH_FAIL;
  } else if (pReqParas->UserNameLen == 5 &&
             memcmp(pReqParas->pUserName, "admin", 5) == 0) {
    if (PasswordParas.PasswordLen == 6 &&
        memcmp(PasswordParas.pPassword, "secret", 6) == 0) {
      Status = 0;
    } else {
      Status = SSH_ERROR_USERAUTH_FAIL;
    }
  } else {
    Status = SSH_ERROR_USERAUTH_FAIL;
  }
  //
```

```c
  return Status;
}

/**********************************************************************
*
*             Public code
*
**********************************************************************
*/

/**********************************************************************
*
*       MainTask()
*
*  Function description
*    Application entry point.
*/
void MainTask(void) {
  SSH_SESSION * pSession;
  int           BoundSocket;
  int           Socket;
  int           Status;
  //
  SEGGER_SYS_Init();
  SEGGER_SYS_IP_Init();
  SSH_Init();
  //
  SEGGER_SYS_IO_Printf("\nemSSH V%s - Shell5 compiled " __DATE__ "
 " __TIME__ "\n",
                       SSH_GetVersionText());
  SEGGER_SYS_IO_Printf("%s    www.segger.com\n\n",
                       SSH_GetCopyrightText());
  //
  // Hook user authentication to display a banner.  Allow "none"
  // and "password" user authentication.
  //
  SSH_SERVICE_Add(&SSH_SERVICE_USERAUTH, _UserauthServiceRequest);
  SSH_USERAUTH_METHOD_Add(&SSH_USERAUTH_METHOD_PASSWORD, _UserauthRequestPassword);
  SSH_USERAUTH_METHOD_Add(&SSH_USERAUTH_METHOD_NONE,     _UserauthRequestNone);
  //
  // Add support for interactive shells.
  //
  SSH_CHANNEL_REQUEST_Add(&SSH_CHANNEL_REQUEST_SHELL,         _ShellRequest);
  SSH_CHANNEL_REQUEST_Add(&SSH_CHANNEL_REQUEST_ENV,           NULL);
  SSH_CHANNEL_REQUEST_Add(&SSH_CHANNEL_REQUEST_PTYREQ,        _TerminalRequest);
  SSH_CHANNEL_REQUEST_Add(&SSH_CHANNEL_REQUEST_WINDOW_CHANGE, NULL);
  //
  // Bind SSH port.
  //
  BoundSocket = SEGGER_SYS_IP_Bind(22);
  if (BoundSocket < 0) {
    _AppExit("Cannot bind port 22!");
  }
  //
  for (;;) {
    //
    // Wait for an incoming connection.
    //
    do {
      Socket = SEGGER_SYS_IP_Accept(BoundSocket);
    } while (Socket < 0);
    //
    SSH_SESSION_Alloc(&pSession);
    if (pSession == 0) {
      _AppExit("No available session!");
    }
    //
    SSH_SESSION_Init(pSession, Socket, &_IP_Transport);
```

```
      SSH_SESSION_ConfBuffers(pSession,
                              _aRxTxBuffer, sizeof(_aRxTxBuffer),
                              _aRxTxBuffer, sizeof(_aRxTxBuffer));
      do {
        Status = SSH_SESSION_Process(pSession);
      } while (Status >= 0);
    }
  }

/*************************** End of file ***************************/
```

3.7   Multiple shells using an RTOS

All the previous examples have been *reactive* in that they output small quantities of data only when new data arrives through keyboard input. This is adequate for a very simple shell, but does not cater for asynchronous I/O. In this example we develop a framework that can handle multiple shells, with buffering, using an RTOS. But, of course, it's equally applicable to a single shell instance.

This example uses SEGGER embOS, but it should be portable to other RTOS APIs with minimal effort. Only the more important, changed pieces are described.

3.7.1   Division of labor

The architecture of this example is to spawn two tasks per shell:

- a task to handle the messaging layer of the SSH protocol and to deal with emptying and filling application buffers;
- a task that runs as the client shell task that is decoupled from SSH and interfaces to the outside using some high-level functions.

3.7.2   Intertask communication

Communication between the tasks is by way of two ring buffers, one for console input and one for console output. The ring buffer is defined by this structure:

```
typedef struct {
  volatile unsigned RdPtr;   // Read pointer
  volatile unsigned WrPtr;   // Write pointer
  volatile unsigned RdCnt;   // Number of bytes read
  volatile unsigned WrCnt;   // Number of bytes written
  unsigned          Capacity;
  U8              * pData;
} RING_BUFFER;
```

The concept is that a task writing into the ring buffer updates (writes) the write pointer member `WrPtr` and a task reading the ring buffer to retrieve data updates the read pointer member `RdPtr`. The ring buffer is empty when the read and write pointers are equal. As data is written into the ring buffer, the write pointer advances, and as data is read out, the read pointer advances to meet it.

This scheme is well known and has the advantage that, for a single reader and a single writer, it requires no locking whatsoever, i.e. it is lock free: only the reader advances the read pointer and only the writer advances the write pointer.

The read and write pointers are declared volatile because the buffer could be written, or read, from an interrupt service routine and change without the compiler being aware of it. In our case, it's a second thread of execution as a task that will modify the pointer behind the compiler's back.

3.7.2.1   Writing to the ring buffer

Writing into the ring buffer is as follows:

```
static unsigned _RING_BUFFER_Wr(      RING_BUFFER * pRB,
                                const void        * pData,
                                      unsigned      DataLen) {
  unsigned N;
  unsigned WrPtr;
  unsigned LChunkLen;
  unsigned RChunkLen;
  //
  N = _RING_BUFFER_QueryCanWrite(pRB);  ❶
  if (N < DataLen) {
    DataLen = N;
```

```
    }
    //
    WrPtr = pRB->WrPtr;
    if (WrPtr + DataLen <= pRB->Capacity) {   ❷
      memcpy(pRB->pData+WrPtr, pData, DataLen);
      WrPtr += DataLen;
      if (WrPtr == pRB->Capacity) {
        WrPtr = 0;
      }
      pRB->WrPtr = WrPtr;
    } else {
      //
      LChunkLen = pRB->Capacity - WrPtr;
      RChunkLen = DataLen - LChunkLen;
      //
      memcpy(pRB->pData+WrPtr, pData, LChunkLen);
      memcpy(pRB->pData, (U8 *)pData+LChunkLen, RChunkLen);
      //
      pRB->WrPtr = RChunkLen;
    }
    //
    pRB->WrCnt += DataLen;   ❸
    //
    return DataLen;   ❶
}
```

❶ **Prevent buffer overflow**

If too much data is written to the buffer, it does not overflow, it only becomes full. The return value of the function indicates the number of bytes written to the buffer.

❷ **Handle buffer wrap**

Data written to a ring buffer "wraps around" the end of the buffer. The code manages this by deciding whether the data will or will not wrap around: if it doesn't, there's one write, and if it does, it's written in two pieces.

❸ **Update write count**

The difference between the write count and read count is the number of characters that are yet to be read from the buffer, and is more efficient to compute than handling the wrapping read and write pointers.

3.7.2.2 Reading from the ring buffer

Reading from the ring buffer follows the write framework and is not further explained:

```
static unsigned _RING_BUFFER_Rd(RING_BUFFER * pRB,
                                 void        * pData,
                                 unsigned      DataLen) {
  unsigned RdPtr;
  unsigned LChunkLen;
  unsigned RChunkLen;
  //
  RdPtr = _RING_BUFFER_QueryCanRead(pRB);
  if (RdPtr < DataLen) {
    DataLen = RdPtr;
  }
  //
  if (DataLen == 0) {
    return DataLen;
  }
  //
  RdPtr = pRB->RdPtr;
  if (RdPtr + DataLen <= pRB->Capacity) {
    memcpy(pData, pRB->pData+RdPtr, DataLen);
```

```
    RdPtr += DataLen;
    if (RdPtr == pRB->Capacity) {
      RdPtr = 0;
    }
    pRB->RdPtr = RdPtr;
  } else {
    LChunkLen = pRB->Capacity - RdPtr;
    RChunkLen = DataLen - LChunkLen;
    //
    memcpy(pData, pRB->pData+RdPtr, LChunkLen);
    memcpy((char *)pData+LChunkLen, pRB->pData, RChunkLen);
    //
    pRB->RdPtr = RChunkLen;
  }
  //
  pRB->RdCnt += DataLen;
  //
  return DataLen;
}
```

3.7.3   Starting the shell

For previous examples, the shell request callback did nothing or very little, such as writing a sign-on message. Now, the shell request callback does what a regular SSH daemon would do, and that is it starts a dedicated shell task:

```
static int _ShellRequest(SSH_SESSION                * pSession,
                         unsigned                     Channel,
                         SSH_CHANNEL_REQUEST_PARAS * pParas) {
  SHELL_CONTEXT * pShell;
  int             Status;
  //
  Status = 0;
  if (pParas->WantReply) {
    Status = SSH_CHANNEL_SendSuccess(pSession, Channel);
  }
  //
  pShell = &_aShell[SSH_SESSION_QueryIndex(pSession)];  ❶
  SSH_CHANNEL_Config(pSession, Channel, 128, &_TerminalAPI, pShell);
  OS_CreateTaskEx(&pShell->ShellTask,
                  "ShellTask",
                  100,
                  _ShellMain,
                  &pShell->aShellTaskStack,
                  sizeof(pShell->aShellTaskStack),
                  10,
                  pSession);
  //
  return Status;
}
```

The callback is much as before but the channel associated with the shell is configured when the shell starts rather than when the terminal is requested.

The shell task is written to be very simple: it uses high-level C-style get and put stream functions that read and write the ring buffers. Because it uses these functions, and the task never calls any SSH function directly, it is completely decoupled from SSH and, as such, the ring buffer could be emptied by a regular telnet task, or a task that manages a plain serial line.

```
static void _ShellMain(void *pContext) {
  SSH_SESSION   * pSession;
  SHELL_CONTEXT * pShell;
  //
  pSession = pContext;
```

```
    pShell = &_aShell[SSH_SESSION_QueryIndex(pSession)];    ❶
    //
    _Puts(pShell, SIGNON);
    for (;;) {
      _Puts(pShell, PROMPT);
      if (_Gets(pShell) == 0) {
        _Puts(pShell, "\r\n...");
        _Puts(pShell, pShell->aCommandLine);
        _Puts(pShell, "\r\n");
      } else {
        pShell->Exit = 1;
      }
    }
  }
```

The only notable feature is that at ❶ we construct a pointer to a shell context we maintain. You'll see this idiom throughout the code.

3.7.4   Efficient output

We now turn our attention to the framework that will service the ring buffers, from both tasks, in order to run the SSH protocol and shell efficiently.

Placing data into the ring buffer when it has remaining space for it is no problem, but when the ring buffer doesn't is the challenge. One very simple implementation scheme would be to poll, but this soaks up processor cycles and is highly inefficient. Many engineers move to a more cycle-efficient scheme by using the RTOS "delay" function to yield processor time to other tasks and not monopolize the CPU. This is an admirable attempt, but it introduces an unintended latency into "restarting" the task once the buffer is full because the task waits for the entire delay period each and every time.

We choose an efficient implementation that has next to zero latency restarting the task when the buffer empties:

```
static void _Puts(SHELL_CONTEXT *pShell, const char *sText) {
  unsigned Len;
  unsigned ChunkLen;
  //
  Len = strlen(sText);
  //
  while (Len > 0) {    ❶
    ChunkLen = _RING_BUFFER_Wr(&pShell->TxBuffer, sText, Len);    ❷
    sText += ChunkLen;
    Len   -= ChunkLen;
    if (Len != 0) {    ❸
      OS_Delay(10000);
    }
  }
}
```

❶   **Write in pieces**

The loop iterates until all data have been transferred to the ring buffer.

❷   **Try to write**

This step tries to write all the data to the ring buffer. If the ring buffer becomes full, only part of that data is written and the return value is the length of the chunk successfully transferred. The data pointer and length are updated to step over the written chunk, ready for the next iteration.

❸   **Wait for the ring buffer empty**

If there is data remaining to be written, it's because the ring buffer is full and cannot accept more data. If this is the case, the calling task is delayed whilst the ring buffer empties. The

delay length is set to ten seconds, but this delay is arbitrarily long: when the ring buffer is emptied by the protocol task and more space is made available in the transmission ring buffer, the shell task is prematurely woken from the delay using `OS_WakeTask` (refer to *Buffering incoming data* on page 79 to see this side of the implementation).

3.7.5   Efficient input

Reading characters is much the same as writing. The `_Getc` function blocks until a character is available and returns that character:

```
static U8 _Getc(SHELL_CONTEXT *pShell) {
  U8 Ch;
  //
  while (_RING_BUFFER_QueryCanRead(&pShell->RxBuffer) == 0) {  ❶
    OS_Delay(10000);
  }
  _RING_BUFFER_Rd(&pShell->RxBuffer, &Ch, 1);  ❷
  return Ch;
}
```

❶ **Wait for buffer fill**

The function `_RING_BUFFER_QueryCanRead()` returns the number of characters that are immediately available in the ring buffer. As we are trying to read one character, we wait for the ring buffer to be nonempty. If there are no characters, the task is put to sleep and, rather like the transmit task, is prematurely woken when the SSH channel delivers more characters.

❷ **Read the character**

We know the buffer is nonempty and can deliver at least one character, so read and return it.

3.7.6   Buffering incoming data

New channel data is delivered by the channel data callback, which you have seen before. Previously we echoed that data to the terminal from within the callback function, but now we send the data to the receive ring buffer:

```
static int _TerminalChannelData(       SSH_SESSION * pSession,
                                       unsigned      Channel,
                                 const U8          * pData,
                                       unsigned      DataLen) {
  SHELL_CONTEXT *pShell;
  //
  pShell = &_aShell[SSH_SESSION_QueryIndex(pSession)];  ❶
  _RING_BUFFER_Wr(&pShell->RxBuffer, pData, DataLen);  ❷
  OS_WakeTask(&pShell->ShellTask);      ❸
  //
  return 0;
}
```

❶ **Get a reference to the shell data**

This was shown before, but we repeat it here.

❷ **Write the incoming data to the ring buffer.**

All incoming data is written to the ring buffer. You might wonder what happens when the ring buffer is full and more data is presented through the SSH protocol to the callback? In fact, this *should not* happen as the SSH protocol uses a windowing scheme rather like TCP where the transmitter knows exactly how much space is available in the receiver's buffer and never sends more than the receiver can accept. This window is maintained in

both directions by the SSH protocol: emSSH fully implements this windowing scheme when receiving, and requires minor client involvement on transmission.

The above says "should not happen." What if it does? That is a protocol error and this callback simply discards the data that it cannot store. To emphasize, this should not happen but, if it does, it will not cause emSSH to fail.

❸ **Wake up the receiver**

We know that we have written data to the receive buffer and, therefore, it cannot be empty, and now it's time to wake up the receiver if it's suspended waiting for input. If the receiver is already awake, the call to `OS_WakeTask()` is benign and does nothing.

3.7.7 Handling the SSH protocol

The SSH protocol is handled by a task dedicated to service the connection:

```
static void _ProtocolMain(void *pContext) {
  SSH_SESSION   * pSession;
  int             Socket;
  int             Status;
  //
  pSession = pContext;
  Socket = SSH_SESSION_QuerySocket(pSession);   ❶
  for (;;) {
    Status = 0;
    if (SEGGER_SYS_IP_QueryCanRead(Socket)) {   ❷
      Status = SSH_SESSION_Process(pSession);
    } else {
      Status = SSH_SESSION_IterateChannels(pSession, _ServiceChannel);   ❸
    }
    if (Status < 0) {   ❹
      OS_TerminateTask(0);
    } else if (Status == 0) {   ❺
      OS_Delay(10);
    }
  }
}
```

❶ **Recover socket**

The function `SSH_SESSION_QuerySocket()` returns the socket index associated with a session set up using `SSH_SESSION_Init()`. We use the socket index to call socket-related functions.

❷ **Inquire if protocol data is ready**

The function `SEGGER_SYS_IP_QueryCanRead()` returns nonzero when data is available to read from the socket. If there is data ready, we process it through the SSH state machine using `SSH_SESSION_Process()`.

❸ **Handle channel data I/O**

If there is no data on the socket, the task sees if any channel in the session requires servicing by iterating over them using `SSH_SESSION_IterateChannels()` with the callback `_ServiceChannel()`.

❹ **Handling an error**

If there is an error during processing, the protocol task is terminated.

❺ **Idling**

After iterating all channels with none of them needing processing, the protocol task is put to sleep in the expectation that some data will arrive on the socket or through the ring

buffer. This is a slight inefficiency, but there is no easily-presented mechanism in embOS and embOS/IP to wait on a socket and some other object simultaneously, so we resort to a simple polling scheme.

3.7.8   Servicing channels

Each channel in a SSH session is serviced from the protocol task, in this example, with the `_ServiceChannel` callback:

```
static int _ServiceChannel(SSH_SESSION *pSession, unsigned Channel) {
  SHELL_CONTEXT * pShell;
  unsigned        Len;
  U8              aData[64];
  int             Status;
  //
  pShell = &_aShell[SSH_SESSION_QueryIndex(pSession)];
  if (pShell == 0) {
    Status = 0;
  } else if (pShell->Exit) {    ❶
    if (SSH_CHANNEL_QueryCanWrite(pSession, Channel) >= 12) {
      Status = SSH_CHANNEL_SendData(pShell->pSession, Channel,
                                    "\r\n\r\nBye!\r\n\r\n", 12);
    }
    SSH_SESSION_Disconnect(pShell->pSession, SSH_DISCONNECT_BY_APPLICATION);
    Status = -1;
  } else if (_RING_BUFFER_QueryCanRead(&pShell->TxBuffer) > 0) {   ❷
    Len = _RING_BUFFER_QueryCanRead(&pShell->TxBuffer);    ❸
    Len = SEGGER_MIN(Len, SSH_CHANNEL_QueryCanWrite(pShell->pSession, Channel));
    ❹
    Len = SEGGER_MIN(Len, sizeof(aData));    ❺
    if (Len > 0) {
      _RING_BUFFER_Rd(&pShell->TxBuffer, aData, Len);      ❻
      Status = SSH_CHANNEL_SendData(pShell->pSession, Channel, aData, Len);
      OS_WakeTask(&pShell->ShellTask);
    }
    Status = 1;   ❼
  } else {
    Status = 0;   ❽
  }
  //
  return Status;
}
```

❶   **Handle an exit request**

Receiving Ctrl+D in the shell task sets the `Exit` flag in the shell context. When the protocol task sees that the shell has requested an exit, it writes a termination message to the peer and disconnects.

The call to `SSH_CHANNEL_QueryCanWrite()` is something that hasn't been presented before. Remember that there is a windowing mechanism for SSH channels and that a transmitter should not send more data than the receiver is able to accept. This code inquires whether the receiver is able to accept 12 bytes and, if so, proceeds to write the log-off message. This is perfectly acceptable because, immediately afterwards, the session is disconnected and no further data is sent to the receiver.

❷   **Limit data transfer size**

Using `_RING_BUFFER_QueryCanRead`, the protocol task sees if there is any data in the ring buffer that could possibly be sent to the receiver. If there is, at ❸ it notes the size of the outstanding data in the transmit buffer. At ❹ it reduces the size to that which can be accepted by the receiver. And at ❺ it limits the size to that which can be stored in a local buffer.

Note that it is entirely possible to remove the local buffer and transmit directly from the ring buffer, but this makes the code more complex and less readable. In the best textbook tradition, this is left as an exercise for the reader.

➏   Send data to receiver

Once the maximum transfer fragment length is calculated, the data is read from the ring buffer and sent to the receiver with `SSH_CHANNEL_SendData()`. Once the data is read and sent, we know that the ring buffer is nonfull and, because it is nonfull, the shell task can be woken if it is suspended waiting for the buffer to empty.

At ➐ we set the return status to one to indicate that we have done some processing and sent data to the receiver, and at ➑ we set it to zero to indicate that the channel state didn't change. This status code is returned by the service callback, passes through `SSH_SESSION_IterateChannels()`, and guides `_ProtocolMain` actions—see *Handling the SSH protocol* on page 80.

3.7.9   Closing down

It's possible for both ends of a connection to terminate a session. When a session is terminated from the client, the SSH server must clean up after itself. We haven't needed to run any special processing in previous examples but, as we now have tasks, this example needs to shut down the session cleanly.

When a channel closes, emSSH activates the `pfOnChannelClose()` callback in the channel API and this is exactly what we need in our example:

```
static const SSH_CHANNEL_API _TerminalAPI = {
  _TerminalChannelData,
  0,
  0,
  _TerminalChannelClose,
};
```

The code that deals with channel closure is straightforward: it terminates the shell task that is associated with the channel.

```
static void _TerminalChannelClose(SSH_SESSION * pSession, unsigned Channel) {
  SHELL_CONTEXT *pShell;
  //
  SSH_USE_PARA(Channel);
  //
  pShell = &_aShell[SSH_SESSION_QueryIndex(pSession)];
  OS_TerminateTask(&pShell->ShellTask);
}
```

3.7.10   Starting up

It may seem strange to show the startup last, but it's easy to code and is merely a rework of the previous startup code:

```
for (;;) {
  //
  // Try to allocate a session for the next incoming connection.
  //
  SSH_SESSION_Alloc(&pSession);   ❶
  if (pSession == 0) {
    OS_Delay(100);
    continue;
  }
  //
  do {
    Socket = SEGGER_SYS_IP_Accept(BoundSocket);
  } while (Socket < 0);
```

```
    //
    SessionIndex = SSH_SESSION_QueryIndex(pSession);    ❷
    pShell = &_aShell[SessionIndex];
    memset(pShell, 0, sizeof(*pShell));
    pShell->pSession = pSession;
    SSH_SESSION_Init(pShell->pSession, Socket, &_IP_Transport);
    SSH_SESSION_ConfBuffers(pShell->pSession,
                            pShell->aRxTxBuffer, sizeof(pShell->aRxTxBuffer),
                            pShell->aRxTxBuffer, sizeof(pShell->aRxTxBuffer));
    _RING_BUFFER_Init(&pShell->RxBuffer, pShell->aRxData, sizeof(pShell->aRxData));
    _RING_BUFFER_Init(&pShell->TxBuffer, pShell->aTxData, sizeof(pShell->aTxData));
    //
    OS_CreateTaskEx(&pShell->ProtocolTask,    ❸
                    "ProtocolTask",
                    100,
                    _ProtocolMain,
                    &pShell->aProtocolTaskStack,
                    sizeof(pShell->aProtocolTaskStack),
                    10,
                    pSession);
}
```

❶   **Wait for a new session**

This waits for an incoming connection only if there is an SSH session available to service it.

❷   **Set up the session**

This sets up the per-session data which is an array of session data indexed by the session number. The session number, or session index, for a session is returned by `SSH_SESSION_QueryIndex()`.

❸   **Start the protocol task**

Once the session is set up, a new SSH protocol handling task is started to handle that specific connection and the loop continues to handle other incoming connections. The SSH protocol task services channel requests and starts shell processes as necessary and acts as an "embedded `sshd`".

This concludes our last example. You should now be familiar with how emSSH can be deployed in a user application.

3.7.11   SSH_Shell6.c complete listing

```
/**********************************************************************
*                  (c) SEGGER Microcontroller GmbH                   *
*                      The Embedded Experts                          *
*                         www.segger.com                             *
**********************************************************************


----------------------- END-OF-HEADER ----------------------------

File        : SSH_Shell6.c
Purpose     : SSH server that offers simultaneous input and output.

*/

/**********************************************************************
*
*       #include Section
*
**********************************************************************
*/

#include "SSH.h"
#include "IP.h"
#include "RTOS.h"

/**********************************************************************
*
*       Defines, configurable
*
**********************************************************************
*/

#define PROMPT                                                 \
  "emSSH> "

#define SIGNON                                                 \
  "\r\n"                                                       \
  "Welcome to the emSSH command line!  Type Ctrl+D to exit.\r\n"     \
  "\r\n"

#define BANNER \
  "\r\n"                                                       \
  "***********************************************************\r\n" \
  "* This server is powered by SEGGER emSSH.  It simply works! *\r\n" \
  "***********************************************************\r\n" \
  "\r\n"

#define USE_RX_TASK 0

/**********************************************************************
*
*       Local data types
*
**********************************************************************
*/

//
// Task priorities
//
enum {
  TASK_PRIO_IP_APP = 150,
  TASK_PRIO_IP_TASK,
  // Priority should be higher than all IP application tasks.
  TASK_PRIO_IP_RX_TASK
  // Must be the highest priority of all IP related tasks, comment out to read packets in ISR
};
```

```c
typedef struct {
  volatile unsigned RdPtr;  // Read pointer
  volatile unsigned WrPtr;  // Write pointer
  volatile unsigned RdCnt;  // Number of bytes read
  volatile unsigned WrCnt;  // Number of bytes written
  unsigned          Capacity;
  U8              * pData;
} RING_BUFFER;

typedef struct {
  SSH_SESSION * pSession;
  unsigned      Cursor;
  int           Ready;
  int           Exit;
  int           Socket;
  RING_BUFFER   TxBuffer;
  RING_BUFFER   RxBuffer;
  OS_TASK       ShellTask;
  OS_TASK       ProtocolTask;
  char          aCommandLine      [70];
  U8            aTxData           [1024];
  U8            aRxData           [128];
  U32           aShellTaskStack   [128];
  U32           aProtocolTaskStack [512];
  U8            aTxBuffer         [4096];
  U8            aRxBuffer         [4096];
} SHELL_CONTEXT;

/*********************************************************************
*
*       Prototypes
*
**********************************************************************
*/

void        MainTask            (void);
static void _ShellMain          (void *pContext);
static void _ProtocolMain       (void *pContext);
static void _TerminalChannelClose(SSH_SESSION *pSession, unsigned Channel);
static int  _TerminalChannelData (     SSH_SESSION * pSession,
                                       unsigned      Channel,
                                 const U8          * pData,
                                       unsigned      DataLen);
static int  _Send
  (int Socket, const char *pData, int DataLen, int Flags);
static int  _Recv               (int Socket,
  char *pData, int DataLen, int Flags);


/*********************************************************************
*
*       Static const data
*
**********************************************************************
*/

static const SSH_TRANSPORT_API _IP_Transport = {
  _Send,
  _Recv,
  closesocket,
};

static const SSH_CHANNEL_API _TerminalAPI = {
  _TerminalChannelData,
  0,
  0,
  _TerminalChannelClose,
```

```c
};
/**********************************************************************
*
*       Static data
*
**********************************************************************
*/

//
// Assign one shell context per session
//
static SHELL_CONTEXT _aShell[SSH_CONFIG_MAX_SESSIONS];

static OS_STACKPTR int _IPStack[2048/sizeof(int)];
static OS_TASK        _IPTCB;

static OS_STACKPTR int _ServerStack[4096/sizeof(int)];
static OS_TASK        _ServerTCB;

#if USE_RX_TASK
static OS_STACKPTR int _IPRxStack[1024/sizeof(int)];
static OS_TASK        _IPRxTCB;
#endif

/**********************************************************************
*
*       Static code
*
**********************************************************************
*/

/**********************************************************************
*
*       _Bind()
*
*  Function description
*    Bind a TCP port.
*
*  Parameters
*    Port - Bound port.
*
*  Return value
*    >= 0 - Bound socket handle.
*    <  0 - Error.
*/
static int _Bind(int Port) {
  int    Socket;
  int    Status;
  int    Enable;
  struct sockaddr_in local_addr;
  //
  local_addr.sin_family = AF_INET;
  local_addr.sin_port = htons(Port);
  local_addr.sin_addr.s_addr = htonl(INADDR_ANY);
  //
  Socket = Status = socket(PF_INET, SOCK_STREAM, 0);
  if (Status >= 0) {
    Enable = 1;

  Status = setsockopt(Socket, SOL_SOCKET, SO_REUSEADDR, (char *)&Enable, sizeof(Enable));
  }
  if (Status >= 0) {
    Status = bind(Socket, (struct sockaddr *)&local_addr, sizeof(local_addr));
  }
  if (Status >= 0) {
    Status = listen(Socket, 10);
  }
```

```c
  return Status >= 0 ? Socket : Status;
}

/**********************************************************************
*
*       _Accept()
*
*  Function description
*    Accept an incoming connection.
*
*  Parameters
*    Socket - Socket bound to port.
*
*  Return value
*    >= 0 - Socket handle.
*    <  0 - Error.
*/
static int _Accept(int Socket) {
  struct sockaddr_in client_addr;
  int                client_addr_len;
  //
  client_addr_len = sizeof(client_addr);
  Socket = accept(Socket, (struct sockaddr *) &client_addr, &client_addr_len);
  if (Socket < 0) {
    return -1;
  } else {
    return Socket;
  }
}

/**********************************************************************
*
*       _Send()
*
*  Function description
*    Send data to socket.
*
*  Parameters
*    Socket  - Socket to write to.
*    pData   - Pointer to octet string to send.
*    DataLen - Octet length of the octet string to send.
*    Flags   - Socket send flags.
*
*  Return value
*    >= 0 - Number of bytes sent.
*    <  0 - Error.
*
*  Additional information
*    The number of bytes sent can be less than the number
*    of bytes that were requested to be written.
*/
static int _Send(int Socket, const char *pData, int DataLen, int Flags) {
  int Status;
  //
  Status = send(Socket, pData, DataLen, Flags);
  //
  return Status < 0 ? -1 : Status;
}

/**********************************************************************
*
*       _Recv()
*
*  Function description
*    Receive data from socket.
*
*  Parameters
*    Socket  - Socket to read from.
```

```
*    pData   - Pointer to object that receives an octet string.
*    DataLen - Octet length of the octet string to receive.
*    Flags   - Socket-based additional flags.
*
*  Return value
*    >= 0 - Number of bytes received.
*    <  0 - Error.
*
*  Additional information
*    The number of bytes received can be less than the number
*    of bytes requested.
*/
static int _Recv(int Socket, char *pData, int DataLen, int Flags) {
  int Status;
  //
  Status = recv(Socket, pData, DataLen, Flags);
  //
  return Status < 0 ? -1 : Status;
}

/**********************************************************************
*
*       _CanRead()
*
*  Function description
*    Check if data can be read from socket.
*
*  Parameters
*    Socket  - Socket to query.
*
*  Return value
*    >  0 - Socket has data to read.
*    <= 0 - Socket has no data.
*/
static int _CanRead(int Socket) {
  IP_fd_set readset;
  int       Status;
  //
  IP_FD_ZERO(&readset);
  IP_FD_SET(Socket, &readset);
  Status = select(&readset, 0, 0, 0);
  //
  return Status > 0;
}

/**********************************************************************
*
*       _RING_BUFFER_Init()
*
*  Function description
*    Initialize rung buffer.
*
*  Parameters
*    pRB      - Pointer to ring buffer.
*    pData    - Pointer to data area to hold ring buffer content.
*    Capacity - Capacity of the ring buffer.
*/
static void _RING_BUFFER_Init(RING_BUFFER *pRB, void *pData, unsigned Capacity) {
  pRB->RdPtr    = 0;
  pRB->WrPtr    = 0;
  pRB->RdCnt    = 0;
  pRB->WrCnt    = 0;
  pRB->Capacity = Capacity;
  pRB->pData    = pData;
}

/**********************************************************************
*
```

```
*          _RING_BUFFER_QueryCanRead()
*
*  Function description
*     Query whether ring buffer can be read.
*
*  Parameters
*    pRB - Pointer to ring buffer.
*
*  Return value
*     Number of bytes that can be read from the buffer.
*/
static unsigned _RING_BUFFER_QueryCanRead(const RING_BUFFER *pRB) {
  return pRB->WrCnt - pRB->RdCnt;
}

/*********************************************************************
*
*          _RING_BUFFER_QueryCanWrite()
*
*  Function description
*     Query whether ring buffer can be written.
*
*  Parameters
*    pRB - Pointer to ring buffer.
*
*  Return value
*     Number of bytes that can be written to the buffer.
*/
static unsigned _RING_BUFFER_QueryCanWrite(const RING_BUFFER *pRB) {
  return pRB->Capacity - (pRB->WrCnt - pRB->RdCnt);
}

/*********************************************************************
*
*          _RING_BUFFER_Wr()
*
*  Function description
*     Write to ring buffer.
*
*  Parameters
*    pRB     - Pointer to ring buffer.
*    pData   - Pointer to data to write.
*    DataLen - Octet length of the data to write.
*
*  Return value
*     Number of bytes that were be written to the buffer.
*     If the buffer becomes full during writing, the
*     number of bytes written will be less than DataLen.
*/
static unsigned _RING_BUFFER_Wr(      RING_BUFFER * pRB,
                                const void        * pData,
                                      unsigned      DataLen) {
  unsigned N;
  unsigned WrPtr;
  unsigned LChunkLen;
  unsigned RChunkLen;
  //
  // If we don't have enough space to write all of this,
  // write part until the ring buffer is full.
  //
  N = _RING_BUFFER_QueryCanWrite(pRB);
  if (N < DataLen) {
    DataLen = N;
  }
  //
  // Divide into two cases: enough space to write without wrapping,
  // and wrapping required.
  //
```

```
    WrPtr = pRB->WrPtr;
    if (WrPtr + DataLen <= pRB->Capacity) {
      memcpy(pRB->pData+WrPtr, pData, DataLen);
      WrPtr += DataLen;
      if (WrPtr == pRB->Capacity) {
        WrPtr = 0;
      }
      pRB->WrPtr = WrPtr;
    } else {
      //
      LChunkLen = pRB->Capacity - WrPtr;
      RChunkLen = DataLen - LChunkLen;
      //
      memcpy(pRB->pData+WrPtr, pData, LChunkLen);
      memcpy(pRB->pData, (U8 *)pData+LChunkLen, RChunkLen);
      //
      pRB->WrPtr = RChunkLen;
    }
    //
    pRB->WrCnt += DataLen;
    //
    return DataLen;
}

/*********************************************************************
*
*       _RING_BUFFER_Rd()
*
*  Function description
*    Read from ring buffer.
*
*  Parameters
*    pRB     - Pointer to ring buffer.
*    pData   - Pointer to object that receives the data.
*    DataLen - Octet length of the data to read.
*
*  Return value
*    Number of bytes that were be read from  the buffer.
*    If the buffer becomes empty during reading, the
*    number of bytes read will be less than DataLen.
*/
static unsigned _RING_BUFFER_Rd(RING_BUFFER * pRB,
                                void        * pData,
                                unsigned      DataLen) {
  unsigned RdPtr;
  unsigned LChunkLen;
  unsigned RChunkLen;
  //
  // If we don't have enough space to write all of this,
  // write part until the ring buffer is full.
  //
  RdPtr = _RING_BUFFER_QueryCanRead(pRB);
  if (RdPtr < DataLen) {
    DataLen = RdPtr;
  }
  //
  // Null read?
  //
  if (DataLen == 0) {
    return DataLen;
  }
  //
  // Divide into two cases: enough space to read without wrapping,
  // and wrapping required.
  //
  RdPtr = pRB->RdPtr;
  if (RdPtr + DataLen <= pRB->Capacity) {
    //
```

```c
    // Enough space to read, no wrapping required.
    //
    memcpy(pData, pRB->pData+RdPtr, DataLen);
    RdPtr += DataLen;
    if (RdPtr == pRB->Capacity) {
      RdPtr = 0;
    }
    pRB->RdPtr = RdPtr;
  } else {
    //
    // Straddles end of buffer, break into two reads.
    //
    LChunkLen = pRB->Capacity - RdPtr;
    RChunkLen = DataLen - LChunkLen;
    //
    memcpy(pData, pRB->pData+RdPtr, LChunkLen);
    memcpy((char *)pData+LChunkLen, pRB->pData, RChunkLen);
    //
    pRB->RdPtr = RChunkLen;
  }
  //
  pRB->RdCnt += DataLen;
  //
  return DataLen;
}

/*********************************************************************
*
*       _AppExit()
*
*  Function description
*    Exit the application with an error.
*
*  Parameters
*    sReason - Reason for exit, displayed for the user.
*/
static void _AppExit(const char *sReason) {
  SSH_Logf(SSH_LOG_APP, sReason);
  SSH_Panic(-100);
}

/*********************************************************************
*
*       _ShellRequest()
*
*  Function description
*    Handle a shell channel request.
*
*  Parameters
*    pSession - Pointer to session.
*    Channel  - Local channel receiving the data.
*    pParas   - Pointer to channel request parameters.
*
*  Return value
*    >= 0 - Success.
*    <  0 - Error.
*/
static int _ShellRequest(SSH_SESSION               * pSession,
                         unsigned                    Channel,
                         SSH_CHANNEL_REQUEST_PARAS * pParas) {
  SHELL_CONTEXT * pShell;
  int             Status;
  //
  Status = 0;
  if (pParas->WantReply) {
    Status = SSH_CHANNEL_SendSuccess(pSession, Channel);
  }
  //
```

```c
  pShell = &_aShell[SSH_SESSION_QueryIndex(pSession)];
  SSH_CHANNEL_Config(pSession, Channel, 128, &_TerminalAPI, pShell);
  OS_CreateTaskEx(&pShell->ShellTask,
                  "ShellTask",
                  100,
                  _ShellMain,
                  &pShell->aShellTaskStack,
                  sizeof(pShell->aShellTaskStack),
                  10,
                  pSession);
  //
  return Status;
}

/*********************************************************************
*
*       _TerminalChannelData()
*
*  Function description
*    Handle data received from peer.
*
*  Parameters
*    pSession - Pointer to session.
*    Channel  - Local channel receiving the data.
*    pData    - Pointer to object that contains the data.
*    DataLen  - Octet length of the object that contains the data.
*
*  Return value
*    >= 0 - Success.
*    <  0 - Error.
*
*  Additional information
*    Provide in-callback handling of a command line processor.
*    This sample supports only one connection at a time.
*/
static int _TerminalChannelData(      SSH_SESSION * pSession,
                                      unsigned      Channel,
                                const U8          * pData,
                                      unsigned      DataLen) {
  SHELL_CONTEXT *pShell;
  //
  SSH_USE_PARA(pSession);
  SSH_USE_PARA(Channel);
  //
  pShell = &_aShell[SSH_SESSION_QueryIndex(pSession)];
  _RING_BUFFER_Wr(&pShell->RxBuffer, pData, DataLen);
  OS_WakeTask(&pShell->ShellTask);
  //
  return 0;
}

/*********************************************************************
*
*       _TerminalChannelClose()
*
*  Function description
*    Handle channel closure.
*
*  Parameters
*    pSession - Pointer to session.
*    Channel  - Local channel receiving the data.
*
*  Additional information
*    When the channel carrying the shell is closed, terminate
*    the associated shell task.
*/
static void _TerminalChannelClose(SSH_SESSION * pSession, unsigned Channel) {
  SHELL_CONTEXT *pShell;
```

```
  //
  SSH_USE_PARA(Channel);
  //
  pShell = &_aShell[SSH_SESSION_QueryIndex(pSession)];
  OS_TerminateTask(&pShell->ShellTask);
}

/**********************************************************************
*
*       _TerminalRequest()
*
*  Function description
*    Request a terminal.
*
*  Parameters
*    pSession - Pointer to session.
*    Channel  - Local channel requesting the terminal.
*    pParas   - Pointer to channel request parameters.
*
*  Return value
*    >= 0 - Success.
*    <  0 - Error.
*/
static int _TerminalRequest(SSH_SESSION                * pSession,
                            unsigned                     Channel,
                            SSH_CHANNEL_REQUEST_PARAS * pParas) {
  int Status;
  //
  if (pParas->WantReply) {
    Status = SSH_CHANNEL_SendSuccess(pSession, Channel);
  } else {
    Status = 0;
  }
  //
  return Status;
}

/**********************************************************************
*
*       _UserauthServiceRequest()
*
*  Function description
*    Request the user authentication service.
*
*  Parameters
*    pSession     - Pointer to session.
*    sServiceName - Service being requested.
*
*  Return value
*    >= 0 - Success.
*    <  0 - Error.
*
*  Additional information
*    Displays a banner before user authentication commences.
*/
static int _UserauthServiceRequest(SSH_SESSION *pSession, const char *sServiceName) {
  int Status;
  //
  Status = SSH_SESSION_SendServiceAccept(pSession, sServiceName);
  if (Status >= 0) {
    Status = SSH_SESSION_SendUserauthBanner(pSession, BANNER, "en");
  }
  //
  return Status;
}

/**********************************************************************
*
```

```
*        _UserauthRequestNone()
*
*  Function description
*    Request authentication of user with method "none".
*
*  Parameters
*    pSession  - Pointer to session.
*    pReqParas - Pointer to user authentication request parameters.
*
*  Return value
*    >= 0 - Success.
*    <  0 - Error.
*/
static int _UserauthRequestNone(SSH_SESSION              * pSession,
                                SSH_USERAUTH_REQUEST_PARAS * pReqParas) {
  SSH_USERAUTH_NONE_PARAS NoneParas;
  int                     Status;
  //
  SSH_USE_PARA(pSession);
  //
  Status = SSH_USERAUTH_NONE_ParseParas(pReqParas, &NoneParas);
  if (Status < 0) {
    Status = SSH_ERROR_USERAUTH_FAIL;
  } else if (pReqParas->UserNameLen == 4 &&
             SSH_MEMCMP(pReqParas->pUserName, "anon", 4) == 0) {
    Status = 0;
  } else {
    Status = SSH_ERROR_USERAUTH_FAIL;
  }
  //
  return Status;
}

/*********************************************************************
*
*        _UserauthRequestPassword()
*
*  Function description
*    Request authentication of user with method "password".
*
*  Parameters
*    pSession  - Pointer to session.
*    pReqParas - Pointer to user authentication request parameters.
*
*  Return value
*    >= 0 - Success.
*    <  0 - Error.
*/
static int _UserauthRequestPassword(SSH_SESSION              * pSession,
                                    SSH_USERAUTH_REQUEST_PARAS * pReqParas) {
  SSH_USERAUTH_PASSWORD_PARAS PasswordParas;
  int                         Status;
  //
  SSH_USE_PARA(pSession);
  //
  Status = SSH_USERAUTH_PASSWORD_ParseParas(pReqParas, &PasswordParas);
  if (Status < 0) {
    Status = SSH_ERROR_USERAUTH_FAIL;
  } else if (pReqParas->UserNameLen == 5 &&
             memcmp(pReqParas->pUserName, "admin", 5) == 0) {
    if (PasswordParas.PasswordLen == 6 &&
        memcmp(PasswordParas.pPassword, "secret", 6) == 0) {
      Status = 0;
    } else {
      Status = SSH_ERROR_USERAUTH_FAIL;
    }
  } else {
    Status = SSH_ERROR_USERAUTH_FAIL;
```

```c
  }
  //
  return Status;
}

/**********************************************************************
*
*       _Puts()
*
*  Function description
*    Send string to peer.
*
*  Parameters
*    pShell - Pointer to shell context.
*    sText  - String to send.
*/
static void _Puts(SHELL_CONTEXT *pShell, const char *sText) {
  unsigned Len;
  unsigned ChunkLen;
  //
  Len = strlen(sText);
  //
  while (Len > 0) {
    ChunkLen = _RING_BUFFER_Wr(&pShell->TxBuffer, sText, Len);
    sText += ChunkLen;
    Len   -= ChunkLen;
    if (Len != 0) {
      OS_Delay(10000);
    }
  }
}

/**********************************************************************
*
*       _Putc()
*
*  Function description
*    Send character peer.
*
*  Parameters
*    pShell - Pointer to shell context.
*    Ch     - Character to send.
*/
static void _Putc(SHELL_CONTEXT *pShell, U8 Ch) {
  char aBuf[2];
  //
  aBuf[0] = (char)Ch;
  aBuf[1] = 0;
  //
  _Puts(pShell, aBuf);
}

/**********************************************************************
*
*       _Getc()
*
*  Parameters
*    pShell - Pointer to shell context.
*
*  Function description
*    Read character from peer.
*
*  Return value
*    Character read.
*/
static U8 _Getc(SHELL_CONTEXT *pShell) {
  U8 Ch;
  //
```

```c
    while (_RING_BUFFER_QueryCanRead(&pShell->RxBuffer) == 0) {
      OS_Delay(10000);
    }
    _RING_BUFFER_Rd(&pShell->RxBuffer, &Ch, 1);
    return Ch;
}

/*********************************************************************
*
*       _Gets()
*
*  Function description
*    Read string from peer.
*
*  Parameters
*    pShell - Pointer to shell context.
*
*  Return value
*    Character read.
*/
static int _Gets(SHELL_CONTEXT *pShell) {
  U8 Ch;
  //
  pShell->Cursor = 0;
  //
  for (;;) {
    Ch = _Getc(pShell);
    if (0x20 <= Ch && Ch <= 0x7E) {
      if (pShell->Cursor < sizeof(pShell->aCommandLine)-1) {
        pShell->aCommandLine[pShell->Cursor++] = Ch;
        _Putc(pShell, Ch);
      }
    } else if (Ch == 0x08 || Ch == 0x7F) {
      if (pShell->Cursor > 0) {
        --pShell->Cursor;
        _Puts(pShell, "\b \b");
      }
    } else if (Ch == '\r') {
      pShell->aCommandLine[pShell->Cursor++] = 0;
      return 0;
    } else if (Ch == 0x04) {
      return -1;
    }
  }
}

/*********************************************************************
*
*       _ShellMain()
*
*  Function description
*    Main command line processor acting as a shell.
*
*  Parameters
*    pContext - Pointer to SSH session.
*/
static void _ShellMain(void *pContext) {
  SSH_SESSION   * pSession;
  SHELL_CONTEXT * pShell;
  //
  pSession = pContext;
  pShell = &_aShell[SSH_SESSION_QueryIndex(pSession)];
  //
  _Puts(pShell, SIGNON);
  for (;;) {
    _Puts(pShell, PROMPT);
    if (_Gets(pShell) == 0) {
      _Puts(pShell, "\r\n...");
```

```c
        _Puts(pShell, pShell->aCommandLine);
        _Puts(pShell, "\r\n");
      } else {
        pShell->Exit = 1;
      }
    }
  }
}

/*********************************************************************
*
*       _ServiceChannel()
*
*  Function description
*    Service a shell channel.
*
*  Parameters
*    pSession - Pointer to SSH session.
*    Channel  - Local channel.
*
*  Return value
*    >  0 - Processed channel.
*    == 0 - Channel required no processing.
*    <  0 - Error.
*/
static int _ServiceChannel(SSH_SESSION *pSession, unsigned Channel) {
  SHELL_CONTEXT * pShell;
  unsigned        Len;
  U8              aData[64];
  int             Status;
  //
  pShell = &_aShell[SSH_SESSION_QueryIndex(pSession)];
  if (pShell == 0) {
    Status = 0;
  } else if (pShell->Exit) {
    if (SSH_CHANNEL_QueryCanWrite(pSession, Channel) >= 12) {
      Status = SSH_CHANNEL_SendData(pShell->pSession, Channel, "\r\n\r\nBye!\r\n
\r\n", 12);
    }
    SSH_SESSION_Disconnect(pShell->pSession, SSH_DISCONNECT_BY_APPLICATION);
    Status = -1;
  } else if (_RING_BUFFER_QueryCanRead(&pShell->TxBuffer) > 0) {
    Len = _RING_BUFFER_QueryCanRead(&pShell->TxBuffer);
    Len = SEGGER_MIN(Len, SSH_CHANNEL_QueryCanWrite(pShell->pSession, Channel));
    Len = SEGGER_MIN(Len, sizeof(aData));
    if (Len > 0) {
      _RING_BUFFER_Rd(&pShell->TxBuffer, aData, Len);
      Status = SSH_CHANNEL_SendData(pShell->pSession, Channel, aData, Len);
      OS_WakeTask(&pShell->ShellTask);
    }
    Status = 1;
  } else {
    Status = 0;
  }
  //
  return Status;
}

/*********************************************************************
*
*       _ProtocolMain()
*
*  Function description
*    Handle SSH protocol.
*
*  Parameters
*    pContext - Pointer to SSH session.
*
*  Additional information
```

```
 *     Acts rather like sshd.
 */
static void _ProtocolMain(void *pContext) {
  SSH_SESSION  * pSession;
  int            Socket;
  int            Status;
  //
  pSession = pContext;
  Socket = SSH_SESSION_QuerySocket(pSession);
  for (;;) {
    Status = 0;
    if (_CanRead(Socket)) {
      Status = SSH_SESSION_Process(pSession);
    } else {
      Status = SSH_SESSION_IterateChannels(pSession, _ServiceChannel);
    }
    if (Status < 0) {
      OS_TerminateTask(0);
    } else if (Status == 0) {
      OS_Delay(10);
    }
  }
}

/*********************************************************************
 *
 *       _SSHTask()
 *
 *  Function description
 *    Listen for incoming connections and start SSH handler tasks.
 *
 *  Additional information
 *    Acts rather like sshd.
 */
static void _SSHTask(void) {
  int            BoundSocket;
  int            Socket;
  unsigned       SessionIndex;
  SSH_SESSION  * pSession;
  SHELL_CONTEXT * pShell;
  //
  // Hook user authentication to display a banner.  Allow "none"
  // and "password" user authentication.
  //
  SSH_SERVICE_Add(&SSH_SERVICE_USERAUTH, _UserauthServiceRequest);
  SSH_USERAUTH_METHOD_Add(&SSH_USERAUTH_METHOD_NONE,     _UserauthRequestNone);
  SSH_USERAUTH_METHOD_Add(&SSH_USERAUTH_METHOD_PASSWORD, _UserauthRequestPassword);
  //
  // Add support for interactive shells.
  //
  SSH_CHANNEL_REQUEST_Add(&SSH_CHANNEL_REQUEST_SHELL,         _ShellRequest);
  SSH_CHANNEL_REQUEST_Add(&SSH_CHANNEL_REQUEST_ENV,          NULL);
  SSH_CHANNEL_REQUEST_Add(&SSH_CHANNEL_REQUEST_PTYREQ,       _TerminalRequest);
  SSH_CHANNEL_REQUEST_Add(&SSH_CHANNEL_REQUEST_WINDOW_CHANGE, NULL);
  //
  // Bind SSH port.
  //
  BoundSocket = _Bind(22);
  if (BoundSocket < 0) {
    _AppExit("Cannot bind port 22!");
  }
  //
  for (;;) {
    //
    // Try to allocate a session for the next incoming connection.
    //
    SSH_SESSION_Alloc(&pSession);
    if (pSession == NULL) {
```

```c
      OS_Delay(100);
      continue;
    }
    //
    do {
      Socket = _Accept(BoundSocket);
    } while (Socket < 0);
    //
    SessionIndex = SSH_SESSION_QueryIndex(pSession);
    pShell = &_aShell[SessionIndex];
    memset(pShell, 0, sizeof(*pShell));
    pShell->pSession = pSession;
    SSH_SESSION_Init(pShell->pSession, Socket, &_IP_Transport);
    SSH_SESSION_ConfBuffers(pShell->pSession,
                            pShell->aRxBuffer, sizeof(pShell->aRxBuffer),
                            pShell->aTxBuffer, sizeof(pShell->aTxBuffer));
    _RING_BUFFER_Init(&pShell->RxBuffer, pShell->aRxData, sizeof(pShell-
>aRxData));
    _RING_BUFFER_Init(&pShell->TxBuffer, pShell->aTxData, sizeof(pShell-
>aTxData));
    //
    OS_CreateTaskEx(&pShell->ProtocolTask,
                    "ProtocolTask",
                    100,
                    _ProtocolMain,
                    &pShell->aProtocolTaskStack,
                    sizeof(pShell->aProtocolTaskStack),
                    10,
                    pSession);
  }
}

/**********************************************************************
*
*             Public code
*
***********************************************************************
*/

/**********************************************************************
*
*        MainTask()
*
*  Function description
*    Application entry point.
*/
void MainTask(void) {
  //
  int IFaceId;
  IP_Init();
  IFaceId = IP_INFO_GetNumInterfaces() - 1;
  // Get the last registered interface ID as this is most likely the interface we want to use in
  SSH_Init();
  //
  OS_CREATETASK(&_IPTCB,   "IP_Task",   IP_Task,    TASK_PRIO_IP_TASK,
  _IPStack);    // Start the IP task
#if USE_RX_TASK
  OS_CREATETASK(&_IPRxTCB, "IP_RxTask", IP_RxTask , TASK_PRIO_IP_RX_TASK, _IPRxStack);
  // Start the IP_RxTask, optional.
#endif
  IP_Connect(IFaceId);
  while (IP_IFaceIsReady() == 0) {
    OS_Delay(50);
  }
  //
  // SSH connections may require more stack than MainTask()
  // is given by the canned startup code.  Therefore we start the
  // SSH-based task with more stack so that it runs correctly and
```

```
  // kill off MainTask.
  //
  OS_CREATETASK(&_ServerTCB, "SSHTask", _SSHTask, TASK_PRIO_IP_APP, _ServerStack);
  OS_TerminateTask(0);
}

/*************************** End of file ***************************/
```

3.8   Adding emSSH to your project

In this section we assume that you have a fully-functioning embOS/IP project that is able to connect to the network and all that is required is to add emSSH to the project. You can use the sample "start" projects as a reference when setting up your own application.

❶ **Set up include directories**

You should make sure that the include path contains the following directories (the order of inclusion is of no importance):

- `Config`
- `CRYPTO`
- `SEGGER`
- `SSH`

> **Note**
>
> Always make sure that you have only one version of each file!

It is frequently a major problem when updating to a new version of emSSH if you have old files included and therefore mix different versions. If you keep emSSH in the directories as suggested (and only in these), this type of problem cannot occur. When updating to a newer version, you should be able to keep your configuration files and leave them unchanged. For safety reasons, we recommend backing up (or at least renaming) the existing directories before updating.

❷ **Add source files**

Add the source code files that you find in the shipment to your project. The `SEGGER` and `CRYPTO` folders are shared components.

❸ **Initialize emSSH**

You initialize emSSH using `SSH_Init()`. You must call `SSH_Init()` before using any other emSSH API function.

With these three steps, emSSH is installed and ready to run. You will need to add some additional code to handle incoming connection requests and configure the services and facilities that emSSH provides as described in previous sections.

Chapter 4

Server identity

A key component of the SSH protocol is that the server is authenticated using public key cryptography: the server presents its public key that corresponds to the public key algorithm negotiated between the client and server.

The following sections describe how this key is generated and installed, and the performance of the key exchange and public key algorithms at the server end of the connection.

4.1   Public key types

emSSH supports four types of public keys:

- RSA
- DSA
- ECDSA
- EdDSA

Each of these is fully described in the SSH RFCs, but you do not need to be familiar with the SSH protocol or RFCs in order to add emSSH to your application.

Public keys for emSSH are created using `SSH_KeyGen` which is capable of generating all key types supported by SSH.

Each of the public key algorithms requires a different type of public and private key pair. This section describes how to generate and install those key pairs for emSSH.

4.2   RSA keys

The public key method `SSH_PK_ALGORITHM_SSH_RSA` requires an RSA public key pair to be installed. For reference, this method corresponds to the `ssh-rsa` public key method in the SSH specifications.

Creating keys

You can generate an RSA public key pair using `SSH_KeyGen`:

```
MacBook:~ paul$ ssh_keygen -rsa -k SSH_ServerKeys_RSA_ -x >SSH_ServerKeys_RSA.c

emSSH KeyGen V2.40 compiled Jun 20 2017 12:12:14
(c) 2014-2017 SEGGER Microcontroller GmbH    www.segger.com

Selecting a random initial seed
Generating probabilistic prime key pair with public modulus of 1024 bits
Checking keys are consistent: OK

MacBook:~ paul$ _
```

`-rsa`
    Commands the key generator to create RSA keys.

`-l 1024`
    Asks that the modulus, or key length, be set to 1,024 bits. The default is 2,048 bits, and we reduce it here by way of example, to keep the presented listing of the key pair within reason.

`-nf`
    Selects non-FIPS generation of the key pair. For key lengths less than 1,024 bits, this is required.

`-k SSH_ServerKeys_RSA_`
    Sets the prefix used for the public key declarations.

`-x`

    Requests that the key structures have external linkage so they can be separately compiled; the alternative, without `-x`, is to generate them with static storage and, in this case, the keys would need to be manually added to an existing source file that uses them.

`>ServerKeys_RSA.c`
    Writes the generated key declarations to the file `ServerKeys_RSA.c`.

The unabridged output of the key generated above is presented in *Generated RSA keys* on page 113.

To generate smaller keys you must drop into non-FIPS mode. We recommend that you use FIPS generation of proven primes and a minimum key length of 2,048 bits, which is the default RSA key generation mode of `SSH_KeyGen`:

```
MacBook:~ paul$ ssh_keygen -rsa -k ServerKeys_RSA -x >ServerKeys_RSA.c

emSSH KeyGen V2.40 compiled Jun 20 2017 12:12:14
(c) 2014-2017 SEGGER Microcontroller GmbH    www.segger.com

Selecting a random initial seed
Generating proven prime key pair with public modulus of 2048 bits
Public encryption exponent is set to 65537
Initial seed is 0x3ACF10CB2B7BDBBEFCCEACB9F58176F4
Checking keys are consistent: OK

MacBook:~ paul$ _
```

The time to create the keys depends upon the length of the modulus, the speed of the PC used to generate them, and randomness: keys can take a few seconds to tens of seconds to

create because of the algorithms involved and how lucky the algorithms are on stumbling upon, or constructing, a prime.

Installing keys

From the listing presented in *Generated RSA keys* on page 113, there are two relevant constant structures that define the public key pair. In this case, the structures are named `ServerKeys_RSA_PrivateKey` and `ServerKeys_RSA_PublicKey`.

To install these into emSSH we must create two functions that deliver them when requested:

```c
static const CRYPTO_RSA_PUBLIC_KEY * _SSH_GetRSAPublicKey(const char *sName) {
  return &ServerKeys_RSA_PublicKey;
}

static const CRYPTO_RSA_PRIVATE_KEY * _SSH_GetRSAPrivateKey(const char *sName) {
  return &ServerKeys_RSA_PrivateKey;
}
```

In the same file we must declare the keys external…

```c
extern const CRYPTO_RSA_PUBLIC_KEY  ServerKeys_RSA_PublicKey;
extern const CRYPTO_RSA_PRIVATE_KEY ServerKeys_RSA_PrivateKey;
```

…and add the two functions to the list of callbacks that emSSH can invoke to retrieve the keys:

```c
static const SSH_HOSTKEY_API _SSH_HostKeyAPI = {
  _SSH_GetRSAPublicKey, _SSH_GetRSAPrivateKey,
  0,                      0,
  0,                      0,
  0,                      0,
};
```

The zeros in the API are placeholders for other types of key: we initialize them to zero to silence compilers that warn when an initializer does not have entries for every member.

Once this framework is set up, it's installed into emSSH using `SSH_SetHostKeyAPI` which you should add to your `SSH_X_Config()` function (see *Runtime configuration* on page 215):

```c
SSH_SetHostKeyAPI(&_SSH_HostKeyAPI);
```

That's it! The RSA keys are set up and installed, ready for use.

4.3   DSA keys

The public key method `SSH_PK_ALGORITHM_SSH_DSA` requires a DSA public key pair to be installed. For reference, this method corresponds to the `ssh-dss` public key method in the SSH specifications.

Creating keys

Generating a DSA key pair is similar to RSA with `SSH_KeyGen`:

```
MacBook:~ paul$ ssh_keygen -dsa -k SSH_ServerKeys_DSA_ -x >SSH_ServerKeys_DSA.c

emSSH KeyGen V2.40 compiled Jun 20 2017 12:12:14
(c) 2014-2017 SEGGER Microcontroller GmbH     www.segger.com

Generating DSA domain (L=2048, N=256) with proven primes

MacBook:~ paul$ _
```

Only the key type differs here, using `-dsa` to generate the keys.

In contrast to RSA which uses the product of two primes of equal length as a modulus, DSA uses a different scheme with two primes (l and n) with differing lengths.

We recommend that you use FIPS generation of proven primes and a minimum l key length of 2,048 bits, which is the default DSA key generation mode of `SSH_Keygen`.

The prime lengths to use for l and n when creating keys can be set on `SSH_KeyGen`'s command line. If generating keys with prime lengths that do not conform to any FIPS combination, you must use non-FIPS algorithms, activated by `-nf`.

The unabridged output of a key generated with l=1024 and n=160 is presented in *Generated DSA keys* on page 131.

Note that the SSH protocol only supports DSA keys with with l=1024 and n=160.

Installing keys

Installing DSA keys is slightly different to installing RSA keys: there are three constant structures that define the public key pair. In this case, the structures are named `ServerKeys_DSA_PrivateKey`, `ServerKeys_DSA_PublicKey`, and `ServerKeys_DSA_DomainParas`.

To install these into emSSH we must create two functions that deliver them when requested:

```c
static void _SSH_GetDSAPublicKey(const char                        * sName,
                                 const CRYPTO_DSA_PUBLIC_KEY    ** ppPublicKey,

  const CRYPTO_DSA_DOMAIN_PARAMS ** ppDomainParas) {
  (void)sName;
  //
  *ppPublicKey   = &SSH_ServerKeys_DSA_PublicKey;
  *ppDomainParas = &SSH_ServerKeys_DSA_DomainParas;
}

static void _SSH_GetDSAPrivateKey(const char                        * sName,
                                  const CRYPTO_DSA_PRIVATE_KEY   ** ppPrivateKey,

  const CRYPTO_DSA_DOMAIN_PARAMS ** ppDomainParas) {
  (void)sName;
  //
  *ppPrivateKey   = &SSH_ServerKeys_DSA_PrivateKey;
  *ppDomainParas = &SSH_ServerKeys_DSA_DomainParas;
}
```

In the same file we must declare the keys and domain external…

```
extern const CRYPTO_DSA_PUBLIC_KEY    SSH_ServerKeys_DSA_PublicKey;
extern const CRYPTO_DSA_PRIVATE_KEY   SSH_ServerKeys_DSA_PrivateKey;
extern const CRYPTO_DSA_DOMAIN_PARAMS SSH_ServerKeys_DSA_DomainParas;
```

…and add the two functions to the list of callbacks that emSSH can invoke to retrieve the keys:

```
static const SSH_HOSTKEY_API _SSH_HostKeyAPI = {
  0,                        0,
  0,                        0,
  0,                        0,
  _SSH_GetDSAPublicKey, _SSH_GetDSAPrivateKey,
};
```

The zeros in the API are placeholders for other types of key.

4.4   ECDSA keys

The public key methods

- `SSH_PK_ALGORITHM_ECDSA_SHA2_NISTP256`
- `SSH_PK_ALGORITHM_ECDSA_SHA2_NISTP384`
- `SSH_PK_ALGORITHM_ECDSA_SHA2_NISTP521`

each require an ECDSA public key pair to be installed, generated from the appropriate curve. For reference, these methods correspond to the `ecdsa-sha2-nistp256`, `ecdsa-sha2-nistp384`, and `ecdsa-sha2-nistp521` public key methods in the SSH specifications.

Creating keys

Generating a ECDSA key pair requires that the curve be specified:

```
MacBook:~ paul$ ssh_keygen -ecdsa -p256 -k SSH_ServerKeys_ECDSA_P256_ -x \
> >SSH_ServerKeys_ECDSA_P256.c

emSSH KeyGen V2.40 compiled Jun 20 2017 12:12:14
(c) 2014-2017 SEGGER Microcontroller GmbH    www.segger.com

Generating random key pair on P-256
Checking generated keys for consistency: OK

MacBook:~ paul$ _
```

Installing keys

To install these into emSSH we must create two functions that deliver them when requested:

```
static const CRYPTO_ECDSA_PUBLIC_KEY * _SSH_GetECDSAPublicKey(const char *sName) {
  if (SSH_STRCMP(sName, "nistp256") == 0) {
    return &SSH_ServerKeys_ECDSA_P256_PublicKey;
  } else {
    return 0;
  }
}

static const CRYPTO_ECDSA_PRIVATE_KEY * _SSH_GetECDSAPrivateKey(const char *sName) {
  if (SSH_STRCMP(sName, "nistp256") == 0) {
    return &SSH_ServerKeys_ECDSA_P256_PrivateKey;
  } else {
    return 0;
  }
}
```

In the same file we must declare the keys external…

```
extern const CRYPTO_ECDSA_PUBLIC_KEY  SSH_ServerKeys_ECDSA_P256_PublicKey;
extern const CRYPTO_ECDSA_PRIVATE_KEY SSH_ServerKeys_ECDSA_P256_PrivateKey;
```

…and add the two functions to the list of callbacks that emSSH can invoke to retrieve the keys:

```
static const SSH_HOSTKEY_API _SSH_HostKeyAPI = {
  0,                        0,
  _SSH_GetECDSAPublicKey, _SSH_GetECDSAPrivateKey,
  0,                        0,
  0,                        0,
};
```

This example installs keys for the P-256 curve, but you can install keys for the curves you wish to support. Refer to *Shipped sample configuration* on page 222 for further information.

4.5   EdDSA keys

The public key methods `SSH_PK_ALGORITHM_SSH_ED25519` and `SSH_PK_ALGORITHM_SSH_ED448` requires an EdDSA public key pair to be installed. For reference, these methods correspond to the `ssh-ed25519` and `ssh=ed448` public key methods in the SSH specifications.

Creating keys

Generating an EdDSA key pair requires no additional information as the key length is fixed:

```
MacBook:~ paul$ ssh_keygen -ed25519 -k SSH_ServerKeys_EdDSA_ -x \
> >SSH_ServerKeys_Ed25519.c

emSSH KeyGen V2.40 compiled Jun 20 2017 12:12:14
(c) 2014-2017 SEGGER Microcontroller GmbH     www.segger.com

Generating random EdDSA key pair

MacBook:~ paul$ _
```

Installing keys

To install these into emSSH we must create two functions that deliver them when requested:

```
static const CRYPTO_EdDSA_PUBLIC_KEY * _SSH_GetEdDSAPublicKey(const char *sName) {
  if (SSH_STRCMP(sName, "ssh-ed25519") == 0) {
    return &SSH_ServerKeys_Ed25519_PublicKey;
  } else {
    return NULL;
  }
}

static const CRYPTO_EdDSA_PRIVATE_KEY
  * _SSH_GetEdDSAPrivateKey(const char *sName) {
  if (SSH_STRCMP(sName, "ssh-ed25519") == 0) {
    return &SSH_ServerKeys_Ed25519_PrivateKey;
  } else {
    return NULL;
  }
}
```

In the same file we must declare the keys external…

```
extern const CRYPTO_EdDSA_PUBLIC_KEY  SSH_ServerKeys_Ed25519_PublicKey;
extern const CRYPTO_EdDSA_PRIVATE_KEY SSH_ServerKeys_Ed25519_PrivateKey;
```

…and add the two functions to the list of callbacks that emSSH can invoke to retrieve the keys:

```
static const SSH_HOSTKEY_API _SSH_HostKeyAPI = {
  NULL,                     NULL,
  NULL,                     NULL,
  _SSH_GetEdDSAPublicKey, _SSH_GetEdDSAPrivateKey,
  NULL,                     NULL
};
```

4.6   SSH key generator reference

emSSH KeyGen generates a public and a private key. The generation parameters can be set via command line options, by default a random 2048 bit key is generated. The keys are saved in a common key file format and can be published and exchanged.

Usage

`SSH_KeyGen.exe [<Options>]`

4.6.1   General options

emSSH KeyGen accepts the following command line options.

Option	Description
`-h`	Print usage information and available command line options.
`-q`	Operate silently and do not print log output.
`-v`	Increase verbosity of log output.
`-k` *string*	Set the object name prefix to *string*. Default is empty.
`-rsa`	Generate RSA keys.
`-dsa`	Generate DSA keys.
`-ecdsa`	Generate ECDSA keys.
`-ed25519`	Generate Ed25519 keys.
`-ed448`	Generate Ed448 keys.

4.6.2   RSA options

emSSH KeyGen accepts the following command line options for RSA key generation.

Option	Description
`-l` *n*	Set the modulus length to *n* bits. Default is *2048*. For modulus lengths that other than 2048 and 3072, *-nf* is required.
`-e` *x*	Set the exponent to *x* bits. Default is *65537*.
`-seed` *n*	Set the initial seed for random number generation to *n*. Default is random.
`-pw` *string*	Generate the initial seed from the pass phrase *string*.
`-f`	Generate proven primes for the key pair according to FIPS algorithms. This option is default and recommended to be used with a key length of 2048 bits.
`-nf`	Generate probabilistic primes for the key pair.

4.6.3   DSA options

emSSH KeyGen accepts the following command line options for DSA key generation.

Option	Description
`-l` *n*	Set L to *n* bits for the prime P. Default is *2048*.
`-n` *n*	Set N to *n* bits for the prime Q. Default is *256*.
`-f`	Generate proven primes for the key pair according to FIPS algorithms. This option is default and recommended to be used with a key length of 2048 bits.
`-nf`	Generate probabilistic primes for the key pair.

4.6.4   ECDSA options

emSSH KeyGen accepts the following command line options for ECDSA key generation.

Option	Description
-p256	Use P-256 as the base curve.
-p384	Use P-384 as the base curve.
-p521	Use P-521 as the base curve.

4.7   Example generated keys

The keys in the emSSH distribution and produced below were generated from the following Windows batch file:

```
@ECHO OFF

REM /**********************************************************************
REM *                     (c) SEGGER Microcontroller GmbH                 *
REM *                       The Embedded Experts                          *
REM *                         www.segger.com                              *
REM **********************************************************************/
REM
REM File    : MakeKeys.bat
REM Purpose : Generate all sample SSH keys.
REM

SSH_KeyGen -rsa -nf -l 1024          -x -k SSH_ServerKeys_RSA_Temp_1024b_   >SSH_ServerKeys_RSA.c
SSH_KeyGen -rsa -nf -l 2048          -x -k SSH_ServerKeys_RSA_Temp_2048b_  >>SSH_ServerKeys_RSA.c
SSH_KeyGen -rsa -nf -l 2048          -x -k SSH_ServerKeys_RSA_Host_2048b_  >>SSH_ServerKeys_RSA.c

SSH_KeyGen -dsa -nf -l 1024 -n 160 -x -k SSH_ServerKeys_DSA_1024b_160b_   >SSH_ServerKeys_DSA.c
SSH_KeyGen -dsa -nf -l 2048 -n 160 -x -k SSH_ServerKeys_DSA_2048b_160b_  >>SSH_ServerKeys_DSA.c
SSH_KeyGen -dsa -nf -l 2048 -n 256 -x -k SSH_ServerKeys_DSA_2048b_256b_  >>SSH_ServerKeys_DSA.c
SSH_KeyGen -dsa -nf -l 3072 -n 256 -x -k SSH_ServerKeys_DSA_3072b_256b_  >>SSH_ServerKeys_DSA.c

SSH_KeyGen -ecdsa -p256              -x -k SSH_ServerKeys_ECDSA_P256_       >SSH_ServerKeys_ECDSA.c
SSH_KeyGen -ecdsa -p384              -x -k SSH_ServerKeys_ECDSA_P384_      >>SSH_ServerKeys_ECDSA.c
SSH_KeyGen -ecdsa -p521              -x -k SSH_ServerKeys_ECDSA_P521_      >>SSH_ServerKeys_ECDSA.c

SSH_KeyGen -ed25519                  -x -k SSH_ServerKeys_Ed25519_          >SSH_ServerKeys_EdDSA.c
SSH_KeyGen -ed448                    -x -k SSH_ServerKeys_Ed448_          >>SSH_ServerKeys_EdDSA.c

ECHO Keys generated OK!

REM /************************* End of file **************************/
```

4.7.1   Generated RSA keys

```c
#include "CRYPTO.h"

const CRYPTO_MPI_LIMB SSH_ServerKeys_RSA_Temp_1024b_PublicKey_N_aLimbs[] = {
  CRYPTO_MPI_LIMB_DATA4(0xBF, 0xBA, 0xFC, 0xB1),
  CRYPTO_MPI_LIMB_DATA4(0x82, 0x6E, 0x76, 0xD1),
  CRYPTO_MPI_LIMB_DATA4(0xC7, 0x85, 0xD7, 0x6D),
  CRYPTO_MPI_LIMB_DATA4(0x62, 0x2F, 0xA6, 0xFE),
  CRYPTO_MPI_LIMB_DATA4(0x28, 0xA9, 0x22, 0xF7),
  CRYPTO_MPI_LIMB_DATA4(0xF4, 0x41, 0x07, 0xBD),
  CRYPTO_MPI_LIMB_DATA4(0x2D, 0x43, 0x2D, 0xC4),
  CRYPTO_MPI_LIMB_DATA4(0x67, 0x9F, 0x00, 0x37),
  CRYPTO_MPI_LIMB_DATA4(0x73, 0xDC, 0x1E, 0xAF),
  CRYPTO_MPI_LIMB_DATA4(0x15, 0x24, 0x3A, 0x3C),
  CRYPTO_MPI_LIMB_DATA4(0xE3, 0x6D, 0x89, 0x82),
  CRYPTO_MPI_LIMB_DATA4(0xDC, 0x79, 0xFB, 0x60),
  CRYPTO_MPI_LIMB_DATA4(0x6E, 0x4B, 0x6D, 0x4A),
  CRYPTO_MPI_LIMB_DATA4(0xAD, 0x95, 0x24, 0xCD),
  CRYPTO_MPI_LIMB_DATA4(0x1B, 0x19, 0x08, 0x92),
  CRYPTO_MPI_LIMB_DATA4(0x75, 0x7F, 0x6D, 0x14),
  CRYPTO_MPI_LIMB_DATA4(0xF3, 0x45, 0x69, 0xD8),
  CRYPTO_MPI_LIMB_DATA4(0xA2, 0x4B, 0x91, 0x15),
  CRYPTO_MPI_LIMB_DATA4(0x00, 0x40, 0x77, 0xB0),
  CRYPTO_MPI_LIMB_DATA4(0xBD, 0xE4, 0x9A, 0x83),
  CRYPTO_MPI_LIMB_DATA4(0x7A, 0x8A, 0xD8, 0x16),
  CRYPTO_MPI_LIMB_DATA4(0x7D, 0x34, 0x64, 0x3A),
  CRYPTO_MPI_LIMB_DATA4(0xE3, 0x0C, 0x28, 0x56),
  CRYPTO_MPI_LIMB_DATA4(0x40, 0xD3, 0x05, 0x2F),
  CRYPTO_MPI_LIMB_DATA4(0xCB, 0x3C, 0xC3, 0xCE),
  CRYPTO_MPI_LIMB_DATA4(0x57, 0x98, 0x9A, 0xED),
  CRYPTO_MPI_LIMB_DATA4(0xD4, 0xF8, 0x56, 0x12),
  CRYPTO_MPI_LIMB_DATA4(0xAA, 0x31, 0x7D, 0xB5),
  CRYPTO_MPI_LIMB_DATA4(0xF4, 0xA3, 0x37, 0xB1),
  CRYPTO_MPI_LIMB_DATA4(0xE0, 0x00, 0x8F, 0x13),
  CRYPTO_MPI_LIMB_DATA4(0xAA, 0x4A, 0x49, 0x19),
  CRYPTO_MPI_LIMB_DATA4(0x35, 0x49, 0x26, 0xCF)
};

const CRYPTO_MPI_LIMB SSH_ServerKeys_RSA_Temp_1024b_PublicKey_E_aLimbs[] = {
  CRYPTO_MPI_LIMB_DATA3(0x01, 0x00, 0x01)
};

const CRYPTO_RSA_PUBLIC_KEY SSH_ServerKeys_RSA_Temp_1024b_PublicKey = {
  { CRYPTO_MPI_INIT_RO(SSH_ServerKeys_RSA_Temp_1024b_PublicKey_N_aLimbs) },
  { CRYPTO_MPI_INIT_RO(SSH_ServerKeys_RSA_Temp_1024b_PublicKey_E_aLimbs) },
};

const CRYPTO_MPI_LIMB SSH_ServerKeys_RSA_Temp_1024b_PrivateKey_D_aLimbs[] = {
  CRYPTO_MPI_LIMB_DATA4(0xC1, 0xA7, 0x91, 0x48),
  CRYPTO_MPI_LIMB_DATA4(0xC0, 0xDC, 0x2A, 0xD8),
  CRYPTO_MPI_LIMB_DATA4(0xEC, 0x43, 0xC4, 0x06),
  CRYPTO_MPI_LIMB_DATA4(0x85, 0x8F, 0xF5, 0x72),
  CRYPTO_MPI_LIMB_DATA4(0x0F, 0xB5, 0xD1, 0x01),
  CRYPTO_MPI_LIMB_DATA4(0x7A, 0xA1, 0xFC, 0x09),
  CRYPTO_MPI_LIMB_DATA4(0x34, 0xCD, 0xF4, 0x8B),
  CRYPTO_MPI_LIMB_DATA4(0x29, 0xAD, 0xAE, 0xFA),
  CRYPTO_MPI_LIMB_DATA4(0xA0, 0xAC, 0xB8, 0x6B),
  CRYPTO_MPI_LIMB_DATA4(0x22, 0x4B, 0xC0, 0xB3),
  CRYPTO_MPI_LIMB_DATA4(0x9D, 0x24, 0x16, 0xA8),
  CRYPTO_MPI_LIMB_DATA4(0x02, 0xA4, 0x19, 0xCE),
  CRYPTO_MPI_LIMB_DATA4(0x8C, 0xA4, 0x82, 0x98),
  CRYPTO_MPI_LIMB_DATA4(0x44, 0xA5, 0x08, 0x8B),
  CRYPTO_MPI_LIMB_DATA4(0x86, 0x88, 0x58, 0xE2),
  CRYPTO_MPI_LIMB_DATA4(0x5A, 0x33, 0x68, 0xEC),
  CRYPTO_MPI_LIMB_DATA4(0xCF, 0x1E, 0x98, 0x6F),
  CRYPTO_MPI_LIMB_DATA4(0x8C, 0xC2, 0xF9, 0x9D),
```

```
  CRYPTO_MPI_LIMB_DATA4(0xA6, 0xEC, 0xAF, 0x34),
  CRYPTO_MPI_LIMB_DATA4(0x5C, 0x89, 0xCB, 0x2D),
  CRYPTO_MPI_LIMB_DATA4(0x40, 0x83, 0xAA, 0x19),
  CRYPTO_MPI_LIMB_DATA4(0x31, 0x70, 0xF3, 0xA2),
  CRYPTO_MPI_LIMB_DATA4(0xF7, 0xE9, 0xB1, 0x3D),
  CRYPTO_MPI_LIMB_DATA4(0x41, 0x7B, 0x7E, 0xA1),
  CRYPTO_MPI_LIMB_DATA4(0xA2, 0x72, 0x37, 0x1B),
  CRYPTO_MPI_LIMB_DATA4(0x36, 0x58, 0x4D, 0x4D),
  CRYPTO_MPI_LIMB_DATA4(0xEC, 0x8B, 0x19, 0xB2),
  CRYPTO_MPI_LIMB_DATA4(0x83, 0x9B, 0x8E, 0x1D),
  CRYPTO_MPI_LIMB_DATA4(0x45, 0x1A, 0xD1, 0xD1),
  CRYPTO_MPI_LIMB_DATA4(0x3F, 0xE3, 0x2C, 0x18),
  CRYPTO_MPI_LIMB_DATA4(0x0A, 0x88, 0x71, 0xE6),
  CRYPTO_MPI_LIMB_DATA4(0x0D, 0xF5, 0x0F, 0x66)
};

const CRYPTO_MPI_LIMB SSH_ServerKeys_RSA_Temp_1024b_PrivateKey_P_aLimbs[] = {
  CRYPTO_MPI_LIMB_DATA4(0x4F, 0x79, 0x5B, 0xFF),
  CRYPTO_MPI_LIMB_DATA4(0x11, 0xCE, 0x43, 0x0C),
  CRYPTO_MPI_LIMB_DATA4(0x88, 0x92, 0xBA, 0x36),
  CRYPTO_MPI_LIMB_DATA4(0x20, 0x16, 0x8A, 0x1F),
  CRYPTO_MPI_LIMB_DATA4(0xD8, 0x52, 0xE8, 0x75),
  CRYPTO_MPI_LIMB_DATA4(0x80, 0xCB, 0x8D, 0x00),
  CRYPTO_MPI_LIMB_DATA4(0xB7, 0x5F, 0x33, 0x78),
  CRYPTO_MPI_LIMB_DATA4(0x6E, 0x0B, 0xA7, 0xDF),
  CRYPTO_MPI_LIMB_DATA4(0xCA, 0x5E, 0xC0, 0x23),
  CRYPTO_MPI_LIMB_DATA4(0x64, 0x83, 0x35, 0x83),
  CRYPTO_MPI_LIMB_DATA4(0x19, 0x19, 0xBE, 0x1C),
  CRYPTO_MPI_LIMB_DATA4(0xBB, 0x08, 0x32, 0x99),
  CRYPTO_MPI_LIMB_DATA4(0x05, 0xB4, 0x90, 0x27),
  CRYPTO_MPI_LIMB_DATA4(0x17, 0x54, 0x3D, 0x79),
  CRYPTO_MPI_LIMB_DATA4(0xA8, 0x1A, 0x7B, 0xA8),
  CRYPTO_MPI_LIMB_DATA4(0xE5, 0x25, 0xED, 0xDC)
};

const CRYPTO_MPI_LIMB SSH_ServerKeys_RSA_Temp_1024b_PrivateKey_Q_aLimbs[] = {
  CRYPTO_MPI_LIMB_DATA4(0x91, 0x6B, 0xB7, 0x24),
  CRYPTO_MPI_LIMB_DATA4(0xED, 0x15, 0x3C, 0x86),
  CRYPTO_MPI_LIMB_DATA4(0x46, 0x3B, 0x76, 0xD3),
  CRYPTO_MPI_LIMB_DATA4(0x29, 0x60, 0xB7, 0x3D),
  CRYPTO_MPI_LIMB_DATA4(0x10, 0x2E, 0x14, 0x5C),
  CRYPTO_MPI_LIMB_DATA4(0xE5, 0x47, 0x66, 0xE0),
  CRYPTO_MPI_LIMB_DATA4(0x24, 0xB2, 0x6B, 0x76),
  CRYPTO_MPI_LIMB_DATA4(0x35, 0x95, 0xBF, 0xB2),
  CRYPTO_MPI_LIMB_DATA4(0x3C, 0x1E, 0x63, 0xDB),
  CRYPTO_MPI_LIMB_DATA4(0xCD, 0x9B, 0x97, 0xD4),
  CRYPTO_MPI_LIMB_DATA4(0x6F, 0xFD, 0x45, 0xA8),
  CRYPTO_MPI_LIMB_DATA4(0x84, 0x77, 0x38, 0x65),
  CRYPTO_MPI_LIMB_DATA4(0x0D, 0xC1, 0xB2, 0xD2),
  CRYPTO_MPI_LIMB_DATA4(0xB1, 0xE0, 0x1E, 0xC8),
  CRYPTO_MPI_LIMB_DATA4(0x9A, 0xDC, 0x03, 0xCC),
  CRYPTO_MPI_LIMB_DATA4(0x2C, 0x39, 0x09, 0xF0)
};

const CRYPTO_MPI_LIMB SSH_ServerKeys_RSA_Temp_1024b_PrivateKey_DP_aLimbs[] = {
  CRYPTO_MPI_LIMB_DATA4(0x97, 0xAC, 0xC4, 0x63),
  CRYPTO_MPI_LIMB_DATA4(0xF9, 0x00, 0xC3, 0x86),
  CRYPTO_MPI_LIMB_DATA4(0x13, 0xFA, 0xDF, 0x93),
  CRYPTO_MPI_LIMB_DATA4(0x4C, 0x10, 0x67, 0x45),
  CRYPTO_MPI_LIMB_DATA4(0xE4, 0x16, 0xDA, 0xE3),
  CRYPTO_MPI_LIMB_DATA4(0xEE, 0x3D, 0x40, 0x5C),
  CRYPTO_MPI_LIMB_DATA4(0x53, 0x6D, 0xBC, 0xDC),
  CRYPTO_MPI_LIMB_DATA4(0xB5, 0x37, 0x4E, 0x7F),
  CRYPTO_MPI_LIMB_DATA4(0x0D, 0x74, 0x52, 0x01),
  CRYPTO_MPI_LIMB_DATA4(0xEA, 0xE1, 0xF7, 0x58),
  CRYPTO_MPI_LIMB_DATA4(0x16, 0xCF, 0x88, 0x8C),
  CRYPTO_MPI_LIMB_DATA4(0xF8, 0x80, 0x5E, 0x02),
  CRYPTO_MPI_LIMB_DATA4(0x9F, 0xA4, 0xCE, 0x39),
```

```
  CRYPTO_MPI_LIMB_DATA4(0xD3, 0x7B, 0xBB, 0x02),
  CRYPTO_MPI_LIMB_DATA4(0x6B, 0x7A, 0x10, 0x42),
  CRYPTO_MPI_LIMB_DATA4(0x4D, 0x64, 0xE7, 0xBB)
};
const CRYPTO_MPI_LIMB SSH_ServerKeys_RSA_Temp_1024b_PrivateKey_DQ_aLimbs[] = {
  CRYPTO_MPI_LIMB_DATA4(0x31, 0xC8, 0xE6, 0xC3),
  CRYPTO_MPI_LIMB_DATA4(0xD9, 0x15, 0x83, 0x1A),
  CRYPTO_MPI_LIMB_DATA4(0x92, 0x18, 0xC8, 0xE9),
  CRYPTO_MPI_LIMB_DATA4(0x73, 0xE8, 0x8A, 0x44),
  CRYPTO_MPI_LIMB_DATA4(0x80, 0x92, 0x15, 0xD8),
  CRYPTO_MPI_LIMB_DATA4(0x02, 0xD9, 0xF9, 0x98),
  CRYPTO_MPI_LIMB_DATA4(0x1C, 0xF6, 0xF3, 0x4C),
  CRYPTO_MPI_LIMB_DATA4(0x6D, 0x40, 0x2C, 0xC6),
  CRYPTO_MPI_LIMB_DATA4(0x0B, 0x8F, 0xF5, 0xC6),
  CRYPTO_MPI_LIMB_DATA4(0x07, 0x73, 0x90, 0xCE),
  CRYPTO_MPI_LIMB_DATA4(0x42, 0x74, 0xA6, 0x24),
  CRYPTO_MPI_LIMB_DATA4(0x3C, 0xD1, 0xA9, 0x39),
  CRYPTO_MPI_LIMB_DATA4(0x55, 0x98, 0xF6, 0x71),
  CRYPTO_MPI_LIMB_DATA4(0x79, 0xE6, 0x08, 0xE0),
  CRYPTO_MPI_LIMB_DATA4(0x07, 0xCF, 0xE0, 0x6A),
  CRYPTO_MPI_LIMB_DATA4(0x4D, 0xD6, 0x2A, 0x2C)
};

const CRYPTO_MPI_LIMB SSH_ServerKeys_RSA_Temp_1024b_PrivateKey_QInv_aLimbs[] = {
  CRYPTO_MPI_LIMB_DATA4(0x6B, 0xB4, 0x0D, 0xF5),
  CRYPTO_MPI_LIMB_DATA4(0x67, 0x15, 0x9E, 0x30),
  CRYPTO_MPI_LIMB_DATA4(0x17, 0x3D, 0xD3, 0xA8),
  CRYPTO_MPI_LIMB_DATA4(0x18, 0x4E, 0xDD, 0x55),
  CRYPTO_MPI_LIMB_DATA4(0x72, 0xEB, 0x2F, 0x5E),
  CRYPTO_MPI_LIMB_DATA4(0xCB, 0xA5, 0x44, 0xC3),
  CRYPTO_MPI_LIMB_DATA4(0x38, 0xC8, 0x89, 0x51),
  CRYPTO_MPI_LIMB_DATA4(0xD8, 0xBB, 0x2B, 0x78),
  CRYPTO_MPI_LIMB_DATA4(0xB8, 0x5C, 0x60, 0xE1),
  CRYPTO_MPI_LIMB_DATA4(0xD3, 0x6E, 0x1F, 0x8D),
  CRYPTO_MPI_LIMB_DATA4(0x26, 0xE6, 0x80, 0xDC),
  CRYPTO_MPI_LIMB_DATA4(0xBD, 0x99, 0x8F, 0xEF),
  CRYPTO_MPI_LIMB_DATA4(0x9F, 0x99, 0x7C, 0x6F),
  CRYPTO_MPI_LIMB_DATA4(0x4D, 0x99, 0x91, 0xB5),
  CRYPTO_MPI_LIMB_DATA4(0x53, 0x91, 0xCF, 0x0F),
  CRYPTO_MPI_LIMB_DATA4(0x14, 0x43, 0xA6, 0x86)
};

const CRYPTO_MPI_LIMB SSH_ServerKeys_RSA_Temp_1024b_PrivateKey_N_aLimbs[] = {
  CRYPTO_MPI_LIMB_DATA4(0xBF, 0xBA, 0xFC, 0xB1),
  CRYPTO_MPI_LIMB_DATA4(0x82, 0x6E, 0x76, 0xD1),
  CRYPTO_MPI_LIMB_DATA4(0xC7, 0x85, 0xD7, 0x6D),
  CRYPTO_MPI_LIMB_DATA4(0x62, 0x2F, 0xA6, 0xFE),
  CRYPTO_MPI_LIMB_DATA4(0x28, 0xA9, 0x22, 0xF7),
  CRYPTO_MPI_LIMB_DATA4(0xF4, 0x41, 0x07, 0xBD),
  CRYPTO_MPI_LIMB_DATA4(0x2D, 0x43, 0x2D, 0xC4),
  CRYPTO_MPI_LIMB_DATA4(0x67, 0x9F, 0x00, 0x37),
  CRYPTO_MPI_LIMB_DATA4(0x73, 0xDC, 0x1E, 0xAF),
  CRYPTO_MPI_LIMB_DATA4(0x15, 0x24, 0x3A, 0x3C),
  CRYPTO_MPI_LIMB_DATA4(0xE3, 0x6D, 0x89, 0x82),
  CRYPTO_MPI_LIMB_DATA4(0xDC, 0x79, 0xFB, 0x60),
  CRYPTO_MPI_LIMB_DATA4(0x6E, 0x4B, 0x6D, 0x4A),
  CRYPTO_MPI_LIMB_DATA4(0xAD, 0x95, 0x24, 0xCD),
  CRYPTO_MPI_LIMB_DATA4(0x1B, 0x19, 0x08, 0x92),
  CRYPTO_MPI_LIMB_DATA4(0x75, 0x7F, 0x6D, 0x14),
  CRYPTO_MPI_LIMB_DATA4(0xF3, 0x45, 0x69, 0xD8),
  CRYPTO_MPI_LIMB_DATA4(0xA2, 0x4B, 0x91, 0x15),
  CRYPTO_MPI_LIMB_DATA4(0x00, 0x40, 0x77, 0xB0),
  CRYPTO_MPI_LIMB_DATA4(0xBD, 0xE4, 0x9A, 0x83),
  CRYPTO_MPI_LIMB_DATA4(0x7A, 0x8A, 0xD8, 0x16),
  CRYPTO_MPI_LIMB_DATA4(0x7D, 0x34, 0x64, 0x3A),
  CRYPTO_MPI_LIMB_DATA4(0xE3, 0x0C, 0x28, 0x56),
  CRYPTO_MPI_LIMB_DATA4(0x40, 0xD3, 0x05, 0x2F),
```

```
  CRYPTO_MPI_LIMB_DATA4(0xCB, 0x3C, 0xC3, 0xCE),
  CRYPTO_MPI_LIMB_DATA4(0x57, 0x98, 0x9A, 0xED),
  CRYPTO_MPI_LIMB_DATA4(0xD4, 0xF8, 0x56, 0x12),
  CRYPTO_MPI_LIMB_DATA4(0xAA, 0x31, 0x7D, 0xB5),
  CRYPTO_MPI_LIMB_DATA4(0xF4, 0xA3, 0x37, 0xB1),
  CRYPTO_MPI_LIMB_DATA4(0xE0, 0x00, 0x8F, 0x13),
  CRYPTO_MPI_LIMB_DATA4(0xAA, 0x4A, 0x49, 0x19),
  CRYPTO_MPI_LIMB_DATA4(0x35, 0x49, 0x26, 0xCF)
};

const CRYPTO_MPI_LIMB SSH_ServerKeys_RSA_Temp_1024b_PrivateKey_E_aLimbs[] = {
  CRYPTO_MPI_LIMB_DATA4(0xBF, 0xBA, 0xFC, 0xB1),
  CRYPTO_MPI_LIMB_DATA4(0x82, 0x6E, 0x76, 0xD1),
  CRYPTO_MPI_LIMB_DATA4(0xC7, 0x85, 0xD7, 0x6D),
  CRYPTO_MPI_LIMB_DATA4(0x62, 0x2F, 0xA6, 0xFE),
  CRYPTO_MPI_LIMB_DATA4(0x28, 0xA9, 0x22, 0xF7),
  CRYPTO_MPI_LIMB_DATA4(0xF4, 0x41, 0x07, 0xBD),
  CRYPTO_MPI_LIMB_DATA4(0x2D, 0x43, 0x2D, 0xC4),
  CRYPTO_MPI_LIMB_DATA4(0x67, 0x9F, 0x00, 0x37),
  CRYPTO_MPI_LIMB_DATA4(0x73, 0xDC, 0x1E, 0xAF),
  CRYPTO_MPI_LIMB_DATA4(0x15, 0x24, 0x3A, 0x3C),
  CRYPTO_MPI_LIMB_DATA4(0xE3, 0x6D, 0x89, 0x82),
  CRYPTO_MPI_LIMB_DATA4(0xDC, 0x79, 0xFB, 0x60),
  CRYPTO_MPI_LIMB_DATA4(0x6E, 0x4B, 0x6D, 0x4A),
  CRYPTO_MPI_LIMB_DATA4(0xAD, 0x95, 0x24, 0xCD),
  CRYPTO_MPI_LIMB_DATA4(0x1B, 0x19, 0x08, 0x92),
  CRYPTO_MPI_LIMB_DATA4(0x75, 0x7F, 0x6D, 0x14),
  CRYPTO_MPI_LIMB_DATA4(0xF3, 0x45, 0x69, 0xD8),
  CRYPTO_MPI_LIMB_DATA4(0xA2, 0x4B, 0x91, 0x15),
  CRYPTO_MPI_LIMB_DATA4(0x00, 0x40, 0x77, 0xB0),
  CRYPTO_MPI_LIMB_DATA4(0xBD, 0xE4, 0x9A, 0x83),
  CRYPTO_MPI_LIMB_DATA4(0x7A, 0x8A, 0xD8, 0x16),
  CRYPTO_MPI_LIMB_DATA4(0x7D, 0x34, 0x64, 0x3A),
  CRYPTO_MPI_LIMB_DATA4(0xE3, 0x0C, 0x28, 0x56),
  CRYPTO_MPI_LIMB_DATA4(0x40, 0xD3, 0x05, 0x2F),
  CRYPTO_MPI_LIMB_DATA4(0xCB, 0x3C, 0xC3, 0xCE),
  CRYPTO_MPI_LIMB_DATA4(0x57, 0x98, 0x9A, 0xED),
  CRYPTO_MPI_LIMB_DATA4(0xD4, 0xF8, 0x56, 0x12),
  CRYPTO_MPI_LIMB_DATA4(0xAA, 0x31, 0x7D, 0xB5),
  CRYPTO_MPI_LIMB_DATA4(0xF4, 0xA3, 0x37, 0xB1),
  CRYPTO_MPI_LIMB_DATA4(0xE0, 0x00, 0x8F, 0x13),
  CRYPTO_MPI_LIMB_DATA4(0xAA, 0x4A, 0x49, 0x19),
  CRYPTO_MPI_LIMB_DATA4(0x35, 0x49, 0x26, 0xCF)
};

const CRYPTO_RSA_PRIVATE_KEY SSH_ServerKeys_RSA_Temp_1024b_PrivateKey = {
  { CRYPTO_MPI_INIT_RO(SSH_ServerKeys_RSA_Temp_1024b_PrivateKey_D_aLimbs) },
  { CRYPTO_MPI_INIT_RO(SSH_ServerKeys_RSA_Temp_1024b_PrivateKey_P_aLimbs) },
  { CRYPTO_MPI_INIT_RO(SSH_ServerKeys_RSA_Temp_1024b_PrivateKey_Q_aLimbs) },
  { CRYPTO_MPI_INIT_RO(SSH_ServerKeys_RSA_Temp_1024b_PrivateKey_DP_aLimbs) },
  { CRYPTO_MPI_INIT_RO(SSH_ServerKeys_RSA_Temp_1024b_PrivateKey_DQ_aLimbs) },
  { CRYPTO_MPI_INIT_RO(SSH_ServerKeys_RSA_Temp_1024b_PrivateKey_QInv_aLimbs) },
  { CRYPTO_MPI_INIT_RO(SSH_ServerKeys_RSA_Temp_1024b_PrivateKey_N_aLimbs) },
  { CRYPTO_MPI_INIT_RO(SSH_ServerKeys_RSA_Temp_1024b_PrivateKey_E_aLimbs) },
};

#include "CRYPTO.h"

const CRYPTO_MPI_LIMB SSH_ServerKeys_RSA_Temp_2048b_PublicKey_N_aLimbs[] = {
  CRYPTO_MPI_LIMB_DATA4(0x21, 0xD9, 0x92, 0xE2),
  CRYPTO_MPI_LIMB_DATA4(0x83, 0x66, 0x56, 0x6A),
  CRYPTO_MPI_LIMB_DATA4(0x1D, 0xB5, 0x70, 0xDC),
  CRYPTO_MPI_LIMB_DATA4(0x81, 0x41, 0x3D, 0x4A),
  CRYPTO_MPI_LIMB_DATA4(0x2F, 0x29, 0xA1, 0xE2),
  CRYPTO_MPI_LIMB_DATA4(0xD2, 0x4F, 0xBE, 0x78),
  CRYPTO_MPI_LIMB_DATA4(0x6B, 0x4D, 0xA5, 0x43),
  CRYPTO_MPI_LIMB_DATA4(0xB5, 0x5E, 0x02, 0xF7),
  CRYPTO_MPI_LIMB_DATA4(0x59, 0x74, 0x29, 0x06),
```

```
  CRYPTO_MPI_LIMB_DATA4(0x61, 0xAD, 0x55, 0x6A),
  CRYPTO_MPI_LIMB_DATA4(0x82, 0xC7, 0xFB, 0x14),
  CRYPTO_MPI_LIMB_DATA4(0x33, 0x04, 0x87, 0x6D),
  CRYPTO_MPI_LIMB_DATA4(0x96, 0x84, 0x57, 0x54),
  CRYPTO_MPI_LIMB_DATA4(0x1D, 0x44, 0x0C, 0x56),
  CRYPTO_MPI_LIMB_DATA4(0xD7, 0x75, 0xAE, 0x42),
  CRYPTO_MPI_LIMB_DATA4(0x7C, 0x46, 0x65, 0x59),
  CRYPTO_MPI_LIMB_DATA4(0x0D, 0xBA, 0xE3, 0xD7),
  CRYPTO_MPI_LIMB_DATA4(0x40, 0x5C, 0x05, 0xA3),
  CRYPTO_MPI_LIMB_DATA4(0x3E, 0x03, 0x81, 0x26),
  CRYPTO_MPI_LIMB_DATA4(0x0B, 0x17, 0x9B, 0xFA),
  CRYPTO_MPI_LIMB_DATA4(0x9E, 0x58, 0x95, 0x24),
  CRYPTO_MPI_LIMB_DATA4(0x8B, 0x7F, 0x25, 0xAA),
  CRYPTO_MPI_LIMB_DATA4(0xEE, 0xBB, 0xE7, 0x5E),
  CRYPTO_MPI_LIMB_DATA4(0x72, 0xF3, 0xF0, 0xC7),
  CRYPTO_MPI_LIMB_DATA4(0xBD, 0xD4, 0xE1, 0xFD),
  CRYPTO_MPI_LIMB_DATA4(0x14, 0x42, 0xB6, 0x4F),
  CRYPTO_MPI_LIMB_DATA4(0xD3, 0x37, 0x64, 0x9D),
  CRYPTO_MPI_LIMB_DATA4(0x17, 0x94, 0xDA, 0xE8),
  CRYPTO_MPI_LIMB_DATA4(0x07, 0xB6, 0x4B, 0xA5),
  CRYPTO_MPI_LIMB_DATA4(0xC6, 0x04, 0x20, 0x6A),
  CRYPTO_MPI_LIMB_DATA4(0x23, 0x06, 0xC8, 0x6D),
  CRYPTO_MPI_LIMB_DATA4(0x7E, 0xE8, 0x23, 0xB8),
  CRYPTO_MPI_LIMB_DATA4(0x8E, 0x72, 0x88, 0xD2),
  CRYPTO_MPI_LIMB_DATA4(0x60, 0xE0, 0x41, 0x8C),
  CRYPTO_MPI_LIMB_DATA4(0x73, 0x75, 0xB8, 0x78),
  CRYPTO_MPI_LIMB_DATA4(0xE5, 0x9D, 0xB7, 0x79),
  CRYPTO_MPI_LIMB_DATA4(0x25, 0xC0, 0x30, 0xD8),
  CRYPTO_MPI_LIMB_DATA4(0xB9, 0x98, 0xA6, 0xEC),
  CRYPTO_MPI_LIMB_DATA4(0x2A, 0x10, 0xB3, 0x93),
  CRYPTO_MPI_LIMB_DATA4(0x17, 0x19, 0xE1, 0xAB),
  CRYPTO_MPI_LIMB_DATA4(0xE1, 0xF5, 0xC9, 0x56),
  CRYPTO_MPI_LIMB_DATA4(0x04, 0xD7, 0x54, 0xD0),
  CRYPTO_MPI_LIMB_DATA4(0xA5, 0x9D, 0xA5, 0x2C),
  CRYPTO_MPI_LIMB_DATA4(0x9D, 0x0A, 0x7C, 0x59),
  CRYPTO_MPI_LIMB_DATA4(0x28, 0x6D, 0xE6, 0xDF),
  CRYPTO_MPI_LIMB_DATA4(0xFA, 0x8D, 0x01, 0x39),
  CRYPTO_MPI_LIMB_DATA4(0x87, 0x0B, 0x3C, 0x52),
  CRYPTO_MPI_LIMB_DATA4(0xE6, 0x8D, 0x6E, 0xC1),
  CRYPTO_MPI_LIMB_DATA4(0x79, 0x49, 0x0B, 0xE4),
  CRYPTO_MPI_LIMB_DATA4(0xB9, 0x84, 0x58, 0x64),
  CRYPTO_MPI_LIMB_DATA4(0x25, 0xF3, 0x10, 0xEB),
  CRYPTO_MPI_LIMB_DATA4(0x41, 0x98, 0x3C, 0xF3),
  CRYPTO_MPI_LIMB_DATA4(0x57, 0x9D, 0xC4, 0x59),
  CRYPTO_MPI_LIMB_DATA4(0x6C, 0x22, 0xB3, 0xFD),
  CRYPTO_MPI_LIMB_DATA4(0x19, 0xED, 0x5D, 0xE5),
  CRYPTO_MPI_LIMB_DATA4(0x69, 0x9B, 0x12, 0x03),
  CRYPTO_MPI_LIMB_DATA4(0x36, 0xAC, 0x4F, 0xBB),
  CRYPTO_MPI_LIMB_DATA4(0xC6, 0xB1, 0x1B, 0x64),
  CRYPTO_MPI_LIMB_DATA4(0xCF, 0xE4, 0xBB, 0xFD),
  CRYPTO_MPI_LIMB_DATA4(0x04, 0xFA, 0xEF, 0x6F),
  CRYPTO_MPI_LIMB_DATA4(0xB8, 0xC3, 0xD2, 0x9E),
  CRYPTO_MPI_LIMB_DATA4(0x18, 0xD3, 0xD1, 0xC6),
  CRYPTO_MPI_LIMB_DATA4(0xB1, 0x75, 0x27, 0x42),
  CRYPTO_MPI_LIMB_DATA4(0x7A, 0xBC, 0x0C, 0xB0)
};

const CRYPTO_MPI_LIMB SSH_ServerKeys_RSA_Temp_2048b_PublicKey_E_aLimbs[] = {
  CRYPTO_MPI_LIMB_DATA3(0x01, 0x00, 0x01)
};

const CRYPTO_RSA_PUBLIC_KEY SSH_ServerKeys_RSA_Temp_2048b_PublicKey = {
  { CRYPTO_MPI_INIT_RO(SSH_ServerKeys_RSA_Temp_2048b_PublicKey_N_aLimbs) },
  { CRYPTO_MPI_INIT_RO(SSH_ServerKeys_RSA_Temp_2048b_PublicKey_E_aLimbs) },
};

const CRYPTO_MPI_LIMB SSH_ServerKeys_RSA_Temp_2048b_PrivateKey_D_aLimbs[] = {
  CRYPTO_MPI_LIMB_DATA4(0x61, 0x53, 0x39, 0x85),
```

```
  CRYPTO_MPI_LIMB_DATA4(0xDE, 0x0B, 0x69, 0x38),
  CRYPTO_MPI_LIMB_DATA4(0x56, 0x71, 0x82, 0x3A),
  CRYPTO_MPI_LIMB_DATA4(0xB1, 0x1F, 0x6B, 0xCD),
  CRYPTO_MPI_LIMB_DATA4(0xF0, 0x2B, 0x2B, 0x04),
  CRYPTO_MPI_LIMB_DATA4(0xDC, 0x35, 0x0E, 0xF6),
  CRYPTO_MPI_LIMB_DATA4(0x41, 0xFC, 0xF6, 0x44),
  CRYPTO_MPI_LIMB_DATA4(0x56, 0x22, 0x16, 0xE3),
  CRYPTO_MPI_LIMB_DATA4(0x79, 0x8E, 0x25, 0x39),
  CRYPTO_MPI_LIMB_DATA4(0x63, 0x87, 0x59, 0xF2),
  CRYPTO_MPI_LIMB_DATA4(0x38, 0x45, 0xB4, 0xA3),
  CRYPTO_MPI_LIMB_DATA4(0x54, 0xBA, 0xE0, 0x08),
  CRYPTO_MPI_LIMB_DATA4(0x24, 0x2B, 0x3C, 0x0D),
  CRYPTO_MPI_LIMB_DATA4(0xFE, 0xF3, 0x03, 0xD5),
  CRYPTO_MPI_LIMB_DATA4(0xA0, 0x7E, 0xEC, 0x66),
  CRYPTO_MPI_LIMB_DATA4(0x28, 0x4B, 0x65, 0x38),
  CRYPTO_MPI_LIMB_DATA4(0xDD, 0x05, 0xA6, 0x2F),
  CRYPTO_MPI_LIMB_DATA4(0x6F, 0x11, 0x80, 0xCF),
  CRYPTO_MPI_LIMB_DATA4(0xF8, 0x03, 0x36, 0xF6),
  CRYPTO_MPI_LIMB_DATA4(0x5A, 0xC6, 0x3F, 0xA7),
  CRYPTO_MPI_LIMB_DATA4(0x8F, 0x0D, 0x9F, 0x09),
  CRYPTO_MPI_LIMB_DATA4(0x33, 0x74, 0xF0, 0x20),
  CRYPTO_MPI_LIMB_DATA4(0x32, 0x42, 0x32, 0xE6),
  CRYPTO_MPI_LIMB_DATA4(0x5B, 0x43, 0xE1, 0xED),
  CRYPTO_MPI_LIMB_DATA4(0xF4, 0x6F, 0xAE, 0x34),
  CRYPTO_MPI_LIMB_DATA4(0x36, 0xCF, 0x0E, 0x8C),
  CRYPTO_MPI_LIMB_DATA4(0x4D, 0xC5, 0x9C, 0x15),
  CRYPTO_MPI_LIMB_DATA4(0x66, 0xE4, 0xDD, 0xE1),
  CRYPTO_MPI_LIMB_DATA4(0xFF, 0x4A, 0x02, 0xA9),
  CRYPTO_MPI_LIMB_DATA4(0x01, 0xD1, 0xD8, 0x39),
  CRYPTO_MPI_LIMB_DATA4(0x23, 0x6E, 0xBA, 0x84),
  CRYPTO_MPI_LIMB_DATA4(0x3C, 0x1E, 0xB6, 0x43),
  CRYPTO_MPI_LIMB_DATA4(0x31, 0x31, 0x30, 0x4C),
  CRYPTO_MPI_LIMB_DATA4(0x4F, 0x83, 0xE9, 0x93),
  CRYPTO_MPI_LIMB_DATA4(0x3D, 0x5F, 0xF7, 0xC1),
  CRYPTO_MPI_LIMB_DATA4(0xA9, 0xB9, 0x03, 0x18),
  CRYPTO_MPI_LIMB_DATA4(0x45, 0x3C, 0xFE, 0x88),
  CRYPTO_MPI_LIMB_DATA4(0x64, 0x0F, 0x89, 0x72),
  CRYPTO_MPI_LIMB_DATA4(0xBA, 0x5A, 0x9F, 0x15),
  CRYPTO_MPI_LIMB_DATA4(0x38, 0x17, 0x41, 0x32),
  CRYPTO_MPI_LIMB_DATA4(0x0D, 0xE2, 0x13, 0x14),
  CRYPTO_MPI_LIMB_DATA4(0xB4, 0xF6, 0xA5, 0xDB),
  CRYPTO_MPI_LIMB_DATA4(0x1B, 0xB9, 0xC3, 0x21),
  CRYPTO_MPI_LIMB_DATA4(0xF8, 0x84, 0x66, 0xC8),
  CRYPTO_MPI_LIMB_DATA4(0x54, 0xE8, 0x51, 0xDD),
  CRYPTO_MPI_LIMB_DATA4(0x95, 0xD2, 0x51, 0x2D),
  CRYPTO_MPI_LIMB_DATA4(0x4A, 0xE1, 0xB8, 0xB9),
  CRYPTO_MPI_LIMB_DATA4(0xBD, 0xC3, 0x90, 0xC0),
  CRYPTO_MPI_LIMB_DATA4(0xC7, 0x11, 0xAC, 0xB7),
  CRYPTO_MPI_LIMB_DATA4(0x8E, 0x7A, 0x46, 0x52),
  CRYPTO_MPI_LIMB_DATA4(0xAF, 0xD4, 0x42, 0x94),
  CRYPTO_MPI_LIMB_DATA4(0x9F, 0x5C, 0x47, 0x66),
  CRYPTO_MPI_LIMB_DATA4(0xEC, 0x11, 0x55, 0x85),
  CRYPTO_MPI_LIMB_DATA4(0x39, 0x39, 0xF7, 0x2D),
  CRYPTO_MPI_LIMB_DATA4(0x01, 0x47, 0x2A, 0x2F),
  CRYPTO_MPI_LIMB_DATA4(0xA5, 0x6F, 0x4B, 0xEB),
  CRYPTO_MPI_LIMB_DATA4(0x7A, 0x93, 0x1C, 0x37),
  CRYPTO_MPI_LIMB_DATA4(0x72, 0xE4, 0x5A, 0x72),
  CRYPTO_MPI_LIMB_DATA4(0x05, 0xD8, 0xF7, 0xEB),
  CRYPTO_MPI_LIMB_DATA4(0xC7, 0xEE, 0x29, 0x73),
  CRYPTO_MPI_LIMB_DATA4(0x2F, 0x33, 0x86, 0x31),
  CRYPTO_MPI_LIMB_DATA4(0xDE, 0xB8, 0x69, 0x56),
  CRYPTO_MPI_LIMB_DATA4(0xB5, 0x4A, 0x97, 0xFC),
  CRYPTO_MPI_LIMB_DATA4(0xC6, 0x3F, 0x62, 0x1B)
};

const CRYPTO_MPI_LIMB SSH_ServerKeys_RSA_Temp_2048b_PrivateKey_P_aLimbs[] = {
  CRYPTO_MPI_LIMB_DATA4(0x35, 0xDA, 0xDE, 0xE4),
  CRYPTO_MPI_LIMB_DATA4(0x2E, 0x27, 0xC7, 0x4F),
```

```
  CRYPTO_MPI_LIMB_DATA4(0x75, 0xE7, 0xF0, 0x97),
  CRYPTO_MPI_LIMB_DATA4(0xAC, 0xC0, 0x1F, 0xA2),
  CRYPTO_MPI_LIMB_DATA4(0x08, 0x5F, 0xAB, 0xA3),
  CRYPTO_MPI_LIMB_DATA4(0xD2, 0xFF, 0x95, 0x38),
  CRYPTO_MPI_LIMB_DATA4(0xC8, 0x60, 0x23, 0xF7),
  CRYPTO_MPI_LIMB_DATA4(0xF8, 0xA1, 0xC5, 0x07),
  CRYPTO_MPI_LIMB_DATA4(0x5D, 0xA0, 0x72, 0xB2),
  CRYPTO_MPI_LIMB_DATA4(0x3A, 0x63, 0x55, 0xE3),
  CRYPTO_MPI_LIMB_DATA4(0xFB, 0xDA, 0x96, 0x2D),
  CRYPTO_MPI_LIMB_DATA4(0xA8, 0x2D, 0x49, 0x6B),
  CRYPTO_MPI_LIMB_DATA4(0x1F, 0x8F, 0x0E, 0x39),
  CRYPTO_MPI_LIMB_DATA4(0x6C, 0x48, 0x1F, 0x79),
  CRYPTO_MPI_LIMB_DATA4(0xE2, 0xAD, 0x30, 0x21),
  CRYPTO_MPI_LIMB_DATA4(0x93, 0x23, 0x69, 0xD3),
  CRYPTO_MPI_LIMB_DATA4(0x68, 0x8E, 0xC2, 0x8C),
  CRYPTO_MPI_LIMB_DATA4(0x38, 0x91, 0x8B, 0x85),
  CRYPTO_MPI_LIMB_DATA4(0xEC, 0xC2, 0xA1, 0xF8),
  CRYPTO_MPI_LIMB_DATA4(0x81, 0xC6, 0xF9, 0x44),
  CRYPTO_MPI_LIMB_DATA4(0x73, 0x59, 0x3A, 0x01),
  CRYPTO_MPI_LIMB_DATA4(0x25, 0x9E, 0x60, 0x9C),
  CRYPTO_MPI_LIMB_DATA4(0xF9, 0xBB, 0x2B, 0x30),
  CRYPTO_MPI_LIMB_DATA4(0xFC, 0x81, 0x03, 0xFE),
  CRYPTO_MPI_LIMB_DATA4(0x08, 0xFD, 0xEE, 0xB7),
  CRYPTO_MPI_LIMB_DATA4(0x30, 0x34, 0x3A, 0x75),
  CRYPTO_MPI_LIMB_DATA4(0xF9, 0x6C, 0x60, 0x46),
  CRYPTO_MPI_LIMB_DATA4(0x83, 0x43, 0x39, 0xA3),
  CRYPTO_MPI_LIMB_DATA4(0xAC, 0x8E, 0xFC, 0x13),
  CRYPTO_MPI_LIMB_DATA4(0xC1, 0x5B, 0x43, 0xED),
  CRYPTO_MPI_LIMB_DATA4(0x1E, 0xED, 0x4F, 0x24),
  CRYPTO_MPI_LIMB_DATA4(0xC7, 0xDB, 0x36, 0xC5)
};

const CRYPTO_MPI_LIMB SSH_ServerKeys_RSA_Temp_2048b_PrivateKey_Q_aLimbs[] = {
  CRYPTO_MPI_LIMB_DATA4(0xBD, 0xC0, 0x54, 0x19),
  CRYPTO_MPI_LIMB_DATA4(0x73, 0x1F, 0xCE, 0x84),
  CRYPTO_MPI_LIMB_DATA4(0x7D, 0x10, 0xB9, 0x8A),
  CRYPTO_MPI_LIMB_DATA4(0xC5, 0xAC, 0x8C, 0x6C),
  CRYPTO_MPI_LIMB_DATA4(0xFA, 0x1D, 0xDB, 0x01),
  CRYPTO_MPI_LIMB_DATA4(0xB5, 0x04, 0x1E, 0xDF),
  CRYPTO_MPI_LIMB_DATA4(0x66, 0xBD, 0x11, 0xFD),
  CRYPTO_MPI_LIMB_DATA4(0xAA, 0x99, 0xA0, 0xC0),
  CRYPTO_MPI_LIMB_DATA4(0x14, 0xB9, 0xBE, 0xFA),
  CRYPTO_MPI_LIMB_DATA4(0x34, 0x66, 0x7B, 0x37),
  CRYPTO_MPI_LIMB_DATA4(0xFC, 0x21, 0x1E, 0x6C),
  CRYPTO_MPI_LIMB_DATA4(0x4C, 0x7B, 0x0B, 0xA3),
  CRYPTO_MPI_LIMB_DATA4(0x93, 0xC1, 0xB9, 0x89),
  CRYPTO_MPI_LIMB_DATA4(0x9D, 0x38, 0x5F, 0x2E),
  CRYPTO_MPI_LIMB_DATA4(0x0D, 0x6A, 0x6C, 0x70),
  CRYPTO_MPI_LIMB_DATA4(0xEB, 0xB3, 0x1E, 0x64),
  CRYPTO_MPI_LIMB_DATA4(0x7D, 0xB7, 0xCF, 0x4D),
  CRYPTO_MPI_LIMB_DATA4(0x48, 0xE8, 0x6B, 0x21),
  CRYPTO_MPI_LIMB_DATA4(0x7F, 0xE6, 0x7B, 0x62),
  CRYPTO_MPI_LIMB_DATA4(0x29, 0x9D, 0xD6, 0x39),
  CRYPTO_MPI_LIMB_DATA4(0xFC, 0x3D, 0xF1, 0x20),
  CRYPTO_MPI_LIMB_DATA4(0x15, 0x3F, 0xA2, 0xDC),
  CRYPTO_MPI_LIMB_DATA4(0x8D, 0x0C, 0x17, 0xEE),
  CRYPTO_MPI_LIMB_DATA4(0xB5, 0x77, 0x8A, 0x53),
  CRYPTO_MPI_LIMB_DATA4(0x6E, 0xC3, 0xE4, 0x1B),
  CRYPTO_MPI_LIMB_DATA4(0xBF, 0x1F, 0x66, 0xC9),
  CRYPTO_MPI_LIMB_DATA4(0x8F, 0x54, 0x11, 0x94),
  CRYPTO_MPI_LIMB_DATA4(0x7B, 0xC0, 0x3B, 0x50),
  CRYPTO_MPI_LIMB_DATA4(0x69, 0x29, 0xB5, 0x90),
  CRYPTO_MPI_LIMB_DATA4(0x7F, 0xFF, 0x4D, 0x17),
  CRYPTO_MPI_LIMB_DATA4(0xAC, 0x44, 0xC4, 0xD3),
  CRYPTO_MPI_LIMB_DATA4(0x82, 0xD7, 0x86, 0xE4)
};

const CRYPTO_MPI_LIMB SSH_ServerKeys_RSA_Temp_2048b_PrivateKey_DP_aLimbs[] = {
```

```
  CRYPTO_MPI_LIMB_DATA4(0x05, 0x37, 0x19, 0xE9),
  CRYPTO_MPI_LIMB_DATA4(0x9F, 0xD4, 0xC3, 0xD8),
  CRYPTO_MPI_LIMB_DATA4(0xCB, 0x20, 0x36, 0xF3),
  CRYPTO_MPI_LIMB_DATA4(0x3D, 0xD9, 0x41, 0x65),
  CRYPTO_MPI_LIMB_DATA4(0xF1, 0x1C, 0x98, 0xE2),
  CRYPTO_MPI_LIMB_DATA4(0xD8, 0x6D, 0x0C, 0x63),
  CRYPTO_MPI_LIMB_DATA4(0xB1, 0xD3, 0x85, 0xC5),
  CRYPTO_MPI_LIMB_DATA4(0xA4, 0x0B, 0x38, 0xEF),
  CRYPTO_MPI_LIMB_DATA4(0x15, 0x01, 0xD9, 0x6B),
  CRYPTO_MPI_LIMB_DATA4(0x74, 0xC0, 0x34, 0x08),
  CRYPTO_MPI_LIMB_DATA4(0x3A, 0x75, 0xA2, 0x4E),
  CRYPTO_MPI_LIMB_DATA4(0x60, 0xC6, 0xCA, 0x53),
  CRYPTO_MPI_LIMB_DATA4(0x5B, 0x7B, 0x52, 0x6B),
  CRYPTO_MPI_LIMB_DATA4(0xB6, 0xB5, 0xC2, 0xA4),
  CRYPTO_MPI_LIMB_DATA4(0xD8, 0x19, 0x63, 0xDD),
  CRYPTO_MPI_LIMB_DATA4(0xD6, 0xFD, 0x05, 0x43),
  CRYPTO_MPI_LIMB_DATA4(0x7D, 0x95, 0x91, 0x67),
  CRYPTO_MPI_LIMB_DATA4(0x7A, 0x41, 0x0B, 0x11),
  CRYPTO_MPI_LIMB_DATA4(0x23, 0x03, 0x55, 0xCE),
  CRYPTO_MPI_LIMB_DATA4(0xCB, 0xF2, 0x13, 0x35),
  CRYPTO_MPI_LIMB_DATA4(0x60, 0x9A, 0xE2, 0x96),
  CRYPTO_MPI_LIMB_DATA4(0x12, 0x95, 0x15, 0xF7),
  CRYPTO_MPI_LIMB_DATA4(0xB8, 0x7B, 0x1C, 0xD4),
  CRYPTO_MPI_LIMB_DATA4(0x08, 0xFB, 0xE9, 0xA6),
  CRYPTO_MPI_LIMB_DATA4(0xE1, 0x02, 0x38, 0x53),
  CRYPTO_MPI_LIMB_DATA4(0xF0, 0x04, 0x30, 0x69),
  CRYPTO_MPI_LIMB_DATA4(0x1F, 0x77, 0x9E, 0x5D),
  CRYPTO_MPI_LIMB_DATA4(0x6A, 0x14, 0xA0, 0x11),
  CRYPTO_MPI_LIMB_DATA4(0xFD, 0xC2, 0x40, 0xF8),
  CRYPTO_MPI_LIMB_DATA4(0x57, 0xC7, 0x88, 0xBF),
  CRYPTO_MPI_LIMB_DATA4(0x7B, 0xCD, 0x76, 0x34),
  CRYPTO_MPI_LIMB_DATA4(0x60, 0x34, 0xC3, 0x25)
};

const CRYPTO_MPI_LIMB SSH_ServerKeys_RSA_Temp_2048b_PrivateKey_DQ_aLimbs[] = {
  CRYPTO_MPI_LIMB_DATA4(0x29, 0xEA, 0x92, 0x56),
  CRYPTO_MPI_LIMB_DATA4(0x0C, 0xE5, 0x9B, 0xE7),
  CRYPTO_MPI_LIMB_DATA4(0x8E, 0x29, 0x28, 0xCF),
  CRYPTO_MPI_LIMB_DATA4(0xD2, 0x9E, 0xDD, 0x13),
  CRYPTO_MPI_LIMB_DATA4(0xC8, 0x06, 0xD7, 0xC7),
  CRYPTO_MPI_LIMB_DATA4(0x09, 0xE0, 0xA0, 0x1B),
  CRYPTO_MPI_LIMB_DATA4(0x22, 0x47, 0xDD, 0xD5),
  CRYPTO_MPI_LIMB_DATA4(0x07, 0xAE, 0xE6, 0x13),
  CRYPTO_MPI_LIMB_DATA4(0x49, 0x87, 0x29, 0x94),
  CRYPTO_MPI_LIMB_DATA4(0x8F, 0xB1, 0xDB, 0x48),
  CRYPTO_MPI_LIMB_DATA4(0x6A, 0xF5, 0x3D, 0xB1),
  CRYPTO_MPI_LIMB_DATA4(0x42, 0x49, 0xE4, 0x50),
  CRYPTO_MPI_LIMB_DATA4(0x95, 0x46, 0xB4, 0x6A),
  CRYPTO_MPI_LIMB_DATA4(0x65, 0x4B, 0x63, 0xE8),
  CRYPTO_MPI_LIMB_DATA4(0xCD, 0x0C, 0x88, 0xAD),
  CRYPTO_MPI_LIMB_DATA4(0x0B, 0xAB, 0x5C, 0x8C),
  CRYPTO_MPI_LIMB_DATA4(0x4D, 0x50, 0x66, 0xF7),
  CRYPTO_MPI_LIMB_DATA4(0x4F, 0x0D, 0x72, 0x29),
  CRYPTO_MPI_LIMB_DATA4(0xCC, 0x02, 0x8B, 0x62),
  CRYPTO_MPI_LIMB_DATA4(0x2D, 0xF8, 0xF1, 0xA6),
  CRYPTO_MPI_LIMB_DATA4(0x5F, 0x00, 0x57, 0xC9),
  CRYPTO_MPI_LIMB_DATA4(0x94, 0x80, 0x54, 0x64),
  CRYPTO_MPI_LIMB_DATA4(0x2E, 0x95, 0xDD, 0x18),
  CRYPTO_MPI_LIMB_DATA4(0xD4, 0x59, 0x25, 0xB8),
  CRYPTO_MPI_LIMB_DATA4(0x5F, 0x7D, 0xEE, 0xD4),
  CRYPTO_MPI_LIMB_DATA4(0x5B, 0x20, 0x7E, 0xF5),
  CRYPTO_MPI_LIMB_DATA4(0xA7, 0x0E, 0xD3, 0x59),
  CRYPTO_MPI_LIMB_DATA4(0xF1, 0xFA, 0x07, 0x68),
  CRYPTO_MPI_LIMB_DATA4(0x17, 0x58, 0x56, 0x69),
  CRYPTO_MPI_LIMB_DATA4(0x48, 0x9A, 0x32, 0x7B),
  CRYPTO_MPI_LIMB_DATA4(0xEC, 0xB7, 0x63, 0xFF),
  CRYPTO_MPI_LIMB_DATA4(0xBD, 0x06, 0x65, 0x65)
};
```

```
const CRYPTO_MPI_LIMB SSH_ServerKeys_RSA_Temp_2048b_PrivateKey_QInv_aLimbs[] = {
  CRYPTO_MPI_LIMB_DATA4(0x9D, 0xA2, 0xF4, 0x5C),
  CRYPTO_MPI_LIMB_DATA4(0xE0, 0x45, 0xB3, 0x53),
  CRYPTO_MPI_LIMB_DATA4(0xBA, 0x9E, 0xD2, 0xB3),
  CRYPTO_MPI_LIMB_DATA4(0x05, 0xC2, 0xD4, 0xC3),
  CRYPTO_MPI_LIMB_DATA4(0x7B, 0x90, 0xF7, 0xCB),
  CRYPTO_MPI_LIMB_DATA4(0x21, 0xC6, 0xC6, 0xD8),
  CRYPTO_MPI_LIMB_DATA4(0x50, 0x7B, 0xBB, 0x1E),
  CRYPTO_MPI_LIMB_DATA4(0x2B, 0xED, 0x5B, 0xFB),
  CRYPTO_MPI_LIMB_DATA4(0xA3, 0x38, 0xB9, 0xBD),
  CRYPTO_MPI_LIMB_DATA4(0xA3, 0x66, 0xB9, 0x3A),
  CRYPTO_MPI_LIMB_DATA4(0xFF, 0x98, 0xB0, 0xCE),
  CRYPTO_MPI_LIMB_DATA4(0x87, 0x70, 0x9D, 0x1E),
  CRYPTO_MPI_LIMB_DATA4(0x1F, 0xB5, 0x69, 0xD4),
  CRYPTO_MPI_LIMB_DATA4(0xBF, 0x70, 0xE9, 0x60),
  CRYPTO_MPI_LIMB_DATA4(0x27, 0x4F, 0x90, 0x3B),
  CRYPTO_MPI_LIMB_DATA4(0x93, 0xA2, 0xA7, 0x5B),
  CRYPTO_MPI_LIMB_DATA4(0xE7, 0xD0, 0xAD, 0x74),
  CRYPTO_MPI_LIMB_DATA4(0x29, 0x9D, 0xEC, 0x4F),
  CRYPTO_MPI_LIMB_DATA4(0x18, 0x53, 0x6A, 0x39),
  CRYPTO_MPI_LIMB_DATA4(0x54, 0xFF, 0x0A, 0x1D),
  CRYPTO_MPI_LIMB_DATA4(0x6F, 0x38, 0xCA, 0xE7),
  CRYPTO_MPI_LIMB_DATA4(0x0A, 0x37, 0xDA, 0x79),
  CRYPTO_MPI_LIMB_DATA4(0x02, 0x17, 0x50, 0xA1),
  CRYPTO_MPI_LIMB_DATA4(0x4B, 0x13, 0xBF, 0x39),
  CRYPTO_MPI_LIMB_DATA4(0x4F, 0x29, 0x35, 0x9A),
  CRYPTO_MPI_LIMB_DATA4(0xD8, 0x44, 0x08, 0xD6),
  CRYPTO_MPI_LIMB_DATA4(0x7F, 0x30, 0x9C, 0xBB),
  CRYPTO_MPI_LIMB_DATA4(0x96, 0x12, 0xB5, 0x25),
  CRYPTO_MPI_LIMB_DATA4(0x04, 0xCB, 0xBE, 0x91),
  CRYPTO_MPI_LIMB_DATA4(0xDC, 0x89, 0xA1, 0xF2),
  CRYPTO_MPI_LIMB_DATA4(0x36, 0x17, 0x33, 0x75),
  CRYPTO_MPI_LIMB_DATA4(0xB2, 0xDB, 0x8A, 0x8A)
};

const CRYPTO_MPI_LIMB SSH_ServerKeys_RSA_Temp_2048b_PrivateKey_N_aLimbs[] = {
  CRYPTO_MPI_LIMB_DATA4(0x21, 0xD9, 0x92, 0xE2),
  CRYPTO_MPI_LIMB_DATA4(0x83, 0x66, 0x56, 0x6A),
  CRYPTO_MPI_LIMB_DATA4(0x1D, 0xB5, 0x70, 0xDC),
  CRYPTO_MPI_LIMB_DATA4(0x81, 0x41, 0x3D, 0x4A),
  CRYPTO_MPI_LIMB_DATA4(0x2F, 0x29, 0xA1, 0xE2),
  CRYPTO_MPI_LIMB_DATA4(0xD2, 0x4F, 0xBE, 0x78),
  CRYPTO_MPI_LIMB_DATA4(0x6B, 0x4D, 0xA5, 0x43),
  CRYPTO_MPI_LIMB_DATA4(0xB5, 0x5E, 0x02, 0xF7),
  CRYPTO_MPI_LIMB_DATA4(0x59, 0x74, 0x29, 0x06),
  CRYPTO_MPI_LIMB_DATA4(0x61, 0xAD, 0x55, 0x6A),
  CRYPTO_MPI_LIMB_DATA4(0x82, 0xC7, 0xFB, 0x14),
  CRYPTO_MPI_LIMB_DATA4(0x33, 0x04, 0x87, 0x6D),
  CRYPTO_MPI_LIMB_DATA4(0x96, 0x84, 0x57, 0x54),
  CRYPTO_MPI_LIMB_DATA4(0x1D, 0x44, 0x0C, 0x56),
  CRYPTO_MPI_LIMB_DATA4(0xD7, 0x75, 0xAE, 0x42),
  CRYPTO_MPI_LIMB_DATA4(0x7C, 0x46, 0x65, 0x59),
  CRYPTO_MPI_LIMB_DATA4(0x0D, 0xBA, 0xE3, 0xD7),
  CRYPTO_MPI_LIMB_DATA4(0x40, 0x5C, 0x05, 0xA3),
  CRYPTO_MPI_LIMB_DATA4(0x3E, 0x03, 0x81, 0x26),
  CRYPTO_MPI_LIMB_DATA4(0x0B, 0x17, 0x9B, 0xFA),
  CRYPTO_MPI_LIMB_DATA4(0x9E, 0x58, 0x95, 0x24),
  CRYPTO_MPI_LIMB_DATA4(0x8B, 0x7F, 0x25, 0xAA),
  CRYPTO_MPI_LIMB_DATA4(0xEE, 0xBB, 0xE7, 0x5E),
  CRYPTO_MPI_LIMB_DATA4(0x72, 0xF3, 0xF0, 0xC7),
  CRYPTO_MPI_LIMB_DATA4(0xBD, 0xD4, 0xE1, 0xFD),
  CRYPTO_MPI_LIMB_DATA4(0x14, 0x42, 0xB6, 0x4F),
  CRYPTO_MPI_LIMB_DATA4(0xD3, 0x37, 0x64, 0x9D),
  CRYPTO_MPI_LIMB_DATA4(0x17, 0x94, 0xDA, 0xE8),
  CRYPTO_MPI_LIMB_DATA4(0x07, 0xB6, 0x4B, 0xA5),
  CRYPTO_MPI_LIMB_DATA4(0xC6, 0x04, 0x20, 0x6A),
  CRYPTO_MPI_LIMB_DATA4(0x23, 0x06, 0xC8, 0x6D),
```

```
  CRYPTO_MPI_LIMB_DATA4(0x7E, 0xE8, 0x23, 0xB8),
  CRYPTO_MPI_LIMB_DATA4(0x8E, 0x72, 0x88, 0xD2),
  CRYPTO_MPI_LIMB_DATA4(0x60, 0xE0, 0x41, 0x8C),
  CRYPTO_MPI_LIMB_DATA4(0x73, 0x75, 0xB8, 0x78),
  CRYPTO_MPI_LIMB_DATA4(0xE5, 0x9D, 0xB7, 0x79),
  CRYPTO_MPI_LIMB_DATA4(0x25, 0xC0, 0x30, 0xD8),
  CRYPTO_MPI_LIMB_DATA4(0xB9, 0x98, 0xA6, 0xEC),
  CRYPTO_MPI_LIMB_DATA4(0x2A, 0x10, 0xB3, 0x93),
  CRYPTO_MPI_LIMB_DATA4(0x17, 0x19, 0xE1, 0xAB),
  CRYPTO_MPI_LIMB_DATA4(0xE1, 0xF5, 0xC9, 0x56),
  CRYPTO_MPI_LIMB_DATA4(0x04, 0xD7, 0x54, 0xD0),
  CRYPTO_MPI_LIMB_DATA4(0xA5, 0x9D, 0xA5, 0x2C),
  CRYPTO_MPI_LIMB_DATA4(0x9D, 0x0A, 0x7C, 0x59),
  CRYPTO_MPI_LIMB_DATA4(0x28, 0x6D, 0xE6, 0xDF),
  CRYPTO_MPI_LIMB_DATA4(0xFA, 0x8D, 0x01, 0x39),
  CRYPTO_MPI_LIMB_DATA4(0x87, 0x0B, 0x3C, 0x52),
  CRYPTO_MPI_LIMB_DATA4(0xE6, 0x8D, 0x6E, 0xC1),
  CRYPTO_MPI_LIMB_DATA4(0x79, 0x49, 0x0B, 0xE4),
  CRYPTO_MPI_LIMB_DATA4(0xB9, 0x84, 0x58, 0x64),
  CRYPTO_MPI_LIMB_DATA4(0x25, 0xF3, 0x10, 0xEB),
  CRYPTO_MPI_LIMB_DATA4(0x41, 0x98, 0x3C, 0xF3),
  CRYPTO_MPI_LIMB_DATA4(0x57, 0x9D, 0xC4, 0x59),
  CRYPTO_MPI_LIMB_DATA4(0x6C, 0x22, 0xB3, 0xFD),
  CRYPTO_MPI_LIMB_DATA4(0x19, 0xED, 0x5D, 0xE5),
  CRYPTO_MPI_LIMB_DATA4(0x69, 0x9B, 0x12, 0x03),
  CRYPTO_MPI_LIMB_DATA4(0x36, 0xAC, 0x4F, 0xBB),
  CRYPTO_MPI_LIMB_DATA4(0xC6, 0xB1, 0x1B, 0x64),
  CRYPTO_MPI_LIMB_DATA4(0xCF, 0xE4, 0xBB, 0xFD),
  CRYPTO_MPI_LIMB_DATA4(0x04, 0xFA, 0xEF, 0x6F),
  CRYPTO_MPI_LIMB_DATA4(0xB8, 0xC3, 0xD2, 0x9E),
  CRYPTO_MPI_LIMB_DATA4(0x18, 0xD3, 0xD1, 0xC6),
  CRYPTO_MPI_LIMB_DATA4(0xB1, 0x75, 0x27, 0x42),
  CRYPTO_MPI_LIMB_DATA4(0x7A, 0xBC, 0x0C, 0xB0)
};

const CRYPTO_MPI_LIMB SSH_ServerKeys_RSA_Temp_2048b_PrivateKey_E_aLimbs[] = {
  CRYPTO_MPI_LIMB_DATA4(0x21, 0xD9, 0x92, 0xE2),
  CRYPTO_MPI_LIMB_DATA4(0x83, 0x66, 0x56, 0x6A),
  CRYPTO_MPI_LIMB_DATA4(0x1D, 0xB5, 0x70, 0xDC),
  CRYPTO_MPI_LIMB_DATA4(0x81, 0x41, 0x3D, 0x4A),
  CRYPTO_MPI_LIMB_DATA4(0x2F, 0x29, 0xA1, 0xE2),
  CRYPTO_MPI_LIMB_DATA4(0xD2, 0x4F, 0xBE, 0x78),
  CRYPTO_MPI_LIMB_DATA4(0x6B, 0x4D, 0xA5, 0x43),
  CRYPTO_MPI_LIMB_DATA4(0xB5, 0x5E, 0x02, 0xF7),
  CRYPTO_MPI_LIMB_DATA4(0x59, 0x74, 0x29, 0x06),
  CRYPTO_MPI_LIMB_DATA4(0x61, 0xAD, 0x55, 0x6A),
  CRYPTO_MPI_LIMB_DATA4(0x82, 0xC7, 0xFB, 0x14),
  CRYPTO_MPI_LIMB_DATA4(0x33, 0x04, 0x87, 0x6D),
  CRYPTO_MPI_LIMB_DATA4(0x96, 0x84, 0x57, 0x54),
  CRYPTO_MPI_LIMB_DATA4(0x1D, 0x44, 0x0C, 0x56),
  CRYPTO_MPI_LIMB_DATA4(0xD7, 0x75, 0xAE, 0x42),
  CRYPTO_MPI_LIMB_DATA4(0x7C, 0x46, 0x65, 0x59),
  CRYPTO_MPI_LIMB_DATA4(0x0D, 0xBA, 0xE3, 0xD7),
  CRYPTO_MPI_LIMB_DATA4(0x40, 0x5C, 0x05, 0xA3),
  CRYPTO_MPI_LIMB_DATA4(0x3E, 0x03, 0x81, 0x26),
  CRYPTO_MPI_LIMB_DATA4(0x0B, 0x17, 0x9B, 0xFA),
  CRYPTO_MPI_LIMB_DATA4(0x9E, 0x58, 0x95, 0x24),
  CRYPTO_MPI_LIMB_DATA4(0x8B, 0x7F, 0x25, 0xAA),
  CRYPTO_MPI_LIMB_DATA4(0xEE, 0xBB, 0xE7, 0x5E),
  CRYPTO_MPI_LIMB_DATA4(0x72, 0xF3, 0xF0, 0xC7),
  CRYPTO_MPI_LIMB_DATA4(0xBD, 0xD4, 0xE1, 0xFD),
  CRYPTO_MPI_LIMB_DATA4(0x14, 0x42, 0xB6, 0x4F),
  CRYPTO_MPI_LIMB_DATA4(0xD3, 0x37, 0x64, 0x9D),
  CRYPTO_MPI_LIMB_DATA4(0x17, 0x94, 0xDA, 0xE8),
  CRYPTO_MPI_LIMB_DATA4(0x07, 0xB6, 0x4B, 0xA5),
  CRYPTO_MPI_LIMB_DATA4(0xC6, 0x04, 0x20, 0x6A),
  CRYPTO_MPI_LIMB_DATA4(0x23, 0x06, 0xC8, 0x6D),
  CRYPTO_MPI_LIMB_DATA4(0x7E, 0xE8, 0x23, 0xB8),
```

```
  CRYPTO_MPI_LIMB_DATA4(0x8E, 0x72, 0x88, 0xD2),
  CRYPTO_MPI_LIMB_DATA4(0x60, 0xE0, 0x41, 0x8C),
  CRYPTO_MPI_LIMB_DATA4(0x73, 0x75, 0xB8, 0x78),
  CRYPTO_MPI_LIMB_DATA4(0xE5, 0x9D, 0xB7, 0x79),
  CRYPTO_MPI_LIMB_DATA4(0x25, 0xC0, 0x30, 0xD8),
  CRYPTO_MPI_LIMB_DATA4(0xB9, 0x98, 0xA6, 0xEC),
  CRYPTO_MPI_LIMB_DATA4(0x2A, 0x10, 0xB3, 0x93),
  CRYPTO_MPI_LIMB_DATA4(0x17, 0x19, 0xE1, 0xAB),
  CRYPTO_MPI_LIMB_DATA4(0xE1, 0xF5, 0xC9, 0x56),
  CRYPTO_MPI_LIMB_DATA4(0x04, 0xD7, 0x54, 0xD0),
  CRYPTO_MPI_LIMB_DATA4(0xA5, 0x9D, 0xA5, 0x2C),
  CRYPTO_MPI_LIMB_DATA4(0x9D, 0x0A, 0x7C, 0x59),
  CRYPTO_MPI_LIMB_DATA4(0x28, 0x6D, 0xE6, 0xDF),
  CRYPTO_MPI_LIMB_DATA4(0xFA, 0x8D, 0x01, 0x39),
  CRYPTO_MPI_LIMB_DATA4(0x87, 0x0B, 0x3C, 0x52),
  CRYPTO_MPI_LIMB_DATA4(0xE6, 0x8D, 0x6E, 0xC1),
  CRYPTO_MPI_LIMB_DATA4(0x79, 0x49, 0x0B, 0xE4),
  CRYPTO_MPI_LIMB_DATA4(0xB9, 0x84, 0x58, 0x64),
  CRYPTO_MPI_LIMB_DATA4(0x25, 0xF3, 0x10, 0xEB),
  CRYPTO_MPI_LIMB_DATA4(0x41, 0x98, 0x3C, 0xF3),
  CRYPTO_MPI_LIMB_DATA4(0x57, 0x9D, 0xC4, 0x59),
  CRYPTO_MPI_LIMB_DATA4(0x6C, 0x22, 0xB3, 0xFD),
  CRYPTO_MPI_LIMB_DATA4(0x19, 0xED, 0x5D, 0xE5),
  CRYPTO_MPI_LIMB_DATA4(0x69, 0x9B, 0x12, 0x03),
  CRYPTO_MPI_LIMB_DATA4(0x36, 0xAC, 0x4F, 0xBB),
  CRYPTO_MPI_LIMB_DATA4(0xC6, 0xB1, 0x1B, 0x64),
  CRYPTO_MPI_LIMB_DATA4(0xCF, 0xE4, 0xBB, 0xFD),
  CRYPTO_MPI_LIMB_DATA4(0x04, 0xFA, 0xEF, 0x6F),
  CRYPTO_MPI_LIMB_DATA4(0xB8, 0xC3, 0xD2, 0x9E),
  CRYPTO_MPI_LIMB_DATA4(0x18, 0xD3, 0xD1, 0xC6),
  CRYPTO_MPI_LIMB_DATA4(0xB1, 0x75, 0x27, 0x42),
  CRYPTO_MPI_LIMB_DATA4(0x7A, 0xBC, 0x0C, 0xB0)
};

const CRYPTO_RSA_PRIVATE_KEY SSH_ServerKeys_RSA_Temp_2048b_PrivateKey = {
  { CRYPTO_MPI_INIT_RO(SSH_ServerKeys_RSA_Temp_2048b_PrivateKey_D_aLimbs) },
  { CRYPTO_MPI_INIT_RO(SSH_ServerKeys_RSA_Temp_2048b_PrivateKey_P_aLimbs) },
  { CRYPTO_MPI_INIT_RO(SSH_ServerKeys_RSA_Temp_2048b_PrivateKey_Q_aLimbs) },
  { CRYPTO_MPI_INIT_RO(SSH_ServerKeys_RSA_Temp_2048b_PrivateKey_DP_aLimbs) },
  { CRYPTO_MPI_INIT_RO(SSH_ServerKeys_RSA_Temp_2048b_PrivateKey_DQ_aLimbs) },
  { CRYPTO_MPI_INIT_RO(SSH_ServerKeys_RSA_Temp_2048b_PrivateKey_QInv_aLimbs) },
  { CRYPTO_MPI_INIT_RO(SSH_ServerKeys_RSA_Temp_2048b_PrivateKey_N_aLimbs) },
  { CRYPTO_MPI_INIT_RO(SSH_ServerKeys_RSA_Temp_2048b_PrivateKey_E_aLimbs) },
};

#include "CRYPTO.h"

const CRYPTO_MPI_LIMB SSH_ServerKeys_RSA_Host_2048b_PublicKey_N_aLimbs[] = {
  CRYPTO_MPI_LIMB_DATA4(0xF9, 0x64, 0x62, 0xA0),
  CRYPTO_MPI_LIMB_DATA4(0x3F, 0x52, 0xAC, 0x02),
  CRYPTO_MPI_LIMB_DATA4(0x34, 0xBE, 0x04, 0x48),
  CRYPTO_MPI_LIMB_DATA4(0xA6, 0xAD, 0x0E, 0x52),
  CRYPTO_MPI_LIMB_DATA4(0x7E, 0xE5, 0x1B, 0x16),
  CRYPTO_MPI_LIMB_DATA4(0x65, 0x7E, 0xBF, 0x93),
  CRYPTO_MPI_LIMB_DATA4(0xBD, 0xA7, 0x5E, 0xB5),
  CRYPTO_MPI_LIMB_DATA4(0x72, 0x98, 0x10, 0xA0),
  CRYPTO_MPI_LIMB_DATA4(0x9D, 0x29, 0xE4, 0x96),
  CRYPTO_MPI_LIMB_DATA4(0xCF, 0xEF, 0x11, 0xD9),
  CRYPTO_MPI_LIMB_DATA4(0x10, 0xED, 0x87, 0x07),
  CRYPTO_MPI_LIMB_DATA4(0x9F, 0x9C, 0xD2, 0x0B),
  CRYPTO_MPI_LIMB_DATA4(0x63, 0x77, 0x4A, 0x45),
  CRYPTO_MPI_LIMB_DATA4(0x26, 0x86, 0xD9, 0xC6),
  CRYPTO_MPI_LIMB_DATA4(0x38, 0x4A, 0x61, 0x0D),
  CRYPTO_MPI_LIMB_DATA4(0xF6, 0xD0, 0x6F, 0x56),
  CRYPTO_MPI_LIMB_DATA4(0x40, 0x81, 0x3E, 0xE3),
  CRYPTO_MPI_LIMB_DATA4(0xE3, 0x0E, 0xEE, 0x25),
  CRYPTO_MPI_LIMB_DATA4(0x50, 0x75, 0xCF, 0x95),
  CRYPTO_MPI_LIMB_DATA4(0x5F, 0xCC, 0xDA, 0xBA),
```

```
    CRYPTO_MPI_LIMB_DATA4(0xA9, 0xEB, 0xDC, 0x63),
    CRYPTO_MPI_LIMB_DATA4(0x1C, 0x00, 0x8A, 0x8C),
    CRYPTO_MPI_LIMB_DATA4(0x43, 0xF9, 0x1E, 0x59),
    CRYPTO_MPI_LIMB_DATA4(0xE3, 0x96, 0xB6, 0xE6),
    CRYPTO_MPI_LIMB_DATA4(0xF4, 0x30, 0xA0, 0x7D),
    CRYPTO_MPI_LIMB_DATA4(0xC5, 0xB6, 0xCC, 0x31),
    CRYPTO_MPI_LIMB_DATA4(0xB5, 0x23, 0xF1, 0x5C),
    CRYPTO_MPI_LIMB_DATA4(0x62, 0xB4, 0x05, 0xC2),
    CRYPTO_MPI_LIMB_DATA4(0xB8, 0xF2, 0x54, 0xE0),
    CRYPTO_MPI_LIMB_DATA4(0x4B, 0xA7, 0x50, 0x0B),
    CRYPTO_MPI_LIMB_DATA4(0x91, 0x57, 0x51, 0x92),
    CRYPTO_MPI_LIMB_DATA4(0x46, 0x12, 0x9D, 0x38),
    CRYPTO_MPI_LIMB_DATA4(0x55, 0x2B, 0x9F, 0xB8),
    CRYPTO_MPI_LIMB_DATA4(0x54, 0xCD, 0xE8, 0x58),
    CRYPTO_MPI_LIMB_DATA4(0x75, 0x2A, 0xAB, 0x9F),
    CRYPTO_MPI_LIMB_DATA4(0x3C, 0xE7, 0x59, 0x87),
    CRYPTO_MPI_LIMB_DATA4(0x9E, 0x9B, 0x03, 0xF5),
    CRYPTO_MPI_LIMB_DATA4(0x12, 0x56, 0xF6, 0x14),
    CRYPTO_MPI_LIMB_DATA4(0x61, 0x3F, 0xDC, 0xCD),
    CRYPTO_MPI_LIMB_DATA4(0x06, 0xED, 0x21, 0x14),
    CRYPTO_MPI_LIMB_DATA4(0x55, 0x2F, 0x85, 0x78),
    CRYPTO_MPI_LIMB_DATA4(0x07, 0xDA, 0xE4, 0xB2),
    CRYPTO_MPI_LIMB_DATA4(0x0E, 0x4A, 0x69, 0x91),
    CRYPTO_MPI_LIMB_DATA4(0xAD, 0xB5, 0x6E, 0xB1),
    CRYPTO_MPI_LIMB_DATA4(0x14, 0x8F, 0x71, 0x22),
    CRYPTO_MPI_LIMB_DATA4(0xF1, 0x45, 0x61, 0x7C),
    CRYPTO_MPI_LIMB_DATA4(0x18, 0x18, 0x0A, 0x1F),
    CRYPTO_MPI_LIMB_DATA4(0x1D, 0xDD, 0xAD, 0x34),
    CRYPTO_MPI_LIMB_DATA4(0xC4, 0x18, 0x77, 0x02),
    CRYPTO_MPI_LIMB_DATA4(0x34, 0xF2, 0x27, 0x34),
    CRYPTO_MPI_LIMB_DATA4(0x77, 0x7C, 0xC0, 0xC2),
    CRYPTO_MPI_LIMB_DATA4(0x6D, 0x7F, 0xA3, 0x68),
    CRYPTO_MPI_LIMB_DATA4(0x53, 0x0E, 0x28, 0x5F),
    CRYPTO_MPI_LIMB_DATA4(0x91, 0xAC, 0x96, 0x77),
    CRYPTO_MPI_LIMB_DATA4(0x98, 0x17, 0xAE, 0x0D),
    CRYPTO_MPI_LIMB_DATA4(0x53, 0xE1, 0x73, 0x84),
    CRYPTO_MPI_LIMB_DATA4(0x5F, 0xD6, 0xFA, 0x6E),
    CRYPTO_MPI_LIMB_DATA4(0x40, 0x8D, 0x84, 0x6A),
    CRYPTO_MPI_LIMB_DATA4(0x68, 0xE1, 0x2E, 0x2A),
    CRYPTO_MPI_LIMB_DATA4(0xF8, 0x71, 0xAA, 0xA7),
    CRYPTO_MPI_LIMB_DATA4(0x6F, 0x41, 0x0C, 0x2F),
    CRYPTO_MPI_LIMB_DATA4(0x2C, 0x33, 0x47, 0x2B),
    CRYPTO_MPI_LIMB_DATA4(0xDA, 0x83, 0x99, 0xAF),
    CRYPTO_MPI_LIMB_DATA4(0xF8, 0x23, 0x7B, 0xE0)
};

const CRYPTO_MPI_LIMB SSH_ServerKeys_RSA_Host_2048b_PublicKey_E_aLimbs[] = {
    CRYPTO_MPI_LIMB_DATA3(0x01, 0x00, 0x01)
};

const CRYPTO_RSA_PUBLIC_KEY SSH_ServerKeys_RSA_Host_2048b_PublicKey = {
    { CRYPTO_MPI_INIT_RO(SSH_ServerKeys_RSA_Host_2048b_PublicKey_N_aLimbs) },
    { CRYPTO_MPI_INIT_RO(SSH_ServerKeys_RSA_Host_2048b_PublicKey_E_aLimbs) },
};

const CRYPTO_MPI_LIMB SSH_ServerKeys_RSA_Host_2048b_PrivateKey_D_aLimbs[] = {
    CRYPTO_MPI_LIMB_DATA4(0x05, 0x4C, 0x7F, 0x0F),
    CRYPTO_MPI_LIMB_DATA4(0xEB, 0xFA, 0x66, 0xF1),
    CRYPTO_MPI_LIMB_DATA4(0x13, 0xC7, 0xB9, 0x05),
    CRYPTO_MPI_LIMB_DATA4(0xEE, 0xC1, 0x05, 0xC7),
    CRYPTO_MPI_LIMB_DATA4(0x9E, 0x46, 0xB3, 0x26),
    CRYPTO_MPI_LIMB_DATA4(0x1D, 0xFA, 0xB0, 0xB6),
    CRYPTO_MPI_LIMB_DATA4(0xFE, 0x36, 0xAD, 0x48),
    CRYPTO_MPI_LIMB_DATA4(0xB3, 0x49, 0xAB, 0xA5),
    CRYPTO_MPI_LIMB_DATA4(0xB7, 0x19, 0x0C, 0x62),
    CRYPTO_MPI_LIMB_DATA4(0xE1, 0xAC, 0x5A, 0xE1),
    CRYPTO_MPI_LIMB_DATA4(0xE5, 0xF9, 0x83, 0x20),
    CRYPTO_MPI_LIMB_DATA4(0x6A, 0x4D, 0xF1, 0x85),
```

```
  CRYPTO_MPI_LIMB_DATA4(0x92, 0x08, 0x6B, 0xA9),
  CRYPTO_MPI_LIMB_DATA4(0xDF, 0xE3, 0xDE, 0xCB),
  CRYPTO_MPI_LIMB_DATA4(0xC2, 0x58, 0x22, 0xFE),
  CRYPTO_MPI_LIMB_DATA4(0xC6, 0x64, 0xD4, 0x8C),
  CRYPTO_MPI_LIMB_DATA4(0xA4, 0x2F, 0xEF, 0xCF),
  CRYPTO_MPI_LIMB_DATA4(0x47, 0x68, 0x1C, 0x5F),
  CRYPTO_MPI_LIMB_DATA4(0x2A, 0xBB, 0x49, 0xB0),
  CRYPTO_MPI_LIMB_DATA4(0x98, 0x10, 0x58, 0x20),
  CRYPTO_MPI_LIMB_DATA4(0x35, 0x39, 0xAE, 0xAF),
  CRYPTO_MPI_LIMB_DATA4(0xFE, 0x0A, 0xCD, 0xBB),
  CRYPTO_MPI_LIMB_DATA4(0xC1, 0xAA, 0xB3, 0x79),
  CRYPTO_MPI_LIMB_DATA4(0xEA, 0x52, 0x3E, 0x1C),
  CRYPTO_MPI_LIMB_DATA4(0x79, 0xEF, 0x80, 0xEC),
  CRYPTO_MPI_LIMB_DATA4(0x50, 0x14, 0x2F, 0x51),
  CRYPTO_MPI_LIMB_DATA4(0x37, 0x04, 0xFD, 0xC6),
  CRYPTO_MPI_LIMB_DATA4(0x51, 0xC2, 0xFD, 0xCF),
  CRYPTO_MPI_LIMB_DATA4(0xF5, 0x61, 0x8B, 0x95),
  CRYPTO_MPI_LIMB_DATA4(0xD2, 0x22, 0x2A, 0x10),
  CRYPTO_MPI_LIMB_DATA4(0xB9, 0x75, 0xEB, 0x6E),
  CRYPTO_MPI_LIMB_DATA4(0x9F, 0xC4, 0xF4, 0x97),
  CRYPTO_MPI_LIMB_DATA4(0x98, 0x75, 0xC5, 0xA9),
  CRYPTO_MPI_LIMB_DATA4(0x25, 0x59, 0x29, 0x63),
  CRYPTO_MPI_LIMB_DATA4(0x9F, 0xF7, 0xA9, 0x70),
  CRYPTO_MPI_LIMB_DATA4(0x87, 0xC8, 0xB2, 0xEA),
  CRYPTO_MPI_LIMB_DATA4(0xD2, 0xEF, 0xBC, 0xF0),
  CRYPTO_MPI_LIMB_DATA4(0xEE, 0xB7, 0x60, 0x00),
  CRYPTO_MPI_LIMB_DATA4(0xCD, 0x52, 0x7D, 0xC5),
  CRYPTO_MPI_LIMB_DATA4(0xEE, 0x42, 0x36, 0x70),
  CRYPTO_MPI_LIMB_DATA4(0xAD, 0xA5, 0x4E, 0xD3),
  CRYPTO_MPI_LIMB_DATA4(0x6E, 0x0E, 0xDD, 0xE0),
  CRYPTO_MPI_LIMB_DATA4(0xB8, 0xFF, 0x87, 0x90),
  CRYPTO_MPI_LIMB_DATA4(0xBD, 0xC4, 0xA2, 0x35),
  CRYPTO_MPI_LIMB_DATA4(0xB6, 0x2A, 0x7E, 0x99),
  CRYPTO_MPI_LIMB_DATA4(0xE7, 0xC6, 0xFC, 0xD0),
  CRYPTO_MPI_LIMB_DATA4(0x44, 0xE0, 0x35, 0x44),
  CRYPTO_MPI_LIMB_DATA4(0x35, 0xB4, 0x2A, 0xEC),
  CRYPTO_MPI_LIMB_DATA4(0xA7, 0x29, 0xCC, 0x76),
  CRYPTO_MPI_LIMB_DATA4(0x35, 0xC7, 0xEA, 0x20),
  CRYPTO_MPI_LIMB_DATA4(0xF9, 0x03, 0x04, 0xBD),
  CRYPTO_MPI_LIMB_DATA4(0xAE, 0xCB, 0x0A, 0x2E),
  CRYPTO_MPI_LIMB_DATA4(0x5E, 0x27, 0x7F, 0xAB),
  CRYPTO_MPI_LIMB_DATA4(0x19, 0xBF, 0x3B, 0x1C),
  CRYPTO_MPI_LIMB_DATA4(0x8F, 0x87, 0x33, 0xDD),
  CRYPTO_MPI_LIMB_DATA4(0x86, 0xD4, 0x32, 0xA6),
  CRYPTO_MPI_LIMB_DATA4(0x68, 0x8C, 0xE2, 0x77),
  CRYPTO_MPI_LIMB_DATA4(0xC6, 0x8B, 0x3F, 0xB2),
  CRYPTO_MPI_LIMB_DATA4(0x32, 0x42, 0xF5, 0x5C),
  CRYPTO_MPI_LIMB_DATA4(0xB4, 0x45, 0x89, 0xA7),
  CRYPTO_MPI_LIMB_DATA4(0xF3, 0xE7, 0xA7, 0xC7),
  CRYPTO_MPI_LIMB_DATA4(0xEE, 0x52, 0xB7, 0x9E),
  CRYPTO_MPI_LIMB_DATA4(0x72, 0x7D, 0x50, 0x4F),
  CRYPTO_MPI_LIMB_DATA4(0x48, 0x9D, 0xDF, 0x0F)
};

const CRYPTO_MPI_LIMB SSH_ServerKeys_RSA_Host_2048b_PrivateKey_P_aLimbs[] = {
  CRYPTO_MPI_LIMB_DATA4(0xE5, 0xE9, 0x80, 0x67),
  CRYPTO_MPI_LIMB_DATA4(0xEC, 0x9C, 0x10, 0x2A),
  CRYPTO_MPI_LIMB_DATA4(0x8C, 0xE9, 0xBE, 0xD8),
  CRYPTO_MPI_LIMB_DATA4(0xC5, 0xB1, 0x0D, 0x00),
  CRYPTO_MPI_LIMB_DATA4(0x4D, 0x0F, 0x17, 0xBA),
  CRYPTO_MPI_LIMB_DATA4(0x71, 0xE4, 0x25, 0xBF),
  CRYPTO_MPI_LIMB_DATA4(0x2C, 0x7E, 0x46, 0xAD),
  CRYPTO_MPI_LIMB_DATA4(0x27, 0xB5, 0x5D, 0x24),
  CRYPTO_MPI_LIMB_DATA4(0x61, 0xBF, 0xED, 0xF2),
  CRYPTO_MPI_LIMB_DATA4(0xA4, 0x5A, 0x1D, 0x3D),
  CRYPTO_MPI_LIMB_DATA4(0x20, 0x35, 0xB0, 0x80),
  CRYPTO_MPI_LIMB_DATA4(0x5C, 0xDD, 0xD6, 0x72),
  CRYPTO_MPI_LIMB_DATA4(0x31, 0xCE, 0xD5, 0x6F),
```

```
  CRYPTO_MPI_LIMB_DATA4(0x65, 0xAE, 0x7A, 0x07),
  CRYPTO_MPI_LIMB_DATA4(0x7E, 0x2B, 0xE7, 0x34),
  CRYPTO_MPI_LIMB_DATA4(0xD1, 0xAD, 0x75, 0xDF),
  CRYPTO_MPI_LIMB_DATA4(0x58, 0x0A, 0x23, 0x45),
  CRYPTO_MPI_LIMB_DATA4(0xA2, 0xC2, 0x88, 0x8B),
  CRYPTO_MPI_LIMB_DATA4(0x02, 0xBC, 0xD0, 0x8B),
  CRYPTO_MPI_LIMB_DATA4(0x3E, 0x4A, 0x90, 0xF9),
  CRYPTO_MPI_LIMB_DATA4(0xBE, 0xF7, 0xB4, 0xD4),
  CRYPTO_MPI_LIMB_DATA4(0x0B, 0x62, 0xBB, 0xDD),
  CRYPTO_MPI_LIMB_DATA4(0x7F, 0x8F, 0x3E, 0x68),
  CRYPTO_MPI_LIMB_DATA4(0x75, 0x60, 0xC7, 0x05),
  CRYPTO_MPI_LIMB_DATA4(0xDC, 0xDD, 0xF8, 0xFA),
  CRYPTO_MPI_LIMB_DATA4(0x02, 0xE1, 0xF4, 0xBE),
  CRYPTO_MPI_LIMB_DATA4(0xE0, 0xFE, 0x32, 0x76),
  CRYPTO_MPI_LIMB_DATA4(0xE3, 0xAC, 0xCC, 0xCE),
  CRYPTO_MPI_LIMB_DATA4(0x38, 0x35, 0x94, 0x38),
  CRYPTO_MPI_LIMB_DATA4(0x9E, 0xF1, 0xCB, 0x1E),
  CRYPTO_MPI_LIMB_DATA4(0x0F, 0x3B, 0xB5, 0x4D),
  CRYPTO_MPI_LIMB_DATA4(0x61, 0xC7, 0xF7, 0xE8)
};

const CRYPTO_MPI_LIMB SSH_ServerKeys_RSA_Host_2048b_PrivateKey_Q_aLimbs[] = {
  CRYPTO_MPI_LIMB_DATA4(0x85, 0x4D, 0xE3, 0x16),
  CRYPTO_MPI_LIMB_DATA4(0x27, 0x37, 0xC3, 0xAA),
  CRYPTO_MPI_LIMB_DATA4(0xD3, 0xA5, 0x0D, 0x78),
  CRYPTO_MPI_LIMB_DATA4(0xFD, 0xEE, 0x66, 0x68),
  CRYPTO_MPI_LIMB_DATA4(0x89, 0xDD, 0x08, 0x6B),
  CRYPTO_MPI_LIMB_DATA4(0x9A, 0x76, 0x48, 0x84),
  CRYPTO_MPI_LIMB_DATA4(0xA1, 0x12, 0xDD, 0xFF),
  CRYPTO_MPI_LIMB_DATA4(0x4C, 0xCB, 0xA9, 0x08),
  CRYPTO_MPI_LIMB_DATA4(0x8A, 0x85, 0x6E, 0xFC),
  CRYPTO_MPI_LIMB_DATA4(0xCE, 0x9D, 0xE2, 0x17),
  CRYPTO_MPI_LIMB_DATA4(0x61, 0x2A, 0xC2, 0xDD),
  CRYPTO_MPI_LIMB_DATA4(0x24, 0x8B, 0xCB, 0xA5),
  CRYPTO_MPI_LIMB_DATA4(0xDA, 0x56, 0xA8, 0x5F),
  CRYPTO_MPI_LIMB_DATA4(0xCF, 0x55, 0x0D, 0xB6),
  CRYPTO_MPI_LIMB_DATA4(0x76, 0xEA, 0x8D, 0x2B),
  CRYPTO_MPI_LIMB_DATA4(0xC8, 0x15, 0x44, 0x6E),
  CRYPTO_MPI_LIMB_DATA4(0x82, 0xC8, 0x57, 0xC0),
  CRYPTO_MPI_LIMB_DATA4(0x6E, 0x6F, 0x81, 0x7F),
  CRYPTO_MPI_LIMB_DATA4(0xCD, 0x93, 0x11, 0x7F),
  CRYPTO_MPI_LIMB_DATA4(0x92, 0x9F, 0xCB, 0xE0),
  CRYPTO_MPI_LIMB_DATA4(0x7C, 0x71, 0xE6, 0xD2),
  CRYPTO_MPI_LIMB_DATA4(0xBB, 0xA2, 0x75, 0x3E),
  CRYPTO_MPI_LIMB_DATA4(0x6F, 0x71, 0x07, 0xB5),
  CRYPTO_MPI_LIMB_DATA4(0x9A, 0xBA, 0xD9, 0x9F),
  CRYPTO_MPI_LIMB_DATA4(0xEB, 0x40, 0x83, 0xA6),
  CRYPTO_MPI_LIMB_DATA4(0x36, 0x81, 0x31, 0x78),
  CRYPTO_MPI_LIMB_DATA4(0x3F, 0xC2, 0x26, 0xC3),
  CRYPTO_MPI_LIMB_DATA4(0xE4, 0x5A, 0xAC, 0x4B),
  CRYPTO_MPI_LIMB_DATA4(0xA5, 0x25, 0x57, 0xA7),
  CRYPTO_MPI_LIMB_DATA4(0x63, 0xBA, 0x7C, 0xF2),
  CRYPTO_MPI_LIMB_DATA4(0xCC, 0x4C, 0x99, 0x95),
  CRYPTO_MPI_LIMB_DATA4(0xF1, 0x90, 0xAC, 0xF6)
};

const CRYPTO_MPI_LIMB SSH_ServerKeys_RSA_Host_2048b_PrivateKey_DP_aLimbs[] = {
  CRYPTO_MPI_LIMB_DATA4(0xF1, 0x48, 0xC3, 0x4B),
  CRYPTO_MPI_LIMB_DATA4(0x36, 0x90, 0x2B, 0xDC),
  CRYPTO_MPI_LIMB_DATA4(0x28, 0x4E, 0x0A, 0x3F),
  CRYPTO_MPI_LIMB_DATA4(0xF0, 0xCF, 0x78, 0x96),
  CRYPTO_MPI_LIMB_DATA4(0xFE, 0x26, 0x42, 0x63),
  CRYPTO_MPI_LIMB_DATA4(0x03, 0x39, 0x34, 0x36),
  CRYPTO_MPI_LIMB_DATA4(0xBB, 0xD1, 0xF5, 0xC8),
  CRYPTO_MPI_LIMB_DATA4(0x8C, 0x0A, 0x6D, 0xE7),
  CRYPTO_MPI_LIMB_DATA4(0x74, 0xDA, 0xC7, 0x87),
  CRYPTO_MPI_LIMB_DATA4(0x62, 0xA0, 0x3E, 0x7B),
  CRYPTO_MPI_LIMB_DATA4(0x21, 0x27, 0x58, 0x88),
```

```
    CRYPTO_MPI_LIMB_DATA4(0xD8, 0xA4, 0xD5, 0xB0),
    CRYPTO_MPI_LIMB_DATA4(0x1F, 0x01, 0xD2, 0x1E),
    CRYPTO_MPI_LIMB_DATA4(0x53, 0x5A, 0x32, 0x8A),
    CRYPTO_MPI_LIMB_DATA4(0xCF, 0x13, 0xA9, 0x0C),
    CRYPTO_MPI_LIMB_DATA4(0x3D, 0x07, 0x41, 0x53),
    CRYPTO_MPI_LIMB_DATA4(0x64, 0xC0, 0x0C, 0xF7),
    CRYPTO_MPI_LIMB_DATA4(0x38, 0x56, 0xAD, 0xB6),
    CRYPTO_MPI_LIMB_DATA4(0xB2, 0xF7, 0x8F, 0xFF),
    CRYPTO_MPI_LIMB_DATA4(0x9B, 0x29, 0xF2, 0x2F),
    CRYPTO_MPI_LIMB_DATA4(0x96, 0xCE, 0x56, 0x07),
    CRYPTO_MPI_LIMB_DATA4(0xEF, 0x89, 0x23, 0x9F),
    CRYPTO_MPI_LIMB_DATA4(0x55, 0x76, 0xE7, 0xFD),
    CRYPTO_MPI_LIMB_DATA4(0xAB, 0x3E, 0x64, 0x46),
    CRYPTO_MPI_LIMB_DATA4(0x7B, 0x0E, 0x55, 0x75),
    CRYPTO_MPI_LIMB_DATA4(0xCF, 0xA1, 0x30, 0x00),
    CRYPTO_MPI_LIMB_DATA4(0x46, 0xDF, 0x5E, 0x7C),
    CRYPTO_MPI_LIMB_DATA4(0xDA, 0xAD, 0x61, 0x3C),
    CRYPTO_MPI_LIMB_DATA4(0xE5, 0xD9, 0x98, 0xA6),
    CRYPTO_MPI_LIMB_DATA4(0xC8, 0x6B, 0x8E, 0xA2),
    CRYPTO_MPI_LIMB_DATA4(0xB8, 0x67, 0x3D, 0xE2),
    CRYPTO_MPI_LIMB_DATA4(0x0E, 0xE2, 0xEE, 0xC1)
};

const CRYPTO_MPI_LIMB SSH_ServerKeys_RSA_Host_2048b_PrivateKey_DQ_aLimbs[] = {
    CRYPTO_MPI_LIMB_DATA4(0x4D, 0xE7, 0xF2, 0x77),
    CRYPTO_MPI_LIMB_DATA4(0x44, 0xDA, 0xD1, 0x43),
    CRYPTO_MPI_LIMB_DATA4(0x8B, 0x59, 0xE0, 0xB2),
    CRYPTO_MPI_LIMB_DATA4(0x76, 0xA5, 0x64, 0x50),
    CRYPTO_MPI_LIMB_DATA4(0x16, 0xEB, 0xBE, 0x96),
    CRYPTO_MPI_LIMB_DATA4(0x6B, 0xC9, 0xEB, 0x80),
    CRYPTO_MPI_LIMB_DATA4(0xCD, 0x1A, 0x61, 0x4B),
    CRYPTO_MPI_LIMB_DATA4(0x6B, 0x04, 0x74, 0x1C),
    CRYPTO_MPI_LIMB_DATA4(0xAB, 0xC7, 0xC6, 0xF7),
    CRYPTO_MPI_LIMB_DATA4(0xE0, 0xA1, 0x0A, 0x3B),
    CRYPTO_MPI_LIMB_DATA4(0x08, 0x9B, 0x7A, 0xDA),
    CRYPTO_MPI_LIMB_DATA4(0xFF, 0x86, 0x2C, 0x9F),
    CRYPTO_MPI_LIMB_DATA4(0x93, 0xDF, 0x40, 0xA5),
    CRYPTO_MPI_LIMB_DATA4(0x84, 0xBA, 0x9D, 0x93),
    CRYPTO_MPI_LIMB_DATA4(0x25, 0x12, 0x9B, 0x11),
    CRYPTO_MPI_LIMB_DATA4(0x75, 0x5E, 0xB3, 0x35),
    CRYPTO_MPI_LIMB_DATA4(0xAA, 0x38, 0x06, 0xAC),
    CRYPTO_MPI_LIMB_DATA4(0x44, 0x8E, 0xC0, 0x87),
    CRYPTO_MPI_LIMB_DATA4(0x49, 0xA0, 0x75, 0x82),
    CRYPTO_MPI_LIMB_DATA4(0x31, 0x21, 0x2E, 0x4C),
    CRYPTO_MPI_LIMB_DATA4(0xC0, 0x9D, 0x83, 0x29),
    CRYPTO_MPI_LIMB_DATA4(0x40, 0x1E, 0xF3, 0xEF),
    CRYPTO_MPI_LIMB_DATA4(0x9E, 0xD6, 0xB3, 0xFA),
    CRYPTO_MPI_LIMB_DATA4(0xEB, 0x07, 0x72, 0x3A),
    CRYPTO_MPI_LIMB_DATA4(0x63, 0x09, 0xEA, 0xDE),
    CRYPTO_MPI_LIMB_DATA4(0x61, 0xEA, 0x18, 0x04),
    CRYPTO_MPI_LIMB_DATA4(0x6F, 0xE5, 0x17, 0x37),
    CRYPTO_MPI_LIMB_DATA4(0x8F, 0x25, 0x4B, 0x9B),
    CRYPTO_MPI_LIMB_DATA4(0x0B, 0x6F, 0x36, 0xC9),
    CRYPTO_MPI_LIMB_DATA4(0xCC, 0x83, 0x79, 0x8D),
    CRYPTO_MPI_LIMB_DATA4(0xC3, 0xCA, 0x86, 0xEB),
    CRYPTO_MPI_LIMB_DATA4(0x63, 0x17, 0xDE, 0x4A)
};

const CRYPTO_MPI_LIMB SSH_ServerKeys_RSA_Host_2048b_PrivateKey_QInv_aLimbs[] = {
    CRYPTO_MPI_LIMB_DATA4(0x6B, 0x7E, 0xBF, 0x8B),
    CRYPTO_MPI_LIMB_DATA4(0x1D, 0xC2, 0x18, 0x8E),
    CRYPTO_MPI_LIMB_DATA4(0xCB, 0xC0, 0xB4, 0x6B),
    CRYPTO_MPI_LIMB_DATA4(0x19, 0x2D, 0xAF, 0x33),
    CRYPTO_MPI_LIMB_DATA4(0x71, 0x4E, 0x9D, 0x30),
    CRYPTO_MPI_LIMB_DATA4(0x14, 0x14, 0xAD, 0x6C),
    CRYPTO_MPI_LIMB_DATA4(0xA4, 0xCE, 0x06, 0xD0),
    CRYPTO_MPI_LIMB_DATA4(0x9F, 0xCA, 0x15, 0x29),
    CRYPTO_MPI_LIMB_DATA4(0x64, 0x23, 0xB0, 0x75),
```

```
    CRYPTO_MPI_LIMB_DATA4(0x8C, 0x4D, 0x28, 0x43),
    CRYPTO_MPI_LIMB_DATA4(0xEF, 0x3C, 0x27, 0x71),
    CRYPTO_MPI_LIMB_DATA4(0xBA, 0x6C, 0x67, 0x5B),
    CRYPTO_MPI_LIMB_DATA4(0xC5, 0x75, 0x70, 0xD4),
    CRYPTO_MPI_LIMB_DATA4(0xBF, 0x43, 0xF7, 0x95),
    CRYPTO_MPI_LIMB_DATA4(0x89, 0xD0, 0xA4, 0xE5),
    CRYPTO_MPI_LIMB_DATA4(0x14, 0xB3, 0x59, 0xE9),
    CRYPTO_MPI_LIMB_DATA4(0x27, 0x75, 0xB3, 0x0C),
    CRYPTO_MPI_LIMB_DATA4(0x29, 0x67, 0x83, 0x15),
    CRYPTO_MPI_LIMB_DATA4(0x61, 0x47, 0xBB, 0x40),
    CRYPTO_MPI_LIMB_DATA4(0x0F, 0xF1, 0x15, 0x2A),
    CRYPTO_MPI_LIMB_DATA4(0x52, 0x0E, 0x83, 0x8F),
    CRYPTO_MPI_LIMB_DATA4(0x08, 0xAE, 0xF0, 0xF4),
    CRYPTO_MPI_LIMB_DATA4(0x5B, 0xAD, 0x7A, 0x4C),
    CRYPTO_MPI_LIMB_DATA4(0xA8, 0xFE, 0x83, 0xC2),
    CRYPTO_MPI_LIMB_DATA4(0x26, 0xFE, 0xDD, 0x89),
    CRYPTO_MPI_LIMB_DATA4(0xBA, 0x42, 0xCB, 0x19),
    CRYPTO_MPI_LIMB_DATA4(0x79, 0x12, 0x0D, 0x6C),
    CRYPTO_MPI_LIMB_DATA4(0x2A, 0x0B, 0x32, 0xE7),
    CRYPTO_MPI_LIMB_DATA4(0xDB, 0xCA, 0x57, 0xA8),
    CRYPTO_MPI_LIMB_DATA4(0xE7, 0x95, 0x7D, 0x1F),
    CRYPTO_MPI_LIMB_DATA4(0xF8, 0x7F, 0x03, 0x4B),
    CRYPTO_MPI_LIMB_DATA4(0xC2, 0x34, 0xB6, 0xAB)
};

const CRYPTO_MPI_LIMB SSH_ServerKeys_RSA_Host_2048b_PrivateKey_N_aLimbs[] = {
    CRYPTO_MPI_LIMB_DATA4(0xF9, 0x64, 0x62, 0xA0),
    CRYPTO_MPI_LIMB_DATA4(0x3F, 0x52, 0xAC, 0x02),
    CRYPTO_MPI_LIMB_DATA4(0x34, 0xBE, 0x04, 0x48),
    CRYPTO_MPI_LIMB_DATA4(0xA6, 0xAD, 0x0E, 0x52),
    CRYPTO_MPI_LIMB_DATA4(0x7E, 0xE5, 0x1B, 0x16),
    CRYPTO_MPI_LIMB_DATA4(0x65, 0x7E, 0xBF, 0x93),
    CRYPTO_MPI_LIMB_DATA4(0xBD, 0xA7, 0x5E, 0xB5),
    CRYPTO_MPI_LIMB_DATA4(0x72, 0x98, 0x10, 0xA0),
    CRYPTO_MPI_LIMB_DATA4(0x9D, 0x29, 0xE4, 0x96),
    CRYPTO_MPI_LIMB_DATA4(0xCF, 0xEF, 0x11, 0xD9),
    CRYPTO_MPI_LIMB_DATA4(0x10, 0xED, 0x87, 0x07),
    CRYPTO_MPI_LIMB_DATA4(0x9F, 0x9C, 0xD2, 0x0B),
    CRYPTO_MPI_LIMB_DATA4(0x63, 0x77, 0x4A, 0x45),
    CRYPTO_MPI_LIMB_DATA4(0x26, 0x86, 0xD9, 0xC6),
    CRYPTO_MPI_LIMB_DATA4(0x38, 0x4A, 0x61, 0x0D),
    CRYPTO_MPI_LIMB_DATA4(0xF6, 0xD0, 0x6F, 0x56),
    CRYPTO_MPI_LIMB_DATA4(0x40, 0x81, 0x3E, 0xE3),
    CRYPTO_MPI_LIMB_DATA4(0xE3, 0x0E, 0xEE, 0x25),
    CRYPTO_MPI_LIMB_DATA4(0x50, 0x75, 0xCF, 0x95),
    CRYPTO_MPI_LIMB_DATA4(0x5F, 0xCC, 0xDA, 0xBA),
    CRYPTO_MPI_LIMB_DATA4(0xA9, 0xEB, 0xDC, 0x63),
    CRYPTO_MPI_LIMB_DATA4(0x1C, 0x00, 0x8A, 0x8C),
    CRYPTO_MPI_LIMB_DATA4(0x43, 0xF9, 0x1E, 0x59),
    CRYPTO_MPI_LIMB_DATA4(0xE3, 0x96, 0xB6, 0xE6),
    CRYPTO_MPI_LIMB_DATA4(0xF4, 0x30, 0xA0, 0x7D),
    CRYPTO_MPI_LIMB_DATA4(0xC5, 0xB6, 0xCC, 0x31),
    CRYPTO_MPI_LIMB_DATA4(0xB5, 0x23, 0xF1, 0x5C),
    CRYPTO_MPI_LIMB_DATA4(0x62, 0xB4, 0x05, 0xC2),
    CRYPTO_MPI_LIMB_DATA4(0xB8, 0xF2, 0x54, 0xE0),
    CRYPTO_MPI_LIMB_DATA4(0x4B, 0xA7, 0x50, 0x0B),
    CRYPTO_MPI_LIMB_DATA4(0x91, 0x57, 0x51, 0x92),
    CRYPTO_MPI_LIMB_DATA4(0x46, 0x12, 0x9D, 0x38),
    CRYPTO_MPI_LIMB_DATA4(0x55, 0x2B, 0x9F, 0xB8),
    CRYPTO_MPI_LIMB_DATA4(0x54, 0xCD, 0xE8, 0x58),
    CRYPTO_MPI_LIMB_DATA4(0x75, 0x2A, 0xAB, 0x9F),
    CRYPTO_MPI_LIMB_DATA4(0x3C, 0xE7, 0x59, 0x87),
    CRYPTO_MPI_LIMB_DATA4(0x9E, 0x9B, 0x03, 0xF5),
    CRYPTO_MPI_LIMB_DATA4(0x12, 0x56, 0xF6, 0x14),
    CRYPTO_MPI_LIMB_DATA4(0x61, 0x3F, 0xDC, 0xCD),
    CRYPTO_MPI_LIMB_DATA4(0x06, 0xED, 0x21, 0x14),
    CRYPTO_MPI_LIMB_DATA4(0x55, 0x2F, 0x85, 0x78),
    CRYPTO_MPI_LIMB_DATA4(0x07, 0xDA, 0xE4, 0xB2),
```

```
  CRYPTO_MPI_LIMB_DATA4(0x0E, 0x4A, 0x69, 0x91),
  CRYPTO_MPI_LIMB_DATA4(0xAD, 0xB5, 0x6E, 0xB1),
  CRYPTO_MPI_LIMB_DATA4(0x14, 0x8F, 0x71, 0x22),
  CRYPTO_MPI_LIMB_DATA4(0xF1, 0x45, 0x61, 0x7C),
  CRYPTO_MPI_LIMB_DATA4(0x18, 0x18, 0x0A, 0x1F),
  CRYPTO_MPI_LIMB_DATA4(0x1D, 0xDD, 0xAD, 0x34),
  CRYPTO_MPI_LIMB_DATA4(0xC4, 0x18, 0x77, 0x02),
  CRYPTO_MPI_LIMB_DATA4(0x34, 0xF2, 0x27, 0x34),
  CRYPTO_MPI_LIMB_DATA4(0x77, 0x7C, 0xC0, 0xC2),
  CRYPTO_MPI_LIMB_DATA4(0x6D, 0x7F, 0xA3, 0x68),
  CRYPTO_MPI_LIMB_DATA4(0x53, 0x0E, 0x28, 0x5F),
  CRYPTO_MPI_LIMB_DATA4(0x91, 0xAC, 0x96, 0x77),
  CRYPTO_MPI_LIMB_DATA4(0x98, 0x17, 0xAE, 0x0D),
  CRYPTO_MPI_LIMB_DATA4(0x53, 0xE1, 0x73, 0x84),
  CRYPTO_MPI_LIMB_DATA4(0x5F, 0xD6, 0xFA, 0x6E),
  CRYPTO_MPI_LIMB_DATA4(0x40, 0x8D, 0x84, 0x6A),
  CRYPTO_MPI_LIMB_DATA4(0x68, 0xE1, 0x2E, 0x2A),
  CRYPTO_MPI_LIMB_DATA4(0xF8, 0x71, 0xAA, 0xA7),
  CRYPTO_MPI_LIMB_DATA4(0x6F, 0x41, 0x0C, 0x2F),
  CRYPTO_MPI_LIMB_DATA4(0x2C, 0x33, 0x47, 0x2B),
  CRYPTO_MPI_LIMB_DATA4(0xDA, 0x83, 0x99, 0xAF),
  CRYPTO_MPI_LIMB_DATA4(0xF8, 0x23, 0x7B, 0xE0)
};

const CRYPTO_MPI_LIMB SSH_ServerKeys_RSA_Host_2048b_PrivateKey_E_aLimbs[] = {
  CRYPTO_MPI_LIMB_DATA4(0xF9, 0x64, 0x62, 0xA0),
  CRYPTO_MPI_LIMB_DATA4(0x3F, 0x52, 0xAC, 0x02),
  CRYPTO_MPI_LIMB_DATA4(0x34, 0xBE, 0x04, 0x48),
  CRYPTO_MPI_LIMB_DATA4(0xA6, 0xAD, 0x0E, 0x52),
  CRYPTO_MPI_LIMB_DATA4(0x7E, 0xE5, 0x1B, 0x16),
  CRYPTO_MPI_LIMB_DATA4(0x65, 0x7E, 0xBF, 0x93),
  CRYPTO_MPI_LIMB_DATA4(0xBD, 0xA7, 0x5E, 0xB5),
  CRYPTO_MPI_LIMB_DATA4(0x72, 0x98, 0x10, 0xA0),
  CRYPTO_MPI_LIMB_DATA4(0x9D, 0x29, 0xE4, 0x96),
  CRYPTO_MPI_LIMB_DATA4(0xCF, 0xEF, 0x11, 0xD9),
  CRYPTO_MPI_LIMB_DATA4(0x10, 0xED, 0x87, 0x07),
  CRYPTO_MPI_LIMB_DATA4(0x9F, 0x9C, 0xD2, 0x0B),
  CRYPTO_MPI_LIMB_DATA4(0x63, 0x77, 0x4A, 0x45),
  CRYPTO_MPI_LIMB_DATA4(0x26, 0x86, 0xD9, 0xC6),
  CRYPTO_MPI_LIMB_DATA4(0x38, 0x4A, 0x61, 0x0D),
  CRYPTO_MPI_LIMB_DATA4(0xF6, 0xD0, 0x6F, 0x56),
  CRYPTO_MPI_LIMB_DATA4(0x40, 0x81, 0x3E, 0xE3),
  CRYPTO_MPI_LIMB_DATA4(0xE3, 0x0E, 0xEE, 0x25),
  CRYPTO_MPI_LIMB_DATA4(0x50, 0x75, 0xCF, 0x95),
  CRYPTO_MPI_LIMB_DATA4(0x5F, 0xCC, 0xDA, 0xBA),
  CRYPTO_MPI_LIMB_DATA4(0xA9, 0xEB, 0xDC, 0x63),
  CRYPTO_MPI_LIMB_DATA4(0x1C, 0x00, 0x8A, 0x8C),
  CRYPTO_MPI_LIMB_DATA4(0x43, 0xF9, 0x1E, 0x59),
  CRYPTO_MPI_LIMB_DATA4(0xE3, 0x96, 0xB6, 0xE6),
  CRYPTO_MPI_LIMB_DATA4(0xF4, 0x30, 0xA0, 0x7D),
  CRYPTO_MPI_LIMB_DATA4(0xC5, 0xB6, 0xCC, 0x31),
  CRYPTO_MPI_LIMB_DATA4(0xB5, 0x23, 0xF1, 0x5C),
  CRYPTO_MPI_LIMB_DATA4(0x62, 0xB4, 0x05, 0xC2),
  CRYPTO_MPI_LIMB_DATA4(0xB8, 0xF2, 0x54, 0xE0),
  CRYPTO_MPI_LIMB_DATA4(0x4B, 0xA7, 0x50, 0x0B),
  CRYPTO_MPI_LIMB_DATA4(0x91, 0x57, 0x51, 0x92),
  CRYPTO_MPI_LIMB_DATA4(0x46, 0x12, 0x9D, 0x38),
  CRYPTO_MPI_LIMB_DATA4(0x55, 0x2B, 0x9F, 0xB8),
  CRYPTO_MPI_LIMB_DATA4(0x54, 0xCD, 0xE8, 0x58),
  CRYPTO_MPI_LIMB_DATA4(0x75, 0x2A, 0xAB, 0x9F),
  CRYPTO_MPI_LIMB_DATA4(0x3C, 0xE7, 0x59, 0x87),
  CRYPTO_MPI_LIMB_DATA4(0x9E, 0x9B, 0x03, 0xF5),
  CRYPTO_MPI_LIMB_DATA4(0x12, 0x56, 0xF6, 0x14),
  CRYPTO_MPI_LIMB_DATA4(0x61, 0x3F, 0xDC, 0xCD),
  CRYPTO_MPI_LIMB_DATA4(0x06, 0xED, 0x21, 0x14),
  CRYPTO_MPI_LIMB_DATA4(0x55, 0x2F, 0x85, 0x78),
  CRYPTO_MPI_LIMB_DATA4(0x07, 0xDA, 0xE4, 0xB2),
  CRYPTO_MPI_LIMB_DATA4(0x0E, 0x4A, 0x69, 0x91),
```

```
    CRYPTO_MPI_LIMB_DATA4(0xAD, 0xB5, 0x6E, 0xB1),
    CRYPTO_MPI_LIMB_DATA4(0x14, 0x8F, 0x71, 0x22),
    CRYPTO_MPI_LIMB_DATA4(0xF1, 0x45, 0x61, 0x7C),
    CRYPTO_MPI_LIMB_DATA4(0x18, 0x18, 0x0A, 0x1F),
    CRYPTO_MPI_LIMB_DATA4(0x1D, 0xDD, 0xAD, 0x34),
    CRYPTO_MPI_LIMB_DATA4(0xC4, 0x18, 0x77, 0x02),
    CRYPTO_MPI_LIMB_DATA4(0x34, 0xF2, 0x27, 0x34),
    CRYPTO_MPI_LIMB_DATA4(0x77, 0x7C, 0xC0, 0xC2),
    CRYPTO_MPI_LIMB_DATA4(0x6D, 0x7F, 0xA3, 0x68),
    CRYPTO_MPI_LIMB_DATA4(0x53, 0x0E, 0x28, 0x5F),
    CRYPTO_MPI_LIMB_DATA4(0x91, 0xAC, 0x96, 0x77),
    CRYPTO_MPI_LIMB_DATA4(0x98, 0x17, 0xAE, 0x0D),
    CRYPTO_MPI_LIMB_DATA4(0x53, 0xE1, 0x73, 0x84),
    CRYPTO_MPI_LIMB_DATA4(0x5F, 0xD6, 0xFA, 0x6E),
    CRYPTO_MPI_LIMB_DATA4(0x40, 0x8D, 0x84, 0x6A),
    CRYPTO_MPI_LIMB_DATA4(0x68, 0xE1, 0x2E, 0x2A),
    CRYPTO_MPI_LIMB_DATA4(0xF8, 0x71, 0xAA, 0xA7),
    CRYPTO_MPI_LIMB_DATA4(0x6F, 0x41, 0x0C, 0x2F),
    CRYPTO_MPI_LIMB_DATA4(0x2C, 0x33, 0x47, 0x2B),
    CRYPTO_MPI_LIMB_DATA4(0xDA, 0x83, 0x99, 0xAF),
    CRYPTO_MPI_LIMB_DATA4(0xF8, 0x23, 0x7B, 0xE0)
};

const CRYPTO_RSA_PRIVATE_KEY SSH_ServerKeys_RSA_Host_2048b_PrivateKey = {
    { CRYPTO_MPI_INIT_RO(SSH_ServerKeys_RSA_Host_2048b_PrivateKey_D_aLimbs) },
    { CRYPTO_MPI_INIT_RO(SSH_ServerKeys_RSA_Host_2048b_PrivateKey_P_aLimbs) },
    { CRYPTO_MPI_INIT_RO(SSH_ServerKeys_RSA_Host_2048b_PrivateKey_Q_aLimbs) },
    { CRYPTO_MPI_INIT_RO(SSH_ServerKeys_RSA_Host_2048b_PrivateKey_DP_aLimbs) },
    { CRYPTO_MPI_INIT_RO(SSH_ServerKeys_RSA_Host_2048b_PrivateKey_DQ_aLimbs) },
    { CRYPTO_MPI_INIT_RO(SSH_ServerKeys_RSA_Host_2048b_PrivateKey_QInv_aLimbs) },
    { CRYPTO_MPI_INIT_RO(SSH_ServerKeys_RSA_Host_2048b_PrivateKey_N_aLimbs) },
    { CRYPTO_MPI_INIT_RO(SSH_ServerKeys_RSA_Host_2048b_PrivateKey_E_aLimbs) },
};
```

4.7.2   Generated DSA keys

```c
#include "CRYPTO.h"

const CRYPTO_MPI_LIMB SSH_ServerKeys_DSA_1024b_160b_P_aLimbs[] = {
  CRYPTO_MPI_LIMB_DATA4(0x37, 0xFA, 0x42, 0x6C),
  CRYPTO_MPI_LIMB_DATA4(0x82, 0xB4, 0x5D, 0x83),
  CRYPTO_MPI_LIMB_DATA4(0x5F, 0x24, 0x48, 0xDE),
  CRYPTO_MPI_LIMB_DATA4(0xA6, 0xB3, 0xAA, 0x6A),
  CRYPTO_MPI_LIMB_DATA4(0x36, 0x00, 0x6D, 0x0B),
  CRYPTO_MPI_LIMB_DATA4(0x02, 0x4E, 0x07, 0xEA),
  CRYPTO_MPI_LIMB_DATA4(0x81, 0x7E, 0x7F, 0x4A),
  CRYPTO_MPI_LIMB_DATA4(0x85, 0x5B, 0x34, 0x9C),
  CRYPTO_MPI_LIMB_DATA4(0x64, 0x60, 0xDE, 0x85),
  CRYPTO_MPI_LIMB_DATA4(0x9B, 0x39, 0xD2, 0x4E),
  CRYPTO_MPI_LIMB_DATA4(0x39, 0x51, 0x07, 0xEA),
  CRYPTO_MPI_LIMB_DATA4(0x81, 0x7E, 0x7F, 0x4A),
  CRYPTO_MPI_LIMB_DATA4(0x85, 0x5B, 0x34, 0x9C),
  CRYPTO_MPI_LIMB_DATA4(0x64, 0x60, 0xDE, 0x85),
  CRYPTO_MPI_LIMB_DATA4(0x9B, 0x39, 0xD2, 0x4E),
  CRYPTO_MPI_LIMB_DATA4(0x70, 0x54, 0x07, 0xEA),
  CRYPTO_MPI_LIMB_DATA4(0x81, 0x7E, 0x7F, 0x4A),
  CRYPTO_MPI_LIMB_DATA4(0x85, 0x5B, 0x34, 0x9C),
  CRYPTO_MPI_LIMB_DATA4(0x64, 0x60, 0xDE, 0x85),
  CRYPTO_MPI_LIMB_DATA4(0x9B, 0x39, 0xD2, 0x4E),
  CRYPTO_MPI_LIMB_DATA4(0xA7, 0x57, 0x07, 0xEA),
  CRYPTO_MPI_LIMB_DATA4(0x81, 0x7E, 0x7F, 0x4A),
  CRYPTO_MPI_LIMB_DATA4(0x85, 0x5B, 0x34, 0x9C),
  CRYPTO_MPI_LIMB_DATA4(0x64, 0x60, 0xDE, 0x85),
  CRYPTO_MPI_LIMB_DATA4(0x9B, 0x39, 0xD2, 0x4E),
  CRYPTO_MPI_LIMB_DATA4(0xDE, 0x5A, 0x07, 0xEA),
  CRYPTO_MPI_LIMB_DATA4(0x81, 0x7E, 0x7F, 0x4A),
  CRYPTO_MPI_LIMB_DATA4(0x85, 0x5B, 0x34, 0x9C),
  CRYPTO_MPI_LIMB_DATA4(0x64, 0x60, 0xDE, 0x85),
  CRYPTO_MPI_LIMB_DATA4(0x9B, 0x39, 0xD2, 0x4E),
  CRYPTO_MPI_LIMB_DATA4(0x15, 0x5E, 0x07, 0xEA),
  CRYPTO_MPI_LIMB_DATA4(0x81, 0x7E, 0x7F, 0xCA)
};

const CRYPTO_MPI_LIMB SSH_ServerKeys_DSA_1024b_160b_Q_aLimbs[] = {
  CRYPTO_MPI_LIMB_DATA4(0x55, 0x67, 0x65, 0x0C),
  CRYPTO_MPI_LIMB_DATA4(0xF2, 0xB3, 0x66, 0xEC),
  CRYPTO_MPI_LIMB_DATA4(0x56, 0xA6, 0x6F, 0xDD),
  CRYPTO_MPI_LIMB_DATA4(0xF8, 0xEC, 0x6E, 0x7E),
  CRYPTO_MPI_LIMB_DATA4(0x20, 0x1B, 0x7F, 0xFD)
};

const CRYPTO_MPI_LIMB SSH_ServerKeys_DSA_1024b_160b_G_aLimbs[] = {
  CRYPTO_MPI_LIMB_DATA4(0x71, 0x6E, 0x64, 0x5D),
  CRYPTO_MPI_LIMB_DATA4(0x20, 0xF4, 0x3A, 0x04),
  CRYPTO_MPI_LIMB_DATA4(0xF6, 0x4B, 0xEB, 0xF5),
  CRYPTO_MPI_LIMB_DATA4(0xC6, 0xB2, 0x73, 0x83),
  CRYPTO_MPI_LIMB_DATA4(0x95, 0x07, 0xB1, 0x30),
  CRYPTO_MPI_LIMB_DATA4(0xD9, 0xB3, 0x67, 0x08),
  CRYPTO_MPI_LIMB_DATA4(0xE6, 0x3B, 0x37, 0xF6),
  CRYPTO_MPI_LIMB_DATA4(0x01, 0x65, 0x52, 0x4E),
  CRYPTO_MPI_LIMB_DATA4(0x52, 0x54, 0x3B, 0xB4),
  CRYPTO_MPI_LIMB_DATA4(0x16, 0xF7, 0x29, 0xF5),
  CRYPTO_MPI_LIMB_DATA4(0x1A, 0xCB, 0x31, 0x36),
  CRYPTO_MPI_LIMB_DATA4(0x5C, 0x47, 0x7E, 0x57),
  CRYPTO_MPI_LIMB_DATA4(0x83, 0x8A, 0xC2, 0x81),
  CRYPTO_MPI_LIMB_DATA4(0x64, 0xDD, 0xF1, 0x09),
  CRYPTO_MPI_LIMB_DATA4(0x2D, 0x1B, 0x72, 0xBF),
  CRYPTO_MPI_LIMB_DATA4(0xB4, 0x8F, 0xA9, 0x66),
  CRYPTO_MPI_LIMB_DATA4(0xFE, 0x21, 0x6C, 0xBF),
  CRYPTO_MPI_LIMB_DATA4(0x1E, 0x83, 0xC1, 0x2A),
  CRYPTO_MPI_LIMB_DATA4(0x2C, 0xEC, 0x3F, 0x3D),
```

```
    CRYPTO_MPI_LIMB_DATA4(0xD1, 0xFA, 0x2F, 0xB6),
    CRYPTO_MPI_LIMB_DATA4(0x71, 0xC6, 0x59, 0x95),
    CRYPTO_MPI_LIMB_DATA4(0xC1, 0x72, 0x6C, 0xD7),
    CRYPTO_MPI_LIMB_DATA4(0xCA, 0xF5, 0x26, 0x17),
    CRYPTO_MPI_LIMB_DATA4(0x35, 0x5C, 0x08, 0xC3),
    CRYPTO_MPI_LIMB_DATA4(0xD3, 0x3E, 0x04, 0xA7),
    CRYPTO_MPI_LIMB_DATA4(0x74, 0x2D, 0xFC, 0x56),
    CRYPTO_MPI_LIMB_DATA4(0x74, 0xFF, 0x83, 0x69),
    CRYPTO_MPI_LIMB_DATA4(0x47, 0x0C, 0x50, 0xE0),
    CRYPTO_MPI_LIMB_DATA4(0x4F, 0x95, 0x6C, 0xB8),
    CRYPTO_MPI_LIMB_DATA4(0x77, 0x31, 0xA3, 0xBF),
    CRYPTO_MPI_LIMB_DATA4(0x62, 0x27, 0xB6, 0x31),
    CRYPTO_MPI_LIMB_DATA4(0x5F, 0x13, 0x30, 0xBA)
};

const CRYPTO_DSA_DOMAIN_PARAMS SSH_ServerKeys_DSA_1024b_160b_DomainParas = {
    { CRYPTO_MPI_INIT_RO(SSH_ServerKeys_DSA_1024b_160b_P_aLimbs) },
    { CRYPTO_MPI_INIT_RO(SSH_ServerKeys_DSA_1024b_160b_Q_aLimbs) },
    { CRYPTO_MPI_INIT_RO(SSH_ServerKeys_DSA_1024b_160b_G_aLimbs) },
};

const CRYPTO_MPI_LIMB SSH_ServerKeys_DSA_1024b_160b_Y_aLimbs[] = {
    CRYPTO_MPI_LIMB_DATA4(0xFE, 0xCC, 0xDA, 0xA2),
    CRYPTO_MPI_LIMB_DATA4(0xEA, 0x33, 0xE3, 0x85),
    CRYPTO_MPI_LIMB_DATA4(0x0A, 0x7E, 0x23, 0xCB),
    CRYPTO_MPI_LIMB_DATA4(0x50, 0x2A, 0x7B, 0x38),
    CRYPTO_MPI_LIMB_DATA4(0xAC, 0xEE, 0x5F, 0x40),
    CRYPTO_MPI_LIMB_DATA4(0x6A, 0x03, 0x27, 0x9D),
    CRYPTO_MPI_LIMB_DATA4(0xE4, 0x52, 0xED, 0x94),
    CRYPTO_MPI_LIMB_DATA4(0x1F, 0x7E, 0x43, 0x04),
    CRYPTO_MPI_LIMB_DATA4(0x83, 0xBE, 0x04, 0x1B),
    CRYPTO_MPI_LIMB_DATA4(0x1E, 0x3C, 0x24, 0xCB),
    CRYPTO_MPI_LIMB_DATA4(0x7F, 0x2C, 0xEB, 0xDB),
    CRYPTO_MPI_LIMB_DATA4(0x12, 0x0C, 0xFA, 0x94),
    CRYPTO_MPI_LIMB_DATA4(0xA9, 0xC2, 0x18, 0xEA),
    CRYPTO_MPI_LIMB_DATA4(0x58, 0x79, 0x81, 0xE0),
    CRYPTO_MPI_LIMB_DATA4(0x26, 0x5D, 0x1F, 0xC0),
    CRYPTO_MPI_LIMB_DATA4(0x5A, 0xC0, 0xA2, 0x20),
    CRYPTO_MPI_LIMB_DATA4(0xF4, 0x75, 0x36, 0x3A),
    CRYPTO_MPI_LIMB_DATA4(0x41, 0x42, 0xA1, 0x20),
    CRYPTO_MPI_LIMB_DATA4(0x61, 0xD3, 0x51, 0x70),
    CRYPTO_MPI_LIMB_DATA4(0x10, 0x6F, 0xA3, 0x70),
    CRYPTO_MPI_LIMB_DATA4(0x18, 0xA2, 0x14, 0xD4),
    CRYPTO_MPI_LIMB_DATA4(0x89, 0x2D, 0x3F, 0x3A),
    CRYPTO_MPI_LIMB_DATA4(0x1A, 0x59, 0xC4, 0x96),
    CRYPTO_MPI_LIMB_DATA4(0xEF, 0x19, 0x30, 0x63),
    CRYPTO_MPI_LIMB_DATA4(0x5E, 0x43, 0x95, 0xA3),
    CRYPTO_MPI_LIMB_DATA4(0x2A, 0x59, 0x59, 0xD1),
    CRYPTO_MPI_LIMB_DATA4(0x01, 0x16, 0xB2, 0x32),
    CRYPTO_MPI_LIMB_DATA4(0xBA, 0x37, 0xE0, 0x4C),
    CRYPTO_MPI_LIMB_DATA4(0x8B, 0xB8, 0x2E, 0x9E),
    CRYPTO_MPI_LIMB_DATA4(0xAC, 0xCF, 0x79, 0x1A),
    CRYPTO_MPI_LIMB_DATA4(0xA7, 0x1E, 0xE3, 0x9A),
    CRYPTO_MPI_LIMB_DATA4(0xA6, 0x39, 0x70, 0x88)
};

 const CRYPTO_DSA_PUBLIC_KEY SSH_ServerKeys_DSA_1024b_160b_PublicKey = {
    { CRYPTO_MPI_INIT_RO(SSH_ServerKeys_DSA_1024b_160b_Y_aLimbs) },
};

static const CRYPTO_MPI_LIMB SSH_ServerKeys_DSA_1024b_160b_X_aLimbs[] = {
    CRYPTO_MPI_LIMB_DATA4(0xEA, 0x46, 0x30, 0xA9),
    CRYPTO_MPI_LIMB_DATA4(0x82, 0x28, 0xF8, 0x52),
    CRYPTO_MPI_LIMB_DATA4(0xD4, 0xCC, 0x33, 0x99),
    CRYPTO_MPI_LIMB_DATA4(0xDC, 0xBF, 0x83, 0x21),
    CRYPTO_MPI_LIMB_DATA4(0x39, 0xB5, 0xFB, 0x5E)
};
```

```c
  const CRYPTO_DSA_PRIVATE_KEY SSH_ServerKeys_DSA_1024b_160b_PrivateKey = {
    { CRYPTO_MPI_INIT_RO(SSH_ServerKeys_DSA_1024b_160b_X_aLimbs) },
};

#include "CRYPTO.h"

const CRYPTO_MPI_LIMB SSH_ServerKeys_DSA_2048b_160b_P_aLimbs[] = {
  CRYPTO_MPI_LIMB_DATA4(0x1D, 0xB6, 0x02, 0x0F),
  CRYPTO_MPI_LIMB_DATA4(0x83, 0x37, 0x49, 0x00),
  CRYPTO_MPI_LIMB_DATA4(0xBC, 0x60, 0x65, 0xAF),
  CRYPTO_MPI_LIMB_DATA4(0xAE, 0x21, 0xD4, 0xED),
  CRYPTO_MPI_LIMB_DATA4(0x97, 0x36, 0x15, 0xB9),
  CRYPTO_MPI_LIMB_DATA4(0xCC, 0xBC, 0x59, 0xB2),
  CRYPTO_MPI_LIMB_DATA4(0x81, 0x5C, 0x04, 0x00),
  CRYPTO_MPI_LIMB_DATA4(0x0D, 0xB7, 0x71, 0x70),
  CRYPTO_MPI_LIMB_DATA4(0x8C, 0x06, 0xB0, 0x68),
  CRYPTO_MPI_LIMB_DATA4(0x97, 0x15, 0x74, 0xC7),
  CRYPTO_MPI_LIMB_DATA4(0xCA, 0xBF, 0x59, 0xB2),
  CRYPTO_MPI_LIMB_DATA4(0x81, 0x5C, 0x04, 0x00),
  CRYPTO_MPI_LIMB_DATA4(0x0D, 0xB7, 0x71, 0x70),
  CRYPTO_MPI_LIMB_DATA4(0x8C, 0x06, 0xB0, 0x68),
  CRYPTO_MPI_LIMB_DATA4(0x97, 0x15, 0x74, 0xC7),
  CRYPTO_MPI_LIMB_DATA4(0xC7, 0xC2, 0x59, 0xB2),
  CRYPTO_MPI_LIMB_DATA4(0x81, 0x5C, 0x04, 0x00),
  CRYPTO_MPI_LIMB_DATA4(0x0D, 0xB7, 0x71, 0x70),
  CRYPTO_MPI_LIMB_DATA4(0x8C, 0x06, 0xB0, 0x68),
  CRYPTO_MPI_LIMB_DATA4(0x97, 0x15, 0x74, 0xC7),
  CRYPTO_MPI_LIMB_DATA4(0xC4, 0xC5, 0x59, 0xB2),
  CRYPTO_MPI_LIMB_DATA4(0x81, 0x5C, 0x04, 0x00),
  CRYPTO_MPI_LIMB_DATA4(0x0D, 0xB7, 0x71, 0x70),
  CRYPTO_MPI_LIMB_DATA4(0x8C, 0x06, 0xB0, 0x68),
  CRYPTO_MPI_LIMB_DATA4(0x97, 0x15, 0x74, 0xC7),
  CRYPTO_MPI_LIMB_DATA4(0xC1, 0xC8, 0x59, 0xB2),
  CRYPTO_MPI_LIMB_DATA4(0x81, 0x5C, 0x04, 0x00),
  CRYPTO_MPI_LIMB_DATA4(0x0D, 0xB7, 0x71, 0x70),
  CRYPTO_MPI_LIMB_DATA4(0x8C, 0x06, 0xB0, 0x68),
  CRYPTO_MPI_LIMB_DATA4(0x97, 0x15, 0x74, 0xC7),
  CRYPTO_MPI_LIMB_DATA4(0xBE, 0xCB, 0x59, 0xB2),
  CRYPTO_MPI_LIMB_DATA4(0x81, 0x5C, 0x04, 0x00),
  CRYPTO_MPI_LIMB_DATA4(0x0D, 0xB7, 0x71, 0x70),
  CRYPTO_MPI_LIMB_DATA4(0x8C, 0x06, 0xB0, 0x68),
  CRYPTO_MPI_LIMB_DATA4(0x97, 0x15, 0x74, 0xC7),
  CRYPTO_MPI_LIMB_DATA4(0xBB, 0xCE, 0x59, 0xB2),
  CRYPTO_MPI_LIMB_DATA4(0x81, 0x5C, 0x04, 0x00),
  CRYPTO_MPI_LIMB_DATA4(0x0D, 0xB7, 0x71, 0x70),
  CRYPTO_MPI_LIMB_DATA4(0x8C, 0x06, 0xB0, 0x68),
  CRYPTO_MPI_LIMB_DATA4(0x97, 0x15, 0x74, 0xC7),
  CRYPTO_MPI_LIMB_DATA4(0xB8, 0xD1, 0x59, 0xB2),
  CRYPTO_MPI_LIMB_DATA4(0x81, 0x5C, 0x04, 0x00),
  CRYPTO_MPI_LIMB_DATA4(0x0D, 0xB7, 0x71, 0x70),
  CRYPTO_MPI_LIMB_DATA4(0x8C, 0x06, 0xB0, 0x68),
  CRYPTO_MPI_LIMB_DATA4(0x97, 0x15, 0x74, 0xC7),
  CRYPTO_MPI_LIMB_DATA4(0xB5, 0xD4, 0x59, 0xB2),
  CRYPTO_MPI_LIMB_DATA4(0x81, 0x5C, 0x04, 0x00),
  CRYPTO_MPI_LIMB_DATA4(0x0D, 0xB7, 0x71, 0x70),
  CRYPTO_MPI_LIMB_DATA4(0x8C, 0x06, 0xB0, 0x68),
  CRYPTO_MPI_LIMB_DATA4(0x97, 0x15, 0x74, 0xC7),
  CRYPTO_MPI_LIMB_DATA4(0xB2, 0xD7, 0x59, 0xB2),
  CRYPTO_MPI_LIMB_DATA4(0x81, 0x5C, 0x04, 0x00),
  CRYPTO_MPI_LIMB_DATA4(0x0D, 0xB7, 0x71, 0x70),
  CRYPTO_MPI_LIMB_DATA4(0x8C, 0x06, 0xB0, 0x68),
  CRYPTO_MPI_LIMB_DATA4(0x97, 0x15, 0x74, 0xC7),
  CRYPTO_MPI_LIMB_DATA4(0xAF, 0xDA, 0x59, 0xB2),
  CRYPTO_MPI_LIMB_DATA4(0x81, 0x5C, 0x04, 0x00),
  CRYPTO_MPI_LIMB_DATA4(0x0D, 0xB7, 0x71, 0x70),
  CRYPTO_MPI_LIMB_DATA4(0x8C, 0x06, 0xB0, 0x68),
  CRYPTO_MPI_LIMB_DATA4(0x97, 0x15, 0x74, 0xC7),
  CRYPTO_MPI_LIMB_DATA4(0xAC, 0xDD, 0x59, 0xB2),
```

```
  CRYPTO_MPI_LIMB_DATA4(0x81, 0x5C, 0x04, 0x00),
  CRYPTO_MPI_LIMB_DATA4(0x0D, 0xB7, 0x71, 0x70),
  CRYPTO_MPI_LIMB_DATA4(0x8C, 0x06, 0xB0, 0xE8)
};

const CRYPTO_MPI_LIMB SSH_ServerKeys_DSA_2048b_160b_Q_aLimbs[] = {
  CRYPTO_MPI_LIMB_DATA4(0xD5, 0xDC, 0xE4, 0xDB),
  CRYPTO_MPI_LIMB_DATA4(0xC3, 0x2D, 0xAD, 0x75),
  CRYPTO_MPI_LIMB_DATA4(0xE9, 0xD4, 0xD4, 0x8F),
  CRYPTO_MPI_LIMB_DATA4(0xBB, 0x5E, 0x3D, 0x67),
  CRYPTO_MPI_LIMB_DATA4(0x26, 0xB2, 0x85, 0xA3)
};

const CRYPTO_MPI_LIMB SSH_ServerKeys_DSA_2048b_160b_G_aLimbs[] = {
  CRYPTO_MPI_LIMB_DATA4(0x6B, 0x69, 0x76, 0x28),
  CRYPTO_MPI_LIMB_DATA4(0x2C, 0xFE, 0x3E, 0x45),
  CRYPTO_MPI_LIMB_DATA4(0xFC, 0x34, 0x7A, 0x24),
  CRYPTO_MPI_LIMB_DATA4(0x3E, 0xEC, 0x72, 0x36),
  CRYPTO_MPI_LIMB_DATA4(0xA2, 0xDF, 0xDF, 0xA3),
  CRYPTO_MPI_LIMB_DATA4(0x7B, 0xC5, 0xEB, 0x73),
  CRYPTO_MPI_LIMB_DATA4(0xE6, 0x46, 0x37, 0xF4),
  CRYPTO_MPI_LIMB_DATA4(0xF7, 0x98, 0x08, 0x43),
  CRYPTO_MPI_LIMB_DATA4(0x29, 0xD5, 0x58, 0x8B),
  CRYPTO_MPI_LIMB_DATA4(0x7A, 0xEB, 0x81, 0x5C),
  CRYPTO_MPI_LIMB_DATA4(0xA6, 0xB3, 0xF2, 0xD8),
  CRYPTO_MPI_LIMB_DATA4(0x74, 0x5C, 0x8B, 0x6E),
  CRYPTO_MPI_LIMB_DATA4(0x11, 0xA6, 0xE4, 0x79),
  CRYPTO_MPI_LIMB_DATA4(0x7A, 0x21, 0x5D, 0xCB),
  CRYPTO_MPI_LIMB_DATA4(0xB4, 0x96, 0x51, 0xB9),
  CRYPTO_MPI_LIMB_DATA4(0x47, 0x56, 0x0A, 0x61),
  CRYPTO_MPI_LIMB_DATA4(0x62, 0xC8, 0xB1, 0x36),
  CRYPTO_MPI_LIMB_DATA4(0xC5, 0xE4, 0x7E, 0xD0),
  CRYPTO_MPI_LIMB_DATA4(0x77, 0xD3, 0xCF, 0xCC),
  CRYPTO_MPI_LIMB_DATA4(0xA5, 0xCC, 0x53, 0x00),
  CRYPTO_MPI_LIMB_DATA4(0xC8, 0x13, 0xA0, 0xF9),
  CRYPTO_MPI_LIMB_DATA4(0x47, 0x08, 0x98, 0x2D),
  CRYPTO_MPI_LIMB_DATA4(0xB3, 0xE3, 0x04, 0xA2),
  CRYPTO_MPI_LIMB_DATA4(0xB0, 0xA9, 0x2D, 0x13),
  CRYPTO_MPI_LIMB_DATA4(0xA3, 0x4D, 0xE5, 0xCE),
  CRYPTO_MPI_LIMB_DATA4(0xB4, 0xC3, 0xF9, 0xB3),
  CRYPTO_MPI_LIMB_DATA4(0x54, 0xA8, 0x84, 0x19),
  CRYPTO_MPI_LIMB_DATA4(0xCF, 0xB5, 0x63, 0x4B),
  CRYPTO_MPI_LIMB_DATA4(0xC0, 0x57, 0x35, 0xD4),
  CRYPTO_MPI_LIMB_DATA4(0x3F, 0xAC, 0x2E, 0x79),
  CRYPTO_MPI_LIMB_DATA4(0x35, 0x39, 0xFE, 0x69),
  CRYPTO_MPI_LIMB_DATA4(0x88, 0xDD, 0xE7, 0x8F),
  CRYPTO_MPI_LIMB_DATA4(0x1D, 0x52, 0xAD, 0xE1),
  CRYPTO_MPI_LIMB_DATA4(0xC7, 0xCA, 0x8D, 0xFE),
  CRYPTO_MPI_LIMB_DATA4(0xB6, 0x22, 0x9C, 0xAD),
  CRYPTO_MPI_LIMB_DATA4(0x30, 0x77, 0x27, 0x04),
  CRYPTO_MPI_LIMB_DATA4(0xC0, 0x19, 0xA3, 0xAC),
  CRYPTO_MPI_LIMB_DATA4(0x33, 0x4B, 0x40, 0xC2),
  CRYPTO_MPI_LIMB_DATA4(0x10, 0x20, 0x73, 0xF7),
  CRYPTO_MPI_LIMB_DATA4(0x59, 0x77, 0x1D, 0x22),
  CRYPTO_MPI_LIMB_DATA4(0xD5, 0xAE, 0xE9, 0xDE),
  CRYPTO_MPI_LIMB_DATA4(0xC5, 0x80, 0x22, 0xEB),
  CRYPTO_MPI_LIMB_DATA4(0xE0, 0xD9, 0x77, 0x22),
  CRYPTO_MPI_LIMB_DATA4(0xB7, 0xA1, 0x4B, 0x61),
  CRYPTO_MPI_LIMB_DATA4(0xBE, 0x46, 0x2B, 0x7C),
  CRYPTO_MPI_LIMB_DATA4(0x77, 0x7F, 0x2B, 0x68),
  CRYPTO_MPI_LIMB_DATA4(0xA2, 0xC7, 0xB0, 0x7F),
  CRYPTO_MPI_LIMB_DATA4(0x18, 0x67, 0xC7, 0x17),
  CRYPTO_MPI_LIMB_DATA4(0x96, 0xF1, 0x06, 0x4A),
  CRYPTO_MPI_LIMB_DATA4(0x03, 0x2C, 0x14, 0x76),
  CRYPTO_MPI_LIMB_DATA4(0xAE, 0x46, 0x4F, 0xEF),
  CRYPTO_MPI_LIMB_DATA4(0x97, 0x50, 0xC9, 0x45),
  CRYPTO_MPI_LIMB_DATA4(0xAC, 0xE1, 0xF7, 0xA5),
  CRYPTO_MPI_LIMB_DATA4(0x9C, 0x0C, 0x8E, 0xC9),
```

```
  CRYPTO_MPI_LIMB_DATA4(0x46, 0x02, 0x8B, 0xE4),
  CRYPTO_MPI_LIMB_DATA4(0x4F, 0xEB, 0x24, 0xA6),
  CRYPTO_MPI_LIMB_DATA4(0x9B, 0x2B, 0x6A, 0x41),
  CRYPTO_MPI_LIMB_DATA4(0xD7, 0x1B, 0x4A, 0x74),
  CRYPTO_MPI_LIMB_DATA4(0x48, 0x8F, 0xDB, 0x3F),
  CRYPTO_MPI_LIMB_DATA4(0x96, 0x00, 0xD6, 0x0B),
  CRYPTO_MPI_LIMB_DATA4(0xA1, 0xA6, 0x0C, 0x63),
  CRYPTO_MPI_LIMB_DATA4(0xA3, 0x86, 0xBC, 0x54),
  CRYPTO_MPI_LIMB_DATA4(0x61, 0x81, 0xC9, 0x37),
  CRYPTO_MPI_LIMB_DATA4(0x9E, 0xCC, 0x6A, 0x83)
};

const CRYPTO_DSA_DOMAIN_PARAMS SSH_ServerKeys_DSA_2048b_160b_DomainParas = {
  { CRYPTO_MPI_INIT_RO(SSH_ServerKeys_DSA_2048b_160b_P_aLimbs) },
  { CRYPTO_MPI_INIT_RO(SSH_ServerKeys_DSA_2048b_160b_Q_aLimbs) },
  { CRYPTO_MPI_INIT_RO(SSH_ServerKeys_DSA_2048b_160b_G_aLimbs) },
};

const CRYPTO_MPI_LIMB SSH_ServerKeys_DSA_2048b_160b_Y_aLimbs[] = {
  CRYPTO_MPI_LIMB_DATA4(0x94, 0xAC, 0x08, 0xB5),
  CRYPTO_MPI_LIMB_DATA4(0x39, 0xB7, 0x07, 0xA4),
  CRYPTO_MPI_LIMB_DATA4(0xE4, 0x6C, 0xFF, 0xEF),
  CRYPTO_MPI_LIMB_DATA4(0x6D, 0x62, 0x83, 0x03),
  CRYPTO_MPI_LIMB_DATA4(0x47, 0x10, 0x6E, 0x56),
  CRYPTO_MPI_LIMB_DATA4(0xEA, 0x8D, 0x98, 0x8C),
  CRYPTO_MPI_LIMB_DATA4(0x3B, 0x0A, 0x68, 0xE4),
  CRYPTO_MPI_LIMB_DATA4(0x03, 0x06, 0xED, 0x24),
  CRYPTO_MPI_LIMB_DATA4(0x46, 0xFA, 0x6B, 0x66),
  CRYPTO_MPI_LIMB_DATA4(0xCD, 0xDF, 0x38, 0x8C),
  CRYPTO_MPI_LIMB_DATA4(0x6A, 0xC5, 0x1E, 0xC6),
  CRYPTO_MPI_LIMB_DATA4(0x9C, 0x4F, 0x7E, 0x36),
  CRYPTO_MPI_LIMB_DATA4(0x43, 0x15, 0x4D, 0x35),
  CRYPTO_MPI_LIMB_DATA4(0x6C, 0xD4, 0x4B, 0x62),
  CRYPTO_MPI_LIMB_DATA4(0x1E, 0x68, 0x64, 0x2E),
  CRYPTO_MPI_LIMB_DATA4(0x40, 0xAA, 0x87, 0x9D),
  CRYPTO_MPI_LIMB_DATA4(0x67, 0x83, 0x36, 0xCB),
  CRYPTO_MPI_LIMB_DATA4(0xF3, 0x74, 0xB8, 0x6E),
  CRYPTO_MPI_LIMB_DATA4(0x69, 0x70, 0x76, 0x10),
  CRYPTO_MPI_LIMB_DATA4(0xFB, 0x6E, 0xF6, 0x09),
  CRYPTO_MPI_LIMB_DATA4(0xE1, 0xE8, 0xEC, 0xD8),
  CRYPTO_MPI_LIMB_DATA4(0xF8, 0xE6, 0xAF, 0xC8),
  CRYPTO_MPI_LIMB_DATA4(0xC5, 0x14, 0xB5, 0x40),
  CRYPTO_MPI_LIMB_DATA4(0xD2, 0x0D, 0x06, 0xAD),
  CRYPTO_MPI_LIMB_DATA4(0xE4, 0xB9, 0xF8, 0xF2),
  CRYPTO_MPI_LIMB_DATA4(0xB0, 0xA2, 0x42, 0x5A),
  CRYPTO_MPI_LIMB_DATA4(0x4B, 0xF3, 0x86, 0xDB),
  CRYPTO_MPI_LIMB_DATA4(0x6F, 0x4D, 0xD3, 0x14),
  CRYPTO_MPI_LIMB_DATA4(0xEC, 0x6F, 0x8D, 0xAD),
  CRYPTO_MPI_LIMB_DATA4(0x89, 0x88, 0xE4, 0x28),
  CRYPTO_MPI_LIMB_DATA4(0xDB, 0xB7, 0xE0, 0x1D),
  CRYPTO_MPI_LIMB_DATA4(0xE3, 0xDD, 0x5B, 0x16),
  CRYPTO_MPI_LIMB_DATA4(0x30, 0xEF, 0x8A, 0x40),
  CRYPTO_MPI_LIMB_DATA4(0x71, 0x33, 0x84, 0x7A),
  CRYPTO_MPI_LIMB_DATA4(0x82, 0x14, 0x0C, 0xB5),
  CRYPTO_MPI_LIMB_DATA4(0x14, 0x47, 0x35, 0x31),
  CRYPTO_MPI_LIMB_DATA4(0xD6, 0x11, 0x25, 0x1C),
  CRYPTO_MPI_LIMB_DATA4(0x74, 0x14, 0xF4, 0x82),
  CRYPTO_MPI_LIMB_DATA4(0x87, 0xB2, 0x28, 0x44),
  CRYPTO_MPI_LIMB_DATA4(0x06, 0x7F, 0x9A, 0x0F),
  CRYPTO_MPI_LIMB_DATA4(0xC3, 0x50, 0xFC, 0x47),
  CRYPTO_MPI_LIMB_DATA4(0x6A, 0xED, 0xC8, 0x6C),
  CRYPTO_MPI_LIMB_DATA4(0x3B, 0x16, 0xBE, 0x38),
  CRYPTO_MPI_LIMB_DATA4(0x73, 0x9F, 0xB0, 0xD2),
  CRYPTO_MPI_LIMB_DATA4(0xC7, 0xBF, 0x1B, 0x97),
  CRYPTO_MPI_LIMB_DATA4(0xDD, 0x42, 0x7E, 0xF5),
  CRYPTO_MPI_LIMB_DATA4(0xB8, 0x5B, 0x82, 0xA9),
  CRYPTO_MPI_LIMB_DATA4(0x00, 0x80, 0x87, 0xEB),
  CRYPTO_MPI_LIMB_DATA4(0x89, 0x90, 0xA4, 0x8F),
```

```c
  CRYPTO_MPI_LIMB_DATA4(0xF2, 0x91, 0xE8, 0x79),
  CRYPTO_MPI_LIMB_DATA4(0x82, 0x18, 0x95, 0x4B),
  CRYPTO_MPI_LIMB_DATA4(0x07, 0x50, 0x90, 0xE8),
  CRYPTO_MPI_LIMB_DATA4(0x92, 0x1F, 0x5A, 0x8F),
  CRYPTO_MPI_LIMB_DATA4(0x57, 0xD9, 0xBD, 0x3F),
  CRYPTO_MPI_LIMB_DATA4(0xCC, 0xAC, 0xE8, 0x5A),
  CRYPTO_MPI_LIMB_DATA4(0x30, 0xD4, 0x67, 0x7E),
  CRYPTO_MPI_LIMB_DATA4(0xB4, 0x6C, 0xC0, 0xE6),
  CRYPTO_MPI_LIMB_DATA4(0x42, 0x03, 0xB6, 0x95),
  CRYPTO_MPI_LIMB_DATA4(0x7B, 0x73, 0xC5, 0xBC),
  CRYPTO_MPI_LIMB_DATA4(0xBA, 0xCA, 0xF0, 0xC2),
  CRYPTO_MPI_LIMB_DATA4(0x0E, 0xA5, 0x32, 0x0F),
  CRYPTO_MPI_LIMB_DATA4(0x5B, 0x8D, 0x7E, 0xDF),
  CRYPTO_MPI_LIMB_DATA4(0xB4, 0x9D, 0x71, 0xEC),
  CRYPTO_MPI_LIMB_DATA4(0x5E, 0x41, 0xA2, 0x11)
};

 const CRYPTO_DSA_PUBLIC_KEY SSH_ServerKeys_DSA_2048b_160b_PublicKey = {
   { CRYPTO_MPI_INIT_RO(SSH_ServerKeys_DSA_2048b_160b_Y_aLimbs) },
};

static const CRYPTO_MPI_LIMB SSH_ServerKeys_DSA_2048b_160b_X_aLimbs[] = {
  CRYPTO_MPI_LIMB_DATA4(0xCA, 0xE4, 0xF3, 0x4F),
  CRYPTO_MPI_LIMB_DATA4(0xAE, 0xAD, 0x4B, 0x7C),
  CRYPTO_MPI_LIMB_DATA4(0xEB, 0x89, 0xC6, 0xF7),
  CRYPTO_MPI_LIMB_DATA4(0x53, 0xFD, 0x81, 0xE6),
  CRYPTO_MPI_LIMB_DATA4(0x33, 0xD5, 0xBD, 0x28)
};

 const CRYPTO_DSA_PRIVATE_KEY SSH_ServerKeys_DSA_2048b_160b_PrivateKey = {
   { CRYPTO_MPI_INIT_RO(SSH_ServerKeys_DSA_2048b_160b_X_aLimbs) },
};

#include "CRYPTO.h"

const CRYPTO_MPI_LIMB SSH_ServerKeys_DSA_2048b_256b_P_aLimbs[] = {
  CRYPTO_MPI_LIMB_DATA4(0x55, 0x18, 0x08, 0xEA),
  CRYPTO_MPI_LIMB_DATA4(0x2A, 0xA1, 0xED, 0x15),
  CRYPTO_MPI_LIMB_DATA4(0x17, 0xA7, 0x30, 0x2D),
  CRYPTO_MPI_LIMB_DATA4(0x1D, 0x5F, 0xDD, 0x79),
  CRYPTO_MPI_LIMB_DATA4(0x18, 0x3F, 0x24, 0x03),
  CRYPTO_MPI_LIMB_DATA4(0x8D, 0x87, 0x18, 0xCF),
  CRYPTO_MPI_LIMB_DATA4(0x1B, 0x02, 0x85, 0xB5),
  CRYPTO_MPI_LIMB_DATA4(0xC9, 0x22, 0x04, 0x6B),
  CRYPTO_MPI_LIMB_DATA4(0xB5, 0x93, 0x83, 0xDF),
  CRYPTO_MPI_LIMB_DATA4(0xE3, 0x6A, 0x1F, 0xD0),
  CRYPTO_MPI_LIMB_DATA4(0xF4, 0x92, 0x18, 0xCF),
  CRYPTO_MPI_LIMB_DATA4(0x1B, 0x02, 0x85, 0xB5),
  CRYPTO_MPI_LIMB_DATA4(0xC9, 0x22, 0x04, 0x6B),
  CRYPTO_MPI_LIMB_DATA4(0xB5, 0x93, 0x83, 0xDF),
  CRYPTO_MPI_LIMB_DATA4(0xE3, 0x6A, 0x1F, 0xD0),
  CRYPTO_MPI_LIMB_DATA4(0x5B, 0x9E, 0x18, 0xCF),
  CRYPTO_MPI_LIMB_DATA4(0x1B, 0x02, 0x85, 0xB5),
  CRYPTO_MPI_LIMB_DATA4(0xC9, 0x22, 0x04, 0x6B),
  CRYPTO_MPI_LIMB_DATA4(0xB5, 0x93, 0x83, 0xDF),
  CRYPTO_MPI_LIMB_DATA4(0xE3, 0x6A, 0x1F, 0xD0),
  CRYPTO_MPI_LIMB_DATA4(0xC2, 0xA9, 0x18, 0xCF),
  CRYPTO_MPI_LIMB_DATA4(0x1B, 0x02, 0x85, 0xB5),
  CRYPTO_MPI_LIMB_DATA4(0xC9, 0x22, 0x04, 0x6B),
  CRYPTO_MPI_LIMB_DATA4(0xB5, 0x93, 0x83, 0xDF),
  CRYPTO_MPI_LIMB_DATA4(0xE3, 0x6A, 0x1F, 0xD0),
  CRYPTO_MPI_LIMB_DATA4(0x29, 0xB5, 0x18, 0xCF),
  CRYPTO_MPI_LIMB_DATA4(0x1B, 0x02, 0x85, 0xB5),
  CRYPTO_MPI_LIMB_DATA4(0xC9, 0x22, 0x04, 0x6B),
  CRYPTO_MPI_LIMB_DATA4(0xB5, 0x93, 0x83, 0xDF),
  CRYPTO_MPI_LIMB_DATA4(0xE3, 0x6A, 0x1F, 0xD0),
  CRYPTO_MPI_LIMB_DATA4(0x90, 0xC0, 0x18, 0xCF),
  CRYPTO_MPI_LIMB_DATA4(0x1B, 0x02, 0x85, 0xB5),
```

```
  CRYPTO_MPI_LIMB_DATA4(0xC9, 0x22, 0x04, 0x6B),
  CRYPTO_MPI_LIMB_DATA4(0xB5, 0x93, 0x83, 0xDF),
  CRYPTO_MPI_LIMB_DATA4(0xE3, 0x6A, 0x1F, 0xD0),
  CRYPTO_MPI_LIMB_DATA4(0xF7, 0xCB, 0x18, 0xCF),
  CRYPTO_MPI_LIMB_DATA4(0x1B, 0x02, 0x85, 0xB5),
  CRYPTO_MPI_LIMB_DATA4(0xC9, 0x22, 0x04, 0x6B),
  CRYPTO_MPI_LIMB_DATA4(0xB5, 0x93, 0x83, 0xDF),
  CRYPTO_MPI_LIMB_DATA4(0xE3, 0x6A, 0x1F, 0xD0),
  CRYPTO_MPI_LIMB_DATA4(0x5E, 0xD7, 0x18, 0xCF),
  CRYPTO_MPI_LIMB_DATA4(0x1B, 0x02, 0x85, 0xB5),
  CRYPTO_MPI_LIMB_DATA4(0xC9, 0x22, 0x04, 0x6B),
  CRYPTO_MPI_LIMB_DATA4(0xB5, 0x93, 0x83, 0xDF),
  CRYPTO_MPI_LIMB_DATA4(0xE3, 0x6A, 0x1F, 0xD0),
  CRYPTO_MPI_LIMB_DATA4(0xC5, 0xE2, 0x18, 0xCF),
  CRYPTO_MPI_LIMB_DATA4(0x1B, 0x02, 0x85, 0xB5),
  CRYPTO_MPI_LIMB_DATA4(0xC9, 0x22, 0x04, 0x6B),
  CRYPTO_MPI_LIMB_DATA4(0xB5, 0x93, 0x83, 0xDF),
  CRYPTO_MPI_LIMB_DATA4(0xE3, 0x6A, 0x1F, 0xD0),
  CRYPTO_MPI_LIMB_DATA4(0x2C, 0xEE, 0x18, 0xCF),
  CRYPTO_MPI_LIMB_DATA4(0x1B, 0x02, 0x85, 0xB5),
  CRYPTO_MPI_LIMB_DATA4(0xC9, 0x22, 0x04, 0x6B),
  CRYPTO_MPI_LIMB_DATA4(0xB5, 0x93, 0x83, 0xDF),
  CRYPTO_MPI_LIMB_DATA4(0xE3, 0x6A, 0x1F, 0xD0),
  CRYPTO_MPI_LIMB_DATA4(0x93, 0xF9, 0x18, 0xCF),
  CRYPTO_MPI_LIMB_DATA4(0x1B, 0x02, 0x85, 0xB5),
  CRYPTO_MPI_LIMB_DATA4(0xC9, 0x22, 0x04, 0x6B),
  CRYPTO_MPI_LIMB_DATA4(0xB5, 0x93, 0x83, 0xDF),
  CRYPTO_MPI_LIMB_DATA4(0xE3, 0x6A, 0x1F, 0xD0),
  CRYPTO_MPI_LIMB_DATA4(0xFA, 0x04, 0x19, 0xCF),
  CRYPTO_MPI_LIMB_DATA4(0x1B, 0x02, 0x85, 0xB5),
  CRYPTO_MPI_LIMB_DATA4(0xC9, 0x22, 0x04, 0x6B),
  CRYPTO_MPI_LIMB_DATA4(0xB5, 0x93, 0x83, 0xDF)
};

const CRYPTO_MPI_LIMB SSH_ServerKeys_DSA_2048b_256b_Q_aLimbs[] = {
  CRYPTO_MPI_LIMB_DATA4(0x9B, 0x07, 0x7F, 0x3B),
  CRYPTO_MPI_LIMB_DATA4(0x7C, 0x17, 0x00, 0xB4),
  CRYPTO_MPI_LIMB_DATA4(0x7C, 0xD6, 0x07, 0x00),
  CRYPTO_MPI_LIMB_DATA4(0xCB, 0xB9, 0x02, 0x07),
  CRYPTO_MPI_LIMB_DATA4(0x9C, 0x28, 0xE1, 0xBC)
};

const CRYPTO_MPI_LIMB SSH_ServerKeys_DSA_2048b_256b_G_aLimbs[] = {
  CRYPTO_MPI_LIMB_DATA4(0xB7, 0x6E, 0x21, 0x32),
  CRYPTO_MPI_LIMB_DATA4(0x77, 0xF1, 0x24, 0xF7),
  CRYPTO_MPI_LIMB_DATA4(0xDF, 0x77, 0xA6, 0x35),
  CRYPTO_MPI_LIMB_DATA4(0x01, 0xD0, 0x0B, 0xD4),
  CRYPTO_MPI_LIMB_DATA4(0x59, 0x10, 0x8C, 0x8E),
  CRYPTO_MPI_LIMB_DATA4(0xA6, 0x50, 0x1D, 0xD6),
  CRYPTO_MPI_LIMB_DATA4(0x6C, 0x65, 0x45, 0x2A),
  CRYPTO_MPI_LIMB_DATA4(0xDE, 0x99, 0x55, 0x69),
  CRYPTO_MPI_LIMB_DATA4(0x06, 0xDF, 0x06, 0xC4),
  CRYPTO_MPI_LIMB_DATA4(0x2D, 0xB9, 0x81, 0xEB),
  CRYPTO_MPI_LIMB_DATA4(0x42, 0x0C, 0x34, 0xB8),
  CRYPTO_MPI_LIMB_DATA4(0x19, 0x38, 0xC6, 0x0A),
  CRYPTO_MPI_LIMB_DATA4(0x21, 0x43, 0x39, 0x9B),
  CRYPTO_MPI_LIMB_DATA4(0xEA, 0x61, 0x75, 0x01),
  CRYPTO_MPI_LIMB_DATA4(0xF0, 0x0B, 0x62, 0x39),
  CRYPTO_MPI_LIMB_DATA4(0x8F, 0x72, 0x17, 0x30),
  CRYPTO_MPI_LIMB_DATA4(0xF1, 0x51, 0xD2, 0xFB),
  CRYPTO_MPI_LIMB_DATA4(0x4D, 0xA9, 0xFF, 0x48),
  CRYPTO_MPI_LIMB_DATA4(0x14, 0x9C, 0x70, 0xBE),
  CRYPTO_MPI_LIMB_DATA4(0x38, 0x0B, 0x63, 0x29),
  CRYPTO_MPI_LIMB_DATA4(0xC2, 0xD7, 0xA1, 0x82),
  CRYPTO_MPI_LIMB_DATA4(0xE4, 0x30, 0x61, 0x74),
  CRYPTO_MPI_LIMB_DATA4(0xBF, 0xA5, 0xE3, 0xEB),
  CRYPTO_MPI_LIMB_DATA4(0x85, 0xF5, 0xC3, 0x03),
  CRYPTO_MPI_LIMB_DATA4(0xC5, 0x51, 0xC6, 0x8F),
```

```
  CRYPTO_MPI_LIMB_DATA4(0xE5, 0x5E, 0x24, 0x94),
  CRYPTO_MPI_LIMB_DATA4(0x87, 0x66, 0x1A, 0xD8),
  CRYPTO_MPI_LIMB_DATA4(0x88, 0xC2, 0x6F, 0xD6),
  CRYPTO_MPI_LIMB_DATA4(0x5F, 0x6F, 0x0C, 0x5B),
  CRYPTO_MPI_LIMB_DATA4(0x2F, 0x63, 0x08, 0x43),
  CRYPTO_MPI_LIMB_DATA4(0xB6, 0xC0, 0xC7, 0x1D),
  CRYPTO_MPI_LIMB_DATA4(0xFA, 0x01, 0x61, 0x49),
  CRYPTO_MPI_LIMB_DATA4(0x2F, 0x10, 0xD5, 0x0E),
  CRYPTO_MPI_LIMB_DATA4(0x1F, 0xED, 0x81, 0x63),
  CRYPTO_MPI_LIMB_DATA4(0x4A, 0x12, 0x48, 0x26),
  CRYPTO_MPI_LIMB_DATA4(0x4B, 0x63, 0x85, 0xB7),
  CRYPTO_MPI_LIMB_DATA4(0x5A, 0x54, 0x0D, 0x97),
  CRYPTO_MPI_LIMB_DATA4(0xFD, 0x2B, 0xE8, 0xFF),
  CRYPTO_MPI_LIMB_DATA4(0x3D, 0x8B, 0x96, 0x4A),
  CRYPTO_MPI_LIMB_DATA4(0x1D, 0x42, 0xA9, 0x20),
  CRYPTO_MPI_LIMB_DATA4(0xD8, 0x0D, 0xDC, 0xF9),
  CRYPTO_MPI_LIMB_DATA4(0x64, 0xF6, 0x1D, 0x31),
  CRYPTO_MPI_LIMB_DATA4(0x63, 0xDC, 0x54, 0xA0),
  CRYPTO_MPI_LIMB_DATA4(0x89, 0xA6, 0xD9, 0x8C),
  CRYPTO_MPI_LIMB_DATA4(0xB3, 0x9D, 0x63, 0xED),
  CRYPTO_MPI_LIMB_DATA4(0xCD, 0x90, 0x2C, 0x6E),
  CRYPTO_MPI_LIMB_DATA4(0xA8, 0x3E, 0x16, 0xA7),
  CRYPTO_MPI_LIMB_DATA4(0xAC, 0x97, 0xC2, 0x6D),
  CRYPTO_MPI_LIMB_DATA4(0x3B, 0x43, 0x2F, 0xCA),
  CRYPTO_MPI_LIMB_DATA4(0xAC, 0x6D, 0x5B, 0x25),
  CRYPTO_MPI_LIMB_DATA4(0xD0, 0x54, 0xB8, 0x29),
  CRYPTO_MPI_LIMB_DATA4(0xC7, 0x02, 0xC6, 0xAD),
  CRYPTO_MPI_LIMB_DATA4(0xC6, 0xFF, 0x80, 0x76),
  CRYPTO_MPI_LIMB_DATA4(0x82, 0xFA, 0x99, 0xB1),
  CRYPTO_MPI_LIMB_DATA4(0x85, 0x2E, 0x05, 0x2A),
  CRYPTO_MPI_LIMB_DATA4(0xDC, 0x8D, 0xCD, 0xFF),
  CRYPTO_MPI_LIMB_DATA4(0x8D, 0x90, 0x17, 0xBE),
  CRYPTO_MPI_LIMB_DATA4(0xD2, 0x40, 0xB2, 0x39),
  CRYPTO_MPI_LIMB_DATA4(0xD4, 0xFC, 0x72, 0xFD),
  CRYPTO_MPI_LIMB_DATA4(0xF6, 0x09, 0x61, 0xE4),
  CRYPTO_MPI_LIMB_DATA4(0x20, 0x3B, 0x9B, 0xA0),
  CRYPTO_MPI_LIMB_DATA4(0xE3, 0x7F, 0x50, 0x4F),
  CRYPTO_MPI_LIMB_DATA4(0xA8, 0x19, 0x2C, 0x9D),
  CRYPTO_MPI_LIMB_DATA4(0xC7, 0xCE, 0x54, 0xCA)
};

const CRYPTO_DSA_DOMAIN_PARAMS SSH_ServerKeys_DSA_2048b_256b_DomainParas = {
  { CRYPTO_MPI_INIT_RO(SSH_ServerKeys_DSA_2048b_256b_P_aLimbs) },
  { CRYPTO_MPI_INIT_RO(SSH_ServerKeys_DSA_2048b_256b_Q_aLimbs) },
  { CRYPTO_MPI_INIT_RO(SSH_ServerKeys_DSA_2048b_256b_G_aLimbs) },
};

const CRYPTO_MPI_LIMB SSH_ServerKeys_DSA_2048b_256b_Y_aLimbs[] = {
  CRYPTO_MPI_LIMB_DATA4(0x42, 0xCD, 0xAA, 0x67),
  CRYPTO_MPI_LIMB_DATA4(0xC9, 0x3B, 0x5B, 0x14),
  CRYPTO_MPI_LIMB_DATA4(0x38, 0xAA, 0x3B, 0x73),
  CRYPTO_MPI_LIMB_DATA4(0x1C, 0x5F, 0x85, 0x0B),
  CRYPTO_MPI_LIMB_DATA4(0xD4, 0x8A, 0x07, 0x91),
  CRYPTO_MPI_LIMB_DATA4(0x20, 0x65, 0xCC, 0x3F),
  CRYPTO_MPI_LIMB_DATA4(0xEE, 0x46, 0x3C, 0x90),
  CRYPTO_MPI_LIMB_DATA4(0xC0, 0x7C, 0x6E, 0x8E),
  CRYPTO_MPI_LIMB_DATA4(0x87, 0xCE, 0xFE, 0x15),
  CRYPTO_MPI_LIMB_DATA4(0x0A, 0xBC, 0xC2, 0x03),
  CRYPTO_MPI_LIMB_DATA4(0x98, 0x09, 0xDA, 0xB8),
  CRYPTO_MPI_LIMB_DATA4(0x2C, 0x91, 0x46, 0x4D),
  CRYPTO_MPI_LIMB_DATA4(0xD8, 0x5E, 0x88, 0x96),
  CRYPTO_MPI_LIMB_DATA4(0x08, 0x52, 0x45, 0xBC),
  CRYPTO_MPI_LIMB_DATA4(0xD7, 0x91, 0x1D, 0x65),
  CRYPTO_MPI_LIMB_DATA4(0x42, 0x11, 0xF3, 0xAC),
  CRYPTO_MPI_LIMB_DATA4(0x39, 0x28, 0xA9, 0x3B),
  CRYPTO_MPI_LIMB_DATA4(0xC9, 0xA6, 0xAB, 0x2E),
  CRYPTO_MPI_LIMB_DATA4(0x9A, 0x91, 0xBA, 0x3B),
  CRYPTO_MPI_LIMB_DATA4(0x28, 0x50, 0xF7, 0xDE),
```

```
  CRYPTO_MPI_LIMB_DATA4(0x0A, 0xEA, 0xCA, 0x4D),
  CRYPTO_MPI_LIMB_DATA4(0x0F, 0x90, 0xD0, 0x37),
  CRYPTO_MPI_LIMB_DATA4(0x92, 0x9B, 0xD2, 0x02),
  CRYPTO_MPI_LIMB_DATA4(0xBE, 0xFF, 0x31, 0x67),
  CRYPTO_MPI_LIMB_DATA4(0xA2, 0x8C, 0x51, 0x63),
  CRYPTO_MPI_LIMB_DATA4(0x04, 0x3C, 0x4B, 0xAA),
  CRYPTO_MPI_LIMB_DATA4(0x13, 0x6D, 0x55, 0x8B),
  CRYPTO_MPI_LIMB_DATA4(0x30, 0x24, 0xCE, 0x42),
  CRYPTO_MPI_LIMB_DATA4(0x18, 0x23, 0x13, 0xEC),
  CRYPTO_MPI_LIMB_DATA4(0xD1, 0xB0, 0xA0, 0x96),
  CRYPTO_MPI_LIMB_DATA4(0x9E, 0x68, 0x76, 0xAE),
  CRYPTO_MPI_LIMB_DATA4(0xAA, 0x41, 0xE3, 0x24),
  CRYPTO_MPI_LIMB_DATA4(0xA3, 0xD5, 0x37, 0x41),
  CRYPTO_MPI_LIMB_DATA4(0x15, 0x3D, 0x47, 0x6A),
  CRYPTO_MPI_LIMB_DATA4(0x77, 0x08, 0x88, 0x16),
  CRYPTO_MPI_LIMB_DATA4(0xCC, 0x41, 0x8F, 0xF4),
  CRYPTO_MPI_LIMB_DATA4(0x29, 0x88, 0x78, 0x42),
  CRYPTO_MPI_LIMB_DATA4(0x30, 0x91, 0x3B, 0x83),
  CRYPTO_MPI_LIMB_DATA4(0x39, 0x65, 0x5F, 0x73),
  CRYPTO_MPI_LIMB_DATA4(0x8B, 0xA7, 0xE5, 0xBC),
  CRYPTO_MPI_LIMB_DATA4(0x25, 0x9E, 0x3B, 0x2C),
  CRYPTO_MPI_LIMB_DATA4(0xCC, 0xE8, 0xDF, 0x6D),
  CRYPTO_MPI_LIMB_DATA4(0x3D, 0x4A, 0x20, 0x63),
  CRYPTO_MPI_LIMB_DATA4(0xD5, 0xF0, 0x69, 0x11),
  CRYPTO_MPI_LIMB_DATA4(0x14, 0x12, 0x83, 0x68),
  CRYPTO_MPI_LIMB_DATA4(0xCC, 0xD7, 0x0A, 0x43),
  CRYPTO_MPI_LIMB_DATA4(0x91, 0x58, 0x32, 0xCB),
  CRYPTO_MPI_LIMB_DATA4(0x64, 0xD8, 0x5F, 0xC6),
  CRYPTO_MPI_LIMB_DATA4(0x97, 0xED, 0x87, 0x45),
  CRYPTO_MPI_LIMB_DATA4(0xD0, 0x05, 0xB6, 0xCE),
  CRYPTO_MPI_LIMB_DATA4(0xC9, 0x1C, 0xE5, 0xA7),
  CRYPTO_MPI_LIMB_DATA4(0x21, 0x75, 0x6F, 0x89),
  CRYPTO_MPI_LIMB_DATA4(0x2B, 0xA1, 0x82, 0xDB),
  CRYPTO_MPI_LIMB_DATA4(0xCB, 0xA3, 0xAB, 0x29),
  CRYPTO_MPI_LIMB_DATA4(0xEA, 0xAA, 0x6C, 0x3A),
  CRYPTO_MPI_LIMB_DATA4(0xC2, 0x7E, 0xCA, 0x01),
  CRYPTO_MPI_LIMB_DATA4(0x30, 0x4E, 0xED, 0x75),
  CRYPTO_MPI_LIMB_DATA4(0x54, 0x8D, 0x6D, 0x4B),
  CRYPTO_MPI_LIMB_DATA4(0x1E, 0x3C, 0x04, 0xF1),
  CRYPTO_MPI_LIMB_DATA4(0x8A, 0x30, 0x95, 0xAF),
  CRYPTO_MPI_LIMB_DATA4(0x85, 0x63, 0x43, 0x7D),
  CRYPTO_MPI_LIMB_DATA4(0xE8, 0xE9, 0xD1, 0x87),
  CRYPTO_MPI_LIMB_DATA4(0x27, 0xDD, 0x59, 0x08),
  CRYPTO_MPI_LIMB_DATA4(0x0C, 0x99, 0xC8, 0x9D)
};

 const CRYPTO_DSA_PUBLIC_KEY SSH_ServerKeys_DSA_2048b_256b_PublicKey = {
  { CRYPTO_MPI_INIT_RO(SSH_ServerKeys_DSA_2048b_256b_Y_aLimbs) },
};

static const CRYPTO_MPI_LIMB SSH_ServerKeys_DSA_2048b_256b_X_aLimbs[] = {
  CRYPTO_MPI_LIMB_DATA4(0x9A, 0xE4, 0xCF, 0xF5),
  CRYPTO_MPI_LIMB_DATA4(0x70, 0xA6, 0x1E, 0x0A),
  CRYPTO_MPI_LIMB_DATA4(0xE5, 0x99, 0x5E, 0xB8),
  CRYPTO_MPI_LIMB_DATA4(0xDB, 0x03, 0x45, 0xAF),
  CRYPTO_MPI_LIMB_DATA4(0xC6, 0xC6, 0x27, 0x6F)
};

 const CRYPTO_DSA_PRIVATE_KEY SSH_ServerKeys_DSA_2048b_256b_PrivateKey = {
  { CRYPTO_MPI_INIT_RO(SSH_ServerKeys_DSA_2048b_256b_X_aLimbs) },
};

#include "CRYPTO.h"

const CRYPTO_MPI_LIMB SSH_ServerKeys_DSA_3072b_256b_P_aLimbs[] = {
  CRYPTO_MPI_LIMB_DATA4(0x7B, 0x19, 0x01, 0xDF),
  CRYPTO_MPI_LIMB_DATA4(0x79, 0xC6, 0x13, 0x90),
  CRYPTO_MPI_LIMB_DATA4(0x0C, 0x3F, 0xC6, 0x84),
```

```
CRYPTO_MPI_LIMB_DATA4(0x97, 0x68, 0xDD, 0xED),
CRYPTO_MPI_LIMB_DATA4(0x1A, 0x44, 0xA9, 0x1F),
CRYPTO_MPI_LIMB_DATA4(0xBB, 0xE2, 0x70, 0x73),
CRYPTO_MPI_LIMB_DATA4(0x21, 0x92, 0x05, 0x3E),
CRYPTO_MPI_LIMB_DATA4(0xCF, 0x12, 0x9C, 0x7A),
CRYPTO_MPI_LIMB_DATA4(0x0B, 0x15, 0x76, 0xD4),
CRYPTO_MPI_LIMB_DATA4(0x5D, 0xC8, 0x37, 0x9E),
CRYPTO_MPI_LIMB_DATA4(0xBD, 0xE2, 0x70, 0x73),
CRYPTO_MPI_LIMB_DATA4(0x21, 0x92, 0x05, 0x3E),
CRYPTO_MPI_LIMB_DATA4(0xCF, 0x12, 0x9C, 0x7A),
CRYPTO_MPI_LIMB_DATA4(0x0B, 0x15, 0x76, 0xD4),
CRYPTO_MPI_LIMB_DATA4(0x5D, 0xC8, 0x37, 0x9E),
CRYPTO_MPI_LIMB_DATA4(0xBE, 0xE2, 0x70, 0x73),
CRYPTO_MPI_LIMB_DATA4(0x21, 0x92, 0x05, 0x3E),
CRYPTO_MPI_LIMB_DATA4(0xCF, 0x12, 0x9C, 0x7A),
CRYPTO_MPI_LIMB_DATA4(0x0B, 0x15, 0x76, 0xD4),
CRYPTO_MPI_LIMB_DATA4(0x5D, 0xC8, 0x37, 0x9E),
CRYPTO_MPI_LIMB_DATA4(0xBF, 0xE2, 0x70, 0x73),
CRYPTO_MPI_LIMB_DATA4(0x21, 0x92, 0x05, 0x3E),
CRYPTO_MPI_LIMB_DATA4(0xCF, 0x12, 0x9C, 0x7A),
CRYPTO_MPI_LIMB_DATA4(0x0B, 0x15, 0x76, 0xD4),
CRYPTO_MPI_LIMB_DATA4(0x5D, 0xC8, 0x37, 0x9E),
CRYPTO_MPI_LIMB_DATA4(0xC0, 0xE2, 0x70, 0x73),
CRYPTO_MPI_LIMB_DATA4(0x21, 0x92, 0x05, 0x3E),
CRYPTO_MPI_LIMB_DATA4(0xCF, 0x12, 0x9C, 0x7A),
CRYPTO_MPI_LIMB_DATA4(0x0B, 0x15, 0x76, 0xD4),
CRYPTO_MPI_LIMB_DATA4(0x5D, 0xC8, 0x37, 0x9E),
CRYPTO_MPI_LIMB_DATA4(0xC1, 0xE2, 0x70, 0x73),
CRYPTO_MPI_LIMB_DATA4(0x21, 0x92, 0x05, 0x3E),
CRYPTO_MPI_LIMB_DATA4(0xCF, 0x12, 0x9C, 0x7A),
CRYPTO_MPI_LIMB_DATA4(0x0B, 0x15, 0x76, 0xD4),
CRYPTO_MPI_LIMB_DATA4(0x5D, 0xC8, 0x37, 0x9E),
CRYPTO_MPI_LIMB_DATA4(0xC2, 0xE2, 0x70, 0x73),
CRYPTO_MPI_LIMB_DATA4(0x21, 0x92, 0x05, 0x3E),
CRYPTO_MPI_LIMB_DATA4(0xCF, 0x12, 0x9C, 0x7A),
CRYPTO_MPI_LIMB_DATA4(0x0B, 0x15, 0x76, 0xD4),
CRYPTO_MPI_LIMB_DATA4(0x5D, 0xC8, 0x37, 0x9E),
CRYPTO_MPI_LIMB_DATA4(0xC3, 0xE2, 0x70, 0x73),
CRYPTO_MPI_LIMB_DATA4(0x21, 0x92, 0x05, 0x3E),
CRYPTO_MPI_LIMB_DATA4(0xCF, 0x12, 0x9C, 0x7A),
CRYPTO_MPI_LIMB_DATA4(0x0B, 0x15, 0x76, 0xD4),
CRYPTO_MPI_LIMB_DATA4(0x5D, 0xC8, 0x37, 0x9E),
CRYPTO_MPI_LIMB_DATA4(0xC4, 0xE2, 0x70, 0x73),
CRYPTO_MPI_LIMB_DATA4(0x21, 0x92, 0x05, 0x3E),
CRYPTO_MPI_LIMB_DATA4(0xCF, 0x12, 0x9C, 0x7A),
CRYPTO_MPI_LIMB_DATA4(0x0B, 0x15, 0x76, 0xD4),
CRYPTO_MPI_LIMB_DATA4(0x5D, 0xC8, 0x37, 0x9E),
CRYPTO_MPI_LIMB_DATA4(0xC5, 0xE2, 0x70, 0x73),
CRYPTO_MPI_LIMB_DATA4(0x21, 0x92, 0x05, 0x3E),
CRYPTO_MPI_LIMB_DATA4(0xCF, 0x12, 0x9C, 0x7A),
CRYPTO_MPI_LIMB_DATA4(0x0B, 0x15, 0x76, 0xD4),
CRYPTO_MPI_LIMB_DATA4(0x5D, 0xC8, 0x37, 0x9E),
CRYPTO_MPI_LIMB_DATA4(0xC6, 0xE2, 0x70, 0x73),
CRYPTO_MPI_LIMB_DATA4(0x21, 0x92, 0x05, 0x3E),
CRYPTO_MPI_LIMB_DATA4(0xCF, 0x12, 0x9C, 0x7A),
CRYPTO_MPI_LIMB_DATA4(0x0B, 0x15, 0x76, 0xD4),
CRYPTO_MPI_LIMB_DATA4(0x5D, 0xC8, 0x37, 0x9E),
CRYPTO_MPI_LIMB_DATA4(0xC7, 0xE2, 0x70, 0x73),
CRYPTO_MPI_LIMB_DATA4(0x21, 0x92, 0x05, 0x3E),
CRYPTO_MPI_LIMB_DATA4(0xCF, 0x12, 0x9C, 0x7A),
CRYPTO_MPI_LIMB_DATA4(0x0B, 0x15, 0x76, 0xD4),
CRYPTO_MPI_LIMB_DATA4(0x5D, 0xC8, 0x37, 0x9E),
CRYPTO_MPI_LIMB_DATA4(0xC8, 0xE2, 0x70, 0x73),
CRYPTO_MPI_LIMB_DATA4(0x21, 0x92, 0x05, 0x3E),
CRYPTO_MPI_LIMB_DATA4(0xCF, 0x12, 0x9C, 0x7A),
CRYPTO_MPI_LIMB_DATA4(0x0B, 0x15, 0x76, 0xD4),
CRYPTO_MPI_LIMB_DATA4(0x5D, 0xC8, 0x37, 0x9E),
CRYPTO_MPI_LIMB_DATA4(0xC9, 0xE2, 0x70, 0x73),
```

```
  CRYPTO_MPI_LIMB_DATA4(0x21, 0x92, 0x05, 0x3E),
  CRYPTO_MPI_LIMB_DATA4(0xCF, 0x12, 0x9C, 0x7A),
  CRYPTO_MPI_LIMB_DATA4(0x0B, 0x15, 0x76, 0xD4),
  CRYPTO_MPI_LIMB_DATA4(0x5D, 0xC8, 0x37, 0x9E),
  CRYPTO_MPI_LIMB_DATA4(0xCA, 0xE2, 0x70, 0x73),
  CRYPTO_MPI_LIMB_DATA4(0x21, 0x92, 0x05, 0x3E),
  CRYPTO_MPI_LIMB_DATA4(0xCF, 0x12, 0x9C, 0x7A),
  CRYPTO_MPI_LIMB_DATA4(0x0B, 0x15, 0x76, 0xD4),
  CRYPTO_MPI_LIMB_DATA4(0x5D, 0xC8, 0x37, 0x9E),
  CRYPTO_MPI_LIMB_DATA4(0xCB, 0xE2, 0x70, 0x73),
  CRYPTO_MPI_LIMB_DATA4(0x21, 0x92, 0x05, 0x3E),
  CRYPTO_MPI_LIMB_DATA4(0xCF, 0x12, 0x9C, 0x7A),
  CRYPTO_MPI_LIMB_DATA4(0x0B, 0x15, 0x76, 0xD4),
  CRYPTO_MPI_LIMB_DATA4(0x5D, 0xC8, 0x37, 0x9E),
  CRYPTO_MPI_LIMB_DATA4(0xCC, 0xE2, 0x70, 0x73),
  CRYPTO_MPI_LIMB_DATA4(0x21, 0x92, 0x05, 0x3E),
  CRYPTO_MPI_LIMB_DATA4(0xCF, 0x12, 0x9C, 0x7A),
  CRYPTO_MPI_LIMB_DATA4(0x0B, 0x15, 0x76, 0xD4),
  CRYPTO_MPI_LIMB_DATA4(0x5D, 0xC8, 0x37, 0x9E),
  CRYPTO_MPI_LIMB_DATA4(0xCD, 0xE2, 0x70, 0x73),
  CRYPTO_MPI_LIMB_DATA4(0x21, 0x92, 0x05, 0x3E),
  CRYPTO_MPI_LIMB_DATA4(0xCF, 0x12, 0x9C, 0x7A),
  CRYPTO_MPI_LIMB_DATA4(0x0B, 0x15, 0x76, 0xD4),
  CRYPTO_MPI_LIMB_DATA4(0x5D, 0xC8, 0x37, 0x9E),
  CRYPTO_MPI_LIMB_DATA4(0xCE, 0xE2, 0x70, 0xF3)
};

const CRYPTO_MPI_LIMB SSH_ServerKeys_DSA_3072b_256b_Q_aLimbs[] = {
  CRYPTO_MPI_LIMB_DATA4(0x5B, 0x83, 0x95, 0x0C),
  CRYPTO_MPI_LIMB_DATA4(0x1D, 0xE1, 0x0C, 0x80),
  CRYPTO_MPI_LIMB_DATA4(0xFE, 0x23, 0x36, 0x94),
  CRYPTO_MPI_LIMB_DATA4(0xD9, 0x1B, 0x06, 0x0F),
  CRYPTO_MPI_LIMB_DATA4(0x76, 0xEC, 0x86, 0xE7)
};

const CRYPTO_MPI_LIMB SSH_ServerKeys_DSA_3072b_256b_G_aLimbs[] = {
  CRYPTO_MPI_LIMB_DATA4(0xEA, 0xC0, 0xBB, 0x6F),
  CRYPTO_MPI_LIMB_DATA4(0x3E, 0xC6, 0x59, 0x61),
  CRYPTO_MPI_LIMB_DATA4(0x6E, 0x52, 0x77, 0x92),
  CRYPTO_MPI_LIMB_DATA4(0x7B, 0xE3, 0x88, 0x82),
  CRYPTO_MPI_LIMB_DATA4(0x98, 0xDC, 0xC7, 0x76),
  CRYPTO_MPI_LIMB_DATA4(0xD7, 0x4B, 0x1B, 0x56),
  CRYPTO_MPI_LIMB_DATA4(0x0C, 0x67, 0x1C, 0x2F),
  CRYPTO_MPI_LIMB_DATA4(0x93, 0x52, 0x8B, 0x75),
  CRYPTO_MPI_LIMB_DATA4(0xB1, 0xE0, 0x3A, 0xC8),
  CRYPTO_MPI_LIMB_DATA4(0xFD, 0x3F, 0x26, 0x1D),
  CRYPTO_MPI_LIMB_DATA4(0xC8, 0x54, 0x63, 0xF2),
  CRYPTO_MPI_LIMB_DATA4(0x3F, 0x49, 0xA4, 0x6A),
  CRYPTO_MPI_LIMB_DATA4(0x83, 0x2F, 0x52, 0x77),
  CRYPTO_MPI_LIMB_DATA4(0x6E, 0x1F, 0x99, 0x94),
  CRYPTO_MPI_LIMB_DATA4(0x35, 0x44, 0xAD, 0x8C),
  CRYPTO_MPI_LIMB_DATA4(0x28, 0x37, 0x9D, 0x36),
  CRYPTO_MPI_LIMB_DATA4(0x31, 0x18, 0xEB, 0x33),
  CRYPTO_MPI_LIMB_DATA4(0x41, 0x72, 0xC1, 0x48),
  CRYPTO_MPI_LIMB_DATA4(0xFC, 0xE6, 0x5F, 0x3F),
  CRYPTO_MPI_LIMB_DATA4(0x1E, 0xB3, 0x10, 0x8E),
  CRYPTO_MPI_LIMB_DATA4(0xAF, 0x3C, 0x65, 0xF8),
  CRYPTO_MPI_LIMB_DATA4(0x46, 0x27, 0x74, 0xED),
  CRYPTO_MPI_LIMB_DATA4(0x2F, 0x71, 0xDF, 0xC6),
  CRYPTO_MPI_LIMB_DATA4(0xA8, 0xED, 0x91, 0xE1),
  CRYPTO_MPI_LIMB_DATA4(0xF0, 0xA9, 0x35, 0xF9),
  CRYPTO_MPI_LIMB_DATA4(0x6A, 0x66, 0x28, 0x13),
  CRYPTO_MPI_LIMB_DATA4(0xDC, 0x16, 0xFA, 0xCF),
  CRYPTO_MPI_LIMB_DATA4(0xEF, 0xA4, 0xBC, 0x5E),
  CRYPTO_MPI_LIMB_DATA4(0x79, 0xD5, 0x5B, 0xBB),
  CRYPTO_MPI_LIMB_DATA4(0xE7, 0xD8, 0x51, 0xFB),
  CRYPTO_MPI_LIMB_DATA4(0xD4, 0xBD, 0x78, 0xD5),
  CRYPTO_MPI_LIMB_DATA4(0x00, 0xDB, 0x39, 0xA4),
```

```
    CRYPTO_MPI_LIMB_DATA4(0x95, 0x4B, 0x13, 0xD9),
    CRYPTO_MPI_LIMB_DATA4(0xF1, 0x7D, 0x42, 0x1E),
    CRYPTO_MPI_LIMB_DATA4(0x8F, 0xE2, 0x9A, 0xF9),
    CRYPTO_MPI_LIMB_DATA4(0x69, 0x68, 0x13, 0xC8),
    CRYPTO_MPI_LIMB_DATA4(0x42, 0x4B, 0x22, 0x40),
    CRYPTO_MPI_LIMB_DATA4(0xA8, 0x6A, 0xC9, 0x65),
    CRYPTO_MPI_LIMB_DATA4(0x31, 0xC0, 0xE7, 0xA8),
    CRYPTO_MPI_LIMB_DATA4(0x37, 0x36, 0xE2, 0xBD),
    CRYPTO_MPI_LIMB_DATA4(0x9E, 0x3D, 0x39, 0x6C),
    CRYPTO_MPI_LIMB_DATA4(0x80, 0xFB, 0xB6, 0xBB),
    CRYPTO_MPI_LIMB_DATA4(0xBF, 0xAC, 0x27, 0x7C),
    CRYPTO_MPI_LIMB_DATA4(0xAF, 0x43, 0xD1, 0xEF),
    CRYPTO_MPI_LIMB_DATA4(0xD6, 0x0C, 0xC3, 0xCD),
    CRYPTO_MPI_LIMB_DATA4(0x9A, 0x30, 0x20, 0x9A),
    CRYPTO_MPI_LIMB_DATA4(0x45, 0x00, 0x0D, 0x36),
    CRYPTO_MPI_LIMB_DATA4(0xC7, 0xA5, 0x5C, 0x05),
    CRYPTO_MPI_LIMB_DATA4(0xD1, 0x54, 0x53, 0xDA),
    CRYPTO_MPI_LIMB_DATA4(0x7E, 0xB5, 0xFC, 0x39),
    CRYPTO_MPI_LIMB_DATA4(0xDD, 0xD4, 0xA8, 0x67),
    CRYPTO_MPI_LIMB_DATA4(0xAE, 0x35, 0x2A, 0x68),
    CRYPTO_MPI_LIMB_DATA4(0xDD, 0x9B, 0x5F, 0x2C),
    CRYPTO_MPI_LIMB_DATA4(0xE7, 0xC3, 0x8E, 0x97),
    CRYPTO_MPI_LIMB_DATA4(0x53, 0x99, 0xFB, 0x46),
    CRYPTO_MPI_LIMB_DATA4(0x7B, 0xD9, 0xCF, 0x4C),
    CRYPTO_MPI_LIMB_DATA4(0xA6, 0xFD, 0x74, 0x27),
    CRYPTO_MPI_LIMB_DATA4(0x4D, 0x9F, 0x46, 0x46),
    CRYPTO_MPI_LIMB_DATA4(0x3A, 0x13, 0x18, 0x1C),
    CRYPTO_MPI_LIMB_DATA4(0x60, 0x71, 0x87, 0x90),
    CRYPTO_MPI_LIMB_DATA4(0x83, 0x99, 0xB4, 0x5A),
    CRYPTO_MPI_LIMB_DATA4(0xA7, 0x14, 0x0A, 0x4B),
    CRYPTO_MPI_LIMB_DATA4(0x42, 0xA4, 0xA2, 0x66),
    CRYPTO_MPI_LIMB_DATA4(0x0A, 0x72, 0x5C, 0xFB),
    CRYPTO_MPI_LIMB_DATA4(0xE7, 0x3E, 0x7B, 0x94),
    CRYPTO_MPI_LIMB_DATA4(0xFC, 0x4A, 0xE8, 0x48),
    CRYPTO_MPI_LIMB_DATA4(0x21, 0xB3, 0xF2, 0xD2),
    CRYPTO_MPI_LIMB_DATA4(0xFB, 0x71, 0x69, 0x37),
    CRYPTO_MPI_LIMB_DATA4(0x35, 0xC2, 0x61, 0x98),
    CRYPTO_MPI_LIMB_DATA4(0x67, 0x0C, 0x01, 0x9F),
    CRYPTO_MPI_LIMB_DATA4(0x16, 0x55, 0x16, 0x58),
    CRYPTO_MPI_LIMB_DATA4(0x4C, 0x33, 0x65, 0xF8),
    CRYPTO_MPI_LIMB_DATA4(0xCE, 0xB1, 0xEC, 0x9A),
    CRYPTO_MPI_LIMB_DATA4(0x67, 0x01, 0xFE, 0x01),
    CRYPTO_MPI_LIMB_DATA4(0xAC, 0x2C, 0xA8, 0x8D),
    CRYPTO_MPI_LIMB_DATA4(0x38, 0x47, 0x5D, 0xD0),
    CRYPTO_MPI_LIMB_DATA4(0xC6, 0x24, 0x30, 0xE5),
    CRYPTO_MPI_LIMB_DATA4(0xA8, 0xF8, 0x16, 0x49),
    CRYPTO_MPI_LIMB_DATA4(0xCD, 0x50, 0x0B, 0xC8),
    CRYPTO_MPI_LIMB_DATA4(0xFA, 0x73, 0xAC, 0x6E),
    CRYPTO_MPI_LIMB_DATA4(0x7A, 0x8C, 0xBF, 0x45),
    CRYPTO_MPI_LIMB_DATA4(0xE0, 0x71, 0x08, 0x91),
    CRYPTO_MPI_LIMB_DATA4(0xE2, 0x7F, 0x0D, 0xDD),
    CRYPTO_MPI_LIMB_DATA4(0x8E, 0xDB, 0x34, 0x04),
    CRYPTO_MPI_LIMB_DATA4(0xD1, 0x09, 0x67, 0x52),
    CRYPTO_MPI_LIMB_DATA4(0x49, 0x62, 0xA3, 0x7D),
    CRYPTO_MPI_LIMB_DATA4(0xD1, 0x06, 0x7C, 0xA8),
    CRYPTO_MPI_LIMB_DATA4(0x78, 0x1F, 0x47, 0x39),
    CRYPTO_MPI_LIMB_DATA4(0x1F, 0x0F, 0xD2, 0x06),
    CRYPTO_MPI_LIMB_DATA4(0xB4, 0x38, 0x91, 0x92),
    CRYPTO_MPI_LIMB_DATA4(0x68, 0x57, 0xAF, 0x0B),
    CRYPTO_MPI_LIMB_DATA4(0xAE, 0xEE, 0x12, 0x21),
    CRYPTO_MPI_LIMB_DATA4(0x76, 0x24, 0xB2, 0x20),
    CRYPTO_MPI_LIMB_DATA4(0xFF, 0x7C, 0xFE, 0x8F),
    CRYPTO_MPI_LIMB_DATA4(0xAE, 0x62, 0x17, 0xD6),
    CRYPTO_MPI_LIMB_DATA4(0x3E, 0x73, 0x02, 0xF2)
};

const CRYPTO_DSA_DOMAIN_PARAMS SSH_ServerKeys_DSA_3072b_256b_DomainParas = {
  { CRYPTO_MPI_INIT_RO(SSH_ServerKeys_DSA_3072b_256b_P_aLimbs) },
```

```
  { CRYPTO_MPI_INIT_RO(SSH_ServerKeys_DSA_3072b_256b_Q_aLimbs) },
  { CRYPTO_MPI_INIT_RO(SSH_ServerKeys_DSA_3072b_256b_G_aLimbs) },
};

const CRYPTO_MPI_LIMB SSH_ServerKeys_DSA_3072b_256b_Y_aLimbs[] = {
  CRYPTO_MPI_LIMB_DATA4(0x5B, 0xDC, 0x7E, 0x0E),
  CRYPTO_MPI_LIMB_DATA4(0x27, 0x0F, 0x78, 0x63),
  CRYPTO_MPI_LIMB_DATA4(0xE1, 0x60, 0x31, 0xBC),
  CRYPTO_MPI_LIMB_DATA4(0xAD, 0x9B, 0xAC, 0x73),
  CRYPTO_MPI_LIMB_DATA4(0x0C, 0x0F, 0xAE, 0xEA),
  CRYPTO_MPI_LIMB_DATA4(0xDA, 0xC8, 0x97, 0xE9),
  CRYPTO_MPI_LIMB_DATA4(0xA7, 0x0E, 0x2D, 0x89),
  CRYPTO_MPI_LIMB_DATA4(0xBF, 0x1D, 0x2B, 0xA8),
  CRYPTO_MPI_LIMB_DATA4(0xE4, 0x1E, 0x18, 0xEC),
  CRYPTO_MPI_LIMB_DATA4(0x09, 0xC3, 0x21, 0x92),
  CRYPTO_MPI_LIMB_DATA4(0x67, 0x81, 0x8A, 0x52),
  CRYPTO_MPI_LIMB_DATA4(0xF5, 0x65, 0xCF, 0x4A),
  CRYPTO_MPI_LIMB_DATA4(0x8E, 0xBB, 0x83, 0x5C),
  CRYPTO_MPI_LIMB_DATA4(0x71, 0xE5, 0x5C, 0xE1),
  CRYPTO_MPI_LIMB_DATA4(0xA3, 0x09, 0xC9, 0xC6),
  CRYPTO_MPI_LIMB_DATA4(0x77, 0xC0, 0xFF, 0x9F),
  CRYPTO_MPI_LIMB_DATA4(0xF0, 0x0E, 0x49, 0x06),
  CRYPTO_MPI_LIMB_DATA4(0x93, 0x29, 0x7F, 0x8B),
  CRYPTO_MPI_LIMB_DATA4(0x02, 0x7B, 0xA3, 0xA6),
  CRYPTO_MPI_LIMB_DATA4(0x5C, 0xDD, 0x9D, 0x6B),
  CRYPTO_MPI_LIMB_DATA4(0xA2, 0x14, 0xCD, 0x5B),
  CRYPTO_MPI_LIMB_DATA4(0x47, 0x79, 0x45, 0x95),
  CRYPTO_MPI_LIMB_DATA4(0xBB, 0x7D, 0xD6, 0xBF),
  CRYPTO_MPI_LIMB_DATA4(0x57, 0x48, 0x26, 0x27),
  CRYPTO_MPI_LIMB_DATA4(0x30, 0xC2, 0x14, 0xFC),
  CRYPTO_MPI_LIMB_DATA4(0xAE, 0x22, 0x8A, 0xAB),
  CRYPTO_MPI_LIMB_DATA4(0x9D, 0x69, 0x6E, 0xCA),
  CRYPTO_MPI_LIMB_DATA4(0xCB, 0x06, 0x96, 0x25),
  CRYPTO_MPI_LIMB_DATA4(0x6B, 0x10, 0x1D, 0x82),
  CRYPTO_MPI_LIMB_DATA4(0x55, 0x46, 0x9C, 0x03),
  CRYPTO_MPI_LIMB_DATA4(0xFF, 0xFC, 0x5B, 0x52),
  CRYPTO_MPI_LIMB_DATA4(0x6A, 0x31, 0x0F, 0x15),
  CRYPTO_MPI_LIMB_DATA4(0x7D, 0x29, 0x5A, 0x11),
  CRYPTO_MPI_LIMB_DATA4(0x33, 0x11, 0xA3, 0x1B),
  CRYPTO_MPI_LIMB_DATA4(0x11, 0xBC, 0xC7, 0xB6),
  CRYPTO_MPI_LIMB_DATA4(0x91, 0xC0, 0xBF, 0x6F),
  CRYPTO_MPI_LIMB_DATA4(0x7C, 0xAE, 0xB5, 0x26),
  CRYPTO_MPI_LIMB_DATA4(0x85, 0x85, 0x14, 0xF7),
  CRYPTO_MPI_LIMB_DATA4(0x2E, 0x88, 0xC0, 0x67),
  CRYPTO_MPI_LIMB_DATA4(0x5B, 0x5F, 0x13, 0xF6),
  CRYPTO_MPI_LIMB_DATA4(0x8A, 0xCF, 0xC2, 0x79),
  CRYPTO_MPI_LIMB_DATA4(0x6C, 0x95, 0xEB, 0x1D),
  CRYPTO_MPI_LIMB_DATA4(0xD0, 0xA5, 0x11, 0xD3),
  CRYPTO_MPI_LIMB_DATA4(0x26, 0x46, 0x39, 0x40),
  CRYPTO_MPI_LIMB_DATA4(0x36, 0xFB, 0x23, 0x6B),
  CRYPTO_MPI_LIMB_DATA4(0x49, 0xD8, 0x41, 0xD5),
  CRYPTO_MPI_LIMB_DATA4(0x32, 0xD0, 0x04, 0xD3),
  CRYPTO_MPI_LIMB_DATA4(0x28, 0x8D, 0x79, 0x3B),
  CRYPTO_MPI_LIMB_DATA4(0x79, 0xD2, 0x15, 0x45),
  CRYPTO_MPI_LIMB_DATA4(0xA9, 0x8E, 0x8F, 0xBB),
  CRYPTO_MPI_LIMB_DATA4(0xBB, 0xC5, 0x1C, 0xD5),
  CRYPTO_MPI_LIMB_DATA4(0x94, 0x8E, 0x8C, 0x67),
  CRYPTO_MPI_LIMB_DATA4(0x80, 0x06, 0xDA, 0xFA),
  CRYPTO_MPI_LIMB_DATA4(0xDD, 0xFE, 0x9B, 0x0D),
  CRYPTO_MPI_LIMB_DATA4(0x47, 0x09, 0x6B, 0x52),
  CRYPTO_MPI_LIMB_DATA4(0xAB, 0x6D, 0xD6, 0x4F),
  CRYPTO_MPI_LIMB_DATA4(0x0E, 0x53, 0xB1, 0x23),
  CRYPTO_MPI_LIMB_DATA4(0xE4, 0xD5, 0xEE, 0x34),
  CRYPTO_MPI_LIMB_DATA4(0xFF, 0xC1, 0x36, 0x1C),
  CRYPTO_MPI_LIMB_DATA4(0x8C, 0x10, 0xB6, 0xA4),
  CRYPTO_MPI_LIMB_DATA4(0xE2, 0x1E, 0x4C, 0xC9),
  CRYPTO_MPI_LIMB_DATA4(0xB3, 0xEF, 0xE6, 0x17),
  CRYPTO_MPI_LIMB_DATA4(0x31, 0x63, 0x3D, 0x34),
```

```
   CRYPTO_MPI_LIMB_DATA4(0x3F, 0x1C, 0xDF, 0xA0),
   CRYPTO_MPI_LIMB_DATA4(0xFD, 0x68, 0xCB, 0x34),
   CRYPTO_MPI_LIMB_DATA4(0x1D, 0xDB, 0x6B, 0x6B),
   CRYPTO_MPI_LIMB_DATA4(0x1F, 0x0E, 0x61, 0x97),
   CRYPTO_MPI_LIMB_DATA4(0x78, 0x87, 0x61, 0x58),
   CRYPTO_MPI_LIMB_DATA4(0xE6, 0x8A, 0x91, 0xBC),
   CRYPTO_MPI_LIMB_DATA4(0x13, 0x6D, 0x7C, 0x89),
   CRYPTO_MPI_LIMB_DATA4(0x74, 0x7A, 0x73, 0x9D),
   CRYPTO_MPI_LIMB_DATA4(0x05, 0xA2, 0x49, 0x29),
   CRYPTO_MPI_LIMB_DATA4(0x84, 0x0F, 0xDE, 0xC1),
   CRYPTO_MPI_LIMB_DATA4(0xA7, 0x9A, 0xD7, 0xAC),
   CRYPTO_MPI_LIMB_DATA4(0xDD, 0xC7, 0xC6, 0xE3),
   CRYPTO_MPI_LIMB_DATA4(0x18, 0x46, 0x28, 0xF8),
   CRYPTO_MPI_LIMB_DATA4(0xD8, 0xCD, 0xD2, 0xC7),
   CRYPTO_MPI_LIMB_DATA4(0x23, 0x3B, 0x36, 0x57),
   CRYPTO_MPI_LIMB_DATA4(0xB9, 0x3D, 0x68, 0x67),
   CRYPTO_MPI_LIMB_DATA4(0x8F, 0x95, 0x0F, 0x7D),
   CRYPTO_MPI_LIMB_DATA4(0xAD, 0xBC, 0x05, 0xE7),
   CRYPTO_MPI_LIMB_DATA4(0x14, 0xFF, 0x31, 0xBD),
   CRYPTO_MPI_LIMB_DATA4(0x37, 0x5F, 0xBA, 0x82),
   CRYPTO_MPI_LIMB_DATA4(0xE2, 0xA4, 0x08, 0x2F),
   CRYPTO_MPI_LIMB_DATA4(0x84, 0x40, 0x84, 0x80),
   CRYPTO_MPI_LIMB_DATA4(0x7F, 0xBC, 0xEA, 0xA3),
   CRYPTO_MPI_LIMB_DATA4(0x4F, 0x80, 0xDE, 0x29),
   CRYPTO_MPI_LIMB_DATA4(0xAA, 0x6A, 0xF7, 0xA1),
   CRYPTO_MPI_LIMB_DATA4(0xFB, 0xF6, 0x0E, 0x59),
   CRYPTO_MPI_LIMB_DATA4(0x2C, 0x9F, 0x40, 0x16),
   CRYPTO_MPI_LIMB_DATA4(0x56, 0x8B, 0x8C, 0xEE),
   CRYPTO_MPI_LIMB_DATA4(0x4F, 0x56, 0xBC, 0x99),
   CRYPTO_MPI_LIMB_DATA4(0x5E, 0xFB, 0x71, 0xAD),
   CRYPTO_MPI_LIMB_DATA4(0x16, 0x33, 0x65, 0xAA),
   CRYPTO_MPI_LIMB_DATA4(0x07, 0x7C, 0x49, 0x89),
   CRYPTO_MPI_LIMB_DATA4(0x2A, 0x98, 0xB9, 0x22)
};

 const CRYPTO_DSA_PUBLIC_KEY SSH_ServerKeys_DSA_3072b_256b_PublicKey = {
   { CRYPTO_MPI_INIT_RO(SSH_ServerKeys_DSA_3072b_256b_Y_aLimbs) },
};

static const CRYPTO_MPI_LIMB SSH_ServerKeys_DSA_3072b_256b_X_aLimbs[] = {
   CRYPTO_MPI_LIMB_DATA4(0xDD, 0x41, 0x4A, 0x34),
   CRYPTO_MPI_LIMB_DATA4(0xAB, 0xD7, 0x86, 0x17),
   CRYPTO_MPI_LIMB_DATA4(0x38, 0x77, 0x72, 0x49),
   CRYPTO_MPI_LIMB_DATA4(0xDA, 0x39, 0xE5, 0x05),
   CRYPTO_MPI_LIMB_DATA4(0x80, 0x67, 0x42, 0x43)
};

 const CRYPTO_DSA_PRIVATE_KEY SSH_ServerKeys_DSA_3072b_256b_PrivateKey = {
   { CRYPTO_MPI_INIT_RO(SSH_ServerKeys_DSA_3072b_256b_X_aLimbs) },
};
```

4.7.3   Generated ECDSA keys

```c
#include "CRYPTO.h"

const CRYPTO_MPI_LIMB SSH_ServerKeys_ECDSA_P256_PublicKey_YX_aLimbs[] = {
  CRYPTO_MPI_LIMB_DATA4(0x13, 0x89, 0x14, 0x23),
  CRYPTO_MPI_LIMB_DATA4(0x51, 0x73, 0x37, 0x9F),
  CRYPTO_MPI_LIMB_DATA4(0xA0, 0x02, 0x14, 0xAE),
  CRYPTO_MPI_LIMB_DATA4(0xFE, 0x37, 0xBB, 0xC6),
  CRYPTO_MPI_LIMB_DATA4(0xE2, 0xA5, 0x73, 0x67),
  CRYPTO_MPI_LIMB_DATA4(0xA4, 0x79, 0x7B, 0xA0),
  CRYPTO_MPI_LIMB_DATA4(0x2B, 0xB8, 0x20, 0xDA),
  CRYPTO_MPI_LIMB_DATA4(0xC9, 0xD0, 0x02, 0x6C)
};

const CRYPTO_MPI_LIMB SSH_ServerKeys_ECDSA_P256_PublicKey_YY_aLimbs[] = {
  CRYPTO_MPI_LIMB_DATA4(0x24, 0xC1, 0xF6, 0x74),
  CRYPTO_MPI_LIMB_DATA4(0x80, 0x9D, 0x1F, 0xC1),
  CRYPTO_MPI_LIMB_DATA4(0x51, 0x39, 0xF0, 0x47),
  CRYPTO_MPI_LIMB_DATA4(0x0D, 0x4A, 0x3B, 0x5F),
  CRYPTO_MPI_LIMB_DATA4(0x2E, 0x77, 0x48, 0x58),
  CRYPTO_MPI_LIMB_DATA4(0x57, 0x8B, 0xDA, 0x1B),
  CRYPTO_MPI_LIMB_DATA4(0xDD, 0xF1, 0x50, 0x80),
  CRYPTO_MPI_LIMB_DATA4(0xB8, 0x92, 0x93, 0xD7)
};

const CRYPTO_ECDSA_PUBLIC_KEY SSH_ServerKeys_ECDSA_P256_PublicKey = { {
  { CRYPTO_MPI_INIT_RO(SSH_ServerKeys_ECDSA_P256_PublicKey_YX_aLimbs) },
  { CRYPTO_MPI_INIT_RO(SSH_ServerKeys_ECDSA_P256_PublicKey_YY_aLimbs) },
  { CRYPTO_MPI_INIT_RO_ZERO },
  { CRYPTO_MPI_INIT_RO_ZERO },
  },
  &CRYPTO_EC_CURVE_secp256r1
};

const CRYPTO_MPI_LIMB SSH_ServerKeys_ECDSA_P256_PrivateKey_X_aLimbs[] = {
  CRYPTO_MPI_LIMB_DATA4(0x22, 0xC9, 0xFA, 0xDC),
  CRYPTO_MPI_LIMB_DATA4(0xD2, 0x37, 0x10, 0x5E),
  CRYPTO_MPI_LIMB_DATA4(0x0E, 0x79, 0x48, 0x24),
  CRYPTO_MPI_LIMB_DATA4(0x22, 0x4D, 0x95, 0x79),
  CRYPTO_MPI_LIMB_DATA4(0x85, 0xA3, 0x9F, 0x37),
  CRYPTO_MPI_LIMB_DATA4(0xA1, 0x2B, 0x48, 0x7C),
  CRYPTO_MPI_LIMB_DATA4(0x2D, 0xA1, 0x23, 0xC7),
  CRYPTO_MPI_LIMB_DATA4(0x76, 0x1E, 0x52, 0x61)
};

const CRYPTO_ECDSA_PRIVATE_KEY SSH_ServerKeys_ECDSA_P256_PrivateKey = {
  { CRYPTO_MPI_INIT_RO(SSH_ServerKeys_ECDSA_P256_PrivateKey_X_aLimbs) },
  &CRYPTO_EC_CURVE_secp256r1
};

#include "CRYPTO.h"

const CRYPTO_MPI_LIMB SSH_ServerKeys_ECDSA_P384_PublicKey_YX_aLimbs[] = {
  CRYPTO_MPI_LIMB_DATA4(0x40, 0x20, 0x32, 0xF1),
  CRYPTO_MPI_LIMB_DATA4(0x6E, 0xFD, 0xB1, 0x3A),
  CRYPTO_MPI_LIMB_DATA4(0xDD, 0x08, 0xD9, 0x3C),
  CRYPTO_MPI_LIMB_DATA4(0x33, 0x3B, 0xC5, 0xE7),
  CRYPTO_MPI_LIMB_DATA4(0xC4, 0xF5, 0x4B, 0x48),
  CRYPTO_MPI_LIMB_DATA4(0xF8, 0xE6, 0x9B, 0x65),
  CRYPTO_MPI_LIMB_DATA4(0xFD, 0x84, 0xA6, 0x2C),
  CRYPTO_MPI_LIMB_DATA4(0xD0, 0x9B, 0x5C, 0x82),
  CRYPTO_MPI_LIMB_DATA4(0x4A, 0xFE, 0x3A, 0xB6),
  CRYPTO_MPI_LIMB_DATA4(0x05, 0x13, 0x1D, 0xE6),
  CRYPTO_MPI_LIMB_DATA4(0x32, 0x89, 0x88, 0x96),
  CRYPTO_MPI_LIMB_DATA4(0x50, 0x4A, 0xF1, 0x0E)
};
```

```
const CRYPTO_MPI_LIMB SSH_ServerKeys_ECDSA_P384_PublicKey_YY_aLimbs[] = {
  CRYPTO_MPI_LIMB_DATA4(0x52, 0x80, 0xFB, 0x08),
  CRYPTO_MPI_LIMB_DATA4(0x23, 0x9B, 0x1B, 0x9D),
  CRYPTO_MPI_LIMB_DATA4(0x13, 0x76, 0x28, 0xBA),
  CRYPTO_MPI_LIMB_DATA4(0x6D, 0xEB, 0xFC, 0x22),
  CRYPTO_MPI_LIMB_DATA4(0xA0, 0x28, 0xA6, 0xCF),
  CRYPTO_MPI_LIMB_DATA4(0xBE, 0x70, 0xAA, 0x7D),
  CRYPTO_MPI_LIMB_DATA4(0x28, 0x02, 0x5A, 0x80),
  CRYPTO_MPI_LIMB_DATA4(0x9A, 0x73, 0x67, 0xD5),
  CRYPTO_MPI_LIMB_DATA4(0x36, 0xDC, 0xD9, 0xD6),
  CRYPTO_MPI_LIMB_DATA4(0x18, 0xDF, 0xB9, 0x85),
  CRYPTO_MPI_LIMB_DATA4(0x0D, 0xBF, 0xAE, 0x0B),
  CRYPTO_MPI_LIMB_DATA4(0x86, 0xEF, 0xB4, 0x29)
};

const CRYPTO_ECDSA_PUBLIC_KEY SSH_ServerKeys_ECDSA_P384_PublicKey = { {
  { CRYPTO_MPI_INIT_RO(SSH_ServerKeys_ECDSA_P384_PublicKey_YX_aLimbs) },
  { CRYPTO_MPI_INIT_RO(SSH_ServerKeys_ECDSA_P384_PublicKey_YY_aLimbs) },
  { CRYPTO_MPI_INIT_RO_ZERO },
  { CRYPTO_MPI_INIT_RO_ZERO },
  },
  &CRYPTO_EC_CURVE_secp384r1
};

const CRYPTO_MPI_LIMB SSH_ServerKeys_ECDSA_P384_PrivateKey_X_aLimbs[] = {
  CRYPTO_MPI_LIMB_DATA4(0xA6, 0x89, 0xF6, 0xF7),
  CRYPTO_MPI_LIMB_DATA4(0x14, 0x18, 0x04, 0x12),
  CRYPTO_MPI_LIMB_DATA4(0xAC, 0xA6, 0x08, 0xE4),
  CRYPTO_MPI_LIMB_DATA4(0xE3, 0x72, 0x4D, 0x8C),
  CRYPTO_MPI_LIMB_DATA4(0xD5, 0x42, 0x7D, 0x1D),
  CRYPTO_MPI_LIMB_DATA4(0x9B, 0x9F, 0x9F, 0xC1),
  CRYPTO_MPI_LIMB_DATA4(0x99, 0x8E, 0xEC, 0x61),
  CRYPTO_MPI_LIMB_DATA4(0x34, 0x5B, 0x4E, 0xA1),
  CRYPTO_MPI_LIMB_DATA4(0xBA, 0x35, 0xC4, 0xD6),
  CRYPTO_MPI_LIMB_DATA4(0x76, 0x65, 0xD8, 0xFF),
  CRYPTO_MPI_LIMB_DATA4(0xB3, 0x9F, 0x87, 0x18),
  CRYPTO_MPI_LIMB_DATA4(0xF5, 0x6A, 0x7A, 0xB6)
};

const CRYPTO_ECDSA_PRIVATE_KEY SSH_ServerKeys_ECDSA_P384_PrivateKey = {
  { CRYPTO_MPI_INIT_RO(SSH_ServerKeys_ECDSA_P384_PrivateKey_X_aLimbs) },
  &CRYPTO_EC_CURVE_secp384r1
};

#include "CRYPTO.h"

const CRYPTO_MPI_LIMB SSH_ServerKeys_ECDSA_P521_PublicKey_YX_aLimbs[] = {
  CRYPTO_MPI_LIMB_DATA4(0x77, 0x5B, 0x00, 0x64),
  CRYPTO_MPI_LIMB_DATA4(0xD4, 0xF7, 0xFF, 0xDF),
  CRYPTO_MPI_LIMB_DATA4(0x16, 0x94, 0x8A, 0x13),
  CRYPTO_MPI_LIMB_DATA4(0x5B, 0x0E, 0x91, 0x0D),
  CRYPTO_MPI_LIMB_DATA4(0x76, 0xEC, 0x6A, 0x1F),
  CRYPTO_MPI_LIMB_DATA4(0xB2, 0x11, 0x54, 0x79),
  CRYPTO_MPI_LIMB_DATA4(0x5B, 0xC2, 0xCB, 0xAA),
  CRYPTO_MPI_LIMB_DATA4(0xD6, 0x96, 0x84, 0xFB),
  CRYPTO_MPI_LIMB_DATA4(0x05, 0x21, 0xB1, 0xEE),
  CRYPTO_MPI_LIMB_DATA4(0x34, 0x32, 0x4E, 0xEB),
  CRYPTO_MPI_LIMB_DATA4(0x20, 0x13, 0x22, 0x21),
  CRYPTO_MPI_LIMB_DATA4(0xE7, 0x81, 0x7A, 0xEA),
  CRYPTO_MPI_LIMB_DATA4(0xAB, 0x41, 0xFC, 0x9A),
  CRYPTO_MPI_LIMB_DATA4(0x13, 0x1B, 0xAC, 0xC4),
  CRYPTO_MPI_LIMB_DATA4(0xD4, 0xF0, 0x18, 0xEC),
  CRYPTO_MPI_LIMB_DATA4(0xE2, 0x87, 0x70, 0xD7),
  CRYPTO_MPI_LIMB_DATA2(0xFC, 0x01)
};

const CRYPTO_MPI_LIMB SSH_ServerKeys_ECDSA_P521_PublicKey_YY_aLimbs[] = {
```

```
  CRYPTO_MPI_LIMB_DATA4(0xAB, 0xAD, 0x3D, 0x22),
  CRYPTO_MPI_LIMB_DATA4(0x9B, 0x2B, 0x64, 0xA7),
  CRYPTO_MPI_LIMB_DATA4(0xB9, 0x15, 0x14, 0x43),
  CRYPTO_MPI_LIMB_DATA4(0x29, 0xBF, 0x68, 0x90),
  CRYPTO_MPI_LIMB_DATA4(0xF7, 0x9C, 0x7C, 0x9E),
  CRYPTO_MPI_LIMB_DATA4(0x85, 0xAB, 0xDE, 0xDB),
  CRYPTO_MPI_LIMB_DATA4(0x62, 0xE2, 0x85, 0x40),
  CRYPTO_MPI_LIMB_DATA4(0x6A, 0x6C, 0x09, 0x38),
  CRYPTO_MPI_LIMB_DATA4(0x22, 0x9C, 0x7B, 0xA1),
  CRYPTO_MPI_LIMB_DATA4(0x28, 0x30, 0x01, 0x1D),
  CRYPTO_MPI_LIMB_DATA4(0x81, 0x52, 0x39, 0x44),
  CRYPTO_MPI_LIMB_DATA4(0xC9, 0x65, 0x66, 0x1A),
  CRYPTO_MPI_LIMB_DATA4(0xF6, 0x66, 0xA0, 0xC6),
  CRYPTO_MPI_LIMB_DATA4(0xFE, 0x41, 0xD4, 0x64),
  CRYPTO_MPI_LIMB_DATA4(0x90, 0x3B, 0xE0, 0x6C),
  CRYPTO_MPI_LIMB_DATA4(0x98, 0x2C, 0xCC, 0xA8),
  CRYPTO_MPI_LIMB_DATA2(0x31, 0x01)
};

const CRYPTO_ECDSA_PUBLIC_KEY SSH_ServerKeys_ECDSA_P521_PublicKey = { {
  { CRYPTO_MPI_INIT_RO(SSH_ServerKeys_ECDSA_P521_PublicKey_YX_aLimbs) },
  { CRYPTO_MPI_INIT_RO(SSH_ServerKeys_ECDSA_P521_PublicKey_YY_aLimbs) },
  { CRYPTO_MPI_INIT_RO_ZERO },
  { CRYPTO_MPI_INIT_RO_ZERO },
  },
  &CRYPTO_EC_CURVE_secp521r1
};

const CRYPTO_MPI_LIMB SSH_ServerKeys_ECDSA_P521_PrivateKey_X_aLimbs[] = {
  CRYPTO_MPI_LIMB_DATA4(0x20, 0x53, 0x38, 0x35),
  CRYPTO_MPI_LIMB_DATA4(0xD6, 0x48, 0x33, 0xD0),
  CRYPTO_MPI_LIMB_DATA4(0x4B, 0xC4, 0xD7, 0xF8),
  CRYPTO_MPI_LIMB_DATA4(0x5F, 0xB3, 0x82, 0xB0),
  CRYPTO_MPI_LIMB_DATA4(0x11, 0xAE, 0xBB, 0xD1),
  CRYPTO_MPI_LIMB_DATA4(0xA9, 0x44, 0x5A, 0xEA),
  CRYPTO_MPI_LIMB_DATA4(0x9A, 0xFB, 0xC7, 0x02),
  CRYPTO_MPI_LIMB_DATA4(0xC1, 0xB0, 0xB8, 0x01),
  CRYPTO_MPI_LIMB_DATA4(0x51, 0x71, 0x1C, 0xCF),
  CRYPTO_MPI_LIMB_DATA4(0x04, 0xB1, 0xCC, 0x06),
  CRYPTO_MPI_LIMB_DATA4(0x7B, 0xA8, 0x10, 0x0B),
  CRYPTO_MPI_LIMB_DATA4(0x3F, 0x23, 0xD7, 0xF8),
  CRYPTO_MPI_LIMB_DATA4(0xA6, 0x09, 0xB4, 0xEA),
  CRYPTO_MPI_LIMB_DATA4(0xD1, 0xD4, 0x5A, 0x1A),
  CRYPTO_MPI_LIMB_DATA4(0x4D, 0x68, 0x30, 0xBF),
  CRYPTO_MPI_LIMB_DATA4(0xFC, 0x9C, 0x33, 0xC5),
  CRYPTO_MPI_LIMB_DATA2(0xC3, 0x01)
};

const CRYPTO_ECDSA_PRIVATE_KEY SSH_ServerKeys_ECDSA_P521_PrivateKey = {
  { CRYPTO_MPI_INIT_RO(SSH_ServerKeys_ECDSA_P521_PrivateKey_X_aLimbs) },
  &CRYPTO_EC_CURVE_secp521r1
};
```

4.7.4   Generated EdDSA keys

```c
#include "CRYPTO.h"

static const CRYPTO_MPI_LIMB SSH_ServerKeys_Ed25519_Y_aLimbs[] = {
  CRYPTO_MPI_LIMB_DATA4(0xBF, 0x72, 0x98, 0xB7),
  CRYPTO_MPI_LIMB_DATA4(0x3C, 0xCC, 0x95, 0x28),
  CRYPTO_MPI_LIMB_DATA4(0x7D, 0x8F, 0xBE, 0x05),
  CRYPTO_MPI_LIMB_DATA4(0x26, 0x2A, 0xA5, 0x07),
  CRYPTO_MPI_LIMB_DATA4(0xFF, 0x11, 0xD7, 0xF1),
  CRYPTO_MPI_LIMB_DATA4(0xE3, 0x48, 0xFF, 0x15),
  CRYPTO_MPI_LIMB_DATA4(0xEE, 0x2B, 0xF4, 0x7E),
  CRYPTO_MPI_LIMB_DATA4(0xEE, 0x0E, 0xC6, 0x82)
};

const CRYPTO_EdDSA_PUBLIC_KEY SSH_ServerKeys_Ed25519_PublicKey = {
  { CRYPTO_MPI_INIT_RO(SSH_ServerKeys_Ed25519_Y_aLimbs) },
};

static const CRYPTO_MPI_LIMB SSH_ServerKeys_Ed25519_SK_aLimbs[] = {
  CRYPTO_MPI_LIMB_DATA4(0x38, 0x8E, 0xA6, 0x49),
  CRYPTO_MPI_LIMB_DATA4(0xCA, 0xEB, 0x0A, 0x91),
  CRYPTO_MPI_LIMB_DATA4(0xD7, 0x30, 0x76, 0x87),
  CRYPTO_MPI_LIMB_DATA4(0x9B, 0x6D, 0xAD, 0x7C),
  CRYPTO_MPI_LIMB_DATA4(0x72, 0xA6, 0xF7, 0xE2),
  CRYPTO_MPI_LIMB_DATA4(0xBC, 0x8F, 0x10, 0xD3),
  CRYPTO_MPI_LIMB_DATA4(0xBE, 0x0C, 0x4C, 0xD9),
  CRYPTO_MPI_LIMB_DATA4(0x47, 0x8B, 0x53, 0x4C)
};

static const CRYPTO_MPI_LIMB SSH_ServerKeys_Ed25519_PK_aLimbs[] = {
  CRYPTO_MPI_LIMB_DATA4(0xBF, 0x72, 0x98, 0xB7),
  CRYPTO_MPI_LIMB_DATA4(0x3C, 0xCC, 0x95, 0x28),
  CRYPTO_MPI_LIMB_DATA4(0x7D, 0x8F, 0xBE, 0x05),
  CRYPTO_MPI_LIMB_DATA4(0x26, 0x2A, 0xA5, 0x07),
  CRYPTO_MPI_LIMB_DATA4(0xFF, 0x11, 0xD7, 0xF1),
  CRYPTO_MPI_LIMB_DATA4(0xE3, 0x48, 0xFF, 0x15),
  CRYPTO_MPI_LIMB_DATA4(0xEE, 0x2B, 0xF4, 0x7E),
  CRYPTO_MPI_LIMB_DATA4(0xEE, 0x0E, 0xC6, 0x82)
};

const CRYPTO_EdDSA_PRIVATE_KEY SSH_ServerKeys_Ed25519_PrivateKey = {
  { CRYPTO_MPI_INIT_RO(SSH_ServerKeys_Ed25519_SK_aLimbs) },
  { CRYPTO_MPI_INIT_RO(SSH_ServerKeys_Ed25519_PK_aLimbs) },
};

#include "CRYPTO.h"

static const CRYPTO_MPI_LIMB SSH_ServerKeys_Ed448_Y_aLimbs[] = {
  CRYPTO_MPI_LIMB_DATA4(0x83, 0x3F, 0x33, 0xA0),
  CRYPTO_MPI_LIMB_DATA4(0xB3, 0x41, 0xEF, 0xB1),
  CRYPTO_MPI_LIMB_DATA4(0xC2, 0xF3, 0x6E, 0xC9),
  CRYPTO_MPI_LIMB_DATA4(0x24, 0x74, 0x4A, 0x93),
  CRYPTO_MPI_LIMB_DATA4(0xE0, 0xB3, 0x92, 0x83),
  CRYPTO_MPI_LIMB_DATA4(0x5B, 0xE5, 0x20, 0x01),
  CRYPTO_MPI_LIMB_DATA4(0x09, 0xD3, 0x08, 0x0E),
  CRYPTO_MPI_LIMB_DATA4(0x55, 0x37, 0x7F, 0x8B),
  CRYPTO_MPI_LIMB_DATA4(0x4A, 0x47, 0x19, 0x1C),
  CRYPTO_MPI_LIMB_DATA4(0xAC, 0xD8, 0x2A, 0xF6),
  CRYPTO_MPI_LIMB_DATA4(0x33, 0xE1, 0xDC, 0xAD),
  CRYPTO_MPI_LIMB_DATA4(0xFE, 0x1A, 0x23, 0x49),
  CRYPTO_MPI_LIMB_DATA4(0x7B, 0x0D, 0x86, 0x84),
  CRYPTO_MPI_LIMB_DATA4(0x77, 0x04, 0xE0, 0xD4)
};

const CRYPTO_EdDSA_PUBLIC_KEY SSH_ServerKeys_Ed448_PublicKey = {
  { CRYPTO_MPI_INIT_RO(SSH_ServerKeys_Ed448_Y_aLimbs) },
```

```
};

static const CRYPTO_MPI_LIMB SSH_ServerKeys_Ed448_SK_aLimbs[] = {
  CRYPTO_MPI_LIMB_DATA4(0x14, 0x23, 0xCC, 0x3C),
  CRYPTO_MPI_LIMB_DATA4(0x95, 0xAB, 0x17, 0xE1),
  CRYPTO_MPI_LIMB_DATA4(0x56, 0x57, 0x30, 0xAE),
  CRYPTO_MPI_LIMB_DATA4(0x39, 0x68, 0x3E, 0xBD),
  CRYPTO_MPI_LIMB_DATA4(0xF8, 0x77, 0xB3, 0xB9),
  CRYPTO_MPI_LIMB_DATA4(0xAF, 0x8A, 0x44, 0x8B),
  CRYPTO_MPI_LIMB_DATA4(0x06, 0x89, 0xA7, 0xBF),
  CRYPTO_MPI_LIMB_DATA4(0xCE, 0xD6, 0xC3, 0x01),
  CRYPTO_MPI_LIMB_DATA4(0xBE, 0x96, 0x17, 0x0A),
  CRYPTO_MPI_LIMB_DATA4(0x91, 0xF9, 0xDA, 0x91),
  CRYPTO_MPI_LIMB_DATA4(0x07, 0xBE, 0xCD, 0x7E),
  CRYPTO_MPI_LIMB_DATA4(0x29, 0xC5, 0x6B, 0xE4),
  CRYPTO_MPI_LIMB_DATA4(0x39, 0x8E, 0x88, 0x1B),
  CRYPTO_MPI_LIMB_DATA4(0x04, 0x3D, 0xC9, 0xA8)
};

static const CRYPTO_MPI_LIMB SSH_ServerKeys_Ed448_PK_aLimbs[] = {
  CRYPTO_MPI_LIMB_DATA4(0x83, 0x3F, 0x33, 0xA0),
  CRYPTO_MPI_LIMB_DATA4(0xB3, 0x41, 0xEF, 0xB1),
  CRYPTO_MPI_LIMB_DATA4(0xC2, 0xF3, 0x6E, 0xC9),
  CRYPTO_MPI_LIMB_DATA4(0x24, 0x74, 0x4A, 0x93),
  CRYPTO_MPI_LIMB_DATA4(0xE0, 0xB3, 0x92, 0x83),
  CRYPTO_MPI_LIMB_DATA4(0x5B, 0xE5, 0x20, 0x01),
  CRYPTO_MPI_LIMB_DATA4(0x09, 0xD3, 0x08, 0x0E),
  CRYPTO_MPI_LIMB_DATA4(0x55, 0x37, 0x7F, 0x8B),
  CRYPTO_MPI_LIMB_DATA4(0x4A, 0x47, 0x19, 0x1C),
  CRYPTO_MPI_LIMB_DATA4(0xAC, 0xD8, 0x2A, 0xF6),
  CRYPTO_MPI_LIMB_DATA4(0x33, 0xE1, 0xDC, 0xAD),
  CRYPTO_MPI_LIMB_DATA4(0xFE, 0x1A, 0x23, 0x49),
  CRYPTO_MPI_LIMB_DATA4(0x7B, 0x0D, 0x86, 0x84),
  CRYPTO_MPI_LIMB_DATA4(0x77, 0x04, 0xE0, 0xD4)
};

const CRYPTO_EdDSA_PRIVATE_KEY SSH_ServerKeys_Ed448_PrivateKey = {
  { CRYPTO_MPI_INIT_RO(SSH_ServerKeys_Ed448_SK_aLimbs) },
  { CRYPTO_MPI_INIT_RO(SSH_ServerKeys_Ed448_PK_aLimbs) },
};
```

Chapter 5

Core API reference

This chapter explains the API functions of emSSH which are needed for secure communication. The emSSH API is kept as simple as possible to provide a straightforward way to integrate emSSH into a product.

5.1   Preprocessor symbols

5.1.1   Version number

Description

Symbol expands to a number that identifies the specific emSSH release.

Definition

```
#define SSH_VERSION    25400
```

Symbols

Definition	Description
SSH_VERSION	Format is "Mmmrr" so, for example, 24401 corresponds to version 2.44a.

5.1.2 Logging flags

Description

Flags that control log output.

Definition

```
#define SSH_LOG_ERROR        (1uL     )
#define SSH_LOG_TRANSPORT    (1uL << 1)
#define SSH_LOG_SUITE        (1uL << 2)
#define SSH_LOG_KEYS         (1uL << 3)
#define SSH_LOG_MESSAGE      (1uL << 4)
#define SSH_LOG_CONFIG       (1uL << 5)
#define SSH_LOG_SCP          (1uL << 6)
#define SSH_LOG_APP          (1uL << 31)
```

Symbols

Definition	Description
SSH_LOG_ERROR	Log all error status returns generated by emSSH.
SSH_LOG_TRANSPORT	Log incoming and outgoing SSH packets.
SSH_LOG_SUITE	Log agreed connection settings.
SSH_LOG_KEYS	Log keys and operations related to keys.
SSH_LOG_MESSAGE	Log decoded SSH messages.
SSH_LOG_CONFIG	Log emSSH configuration on startup.
SSH_LOG_SCP	Log secure copy (SCP) operations.
SSH_LOG_APP	Log application messages.

Additional information

Flags are added using `SSH_AddLogFilter()` and removed using `SSH_RemoveLogFilter()`.

5.1.3   Warning flags

Description

Flags that control warning output.

Definition

```
#define SSH_WARN_CONFIG     (1uL     )
#define SSH_WARN_PROTOCOL   (1uL << 1)
#define SSH_WARN_SCP        (1uL << 2)
```

Symbols

Definition	Description
SSH_WARN_CONFIG	Warn on configuration issues on startup.
SSH_WARN_PROTOCOL	Warn on protocol-related issues.
SSH_WARN_SCP	Warn on secure copy (SCP) issues.

Additional information

Flags   are   added   using   `SSH_AddWarnFilter()`   and   removed   using
`SSH_RemoveWarnFilter()`.

5.2   Data types

5.2.1   SSH_CHANNEL_REQUEST_PARAS

Description

Broken-down parameters for channel request.

Type definition

```
typedef struct {
  SSH_CHANNEL_REQUEST * pRequest;
  int                   WantReply;
  U8                  * pData;
  unsigned              DataLen;
} SSH_CHANNEL_REQUEST_PARAS;
```

Structure members

Member	Description
pRequest	Pointer to channel request
WantReply	Boolean, whether the request requires a channel status reply
pData	Pointer to channel request parameter string
DataLen	Octet length of the channel request parameter string

5.2.2   SSH_PTYREQ_PARAS

Description

Broken-down parameters from pseudo-terminal request.

Type definition

```
typedef struct {
  U32       TerminalWidth;
  U32       TerminalHeight;
  U32       TerminalPixelWidth;
  U32       TerminalPixelHeight;
  void    * pTerminalName;
  unsigned  TerminalNameLen;
  void    * pTerminalModes;
  unsigned  TerminalModesLen;
} SSH_PTYREQ_PARAS;
```

Structure members

Member	Description
TerminalWidth	Terminal width in characters.
TerminalHeight	Terminal height in rows.
TerminalPixelWidth	Terminal width in pixels.
TerminalPixelHeight	Terminal height in pixels.
pTerminalName	Pointer to terminal name string.
TerminalNameLen	Octet length of terminal name string.
pTerminalModes	Terminal modes string.
TerminalModesLen	Octet length of terminal modes string.

See also

```
SSH_PTYREQ_ParseParas()
```

5.2.3   SSH_WINDOW_CHANGE_PARAS

Description

Broken-down parameters from window change request.

Type definition

```
typedef struct {
  U32  TerminalWidth;
  U32  TerminalHeight;
  U32  TerminalPixelWidth;
  U32  TerminalPixelHeight;
} SSH_WINDOW_CHANGE_PARAS;
```

Structure members

Member	Description
TerminalWidth	Terminal width in characters.
TerminalHeight	Terminal height in rows.
TerminalPixelWidth	Terminal width in pixels.
TerminalPixelHeight	Terminal height in pixels.

See also

SSH_WINDOW_CHANGE_ParseParas()

5.2.4   SSH_USERAUTH_REQUEST_PARAS

Description

Broken-down parameters from user authentication request.

Type definition

```
typedef struct {
  void       * pUserName;
  unsigned     UserNameLen;
  void       * pServiceName;
  unsigned     ServiceNameLen;
  void       * pMethodName;
  unsigned     MethodNameLen;
  CRYPTO_TLV * pTLV;
} SSH_USERAUTH_REQUEST_PARAS;
```

Structure members

Member	Description
pUserName	Pointer to user name string.
UserNameLen	Octet length of the user name string.
pServiceName	Pointer to service name string.
ServiceNameLen	Octet length of the service name string.
pMethodName	Pointer to the method name string.
MethodNameLen	Octet length of the method name string.
pTLV	Internal use.

See also

SSH_USERAUTH_NONE_ParseParas() and SSH_USERAUTH_PASSWORD_ParseParas()

5.2.5   SSH_USERAUTH_PASSWORD_PARAS

Description

Broken-down parameters for password user-authentication request.

Type definition

```
typedef struct {
  void    * pPassword;
  unsigned  PasswordLen;
} SSH_USERAUTH_PASSWORD_PARAS;
```

Structure members

Member	Description
pPassword	Pointer to password string.
PasswordLen	Octet length of the password string.

See also

SSH_USERAUTH_PASSWORD_ParseParas()

5.3   Control functions

The table below lists the functions provided by the emSSH API. Detailed description of each function is found in the sections that follow.

Function	Description
SSH_Init()	Initialize the SSH module.
SSH_Exit()	Finalize the SSH module.
SSH_SetHostKeyAPI()	Configure the server identity API.

5.3.1   SSH_Init()

Description

Initialize the SSH module.

Prototype

```
void SSH_Init(void);
```

Additional information

Before using an SSH service, you must call `SSH_Init()`.

5.3.2   SSH_Exit()

Description

Finalize the SSH module.

Prototype

```
void SSH_Exit(void);
```

Additional information

This function deinitializes the SSH module. Once finalized, no further calls must be made to the emSSH API.

5.3.3   SSH_SetHostKeyAPI()

Description

Configure the server identity API.

Prototype

```
void SSH_SetHostKeyAPI(const SSH_HOSTKEY_API * pAPI);
```

Parameters

Parameter	Description
pAPI	Pointer to server identity API.

5.4   Configuration functions

The table below lists the functions that configure emSSH for operation.

Function	Description
SSH_KEY_EXCHANGE_ALGORITHM_Add()	Install a key exchange algorithm.
SSH_PUBLIC_KEY_ALGORITHM_Add()	Install a public key algorithm.
SSH_ENCRYPTION_ALGORITHM_Add()	Install a bulk encryption algorithm.
SSH_MAC_ALGORITHM_Add()	Install a MAC algorithm.
SSH_COMPRESSION_ALGORITHM_Add()	Install a compression algorithm.
SSH_USERAUTH_METHOD_Add()	Install a user authentication method.
SSH_CHANNEL_REQUEST_Add()	Install a channel request callback.
SSH_SERVICE_Add()	Install a service.
SSH_MEM_Add()	Add memory to emSSH.

5.4.1   SSH_KEY_EXCHANGE_ALGORITHM_Add()

Description

Install a key exchange algorithm.

Prototype

```
void SSH_KEY_EXCHANGE_ALGORITHM_Add(SSH_KEY_EXCHANGE_ALGORITHM * pAPI);
```

Parameters

Parameter	Description
pAPI	Pointer to API to install.

Additional information

Only installed key exchange algorithms will be advertised to clients by the SSH server.

Implemented key exchage algorithms

The following key exchange algorithms are supported by emSSH:

```
extern SSH_KEY_EXCHANGE_ALGORITHM SSH_KEY_EXCHANGE_RSA1024_SHA1;
extern SSH_KEY_EXCHANGE_ALGORITHM SSH_KEY_EXCHANGE_RSA2048_SHA256;
extern SSH_KEY_EXCHANGE_ALGORITHM SSH_KEY_EXCHANGE_DH_GROUP1_SHA1;
extern SSH_KEY_EXCHANGE_ALGORITHM SSH_KEY_EXCHANGE_DH_GROUP14_SHA1;
extern SSH_KEY_EXCHANGE_ALGORITHM SSH_KEY_EXCHANGE_DH_GROUP14_SHA256;
extern SSH_KEY_EXCHANGE_ALGORITHM SSH_KEY_EXCHANGE_DH_GROUP15_SHA512;
extern SSH_KEY_EXCHANGE_ALGORITHM SSH_KEY_EXCHANGE_DH_GROUP16_SHA512;
extern SSH_KEY_EXCHANGE_ALGORITHM SSH_KEY_EXCHANGE_DH_GROUP17_SHA512;
extern SSH_KEY_EXCHANGE_ALGORITHM SSH_KEY_EXCHANGE_DH_GROUP18_SHA512;
extern SSH_KEY_EXCHANGE_ALGORITHM SSH_KEY_EXCHANGE_DH_GROUP_EXCHANGE_SHA1;
extern SSH_KEY_EXCHANGE_ALGORITHM SSH_KEY_EXCHANGE_DH_GROUP_EXCHANGE_SHA256;
extern SSH_KEY_EXCHANGE_ALGORITHM SSH_KEY_EXCHANGE_ECDH_SHA2_NISTP256;
extern SSH_KEY_EXCHANGE_ALGORITHM SSH_KEY_EXCHANGE_ECDH_SHA2_NISTP384;
extern SSH_KEY_EXCHANGE_ALGORITHM SSH_KEY_EXCHANGE_ECDH_SHA2_NISTP521;
extern SSH_KEY_EXCHANGE_ALGORITHM SSH_KEY_EXCHANGE_CURVE25519_SHA256;
extern SSH_KEY_EXCHANGE_ALGORITHM SSH_KEY_EXCHANGE_CURVE448_SHA256;
extern SSH_KEY_EXCHANGE_ALGORITHM SSH_KEY_EXCHANGE_VENDOR_SCS_DH_GROUP14_SHA224;
extern SSH_KEY_EXCHANGE_ALGORITHM SSH_KEY_EXCHANGE_VENDOR_SCS_DH_GROUP14_SHA256;
extern SSH_KEY_EXCHANGE_ALGORITHM SSH_KEY_EXCHANGE_VENDOR_SCS_DH_GROUP15_SHA256;
extern SSH_KEY_EXCHANGE_ALGORITHM SSH_KEY_EXCHANGE_VENDOR_SCS_DH_GROUP15_SHA384;
extern SSH_KEY_EXCHANGE_ALGORITHM SSH_KEY_EXCHANGE_VENDOR_SCS_DH_GROUP16_SHA384;
extern SSH_KEY_EXCHANGE_ALGORITHM SSH_KEY_EXCHANGE_VENDOR_SCS_DH_GROUP16_SHA512;
extern SSH_KEY_EXCHANGE_ALGORITHM SSH_KEY_EXCHANGE_VENDOR_SCS_DH_GROUP18_SHA512;
extern SSH_KEY_EXCHANGE_ALGORITHM SSH_KEY_EXCHANGE_VENDOR_SCS_DH_GROUP_EXCHANGE_SHA224;
extern SSH_KEY_EXCHANGE_ALGORITHM SSH_KEY_EXCHANGE_VENDOR_SCS_DH_GROUP_EXCHANGE_SHA384;
extern SSH_KEY_EXCHANGE_ALGORITHM SSH_KEY_EXCHANGE_VENDOR_SCS_DH_GROUP_EXCHANGE_SHA512;
extern SSH_KEY_EXCHANGE_ALGORITHM SSH_KEY_EXCHANGE_VENDOR_LIBSSH_CURVE25519_SHA256;
```

5.4.2   SSH_PUBLIC_KEY_ALGORITHM_Add()

Description

Install a public key algorithm.

Prototype

```
void SSH_PUBLIC_KEY_ALGORITHM_Add(SSH_PUBLIC_KEY_ALGORITHM * pAPI);
```

Parameters

Parameter	Description
pAPI	Pointer to API to install.

Additional information

Only installed public key algorithms will be advertised to clients by the SSH server.

Implemented public key algorithms

The following public key algorithms are supported by emSSH:

```
extern SSH_PUBLIC_KEY_ALGORITHM SSH_PK_ALGORITHM_SSH_DSS;
extern SSH_PUBLIC_KEY_ALGORITHM SSH_PK_ALGORITHM_SSH_RSA;
extern SSH_PUBLIC_KEY_ALGORITHM SSH_PK_ALGORITHM_ECDSA_SHA2_NISTP256;
extern SSH_PUBLIC_KEY_ALGORITHM SSH_PK_ALGORITHM_ECDSA_SHA2_NISTP384;
extern SSH_PUBLIC_KEY_ALGORITHM SSH_PK_ALGORITHM_ECDSA_SHA2_NISTP521;
extern SSH_PUBLIC_KEY_ALGORITHM SSH_PK_ALGORITHM_SSH_ED25519;
extern SSH_PUBLIC_KEY_ALGORITHM SSH_PK_ALGORITHM_RSA_SHA2_256;
extern SSH_PUBLIC_KEY_ALGORITHM SSH_PK_ALGORITHM_RSA_SHA2_512;
extern SSH_PUBLIC_KEY_ALGORITHM SSH_PK_ALGORITHM_VENDOR_SCS_SSH_DSS_SHA256;
extern SSH_PUBLIC_KEY_ALGORITHM SSH_PK_ALGORITHM_VENDOR_SCS_SSH_RSA_SHA224;
extern SSH_PUBLIC_KEY_ALGORITHM SSH_PK_ALGORITHM_VENDOR_SCS_SSH_RSA_SHA256;
extern SSH_PUBLIC_KEY_ALGORITHM SSH_PK_ALGORITHM_VENDOR_SCS_SSH_RSA_SHA384;
extern SSH_PUBLIC_KEY_ALGORITHM SSH_PK_ALGORITHM_VENDOR_SCS_SSH_RSA_SHA512;
```

5.4.3   SSH_ENCRYPTION_ALGORITHM_Add()

Description

Install a bulk encryption algorithm.

Prototype

```
void SSH_ENCRYPTION_ALGORITHM_Add(SSH_ENCRYPTION_ALGORITHM * pAPI);
```

Parameters

Parameter	Description
pAPI	Pointer to API to install.

Additional information

Only installed encryption algorithms will be advertised to clients by the SSH server.

Implemented encryption algorithms

The following encryptions algorithms are supported by emSSH:

```
extern SSH_ENCRYPTION_ALGORITHM SSH_ENCRYPTION_ALGORITHM_RC4;
extern SSH_ENCRYPTION_ALGORITHM SSH_ENCRYPTION_ALGORITHM_RC4_128;
extern SSH_ENCRYPTION_ALGORITHM SSH_ENCRYPTION_ALGORITHM_RC4_256;
extern SSH_ENCRYPTION_ALGORITHM SSH_ENCRYPTION_ALGORITHM_3DES_CBC;
extern SSH_ENCRYPTION_ALGORITHM SSH_ENCRYPTION_ALGORITHM_3DES_CTR;
extern SSH_ENCRYPTION_ALGORITHM SSH_ENCRYPTION_ALGORITHM_AES128_CBC;
extern SSH_ENCRYPTION_ALGORITHM SSH_ENCRYPTION_ALGORITHM_AES192_CBC;
extern SSH_ENCRYPTION_ALGORITHM SSH_ENCRYPTION_ALGORITHM_AES256_CBC;
extern SSH_ENCRYPTION_ALGORITHM SSH_ENCRYPTION_ALGORITHM_AES128_CTR;
extern SSH_ENCRYPTION_ALGORITHM SSH_ENCRYPTION_ALGORITHM_AES192_CTR;
extern SSH_ENCRYPTION_ALGORITHM SSH_ENCRYPTION_ALGORITHM_AES256_CTR;
extern SSH_ENCRYPTION_ALGORITHM SSH_ENCRYPTION_ALGORITHM_CAMELLIA128_CBC;
extern SSH_ENCRYPTION_ALGORITHM SSH_ENCRYPTION_ALGORITHM_CAMELLIA192_CBC;
extern SSH_ENCRYPTION_ALGORITHM SSH_ENCRYPTION_ALGORITHM_CAMELLIA256_CBC;
extern SSH_ENCRYPTION_ALGORITHM SSH_ENCRYPTION_ALGORITHM_CAMELLIA128_CTR;
extern SSH_ENCRYPTION_ALGORITHM SSH_ENCRYPTION_ALGORITHM_CAMELLIA192_CTR;
extern SSH_ENCRYPTION_ALGORITHM SSH_ENCRYPTION_ALGORITHM_CAMELLIA256_CTR;
extern SSH_ENCRYPTION_ALGORITHM SSH_ENCRYPTION_ALGORITHM_BLOWFISH_CBC;
extern SSH_ENCRYPTION_ALGORITHM SSH_ENCRYPTION_ALGORITHM_BLOWFISH_CTR;
extern SSH_ENCRYPTION_ALGORITHM SSH_ENCRYPTION_ALGORITHM_TWOFISH128_CBC;
extern SSH_ENCRYPTION_ALGORITHM SSH_ENCRYPTION_ALGORITHM_TWOFISH192_CBC;
extern SSH_ENCRYPTION_ALGORITHM SSH_ENCRYPTION_ALGORITHM_TWOFISH256_CBC;
extern SSH_ENCRYPTION_ALGORITHM SSH_ENCRYPTION_ALGORITHM_TWOFISH128_CTR;
extern SSH_ENCRYPTION_ALGORITHM SSH_ENCRYPTION_ALGORITHM_TWOFISH192_CTR;
extern SSH_ENCRYPTION_ALGORITHM SSH_ENCRYPTION_ALGORITHM_TWOFISH256_CTR;
extern SSH_ENCRYPTION_ALGORITHM SSH_ENCRYPTION_ALGORITHM_TWOFISH_CBC;
extern SSH_ENCRYPTION_ALGORITHM SSH_ENCRYPTION_ALGORITHM_CAST128_CBC;
extern SSH_ENCRYPTION_ALGORITHM SSH_ENCRYPTION_ALGORITHM_CAST128_CTR;
extern SSH_ENCRYPTION_ALGORITHM SSH_ENCRYPTION_ALGORITHM_VENDOR_OPENSSH_AES128_GCM;
extern SSH_ENCRYPTION_ALGORITHM SSH_ENCRYPTION_ALGORITHM_VENDOR_OPENSSH_AES256_GCM;
extern SSH_ENCRYPTION_ALGORITHM SSH_ENCRYPTION_ALGORITHM_VENDOR_SCS_SEED_CBC;
extern SSH_ENCRYPTION_ALGORITHM SSH_ENCRYPTION_ALGORITHM_VENDOR_LIU_RIJNDAEL_CBC;
```

5.4.4   SSH_MAC_ALGORITHM_Add()

Description

Install a MAC algorithm.

Prototype

```
void SSH_MAC_ALGORITHM_Add(SSH_MAC_ALGORITHM * pAPI);
```

Parameters

Parameter	Description
pAPI	Pointer to API to install.

Additional information

Only installed MAC algorithms will be advertised to clients by the SSH server.

Implemented MAC algorithms

The following MAC algorithms are supported by emSSH:

```
extern SSH_MAC_ALGORITHM SSH_MAC_ALGORITHM_HMAC_MD5;
extern SSH_MAC_ALGORITHM SSH_MAC_ALGORITHM_HMAC_MD5_96;
extern SSH_MAC_ALGORITHM SSH_MAC_ALGORITHM_HMAC_SHA1;
extern SSH_MAC_ALGORITHM SSH_MAC_ALGORITHM_HMAC_SHA1_96;
extern SSH_MAC_ALGORITHM SSH_MAC_ALGORITHM_HMAC_SHA2_256;
extern SSH_MAC_ALGORITHM SSH_MAC_ALGORITHM_HMAC_SHA2_512;
extern SSH_MAC_ALGORITHM SSH_MAC_ALGORITHM_VENDOR_OPENSSH_HMAC_RIPEMD160;
extern SSH_MAC_ALGORITHM SSH_MAC_ALGORITHM_VENDOR_OPENSSH_HMAC_MD5_ETM;
extern SSH_MAC_ALGORITHM SSH_MAC_ALGORITHM_VENDOR_OPENSSH_HMAC_MD5_96_ETM;
extern SSH_MAC_ALGORITHM SSH_MAC_ALGORITHM_VENDOR_OPENSSH_HMAC_SHA1_ETM;
extern SSH_MAC_ALGORITHM SSH_MAC_ALGORITHM_VENDOR_OPENSSH_HMAC_SHA1_96_ETM;
extern SSH_MAC_ALGORITHM SSH_MAC_ALGORITHM_VENDOR_OPENSSH_HMAC_SHA2_256_ETM;
extern SSH_MAC_ALGORITHM SSH_MAC_ALGORITHM_VENDOR_OPENSSH_HMAC_SHA2_512_ETM;
extern SSH_MAC_ALGORITHM SSH_MAC_ALGORITHM_VENDOR_OPENSSH_HMAC_RIPEMD160_ETM;
extern SSH_MAC_ALGORITHM SSH_MAC_ALGORITHM_VENDOR_SCS_HMAC_SHA224;
extern SSH_MAC_ALGORITHM SSH_MAC_ALGORITHM_VENDOR_SCS_HMAC_SHA256;
extern SSH_MAC_ALGORITHM SSH_MAC_ALGORITHM_VENDOR_SCS_HMAC_SHA384;
extern SSH_MAC_ALGORITHM SSH_MAC_ALGORITHM_VENDOR_SCS_HMAC_SHA512;
```

5.4.5   SSH_COMPRESSION_ALGORITHM_Add()

Description

Install a compression algorithm.

Prototype

```
void SSH_COMPRESSION_ALGORITHM_Add(SSH_COMPRESSION_ALGORITHM * pAPI);
```

Parameters

Parameter	Description
pAPI	Pointer to API to install.

Additional information

Only installed compression algorithms will be advertised to clients by the SSH server.

Implemented compression algorithms

The following compression algorithms are supported by emSSH:

```
extern SSH_COMPRESSION_ALGORITHM SSH_COMPRESSION_ALGORITHM_NONE;
```

5.4.6   SSH_USERAUTH_METHOD_Add()

Description

Install a user authentication method.

Prototype

```
void SSH_USERAUTH_METHOD_Add(SSH_USERAUTH_METHOD       * pAPI,
                             SSH_USERAUTH_REQUEST_FUNC * pfCallback);
```

Parameters

Parameter	Description
pAPI	Pointer to API to install.
pfCallback	Callback to activate when user autentication is requested.

Additional information

Only installed user authentication methods will be advertised to clients by the SSH server.

Implemented user authentication methods

The following user authentication methods are supported by emSSH:

```
extern SSH_USERAUTH_METHOD SSH_USERAUTH_METHOD_None;
extern SSH_USERAUTH_METHOD SSH_USERAUTH_METHOD_Password;
```

5.4.7   SSH_CHANNEL_REQUEST_Add()

Description

Install a channel request callback.

Prototype

```
void SSH_CHANNEL_REQUEST_Add(SSH_CHANNEL_REQUEST      * pAPI,
                             SSH_CHANNEL_REQUEST_FUNC * pfCallback);
```

Parameters

Parameter	Description
pAPI	Pointer to API to install.
pfCallback	Callback to activate when the request is received.

Implemented channel requests

The following channel requests are supported by emSSH:

```
extern SSH_CHANNEL_REQUEST SSH_CHANNEL_REQUEST_PTY_REQ;
extern SSH_CHANNEL_REQUEST SSH_CHANNEL_REQUEST_ENV;
extern SSH_CHANNEL_REQUEST SSH_CHANNEL_REQUEST_SHELL;
extern SSH_CHANNEL_REQUEST SSH_CHANNEL_REQUEST_EXEC;
extern SSH_CHANNEL_REQUEST SSH_CHANNEL_REQUEST_BREAK;
```

5.4.8   SSH_SERVICE_Add()

Description

Install a service.

Prototype

```
void SSH_SERVICE_Add(SSH_SERVICE              * pAPI,
                     SSH_SERVICE_REQUEST_FUNC * pfCallback);
```

Parameters

Parameter	Description
pAPI	Pointer to API to install.
pfCallback	Callback to activate when the request is received.

Implemented services

The following services are supported by emSSH:

```
extern SSH_SERVICE SSH_SERVICE_USERAUTH;
extern SSH_SERVICE SSH_SERVICE_CONNECTION;
```

5.4.9   SSH_MEM_Add()

Description

Add memory to emSSH.

Prototype

```
void SSH_MEM_Add(void     * pStore,
                 unsigned   NumBytesStore);
```

Parameters

Parameter	Description
pStore	Pointer to the first byte of memory to be added. This must be correctly aligned for the processor and compiler combination.
NumBytesStore	Number of bytes in memory block.

Additional information

This function must be called a maximum of one time to add memory to emSSH. Once the memory is added, the heap implementation that manages it is selected. If emSSH is to use the C system heap, this function should not be called.

5.5   Information functions

The table below lists the functions that return emSSH information.

Function	Description
SSH_GetVersionText()	Get emSSH version as printable string.
SSH_GetCopyrightText()	Get emSSH copyright as printable string.

5.5.1 SSH_GetVersionText()

Description

Get emSSH version as printable string.

Prototype

```
char *SSH_GetVersionText(void);
```

Return value

Zero-terminated version string.

5.5.2   SSH_GetCopyrightText()

Description

Get emSSH copyright as printable string.

Prototype

```
char *SSH_GetCopyrightText(void);
```

Return value

Zero-terminated copyright string.

5.6   Session control functions

The table below lists the functions that are used for SSH sessions.

Function	Description
SSH_SESSION_Alloc()	Allocate a session.
SSH_SESSION_Init()	Initialize an SSH session.
SSH_SESSION_ConfBuffers()	Configure session buffers.
SSH_SESSION_Disconnect()	Disconnect session.
SSH_SESSION_DisconnectEx()	Disconnect session.
SSH_SESSION_SendUserauthBanner()	Send a USERAUTH_BANNER message.
SSH_SESSION_SendServiceAccept()	Send the SSH Service Accept message.
SSH_SESSION_Process()	Run SSH state machine.
SSH_SESSION_IterateChannels()	Iterate over session channels.
SSH_SESSION_QueryIndex()	Query session index.
SSH_SESSION_QuerySocket()	Query session socket.

5.6.1   SSH_SESSION_Alloc()

Description

Allocate a session.

Prototype

```
void SSH_SESSION_Alloc(SSH_SESSION ** ppSession);
```

Parameters

Parameter	Description
ppSession	Pointer that receives a pointer to the allocated session. If no session is free, this receives the null pointer.

5.6.2   SSH_SESSION_Init()

Description

Initialize an SSH session.

Prototype

```
void SSH_SESSION_Init(      SSH_SESSION       * pSelf,
                            int                 Socket,
                      const SSH_TRANSPORT_API * pTransportAPI);
```

Parameters

Parameter	Description
pSelf	Pointer to SSH session.
Socket	Socket reference that will carry the data.
pTransportAPI	Pointer to API that carries the data.

5.6.3   SSH_SESSION_ConfBuffers()

Description

Configure session buffers.

Prototype

```
void SSH_SESSION_ConfBuffers(SSH_SESSION * pSelf,
                             void        * pRxBuffer,
                             unsigned      RxBufferLen,
                             void        * pTxBuffer,
                             unsigned      TxBufferLen);
```

Parameters

Parameter	Description
pSelf	Pointer to SSH session.
pRxBuffer	Pointer to object to use as the receive buffer.
RxBufferLen	Octet length of the receive buffer.
pTxBuffer	Pointer to object to use as the transmit buffer.
TxBufferLen	Octet length of the transmit buffer.

Additional information

The receive buffer and the transmit buffer can be shared. However, if you provide the same buffer for both receive and transmit, you must be aware that any data provided to you in a callback is volatile and held in the receive buffer. If your callback function initiates a transmit, all data in the receive buffer becomes invalid.

5.6.4   SSH_SESSION_Disconnect()

Description

Disconnect session.

Prototype

```
void SSH_SESSION_Disconnect(SSH_SESSION * pSelf,
                            unsigned      ReasonCode);
```

Parameters

Parameter	Description
pSelf	Pointer to SSH session.
ReasonCode	Disconnection reason provided to peer.

Additional information

This function closes all channels associated with an SSH session, disconnects the session, and returns all resources associated with the session for reuse.

5.6.5   SSH_SESSION_DisconnectEx()

Description

Disconnect session.

Prototype

```
void SSH_SESSION_DisconnectEx(       SSH_SESSION * pSelf,
                                     unsigned      ReasonCode,
                               const char        * sDescription,
                               const char        * sLanguageTag);
```

Parameters

Parameter	Description
pSelf	Pointer to SSH session.
ReasonCode	Disconnection reason provided to peer.
sDescription	Description optionally presented to peer.
sLanguageTag	Language for peer message.

Additional information

This function closes all channels associated with an SSH session, disconnects the session, and returns all resources associated with the session for reuse.

5.6.6   SSH_SESSION_SendUserauthBanner()

Description

Send a USERAUTH_BANNER message.

Prototype

```
int SSH_SESSION_SendUserauthBanner(      SSH_SESSION * pSelf,
                                   const char        * sMessage,
                                   const char        * sLanguageTag);
```

Parameters

Parameter	Description
pSelf	Pointer to SSH session.
sMessage	Pointer to zero-terminated banner string.
sLanguageTag	Pointer to zero-terminated language tag.

Return value

≥ 0   Success.
< 0   Error.

5.6.7   SSH_SESSION_SendServiceAccept()

Description

Send the SSH Service Accept message.

Prototype

```
int SSH_SESSION_SendServiceAccept(      SSH_SESSION * pSelf,
                                  const char        * sServiceName);
```

Parameters

Parameter	Description
pSelf	Pointer to SSH session.
sServiceName	Service name.

Return value

≥ 0   Success.
< 0   Error.

5.6.8   SSH_SESSION_Process()

Description

Run SSH state machine.

Prototype

```
int SSH_SESSION_Process(SSH_SESSION * pSelf);
```

Parameters

Parameter	Description
pSelf	Pointer to SSH session.

Return value

≥ 0   Success.
< 0   Error, session disconnected.

Additional information

This function processes incoming SSH messages and manages the SSH connection. If any error is detected during processing, the connection is disconnected.

5.6.9   SSH_SESSION_IterateChannels()

Description

Iterate over session channels.

Prototype

```
int SSH_SESSION_IterateChannels(SSH_SESSION              * pSelf,
                                SSH_CHANNEL_SERVICE_FUNC * pfCallback);
```

Parameters

Parameter	Description
pSelf	Pointer to SSH session.
pfCallback	Callback function to invoke for each channel.

Return value

≥ 0   Success (maximum value returned by any callback).
< 0   Processing halted (final negative value returned by callback)

Additional information

This function iterates over each channel associated with the provided session, and for each channel it invokes the callback. The return value of the guides further processing: nonnegative indicates that iteration continues, and negative indicates an immediate halt to further iteration.

When the callback delivers a negative result, that result is passed through to the caller of this function.

When the callback delivers a nonnegative result, that result is compared to a local maximum value maintained over all channels, and if greater, the result replaces the local maximum value. When all channels are successfully iterated, that local maximum is delivered as the function result.

This particular behavior is useful to indicate whether any processing took place in a callback, for instance. The callback can deliver a zero result to indicate no processing, and a positive value to indicate that processing did take place: the caller of this function can use the result as an indication that some channel processed data, and this is better style than using a global variable.

5.6.10   SSH_SESSION_QueryIndex()

Description

Query session index.

Prototype

```
unsigned SSH_SESSION_QueryIndex(SSH_SESSION * pSelf);
```

Parameters

Parameter	Description
pSelf	Pointer to SSH session.

Return value

Session index associated with session.

Additional information

The session index will be a small nonnegative integer that can be used to index "parallel data" to the session.

5.6.11   SSH_SESSION_QuerySocket()

Description

Query session socket.

Prototype

```
int SSH_SESSION_QuerySocket(SSH_SESSION * pSelf);
```

Parameters

Parameter	Description
pSelf	Pointer to SSH session.

Return value

Socket ID associated with session.

Additional information

The socket association is made when the session is initialized by `SSH_SESSION_Init`.

5.7   Channel functions

Function	Description
SSH_CHANNEL_Config()	Configure channel.
SSH_CHANNEL_Close()	Close a channel.
SSH_CHANNEL_SendData()	Send data to peer.
SSH_CHANNEL_SendEOF()	Send EOF message to peer.
SSH_CHANNEL_SendSuccess()	Send a channel success message.
SSH_CHANNEL_SendFailure()	Send a channel failure message.
SSH_CHANNEL_SendCompletion()	Send a channel success or failure message.
SSH_CHANNEL_SendRequestExitStatus()	Send a channel "exit-status" message.
SSH_CHANNEL_QueryUserContext()	Retrieve user context.

5.7.1   SSH_CHANNEL_Config()

Description

Configure channel.

Prototype

```
int SSH_CHANNEL_Config(       SSH_SESSION     * pSelf,
                              unsigned          Channel,
                              unsigned          BufferSize,
                        const SSH_CHANNEL_API * pAPI,
                              void            * pUserContext);
```

Parameters

Parameter	Description
pSelf	Pointer to SSH session.
Channel	The local channel ID to configure.
BufferSize	Window size advertised to the peer.
pAPI	Pointer to channel API to use for this channel.
pUserContext	Pointer to optional user context to associate with this channel.

Return value

≥ 0   Success.
< 0   Error.

5.7.2   SSH_CHANNEL_Close()

Description

Close a channel.

Prototype

```
void SSH_CHANNEL_Close(SSH_SESSION * pSelf,
                       unsigned      Channel);
```

Parameters

Parameter	Description
pSelf	Pointer to SSH session.
Channel	Local channel to close.

5.7.3   SSH_CHANNEL_SendData()

Description

Send data to peer.

Prototype

```
int SSH_CHANNEL_SendData(      SSH_SESSION * pSelf,
                               unsigned      Channel,
                         const void        * pData,
                               unsigned      DataLen);
```

Parameters

Parameter	Description
pSelf	Pointer to SSH session.
Channel	Local channel ID.
pData	Pointer to object to send to peer.
DataLen	Octet length of the object to send to peer.

Return value

≥ 0   Success.
< 0   Error.

5.7.4   SSH_CHANNEL_SendEOF()

Description

Send EOF message to peer.

Prototype

```
int SSH_CHANNEL_SendEOF(SSH_SESSION * pSelf,
                        unsigned       Channel);
```

Parameters

Parameter	Description
pSelf	Pointer to SSH session.
Channel	Local channel ID.

Return value

≥ 0   Success.
< 0   Error.

5.7.5   SSH_CHANNEL_SendSuccess()

Description

Send a channel success message.

Prototype

```
int SSH_CHANNEL_SendSuccess(SSH_SESSION * pSelf,
                            unsigned      Channel);
```

Parameters

Parameter	Description
pSelf	Pointer to SSH session.
Channel	The local channel ID.

Return value

≥ 0   Success.
< 0   Error.

Additional information

The local channel ID is translated into the remote channel ID when sending this message.

5.7.6   SSH_CHANNEL_SendFailure()

Description

Send a channel failure message.

Prototype

```
int SSH_CHANNEL_SendFailure(SSH_SESSION * pSelf,
                            unsigned      Channel);
```

Parameters

Parameter	Description
pSelf	Pointer to SSH session.
Channel	The local channel ID.

Return value

≥ 0   Success.
< 0   Error.

Additional information

The local channel ID is translated into the remote channel ID when sending this message.

5.7.7   SSH_CHANNEL_SendCompletion()

Description

Send a channel success or failure message.

Prototype

```
int SSH_CHANNEL_SendCompletion(SSH_SESSION * pSelf,
                               unsigned      Channel,
                               int           Status);
```

Parameters

Parameter	Description
pSelf	Pointer to SSH session.
Channel	The local channel ID.
Status	Completion status: < 0 sends a failure response and ≥ 0 send a success response.

Return value

≥ 0   Success.
< 0   Error.

5.7.8   SSH_CHANNEL_SendRequestExitStatus()

Description

Send a channel "exit-status" message.

Prototype

```
int SSH_CHANNEL_SendRequestExitStatus(SSH_SESSION * pSelf,
                                      unsigned      Channel,
                                      U32           ExitStatus);
```

Parameters

Parameter	Description
pSelf	Pointer to SSH session.
Channel	The local channel ID.
ExitStatus	Exit status to deliver.

Return value

≥ 0   Success.
< 0   Error.

5.7.9   SSH_CHANNEL_QueryUserContext()

Description

Retrieve user context.

Prototype

```
void *SSH_CHANNEL_QueryUserContext(SSH_SESSION * pSelf,
                                   unsigned      Channel);
```

Parameters

Parameter	Description
pSelf	Pointer to SSH session.
Channel	Local channel ID.

Return value

The user context associated with the local channel as set by `SSH_CHANNEL_Config`.

5.7.10   SSH_CHANNEL_QueryCanWrite()

Description

Retrieve number of bytes that can be written to the channel.

Prototype

```
unsigned SSH_CHANNEL_QueryCanWrite(SSH_SESSION * pSelf,
                                   unsigned      Channel);
```

Parameters

Parameter	Description
pSelf	Pointer to SSH session.
Channel	Local channel ID.

Return value

The maximum number of bytes that can be sent to the peer at this time which is the outgoing window size.

5.8   Request parsing functions

Function	Description
`SSH_PTYREQ_ParseParas()`	Parse `CHANNEL_REQUEST` parameters for request "pty-req".
`SSH_USERAUTH_NONE_ParseParas()`	Parse `USERAUTH_REQUEST` message parameters for method "none".
`SSH_USERAUTH_PASSWORD_ParseParas()`	Parse `USERAUTH_REQUEST` message parameters for method "password".

5.8.1   SSH_PTYREQ_ParseParas()

Description

Parse `CHANNEL_REQUEST` parameters for request "pty-req".

Prototype

```
int SSH_PTYREQ_ParseParas(SSH_CHANNEL_REQUEST_PARAS * pReqParas,
                          SSH_PTYREQ_PARAS          * pPtyParas);
```

Parameters

Parameter	Description
pReqParas	Pointer to channel request parameters.
pPtyParas	Pointer to object that receives the decoded "pty-req" parameters.

Return value

≥ 0   Success.
< 0   Error.

5.8.2   SSH_WINDOW_CHANGE_ParseParas()

Description

Parse `WIINDOW_CHANGE` parameters for request "window-change".

Prototype

```
int SSH_WINDOW_CHANGE_ParseParas(SSH_CHANNEL_REQUEST_PARAS * pReqParas,
                                 SSH_WINDOW_CHANGE_PARAS   * pWinChangeParas);
```

Parameters

Parameter	Description
pReqParas	Pointer to channel request parameters.
pWinChangeParas	Pointer to object that receives the decoded "window-change" parameters.

Return value

≥ 0   Success.
< 0   Error.

5.8.3   SSH_USERAUTH_NONE_ParseParas()

Description

Parse `USERAUTH_REQUEST` message parameters for method "none".

Prototype

```
int SSH_USERAUTH_NONE_ParseParas(SSH_USERAUTH_REQUEST_PARAS * pReqParas,
                                 SSH_USERAUTH_NONE_PARAS     * pMethodParas);
```

Parameters

Parameter	Description
pReqParas	Pointer to user authentication request parameters.
pMethodParas	Pointer to object that receives the decoded "none" method parameters.

Return value

≥ 0   Success.
< 0   Error.

5.8.4   SSH_USERAUTH_PASSWORD_ParseParas()

Description

Parse `USERAUTH_REQUEST` message parameters for method "password".

Prototype

```
int SSH_USERAUTH_PASSWORD_ParseParas(SSH_USERAUTH_REQUEST_PARAS  * pReqParas,
                                     SSH_USERAUTH_PASSWORD_PARAS * pMethodParas);
```

Parameters

Parameter	Description
pReqParas	Pointer to user authentication request parameters.
pMethodParas	Pointer to object that receives the decoded "password" method parameters.

Return value

≥ 0   Success.
< 0   Error.

5.9   Diagnostic functions

The table below lists the diagnostic functions provided by the emSSH API.

Function	Description
SSH_AddLogFilter()	Add filters to the active log filter.
SSH_AddWarnFilter()	Add filters to the active warning filter.
SSH_RemoveLogFilter()	Remove filters from the active log filter.
SSH_RemoveWarnFilter()	Remove filters from the active warning filter.
SSH_SetLogFilter()	Set active log filter.
SSH_SetWarnFilter()	Set the active warning filter.

5.9.1   SSH_AddLogFilter()

Description

Add filters to the active log filter.

Prototype

```
U32 SSH_AddLogFilter(U32 FilterMask);
```

Parameters

Parameter	Description
FilterMask	Filters to enable.

Return value

The log filter mask before addition.

Additional information

This function adds to the existing log filter mask using a bitwise or of the new filter mask and the given filter mask.

See also

Logging flags on page 152.

5.9.2   SSH_AddWarnFilter()

Description

Add filters to the active warning filter.

Prototype

```
U32 SSH_AddWarnFilter(U32 FilterMask);
```

Parameters

Parameter	Description
FilterMask	Filters to enable.

Return value

The warning filter mask before addition.

Additional information

This function adds to the existing warning filter mask using a bitwise or of the new filter mask and the given filter mask.

See also

Warning flags on page 153.

5.9.3   SSH_RemoveLogFilter()

Description

Remove filters from the active log filter.

Prototype

```
U32 SSH_RemoveLogFilter(U32 FilterMask);
```

Parameters

Parameter	Description
FilterMask	Filters to disable.

Return value

The log filter mask before removal.

Additional information

This function removes the filters specified in `FilterMask` from the existing log filter.

See also

Logging flags on page 152.

5.9.4   SSH_RemoveWarnFilter()

Description

Remove filters from the active warning filter.

Prototype

```
U32 SSH_RemoveWarnFilter(U32 FilterMask);
```

Parameters

Parameter	Description
FilterMask	Filters to disable.

Return value

The warning filter mask before removal.

Additional information

This function removes the filters specified in `FilterMask` from the existing log filter.

See also

Warning flags on page 153.

5.9.5   SSH_SetLogFilter()

Description

Set active log filter.

Prototype

```
U32 SSH_SetLogFilter(U32 FilterMask);
```

Parameters

Parameter	Description
FilterMask	Filters to enable.

Return value

The log filter mask before replacement.

Additional information

This function sets the log filter mask to use, entirely replacing the previously set filter mask. The bits set in the filter mask enable appropriate messages to the log.

See also

Logging flags on page 152.

5.9.6   SSH_SetWarnFilter()

Description

Set the active warning filter.

Prototype

```
U32 SSH_SetWarnFilter(U32 FilterMask);
```

Parameters

Parameter	Description
FilterMask	Filters to enable.

Return value

The warning filter mask before replacement.

Additional information

This function sets the warning filter mask to use, entirely replacing the previously set filter mask. The bits set in the filter mask enable appropriate warnings.

See also

Warning flags on page 153.

5.10   Internal functions, variables and data structures

Internal functions of emSSH are not explained here as they are not required to use emSSH. The application should not rely on any of the internal elements, since they may be subject to change.

> **Note**
>
> Only the documented API functions are guaranteed to remain unchanged and compatible in future versions of emSSH.

Chapter 6

Configuring emSSH

This chapter describes the available compile-time and runtime configuration options.

emSSH's functionality (i.e. its feature set) is completely configurable using runtime calls to select the way that emSSH performs as a client and a server. However, there are choices to be made between different implementations of some computationally intensive algorithms. At the source level you can trade speed of execution for a more compact implementation to reduce code space — this configuration is made using preprocessor symbols defined in a particular manner when compiling emSSH from source code.

6.1   Compile-time definitions

The following definitions can be configured for emSSH.

6.1.1   Maximum number of sessions

Default

```
#define SSH_CONFIG_MAX_SESSIONS    2
```

Override

To define a non-default value, define this symbol in `SSH_Conf.h`.

Description

This defines the maximum number of sessions that can be open, simultaneously, in emSSH.

6.1.2   Maximum number of channels

Default

```
#define SSH_CONFIG_MAX_CHANNELS    SSH_CONFIG_MAX_SESSIONS
```

Override

To define a non-default value, define this symbol in `SSH_Conf.h`.

Description

This defines the maximum number of channels that emSSH will manage. Current implemented protocols use no more than one channel per session, therefore it suffices to default the maximum number of channels to the number of sessions.

6.2   Runtime configuration

Before a secure connection is established, emSSH must be configured with the supporting cryptographic algorithms needed for a secure connection. `SSH_X_Config.c` is configured to match the requirements of most applications and can be taken as an example. You must configure:

- Key exchange algorithms that emSSH will support. See *Adding key exchange algorithms* on page 216.
- Public key algorithms that emSSH will support, such as RSA and DSA. See *Adding public key algorithms* on page 217.
- Bulk encryption ciphers, such as DES and AES. See *Adding ciphers* on page 218.
- Hash and MAC algorithms, such as SHA and MD5. See *Adding MACs* on page 219.
- Compression algorithms to optimize bandwidth. See *Adding compression algorithms* on page 220.

You configure the capabilities of emSSH in the function `SSH_X_Config()` that is called as part of the SSH initialization carried out by `SSH_Init`. `SSH_X_Config()` must be provided in your application as a function with external linkage, and an example is shipped with emSSH.

6.2.1   Adding key exchange algorithms

You   must   add   the   required   key   exchange   algorithms   using
`SSH_KEY_EXCHANGE_ALGORITHM_Add()` when configuring emSSH. The standard key exchange algorithms are:

* `SSH_KEY_EXCHANGE_DH_GROUP1_SHA1`
* `SSH_KEY_EXCHANGE_DH_GROUP14_SHA1`
* `SSH_KEY_EXCHANGE_ECDH_SHA2_NISTP256`
* `SSH_KEY_EXCHANGE_ECDH_SHA2_NISTP384`
* `SSH_KEY_EXCHANGE_ECDH_SHA2_NISTP521`

Only   the   algorithms   `SSH_KEY_EXCHANGE_DH_GROUP1_SHA1`   and
`SSH_KEY_EXCHANGE_DH_GROUP14_SHA1` are mandated by SSH.

Example

```
SSH_KEY_EXCHANGE_ALGORITHM_Add(&SSH_KEY_EXCHANGE_DH_GROUP1_SHA1);
SSH_KEY_EXCHANGE_ALGORITHM_Add(&SSH_KEY_EXCHANGE_DH_GROUP14_SHA1);
```

Additional key exchanges are provided for working with other clients:

* `SSH_KEY_EXCHANGE_RSA1024_SHA1`
* `SSH_KEY_EXCHANGE_RSA2048_SHA256`
* `SSH_KEY_EXCHANGE_DH_GROUP14_SHA256`
* `SSH_KEY_EXCHANGE_DH_GROUP15_SHA512`
* `SSH_KEY_EXCHANGE_DH_GROUP16_SHA512`
* `SSH_KEY_EXCHANGE_DH_GROUP17_SHA512`
* `SSH_KEY_EXCHANGE_DH_GROUP18_SHA512`
* `SSH_KEY_EXCHANGE_DH_GROUP_EXCHANGE_SHA1`
* `SSH_KEY_EXCHANGE_DH_GROUP_EXCHANGE_SHA256`
* `SSH_KEY_EXCHANGE_CURVE25519_SHA256`
* `SSH_KEY_EXCHANGE_VENDOR_SCS_DH_GROUP14_SHA224`
* `SSH_KEY_EXCHANGE_VENDOR_SCS_DH_GROUP14_SHA256`
* `SSH_KEY_EXCHANGE_VENDOR_SCS_DH_GROUP15_SHA256`
* `SSH_KEY_EXCHANGE_VENDOR_SCS_DH_GROUP15_SHA384`
* `SSH_KEY_EXCHANGE_VENDOR_SCS_DH_GROUP16_SHA384`
* `SSH_KEY_EXCHANGE_VENDOR_SCS_DH_GROUP16_SHA512`
* `SSH_KEY_EXCHANGE_VENDOR_SCS_DH_GROUP18_SHA512`
* `SSH_KEY_EXCHANGE_VENDOR_SCS_DH_GROUP_EXCHANGE_SHA224`
* `SSH_KEY_EXCHANGE_VENDOR_SCS_DH_GROUP_EXCHANGE_SHA384`
* `SSH_KEY_EXCHANGE_VENDOR_SCS_DH_GROUP_EXCHANGE_SHA512`
* `SSH_KEY_EXCHANGE_VENDOR_LIBSSH_CURVE25519_SHA256`

6.2.2   Adding public key algorithms

You must add the required key exchange algorithms using `SSH_KEY_EXCHANGE_ALGORITHM_Add()` when configuring emSSH. The standard key exchange algorithms are:

- `SSH_PK_ALGORITHM_SSH_DSS`
- `SSH_PK_ALGORITHM_SSH_RSA`
- `SSH_PK_ALGORITHM_ECDSA_SHA2_NISTP256`
- `SSH_PK_ALGORITHM_ECDSA_SHA2_NISTP384`
- `SSH_PK_ALGORITHM_ECDSA_SHA2_NISTP521`
- `SSH_PK_ALGORITHM_SSH_ED25519`

Only the algorithm `SSH_PK_ALGORITHM_SSH_DSS` is mandated by SSH.

Example

```
SSH_PUBLIC_KEY_ALGORITHM_Add(&SSH_PK_ALGORITHM_SSH_DSS);
```

Additional public key algorithms are provided for working with other clients:

- `SSH_PK_ALGORITHM_RSA_SHA2_256`
- `SSH_PK_ALGORITHM_RSA_SHA2_512`
- `SSH_PK_ALGORITHM_VENDOR_SCS_SSH_DSS_SHA256`
- `SSH_PK_ALGORITHM_VENDOR_SCS_SSH_RSA_SHA224`
- `SSH_PK_ALGORITHM_VENDOR_SCS_SSH_RSA_SHA256`
- `SSH_PK_ALGORITHM_VENDOR_SCS_SSH_RSA_SHA384`
- `SSH_PK_ALGORITHM_VENDOR_SCS_SSH_RSA_SHA512`

6.2.3 Adding ciphers

You must add the required bulk cipher implementation using `SSH_ENCRYPTION_ALGORITHM_Add()` when configuring emSSH. The standard cipher implementations are:

- `SSH_ENCRYPTION_ALGORITHM_RC4`
- `SSH_ENCRYPTION_ALGORITHM_RC4_128`
- `SSH_ENCRYPTION_ALGORITHM_RC4_256`
- `SSH_ENCRYPTION_ALGORITHM_3DES_CBC`
- `SSH_ENCRYPTION_ALGORITHM_3DES_CTR`
- `SSH_ENCRYPTION_ALGORITHM_AES128_CBC`
- `SSH_ENCRYPTION_ALGORITHM_AES192_CBC`
- `SSH_ENCRYPTION_ALGORITHM_AES256_CBC`
- `SSH_ENCRYPTION_ALGORITHM_AES128_CTR`
- `SSH_ENCRYPTION_ALGORITHM_AES192_CTR`
- `SSH_ENCRYPTION_ALGORITHM_AES256_CTR`

Other encryption algorithms are:

- `SSH_ENCRYPTION_ALGORITHM_CAMELLIA128_CBC`
- `SSH_ENCRYPTION_ALGORITHM_CAMELLIA192_CBC`
- `SSH_ENCRYPTION_ALGORITHM_CAMELLIA256_CBC`
- `SSH_ENCRYPTION_ALGORITHM_CAMELLIA128_CTR`
- `SSH_ENCRYPTION_ALGORITHM_CAMELLIA192_CTR`
- `SSH_ENCRYPTION_ALGORITHM_CAMELLIA256_CTR`
- `SSH_ENCRYPTION_ALGORITHM_BLOWFISH_CBC`
- `SSH_ENCRYPTION_ALGORITHM_BLOWFISH_CTR`
- `SSH_ENCRYPTION_ALGORITHM_TWOFISH128_CBC`
- `SSH_ENCRYPTION_ALGORITHM_TWOFISH192_CBC`
- `SSH_ENCRYPTION_ALGORITHM_TWOFISH256_CBC`
- `SSH_ENCRYPTION_ALGORITHM_TWOFISH128_CTR`
- `SSH_ENCRYPTION_ALGORITHM_TWOFISH192_CTR`
- `SSH_ENCRYPTION_ALGORITHM_TWOFISH256_CTR`
- `SSH_ENCRYPTION_ALGORITHM_TWOFISH_CBC`
- `SSH_ENCRYPTION_ALGORITHM_CAST128_CBC`
- `SSH_ENCRYPTION_ALGORITHM_CAST128_CTR`
- `SSH_ENCRYPTION_ALGORITHM_VENDOR_OPENSSH_AES128_GCM`
- `SSH_ENCRYPTION_ALGORITHM_VENDOR_OPENSSH_AES256_GCM`
- `SSH_ENCRYPTION_ALGORITHM_VENDOR_OPENSSH_CHACHA20_POLY1305`
- `SSH_ENCRYPTION_ALGORITHM_VENDOR_LIU_RIJNDAEL_CBC`
- `SSH_ENCRYPTION_ALGORITHM_VENDOR_SCS_SEED_CBC`

Only the algorithm `SSH_ENCRYPTION_ALGORITHM_3DES_CBC` is mandated by SSH.

These encryption algorithms use the underlying emCrypt cipher implementation and will benefit from hardware acceleration if there is an installed hardware accelerator at the emCrypt layer.

Example

```
SSH_ENCRYPTION_ALGORITHM_Add(&SSH_ENCRYPTION_ALGORITHM_3DES_CBC);
```

6.2.4   Adding MACs

You must add required message authentication code implementations using `SSH_MAC_ALGORITHM_Add()`. The MAC implementations are:

* `SSH_MAC_ALGORITHM_HMAC_MD5`
* `SSH_MAC_ALGORITHM_HMAC_MD5_96`
* `SSH_MAC_ALGORITHM_HMAC_SHA1`
* `SSH_MAC_ALGORITHM_HMAC_SHA1_96`
* `SSH_MAC_ALGORITHM_HMAC_SHA2_256`
* `SSH_MAC_ALGORITHM_HMAC_SHA2_512`

MACs defined by OpenSSH are:

* `SSH_MAC_ALGORITHM_VENDOR_OPENSSH_HMAC_RIPEMD160`
* `SSH_MAC_ALGORITHM_VENDOR_OPENSSH_HMAC_MD5_ETM`
* `SSH_MAC_ALGORITHM_VENDOR_OPENSSH_HMAC_MD5_96_ETM`
* `SSH_MAC_ALGORITHM_VENDOR_OPENSSH_HMAC_SHA1_ETM`
* `SSH_MAC_ALGORITHM_VENDOR_OPENSSH_HMAC_SHA1_96_ETM`
* `SSH_MAC_ALGORITHM_VENDOR_OPENSSH_HMAC_SHA2_256_ETM`
* `SSH_MAC_ALGORITHM_VENDOR_OPENSSH_HMAC_SHA2_512_ETM`
* `SSH_MAC_ALGORITHM_VENDOR_OPENSSH_HMAC_RIPEMD160_ETM`

MACs defined by SSH Secure Communications are:

* `SSH_MAC_ALGORITHM_VENDOR_SCS_HMAC_SHA224`
* `SSH_MAC_ALGORITHM_VENDOR_SCS_HMAC_SHA256`
* `SSH_MAC_ALGORITHM_VENDOR_SCS_HMAC_SHA384`
* `SSH_MAC_ALGORITHM_VENDOR_SCS_HMAC_SHA512`

Only the MAC algorithm `SSH_MAC_ALGORITHM_HMAC_SHA1` is mandated by SSH.

Example

```
SSH_MAC_ALGORITHM_Add(&SSH_MAC_ALGORITHM_HMAC_SHA1);
```

6.2.5   Adding compression algorithms

emSSH does not support dynamic data compression and has only a single "none" compression scheme. Even though there is only a single scheme at this time, you must configure emSSH with it using `SSH_COMPRESSION_ALGORITHM_Add()`.

- `SSH_COMPRESSION_ALGORITHM_NONE`

Example

```
SSH_COMPRESSION_ALGORITHM_Add(&SSH_COMPRESSION_ALGORITHM_NONE);
```

6.3   Sample minimal configuration

The following adds only what is absolutely mandated by the SSH specification and constitutes a minimal configuration of emSSH.

```c
void SSH_X_Config(void) {
  //
  // Add memory to be used by emSSH.
  //
  static U32 _aStore[8*1024 / sizeof(U32)];
  SSH_MEM_Add(&_aStore[0], sizeof(_aStore));
  //
  // Add [RFC4253] REQUIRED algorithms.
  //
  SSH_KEY_EXCHANGE_ALGORITHM_Add(&SSH_KEY_EXCHANGE_DH_GROUP1_SHA1);
  SSH_KEY_EXCHANGE_ALGORITHM_Add(&SSH_KEY_EXCHANGE_DH_GROUP14_SHA1);
  SSH_PUBLIC_KEY_ALGORITHM_Add(&SSH_PK_ALGORITHM_SSH_DSS);
  SSH_ENCRYPTION_ALGORITHM_Add(&SSH_ENCRYPTION_ALGORITHM_3DES_CBC);
  SSH_MAC_ALGORITHM_Add(&SSH_MAC_ALGORITHM_HMAC_SHA1);
  SSH_COMPRESSION_ALGORITHM_Add(&SSH_COMPRESSION_ALGORITHM_NONE);
}
```

6.4   Shipped sample configuration

```c
/**********************************************************************
*                    (c) SEGGER Microcontroller GmbH                 *
*                         The Embedded Experts                       *
*                            www.segger.com                          *
**********************************************************************

----------------------- END-OF-HEADER ----------------------------

File      : SSH_X_Config.c
Purpose   : Sample emSSH configuration.

*/

/**********************************************************************
*
*       #include Section
*
**********************************************************************
*/

#include "SSH.h"

/**********************************************************************
*
*       Defines, configurable
*
**********************************************************************
*/

#define ALLOC_SIZE                  (8 * 1024)   // Memory for emSSH to use.

#define INSTALL_RSA_KEY_EXCHANGE          0   // Install RSA key exchange algorithms
#define INSTALL_VENDOR_SCS_EXTENSIONS     0
  // Install SSH Communication Security private algorithms
#define INSTALL_VENDOR_OPENSSH_EXTENSIONS 1   // Install OpenSSH private algorithms
#define INSTALL_VENDOR_LIU_EXTENSIONS     1   // Install LIU private algorithms
#define INSTALL_SLOW_DH_GROUPS            0   // Install slower DH groups 15 through 18

/**********************************************************************
*
*       External const data
*
**********************************************************************
*/

extern const CRYPTO_RSA_PUBLIC_KEY    SSH_ServerKeys_RSA_Host_2048b_PublicKey;
extern const CRYPTO_RSA_PRIVATE_KEY   SSH_ServerKeys_RSA_Host_2048b_PrivateKey;
extern const CRYPTO_RSA_PUBLIC_KEY    SSH_ServerKeys_RSA_Temp_1024b_PublicKey;
extern const CRYPTO_RSA_PRIVATE_KEY   SSH_ServerKeys_RSA_Temp_1024b_PrivateKey;
extern const CRYPTO_RSA_PUBLIC_KEY    SSH_ServerKeys_RSA_Temp_2048b_PublicKey;
extern const CRYPTO_RSA_PRIVATE_KEY   SSH_ServerKeys_RSA_Temp_2048b_PrivateKey;
extern const CRYPTO_DSA_PUBLIC_KEY    SSH_ServerKeys_DSA_1024b_160b_PublicKey;
extern const CRYPTO_DSA_PRIVATE_KEY   SSH_ServerKeys_DSA_1024b_160b_PrivateKey;
extern const CRYPTO_DSA_DOMAIN_PARAMS SSH_ServerKeys_DSA_1024b_160b_DomainParas;
extern const CRYPTO_DSA_PUBLIC_KEY    SSH_ServerKeys_DSA_2048b_160b_PublicKey;
extern const CRYPTO_DSA_PRIVATE_KEY   SSH_ServerKeys_DSA_2048b_160b_PrivateKey;
extern const CRYPTO_DSA_DOMAIN_PARAMS SSH_ServerKeys_DSA_2048b_160b_DomainParas;
extern const CRYPTO_DSA_PUBLIC_KEY    SSH_ServerKeys_DSA_2048b_256b_PublicKey;
extern const CRYPTO_DSA_PRIVATE_KEY   SSH_ServerKeys_DSA_2048b_256b_PrivateKey;
extern const CRYPTO_DSA_DOMAIN_PARAMS SSH_ServerKeys_DSA_2048b_256b_DomainParas;
extern const CRYPTO_DSA_PUBLIC_KEY    SSH_ServerKeys_DSA_3072b_256b_PublicKey;
extern const CRYPTO_DSA_PRIVATE_KEY   SSH_ServerKeys_DSA_3072b_256b_PrivateKey;
extern const CRYPTO_DSA_DOMAIN_PARAMS SSH_ServerKeys_DSA_3072b_256b_DomainParas;
extern const CRYPTO_ECDSA_PUBLIC_KEY  SSH_ServerKeys_ECDSA_P256_PublicKey;
extern const CRYPTO_ECDSA_PRIVATE_KEY SSH_ServerKeys_ECDSA_P256_PrivateKey;
extern const CRYPTO_ECDSA_PUBLIC_KEY  SSH_ServerKeys_ECDSA_P384_PublicKey;
extern const CRYPTO_ECDSA_PRIVATE_KEY SSH_ServerKeys_ECDSA_P384_PrivateKey;
extern const CRYPTO_ECDSA_PUBLIC_KEY  SSH_ServerKeys_ECDSA_P521_PublicKey;
extern const CRYPTO_ECDSA_PRIVATE_KEY SSH_ServerKeys_ECDSA_P521_PrivateKey;
extern const CRYPTO_EdDSA_PUBLIC_KEY  SSH_ServerKeys_Ed25519_PublicKey;
extern const CRYPTO_EdDSA_PRIVATE_KEY SSH_ServerKeys_Ed25519_PrivateKey;
extern const CRYPTO_EdDSA_PUBLIC_KEY  SSH_ServerKeys_Ed448_PublicKey;
extern const CRYPTO_EdDSA_PRIVATE_KEY SSH_ServerKeys_Ed448_PrivateKey;
```

```c
/**********************************************************************
*
*       Static data
*
**********************************************************************
*/

static U32 _aMemory[ALLOC_SIZE / sizeof(U32)];

/**********************************************************************
*
*       Static code
*
**********************************************************************
*/

/**********************************************************************
*
*       _SSH_GetECDSAPublicKey()
*
*  Function description
*    Get pointer to ECDSA public key.
*
*  Parameters
*    sName - Curve name corresponding to requested key.
*
*  Return value
*    == 0 - No public key for curve.
*    != 0 - Public key for curve.
*/
static const CRYPTO_ECDSA_PUBLIC_KEY * _SSH_GetECDSAPublicKey(const char *sName) {
  if (SSH_STRCMP(sName, "nistp256") == 0) {
    return &SSH_ServerKeys_ECDSA_P256_PublicKey;
  } else if (SSH_STRCMP(sName, "nistp384") == 0) {
    return &SSH_ServerKeys_ECDSA_P384_PublicKey;
  } else if (SSH_STRCMP(sName, "nistp521") == 0) {
    return &SSH_ServerKeys_ECDSA_P521_PublicKey;
  } else {
    return NULL;
  }
}

/**********************************************************************
*
*       _SSH_GetECDSAPrivateKey()
*
*  Function description
*    Get pointer to ECDSA private key.
*
*  Parameters
*    sName - Curve name corresponding to requested key.
*
*  Return value
*    == 0 - No private key for curve.
*    != 0 - Private key for curve.
*/
static const CRYPTO_ECDSA_PRIVATE_KEY * _SSH_GetECDSAPrivateKey(const char *sName) {
  if (SSH_STRCMP(sName, "nistp256") == 0) {
    return &SSH_ServerKeys_ECDSA_P256_PrivateKey;
  } else if (SSH_STRCMP(sName, "nistp384") == 0) {
    return &SSH_ServerKeys_ECDSA_P384_PrivateKey;
  } else if (SSH_STRCMP(sName, "nistp521") == 0) {
    return &SSH_ServerKeys_ECDSA_P521_PrivateKey;
  } else {
    return NULL;
  }
}

/**********************************************************************
*
*       _SSH_GetRSAPublicKey()
*
*  Function description
*    Get pointer to RSA public key.
*
```

```c
*  Parameters
*    sName - Method name for RSA public key.
*
*  Return value
*    == 0 - No public key for method.
*    != 0 - Public key for method.
*/
static const CRYPTO_RSA_PUBLIC_KEY * _SSH_GetRSAPublicKey(const char *sName) {
  if (SSH_STRCMP(sName, "rsa1024-sha1") == 0) {
    // Ephemeral key for key exchange
    return &SSH_ServerKeys_RSA_Temp_1024b_PublicKey;
  } else if (SSH_STRCMP(sName, "rsa2048-sha256") == 0) {
    // Ephemeral key for key exchange
    return &SSH_ServerKeys_RSA_Temp_2048b_PublicKey;
  } else {
    // Host key
    return &SSH_ServerKeys_RSA_Host_2048b_PublicKey;
  }
}

/**********************************************************************
*
*       _SSH_GetRSAPrivateKey()
*
*  Function description
*    Get pointer to RSA private key.
*
*  Parameters
*    sName - Method name for RSA private key.
*
*  Return value
*    == 0 - No private key for method.
*    != 0 - Private key for method.
*/
static const CRYPTO_RSA_PRIVATE_KEY * _SSH_GetRSAPrivateKey(const char *sName) {
  if (SSH_STRCMP(sName, "rsa1024-sha1") == 0) {
    // Ephemeral key for key exchange
    return &SSH_ServerKeys_RSA_Temp_1024b_PrivateKey;
  } else if (SSH_STRCMP(sName, "rsa2048-sha256") == 0) {
    // Ephemeral key for key exchange
    return &SSH_ServerKeys_RSA_Temp_2048b_PrivateKey;
  } else {
    return &SSH_ServerKeys_RSA_Host_2048b_PrivateKey;
  }
}

/**********************************************************************
*
*       _SSH_GetDSAPublicKey()
*
*  Function description
*    Get pointer to DSA public key.
*
*  Parameters
*    sName        - Method name for DSA public key.
*    ppPublicKey  - Address of pointer that receives the public key.
*    ppDomainParas - Address of pointer that receives the domain parameters.
*/
static void _SSH_GetDSAPublicKey(const char                      * sName,
                                 const CRYPTO_DSA_PUBLIC_KEY    ** ppPublicKey,
                                 const CRYPTO_DSA_DOMAIN_PARAMS ** ppDomainParas) {
  //
  if (SSH_STRCMP(sName, "ssh-dss") == 0) {
    *ppPublicKey   = &SSH_ServerKeys_DSA_2048b_160b_PublicKey;
    *ppDomainParas = &SSH_ServerKeys_DSA_2048b_160b_DomainParas;
  } else if (SSH_STRCMP(sName, "ssh-dss-sha256@ssh.com") == 0) {
    *ppPublicKey   = &SSH_ServerKeys_DSA_2048b_256b_PublicKey;
    *ppDomainParas = &SSH_ServerKeys_DSA_2048b_256b_DomainParas;
  } else {
    *ppPublicKey   = NULL;
    *ppDomainParas = NULL;
  }
}

/**********************************************************************
*
```

```
*        _SSH_GetDSAPrivateKey()
*
*  Function description
*    Get pointer to DSA private key.
*
*  Parameters
*    sName        - Method name for DSA private key.
*    ppPrivateKey - Address of pointer that receives the private key.
*    ppDomainParas - Address of pointer that receives the domain parameters.
*/
static void _SSH_GetDSAPrivateKey(const char                          * sName,
                                  const CRYPTO_DSA_PRIVATE_KEY   ** ppPrivateKey,
                                  const CRYPTO_DSA_DOMAIN_PARAMS ** ppDomainParas) {
  //
  if (SSH_STRCMP(sName, "ssh-dss") == 0) {
    *ppPrivateKey  = &SSH_ServerKeys_DSA_2048b_160b_PrivateKey;
    *ppDomainParas = &SSH_ServerKeys_DSA_2048b_160b_DomainParas;
  } else if (SSH_STRCMP(sName, "ssh-dss-sha256@ssh.com") == 0) {
    *ppPrivateKey  = &SSH_ServerKeys_DSA_2048b_256b_PrivateKey;
    *ppDomainParas = &SSH_ServerKeys_DSA_2048b_256b_DomainParas;
  } else {
    *ppPrivateKey  = NULL;
    *ppDomainParas = NULL;
  }
}

/*********************************************************************
*
*        _SSH_GetEdDSAPublicKey()
*
*  Function description
*    Get pointer to EdDSA public key.
*
*  Parameters
*    sName - Method name for EdDSA public key.
*
*  Return value
*    == 0 - No public key for method.
*    != 0 - Public key for method.
*/
static const CRYPTO_EdDSA_PUBLIC_KEY * _SSH_GetEdDSAPublicKey(const char *sName) {
  if (SSH_STRCMP(sName, "ssh-ed25519") == 0) {
    return &SSH_ServerKeys_Ed25519_PublicKey;
  } else if (SSH_STRCMP(sName, "ssh-ed448") == 0) {
    return &SSH_ServerKeys_Ed448_PublicKey;
  } else {
    return NULL;
  }
}

/*********************************************************************
*
*        _SSH_GetEdDSAPrivateKey()
*
*  Function description
*    Get pointer to EdDSA private key.
*
*  Parameters
*    sName - Method name for EdDSA private key.
*
*  Return value
*    == 0 - No private key for method.
*    != 0 - Private key for method.
*/
static const CRYPTO_EdDSA_PRIVATE_KEY  * _SSH_GetEdDSAPrivateKey(const char *sName) {
  if (SSH_STRCMP(sName, "ssh-ed25519") == 0) {
    return &SSH_ServerKeys_Ed25519_PrivateKey;
  } else if (SSH_STRCMP(sName, "ssh-ed448") == 0) {
    return &SSH_ServerKeys_Ed448_PrivateKey;
  } else {
    return NULL;
  }
}

/*********************************************************************
*
```

```
 *       Static const data
 *
 ***********************************************************************
 */

static const SSH_HOSTKEY_API _SSH_HostKeyAPI = {
  _SSH_GetRSAPublicKey,    _SSH_GetRSAPrivateKey,
  _SSH_GetECDSAPublicKey,  _SSH_GetECDSAPrivateKey,
  _SSH_GetEdDSAPublicKey,  _SSH_GetEdDSAPrivateKey,
  _SSH_GetDSAPublicKey,    _SSH_GetDSAPrivateKey,
};

/***********************************************************************
 *
 *       Public code
 *
 ***********************************************************************
 */

/***********************************************************************
 *
 *       SSH_X_Config()
 *
 *  Function description
 *    Installs everything required for SSH use; called from SSH_Init().
 */
void SSH_X_Config(void) {
  //
  SSH_MEM_Add(&_aMemory[0], sizeof(_aMemory));
  //
  // Set up the public key API.
  //
  SSH_SetHostKeyAPI(&_SSH_HostKeyAPI);
  //
  // Define log and warn filter
  // Note: The terminal I/O emulation might affect the timing
  // of your communication, since the debugger might stop the
  // target for every terminal I/O output depending on the used
  // I/O interface.
  //
  SSH_SetWarnFilter(~0U);                    // Output all warnings.
  SSH_SetLogFilter(0 * SSH_LOG_TRANSPORT +  // Change 0 to 1 in each of these
                   0 * SSH_LOG_SUITE     +  // to enable that log output.
                   0 * SSH_LOG_KEYS      +
                   0 * SSH_LOG_MESSAGE   +
                   1 * SSH_LOG_CONFIG    +
                   0 * SSH_LOG_SCP       +
                   1 * SSH_LOG_APP);
  //
  // The Userauth and Connection services are essential, without them
  // SSH will simply not work.
  //
  SSH_SERVICE_Add(&SSH_SERVICE_USERAUTH,   0);
  SSH_SERVICE_Add(&SSH_SERVICE_CONNECTION, 0);
  //
  // Key exchange algorithms.  Some are not added by default as they are a little slow
  //
#if INSTALL_RSA_KEY_EXCHANGE
  SSH_KEY_EXCHANGE_ALGORITHM_Add(&SSH_KEY_EXCHANGE_RSA1024_SHA1);
  SSH_KEY_EXCHANGE_ALGORITHM_Add(&SSH_KEY_EXCHANGE_RSA2048_SHA256);
#endif
  SSH_KEY_EXCHANGE_ALGORITHM_Add(&SSH_KEY_EXCHANGE_CURVE25519_SHA256);
  SSH_KEY_EXCHANGE_ALGORITHM_Add(&SSH_KEY_EXCHANGE_VENDOR_LIBSSH_CURVE25519_SHA256);
  SSH_KEY_EXCHANGE_ALGORITHM_Add(&SSH_KEY_EXCHANGE_CURVE448_SHA512);
  SSH_KEY_EXCHANGE_ALGORITHM_Add(&SSH_KEY_EXCHANGE_DH_GROUP1_SHA1);
  SSH_KEY_EXCHANGE_ALGORITHM_Add(&SSH_KEY_EXCHANGE_DH_GROUP14_SHA1);
  SSH_KEY_EXCHANGE_ALGORITHM_Add(&SSH_KEY_EXCHANGE_DH_GROUP14_SHA256);
  SSH_KEY_EXCHANGE_ALGORITHM_Add(&SSH_KEY_EXCHANGE_ECDH_SHA2_NISTP256);
  SSH_KEY_EXCHANGE_ALGORITHM_Add(&SSH_KEY_EXCHANGE_ECDH_SHA2_NISTP384);
  SSH_KEY_EXCHANGE_ALGORITHM_Add(&SSH_KEY_EXCHANGE_ECDH_SHA2_NISTP521);
  SSH_KEY_EXCHANGE_ALGORITHM_Add(&SSH_KEY_EXCHANGE_DH_GROUP_EXCHANGE_SHA1);
  SSH_KEY_EXCHANGE_ALGORITHM_Add(&SSH_KEY_EXCHANGE_DH_GROUP_EXCHANGE_SHA256);
#if INSTALL_SLOW_DH_GROUPS
  SSH_KEY_EXCHANGE_ALGORITHM_Add(&SSH_KEY_EXCHANGE_DH_GROUP16_SHA512);
  SSH_KEY_EXCHANGE_ALGORITHM_Add(&SSH_KEY_EXCHANGE_DH_GROUP18_SHA512);
#endif
```

```c
#if INSTALL_VENDOR_SCS_EXTENSIONS
  SSH_KEY_EXCHANGE_ALGORITHM_Add(&SSH_KEY_EXCHANGE_VENDOR_SCS_DH_GROUP14_SHA224);
  SSH_KEY_EXCHANGE_ALGORITHM_Add(&SSH_KEY_EXCHANGE_VENDOR_SCS_DH_GROUP14_SHA256);
  SSH_KEY_EXCHANGE_ALGORITHM_Add(&SSH_KEY_EXCHANGE_VENDOR_SCS_DH_GROUP_EXCHANGE_SHA224);
  SSH_KEY_EXCHANGE_ALGORITHM_Add(&SSH_KEY_EXCHANGE_VENDOR_SCS_DH_GROUP_EXCHANGE_SHA384);
  SSH_KEY_EXCHANGE_ALGORITHM_Add(&SSH_KEY_EXCHANGE_VENDOR_SCS_DH_GROUP_EXCHANGE_SHA512);
#if INSTALL_SLOW_DH_GROUPS
  SSH_KEY_EXCHANGE_ALGORITHM_Add(&SSH_KEY_EXCHANGE_VENDOR_SCS_DH_GROUP15_SHA256);
  SSH_KEY_EXCHANGE_ALGORITHM_Add(&SSH_KEY_EXCHANGE_VENDOR_SCS_DH_GROUP15_SHA384);
  SSH_KEY_EXCHANGE_ALGORITHM_Add(&SSH_KEY_EXCHANGE_VENDOR_SCS_DH_GROUP16_SHA384);
  SSH_KEY_EXCHANGE_ALGORITHM_Add(&SSH_KEY_EXCHANGE_VENDOR_SCS_DH_GROUP16_SHA512);
  SSH_KEY_EXCHANGE_ALGORITHM_Add(&SSH_KEY_EXCHANGE_VENDOR_SCS_DH_GROUP18_SHA512);
#endif
#endif
  //
  // Public key algorithms.
  //
  SSH_PUBLIC_KEY_ALGORITHM_Add(&SSH_PK_ALGORITHM_SSH_DSS);
  SSH_PUBLIC_KEY_ALGORITHM_Add(&SSH_PK_ALGORITHM_RSA_SHA2_512);
  SSH_PUBLIC_KEY_ALGORITHM_Add(&SSH_PK_ALGORITHM_RSA_SHA2_256);
  SSH_PUBLIC_KEY_ALGORITHM_Add(&SSH_PK_ALGORITHM_SSH_RSA);
  SSH_PUBLIC_KEY_ALGORITHM_Add(&SSH_PK_ALGORITHM_ECDSA_SHA2_NISTP256);
  SSH_PUBLIC_KEY_ALGORITHM_Add(&SSH_PK_ALGORITHM_ECDSA_SHA2_NISTP384);
  SSH_PUBLIC_KEY_ALGORITHM_Add(&SSH_PK_ALGORITHM_ECDSA_SHA2_NISTP521);
  SSH_PUBLIC_KEY_ALGORITHM_Add(&SSH_PK_ALGORITHM_SSH_ED25519);
#if INSTALL_VENDOR_SCS_EXTENSIONS
  SSH_PUBLIC_KEY_ALGORITHM_Add(&SSH_PK_ALGORITHM_VENDOR_SCS_SSH_DSS_SHA256);
  SSH_PUBLIC_KEY_ALGORITHM_Add(&SSH_PK_ALGORITHM_VENDOR_SCS_SSH_RSA_SHA224);
  SSH_PUBLIC_KEY_ALGORITHM_Add(&SSH_PK_ALGORITHM_VENDOR_SCS_SSH_RSA_SHA384);
  SSH_PUBLIC_KEY_ALGORITHM_Add(&SSH_PK_ALGORITHM_VENDOR_SCS_SSH_RSA_SHA256);
  SSH_PUBLIC_KEY_ALGORITHM_Add(&SSH_PK_ALGORITHM_VENDOR_SCS_SSH_RSA_SHA512);
#endif
  //
  // Encryption algorithms.
  //
  SSH_ENCRYPTION_ALGORITHM_Add(&SSH_ENCRYPTION_ALGORITHM_RC4);
  SSH_ENCRYPTION_ALGORITHM_Add(&SSH_ENCRYPTION_ALGORITHM_RC4_128);
  SSH_ENCRYPTION_ALGORITHM_Add(&SSH_ENCRYPTION_ALGORITHM_RC4_256);
  SSH_ENCRYPTION_ALGORITHM_Add(&SSH_ENCRYPTION_ALGORITHM_3DES_CBC);
  SSH_ENCRYPTION_ALGORITHM_Add(&SSH_ENCRYPTION_ALGORITHM_3DES_CTR);
  SSH_ENCRYPTION_ALGORITHM_Add(&SSH_ENCRYPTION_ALGORITHM_AES256_CBC);
  SSH_ENCRYPTION_ALGORITHM_Add(&SSH_ENCRYPTION_ALGORITHM_AES192_CBC);
  SSH_ENCRYPTION_ALGORITHM_Add(&SSH_ENCRYPTION_ALGORITHM_AES128_CBC);
  SSH_ENCRYPTION_ALGORITHM_Add(&SSH_ENCRYPTION_ALGORITHM_AES128_CTR);
  SSH_ENCRYPTION_ALGORITHM_Add(&SSH_ENCRYPTION_ALGORITHM_AES192_CTR);
  SSH_ENCRYPTION_ALGORITHM_Add(&SSH_ENCRYPTION_ALGORITHM_AES256_CTR);
#if INSTALL_VENDOR_OPENSSH_EXTENSIONS
  SSH_ENCRYPTION_ALGORITHM_Add(&SSH_ENCRYPTION_ALGORITHM_VENDOR_OPENSSH_AES128_GCM);
  SSH_ENCRYPTION_ALGORITHM_Add(&SSH_ENCRYPTION_ALGORITHM_VENDOR_OPENSSH_AES256_GCM);
  SSH_ENCRYPTION_ALGORITHM_Add(&SSH_ENCRYPTION_ALGORITHM_VENDOR_OPENSSH_CHACHA20_POLY1305);
#endif
  SSH_ENCRYPTION_ALGORITHM_Add(&SSH_ENCRYPTION_ALGORITHM_CAMELLIA128_CBC);
  SSH_ENCRYPTION_ALGORITHM_Add(&SSH_ENCRYPTION_ALGORITHM_CAMELLIA192_CBC);
  SSH_ENCRYPTION_ALGORITHM_Add(&SSH_ENCRYPTION_ALGORITHM_CAMELLIA256_CBC);
  SSH_ENCRYPTION_ALGORITHM_Add(&SSH_ENCRYPTION_ALGORITHM_CAMELLIA128_CTR);
  SSH_ENCRYPTION_ALGORITHM_Add(&SSH_ENCRYPTION_ALGORITHM_CAMELLIA192_CTR);
  SSH_ENCRYPTION_ALGORITHM_Add(&SSH_ENCRYPTION_ALGORITHM_CAMELLIA256_CTR);
  SSH_ENCRYPTION_ALGORITHM_Add(&SSH_ENCRYPTION_ALGORITHM_BLOWFISH_CBC);
  SSH_ENCRYPTION_ALGORITHM_Add(&SSH_ENCRYPTION_ALGORITHM_BLOWFISH_CTR);
  SSH_ENCRYPTION_ALGORITHM_Add(&SSH_ENCRYPTION_ALGORITHM_TWOFISH128_CBC);
  SSH_ENCRYPTION_ALGORITHM_Add(&SSH_ENCRYPTION_ALGORITHM_TWOFISH192_CBC);
  SSH_ENCRYPTION_ALGORITHM_Add(&SSH_ENCRYPTION_ALGORITHM_TWOFISH256_CBC);
  SSH_ENCRYPTION_ALGORITHM_Add(&SSH_ENCRYPTION_ALGORITHM_TWOFISH128_CTR);
  SSH_ENCRYPTION_ALGORITHM_Add(&SSH_ENCRYPTION_ALGORITHM_TWOFISH192_CTR);
  SSH_ENCRYPTION_ALGORITHM_Add(&SSH_ENCRYPTION_ALGORITHM_TWOFISH256_CTR);
  SSH_ENCRYPTION_ALGORITHM_Add(&SSH_ENCRYPTION_ALGORITHM_TWOFISH_CBC);
  SSH_ENCRYPTION_ALGORITHM_Add(&SSH_ENCRYPTION_ALGORITHM_CAST128_CBC);
  SSH_ENCRYPTION_ALGORITHM_Add(&SSH_ENCRYPTION_ALGORITHM_CAST128_CTR);
#if INSTALL_VENDOR_LIU_EXTENSIONS
  SSH_ENCRYPTION_ALGORITHM_Add(&SSH_ENCRYPTION_ALGORITHM_VENDOR_LIU_RIJNDAEL_CBC);
#endif
#if INSTALL_VENDOR_SCS_EXTENSIONS
  SSH_ENCRYPTION_ALGORITHM_Add(&SSH_ENCRYPTION_ALGORITHM_VENDOR_SCS_SEED_CBC);
#endif
  //
  // MAC algorithms.
```

```
    //
    SSH_MAC_ALGORITHM_Add(&SSH_MAC_ALGORITHM_HMAC_MD5);
    SSH_MAC_ALGORITHM_Add(&SSH_MAC_ALGORITHM_HMAC_MD5_96);
    SSH_MAC_ALGORITHM_Add(&SSH_MAC_ALGORITHM_HMAC_SHA1);
    SSH_MAC_ALGORITHM_Add(&SSH_MAC_ALGORITHM_HMAC_SHA1_96);
    SSH_MAC_ALGORITHM_Add(&SSH_MAC_ALGORITHM_HMAC_SHA2_256);
    SSH_MAC_ALGORITHM_Add(&SSH_MAC_ALGORITHM_HMAC_SHA2_512);
#if INSTALL_VENDOR_OPENSSH_EXTENSIONS
    SSH_MAC_ALGORITHM_Add(&SSH_MAC_ALGORITHM_VENDOR_OPENSSH_HMAC_RIPEMD160);
    SSH_MAC_ALGORITHM_Add(&SSH_MAC_ALGORITHM_VENDOR_OPENSSH_HMAC_MD5_ETM);
    SSH_MAC_ALGORITHM_Add(&SSH_MAC_ALGORITHM_VENDOR_OPENSSH_HMAC_SHA1_ETM);
    SSH_MAC_ALGORITHM_Add(&SSH_MAC_ALGORITHM_VENDOR_OPENSSH_HMAC_SHA2_256_ETM);
    SSH_MAC_ALGORITHM_Add(&SSH_MAC_ALGORITHM_VENDOR_OPENSSH_HMAC_SHA2_512_ETM);
    SSH_MAC_ALGORITHM_Add(&SSH_MAC_ALGORITHM_VENDOR_OPENSSH_HMAC_RIPEMD160_ETM);
#endif
#if INSTALL_VENDOR_SCS_EXTENSIONS
    SSH_MAC_ALGORITHM_Add(&SSH_MAC_ALGORITHM_VENDOR_SCS_HMAC_SHA224);
    SSH_MAC_ALGORITHM_Add(&SSH_MAC_ALGORITHM_VENDOR_SCS_HMAC_SHA256);
    SSH_MAC_ALGORITHM_Add(&SSH_MAC_ALGORITHM_VENDOR_SCS_HMAC_SHA384);
    SSH_MAC_ALGORITHM_Add(&SSH_MAC_ALGORITHM_VENDOR_SCS_HMAC_SHA512);
#endif
    //
    // Compression algorithms.
    //
    SSH_COMPRESSION_ALGORITHM_Add(&SSH_COMPRESSION_ALGORITHM_NONE);
}

/************************** End of file **************************/
```

Chapter 7

Configuring cryptography

7.1   Overview

In addition to configuring emSSH capabilities at the protocol level, it is necessary to configure how these are implemented by the shared cryptographic library, emCrypt.

emCrypt provides cryptographic services for all SEGGER security products (emSSL, emSSH, emSecure-RSA, and emSecure-ECDSA) and must be configured before it is used stand-alone or with one of these products.

There are two parts to emCrypt configuration:

- *Compile-time configuration* which defines the static configuration of the software, selecting the way each particular algorithm is implemented. Particular configurations are selected by setting various preprocessor symbols when compiling.
- *Runtime configuration* where the cryptographic algorithms are configured for higher-level clients such as emSSL and emSSH. Particilar configurations are selected by installing cryptographic services when initializing the cryptographic library.

The following sections describe compile-time and runtime configuation of emCrypt.

7.2   Compile-time configuration

In order to configure how compute-intensive cryptographic algorithms are compiled to balance code size and execution performance you can change the compile-time flags in the configuration file `CRYPTO_Conf.h`.

The default configuration strikes a balance between code size and execution speed and runs well on the broadest range of hardware.

> **Note**
>
> All user configuration of the cryptographic algorithms must be made in the file `CRYPTO_Conf.h` *and only that file*. Do not make any adjustments to the file `CRYPTO_ConfDefaults.h`.

7.2.1   Multiprecision integer, bits per limb

Default

```
#define CRYPTO_MPI_BITS_PER_LIMB      32
```

Override

To define a non-default value, define this symbol in `CRYPTO_Conf.h`.

Description

This preprocessor symbol configures the number of bits per limb for multiprecision integer algorithms. The default of 32 matches 32-bit targets well, such as ARM and PIC32. In general, it is best to set the number of bits per limb to the number of bits in the standard `int` or `unsigned` type used by the target compiler.

Supported configurations are:

* 32 — requires the target compiler to support 64-bit types natively (i.e. `unsigned long long` or `unsigned __int64`),
* 16 — which should run on any ISO compiler whose native integer types are 16 or 32 bit and supports 32-bit `unsigned long`.
* 8 — 8-bit limb sizes are supported and selecting this size may well lead to better multiplication performance on 8-bit architectures.

7.2.2   Hashes

7.2.2.1   MD5

Default

```
#define CRYPTO_CONFIG_MD5_OPTIMIZE      0
```

Override

To define a non-default value, define this symbol in `CRYPTO_Conf.h`.

Description

Set this preprocessor symbol to zero to optimize the MD5 hash functions for size rather than for speed. When optimized for speed, the MD5 function is open coded and faster, but is significantly larger.

Profile

The following table shows required context size, lookup table (LUT) size, and code size in kilobytes for each configuration value. All values are approximate and for a Cortex-M3 processor.

Setting	Context size	LUT	LUT size	Code size	Total size
0	0.16 KB	Flash	0.3 KB	0.4 KB	0.7 KB
1	0.16 KB	-	-	2.0 KB	2.0 KB

7.2.2.2   RIPEMD-160

Default

```
#define CRYPTO_CONFIG_RIPEMD160_OPTIMIZE    0
```

Override

To define a non-default value, define this symbol in `CRYPTO_Conf.h`.

Description

Set this preprocessor symbol to zero to optimize the RIPEMD-160 hash functions for size rather than for speed. When optimized for speed, the RIPEMD-160 function is open coded and faster, but is significantly larger.

Profile

The following table shows required context size, lookup table (LUT) size, and code size in kilobytes for each configuration value. All values are approximate and for a Cortex-M3 processor.

Setting	Context size	LUT	LUT size	Code size	Total size
0	0.16 KB	Flash	0.3 KB	0.7 KB	1.0 KB
1	0.16 KB	-	-	4.6 KB	4.6 KB

7.2.2.3   SHA-1

Default

```
#define CRYPTO_CONFIG_SHA1_OPTIMIZE      0
```

Override

To define a non-default value, define this symbol in `CRYPTO_Conf.h`.

Description

Set this preprocessor symbol to zero to optimize the SHA-1 hash functions for size rather than for speed. When optimized for speed, the SHA-1 function is open coded and faster, but is significantly larger.

Profile

The following table shows required context size, lookup table (LUT) size, and code size in kilobytes for each configuration value. All values are approximate and for a Cortex-M4 processor.

Setting	Context size	LUT	LUT size	Code size	Total size
0	0.16 KB	-	-	0.6 KB	0.6 KB
1	0.16 KB	-	-	3.6 KB	3.6 KB

7.2.2.4   SHA-256

Default

```
#define CRYPTO_CONFIG_SHA256_OPTIMIZE      0
```

Override

To define a non-default value, define this symbol in `CRYPTO_Conf.h`.

Description

Set this preprocessor symbol to zero to optimize the SHA-256 hash functions for size rather than for speed. When optimized for speed, the SHA-256 function is open coded and faster, but is significantly larger.

Profile

The following table shows required context size, lookup table (LUT) size, and code size in kilobytes for each configuration value. All values are approximate and for a Cortex-M3 processor.

Setting	Context size	LUT	LUT size	Code size	Total size
0	0.17 KB	Flash	0.3 KB	0.5 KB	0.8 KB
1	0.17 KB	-	-	7.7 KB	7.7 KB

7.2.2.5   SHA-512

Default

```
#define CRYPTO_CONFIG_SHA512_OPTIMIZE      0
```

Override

To define a non-default value, define this symbol in `CRYPTO_Conf.h`.

Description

Set this preprocessor symbol to zero to optimize the SHA-512 hash functions for size rather than for speed. When optimized for speed, the SHA-512 function is open coded and faster, but is significantly larger.

Profile

The following table shows required context size, lookup table (LUT) size, and code size in kilobytes for each configuration value. All values are approximate and for a Cortex-M3 processor.

Setting	Context size	LUT	LUT size	Code size	Total size
0	0.20 KB	Flash	0.7 KB	1.1 KB	1.8 KB
1	0.20 KB	Flash	0.7 KB	10.3 KB	11.0 KB
2	0.20 KB	Flash	0.1 KB	41.5 KB	41.6 KB

7.2.3   Ciphers

7.2.3.1   AES

Default

```
#define CRYPTO_CONFIG_AES_OPTIMIZE      2
```

Override

To define a non-default value, define this symbol in `CRYPTO_Conf.h`.

Description

Set this preprocessor symbol nonzero to optimize AES to use tables for matrix multiplication. Optimization levels are 0 through 7 with larger numbers generally producing better performance.

Profile

The following table shows required context size, lookup table (LUT) size, and code size in kilobytes for each configuration value. All values are approximate and for a Cortex-M3 processor.

Setting	Context size	LUT	LUT size	Code size	Total size
0	0.24 KB	Flash	2.0 KB	3.2 KB	5.2 KB
1	0.24 KB	Flash	2.0 KB	2.7 KB	4.7 KB
2	0.24 KB	Flash	8.5 KB	2.4 KB	10.9 KB
3	0.24 KB	Flash	1.9 KB	12.5 KB	14.4 KB
4	0.24 KB	RAM	2.0 KB	3.2 KB	5.2 KB
5	0.24 KB	RAM	2.0 KB	2.7 KB	4.7 KB
6	0.24 KB	RAM	8.5 KB	2.4 KB	10.9 KB
7	0.24 KB	RAM	1.9 KB	12.5 KB	14.4 KB

7.2.3.2   DES

Default

```
#define CRYPTO_CONFIG_DES_OPTIMIZE      0
```

Override

To define a non-default value, define this symbol in `CRYPTO_Conf.h`.

Description

Set this preprocessor symbol nonzero to optimize DES and 3DES to place tables in RAM rather than flash. Optimization levels are 0 through 5 with larger numbers generally producing better performance.

Profile

The following table shows required context size, lookup table (LUT) size, and code size in kilobytes for each configuration value. All values are approximate and for a Cortex-M3 processor.

Setting	Context size	LUT	LUT size	Code size	Total size
0	0.38 KB	Flash	2.1 KB	1.3 KB	3.4 KB
1	0.38 KB	Flash	2.1 KB	2.1 KB	4.2 KB
2	0.38 KB	Flash	2.1 KB	5.3 KB	7.4 KB
3	0.38 KB	RAM	2.1 KB	1.3 KB	3.4 KB
4	0.38 KB	RAM	2.1 KB	2.1 KB	4.2 KB
5	0.38 KB	RAM	2.1 KB	5.3 KB	7.4 KB

7.2.3.3   SEED

Default

```
#define CRYPTO_CONFIG_SEED_OPTIMIZE      0
```

Override

To define a non-default value, define this symbol in `CRYPTO_Conf.h`.

Description

Set this preprocessor symbol nonzero to optimize SEED to place tables in RAM rather than flash and to optimized the table sizes. Optimization levels are 0 through 3 with larger numbers generally producing better performance.

Profile

The following table shows required context size, lookup table (LUT) size, and code size in kilobytes for each configuration value. All values are approximate and for a Cortex-M3 processor.

Setting	Context size	LUT	LUT size	Code size	Total size
0	0.14 KB	Flash	0.5 KB	0.5 KB	1.0 KB
1	0.14 KB	Flash	4.0 KB	0.4 KB	4.4 KB
2	0.14 KB	RAM	0.5 KB	0.5 KB	1.0 KB
3	0.14 KB	RAM	4.0 KB	0.4 KB	4.4 KB

7.2.3.4   ARIA

Default

```
#define CRYPTO_CONFIG_ARIA_OPTIMIZE      0
```

Override

To define a non-default value, define this symbol in `CRYPTO_Conf.h`.

Description

Set this preprocessor symbol nonzero to optimize ARIA to place tables in RAM rather than flash.

Profile

The following table shows required context size, lookup table (LUT) size, and code size in kilobytes for each configuration value. All values are approximate and for a Cortex-M3 processor.

Setting	Context size	LUT	LUT size	Code size	Total size
0	0.28 KB	Flash	1.0 KB	1.9 KB	2.9 KB
1	0.28 KB	RAM	1.0 KB	1.9 KB	2.9 KB

7.2.3.5   CAST

Default

```
#define CRYPTO_CONFIG_CAST_OPTIMIZE       0
```

Override

To define a non-default value, define this symbol in `CRYPTO_Conf.h`.

Description

Set this preprocessor symbol nonzero to optimize CAST to place tables in RAM rather than flash. Optimization levels are 0 through 1 with larger numbers generally producing better performance.

Profile

The following table shows required context size, lookup table (LUT) size, and code size in kilobytes for each configuration value. All values are approximate and for a Cortex-M3 processor.

Setting	Context size	LUT	LUT size	Code size	Total size
0	0.10 KB	Flash	8.0 KB	3.5 KB	11.5 KB
1	0.10 KB	RAM	8.0 KB	3.7 KB	11.7 KB

7.2.3.6 Camellia

Default

```
#define CRYPTO_CONFIG_CAMELLIA_OPTIMIZE       0
```

Override

To define a non-default value, define this symbol in `CRYPTO_Conf.h`.

Description

Set this preprocessor symbol nonzero to optimize Camellia to use more efficient tables. Optimization levels are 0 (smallest) to 3 (fastest).

Profile

The following table shows required context size, lookup table (LUT) size, and code size in kilobytes for each configuration value. All values are approximate and for a Cortex-M3 processor.

Setting	Context size	LUT	LUT size	Code size	Total size
0	0.27 KB	Flash	1.0 KB	28.8 KB	29.8 KB
1	0.27 KB	Flash	4.0 KB	20.7 KB	24.7 KB
2	0.27 KB	RAM	1.0 KB	28.8 KB	29.8 KB
3	0.27 KB	RAM	4.0 KB	20.7 KB	24.7 KB

7.2.3.7   Blowfish

Default

```
#define CRYPTO_CONFIG_BLOWFISH_OPTIMIZE      0
```

Override

To define a non-default value, define this symbol in `CRYPTO_Conf.h`.

Description

Set this preprocessor symbol nonzero to optimize Blowfish to use more efficient tables. Optimization levels are 0 to 1.

Profile

The following table shows required context size, lookup table (LUT) size, and code size in kilobytes for each configuration value. All values are approximate and for a Cortex-M3 processor.

Setting	Context size	LUT	LUT size	Code size	Total size
0	4.0 KB	Flash	4.0 KB	0.7 KB	4.7 KB
1	4.0 KB	RAM	4.0 KB	1.1 KB	5.1 KB

7.2.3.8 Twofish

Default

```
#define CRYPTO_CONFIG_TWOFISH_OPTIMIZE      1
```

Override

To define a non-default value, define this symbol in `CRYPTO_Conf.h`.

Description

Set this preprocessor symbol nonzero to optimize Twofish to use more efficient tables. Optimization levels are 0 (smallest) to 15 (fastest).

Profile

The following table shows required context size, lookup table (LUT) size, and code size in kilobytes for each configuration value. All values are approximate and for a Cortex-M3 processor.

Setting	Context size	LUT	LUT size	Code size	Total size
0	0.2 KB	Flash	0.6 KB	3.4 KB	4.0 KB
1	0.2 KB	Flash	4.6 KB	3.1 KB	7.7 KB
2	0.2 KB	Flash	8.5 KB	3.2 KB	11.7 KB
3	0.2 KB	Flash	12.5 KB	2.8 KB	15.3 KB
4	4.2 KB	Flash	0.6 KB	3.4 KB	4.0 KB
5	4.2 KB	Flash	4.6 KB	3.1 KB	7.7 KB
6	4.2 KB	Flash	8.5 KB	3.2 KB	11.7 KB
7	4.2 KB	Flash	12.5 KB	2.8 KB	15.3 KB
8	0.2 KB	RAM	0.6 KB	3.4 KB	4.0 KB
9	0.2 KB	RAM	4.6 KB	3.1 KB	7.7 KB
10	0.2 KB	RAM	8.5 KB	3.2 KB	11.7 KB
11	0.2 KB	RAM	12.5 KB	2.8 KB	15.3 KB
12	4.2 KB	RAM	0.6 KB	3.4 KB	4.0 KB
13	4.2 KB	RAM	4.6 KB	3.1 KB	7.7 KB
14	4.2 KB	RAM	8.5 KB	3.2 KB	11.7 KB
15	4.2 KB	RAM	12.5 KB	2.8 KB	15.3 KB

7.3   Runtime configuration

The ciphers and hash functions that clients require must be installed as part of emCrypt configuration. The function `CRYPTO_X_Conf()` is called from `CRYPTO_Init()` to install any required cryptographic support.

emCrypt provides software implementations of all ciphers and hashes, but also supports plug-in hardware accelerators.

7.3.1   Hashes

emCrypt provides the following software implementations of the following hash algorithms for emSSH:

* MD5
* RIPEMD-160
* SHA-1
* SHA-256
* SHA-512

This section summarizes how to install the software hash implementations. For details on how to plug in hardware-assisted hash algorithms for a particular device, see *Plug-in hardware accelerators* on page 259.

7.3.1.1   MD5

Prototype

```
extern const CRYPTO_HASH_API CRYPTO_HASH_MD5_SW;
```

Description

This API provides a software-only implementation of MD5.

Installation

```
void CRYPTO_X_Conf(void) {
  CRYPTO_MD5_Install(&CRYPTO_HASH_MD5_SW, NULL);
}
```

See also

See *MD5* on page 233 for details on how to configure the performance and footprint of this algorithm.

7.3.1.2   RIPEMD-160

Prototype

```
extern const CRYPTO_HASH_API CRYPTO_HASH_RIPEMD160_SW;
```

Description

This API provides a software-only implementation of RIPEMD-160.

Installation

```
void CRYPTO_X_Conf(void) {
  CRYPTO_RIPEMD160_Install(&CRYPTO_HASH_RIPEMD160_SW, NULL);
}
```

See also

See *RIPEMD-160* on page 234 for details on how to configure the performance and footprint of this algorithm.

7.3.1.3   SHA-1

Prototype

```
extern const CRYPTO_HASH_API CRYPTO_HASH_SHA1_SW;
```

Description

This API provides a software-only implementation of SHA-1.

Installation

```
void CRYPTO_X_Conf(void) {
  CRYPTO_SHA1_Install(&CRYPTO_HASH_SHA1_SW, NULL);
}
```

See also

See *SHA-1* on page 235 for details on how to configure the performance and footprint of this algorithm.

7.3.1.4   SHA-256

Prototype

```
extern const CRYPTO_HASH_API CRYPTO_HASH_SHA256_SW;
```

Description

This API provides a software-only implementation of SHA-256 and SHA-224.

Installation

```
void CRYPTO_X_Conf(void) {
  CRYPTO_SHA256_Install(&CRYPTO_HASH_SHA256_SW, NULL);
}
```

See also

See *SHA-256* on page 236 for details on how to configure the performance and footprint of this algorithm.

7.3.1.5   SHA-512

Prototype

```
extern const CRYPTO_HASH_API CRYPTO_HASH_SHA512_SW;
```

Description

This API provides a software-only implementation of SHA-512 and SHA-384.

Installation

```
void CRYPTO_X_Conf(void) {
  CRYPTO_SHA512_Install(&CRYPTO_HASH_SHA512_SW, NULL);
}
```

See also

See *SHA-512* on page 237 for details on how to configure the performance and footprint of this algorithm.

7.3.2   Ciphers

emCrypt provides the following software implementations of the following cipher algorithms for emSSH:

- AES
- DES
- SEED
- ARIA
- CAST
- Camellia
- Blowfish
- Twofish

This section summarizes how to install the software cipher implementations. For details on how to plug in hardware-assisted cipher algorithms for a particular device, see *Plug-in hardware accelerators* on page 259.

7.3.2.1   AES

Prototype

```
extern const CRYPTO_CIPHER_API CRYPTO_CIPHER_AES_SW;
```

Description

This API provides a software-only implementation of AES.

Installation

```
void CRYPTO_X_Conf(void) {
  CRYPTO_AES_Install(&CRYPTO_CIPHER_AES_SW, NULL);
}
```

See also

See *AES* on page 238 for details on how to configure the performance and footprint of this algorithm.

7.3.2.2   DES

Prototype

```
extern const CRYPTO_CIPHER_API CRYPTO_CIPHER_TDES_SW;
```

Description

This API provides a software-only implementation of DES.

Installation

```
void CRYPTO_X_Conf(void) {
  CRYPTO_TDES_Install(&CRYPTO_CIPHER_TDES_SW, NULL);
}
```

See also

See *DES* on page 239 for details on how to configure the performance and footprint of this algorithm.

7.3.2.3   SEED

Prototype

```
extern const CRYPTO_CIPHER_API CRYPTO_CIPHER_SEED_SW;
```

Description

This API provides a software-only implementation of SEED.

Installation

```
void CRYPTO_X_Conf(void) {
  CRYPTO_SEED_Install(&CRYPTO_CIPHER_SEED_SW, NULL);
}
```

See also

See *SEED* on page 240 for details on how to configure the performance and footprint of this algorithm.

7.3.2.4   ARIA

Prototype

```
extern const CRYPTO_CIPHER_API CRYPTO_CIPHER_ARIA_SW;
```

Description

This API provides a software-only implementation of ARIA.

Installation

```
void CRYPTO_X_Conf(void) {
  CRYPTO_ARIA_Install(&CRYPTO_CIPHER_ARIA_SW, NULL);
}
```

See also

See *ARIA* on page 241 for details on how to configure the performance and footprint of this algorithm.

7.3.2.5   CAST

Prototype

```
extern const CRYPTO_CIPHER_API CRYPTO_CIPHER_CAST_SW;
```

Description

This API provides a software-only implementation of CAST.

Installation

```
void CRYPTO_X_Conf(void) {
  CRYPTO_CAST_Install(&CRYPTO_CIPHER_CAST_SW, NULL);
}
```

See also

See *CAST* on page 242 for details on how to configure the performance and footprint of this algorithm.

7.3.2.6   Camellia

Prototype

```
extern const CRYPTO_CIPHER_API CRYPTO_CIPHER_CAMELLIA_SW;
```

Description

This API provides a software-only implementation of Camellia.

Installation

```
void CRYPTO_X_Conf(void) {
  CRYPTO_CAMELLIA_Install(&CRYPTO_CIPHER_CAMELLIA_SW, NULL);
}
```

See also

See *Camellia* on page 243 for details on how to configure the performance and footprint of this algorithm.

7.3.2.7   Blowfish

Prototype

```
extern const CRYPTO_CIPHER_API CRYPTO_CIPHER_BLOWFISH_SW;
```

Description

This API provides a software-only implementation of Blowfish.

Installation

```
void CRYPTO_X_Conf(void) {
  CRYPTO_BLOWFISH_Install(&CRYPTO_CIPHER_BLOWFISH_SW, NULL);
}
```

See also

See *Blowfish* on page 244 for details on how to configure the performance and footprint of this algorithm.

7.3.2.8   Twofish

Prototype

```
extern const CRYPTO_CIPHER_API CRYPTO_CIPHER_TWOFISH_SW;
```

Description

This API provides a software-only implementation of Twofish.

Installation

```
void CRYPTO_X_Conf(void) {
  CRYPTO_TWOFISH_Install(&CRYPTO_CIPHER_TWOFISH_SW, NULL);
}
```

See also

See *Twofish* on page 245 for details on how to configure the performance and footprint of this algorithm.

7.3.3   Plug-in hardware accelerators

SEGGER security products are written in a way such that underlying cryptographic operations can be exchanged in order to benefit from hardware acceleration or vendor libraries optimized for a particular device.

emSSH requires no additional hardware in order to execute its underlying cryptographic operations: public key algorithms, bulk encipherment, and message authentication are completely implemented in software. However, there are many devices that offer hardware acceleration for one or more of these operations and there is nothing that prevents emSSH from utilizing any such capability.

For further information on hardware acceleration, refer to the following sections:

- *LPC18S and LPC43S AES ROM (Add-on)* on page 260
- *Kinetis CAU coprocessor (Add-on)* on page 262
- *STM32 CRYP coprocessor (Add-on)* on page 268
- *STM32 AES coprocessor (Add-on)* on page 271
- *STM32 HASH coprocessor (Add-on)* on page 272
- *EFM32 CRYPTO coprocessor (Add-on)* on page 275

For background information on hardware acceleration, refer to *Hardware acceleration* on page 409.

7.3.3.1   LPC18S and LPC43S AES ROM (Add-on)

The LPC18S*xx* and LPC43S*xx* microcontrollers provide an AES-128 hardware accelerator. The capabilities of this accelerator are exposed through a ROM-based API which insulates the programmer from changes to or variants of the underlying accelerator hardware.

emSSH has specialized hardware-assisted AES ciphering for the following cryptographic algorithms:

*   AES-128 in ECB and CBC modes.

All other AES-128 cipher modes (e.g. AES-GCM and AES-CCM) use hardware-assisted ciphering of individual blocks with software managing the cipher mode. All ciphering with AES-192 and AES-256 falls back to using a pure software AES kernel.

7.3.3.1.1   Installing LPC ROM hardware support

The following hardware-assisted interfaces are available:

```
extern const CRYPTO_CIPHER_API CRYPTO_CIPHER_AES_HW_LPC_ROM;
```

If all you require is AES-128, you can install hardware support using:

```
void CRYPTO_X_Config(void) {
  CRYPTO_AES_Install(&CRYPTO_CIPHER_AES_HW_LPC_ROM, 0);
}
```

However, if you require AES-192 or AES-256 in addition to AES-128, you must install a software fallback for these key sizes:

```
void CRYPTO_X_Config(void) {
  CRYPTO_AES_Install(&CRYPTO_CIPHER_AES_HW_LPC_ROM,
                     &CRYPTO_CIPHER_AES_SW);
}
```

7.3.3.1.2   LPC cryptographic units

The emSSH implementation of hardware assistance requires one cryptographic unit with index #0 that covers ciphering. See *CRYPTO-OS integration* on page 326 for further details.

7.3.3.1.3 Sample LPS18S setup

The following is the cryptographic setup for the NXP LPCXpresso18S37 board:

```c
/**********************************************************************
*               (c) SEGGER Microcontroller GmbH & Co. KG             *
*                     The Embedded Experts                           *
*                        www.segger.com                              *
**********************************************************************


----------------------- END-OF-HEADER -----------------------------

File        : CRYPTO_X_Config_LPC18S37.c
Purpose     : Configure CRYPTO for LPC18S37 devices.

*/

/**********************************************************************
*
*       #include Section
*
**********************************************************************
*/

#include "CRYPTO.h"

/**********************************************************************
*
*       Public code
*
**********************************************************************
*/

/**********************************************************************
*
*       CRYPTO_X_Panic()
*
*  Function description
*    Hang when something unexpected happens.
*/
void CRYPTO_X_Panic(void) {
  for (;;) {
    /* Hang */
  }
}

/**********************************************************************
*
*       CRYPTO_X_Config()
*
*  Function description
*    Configure hardware assist for CRYPTO component.
*/
void CRYPTO_X_Config(void) {
  CRYPTO_AES_Install       (&CRYPTO_CIPHER_AES_HW_LPC_ROM, &CRYPTO_CIPHER_AES_SW);
  CRYPTO_TDES_Install      (&CRYPTO_CIPHER_TDES_SW,        0);
  CRYPTO_MD5_Install       (&CRYPTO_HASH_MD5_SW,           0);
  CRYPTO_SHA1_Install      (&CRYPTO_HASH_SHA1_SW,          0);
  CRYPTO_SHA256_Install    (&CRYPTO_HASH_SHA256_SW,        0);
  CRYPTO_SHA512_Install    (&CRYPTO_HASH_SHA512_SW,        0);
  CRYPTO_RIPEMD160_Install (&CRYPTO_HASH_RIPEMD160_SW,     0);
}

/************************** End of file ****************************/
```

7.3.3.2   Kinetis CAU coprocessor (Add-on)

The Kinetis Cryptographic Acceleration Unit (CAU) is a primitive accelerator presented as a memory-mapped peripheral.

emSSH has specialized hardware-assisted ciphering and hashing support for the following cryptographic algorithms using the CAU:

- TDES in ECB and CBC modes with keying options 1, 2, and 3.
- AES-128, AES-192, and AES-256 in ECB and CBC modes.
- MD5
- SHA-1
- SHA-256

All other cipher modes (e.g. AES-GCM and AES-CCM) use hardware-assisted ciphering of individual blocks with software manging the cipher mode.

7.3.3.2.1   Installing CAU hardware support

The following hardware-assisted interfaces are available:

```
extern const CRYPTO_CIPHER_API CRYPTO_CIPHER_AES_HW_Kinetis_CAU;
extern const CRYPTO_CIPHER_API CRYPTO_CIPHER_TDES_HW_Kinetis_CAU;
extern const CRYPTO_HASH_API   CRYPTO_HASH_MD5_HW_Kinetis_CAU;
extern const CRYPTO_HASH_API   CRYPTO_HASH_SHA1_HW_Kinetis_CAU;
extern const CRYPTO_HASH_API   CRYPTO_HASH_SHA224_HW_Kinetis_CAU;
extern const CRYPTO_HASH_API   CRYPTO_HASH_SHA256_HW_Kinetis_CAU;
```

You can install hardware support using:

```
void CRYPTO_X_Config(void) {
  CRYPTO_MD5_Install   (&CRYPTO_HASH_MD5_HW_Kinetis_CAU,    NULL);
  CRYPTO_SHA1_Install  (&CRYPTO_HASH_SHA1_HW_Kinetis_CAU,   NULL);
  CRYPTO_SHA224_Install(&CRYPTO_HASH_SHA224_HW_Kinetis_CAU, NULL);
  CRYPTO_SHA256_Install(&CRYPTO_HASH_SHA256_HW_Kinetis_CAU, NULL);
  CRYPTO_AES_Install   (&CRYPTO_CIPHER_AES_HW_Kinetis_CAU,  NULL);
  CRYPTO_TDES_Install  (&CRYPTO_CIPHER_TDES_HW_Kinetis_CAU, NULL);
}
```

> **Note**
>
> Whilst there is an MD5 accelerator, hardware-assisted MD5 is slower than a pure software implementation of MD5 using Thumb-2 so we recommend that you do not install the MD5 accelerator.

7.3.3.2.2   Kinetis cryptographic units

The emSSH implementation of hardware assistance requires one cryptographic unit with index #0 covering both ciphering and hashing. See *CRYPTO-OS integration* on page 326 for further details.

7.3.3.2.3 Sample Kinetis setup

The following is the cryptographic setup for the SEGGER emPower board based on the Kinetis K66 device.

```c
/**********************************************************************
*                (c) SEGGER Microcontroller GmbH & Co. KG           *
*                        The Embedded Experts                       *
*                          www.segger.com                           *
**********************************************************************

------------------------- END-OF-HEADER ----------------------------
File      : CRYPTO_X_Config_K66.c
Purpose   : Configure CRYPTO for K66 devices.

*/

/**********************************************************************
*
*       #include Section
*
**********************************************************************
*/

#include "CRYPTO.h"

/**********************************************************************
*
*       Public code
*
**********************************************************************
*/

/**********************************************************************
*
*       CRYPTO_X_Panic()
*
*  Function description
*    Hang when something unexpected happens.
*/
void CRYPTO_X_Panic(void) {
  for (;;) {
    /* Hang */
  }
}

/**********************************************************************
*
*       CRYPTO_X_Config()
*
*  Function description
*    Configure hardware assist for CRYPTO component.
*/
void CRYPTO_X_Config(void) {
  volatile U32 *pReg;
  //
  // Install hardware assistance.
  //
  CRYPTO_MD5_Install      (&CRYPTO_HASH_MD5_HW_Kinetis_CAU,    NULL);
  CRYPTO_SHA1_Install     (&CRYPTO_HASH_SHA1_HW_Kinetis_CAU,   NULL);
  CRYPTO_SHA224_Install   (&CRYPTO_HASH_SHA224_HW_Kinetis_CAU, NULL);
  CRYPTO_SHA256_Install   (&CRYPTO_HASH_SHA256_HW_Kinetis_CAU, NULL);
  CRYPTO_AES_Install      (&CRYPTO_CIPHER_AES_HW_Kinetis_CAU,  NULL);
  CRYPTO_TDES_Install     (&CRYPTO_CIPHER_TDES_HW_Kinetis_CAU, NULL);
  //
  // Software ciphers.
  //
```

```
  CRYPTO_CAST_Install     (&CRYPTO_CIPHER_CAST_SW,      NULL);
  CRYPTO_SEED_Install     (&CRYPTO_CIPHER_SEED_SW,      NULL);
  CRYPTO_ARIA_Install     (&CRYPTO_CIPHER_ARIA_SW,      NULL);
  CRYPTO_CAMELLIA_Install (&CRYPTO_CIPHER_CAMELLIA_SW, NULL);
  CRYPTO_BLOWFISH_Install (&CRYPTO_CIPHER_BLOWFISH_SW, NULL);
  CRYPTO_TWOFISH_Install  (&CRYPTO_CIPHER_TWOFISH_SW,  NULL);
  //
  // Software hashing.
  //
  CRYPTO_SHA512_Install    (&CRYPTO_HASH_SHA512_SW,     NULL);
  CRYPTO_RIPEMD160_Install(&CRYPTO_HASH_RIPEMD160_SW, NULL);
  //
  // Turn on clocks to RNGA, bit 0 of SIM_SCGC3, and install RNG.
  //
  pReg = (void *)0x40048030;
  *pReg |= 1;
  //
  // Install Hash_DRBG-SHA-256 with RNGA entropy.
  //
  CRYPTO_RNG_InstallEx(&CRYPTO_RNG_DRBG_HASH_SHA256, &CRYPTO_RNG_HW_Kinetis_RNGA);
  //
  // Install small modular exponentiation functions.
  //
  CRYPTO_MPI_SetPublicModExp (CRYPTO_MPI_ModExp_Basic_Fast);
  CRYPTO_MPI_SetPrivateModExp(CRYPTO_MPI_ModExp_Basic_Fast);
}

/*************************** End of file ***************************/
```

7.3.3.3   iMX RT10xx data coprocessor (Add-on)

The iMX RT10xx Data Coprocessor (DCP) is a programmable crotographic accelerator presented as a memory-mapped peripheral.

emSSH has specialized hardware-assisted ciphering and hashing support for the following cryptographic algorithms using the DCP:

* AES-128 in ECB and CBC modes.
* SHA-1
* SHA-256

All other cipher modes (e.g. AES-GCM and AES-CCM) use hardware-assisted ciphering of individual blocks with software manging the cipher mode.

7.3.3.3.1   Installing iMX RT10xx hardware support

The following hardware-assisted interfaces are available:

```
extern const CRYPTO_CIPHER_API CRYPTO_CIPHER_AES_HW_RT10xx_DCP;
extern const CRYPTO_HASH_API   CRYPTO_HASH_SHA1_HW_RT10xx_DCP;
extern const CRYPTO_HASH_API   CRYPTO_HASH_SHA256_HW_RT10xx_DCP;
```

You can install hardware support using:

```
void CRYPTO_X_Config(void) {
  CRYPTO_SHA1_Install  (&CRYPTO_HASH_SHA1_HW_RT10xx_DCP,
                         &CRYPTO_HASH_SHA1_SW);
  CRYPTO_SHA256_Install(&CRYPTO_HASH_SHA256_HW_RT10xx_DCP,
                         &CRYPTO_HASH_SHA256_SW);
  CRYPTO_AES_Install   (&CRYPTO_CIPHER_AES_HW_RT10xx_DCP,
                         &CRYPTO_CIPHER_AES_SW);
  //
  // Install Hash_DRBG-SHA-256 with TRNG entropy.
  //
  CRYPTO_RNG_InstallEx(&CRYPTO_RNG_DRBG_HASH_SHA256,
                        &CRYPTO_RNG_HW_RT10xx_TRNG);
}
```

7.3.3.3.2   RT10xx cryptographic units

The emSSH implementation of hardware assistance requires one cryptographic unit with index #0 covering both ciphering and hashing. See *CRYPTO-OS integration* on page 326 for further details.

7.3.3.3.3   Sample Kinetis setup

The following is the cryptographic setup for the SEGGER RT1051 Trace Reference board.

```c
/**********************************************************************
*                  (c) SEGGER Microcontroller GmbH                    *
*                       The Embedded Experts                          *
*                          www.segger.com                             *
***********************************************************************

----------------------- END-OF-HEADER -----------------------------

File        : CRYPTO_X_Config_RT10xx.c
Purpose     : Configure CRYPTO for iMX RT10xx devices.

*/

/**********************************************************************
*
*       #include Section
*
***********************************************************************
*/

#include "CRYPTO.h"

/**********************************************************************
*
*       Public code
*
***********************************************************************
*/

/**********************************************************************
*
*       CRYPTO_X_Panic()
*
*  Function description
*    Hang when something unexpected happens.
*/
void CRYPTO_X_Panic(void) {
  for (;;) {
    /* Hang */
  }
}

/**********************************************************************
*
*       CRYPTO_X_Config()
*
*  Function description
*    Configure hardware assist for CRYPTO component.
*/
void CRYPTO_X_Config(void) {
  //
  // Install hardware assistance.
  //
  CRYPTO_SHA1_Install     (&CRYPTO_HASH_SHA1_HW_RT10xx_DCP,
  &CRYPTO_HASH_SHA1_SW);
  CRYPTO_SHA256_Install
  (&CRYPTO_HASH_SHA256_HW_RT10xx_DCP, &CRYPTO_HASH_SHA256_SW);
  CRYPTO_AES_Install      (&CRYPTO_CIPHER_AES_HW_RT10xx_DCP,
  &CRYPTO_CIPHER_AES_SW);
  //
  // Software ciphers.
  //
  CRYPTO_TDES_Install     (&CRYPTO_CIPHER_TDES_SW,      NULL);
```

```
    CRYPTO_CAST_Install     (&CRYPTO_CIPHER_CAST_SW,     NULL);
    CRYPTO_SEED_Install     (&CRYPTO_CIPHER_SEED_SW,     NULL);
    CRYPTO_ARIA_Install     (&CRYPTO_CIPHER_ARIA_SW,     NULL);
    CRYPTO_CAMELLIA_Install (&CRYPTO_CIPHER_CAMELLIA_SW, NULL);
    CRYPTO_BLOWFISH_Install (&CRYPTO_CIPHER_BLOWFISH_SW, NULL);
    CRYPTO_TWOFISH_Install  (&CRYPTO_CIPHER_TWOFISH_SW,  NULL);
    //
    // Software hashing.
    //
    CRYPTO_MD5_Install      (&CRYPTO_HASH_MD5_SW,        NULL);
    CRYPTO_SHA1_Install     (&CRYPTO_HASH_SHA1_SW,       NULL);
    CRYPTO_SHA224_Install   (&CRYPTO_HASH_SHA224_SW,     NULL);
    CRYPTO_SHA256_Install   (&CRYPTO_HASH_SHA256_SW,     NULL);
    CRYPTO_SHA512_Install   (&CRYPTO_HASH_SHA512_SW,     NULL);
    CRYPTO_RIPEMD160_Install(&CRYPTO_HASH_RIPEMD160_SW, NULL);
    //
    // Install Hash_DRBG-SHA-256 with TRNG entropy.
    //
    CRYPTO_RNG_InstallEx(&CRYPTO_RNG_DRBG_HASH_SHA256, &CRYPTO_RNG_HW_RT10xx_TRNG);
    //
    // Install small modular exponentiation functions.
    //
    CRYPTO_MPI_SetPublicModExp (CRYPTO_MPI_ModExp_Basic_Fast);
    CRYPTO_MPI_SetPrivateModExp(CRYPTO_MPI_ModExp_Basic_Fast);
}

/*************************** End of file ***************************/
```

7.3.3.4   STM32 CRYP coprocessor (Add-on)

The STM32 cryptographic processor (CRYP) is a capable hardware accelerator presented as a memory-mapped peripheral that accelerates AES and TDES encryption and decryption. There are two variants of the CRYP processor with different capabilities present on the following family members:

- STM32F41x CRYP, hereafter referred to as the *standard CRYP processor*, and
- STM32F43x/F47x CRYP, hereafter referred to as the *enhanced CRYP processor*.

emSSH has support for the following cryptographic algorithms using both CRYP variants:

- DES in ECB and CBC modes.
- TDES in ECB and CBC modes with keying options 1, 2, and 3.
- AES-128, AES-192, and AES-256 in ECB and CBC modes.

For the enhanced CRYP processor, direct acceleration is provided for:

- AES-128, AES-192, and AES-256 in CCM(12,4) and GCM(12,4) modes.

For the standard CRYP processor, acceleration is provided for:

- AES-128, AES-192, and AES-256 ciphering with GCM and CCM in software.

For CCM and GCM modes, the CRYP processor supports only fixed 16-byte authentication tags and 12-byte IVs with 4-byte counters. Therefore, AES-CCM acceleration is not immediately suitable for authenticated encryption in SSH as SSH requires zero-length IVs with 16-byte counters.

7.3.3.4.1   Installing CRYP hardware support

The following interfaces are provided:

```
extern const CRYPTO_CIPHER_API CRYPTO_CIPHER_AES_HW_STM32_CRYP;
extern const CRYPTO_CIPHER_API CRYPTO_CIPHER_TDES_HW_STM32_CRYP;
```

You can install hardware support using:

```
void CRYPTO_X_Config(void) {
  CRYPTO_AES_Install (&CRYPTO_CIPHER_AES_HW_STM32_CRYP);
  CRYPTO_TDES_Install(&CRYPTO_CIPHER_TDES_HW_STM32_CRYP);
}
```

7.3.3.4.2   Enabling the CRYP coprocessor

You must enable clocks and reset the CRYP peripheral before reading or writing its registers. For the STM32F7 device, the following code is sufficient to enable and reset the peripheral:

```
volatile U32 *pReg;
//
pReg = (volatile U32 *)0x40023834;  // RCC_AHB2ENR
*pReg |= 1U << 4;                    // RCC_AHB2ENR.CRYPEN=1
pReg = (volatile U32 *)0x40023814;  // RCC_AHB2RSTR
*pReg |= 1U << 4;                    // RCC_AHB2RSTR.CRYPRST=1
*pReg &= ~(1U << 4);                 // RCC_AHB2RSTR.CRYPRST=0
```

7.3.3.4.3   STM32 cryptographic units

The emSSH implementation of hardware assistance requires one cryptographic unit with index #0 that covers ciphering. See *CRYPTO-OS integration* on page 326 for further details.

7.3.3.4.4 Sample STM32F756 setup

The following is the cryptographic setup for the STMicroelectronics STM32756G-EVAL board:

```c
/**********************************************************************
*                (c) SEGGER Microcontroller GmbH & Co. KG           *
*                     The Embedded Experts                          *
*                        www.segger.com                             *
**********************************************************************

----------------------- END-OF-HEADER -----------------------------

File        : CRYPTO_X_Config_STM32F75x.c
Purpose     : Configure CRYPTO for STM32F4/F7 boards with crypto.

*/

/**********************************************************************
*
*       #include Section
*
**********************************************************************
*/

#include "CRYPTO.h"

/**********************************************************************
*
*       Public code
*
**********************************************************************
*/

/**********************************************************************
*
*       CRYPTO_X_Panic()
*
*  Function description
*    Hang when something unexpected happens.
*/
void CRYPTO_X_Panic(void) {
  for (;;) {
    /* Hang */
  }
}

/**********************************************************************
*
*       CRYPTO_X_Config()
*
*  Function description
*    Configure hardware assist for CRYPTO component.
*/
void CRYPTO_X_Config(void) {
  volatile U32 *pReg;
  //
  // Turn on clocks to the CRYP accelerator and reset it.
  //
  pReg = (volatile U32 *)0x40023834;  // RCC_AHB2ENR
  *pReg |= 1u << 4;                    // RCC_AHB2ENR.CRYPEN=1
  pReg = (volatile U32 *)0x40023814;  // RCC_AHB2RSTR
  *pReg |= 1u << 4;                    // RCC_AHB2RSTR.CRYPRST=1
  *pReg &= ~(1u << 4);                 // RCC_AHB2RSTR.CRYPRST=0
  //
  // Install cipher hardware assistance.
  //
```

```
  CRYPTO_AES_Install      (&CRYPTO_CIPHER_AES_HW_STM32_CRYP,  NULL);
  CRYPTO_TDES_Install     (&CRYPTO_CIPHER_TDES_HW_STM32_CRYP, NULL);
  //
  // Turn on clocks to the HASH accelerator and reset it.
  //
  pReg = (volatile U32 *)0x40023834;  // RCC_AHB2ENR
  *pReg |= 1u << 5;                   // RCC_AHB2ENR.HASHEN=1
  pReg = (volatile U32 *)0x40023814;  // RCC_AHB2RSTR
  *pReg |= 1u << 5;                   // RCC_AHB2RSTR.HASHRST=1
  *pReg &= ~(1u << 5);                // RCC_AHB2RSTR.HASHRST=0
  //
  // Install hardware hashing with software fallback (required).
  //
  CRYPTO_MD5_Install      (&CRYPTO_HASH_MD5_HW_STM32_HASH,
  &CRYPTO_HASH_MD5_SW);
  CRYPTO_SHA1_Install     (&CRYPTO_HASH_SHA1_HW_STM32_HASH,
  &CRYPTO_HASH_SHA1_SW);
  CRYPTO_SHA224_Install
  (&CRYPTO_HASH_SHA224_HW_STM32_HASH, &CRYPTO_HASH_SHA224_SW);
  CRYPTO_SHA256_Install
  (&CRYPTO_HASH_SHA256_HW_STM32_HASH, &CRYPTO_HASH_SHA256_SW);
  //
  // Software hashing.
  //
  CRYPTO_RIPEMD160_Install(&CRYPTO_HASH_RIPEMD160_SW,  NULL);
  CRYPTO_SHA512_Install   (&CRYPTO_HASH_SHA512_SW,     NULL);
  CRYPTO_SEED_Install     (&CRYPTO_CIPHER_SEED_SW,     NULL);
  CRYPTO_ARIA_Install     (&CRYPTO_CIPHER_ARIA_SW,     NULL);
  CRYPTO_CAMELLIA_Install (&CRYPTO_CIPHER_CAMELLIA_SW, NULL);
  //
  // Turn on clocks to the RNG and reset it.
  //
  pReg = (volatile U32 *)0x40023834;  // RCC_AHB2ENR
  *pReg |= 1u << 6;                   // RCC_AHB2ENR.RNGEN=1
  pReg = (volatile U32 *)0x40023814;  // RCC_AHB2RSTR
  *pReg |= 1u << 6;                   // RCC_AHB2RSTR.RNGRST=1
  *pReg &= ~(1u << 6);                // RCC_AHB2RSTR.RNGRST=0
  //
  // Random number generator.
  //
  CRYPTO_RNG_InstallEx    (&CRYPTO_RNG_HW_STM32_RNG, &CRYPTO_RNG_HW_STM32_RNG);
  //
  // Install small modular exponentiation functions.
  //
  CRYPTO_MPI_SetPublicModExp (CRYPTO_MPI_ModExp_Basic_Fast);
  CRYPTO_MPI_SetPrivateModExp(CRYPTO_MPI_ModExp_Basic_Fast);
}

/************************* End of file *************************/
```

7.3.3.5   STM32 AES coprocessor (Add-on)

The STM32 AES hardware accelerator (AES) is a hardware accelerator presented as a memory-mapped peripheral that accelerates AES-128 and AES-256 encryption and decryption. The AES accelerator is present on selected STM32L4 devices.

emSSH has support for the following cryptographic algorithms using the AES hardware accelerator:

* AES-128 and AES-256 in ECB and CBC modes.

7.3.3.5.1   Installing AES hardware support

The following interfaces are provided:

```
extern const CRYPTO_CIPHER_API CRYPTO_CIPHER_AES_HW_STM32_AES;
```

You can install hardware support for AES-128 and AES-192 *only* using:

```
void CRYPTO_X_Config(void) {
   CRYPTO_AES_Install (&CRYPTO_CIPHER_AES_HW_STM32_AES, NULL);
}
```

If you require AES-192 support, you must install a software fallback that is used when ciphering with a 192-bit key:

```
void CRYPTO_X_Config(void) {
   CRYPTO_AES_Install (&CRYPTO_CIPHER_AES_HW_STM32_AES,
                       &CRYPTO_CIPHER_AES_SW);
}
```

7.3.3.5.2   Enabling the AES coprocessor

You must enable clocks and reset the AES peripheral before reading or writing its registers. For the STM32L4A6 device, the following code is sufficient to enable and reset the peripheral:

```
volatile U32 *pReg;
//
pReg = (volatile U32 *)0x4002104C;   // RCC_AHB2ENR
*pReg |= 1U << 4;                     // RCC_AHB2ENR.AESEN=1
pReg = (volatile U32 *)0x4002102C;   // RCC_AHB2RSTR
*pReg |= 1U << 16;                    // RCC_AHB2RSTR.AESRST=1
*pReg &= ~(1U << 16);                 // RCC_AHB2RSTR.AESRST=0
```

7.3.3.5.3   STM32 cryptographic units

The emSSH implementation of hardware assistance requires one cryptographic unit with index #0 that covers ciphering. See *CRYPTO-OS integration* on page 326 for further details.

7.3.3.6   STM32 HASH coprocessor (Add-on)

The STM32 hash coprocessor (HASH) is a hardware accelerator presented as a memory-mapped peripheral that accelerates calculation of MD5, SHA-1, SHA-224 and SHA-256 message digests.

emSSH has HASh accelerator support for the following cryptographic algorithms:

- MD5 message digest.
- SHA-1 message digest.
- SHA-224 and SHA-256 message digest.

7.3.3.6.1   Installing HASH hardware support

The following interfaces are provided:

```
extern const CRYPTO_HASH_API CRYPTO_HASH_MD5_HW_STM32_HASH;
extern const CRYPTO_HASH_API CRYPTO_HASH_SHA1_HW_STM32_HASH;
extern const CRYPTO_HASH_API CRYPTO_HASH_SHA224_HW_STM32_HASH;
extern const CRYPTO_HASH_API CRYPTO_HASH_SHA256_HW_STM32_HASH;
```

You can install hardware support using:

```
void CRYPTO_X_Config(void) {
  CRYPTO_MD5_Install   (&CRYPTO_HASH_MD5_HW_STM32_HASH);
  CRYPTO_SHA1_Install  (&CRYPTO_HASH_SHA1_HW_STM32_HASH);
  CRYPTO_SHA224_Install(&CRYPTO_HASH_SHA224_HW_STM32_HASH);
  CRYPTO_SHA256_Install(&CRYPTO_HASH_SHA256_HW_STM32_HASH);
}
```

7.3.3.6.2   Enabling the HASH coprocessor

You must enable clocks and reset the HASH peripheral before reading or writing its registers.

For the STM32F7 device, the following code is sufficient to enable and reset the peripheral:

```
volatile U32 *pReg;
//
pReg = (volatile U32 *)0x40023834;  // RCC_AHB2ENR
*pReg |= 1u << 5;                    // RCC_AHB2ENR.HASHEN=1
pReg = (volatile U32 *)0x40023814;  // RCC_AHB2RSTR
*pReg |= 1u << 5;                    // RCC_AHB2RSTR.HASHRST=1
*pReg &= ~(1u << 5);                 // RCC_AHB2RSTR.HASHRST=0
```

For the STM32L4 device, the following code is sufficient to enable and reset the peripheral:

```
volatile U32 *RCC_AHB2RSTR = (U32 *)0x4002102C;
volatile U32 *RCC_AHB2ENR  = (U32 *)0x4002104C;
//
*RCC_AHB2ENR   |= 1<<17;
*RCC_AHB2RSTR  |= 1<<17;
*RCC_AHB2RSTR  &= ~(1<<17);
```

7.3.3.6.3   STM32 cryptographic units

The emSSH implementation of hardware assistance requires two cryptographic units with indexes #0 and #1 that cover ciphering (unit #0) and hashing (unit #1). See *CRYPTO-OS integration* on page 326 for further details.

7.3.3.6.4 Sample STM32F756 setup

The following is the cryptographic setup for the STMicroelectronics STM32756G-EVAL board:

```
/**********************************************************************
*                (c) SEGGER Microcontroller GmbH & Co. KG            *
*                      The Embedded Experts                          *
*                         www.segger.com                             *
***********************************************************************

----------------------- END-OF-HEADER -----------------------------

File       : CRYPTO_X_Config_STM32F75x.c
Purpose    : Configure CRYPTO for STM32F4/F7 boards with crypto.

*/

/**********************************************************************
*
*       #include Section
*
***********************************************************************
*/

#include "CRYPTO.h"

/**********************************************************************
*
*       Public code
*
***********************************************************************
*/

/**********************************************************************
*
*       CRYPTO_X_Panic()
*
*  Function description
*    Hang when something unexpected happens.
*/
void CRYPTO_X_Panic(void) {
  for (;;) {
    /* Hang */
  }
}

/**********************************************************************
*
*       CRYPTO_X_Config()
*
*  Function description
*    Configure hardware assist for CRYPTO component.
*/
void CRYPTO_X_Config(void) {
  volatile U32 *pReg;
  //
  // Turn on clocks to the CRYP accelerator and reset it.
  //
  pReg = (volatile U32 *)0x40023834;  // RCC_AHB2ENR
  *pReg |= 1u << 4;                     // RCC_AHB2ENR.CRYPEN=1
  pReg = (volatile U32 *)0x40023814;  // RCC_AHB2RSTR
  *pReg |= 1u << 4;                     // RCC_AHB2RSTR.CRYPRST=1
  *pReg &= ~(1u << 4);                  // RCC_AHB2RSTR.CRYPRST=0
  //
  // Install cipher hardware assistance.
  //
```

```
  CRYPTO_AES_Install      (&CRYPTO_CIPHER_AES_HW_STM32_CRYP,  NULL);
  CRYPTO_TDES_Install     (&CRYPTO_CIPHER_TDES_HW_STM32_CRYP, NULL);
  //
  // Turn on clocks to the HASH accelerator and reset it.
  //
  pReg = (volatile U32 *)0x40023834;  // RCC_AHB2ENR
  *pReg |= 1u << 5;                    // RCC_AHB2ENR.HASHEN=1
  pReg = (volatile U32 *)0x40023814;  // RCC_AHB2RSTR
  *pReg |= 1u << 5;                    // RCC_AHB2RSTR.HASHRST=1
  *pReg &= ~(1u << 5);                 // RCC_AHB2RSTR.HASHRST=0
  //
  // Install hardware hashing with software fallback (required).
  //
  CRYPTO_MD5_Install      (&CRYPTO_HASH_MD5_HW_STM32_HASH,
  &CRYPTO_HASH_MD5_SW);
  CRYPTO_SHA1_Install     (&CRYPTO_HASH_SHA1_HW_STM32_HASH,
  &CRYPTO_HASH_SHA1_SW);
  CRYPTO_SHA224_Install
  (&CRYPTO_HASH_SHA224_HW_STM32_HASH, &CRYPTO_HASH_SHA224_SW);
  CRYPTO_SHA256_Install
  (&CRYPTO_HASH_SHA256_HW_STM32_HASH, &CRYPTO_HASH_SHA256_SW);
  //
  // Software hashing.
  //
  CRYPTO_RIPEMD160_Install(&CRYPTO_HASH_RIPEMD160_SW,  NULL);
  CRYPTO_SHA512_Install   (&CRYPTO_HASH_SHA512_SW,     NULL);
  CRYPTO_SEED_Install     (&CRYPTO_CIPHER_SEED_SW,     NULL);
  CRYPTO_ARIA_Install     (&CRYPTO_CIPHER_ARIA_SW,     NULL);
  CRYPTO_CAMELLIA_Install (&CRYPTO_CIPHER_CAMELLIA_SW, NULL);
  //
  // Turn on clocks to the RNG and reset it.
  //
  pReg = (volatile U32 *)0x40023834;  // RCC_AHB2ENR
  *pReg |= 1u << 6;                    // RCC_AHB2ENR.RNGEN=1
  pReg = (volatile U32 *)0x40023814;  // RCC_AHB2RSTR
  *pReg |= 1u << 6;                    // RCC_AHB2RSTR.RNGRST=1
  *pReg &= ~(1u << 6);                 // RCC_AHB2RSTR.RNGRST=0
  //
  // Random number generator.
  //
  CRYPTO_RNG_InstallEx    (&CRYPTO_RNG_HW_STM32_RNG, &CRYPTO_RNG_HW_STM32_RNG);
  //
  // Install small modular exponentiation functions.
  //
  CRYPTO_MPI_SetPublicModExp (CRYPTO_MPI_ModExp_Basic_Fast);
  CRYPTO_MPI_SetPrivateModExp(CRYPTO_MPI_ModExp_Basic_Fast);
}

/************************** End of file **************************/
```

7.3.3.7   EFM32 CRYPTO coprocessor (Add-on)

The EFM32 cryptographic coprocessor (CRYPTO) is presented as a memory-mapped peripheral.

emSSH has specialized hardware-assisted hashing support for the following cryptographic algorithms using the CRYPTO coprocessor:

- SHA-1

7.3.3.7.1   Installing CRYPTO hardware support

The following hardware-assisted interfaces are available:

```
extern const CRYPTO_HASH_API CRYPTO_HASH_SHA1_HW_EFM32_CRYPTO;
```

You can install hardware support using:

```
void CRYPTO_X_Config(void) {
  CRYPTO_SHA1_Install(&CRYPTO_HASH_SHA1_HW_EFM32_CRYPTO, NULL);
}
```

7.3.3.7.2   EFM32 cryptographic units

The emSSH implementation of hardware assistance requires one cryptographic unit with index #0 covering hashing and RSA operations. See *CRYPTO-OS integration* on page 326 for further details.

If you wish to reduce power consumption, it is possible to enable and clocks to the crypto unit when `CRYPTO_OS_Claim()` is called and disable them `CRYPTO_OS_Unclaim()` is called (for cryptographic unit #0).

7.3.3.7.3   Modular exponentiation API

Function	Description
Windowing, Montgomery reduction	
CRYPTO_MPI_ModExp_Montgomery_2b_FW_EFM32_CRYPTO()	Modular exponentiation, Montgomery reduction, 2-bit window.
CRYPTO_MPI_ModExp_Montgomery_3b_FW_EFM32_CRYPTO()	Modular exponentiation, Montgomery reduction, 3-bit window.
CRYPTO_MPI_ModExp_Montgomery_4b_FW_EFM32_CRYPTO()	Modular exponentiation, Montgomery reduction, 4-bit window.
CRYPTO_MPI_ModExp_Montgomery_5b_FW_EFM32_CRYPTO()	Modular exponentiation, Montgomery reduction, 5-bit window.
CRYPTO_MPI_ModExp_Montgomery_6b_FW_EFM32_CRYPTO()	Modular exponentiation, Montgomery reduction, 6-bit window.
CRYPTO_MPI_ModExp_Montgomery_2b_RM_EFM32_CRYPTO()	Modular exponentiation, Montgomery reduction, 2-bit window.
CRYPTO_MPI_ModExp_Montgomery_3b_RM_EFM32_CRYPTO()	Modular exponentiation, Montgomery reduction, 3-bit window.
CRYPTO_MPI_ModExp_Montgomery_4b_RM_EFM32_CRYPTO()	Modular exponentiation, Montgomery reduction, 4-bit window.
CRYPTO_MPI_ModExp_Montgomery_5b_RM_EFM32_CRYPTO()	Modular exponentiation, Montgomery reduction, 5-bit window.
CRYPTO_MPI_ModExp_Montgomery_6b_RM_EFM32_CRYPTO()	Modular exponentiation, Montgomery reduction, 6-bit window.

7.3.3.7.3.1 CRYPTO_MPI_ModExp_Montgomery_2b_FW_EFM32_CRYPTO()

Description

Modular exponentiation, Montgomery reduction, 2-bit window.

Prototype

```
int CRYPTO_MPI_ModExp_Montgomery_2b_FW_EFM32_CRYPTO
                                        (       CRYPTO_MPI          * pSelf,
                                          const CRYPTO_MPI          * pExponent,
                                          const CRYPTO_MPI          * pModulus,
                                                CRYPTO_MEM_CONTEXT * pMem);
```

Parameters

Parameter	Description
pSelf	Pointer to MPI that contains the base; exponential on return.
pExponent	Pointer to MPI that contains the exponent.
pModulus	Pointer to MPI that contains the modulus.
pMem	Memory allocator to use for temporary data.

Return value

< 0   Processing error
≥ 0   Success

7.3.3.7.3.2   CRYPTO_MPI_ModExp_Montgomery_3b_FW_EFM32_CRYPTO()

Description

Modular exponentiation, Montgomery reduction, 3-bit window.

Prototype

```
int CRYPTO_MPI_ModExp_Montgomery_3b_FW_EFM32_CRYPTO
                                    (       CRYPTO_MPI         * pSelf,
                                      const CRYPTO_MPI         * pExponent,
                                      const CRYPTO_MPI         * pModulus,
                                            CRYPTO_MEM_CONTEXT * pMem);
```

Parameters

Parameter	Description
pSelf	Pointer to MPI that contains the base; exponential on return.
pExponent	Pointer to MPI that contains the exponent.
pModulus	Pointer to MPI that contains the modulus.
pMem	Memory allocator to use for temporary data.

Return value

< 0   Processing error
≥ 0   Success

7.3.3.7.3.3   CRYPTO_MPI_ModExp_Montgomery_4b_FW_EFM32_CRYPTO()

Description

Modular exponentiation, Montgomery reduction, 4-bit window.

Prototype

```
int CRYPTO_MPI_ModExp_Montgomery_4b_FW_EFM32_CRYPTO
                                        (       CRYPTO_MPI          * pSelf,
                                          const CRYPTO_MPI          * pExponent,
                                          const CRYPTO_MPI          * pModulus,
                                                CRYPTO_MEM_CONTEXT * pMem);
```

Parameters

Parameter	Description
pSelf	Pointer to MPI that contains the base; exponential on return.
pExponent	Pointer to MPI that contains the exponent.
pModulus	Pointer to MPI that contains the modulus.
pMem	Memory allocator to use for temporary data.

Return value

< 0   Processing error
≥ 0   Success

7.3.3.7.3.4   CRYPTO_MPI_ModExp_Montgomery_5b_FW_EFM32_CRYPTO()

Description

Modular exponentiation, Montgomery reduction, 5-bit window.

Prototype

```
int CRYPTO_MPI_ModExp_Montgomery_5b_FW_EFM32_CRYPTO
                                     (       CRYPTO_MPI         * pSelf,
                                       const CRYPTO_MPI         * pExponent,
                                       const CRYPTO_MPI         * pModulus,
                                             CRYPTO_MEM_CONTEXT * pMem);
```

Parameters

Parameter	Description
pSelf	Pointer to MPI that contains the base; exponential on return.
pExponent	Pointer to MPI that contains the exponent.
pModulus	Pointer to MPI that contains the modulus.
pMem	Memory allocator to use for temporary data.

Return value

< 0   Processing error
≥ 0   Success

7.3.3.7.3.5   CRYPTO_MPI_ModExp_Montgomery_6b_FW_EFM32_CRYPTO()

Description

Modular exponentiation, Montgomery reduction, 6-bit window.

Prototype

```
int CRYPTO_MPI_ModExp_Montgomery_6b_FW_EFM32_CRYPTO
                                      (       CRYPTO_MPI          * pSelf,
                                        const CRYPTO_MPI          * pExponent,
                                        const CRYPTO_MPI          * pModulus,
                                              CRYPTO_MEM_CONTEXT * pMem);
```

Parameters

Parameter	Description
pSelf	Pointer to MPI that contains the base; exponential on return.
pExponent	Pointer to MPI that contains the exponent.
pModulus	Pointer to MPI that contains the modulus.
pMem	Memory allocator to use for temporary data.

Return value

< 0   Processing error
≥ 0   Success

7.3.3.7.3.6   CRYPTO_MPI_ModExp_Montgomery_2b_RM_EFM32_CRYPTO()

Description

Modular exponentiation, Montgomery reduction, 2-bit window.

Prototype

```
int CRYPTO_MPI_ModExp_Montgomery_2b_RM_EFM32_CRYPTO
                                      (      CRYPTO_MPI         * pSelf,
                                       const CRYPTO_MPI         * pExponent,
                                       const CRYPTO_MPI         * pModulus,
                                             CRYPTO_MEM_CONTEXT * pMem);
```

Parameters

Parameter	Description
pSelf	Pointer to MPI that contains the base; exponential on return.
pExponent	Pointer to MPI that contains the exponent.
pModulus	Pointer to MPI that contains the modulus.
pMem	Memory allocator to use for temporary data.

Return value

< 0   Processing error
≥ 0   Success

7.3.3.7.3.7   CRYPTO_MPI_ModExp_Montgomery_3b_RM_EFM32_CRYPTO()

Description

Modular exponentiation, Montgomery reduction, 3-bit window.

Prototype

```
int CRYPTO_MPI_ModExp_Montgomery_3b_RM_EFM32_CRYPTO
                                          (       CRYPTO_MPI          * pSelf,
                                            const CRYPTO_MPI          * pExponent,
                                            const CRYPTO_MPI          * pModulus,
                                                  CRYPTO_MEM_CONTEXT * pMem);
```

Parameters

Parameter	Description
pSelf	Pointer to MPI that contains the base; exponential on return.
pExponent	Pointer to MPI that contains the exponent.
pModulus	Pointer to MPI that contains the modulus.
pMem	Memory allocator to use for temporary data.

Return value

< 0   Processing error
≥ 0   Success

7.3.3.7.3.8   CRYPTO_MPI_ModExp_Montgomery_4b_RM_EFM32_CRYPTO()

Description

Modular exponentiation, Montgomery reduction, 4-bit window.

Prototype

```
int CRYPTO_MPI_ModExp_Montgomery_4b_RM_EFM32_CRYPTO
                                          (       CRYPTO_MPI          * pSelf,
                                            const CRYPTO_MPI          * pExponent,
                                            const CRYPTO_MPI          * pModulus,
                                                  CRYPTO_MEM_CONTEXT * pMem);
```

Parameters

Parameter	Description
pSelf	Pointer to MPI that contains the base; exponential on return.
pExponent	Pointer to MPI that contains the exponent.
pModulus	Pointer to MPI that contains the modulus.
pMem	Memory allocator to use for temporary data.

Return value

< 0   Processing error
≥ 0   Success

7.3.3.7.3.9 CRYPTO_MPI_ModExp_Montgomery_5b_RM_EFM32_CRYPTO()

Description

Modular exponentiation, Montgomery reduction, 5-bit window.

Prototype

```
int CRYPTO_MPI_ModExp_Montgomery_5b_RM_EFM32_CRYPTO
                                    (       CRYPTO_MPI         * pSelf,
                                      const CRYPTO_MPI         * pExponent,
                                      const CRYPTO_MPI         * pModulus,
                                            CRYPTO_MEM_CONTEXT * pMem);
```

Parameters

Parameter	Description
pSelf	Pointer to MPI that contains the base; exponential on return.
pExponent	Pointer to MPI that contains the exponent.
pModulus	Pointer to MPI that contains the modulus.
pMem	Memory allocator to use for temporary data.

Return value

< 0   Processing error
≥ 0   Success

7.3.3.7.3.10   CRYPTO_MPI_ModExp_Montgomery_6b_RM_EFM32_CRYPTO()

Description

Modular exponentiation, Montgomery reduction, 6-bit window.

Prototype

```
int CRYPTO_MPI_ModExp_Montgomery_6b_RM_EFM32_CRYPTO
                                      (      CRYPTO_MPI          * pSelf,
                                       const CRYPTO_MPI          * pExponent,
                                       const CRYPTO_MPI          * pModulus,
                                             CRYPTO_MEM_CONTEXT * pMem);
```

Parameters

Parameter	Description
pSelf	Pointer to MPI that contains the base; exponential on return.
pExponent	Pointer to MPI that contains the exponent.
pModulus	Pointer to MPI that contains the modulus.
pMem	Memory allocator to use for temporary data.

Return value

< 0   Processing error
≥ 0   Success

7.3.3.7.4 Performance

7.3.3.7.4.1 SHA-1

Output from the benchmark `CRYPTO_Bench_SHA1` is shown below.

```
(c) 2014-2017 SEGGER Microcontroller GmbH & Co. KG    www.segger.com
SHA-1 Benchmark V2.00 compiled May 24 2017 12:06:22


Compiler: clang 4.0.0 (tags/RELEASE_400/final)
System:   Processor speed                   = 19.000 MHz
Config:   CRYPTO_CONFIG_SHA1_OPTIMIZE    = 1
Config:   CRYPTO_CONFIG_SHA1_HW_OPTIMIZE = 1


+-------------+-----------+
| Algorithm   | Hash MB/s |
+-------------+-----------+
| SHA-1       |      0.76 |
| SHA-1 (HW)  |      6.77 |
+-------------+-----------+

Benchmark complete
```

7.3.3.7.5   Sample EFM32 setup

The following is the cryptographic setup for the Silicon Labs Pearl and Jade Gecko devices:

```c
/**********************************************************************
*                (c) SEGGER Microcontroller GmbH & Co. KG          *
*                     The Embedded Experts                         *
*                        www.segger.com                            *
***********************************************************************

----------------------- END-OF-HEADER -----------------------------

File        : CRYPTO_X_Config_EFM32.c
Purpose     : Configure CRYPTO for EFM32 Pearl and Jade Geckos.

*/

/**********************************************************************
*
*       #include Section
*
***********************************************************************
*/

#include "CRYPTO.h"

/**********************************************************************
*
*       Public code
*
***********************************************************************
*/

/**********************************************************************
*
*       CRYPTO_X_Panic()
*
*  Function description
*    Hang when something unexpected happens.
*/
void CRYPTO_X_Panic(void) {
  for (;;) {
    /* Hang */
  }
}

/**********************************************************************
*
*       CRYPTO_X_Config()
*
*  Function description
*    Configure hardware assist for CRYPTO component.
*/
void CRYPTO_X_Config(void) {
  volatile U32 *HFBUSCLKEN0;
  //
  CRYPTO_MD5_Install       (&CRYPTO_HASH_MD5_SW,                 NULL);
  CRYPTO_SHA1_Install      (&CRYPTO_HASH_SHA1_HW_EFM32_CRYPTO,   NULL);
  CRYPTO_SHA224_Install    (&CRYPTO_HASH_SHA224_SW,              NULL);
  CRYPTO_SHA256_Install    (&CRYPTO_HASH_SHA256_SW,              NULL);
  CRYPTO_AES_Install       (&CRYPTO_CIPHER_AES,                  NULL);
  CRYPTO_TDES_Install      (&CRYPTO_CIPHER_TDES,                 NULL);
  CRYPTO_SHA512_Install    (&CRYPTO_HASH_SHA512_SW,             NULL);
  CRYPTO_RIPEMD160_Install (&CRYPTO_HASH_RIPEMD160_SW,          NULL);
  //
  // Clock CRYPTO peripheral.
  //
```

```
  HFBUSCLKEN0 = (void *)0x400E40B0UL;
  *HFBUSCLKEN0 |= 1UL << 1;  // Turn on clock to CRYPTO unit
}

/************************** End of file **************************/
```

7.3.4   Secure random numbers

In order to guarantee the privacy of communications, it is vital that emSSH can call upon a stream of random numbers. Many microcontrollers that have Ethernet peripherals also provide cryptographic accelerators and true random number generators (RNGs), but not all do.

The sample implementation shipped for Windows use Microsoft's cryptographically-secure random number API to gather randomness, which satisfies emSSH's requirements.

For embedded targets, it isn't necessary to provide a fast random number generator, the hardware RNG will fit perfectly — connection time is dominated by public key operations, not tens of bytes of (relatively) slowly-gathered random data.

For devices that have no true random source, it suffices to gather a few hundred bits of random data from jitter in some physical timer or readings of the low order bits of some ADC, and feed that to a software random bit generator.

7.3.4.1   Installing random sources

The function `CRYPTO_RNG_Install` installs a source of randomness that the emCrypt component can use. The source of randomness can be either hardware or software, and the quality of that randomness is important.

7.3.4.1.1   Hardware-only random sources

emCrypt has add-on drivers for the Kinetis RNGA and STM32 RNG peripherals. You can install the hardware source as both the random bit generator and the source of entropy, for example:

```
CRYPTO_RNG_Install(&CRYPTO_RNG_HW_Kinetis_RNGA);
```

This only provides secure random data if the hardware produces secure random data: the STM32 RNG and Kinetis RNGA have provisos that the random data they produce are potentially not secure.

7.3.4.1.2   Secure random bit generator with hardware entropy

emCrypt supports using hardware sources of entropy to seed a deterministic random bit generator, and emCrypt fully implements the NIST DRBG random bit generators.

`CRYPTO_RNG_InstallEx` installs both a source of entropy and the random bit generator that it seeds: emCrypt will use the random bit generator to acquire random data and the entropy source feeds the random bit generator.

The DRBGs implemented are:

```
extern const CRYPTO_RNG_API CRYPTO_RNG_DRBG_HASH_SHA1;
extern const CRYPTO_RNG_API CRYPTO_RNG_DRBG_HASH_SHA224;
extern const CRYPTO_RNG_API CRYPTO_RNG_DRBG_HASH_SHA256;
extern const CRYPTO_RNG_API CRYPTO_RNG_DRBG_HASH_SHA384;
extern const CRYPTO_RNG_API CRYPTO_RNG_DRBG_HASH_SHA512;
extern const CRYPTO_RNG_API CRYPTO_RNG_DRBG_HASH_SHA512_224;
extern const CRYPTO_RNG_API CRYPTO_RNG_DRBG_HASH_SHA512_256;
extern const CRYPTO_RNG_API CRYPTO_RNG_DRBG_HMAC_SHA1;
extern const CRYPTO_RNG_API CRYPTO_RNG_DRBG_HMAC_SHA224;
extern const CRYPTO_RNG_API CRYPTO_RNG_DRBG_HMAC_SHA256;
extern const CRYPTO_RNG_API CRYPTO_RNG_DRBG_HMAC_SHA384;
extern const CRYPTO_RNG_API CRYPTO_RNG_DRBG_HMAC_SHA512;
extern const CRYPTO_RNG_API CRYPTO_RNG_DRBG_HMAC_SHA512_224;
extern const CRYPTO_RNG_API CRYPTO_RNG_DRBG_HMAC_SHA512_256;
extern const CRYPTO_RNG_API CRYPTO_RNG_DRBG_CTR_TDES;
extern const CRYPTO_RNG_API CRYPTO_RNG_DRBG_CTR_AES128;
extern const CRYPTO_RNG_API CRYPTO_RNG_DRBG_CTR_AES192;
extern const CRYPTO_RNG_API CRYPTO_RNG_DRBG_CTR_AES256;
```

The following installs both the DRBG and a source of entropy:

```
CRYPTO_RNG_InstallEx(&CRYPTO_RNG_DRBG_HASH_SHA256,
                     &CRYPTO_RNG_HW_Kinetis_RNGA);
```

7.4   Example configurations

7.4.1   Minimal Cortex-M configuration

The following is a configuration that installs cryptographic support without hardware acceleration for Cortex-M devices:

```c
/**********************************************************************
*               (c) SEGGER Microcontroller GmbH & Co. KG             *
*                       The Embedded Experts                         *
*                          www.segger.com                            *
**********************************************************************

----------------------- END-OF-HEADER -----------------------------

File        : CRYPTO_X_Config_SSH_CM.c
Purpose     : Configure CRYPTO for full SSH with no hardware
              accelerators and a dummy, insecure, random number
              generator.

Additional information:
  The dummy random number generator does not generate secure random
  numbers, but can be run on any hardware with memory at 0x20000000.
  To provide secure random numbers modify it according to the hardware
  capabilities.

  Random number generators for different hardware is available from
  SEGGER upon request.
*/

/**********************************************************************
*
*       #include Section
*
**********************************************************************
*/

#include "CRYPTO.h"

/**********************************************************************
*
*       Local functions
*
**********************************************************************
*/

/**********************************************************************
*
*       _RNG_Get()
*
*  Function description
*    Get random data from RNG.
*
*  Parameters
*    pData   - Pointer to the object that receives the random data.
*    DataLen - Octet length of the random data.
*/
static void _RNG_Get(U8 *pData, unsigned DataLen) {
  if (pData && DataLen) {
    while (DataLen--) {
      *pData++ = *((volatile U8*)0x20000000 + DataLen);
    }
  }
}
```

```c
/*********************************************************************
*
*       Public code
*
**********************************************************************
*/

/*********************************************************************
*
*       CRYPTO_X_Panic()
*
*  Function description
*    Hang when something unexpected happens.
*/
void CRYPTO_X_Panic(void) {
  for (;;) {
    /* Hang */
  }
}

/*********************************************************************
*
*       CRYPTO_X_Config()
*
*  Function description
*    Configure no hardware assist for CRYPTO component.
*/
void CRYPTO_X_Config(void) {
  static const CRYPTO_RNG_API _RNG = {
    NULL,
    _RNG_Get,
    NULL,
    NULL
  };
  //
  // Install pure software implementations.
  //
  CRYPTO_MD5_Install       (&CRYPTO_HASH_MD5_SW,        NULL);
  CRYPTO_RIPEMD160_Install(&CRYPTO_HASH_RIPEMD160_SW,   NULL);
  CRYPTO_SHA1_Install      (&CRYPTO_HASH_SHA1_SW,       NULL);
  CRYPTO_SHA224_Install    (&CRYPTO_HASH_SHA224_SW,     NULL);
  CRYPTO_SHA256_Install    (&CRYPTO_HASH_SHA256_SW,     NULL);
  CRYPTO_SHA512_Install    (&CRYPTO_HASH_SHA512_SW,     NULL);
  CRYPTO_AES_Install       (&CRYPTO_CIPHER_AES_SW,      NULL);
  CRYPTO_TDES_Install      (&CRYPTO_CIPHER_TDES_SW,     NULL);
  CRYPTO_CAST_Install      (&CRYPTO_CIPHER_CAST_SW,     NULL);
  CRYPTO_BLOWFISH_Install  (&CRYPTO_CIPHER_BLOWFISH_SW, NULL);
  CRYPTO_TWOFISH_Install   (&CRYPTO_CIPHER_TWOFISH_SW,  NULL);
  CRYPTO_CAMELLIA_Install  (&CRYPTO_CIPHER_CAMELLIA_SW, NULL);
  //
  // Install RNG using Hash_DRBG-SHA256 with "random" data from
  // RAM.
  //
  CRYPTO_RNG_InstallEx(&CRYPTO_RNG_DRBG_HASH_SHA256, &_RNG);
  //
  // Install small modular exponentiation functions.
  //
  CRYPTO_MPI_SetPublicModExp (CRYPTO_MPI_ModExp_Basic_Fast);
  CRYPTO_MPI_SetPrivateModExp(CRYPTO_MPI_ModExp_Basic_Fast);
}

/************************** End of file **************************/
```

7.4.2   Windows configuration

The following is a configuration that installs cryptographic support without hardware acceleration for Windows:

```c
/**********************************************************************
*                   (c) SEGGER Microcontroller GmbH                  *
*                       The Embedded Experts                         *
*                           www.segger.com                           *
**********************************************************************

----------------------- END-OF-HEADER -----------------------------

File        : CRYPTO_X_Config_Full_Win32.c
Purpose     : Configure full cryptography for x86 Win32.

*/

/**********************************************************************
*
*       #include Section
*
**********************************************************************
*/

#define _CRT_RAND_S  /*emDoc ignore*/
#include <stdlib.h>
#include <stdio.h>
#include "CRYPTO.h"

/**********************************************************************
*
*       Static code
*
**********************************************************************
*/


/**********************************************************************
*
*       _RNG_Get()
*
*  Function description
*    Get entropy from Win32 secure random API.
*
*  Parameters
*    pData   - Pointer to object that receives the random bitstream.
*    DataLen - Octet length of the object.
*/
static void _RNG_Get(U8 *pData, unsigned DataLen) {
  unsigned V;
  unsigned L;
  //
  // Use Windows cryptographically-secure random number source.
  //
  while (DataLen > 0) {
#if _MSC_VER <= 1200  // VC6 or earlier.
    V = (unsigned)rand();
#else
    (void)rand_s(&V);
#endif
    for (L = SEGGER_MIN(DataLen, 4); L > 0; --L) {
      *pData = V & 0xFF;
      V >>= 8;
     ++pData;
     --DataLen;
    }
  }
```

```
}
/**********************************************************************
*
*       Public code
*
**********************************************************************
*/

/**********************************************************************
*
*       CRYPTO_X_Panic()
*
*  Function description
*    Hang when something unexpected happens.
*/
void CRYPTO_X_Panic(void) {
  fprintf(stderr, "CRYPTO: panic, system halted.\n");
  exit(100);
}

/**********************************************************************
*
*       CRYPTO_X_FastMultiplier()
*
*  Function description
*    Look up specialist fast-multiplier function.
*/
CRYPTO_MPI_COMBA_MUL_FUNC CRYPTO_X_FastMultiplier(unsigned Bits) {
  switch (Bits / CRYPTO_MPI_BITS_PER_LIMB) {
  case  112 / CRYPTO_MPI_BITS_PER_LIMB: return CRYPTO_MPI_Mul_Comba_112;
  case  128 / CRYPTO_MPI_BITS_PER_LIMB: return CRYPTO_MPI_Mul_Comba_128;
  case  160 / CRYPTO_MPI_BITS_PER_LIMB: return CRYPTO_MPI_Mul_Comba_160;
  case  192 / CRYPTO_MPI_BITS_PER_LIMB: return CRYPTO_MPI_Mul_Comba_192;
  case  224 / CRYPTO_MPI_BITS_PER_LIMB: return CRYPTO_MPI_Mul_Comba_224;
  case  256 / CRYPTO_MPI_BITS_PER_LIMB: return CRYPTO_MPI_Mul_Comba_256;
  case  384 / CRYPTO_MPI_BITS_PER_LIMB: return CRYPTO_MPI_Mul_Comba_384;
  case  448 / CRYPTO_MPI_BITS_PER_LIMB: return CRYPTO_MPI_Mul_Comba_448;
  case  512 / CRYPTO_MPI_BITS_PER_LIMB: return CRYPTO_MPI_Mul_Comba_512;
  case  544 / CRYPTO_MPI_BITS_PER_LIMB: return CRYPTO_MPI_Mul_Comba_544;
  case  768 / CRYPTO_MPI_BITS_PER_LIMB: return CRYPTO_MPI_Mul_Comba_768;
  case 1024 / CRYPTO_MPI_BITS_PER_LIMB: return CRYPTO_MPI_Mul_Comba_1024;
  case 2048 / CRYPTO_MPI_BITS_PER_LIMB: return CRYPTO_MPI_Mul_Comba_2048;
  case 3072 / CRYPTO_MPI_BITS_PER_LIMB: return CRYPTO_MPI_Mul_Comba_3072;
  default: return 0;
  }
}

/**********************************************************************
*
*       CRYPTO_X_Config()
*
*  Function description
*    Configure hardware assist for CRYPTO component.
*/
void CRYPTO_X_Config(void) {
  static const CRYPTO_RNG_API _RNG_Win32 = {
    NULL,
    _RNG_Get,
    NULL,
    NULL
  };
  //
  // Install pure software implementations.
  //
  CRYPTO_MD5_Install      (&CRYPTO_HASH_MD5_SW,       NULL);
  CRYPTO_RIPEMD160_Install(&CRYPTO_HASH_RIPEMD160_SW, NULL);
  CRYPTO_SHA1_Install     (&CRYPTO_HASH_SHA1_SW,      NULL);
```

```
  CRYPTO_SHA224_Install   (&CRYPTO_HASH_SHA224_SW,      NULL);
  CRYPTO_SHA256_Install   (&CRYPTO_HASH_SHA256_SW,      NULL);
  CRYPTO_SHA512_Install   (&CRYPTO_HASH_SHA512_SW,      NULL);
  CRYPTO_SHA3_224_Install (&CRYPTO_HASH_SHA3_224_SW,    NULL);
  CRYPTO_SHA3_256_Install (&CRYPTO_HASH_SHA3_256_SW,    NULL);
  CRYPTO_SHA3_384_Install (&CRYPTO_HASH_SHA3_384_SW,    NULL);
  CRYPTO_SHA3_512_Install (&CRYPTO_HASH_SHA3_512_SW,    NULL);
  CRYPTO_SM3_Install      (&CRYPTO_HASH_SM3_SW,         NULL);
  CRYPTO_BLAKE2S_Install  (&CRYPTO_HASH_BLAKE2S_SW,     NULL);
  CRYPTO_BLAKE2B_Install  (&CRYPTO_HASH_BLAKE2B_SW,     NULL);
  CRYPTO_AES_Install      (&CRYPTO_CIPHER_AES_SW,       NULL);
  CRYPTO_TDES_Install     (&CRYPTO_CIPHER_TDES_SW,      NULL);
  CRYPTO_CAST_Install     (&CRYPTO_CIPHER_CAST_SW,      NULL);
  CRYPTO_ARIA_Install     (&CRYPTO_CIPHER_ARIA_SW,      NULL);
  CRYPTO_SEED_Install     (&CRYPTO_CIPHER_SEED_SW,      NULL);
  CRYPTO_IDEA_Install     (&CRYPTO_CIPHER_IDEA_SW,      NULL);
  CRYPTO_CAMELLIA_Install (&CRYPTO_CIPHER_CAMELLIA_SW, NULL);
  CRYPTO_BLOWFISH_Install (&CRYPTO_CIPHER_BLOWFISH_SW, NULL);
  CRYPTO_TWOFISH_Install  (&CRYPTO_CIPHER_TWOFISH_SW,  NULL);
  //
  // Install RNG using Hash_DRBG-SHA256 using Win32 s_rand()
  // as an entropy source.
  //
  CRYPTO_RNG_InstallEx(&CRYPTO_RNG_DRBG_HASH_SHA256, &_RNG_Win32);
  //
  // Install small modular exponentiation functions.
  //
  CRYPTO_MPI_SetPublicModExp (CRYPTO_MPI_ModExp_Basic_Fast);
  CRYPTO_MPI_SetPrivateModExp(CRYPTO_MPI_ModExp_Basic_Fast);
}

/*************************** End of file ***************************/
```

7.4.3   Linux configuration

The following is a configuration that installs cryptographic support without hardware acceleration for Linux:

```c
/**********************************************************************
*                (c) SEGGER Microcontroller GmbH & Co. KG            *
*                        The Embedded Experts                        *
*                           www.segger.com                           *
**********************************************************************

------------------------- END-OF-HEADER -----------------------------

File        : CRYPTO_X_Config_Full_Linux.c
Purpose     : Configure full cryptography for Linux.

*/

/**********************************************************************
*
*       #include Section
*
**********************************************************************
*/

#include "CRYPTO.h"
#include <stdlib.h>
#include <stdio.h>

/**********************************************************************
*
*       Static code
*
**********************************************************************
*/

/**********************************************************************
*
*       _RNG_Get()
*
*  Function description
*    Get entropy from /dev/urandom.
*
*  Parameters
*    pData   - Pointer to object that receives the random bitstream.
*    DataLen - Octet length of the object.
*/
static void _RNG_Get(U8 *pData, unsigned DataLen) {
  static FILE *pFile;
  //
  pFile = fopen("/dev/urandom", "rb");
  if (pFile == NULL) {
    CRYPTO_X_Panic();
  }
  fread(pData, 1, DataLen, pFile);
  fclose(pFile);
}

/**********************************************************************
*
*       Public code
*
**********************************************************************
*/

/**********************************************************************
*
```

```
*        CRYPTO_X_Panic()
*
*  Function description
*     Hang when something unexpected happens.
*/
void CRYPTO_X_Panic(void) {
  fprintf(stderr, "CRYPTO: panic, system halted.\n");
  exit(100);
}

/**********************************************************************
*
*        CRYPTO_X_Config()
*
*  Function description
*     Configure hardware assist for CRYPTO component.
*/
void CRYPTO_X_Config(void) {
  static const CRYPTO_RNG_API _RNG_Linux = {
    NULL,
    _RNG_Get,
    NULL,
    NULL
  };
  //
  // Install pure software implementations.
  //
  CRYPTO_MD5_Install      (&CRYPTO_HASH_MD5_SW,        NULL);
  CRYPTO_RIPEMD160_Install(&CRYPTO_HASH_RIPEMD160_SW,  NULL);
  CRYPTO_SHA1_Install     (&CRYPTO_HASH_SHA1_SW,       NULL);
  CRYPTO_SHA224_Install   (&CRYPTO_HASH_SHA224_SW,     NULL);
  CRYPTO_SHA256_Install   (&CRYPTO_HASH_SHA256_SW,     NULL);
  CRYPTO_SHA512_Install   (&CRYPTO_HASH_SHA512_SW,     NULL);
  CRYPTO_SHA3_224_Install (&CRYPTO_HASH_SHA3_224_SW,   NULL);
  CRYPTO_SHA3_256_Install (&CRYPTO_HASH_SHA3_256_SW,   NULL);
  CRYPTO_SHA3_384_Install (&CRYPTO_HASH_SHA3_384_SW,   NULL);
  CRYPTO_SHA3_512_Install (&CRYPTO_HASH_SHA3_512_SW,   NULL);
  CRYPTO_AES_Install      (&CRYPTO_CIPHER_AES_SW,      NULL);
  CRYPTO_TDES_Install     (&CRYPTO_CIPHER_TDES_SW,     NULL);
  CRYPTO_CAST_Install     (&CRYPTO_CIPHER_CAST_SW,     NULL);
  CRYPTO_ARIA_Install     (&CRYPTO_CIPHER_ARIA_SW,     NULL);
  CRYPTO_SEED_Install     (&CRYPTO_CIPHER_SEED_SW,     NULL);
  CRYPTO_CAMELLIA_Install (&CRYPTO_CIPHER_CAMELLIA_SW, NULL);
  CRYPTO_BLOWFISH_Install (&CRYPTO_CIPHER_BLOWFISH_SW, NULL);
  CRYPTO_TWOFISH_Install  (&CRYPTO_CIPHER_TWOFISH_SW,  NULL);
  //
  // Install RNG using Hash_DRBG-SHA256 with /dev/urandom as an entropy source.
  //
  CRYPTO_RNG_InstallEx    (&CRYPTO_RNG_DRBG_HASH_SHA256, &_RNG_Linux);
  //
  // Install small modular exponentiation functions.
  //
  CRYPTO_MPI_SetPublicModExp (CRYPTO_MPI_ModExp_Basic_Fast);
  CRYPTO_MPI_SetPrivateModExp(CRYPTO_MPI_ModExp_Basic_Fast);
}

/************************* End of file **************************/
```

7.5   emCrypt API reference

The following sections are extracted from the full emCrypt documentation for reference.

7.5.1   API functions

Function	Description
Hashes	
CRYPTO_MD5_Install()	Install MD5 hash implementation.
CRYPTO_RIPEMD160_Install()	Install RIPEMD-160 hash implementation.
CRYPTO_SHA1_Install()	Install SHA-1 hash implementation.
CRYPTO_SHA256_Install()	Install SHA-256 hash implementation.
CRYPTO_SHA512_Install()	Install SHA-512 hash implementation.
Ciphers	
CRYPTO_AES_Install()	Install cipher.
CRYPTO_TDES_Install()	Install cipher.
CRYPTO_CAST_Install()	Install cipher.
CRYPTO_SEED_Install()	Install cipher.
CRYPTO_ARIA_Install()	Install cipher.
CRYPTO_CAMELLIA_Install()	Install cipher.
CRYPTO_BLOWFISH_Install()	Install cipher.
CRYPTO_TWOFISH_Install()	Install cipher.

7.5.1.1   CRYPTO_MD5_Install()

Description

Install MD5 hash implementation.

Prototype

```
void CRYPTO_MD5_Install(const CRYPTO_HASH_API * pHWAPI,
                        const CRYPTO_HASH_API * pSWAPI);
```

Parameters

Parameter	Description
pHWAPI	Pointer to API to use as the preferred implementation.
pSWAPI	Pointer to API to use as the fallback implementation.

7.5.1.2   CRYPTO_RIPEMD160_Install()

Description

Install RIPEMD-160 hash implementation.

Prototype

```
void CRYPTO_RIPEMD160_Install(const CRYPTO_HASH_API * pHWAPI,
                              const CRYPTO_HASH_API * pSWAPI);
```

Parameters

Parameter	Description
pHWAPI	Pointer to API to use as the preferred implementation.
pSWAPI	Pointer to API to use as the fallback implementation.

7.5.1.3   CRYPTO_SHA1_Install()

Description

Install SHA-1 hash implementation.

Prototype

```
void CRYPTO_SHA1_Install(const CRYPTO_HASH_API * pHWAPI,
                         const CRYPTO_HASH_API * pSWAPI);
```

Parameters

Parameter	Description
pHWAPI	Pointer to API to use as the preferred implementation.
pSWAPI	Pointer to API to use as the fallback implementation.

7.5.1.4   CRYPTO_SHA256_Install()

Description

Install SHA-256 hash implementation.

Prototype

```
void CRYPTO_SHA256_Install(const CRYPTO_HASH_API * pHWAPI,
                           const CRYPTO_HASH_API * pSWAPI);
```

Parameters

Parameter	Description
pHWAPI	Pointer to API to use as the preferred implementation.
pSWAPI	Pointer to API to use as the fallback implementation.

7.5.1.5   CRYPTO_SHA512_Install()

Description

Install SHA-512 hash implementation.

Prototype

```
void CRYPTO_SHA512_Install(const CRYPTO_HASH_API * pHWAPI,
                           const CRYPTO_HASH_API * pSWAPI);
```

Parameters

Parameter	Description
pHWAPI	Pointer to API to use as the preferred implementation.
pSWAPI	Pointer to API to use as the fallback implementation.

7.5.1.6   CRYPTO_AES_Install()

Description

Install cipher.

Prototype

```
void CRYPTO_AES_Install(const CRYPTO_CIPHER_API * pHWAPI,
                        const CRYPTO_CIPHER_API * pSWAPI);
```

Parameters

Parameter	Description
pHWAPI	Pointer to API to use as the preferred implementation.
pSWAPI	Pointer to API to use as the fallback implementation.

7.5.1.7   CRYPTO_TDES_Install()

Description

Install cipher.

Prototype

```
void CRYPTO_TDES_Install(const CRYPTO_CIPHER_API * pHWAPI,
                         const CRYPTO_CIPHER_API * pSWAPI);
```

Parameters

Parameter	Description
pHWAPI	Pointer to API to use as the preferred implementation.
pSWAPI	Pointer to API to use as the fallback implementation.

7.5.1.8   CRYPTO_CAST_Install()

Description

Install cipher.

Prototype

```
void CRYPTO_CAST_Install(const CRYPTO_CIPHER_API * pHWAPI,
                         const CRYPTO_CIPHER_API * pSWAPI);
```

Parameters

Parameter	Description
pHWAPI	Pointer to API to use as the preferred implementation.
pSWAPI	Pointer to API to use as the fallback implementation.

7.5.1.9   CRYPTO_SEED_Install()

Description

Install cipher.

Prototype

```
void CRYPTO_SEED_Install(const CRYPTO_CIPHER_API * pHWAPI,
                         const CRYPTO_CIPHER_API * pSWAPI);
```

Parameters

Parameter	Description
pHWAPI	Pointer to API to use as the preferred implementation.
pSWAPI	Pointer to API to use as the fallback implementation.

7.5.1.10   CRYPTO_ARIA_Install()

Description

Install cipher.

Prototype

```
void CRYPTO_ARIA_Install(const CRYPTO_CIPHER_API * pHWAPI,
                         const CRYPTO_CIPHER_API * pSWAPI);
```

Parameters

Parameter	Description
pHWAPI	Pointer to API to use as the preferred implementation.
pSWAPI	Pointer to API to use as the fallback implementation.

7.5.1.11   CRYPTO_CAMELLIA_Install()

Description

Install cipher.

Prototype

```
void CRYPTO_CAMELLIA_Install(const CRYPTO_CIPHER_API * pHWAPI,
                             const CRYPTO_CIPHER_API * pSWAPI);
```

Parameters

Parameter	Description
pHWAPI	Pointer to API to use as the preferred implementation.
pSWAPI	Pointer to API to use as the fallback implementation.

7.5.1.12 CRYPTO_BLOWFISH_Install()

Description

Install cipher.

Prototype

```
void CRYPTO_BLOWFISH_Install(const CRYPTO_CIPHER_API * pHWAPI,
                             const CRYPTO_CIPHER_API * pSWAPI);
```

Parameters

Parameter	Description
pHWAPI	Pointer to API to use as the preferred implementation.
pSWAPI	Pointer to API to use as the fallback implementation.

7.5.1.13   CRYPTO_TWOFISH_Install()

Description

Install cipher.

Prototype

```
void CRYPTO_TWOFISH_Install(const CRYPTO_CIPHER_API * pHWAPI,
                            const CRYPTO_CIPHER_API * pSWAPI);
```

Parameters

Parameter	Description
pHWAPI	Pointer to API to use as the preferred implementation.
pSWAPI	Pointer to API to use as the fallback implementation.

Chapter 8

OS Integration

8.1   SSH-OS integration

emSSH can be configured for use in a multitasking environment. The interface to the operating system is encapsulated in a single file and a number of standard integrations exist.

This section provides descriptions of the functions required to fully support emSSH in multitasking environments.

8.1.1   SSH-OS API

Function	Description
SSH_OS_Init()	Initialize SSH-OS interface.
SSH_OS_Lock()	Lock emSSH.
SSH_OS_Unlock()	Unlock emSSH.
SSH_OS_DisableInterrupt()	Disables interrupts.
SSH_OS_EnableInterrupt()	Enables interrupts.
SSH_OS_GetTime32()	Return the current system time in ms.
SSH_OS_GetTaskName()	Retrieves the task name.

8.1.1.1   SSH_OS_Init()

Description

Initialize SSH-OS interface.

Prototype

```
void SSH_OS_Init(void);
```

Additional information

Creates and initializes all objects required for task synchronization.

8.1.1.2   SSH_OS_Lock()

Description

Lock emSSH.

Prototype

```
void SSH_OS_Lock(void);
```

Additional information

emSSH requires a single lock, typically a resource semaphore or mutex. This function locks this object, guarding sections of emSSH against other threads.

It is required that the lock is "recursive" or "counts" and can be locked and unlocked several times by the same calling task.

8.1.1.3 SSH_OS_Unlock()

Description

Unlock emSSH.

Prototype

```
void SSH_OS_Unlock(void);
```

Additional information

This function is paired with `SSH_OS_Lock` to unlock the resource semaphore or mutex previously locked by `SSH_OS_Lock`.

8.1.1.4   SSH_OS_DisableInterrupt()

Description

Disables interrupts.

Prototype

```c
void SSH_OS_DisableInterrupt(void);
```

Additional information

It is required that the implementation maintains a count of the number of times the interrupt has been disabled. Only when all interrupt disables have been matched with corresponding enables will interrupts be serviced.

8.1.1.5   SSH_OS_EnableInterrupt()

Description

Enables interrupts.

Prototype

```
void SSH_OS_EnableInterrupt(void);
```

Additional information

This function is paired with `SSH_OS_DisableInterrupt` to enable interrupts previously disabled by `SSH_OS_DisableInterrupt`.

8.1.1.6 SSH_OS_GetTime32()

Description

Return the current system time in ms.

Prototype

```
U32 SSH_OS_GetTime32(void);
```

Return value

System time in ms.

8.1.1.7   SSH_OS_GetTaskName()

Description

Retrieves the task name.

Prototype

```
char *SSH_OS_GetTaskName(void * pTask);
```

Parameters

Parameter	Description
pTask	Pointer to a task identifier such as a task control block.

Return value

Pointer to zero-terminated string containing task name.

8.1.2   SSH-OS binding for embOS

The following is a sample binding for SEGGER embOS, `SSH_OS_embOS.c`:

```c
/**********************************************************************
*               (c) SEGGER Microcontroller GmbH & Co. KG            *
*                       The Embedded Experts                        *
*                          www.segger.com                           *
**********************************************************************

----------------------- END-OF-HEADER ----------------------------

File    : SSH_OS_embOS.c
Purpose : Kernel abstraction for embOS

*/

/**********************************************************************
*
*       #include Section
*
**********************************************************************
*/

#include "SSH.h"
#include "RTOS.h"

/**********************************************************************
*
*       Static data
*
**********************************************************************
*/

static U8 _IsInited;

/**********************************************************************
*
*       Public data
*
**********************************************************************
*/

OS_RSEMA SSH_OS_RSema;     // Public only to allow inlining
 (direct call from SSH core)

/**********************************************************************
*
*       Public code
*
**********************************************************************
*/

/**********************************************************************
*
*       SSH_OS_Init()
*
*  Function description
*    Initialize SSH-OS interface.
*
*  Additional information
*    Creates and initializes all objects required for task
*    synchronization.
*/
void SSH_OS_Init(void) {
  if (_IsInited == 0) {
    OS_CREATERSEMA(&SSH_OS_RSema);
```

```
    _IsInited = 1;
  }
}

/**********************************************************************
*
*       SSH_OS_DisableInterrupt()
*
*  Function description
*    Disables interrupts.
*
*  Additional information
*    It is required that the implementation maintains a count of the
*    number of times the interrupt has been disabled.  Only when
*    all interrupt disables have been matched with corresponding
*    enables will interrupts be serviced.
*/
void SSH_OS_DisableInterrupt(void) {
  OS_IncDI();
}

/**********************************************************************
*
*       SSH_OS_EnableInterrupt()
*
*  Function description
*    Enables interrupts.
*
*  Additional information
*    This function is paired with SSH_OS_DisableInterrupt to enable
*    interrupts previously disabled by SSH_OS_DisableInterrupt.
*/
void SSH_OS_EnableInterrupt(void) {
  OS_DecRI();
}

/**********************************************************************
*
*       SSH_OS_Lock
*
*  Function description
*    Lock emSSH.
*
*  Additional information
*    emSSH requires a single lock, typically a resource semaphore or
*    mutex. This function locks this object, guarding sections of
*    emSSH against other threads.
*
*    It is required that the lock is "recursive" or "counts" and
*    can be locked and unlocked several times by the same calling task.
*/
void SSH_OS_Lock(void) {
  OS_Use(&SSH_OS_RSema);
}

/**********************************************************************
*
*       SSH_OS_Unlock
*
*  Function description
*    Unlock emSSH.
*
*  Additional information
*    This function is paired with SSH_OS_Lock to unlock the resource
*    semaphore or mutex previously locked by SSH_OS_Lock.
*/
void SSH_OS_Unlock(void) {
  OS_Unuse(&SSH_OS_RSema);
```

```
}
/*********************************************************************
*
*       SSH_OS_GetTime32()
*
*  Function description
*    Return the current system time in ms.
*
*  Return value
*    System time in ms.
*/
U32 SSH_OS_GetTime32(void) {
  return OS_GetTime32();
}

/*********************************************************************
*
*       SSH_OS_GetTaskName()
*
*  Function description
*    Retrieves the task name.
*
*  Parameters
*    pTask - Pointer to a task identifier such as a task control block.
*
*  Return value
*    Pointer to zero-terminated string containing task name.
*/
const char * SSH_OS_GetTaskName(void *pTask) {
  return OS_GetTaskName((OS_TASK*)pTask);
}

/************************** End of file **************************/
```

8.1.3   SSH-OS binding for bare metal

The following is a sample binding for a bare metal system that has no tasking,
`SSH_OS_None.c`:

```
/**********************************************************************
*              SEGGER MICROCONTROLLER GmbH & Co. KG                   *
*        Solutions for real time microcontroller applications        *
**********************************************************************
*                                                                    *
*        (c) 2003-2014     SEGGER Microcontroller GmbH & Co KG        *
*                                                                    *
*         Internet: www.segger.com   Support:  support@segger.com    *
*                                                                    *
**********************************************************************
*                                                                    *
*        SSH library                                                 *
*                                                                    *
**********************************************************************

----------------------------------------------------------------------
File    : SSH_OS_None.c
Purpose : Kernel abstraction for usage of emSSH without any RTOS.
--------   END-OF-HEADER   --------------------------------------------
*/

#include "SSH_Int.h"

/**********************************************************************
*
*       Configuration
*
**********************************************************************
*/

/**********************************************************************
*
*       SSH_OS_Init()
*
* Function description
*   Initialize all required variables.
*/
void SSH_OS_Init(void) {
}

/**********************************************************************
*
*       SSH_OS_DisableInterrupt
*/
void SSH_OS_DisableInterrupt(void) {
}

/**********************************************************************
*
*       SSH_OS_EnableInterrupt
*/
void SSH_OS_EnableInterrupt(void) {
}

/**********************************************************************
*
*       SSH_OS_Lock()
*
* Function description
*   The stack requires a single lock, typically a resource semaphore
*   or mutex. This function locks this object, guarding sections of
*   the stack code against other threads.
*   If the entire stack executes from a single task, no
```

```c
*    functionality is required here.
*/
void SSH_OS_Lock(void) {
}

/**********************************************************************
*
*       SSH_OS_Unlock()
*
* Function description
*   Unlocks the single lock used locked by a previous call to
*   SSH_OS_Lock().
*   If the entire stack executes from a single task, no
*   functionality is required here.
*/
void SSH_OS_Unlock(void) {
}

/**********************************************************************
*
*       SSH_OS_GetTime32()
*
*  Function description
*    Return the current system time in ms.
*    The value will wrap around after app. 49.7 days.
*/
U32 SSH_OS_GetTime32(void) {
  return 0;
}

/**********************************************************************
*
*       SSH_OS_GetTaskName()
*
* Function description
*   Retrieves the task name (if available from the OS and not in
*   interrupt) for the currently active task.
*
* Parameters
*   pTask: Pointer to a task identifier such as a task control block.
*
* Return value
*   Terminated string with task name.
*/
const char * SSH_OS_GetTaskName(void *pTask) {
  SSH_USE_PARA(pTask);  // Avoid warning 'parameter
 "pTask" was never referenced'.
  return "emSSH";
}

/************************** End of file **************************/
```

8.2   CRYPTO-OS integration

In a threaded execution environment individual hardware resources must be protected from simultaneous use by more than one thread. emSSH does this by surrounding use of hardware resources by calls to an OS binding layer.

To use a shared resource, emSSH will either:

- Call `CRYPTO_OS_Claim()`, use the resource, and call `CRYPTO_OS_Unclaim()` to release it, or
- Call `CRYPTO_OS_Request()` to request access to the resource. If access is granted, emSSH uses the resource and then calls `CRYPTO_OS_Unclaim()` to release it. In the case where access to the resource is not granted, emSSH will not use the resource and will not call `CRYPTO_OS_Unclaim()`.

The parameter `Unit` is a zero-based index to the hardware being requested and is defined by the specific hardware platform or target device that is in use. No hardware acceleration interface in emSSH requires more than three units (e.g. a ciphering unit, a hashing unit, and a random number generation unit). The specific requirements for each device are described in the relevant sections.

As an OS layer may well need to create mutexes or semaphores corresponding to each unit, `CRYPTO_OS_Init()` is called as part of emSSH initialization.

8.2.1   CRYPTO-OS API

Function	Description
CRYPTO_OS_Init()	Initialize CRYPTO binding to OS.
CRYPTO_OS_Claim()	Claim a hardware resource.
CRYPTO_OS_Request()	Test-and-claim a hardware resource.
CRYPTO_OS_Unclaim()	Release claim on a hardware resource.

8.2.1.1   CRYPTO_OS_Init()

Description

Initialize CRYPTO binding to OS.

Prototype

```
void CRYPTO_OS_Init(void);
```

Additional information

This function should initialize any semaphores or mutexes used for protecting each hardware unit.

8.2.1.2   CRYPTO_OS_Claim()

Description

Claim a hardware resource.

Prototype

```
void CRYPTO_OS_Claim(unsigned Unit);
```

Parameters

Parameter	Description
Unit	Zero-based index to hardware resource.

Additional information

Claim the hardware resource that corresponds to the unit index. In a threaded environment, this function should block a task requesting a resource that is already in use by using a semaphore or mutex, for example. For a super-loop or non-threaded application where there is no possibility of concurrent use of the hardware resource, this function can be empty.

8.2.1.3   CRYPTO_OS_Request()

Description

Test-and-claim a hardware resource.

Prototype

```
int CRYPTO_OS_Request(unsigned Unit);
```

Parameters

Parameter	Description
Unit	Zero-based index to hardware resource.

Return value

= 0   Resource is already in use and was not claimed.
≠ 0   Resource claimed.

Additional information

Attempt to claim the hardware resource that corresponds to the unit index. In a threaded environment, this function is a nonblocking test-and-lock of a semaphore or mutex. For a super-loop or non-threaded application where there is no possibility of concurrent use of the hardware resource, this function should always return nonzero, i.e. resource claimed.

8.2.1.4   CRYPTO_OS_Unclaim()

Description

Release claim on a hardware resource.

Prototype

```
void CRYPTO_OS_Unclaim(unsigned Unit);
```

Parameters

Parameter	Description
Unit	Zero-based index to hardware resource.

Additional information

Release the claim the hardware resource that corresponds to the unit index. This will only be called to unclaim a claimed resource.

8.2.2   CRYPTO-OS binding for embOS

The following is a sample binding for SEGGER embOS, `CRYPTO_OS_embOS.c`:

```c
/*********************************************************************
*               (c) SEGGER Microcontroller GmbH & Co. KG           *
*                       The Embedded Experts                        *
*                          www.segger.com                          *
**********************************************************************

------------------------- END-OF-HEADER ----------------------------

File        : CRYPTO_OS_embOS.c
Purpose     : SEGGER embOS CRYPTO-OS binding.

*/

/*********************************************************************
*
*       #include section
*
**********************************************************************
*/

#include "CRYPTO.h"
#include "RTOS.h"

/*********************************************************************
*
*       Preprocessor definitions, configurable
*
**********************************************************************
*/

#ifndef   CRYPTO_CONFIG_OS_MAX_UNIT
  #define CRYPTO_CONFIG_OS_MAX_UNIT      (CRYPTO_OS_MAX_INTERNAL_UNIT + 3)
#endif

/*********************************************************************
*
*       Static data
*
**********************************************************************
*/

static OS_SEMAPHORE _aSema[CRYPTO_CONFIG_OS_MAX_UNIT];

/*********************************************************************
*
*       Public functions
*
**********************************************************************
*/

/*********************************************************************
*
*       CRYPTO_OS_Claim()
*
*  Function description
*    Claim a hardware resource.
*
*  Parameters
*    Unit - Zero-based index to hardware resource.
*/
void CRYPTO_OS_Claim(unsigned Unit) {
  if (Unit >= CRYPTO_CONFIG_OS_MAX_UNIT) {
    OS_Error(OS_ERR_HW_NOT_AVAILABLE);
```

```
  }
  //
  OS_WaitCSema(&_aSema[Unit]);
}

/**********************************************************************
*
*       CRYPTO_OS_Request()
*
*  Function description
*    Request a hardware resource.
*
*  Parameters
*    Unit - Zero-based index to hardware resource.
*
*  Return value
*    == 0 - Resource is already in use and was not claimed.
*    != 0 - Resource claimed.
*/
int CRYPTO_OS_Request(unsigned Unit) {
  if (Unit >= CRYPTO_CONFIG_OS_MAX_UNIT) {
    OS_Error(OS_ERR_HW_NOT_AVAILABLE);
  }
  //
  return OS_CSemaRequest(&_aSema[Unit]);
}

/**********************************************************************
*
*       CRYPTO_OS_Unclaim()
*
*  Function description
*    Release claim on a hardware resource.
*
*  Parameters
*    Unit - Zero-based index to hardware resource.
*/
void CRYPTO_OS_Unclaim(unsigned Unit) {
  if (Unit >= CRYPTO_CONFIG_OS_MAX_UNIT) {
    OS_Error(OS_ERR_HW_NOT_AVAILABLE);
  }
  //
  OS_SignalCSema(&_aSema[Unit]);
}

/**********************************************************************
*
*       CRYPTO_OS_Init()
*
*  Function description
*    Initialize CRYPTO binding to OS.
*/
void CRYPTO_OS_Init(void) {
  unsigned Unit;
  //
  for (Unit = 0; Unit < CRYPTO_CONFIG_OS_MAX_UNIT; ++Unit) {
    OS_CreateCSema(&_aSema[Unit], 1);
  }
}

/**********************************************************************
*
*       CRYPTO_OS_Exit()
*
*  Function description
*    Deinitialize CRYPTO binding to OS.
*/
void CRYPTO_OS_Exit(void) {
```

```
  unsigned Unit;
  //
  for (Unit = 0; Unit < CRYPTO_CONFIG_OS_MAX_UNIT; ++Unit) {
    OS_DeleteCSema(&_aSema[Unit]);
  }
}

/*************************** End of file ***************************/
```

8.2.3   CRYPTO-OS binding for bare metal

The following is a sample binding for a bare metal system that has no tasking, `CRYPTO_OS_None.c`:

```c
/*********************************************************************
*                  (c) SEGGER Microcontroller GmbH                  *
*                     The Embedded Experts                          *
*                        www.segger.com                             *
**********************************************************************

----------------------- END-OF-HEADER -----------------------------

File       : CRYPTO_OS_None.c
Purpose    : Bare metal CRYPTO-OS binding.

*/

#include "CRYPTO.h"

/*********************************************************************
*
*       Public code
*
**********************************************************************
*/

/*********************************************************************
*
*       CRYPTO_OS_Claim()
*
*  Function description
*    Claim a hardware resource.
*
*  Parameters
*    Unit - Zero-based index to hardware resource.
*/
void CRYPTO_OS_Claim(unsigned Unit) {
  CRYPTO_USE_PARA(Unit);
}

/*********************************************************************
*
*       CRYPTO_OS_Request()
*
*  Function description
*    Test-and-claim a hardware resource.
*
*  Parameters
*    Unit - Zero-based index to hardware resource.
*
*  Return value
*    == 0 - Resource is already in use and was not claimed.
*    != 0 - Resource claimed.
*/
int CRYPTO_OS_Request(unsigned Unit) {
  CRYPTO_USE_PARA(Unit);
  return 1;
}

/*********************************************************************
*
*       CRYPTO_OS_Unclaim()
*
*  Function description
*    Release claim on a hardware resource.
*
```

```c
*   Parameters
*     Unit - Zero-based index to hardware resource.
*/
void CRYPTO_OS_Unclaim(unsigned Unit) {
  CRYPTO_USE_PARA(Unit);
}

/*********************************************************************
*
*       CRYPTO_OS_Init()
*
*  Function description
*    Initialize CRYPTO binding to OS.
*/
void CRYPTO_OS_Init(void) {
  /* Nothing to do. */
}

/*********************************************************************
*
*       CRYPTO_OS_Init()
*
*  Function description
*    Deinitialize CRYPTO binding to OS.
*/
void CRYPTO_OS_Exit(void) {
  /* Nothing to do. */
}

/************************** End of file **************************/
```

Chapter 9

Secure copy (add-on)

This chapter explains the API functions of emSSH which implement the secure copy function (SCP). Secure copy enables transfer of files securely to or from a remote system.

9.1   Using secure copy

The emSSH secure copy implementation emulates the Unix `scp` applicaion in order to provide file copy from the embedded target ("source mode") and file copy to the embedded target ("sink mode"). In emSSH these two modes are separated so that if you only need to copy files from the target or to the target, you do not pay in additional code and data space for the particular copy mode you never use.

9.1.1   Initializing secure copy

To initialize secure copy for sink and source modes, call `SSH_SCP_SINK_Init()`, `SSH_SCP_SOURCE_Init()`, or both. These functions take the file system API to use and the root of the file system to expose to secure copy clients.

```
SSH_SCP_SINK_Init  (&SSH_SCP_SINK_FS_Win32,   NULL);
SSH_SCP_SOURCE_Init(&SSH_SCP_SOURCE_FS_Win32, NULL);
```

If the file system root is `NULL`, emSSH sets the file system root to "/SEGGER".

Source and sink file systems are provided, in source form, for Win32 and emFile targets such that you can customize them if you need to. These file system interfaces are exposed through the following:

```
extern const SSH_SCP_SINK_FS_API   SSH_SCP_SINK_FS_Win32;
extern const SSH_SCP_SINK_FS_API   SSH_SCP_SINK_FS_FS;
extern const SSH_SCP_SOURCE_FS_API SSH_SCP_SOURCE_FS_Win32;
extern const SSH_SCP_SOURCE_FS_API SSH_SCP_SOURCE_FS_FS;
```

9.1.2   Adding support for execution

As with the shell example, secure copy is requested through an "exec" service request. You must install a "exec" service handler:

```
SSH_CHANNEL_REQUEST_Add(&SSH_CHANNEL_REQUEST_EXEC, _ExecRequest);
```

The execution request handler uses emSSH interface calls to see whether such a request should be accepted using `SSH_SCP_SINK_Accept()` and `SSH_SCP_SOURCE_Accept()`. If the request is accepted, i.e. the form of the command line is correct and there is an unused `scp` session available, the secure copy is started using `SSH_SCP_SINK_Start()` and `SSH_SCP_SOURCE_Start()`.

```
static int _ExecRequest(SSH_SESSION                 * pSelf,
                        unsigned                      Channel,
                        SSH_CHANNEL_REQUEST_PARAS * pParas) {
  int Status;
  //
  Status = SSH_SCP_SINK_Accept(pSelf, Channel, pParas);
  if (Status >= 0) {
    if (pParas->WantReply) {
      Status = SSH_CHANNEL_SendSuccess(pSelf, Channel);
    }
    Status = SSH_SCP_SINK_Start(pSelf, Status);
  } else {
    Status = SSH_SCP_SOURCE_Accept(pSelf, Channel, pParas);
    if (pParas->WantReply) {
      Status = SSH_CHANNEL_SendCompletion(pSelf, Channel, Status);
    }
    if (Status >= 0) {
      Status = SSH_SCP_SOURCE_Start(pSelf, Status);
    }
  }
  //
```

```
   if (Status < 0) {
     SSH_CHANNEL_Close(pSelf, Channel);
   }
   //
   return Status;
 }
```

Note that if you require only a readable file system or only a writable file system, you only need to initialize and start the using the appropriate source or sink API calls.

9.1.3   Processing requests

Configuring a channel and processing secure copy requests is done in much the same way as shells and terminals. One thing to note is that the receive and transmit buffers, when using secure copy, cannot be shared and separate buffers must be provided for each.

```
static void _SSH_Buffering(SSH_CONTEXT * pContext) {
  int Status;
  //
  SSH_SESSION_Alloc(&pContext->pSession);
  if (pContext->pSession == NULL) {
    return;
  }
  //
  SSH_SESSION_Init(pContext->pSession, pContext->Socket, &_IP_Transport);
  SSH_SESSION_ConfBuffers(pContext->pSession,
                          _aRxBuffer, sizeof(_aRxBuffer),
                          _aTxBuffer, sizeof(_aTxBuffer));
  //
  do {
    Status = SSH_SESSION_Process(pContext->pSession);
  } while (Status >= 0);
  //
  SEGGER_SYS_IP_Close(pContext->Socket);
  pContext->Socket = 0;
}
```

9.1.4   Sample application

The following application is a sample that shows how to use the secure copy facility.

```c
/*********************************************************************
*                 (c) SEGGER Microcontroller GmbH & Co. KG           *
*                     The Embedded Experts                           *
*                        www.segger.com                              *
**********************************************************************

------------------------- END-OF-HEADER -----------------------------

File        : SSH_SCP_FS_Server.c
Purpose     : Simple SCP server that targets the embedded or null file
              system.

*/

/*********************************************************************
*
*       #include Section
*
**********************************************************************
*/

#include "SSH.h"
#include "SEGGER_SYS.h"
#include "SEGGER.h"
#ifndef WIN32
#include "FS.h"
#endif

/*********************************************************************
*
*       Defines, configurable
*
**********************************************************************
*/

#define BANNER \
  "\r\n"                                                              \
  "*************************************************************\r\n" \
  "* This server is powered by SEGGER emSSH.  It simply works! *\r\n" \
  "*************************************************************\r\n" \
  "\r\n"

/*********************************************************************
*
*       Local types
*
**********************************************************************
*/

typedef struct {
  int           Socket;
  SSH_SESSION * pSession;
} SSH_CONTEXT;

/*********************************************************************
*
*       Static const data
*
**********************************************************************
*/

static const SSH_TRANSPORT_API _IP_Transport = {
  SEGGER_SYS_IP_Send,
  SEGGER_SYS_IP_Recv,
  SEGGER_SYS_IP_Close,
};

/*********************************************************************
*
*       Static data
*
```

```
******************************************************************
*/

static U32        _aRxBuffer[SSH_SCP_INITIAL_WINDOW_SIZE/4];
static U32        _aTxBuffer[SSH_SCP_INITIAL_WINDOW_SIZE/4];
static SSH_CONTEXT _aTask   [SSH_SCP_CONFIG_MAX_SESSIONS];

/******************************************************************
*
*       Static code
*
******************************************************************
*/

/******************************************************************
*
*       _PrintAppSignOn()
*
*  Function description
*    Displays the application's help information on stderr.
*/
static void _PrintAppSignOn(void) {
  SEGGER_SYS_IO_Printf("\n");
  SEGGER_SYS_IO_Printf("emSSH SCP Server V%s ", SSH_GetVersionText());
  SEGGER_SYS_IO_Printf("compiled " __DATE__ " " __TIME__ "\n");
  SEGGER_SYS_IO_Printf("(c) 2015-2019 SEGGER Microcontroller GmbH    www.segger.com\n\n");
}

/******************************************************************
*
*       _UserauthServiceRequest()
*
*  Function description
*    Request the user authentication service.
*
*  Parameters
*    pSelf       - Pointer to session.
*    sServiceName - Service being requested.
*
*  Return value
*    >= 0 - Success.
*    <  0 - Error.
*
*  Additional information
*    Displays a banner before user authentication commences.
*/
static int _UserauthServiceRequest(SSH_SESSION *pSelf, const char *sServiceName) {
  int Status;
  //
  Status = SSH_SESSION_SendServiceAccept(pSelf, sServiceName);
  if (Status >= 0) {
    Status = SSH_SESSION_SendUserauthBanner(pSelf, BANNER, "en");
  }
  //
  return Status;
}

/******************************************************************
*
*       _UserauthRequestNone()
*
*  Function description
*    Request authentication of user with method "none".
*
*  Parameters
*    pSession  - Pointer to session.
*    pReqParas - Pointer to user authentication request parameters.
*
*  Return value
*    >= 0 - Success.
*    <  0 - Error.
*/
static int _UserauthRequestNone(SSH_SESSION                 *pSession,
                                SSH_USERAUTH_REQUEST_PARAS *pReqParas) {
  SSH_USERAUTH_NONE_PARAS NoneParas;
  int                     Status;
```

```c
  //
  SSH_USE_PARA(pSession);
  //
  Status = SSH_USERAUTH_NONE_ParseParas(pReqParas, &NoneParas);
  if (Status < 0) {
    Status = SSH_ERROR_USERAUTH_FAIL;
  } else if (pReqParas->UserNameLen == 4 &&
             SSH_MEMCMP(pReqParas->pUserName, "anon", 4) == 0) {
    Status = 0;
  } else {
    Status = SSH_ERROR_USERAUTH_FAIL;
  }
  //
  return Status;
}

/**********************************************************************
*
*       _UserauthRequestPassword()
*
*  Function description
*    Request authentication of user with method "password".
*
*  Parameters
*    pSession  - Pointer to session.
*    pReqParas - Pointer to user authentication request parameters.
*
*  Return value
*    >= 0 - Success.
*    <  0 - Error.
*/
static int _UserauthRequestPassword(SSH_SESSION               *pSession,
                                    SSH_USERAUTH_REQUEST_PARAS *pReqParas) {
  SSH_USERAUTH_PASSWORD_PARAS PasswordParas;
  int                         Status;
  //
  SSH_USE_PARA(pSession);
  //
  Status = SSH_USERAUTH_PASSWORD_ParseParas(pReqParas, &PasswordParas);
  if (Status < 0) {
    Status = SSH_ERROR_USERAUTH_FAIL;
  } else if (pReqParas->UserNameLen == 5 && SSH_STRNCMP(pReqParas-
>pUserName, "admin", 5) == 0) {

    if (PasswordParas.PasswordLen == 6 && SSH_MEMCMP(PasswordParas.pPassword, "secret", 6) == 0) {
      Status = 0;
    } else {
      Status = SSH_ERROR_USERAUTH_FAIL;
    }
  } else {
    Status = SSH_ERROR_USERAUTH_FAIL;
  }
  //
  return Status;
}

/**********************************************************************
*
*       _ExecRequest()
*
*  Function description
*    Request execution of an application.
*
*  Parameters
*    pSelf   - Pointer to session.
*    Channel - Zero-based channel index of request.
*    pParas  - Pointer to channel request parameters.
*
*  Return value
*    >= 0 - Success.
*    <  0 - Error.
*/
static int _ExecRequest(SSH_SESSION               * pSelf,
                        unsigned                    Channel,
                        SSH_CHANNEL_REQUEST_PARAS * pParas) {
  int Status;
```

```c
    //
    Status = SSH_SCP_SINK_Accept(pSelf, Channel, pParas);
    if (Status >= 0) {
      if (pParas->WantReply) {
        Status = SSH_CHANNEL_SendSuccess(pSelf, Channel);
      }
      Status = SSH_SCP_SINK_Start(pSelf, Status);
    } else {
      Status = SSH_SCP_SOURCE_Accept(pSelf, Channel, pParas);
      if (pParas->WantReply) {
        Status = SSH_CHANNEL_SendCompletion(pSelf, Channel, Status);
      }
      if (Status >= 0) {
        Status = SSH_SCP_SOURCE_Start(pSelf, Status);
      }
    }
    //
    if (Status < 0) {
      SSH_CHANNEL_Close(pSelf, Channel);
    }
    //
    return Status;
}

/*********************************************************************
*
*       _SSH_Buffering()
*
*  Function description
*    Handles SSH connection.
*
*  Parameters
*    pVoid - Pointer to user-provided context.
*/
static void _SSH_Buffering(SSH_CONTEXT * pContext) {
  int Status;
  //
  SSH_SESSION_Alloc(&pContext->pSession);
  if (pContext->pSession == NULL) {
    return;
  }
  //
  SSH_SESSION_Init(pContext->pSession, pContext->Socket, &_IP_Transport);
  SSH_SESSION_ConfBuffers(pContext->pSession,
                          _aRxBuffer, sizeof(_aRxBuffer),
                          _aTxBuffer, sizeof(_aTxBuffer));
  //
  do {
    Status = SSH_SESSION_Process(pContext->pSession);
  } while (Status >= 0);
  //
  SEGGER_SYS_IP_Close(pContext->Socket);
  pContext->Socket = 0;
}

/*********************************************************************
*
*       Public code
*
**********************************************************************
*/

/*********************************************************************
*
*       MainTask()
*
*  Function description
*    Application entry point.
*/
void MainTask(void);
void MainTask(void) {
  int BoundSocket;
  int Socket;
  int i;
  const char * sVolumeName = "";
  //
```

```c
  SEGGER_SYS_Init();
  SEGGER_SYS_IP_Init();
  SSH_Init();
  //
  // Initialize SCP in sink and source mode.
  //
#ifdef WIN32
  SSH_SCP_SINK_Init  (&SSH_SCP_SINK_FS_Win32,   NULL);
  SSH_SCP_SOURCE_Init(&SSH_SCP_SOURCE_FS_Win32, NULL);
#else
  FS_Init();
  FS_FAT_SupportLFN();
  FS_FormatLLIfRequired(sVolumeName);
  if (FS_IsHLFormatted(sVolumeName) == 0) {
    FS_X_Log("High-level format\n");
    (void)FS_Format(sVolumeName, NULL);
  }
  FS_MkDir("\\SEGGER");
  //
  SSH_SCP_SINK_Init  (&SSH_SCP_SINK_FS_FS,   NULL);
  SSH_SCP_SOURCE_Init(&SSH_SCP_SOURCE_FS_FS, NULL);
#endif
  //
  // Hook the userauth service to show a banner.
  //
  SSH_SERVICE_Add(&SSH_SERVICE_USERAUTH, &_UserauthServiceRequest);
  //
  // Support None and Password user authentication methods.
  //
  SSH_USERAUTH_METHOD_Add(&SSH_USERAUTH_METHOD_NONE,     &_UserauthRequestNone);
  SSH_USERAUTH_METHOD_Add(&SSH_USERAUTH_METHOD_PASSWORD, &_UserauthRequestPassword);
  //
  // Add support for scp execution.
  //
  SSH_CHANNEL_REQUEST_Add(&SSH_CHANNEL_REQUEST_EXEC, _ExecRequest);
  SSH_CHANNEL_REQUEST_Add(&SSH_CHANNEL_REQUEST_ENV,  NULL);
  //
  // Bind SSH port.
  //
  BoundSocket = SEGGER_SYS_IP_Bind(22);
  if (BoundSocket < 0) {
    SEGGER_SYS_OS_Halt(100);
  }
  _PrintAppSignOn();
  //
  for (;;) {
    //
    // Wait for an incoming connection.
    //
    SEGGER_SYS_IO_Printf("Awaiting connection...\n");
    Socket = SEGGER_SYS_IP_Accept(BoundSocket);
    if (Socket < 0) {
      continue;
    }
    //
    // Open an SSH connection.
    //
    SEGGER_SYS_IO_Printf("Connection made...\n");
    //
    // Create a task to handle the socket connections.
    //
    for (i = 0; i < (int)SEGGER_COUNTOF(_aTask); ++i) {
      if (_aTask[i].Socket == 0) {
        break;
      }
    }
    if (i < (int)SEGGER_COUNTOF(_aTask)) {
      _aTask[i].Socket = Socket;
      _SSH_Buffering(&_aTask[i]);
    } else {
      SEGGER_SYS_IP_Close(Socket);
    }
  }
}

/*************************** End of file ***************************/
```

9.2   Configuring secure copy

9.2.1   Compile-time definitions

9.2.1.1   Maximum number of SCP sessions

Default

```
#define SSH_SCP_CONFIG_MAX_SESSIONS   1
```

Override

To define a non-default value, define this symbol in `SSH_Conf.h`.

Description

This defines the maximum number of simultaneous secure copy operations supported. This must be set less than or equal to the number of SSH sessions configured by `SSH_CONFIG_MAX_SESSIONS`.

Configuring fewer sessions will reduce the RAM footprint of the `scp` protocol.

See also

Maximum number of sessions on page 213.

9.2.1.2   Maximum path length

Default

```
#define SSH_SCP_CONFIG_PATH_MAX      260
```

Override

To define a non-default value, define this symbol in `SSH_Conf.h`.

Description

This defines the maximum length of a path name. Path names vary between file system implementations, and so do their maximum lengths. This definition is used to reserve space to handle path names inside emSSH's `scp` protocol implementation and can be configured to a value smaller than the underlying file system's maximum path name.

Note that emSSH fully enforces the maximum path name constraint without buffer overflow and will not allow any file transfer with longer path names.

9.2.1.3   Initial window size

Default

```
#define SSH_SCP_INITIAL_WINDOW_SIZE    8192
```

Override

To define a non-default value, define this symbol in `SSH_Conf.h`.

Description

This defines the initial advertised window sizes. With larger window sizes, more data will be sent through the secure channel before a subsequent window size adjustment message is sent. Larger window sizes provide better transfer performance but only if the target system is able to process the incoming data in an efficient fashion such that the sender is not throttled.

9.2.1.4 Copy-buffer size

Default

```
#define SSH_SCP_FILE_CHUNK_SIZE    1024
```

Override

To define a non-default value, define this symbol in `SSH_Conf.h`.

Description

The defines the buffer size, and therefore the maximum number of bytes read from a file, in SCP source mode. This buffer buffer is allocated on the stack and therefore the calling task's workspace must be sized accordingly.

9.2.1.5 Root of virtual file system

Default

```
#define SSH_SCP_CONFIG_VIRTUAL_ROOT  "/SEGGER"
```

Override

To define a non-default value, define this symbol in `SSH_Conf.h`.

Description

This defines the root of the file system accessible from the `scp` protocol if no other root is provided when configuring source and sink mode.

9.3   API Reference

The table below lists the secure copy functions provided by the emSSH API.

Function	Description
Sink functions	
`SSH_SCP_SINK_Init()`	Initialize SCP sink component.
`SSH_SCP_SINK_Accept()`	Query acceptance of SCP sink request.
`SSH_SCP_SINK_Start()`	Initiate SCP sink protocol.
Source functions	
`SSH_SCP_SOURCE_Init()`	Initialize SCP sink component.
`SSH_SCP_SOURCE_Accept()`	Query acceptance of SCP source request.
`SSH_SCP_SOURCE_Start()`	Initiate SCP source protocol.

9.3.1   SSH_SCP_SINK_Init()

Description

Initialize SCP sink component.

Prototype

```
int SSH_SCP_SINK_Init(const SSH_SCP_SINK_FS_API * pAPI,
                      const char                 * sRoot);
```

Parameters

Parameter	Description
pAPI	Pointer to file system API.
sRoot	System-specific root of file system.

Return value

≥ 0   Success.
< 0   Error initiating SCP sink.

9.3.2   SSH_SCP_SINK_Accept()

Description

Query acceptance of SCP sink request.

Prototype

```
int SSH_SCP_SINK_Accept(SSH_SESSION            * pSession,
                        unsigned                 Channel,
                        SSH_CHANNEL_REQUEST_PARAS * pParas);
```

Parameters

Parameter	Description
pSession	Pointer to SSH session.
Channel	Local channel ID of incoming channel request.
pParas	Pointer to channel request parameters.

Return value

≥ 0   Success, SCP sink request started, index to be used.
< 0   Not SCP sink request.

9.3.3   SSH_SCP_SINK_Start()

Description

Initiate SCP sink protocol.

Prototype

```
int SSH_SCP_SINK_Start(SSH_SESSION * pSelf,
                       unsigned      Index);
```

Parameters

Parameter	Description
pSelf	Pointer to SSH session.
Index	Zero-based SCP session index, returned by SSH_SCP_SINK_Accept().

Return value

≥ 0   Success.
< 0   Error initiating session.

9.3.4   SSH_SCP_SOURCE_Init()

Description

Initialize SCP sink component.

Prototype

```
int SSH_SCP_SOURCE_Init(const SSH_SCP_SOURCE_FS_API * pAPI,
                        const char                  * sRoot);
```

Parameters

Parameter	Description
pAPI	Pointer to file system API.
sRoot	System-specific root of file system.

Return value

≥ 0   Success.
< 0   Error initiating SCP source.

9.3.5   SSH_SCP_SOURCE_Accept()

Description

Query acceptance of SCP source request.

Prototype

```
int SSH_SCP_SOURCE_Accept(SSH_SESSION              * pSession,
                          unsigned                   Channel,
                          SSH_CHANNEL_REQUEST_PARAS * pParas);
```

Parameters

Parameter	Description
pSession	Pointer to SSH session.
Channel	Local channel ID of incoming channel request.
pParas	Pointer to channel request parameters.

Return value

≥ 0   Success, SCP source request started, index to be used.
< 0   Not SCP source request.

9.3.6 SSH_SCP_SOURCE_Start()

Description

Initiate SCP source protocol.

Prototype

```
int SSH_SCP_SOURCE_Start(SSH_SESSION * pSelf,
                         unsigned       Index);
```

Parameters

Parameter	Description
pSelf	Pointer to SSH session.
Index	Zero-based SCP session index, returned by SSH_SCP_SOURCE_Accept().

Return value

≥ 0   Success.
< 0   Error initiating session.

9.4   File system bindings

9.4.1   Sink to file system binding

9.4.1.1   SSH_SCP_SINK_FS_API

Type definition

```
typedef struct {
  SSH_SCP_SINK_FS_CONFIG        * pfConfig;
  SSH_SCP_SINK_FS_INIT          * pfInit;
  SSH_SCP_SINK_FS_CREATE_FILE   * pfCreateFile;
  SSH_SCP_SINK_FS_WRITE_FILE    * pfWriteFile;
  SSH_SCP_SINK_FS_CLOSE_FILE    * pfCloseFile;
  SSH_SCP_SINK_FS_ENTER_FOLDER  * pfEnterFolder;
  SSH_SCP_SINK_FS_EXIT_FOLDER   * pfExitFolder;
  SSH_SCP_SINK_FS_DECODE_STATUS * pfDecodeStatus;
} SSH_SCP_SINK_FS_API;
```

Structure members

Member	Description
pfConfig	Configure sink
pfInit	Initialize sink transfer instance
pfCreateFile	Create file
pfWriteFile	Write data to file
pfCloseFile	Close file
pfEnterFolder	Enter folder
pfExitFolder	Exit folder
pfDecodeStatus	Decode error status

9.4.1.2   SSH_SCP_SINK_FS_CONFIG

Description

Configure SCP sink.

Type definition

```
typedef int SSH_SCP_SINK_FS_CONFIG(const char * sRoot);
```

Parameters

Parameter	Description
sRoot	Zero-terminated string defining SCP file system virtual root.

Return value

≥ 0   Success.
< 0   Error status.

Additional information

This function is called before any other and is passed the host folder that is the root of the file system exposed using secure copy functions.

9.4.1.3   SSH_SCP_SINK_FS_INIT

Description

Initialize SCP sink instance.

Type definition

```
typedef int SSH_SCP_SINK_FS_INIT(       unsigned Index,
                                 const char   * pPath,
                                       unsigned PathLen);
```

Parameters

Parameter	Description
Index	Zero-based index of SCP file system transfer instance to initialize, range [0…SSH_SCP_CONFIG_MAX_SESSIONS-1].
pPath	Pointer to root-relative path string for this transfer.
PathLen	Octet length of root-relative path string for this transfer.

Return value

≥ 0   Success.
< 0   Error status.

Additional information

This function is called to initialize a file system transfer instance. It is called before any other transfer takes place.

9.4.1.4   SSH_SCP_SINK_FS_CREATE_FILE

Description

Create file.

Type definition

```
typedef int SSH_SCP_SINK_FS_CREATE_FILE(        unsigned Index,
                                                unsigned Mode,
                                                U32      Len,
                                                U32      ModTime,
                                                U32      AccTime,
                                          const char   * sName);
```

Parameters

Parameter	Description
Index	Zero-based index of SCP file system transfer instance to initialize, range [0…SSH_SCP_CONFIG_MAX_SESSIONS-1].
Mode	Standard Unix file mode.
Len	Length of file. This can be used to preallocate the file to increase performance.
ModTime	Last modification time, Unix format, number of seconds since 1 Jan 1970. Zero indicates time is not set.
AccTime	Last access time, Unix format, number of seconds since 1 Jan 1970. Zero indicates time is not set.
sName	Zero-terminated file name (not path name) of the file to create.

Return value

≥ 0   Success.
< 0   Error status.

Additional information

The controlling SCP sink software will create a file, write to it, and finally close it. There will never be a case where two files are open using the same index.

9.4.1.5   SSH_SCP_SINK_FS_WRITE_FILE

Description

Write data to file.

Type definition

```
typedef int SSH_SCP_SINK_FS_WRITE_FILE(         unsigned Index,
                                        const U8    * pData,
                                                unsigned DataLen);
```

Parameters

Parameter	Description
Index	Zero-based index of SCP file system transfer instance to initialize, range [0…SSH_SCP_CONFIG_MAX_SESSIONS-1].
pData	Pointer to object to write.
DataLen	Length of the object to write.

Return value

≥ 0   Success.
< 0   Error status.

9.4.1.6   SSH_SCP_SINK_FS_CLOSE_FILE

Description

Close file.

Type definition

```c
typedef int SSH_SCP_SINK_FS_CLOSE_FILE(unsigned Index);
```

Parameters

Parameter	Description
Index	Zero-based index of SCP file system transfer instance to initialize, range [0…SSH_SCP_CONFIG_MAX_SESSIONS-1].

Return value

≥ 0   Success.
< 0   Error status.

9.4.1.7   SSH_SCP_SINK_FS_ENTER_FOLDER

Description

Enter folder.

Type definition

```
typedef int SSH_SCP_SINK_FS_ENTER_FOLDER(        unsigned Index,
                                                 unsigned Mode,
                                                 U32      ModTime,
                                                 U32      AccTime,
                                         const char    * sName);
```

Parameters

Parameter	Description
Index	Zero-based index of SCP file system transfer instance to initialize, range [0…SSH_SCP_CONFIG_MAX_SESSIONS-1].
Mode	Standard Unix file mode.
ModTime	Last modification time, Unix format, number of seconds since 1 Jan 1970. Zero indicates time is not set.
AccTime	Last access time, Unix format, number of seconds since 1 Jan 1970. Zero indicates time is not set.
sName	Zero-terminated folder name (not path name) of the folder to create.

Return value

≥ 0   Success.
< 0   Error status.

Additional information

The controlling SCP sink software will call Enter Folder to create a folder; all subsequent Create File requests require that files are created in this folder. When folder copy is complete, Exit Folder will return to the parent folder.

The implementation of this method should create a folder with the given mode if the folder does not exist.

9.4.1.8   SSH_SCP_SINK_FS_EXIT_FOLDER

Description

Exit folder.

Type definition

```
typedef int SSH_SCP_SINK_FS_EXIT_FOLDER(unsigned Index);
```

Parameters

Parameter	Description
Index	Zero-based index of SCP file system transfer instance to initialize, range [0…SSH_SCP_CONFIG_MAX_SESSIONS-1].

Return value

≥ 0   Success.
< 0   Error status.

Additional information

The implementation of this method should return to the parent folder of the current folder.

9.4.1.9   SSH_SCP_SINK_FS_DECODE_STATUS

Description

Decode error status.

Type definition

```
typedef const char * SSH_SCP_SINK_FS_DECODE_STATUS(int Status);
```

Parameters

Parameter	Description
Status	Status to decode.

Return value

= NULL   When status is "OK, no error".
≠ NULL   Zero-terminated string describing the error.

9.4.2   Sample bindings

This section presents sample bindings for source and sink modes for both Windows and emFile.

9.4.2.1   SSH-SCP-SOURCE for Windows

```c
/**********************************************************************
*               (c) SEGGER Microcontroller GmbH & Co. KG           *
*                       The Embedded Experts                       *
*                          www.segger.com                          *
**********************************************************************

----------------------- END-OF-HEADER ----------------------------

File        : SSH_SCP_SOURCE_FS_Win32.c
Purpose     : SSH-SCP source file system access for Win32.

*/

/**********************************************************************
*
*       #include Section
*
**********************************************************************
*/

#include "SSH.h"
#include <stdio.h>
#include <windows.h>

/**********************************************************************
*
*       Local types
*
**********************************************************************
*/

typedef struct {
  FILE            * pFile;
  WIN32_FIND_DATA   FindData;
  HANDLE            FindHandle;
  char              aPath[SSH_SCP_CONFIG_PATH_MAX];  // Full path constructed from file
                                                     // system root path plus path from
                                                     // scp command line
} SSH_SCP_SOURCE_FS_WIN32_CONTEXT;

typedef struct {
  SSH_SCP_SOURCE_FS_WIN32_CONTEXT aContext [SSH_SCP_CONFIG_MAX_SESSIONS];
  char                            aRootPath[SSH_SCP_CONFIG_PATH_MAX];
} SSH_SCP_SOURCE_GLOBALS;

/**********************************************************************
*
*       Static data
*
**********************************************************************
*/

static SSH_SCP_SOURCE_GLOBALS _SSH_SCP_SOURCE_FS_WIN32_Globals;

/**********************************************************************
*
*       Static code
*
**********************************************************************
*/

/**********************************************************************
*
*       _SSH_SCP_SOURCE_PathTrim()
*
*  Function description
*    Remove all characters from final '/' or '\' to end.
```

```
 *
 *  Parameters
 *    sName - Pointer to zero-terminated string.
 */
static void _SSH_SCP_SOURCE_PathTrim(char *sName) {
  char *s;
  //
  s = SSH_STRRCHR(sName, '/');
  if (s == NULL) {
    s = SSH_STRRCHR(sName, '\\');
  }
  if (s != NULL && s != sName) {
    *s = 0;
  }
}

/*********************************************************************
 *
 *       _SSH_SCP_SOURCE_PathAdd()
 *
 *  Function description
 *    Append path data to length-limited string.
 *
 *  Parameters
 *    sName   - Pointer to zero-terminated string.
 *    NameLen - Capacity of the zero-terminated string.
 *    sText   - Zero-terminated text to append to zero-terminated string.
 *
 *  Return value
 *    == 0                       - Appended successfully.
 *    == SSH_ERROR_PATH_TOO_LONG - String overflow.
 */
static int _SSH_SCP_SOURCE_PathAdd(char *sName, unsigned NameLen, const char *sText) {
  if (sText[0] == '/' && sName[0] != 0 && sName[SSH_StrLen(sName)-1] == '/') {
    ++sText;
  }
  if (SSH_StrLen(sName) + SSH_StrLen(sText) + 1 >= NameLen) {
    return SSH_ERROR_PATH_TOO_LONG;
  }
  //
  SSH_STRNCAT(sName, sText, NameLen);
  return 0;
}

/*********************************************************************
 *
 *       _SSH_SCP_SOURCE_FS_ConstructData()
 *
 *  Function description
 *    Fill in file information structure.
 *
 *  Parameters
 *    pContext  - Pointer to context.
 *    pInfo     - Pointer to object that receives the file information.
 *    pFindData - Pointer to Win32 find data.
 *
 *  Return value
 *    >= 0                       - File information filled in successfully.
 *    == SSH_ERROR_PATH_TOO_LONG - Full path name is too long.
 */
static int _SSH_SCP_SOURCE_FS_ConstructData(SSH_SCP_SOURCE_FS_WIN32_CONTEXT * pContext,
                                            SSH_SCP_SOURCE_FILE_INFO        * pInfo,
                                            WIN32_FIND_DATA                 * pFindData) {
  int Status;
  //
  pInfo->Length   = pFindData->nFileSizeLow;
  pInfo->IsFolder = (pFindData->dwFileAttributes & FILE_ATTRIBUTE_DIRECTORY) != 0;
  pInfo->Mode     = (pFindData->dwFileAttributes & FILE_ATTRIBUTE_READONLY) ? 0444 : 0666;
  //
  pInfo->aPathName[0] = 0;
  Status = _SSH_SCP_SOURCE_PathAdd(pInfo->aPathName, sizeof(pInfo->aPathName), pContext-
>aPath);
  if (Status >= 0) {
    _SSH_SCP_SOURCE_PathTrim(pInfo->aPathName);
    Status = _SSH_SCP_SOURCE_PathAdd(pInfo->aPathName, sizeof(pInfo->aPathName), "/");
    if (Status >= 0) {
```

```c
        Status = _SSH_SCP_SOURCE_PathAdd(pInfo->aPathName, sizeof(pInfo-
>aPathName), pFindData->cFileName);
    }
  }
  return Status;
}

/**********************************************************************
*
*       _SSH_SCP_SOURCE_FS_WIN32_Config()
*
*  Function description
*    Implementation of Config API call.
*
*  Parameters
*    sRoot - Pointer to zero-terminated virtual SCP root directory.
*
*  Return value
*    >= 0 - Success.
*    <  0 - Error status.
*/
static int _SSH_SCP_SOURCE_FS_WIN32_Config(const char *sRoot) {
  _SSH_SCP_SOURCE_FS_WIN32_Globals.aRootPath[0] = 0;
  return _SSH_SCP_SOURCE_PathAdd(_SSH_SCP_SOURCE_FS_WIN32_Globals.aRootPath,
                                 sizeof(_SSH_SCP_SOURCE_FS_WIN32_Globals.aRootPath),
                                 sRoot);
}

/**********************************************************************
*
*       _SSH_SCP_SOURCE_FS_WIN32_Init()
*
*  Function description
*    Implementation of Init API call.
*
*  Parameters
*    Index - Zero-based SCP session index.
*    sPath - Pointer to zero-terminated initial path from SCP command line.
*
*  Return value
*    >= 0 - Success.
*    <  0 - Error status.
*/
static int _SSH_SCP_SOURCE_FS_WIN32_Init(unsigned Index, const char *sPath) {
  SSH_SCP_SOURCE_FS_WIN32_CONTEXT * pContext;
  int                               Status;
  //
  pContext = &_SSH_SCP_SOURCE_FS_WIN32_Globals.aContext[Index];
  //
  SSH_MEMSET(pContext, 0, sizeof(*pContext));
  pContext->FindHandle = INVALID_HANDLE_VALUE;
  //
  pContext->aPath[0] = 0;
  Status = _SSH_SCP_SOURCE_PathAdd(pContext->aPath, sizeof(pContext-
>aPath), _SSH_SCP_SOURCE_FS_WIN32_Globals.aRootPath);
  if (Status >= 0) {
    Status = _SSH_SCP_SOURCE_PathAdd(pContext->aPath, sizeof(pContext->aPath), "/");
    if (Status >= 0) {
      Status = _SSH_SCP_SOURCE_PathAdd(pContext->aPath, sizeof(pContext->aPath), sPath);
    }
  }
  return 0;
}

/**********************************************************************
*
*       _SSH_SCP_SOURCE_FS_WIN32_FindFirst()
*
*  Function description
*    Implementation of Find First API call.
*
*  Parameters
*    Index - Zero-based SCP session index.
*    pInfo - Pointer to object that receives file information.
*
*  Return value
```

```
*     >= 0                          - Success.
*     == SSH_ERROR_NO_MORE_FILES - Folder enumeration complete (i.e. empty folder).
*     <  0                          - Error status.
*/
static int _SSH_SCP_SOURCE_FS_WIN32_FindFirst(unsigned Index, SSH_SCP_SOURCE_FILE_INFO *pInfo) {
  SSH_SCP_SOURCE_FS_WIN32_CONTEXT * pContext;
  WIN32_FIND_DATA                 FindData;
  //
  pContext = &_SSH_SCP_SOURCE_FS_WIN32_Globals.aContext[Index];
  //
  // Close down any open handle; this is just belts and braces.
  //
  if (pContext->FindHandle != INVALID_HANDLE_VALUE) {
    FindClose(pContext->FindHandle);
    pContext->FindHandle = INVALID_HANDLE_VALUE;
  }
  //
  pContext->FindHandle = FindFirstFileA(pContext->aPath, &FindData);
  if (pContext->FindHandle == INVALID_HANDLE_VALUE) {
    return SSH_ERROR_NO_MORE_FILES;
  }
  //
  return _SSH_SCP_SOURCE_FS_ConstructData(pContext, pInfo, &FindData);
}

/*********************************************************************
*
*       _SSH_SCP_SOURCE_FS_WIN32_FindNext()
*
*  Function description
*    Implementation of Find Next API call.
*
*  Parameters
*    Index - Zero-based SCP session index.
*    pInfo - Pointer to object that receives file information.
*
*  Return value
*     >= 0                          - Success.
*     == SSH_ERROR_NO_MORE_FILES - Folder enumeration complete, no additional information.
*     <  0                          - Error status.
*/
static int _SSH_SCP_SOURCE_FS_WIN32_FindNext(unsigned Index, SSH_SCP_SOURCE_FILE_INFO *pInfo) {
  SSH_SCP_SOURCE_FS_WIN32_CONTEXT * pContext;
  WIN32_FIND_DATA                 FindData;
  //
  pContext = &_SSH_SCP_SOURCE_FS_WIN32_Globals.aContext[Index];
  //
  if (pContext->FindHandle == INVALID_HANDLE_VALUE) {
    return SSH_ERROR_NO_MORE_FILES;
  }
  //
  if (FindNextFileA(pContext->FindHandle, &FindData) == 0) {
    FindClose(pContext->FindHandle);
    pContext->FindHandle = INVALID_HANDLE_VALUE;
    return SSH_ERROR_NO_MORE_FILES;
  } else {
    return _SSH_SCP_SOURCE_FS_ConstructData(pContext, pInfo, &FindData);
  }
}

/*********************************************************************
*
*       _SSH_SCP_SOURCE_FS_WIN32_FindClose()
*
*  Function description
*    Implementation of Find Close API call.
*
*  Parameters
*    Index - Zero-based SCP session index.
*
*  Return value
*     >= 0 - Success.
*     <  0 - Error status.
*/
static int _SSH_SCP_SOURCE_FS_WIN32_FindClose(unsigned Index) {
  SSH_SCP_SOURCE_FS_WIN32_CONTEXT *pContext;
```

```
    //
    pContext = &_SSH_SCP_SOURCE_FS_WIN32_Globals.aContext[Index];
    //
    if (pContext->FindHandle != INVALID_HANDLE_VALUE) {
      FindClose(pContext->FindHandle);
      pContext->FindHandle = INVALID_HANDLE_VALUE;
    }
    //
    return 0;
}

/**********************************************************************
*
*       _SSH_SCP_SOURCE_FS_WIN32_OpenFile()
*
*  Function description
*    Implementation of Open File API call.
*
*  Parameters
*    Index - Zero-based SCP session index.
*    sName - Pointer to zero-terminated full path name to file.
*            Note that this is writable to allow in-place modification
*            of path separators.
*
*  Return value
*    >= 0 - Success.
*    <  0 - Error status.
*/
static int _SSH_SCP_SOURCE_FS_WIN32_OpenFile(unsigned Index, char *sName) {
    SSH_SCP_SOURCE_FS_WIN32_CONTEXT *pContext;
    //
    pContext = &_SSH_SCP_SOURCE_FS_WIN32_Globals.aContext[Index];
    //
    if (pContext->pFile != NULL) {
      fclose(pContext->pFile);
    }
    //
    // Close down any open file; this is just belts and braces.
    //
    pContext->pFile = fopen(sName, "rb");
    if (pContext->pFile == NULL) {
      return SSH_ERROR_CANT_READ_FILE;
    } else {
      return 0;
    }
}

/**********************************************************************
*
*       _SSH_SCP_SOURCE_FS_WIN32_ReadFile()
*
*  Function description
*    Implementation of Read File API call.
*
*  Parameters
*    Index   - Zero-based SCP session index.
*    pData   - Pointer to object that receives the file data.
*    DataLen - Octet length of the object that receives the file data.
*
*  Return value
*    >  0 - Number of octets read from file (with more data to read from file).
*    == 0 - End of file indication (with no more data to read from file).
*    <  0 - Error status.
*/
static int _SSH_SCP_SOURCE_FS_WIN32_ReadFile(unsigned Index, U8 *pData, unsigned DataLen) {
    SSH_SCP_SOURCE_FS_WIN32_CONTEXT * pContext;
    unsigned                         L;
    //
    pContext = &_SSH_SCP_SOURCE_FS_WIN32_Globals.aContext[Index];
    //
    if (pContext->pFile == NULL) {
      return SSH_ERROR_CANT_READ_FILE;
    } else {
      L = (unsigned)fread(pData, 1, DataLen, pContext->pFile);
      if (ferror(pContext->pFile)) {
        return SSH_ERROR_CANT_READ_FILE;
```

```
    } else {
      return L;
    }
  }
}

/**********************************************************************
*
*       _SSH_SCP_SOURCE_FS_WIN32_CloseFile()
*
*  Function description
*    Implementation of Close File API call.
*
*  Parameters
*    Index - Zero-based SCP session index.
*
*  Return value
*    >= 0 - Success.
*    <  0 - Error status.
*/
static int _SSH_SCP_SOURCE_FS_WIN32_CloseFile(unsigned Index) {
  SSH_SCP_SOURCE_FS_WIN32_CONTEXT * pContext;
  //
  pContext = &_SSH_SCP_SOURCE_FS_WIN32_Globals.aContext[Index];
  //
  if (pContext->pFile) {
    fclose(pContext->pFile);
  }
  pContext->pFile = NULL;
  //
  return 0;
}

/**********************************************************************
*
*       _SSH_SCP_SOURCE_FS_WIN32_DecodeStatus()
*
*  Function description
*    Decode error status.
*
*  Parameters
*    Status - Error status.
*
*  Return value
*    Pointer to zero-terminate string if error, or NULL if success.
*/
static const char * _SSH_SCP_SOURCE_FS_WIN32_DecodeStatus(int Status) {
  return Status == 0 ? NULL : "General failure";
}

/**********************************************************************
*
*       Public data
*
**********************************************************************
*/

const SSH_SCP_SOURCE_FS_API SSH_SCP_SOURCE_FS_Win32 = {
  _SSH_SCP_SOURCE_FS_WIN32_Config,
  _SSH_SCP_SOURCE_FS_WIN32_Init,
  _SSH_SCP_SOURCE_FS_WIN32_FindFirst,
  _SSH_SCP_SOURCE_FS_WIN32_FindNext,
  _SSH_SCP_SOURCE_FS_WIN32_FindClose,
  _SSH_SCP_SOURCE_FS_WIN32_OpenFile,
  _SSH_SCP_SOURCE_FS_WIN32_ReadFile,
  _SSH_SCP_SOURCE_FS_WIN32_CloseFile,
  _SSH_SCP_SOURCE_FS_WIN32_DecodeStatus,
};

/*************************** End of file ***************************/
```

9.4.2.2 SSH-SCP-SINK for Windows

```c
/*********************************************************************
*                  (c) SEGGER Microcontroller GmbH                   *
*                      The Embedded Experts                          *
*                         www.segger.com                             *
**********************************************************************

----------------------- END-OF-HEADER ----------------------------

File      : SSH_SCP_SINK_FS_Win32.c
Purpose   : Adaptation of SSH SCP file system API for Windows.

*/

/*********************************************************************
*
*       #include Section
*
**********************************************************************
*/

#include "SSH.h"
#include <stdio.h>
#include <windows.h>

/*********************************************************************
*
*       Local data types
*
**********************************************************************
*/

typedef struct {
  FILE * pFile;
  char   acPath[SSH_SCP_CONFIG_PATH_MAX];
} SSH_SCP_FS_WIN32_CONTEXT;

/*********************************************************************
*
*       Static data
*
**********************************************************************
*/

static struct {
  char                     acRoot  [SSH_SCP_CONFIG_PATH_MAX];
  SSH_SCP_FS_WIN32_CONTEXT aContext[SSH_SCP_CONFIG_MAX_SESSIONS];
} SSH_SCP_FS_WIN32_Globals;

/*********************************************************************
*
*       Static code
*
**********************************************************************
*/


/*********************************************************************
*
*       _SSH_SCP_SINK_FS_WIN32_PathNorm()
*
*  Function description
*    Normalize path to Windows backslash directory delimiters.
*
*  Parameters
*    sPath - Pointer to zero-terminated path string.
*/
static void _SSH_SCP_SINK_FS_WIN32_PathNorm(char *sPath) {
  while (*sPath) {
    if (*sPath == '/') {
      *sPath = '\\';
    }
    ++sPath;
  }
```

```c
}

/**********************************************************************
*
*       _SSH_SCP_SINK_FS_WIN32_PathAddX()
*
*  Function description
*    Append path data to length-limited string.
*
*  Parameters
*    sName   - Pointer to zero-terminated string.
*    NameLen - Capacity of the zero-terminated string.
*    sText   - Pointer to text to append to zero-terminated string.
*    TextLen - Octet length of the text to append to zero-terminated string.
*
*  Return value
*    == 0                        - Appended successfully.
*    == SSH_ERROR_PATH_TOO_LONG - String overflow.
*/
static int _SSH_SCP_SINK_FS_WIN32_PathAddX(char *sName, unsigned NameLen, const char *sText, unsigned TextLe
  if (TextLen > 0 && sText[0] == '/' && sName[0] !
= 0 && sName[SSH_StrLen(sName)-1] == '/') {
    ++sText;
    --TextLen;
  }
  if (SSH_StrLen(sName) + TextLen + 1 >= NameLen) {
    return SSH_ERROR_PATH_TOO_LONG;
  }
  //
  sName = SSH_STRCHR(sName, '\0');
  SSH_MEMCPY(sName, sText, TextLen);
  sName[TextLen] = 0;
  return 0;
}

/**********************************************************************
*
*       _SSH_SCP_SINK_FS_WIN32_PathAdd()
*
*  Function description
*    Append path data to length-limited string.
*
*  Parameters
*    sName   - Pointer to zero-terminated string.
*    NameLen - Capacity of the zero-terminated string.
*    sText   - Pointer zero-terminated text to append to zero-terminated string.
*
*  Return value
*    == 0                        - Appended successfully.
*    == SSH_ERROR_PATH_TOO_LONG - String overflow.
*/
static int _SSH_SCP_SINK_FS_WIN32_PathAdd(char *sName, unsigned NameLen, const char *sText) {
  return _SSH_SCP_SINK_FS_WIN32_PathAddX(sName, NameLen, sText, SSH_StrLen(sText));
}

/**********************************************************************
*
*       _SSH_SCP_SINK_FS_WIN32_PathTrim()
*
*  Function description
*    Remove all characters from final '/' or '\' to end.
*
*  Parameters
*    sName - Pointer to zero-terminated string.
*/
static void _SSH_SCP_SINK_FS_WIN32_PathTrim(char *sName) {
  char *s;
  //
  s = SSH_STRRCHR(sName, '/');
  if (s == NULL) {
    s = SSH_STRRCHR(sName, '\\');
  }
  if (s != NULL && s != sName) {
    *s = 0;
  }
}
```

```
/**********************************************************************
*
*       _SSH_SCP_SINK_FS_WIN32_Config()
*
*  Function description
*    Implementation of Config API call.
*
*  Parameters
*    sRoot - Pointer to zero-terminated virtual SCP root directory.
*
*  Return value
*    >= 0 - Success.
*    <  0 - Error status.
*/
static int _SSH_SCP_SINK_FS_WIN32_Config(const char *sRoot) {
  SSH_SCP_FS_WIN32_Globals.acRoot[0] = 0;
  return _SSH_SCP_SINK_FS_WIN32_PathAdd(SSH_SCP_FS_WIN32_Globals.acRoot,
                                        sizeof(SSH_SCP_FS_WIN32_Globals.acRoot),
                                        sRoot);
}

/**********************************************************************
*
*       _SSH_SCP_SINK_FS_WIN32_Init()
*
*  Function description
*    Implementation of Init API call.
*
*  Parameters
*    Index   - Zero-based SCP session index.
*    pPath   - Pointer to root-relative path string for this transfer.
*    PathLen - Octet length of root-relative path string for this transfer.
*
*  Return value
*    >= 0 - Success.
*    <  0 - Error status.
*/
static int _SSH_SCP_SINK_FS_WIN32_Init(unsigned Index, const char *pPath, unsigned PathLen) {
  SSH_SCP_FS_WIN32_CONTEXT * pSelf;
  int                        Status;
  //
  SSH_ASSERT(Index < SSH_SCP_CONFIG_MAX_SESSIONS);
  //
  pSelf = &SSH_SCP_FS_WIN32_Globals.aContext[Index];
  pSelf->pFile     = NULL;
  pSelf->acPath[0] = 0;
  //
  Status = _SSH_SCP_SINK_FS_WIN32_PathAdd(pSelf->acPath,
                                          sizeof(pSelf->acPath),
                                          SSH_SCP_FS_WIN32_Globals.acRoot);
  if (Status >= 0) {
    Status = _SSH_SCP_SINK_FS_WIN32_PathAddX(pSelf->acPath,
                                             sizeof(pSelf->acPath),
                                             pPath,
                                             PathLen);
    if (pSelf->acPath[0] == 0) {
      SSH_STRCPY(pSelf->acPath, "/");
    }
    _SSH_SCP_SINK_FS_WIN32_PathNorm(pSelf->acPath);
  }
  //
  return Status;
}

/**********************************************************************
*
*       _SSH_SCP_SINK_FS_WIN32_CreateFile()
*
*  Function description
*    Implementation of Create File API call.
*
*  Parameters
*    Index  - Zero-based SCP session index.
*    Mode   - Unix file mode.
```

```
*    Len     - Length of file.
*    ModTime - Last modification time, Unix format, number of seconds
*              since 1 Jan 1970.
*    AccTime - Last access time, Unix format, number of seconds since
*              1 Jan 1970.
*    sName   - Zero-terminated full path name.
*
*  Return value
*    >= 0 - Success.
*    <  0 - Error status.
*/
static int _SSH_SCP_SINK_FS_WIN32_CreateFile(       unsigned   Index,
                                                    unsigned   Mode,
                                                    U32        Len,
                                                    U32        ModTime,
                                                    U32        AccTime,
                                             const char      * sName) {
  SSH_SCP_FS_WIN32_CONTEXT * pSelf;
  int                        Status;
  //
  SSH_USE_PARA(Len);
  SSH_USE_PARA(ModTime);
  SSH_USE_PARA(AccTime);
  //
  SSH_ASSERT(Index < SSH_SCP_CONFIG_MAX_SESSIONS);
  //
  pSelf = &SSH_SCP_FS_WIN32_Globals.aContext[Index];
  //
  Status = _SSH_SCP_SINK_FS_WIN32_PathAdd(pSelf->acPath, sizeof(pSelf->acPath), "/");
  if (Status >= 0) {
    Status = _SSH_SCP_SINK_FS_WIN32_PathAdd(pSelf->acPath, sizeof(pSelf->acPath), sName);
    if (Status >= 0) {
      //
      // Create file and, if no Posix owner write permission, make read only.
      //
      _SSH_SCP_SINK_FS_WIN32_PathNorm(pSelf->acPath);
      //
      pSelf->pFile = fopen(pSelf->acPath, "wb");
      if (pSelf->pFile == NULL) {
        Status = SSH_ERROR_CANT_CREATE_FILE;
      } else {
        if ((Mode & 0200) == 0) {
          (void)SetFileAttributesA(pSelf->acPath, FILE_ATTRIBUTE_READONLY);
        }
      }
      //
      _SSH_SCP_SINK_FS_WIN32_PathTrim(pSelf->acPath);
    }
  }
  //
  return Status;
}

/**********************************************************************
*
*       _SSH_SCP_SINK_FS_WIN32_WriteFile()
*
*  Function description
*    Implementation of Write File API call.
*
*  Parameters
*    Index   - Zero-based SCP session index.
*    pData   - Pointer to object to write.
*    DataLen - Octet length of the object to write.
*
*  Return value
*    >= 0 - Success.
*    <  0 - Error status.
*/
static int _SSH_SCP_SINK_FS_WIN32_WriteFile(unsigned Index, const U8 *pData, unsigned DataLen) {
  SSH_SCP_FS_WIN32_CONTEXT * pSelf;
  //
  pSelf = &SSH_SCP_FS_WIN32_Globals.aContext[Index];
  //
  (void)fwrite(pData, 1, DataLen, pSelf->pFile);
  if (ferror(pSelf->pFile)) {
```

```c
      return SSH_ERROR_CANT_WRITE_FILE;
    } else {
      return 0;
    }
}

/*********************************************************************
*
*       _SSH_SCP_SINK_FS_WIN32_CloseFile()
*
*  Function description
*    Implementation of Close File API call.
*
*  Parameters
*    Index - Zero-based SCP session index.
*
*  Return value
*    >= 0 - Success.
*    <  0 - Error status.
*/
static int _SSH_SCP_SINK_FS_WIN32_CloseFile(unsigned Index) {
  SSH_SCP_FS_WIN32_CONTEXT * pSelf;
  int                        Status;
  //
  SSH_ASSERT(Index < SSH_SCP_CONFIG_MAX_SESSIONS);
  //
  pSelf = &SSH_SCP_FS_WIN32_Globals.aContext[Index];
  //
  Status = 0;
  if (pSelf->pFile != NULL) {
    if (fclose(pSelf->pFile) < 0) {
      Status = SSH_ERROR_CANT_WRITE_FILE;
    }
    pSelf->pFile = NULL;
  }
  //
  return Status;
}

/*********************************************************************
*
*       _SSH_SCP_SINK_FS_WIN32_EnterFolder()
*
*  Function description
*    Implementation of Enter Folder API call.
*
*  Parameters
*    Index   - Zero-based SCP session index.
*    Mode    - Folder mode.
*    ModTime - Last modification time, Unix format, number of seconds
*              since 1 Jan 1970.  Zero indicates time is not set.
*    AccTime - Last access time, Unix format, number of seconds since
*              1 Jan 1970.  Zero indicates time is not set.
*    sName   - Zero-terminated folder name.
*
*  Return value
*    >= 0 - Success.
*    <  0 - Error status.
*/
static int _SSH_SCP_SINK_FS_WIN32_EnterFolder(      unsigned    Index,
                                                    unsigned    Mode,
                                                    U32         ModTime,
                                                    U32         AccTime,
                                              const char      * sName) {
  SSH_SCP_FS_WIN32_CONTEXT * pSelf;
  int                        Status;
  //
  SSH_USE_PARA(Mode);
  SSH_ASSERT(Index < SSH_SCP_CONFIG_MAX_SESSIONS);
  //
  pSelf = &SSH_SCP_FS_WIN32_Globals.aContext[Index];
  Status = _SSH_SCP_SINK_FS_WIN32_PathAdd(pSelf->acPath, sizeof(pSelf->acPath), sName);
  if (Status >= 0) {
    if (CreateDirectoryA(pSelf->acPath, NULL) == 0) {
      //
      // Create failed.
```

```
      //
      if (GetLastError() == ERROR_ALREADY_EXISTS) {
        Status = 0;
      } else {
        Status = SSH_ERROR_CANT_CREATE_FOLDER;
      }
    } else {
      //
      // Create succeeded.
      //
      Status = 0;
    }
    _SSH_SCP_SINK_FS_WIN32_PathTrim(pSelf->acPath);
  }
  //
  return Status;
}

/*********************************************************************
*
*       _SSH_SCP_SINK_FS_WIN32_ExitFolder()
*
*  Function description
*    Implementation of Exit Folder API call.
*
*  Parameters
*    Index - Zero-based SCP session index.
*
*  Return value
*    >= 0 - Success.
*    <  0 - Error status.
*/
static int _SSH_SCP_SINK_FS_WIN32_ExitFolder(unsigned Index) {
  SSH_ASSERT(Index < SSH_SCP_CONFIG_MAX_SESSIONS);
  //
  _SSH_SCP_SINK_FS_WIN32_PathTrim(SSH_SCP_FS_WIN32_Globals.aContext[Index].acPath);
  return 0;
}

/*********************************************************************
*
*       _SSH_SCP_SINK_FS_WIN32_DecodeStatus()
*
*  Function description
*    Decode error status.
*
*  Parameters
*    Status - Error status.
*
*  Return value
*    Pointer to zero-terminate string if error, or NULL if success.
*/
static const char * _SSH_SCP_SINK_FS_WIN32_DecodeStatus(int Status) {
  return Status == 0 ? 0 : "General error";
}

/*********************************************************************
*
*       Public data
*
*********************************************************************
*/

const SSH_SCP_SINK_FS_API SSH_SCP_SINK_FS_Win32 = {
  _SSH_SCP_SINK_FS_WIN32_Config,
  _SSH_SCP_SINK_FS_WIN32_Init,
  _SSH_SCP_SINK_FS_WIN32_CreateFile,
  _SSH_SCP_SINK_FS_WIN32_WriteFile,
  _SSH_SCP_SINK_FS_WIN32_CloseFile,
  _SSH_SCP_SINK_FS_WIN32_EnterFolder,
  _SSH_SCP_SINK_FS_WIN32_ExitFolder,
  _SSH_SCP_SINK_FS_WIN32_DecodeStatus,
};

/************************** End of file **************************/
```

9.4.2.3   SSH-SCP-SOURCE for emFile

```c
/**********************************************************************
*                (c) SEGGER Microcontroller GmbH & Co. KG            *
*                     The Embedded Experts                           *
*                        www.segger.com                              *
**********************************************************************


------------------------- END-OF-HEADER ----------------------------

File       : SSH_SCP_SOURCE_FS_FS.c
Purpose    : SSH-SCP source file system access for emFile.

*/

/**********************************************************************
*
*       #include Section
*
**********************************************************************
*/

#include "SSH.h"
#include "FS.h"

/**********************************************************************
*
*       Local types
*
**********************************************************************
*/

typedef struct {
  FS_FILE       * pFile;
  FS_FIND_DATA    FindData;
  int             FindActive;
  char            aPath[SSH_SCP_CONFIG_PATH_MAX];      // Folder to search.
  const char    * sPattern;                            // Pattern to match
  char            aFileName[SSH_SCP_CONFIG_PATH_MAX];  // Found file name.
} SSH_SCP_SOURCE_FS_FS_CONTEXT;

typedef struct {
  SSH_SCP_SOURCE_FS_FS_CONTEXT aContext [SSH_SCP_CONFIG_MAX_SESSIONS];
  char                         aRootPath[SSH_SCP_CONFIG_PATH_MAX];
} SSH_SCP_SOURCE_GLOBALS;

/**********************************************************************
*
*       Static data
*
**********************************************************************
*/

static SSH_SCP_SOURCE_GLOBALS _SSH_SCP_SOURCE_FS_FS_Globals;

/**********************************************************************
*
*       Static code
*
**********************************************************************
*/

/**********************************************************************
*
*       _Upper()
*
*  Function description
*    Convert character to uppercase.
*
*  Parameters
*    Ch - Character to convert.
*
*  Return value
*    uppercased character.
*/
```

```c
static int _Upper(int Ch) {
  if ('a' <= Ch && Ch <= 'z') {
    Ch = Ch - 'a' + 'A';
  }
  return Ch;
}

/**********************************************************************
*
*       _SSH_SCP_SOURCE_Match()
*
*  Function description
*    Match string against pattern.
*
*  Parameters
*    sPat - Zero-terminated pattern string.
*    sStr - Zero-terminated string to match.
*
*  Return value
*    0 -- No match.
*    1 -- Match.
*/
static int _SSH_SCP_SOURCE_Match(const char *sPat, const char *sStr) {
  const char   * s;
  const char   * p;
  int            Star;
  //
  Star = 0;
  //
Again:
  for (s = sStr, p = sPat; *s; ++s, ++p) {
    switch (*p) {
    case '?':
      break;
      //
    case '*':
      Star = 1;
      sStr = s;
      sPat = p;
      if (*++sPat == 0) {
        return 1;
      }
      goto Again;
      //
    default:
      if (_Upper(*s) != _Upper(*p)) {
        if (!Star) {
          return 0;
        }
        ++sStr;
        goto Again;
      }
      break;
    }
  }
  if (*p == '*') {
    ++p;
  }
  return *p == 0;
}

/**********************************************************************
*
*       _SSH_SCP_SOURCE_IsPathSep()
*
*  Function description
*    Is character a path separator?
*
*  Parameters
*    Ch - Character to test.
*
*  Return value
*    == 0 - Not a path separator.
*    != 0 - Is a path separator.
*/
static int _SSH_SCP_SOURCE_IsPathSep(int Ch) {
```

```c
  return Ch == '/' || Ch == '\\';
}

/**********************************************************************
*
*       _SSH_SCP_SOURCE_FS_FS_PathNorm()
*
*  Function description
*    Normalize path to emFile directory delimiters.
*
*  Parameters
*    sPath - Pointer to zero-terminated path string.
*/
static void _SSH_SCP_SOURCE_FS_FS_PathNorm(char *sPath) {
  while (*sPath) {
    if (_SSH_SCP_SOURCE_IsPathSep(*sPath)) {
      *sPath = FS_DIRECTORY_DELIMITER;
    }
    ++sPath;
  }
}

/**********************************************************************
*
*       _SSH_SCP_SOURCE_PathTrim()
*
*  Function description
*    Remove all characters from final '/' or '\' to end.
*
*  Parameters
*    sName - Pointer to zero-terminated string.
*/
static void _SSH_SCP_SOURCE_PathTrim(char *sName) {
  char *s;
  //
  s = SSH_STRRCHR(sName, '/');
  if (s == NULL) {
    s = SSH_STRRCHR(sName, '\\');
  }
  if (s != NULL && s != sName) {
    *s = 0;
  }
}

/**********************************************************************
*
*       _SSH_SCP_SOURCE_PathAdd()
*
*  Function description
*    Append path data to length-limited string.
*
*  Parameters
*    sName   - Pointer to zero-terminated string.
*    NameLen - Capacity of the zero-terminated string.
*    sText   - Zero-terminated text to append to zero-terminated string.
*
*  Return value
*    == 0                       - Appended successfully.
*    == SSH_ERROR_PATH_TOO_LONG - String overflow.
*/
static int _SSH_SCP_SOURCE_PathAdd(char *sName, unsigned NameLen, const char *sText) {
  if (_SSH_SCP_SOURCE_IsPathSep(sText[0]) && sName[0] !
= 0 && _SSH_SCP_SOURCE_IsPathSep(sName[SSH_STRLEN(sName)-1])) {
    ++sText;
  }
  if (SSH_STRLEN(sName) + SSH_STRLEN(sText) + 1 >= NameLen) {
    return SSH_ERROR_PATH_TOO_LONG;
  }
  //
  SSH_STRNCAT(sName, sText, NameLen);
  return 0;
}

/**********************************************************************
*
*       _SSH_SCP_SOURCE_FS_ConstructData()
```

```
*
*  Function description
*    Fill in file information structure.
*
*  Parameters
*    pContext  - Pointer to context.
*    pInfo     - Pointer to object that receives the file information.
*
*  Return value
*    >= 0                     - File information filled in successfully.
*    == SSH_ERROR_PATH_TOO_LONG - Full path name is too long.
*/
static int _SSH_SCP_SOURCE_FS_FS_ConstructData(SSH_SCP_SOURCE_FS_FS_CONTEXT * pContext,
                                               SSH_SCP_SOURCE_FILE_INFO     * pInfo) {
  int Status;
  //
  pInfo->Length   = pContext->FindData.FileSize;
  pInfo->IsFolder = (pContext->FindData.Attributes & FS_ATTR_DIRECTORY) != 0;
  pInfo->Mode     = (pContext->FindData.Attributes & FS_ATTR_READ_ONLY) ? 0444 : 0666;
  //
  pInfo->aPathName[0] = 0;
  Status = _SSH_SCP_SOURCE_PathAdd(pInfo->aPathName, sizeof(pInfo->aPathName), pContext-
>aPath);
  if (Status >= 0) {
    Status = _SSH_SCP_SOURCE_PathAdd(pInfo->aPathName, sizeof(pInfo->aPathName), "/");
    if (Status >= 0) {
      Status = _SSH_SCP_SOURCE_PathAdd(pInfo->aPathName, sizeof(pInfo->aPathName), pContext-
>FindData.sFileName);
      if (Status >= 0) {
        _SSH_SCP_SOURCE_FS_FS_PathNorm(pInfo->aPathName);
      }
    }
  }
  return Status;
}

/**********************************************************************
*
*       _SSH_SCP_SOURCE_FS_FS_Config()
*
*  Function description
*    Implementation of Config API call.
*
*  Parameters
*    sRoot - Pointer to zero-terminated virtual SCP root directory.
*
*  Return value
*    >= 0 - Success.
*    <  0 - Error status.
*/
static int _SSH_SCP_SOURCE_FS_FS_Config(const char *sRoot) {
  _SSH_SCP_SOURCE_FS_FS_Globals.aRootPath[0] = 0;
  return _SSH_SCP_SOURCE_PathAdd(_SSH_SCP_SOURCE_FS_FS_Globals.aRootPath,
                                 sizeof(_SSH_SCP_SOURCE_FS_FS_Globals.aRootPath),
                                 sRoot);
}

/**********************************************************************
*
*       _SSH_SCP_SOURCE_FS_FS_Init()
*
*  Function description
*    Implementation of Init API call.
*
*  Parameters
*    Index - Zero-based SCP session index.
*    sPath - Pointer to zero-terminated initial path from SCP command line.
*
*  Return value
*    >= 0 - Success.
*    <  0 - Error status.
*/
static int _SSH_SCP_SOURCE_FS_FS_Init(unsigned Index, const char *sPath) {
  SSH_SCP_SOURCE_FS_FS_CONTEXT * pContext;
  int                            Status;
  //
```

```c
    pContext = &_SSH_SCP_SOURCE_FS_FS_Globals.aContext[Index];
    //
    SSH_MEMSET(pContext, 0, sizeof(*pContext));
    pContext->FindActive = 0;
    //
    pContext->aPath[0] = 0;
    Status = _SSH_SCP_SOURCE_PathAdd(pContext->aPath, sizeof(pContext-
>aPath), _SSH_SCP_SOURCE_FS_FS_Globals.aRootPath);
    if (Status >= 0) {
      Status = _SSH_SCP_SOURCE_PathAdd(pContext->aPath, sizeof(pContext->aPath), "/");
      if (Status >= 0) {
        Status = _SSH_SCP_SOURCE_PathAdd(pContext->aPath, sizeof(pContext->aPath), sPath);
        if (Status >= 0 && pContext->aPath[SSH_STRLEN(pContext->aPath)-1] == '/') {
          Status = _SSH_SCP_SOURCE_PathAdd(pContext->aPath, sizeof(pContext->aPath), "*");
        }
        if (Status >= 0) {
          _SSH_SCP_SOURCE_FS_FS_PathNorm(pContext->aPath);
          _SSH_SCP_SOURCE_PathTrim(pContext->aPath);
          pContext->sPattern = SSH_STRCHR(pContext->aPath, 0) + 1;
        }
      }
    }
  }
  return 0;
}

/*********************************************************************
*
*       _SSH_SCP_SOURCE_FS_FS_FindFirst()
*
*  Function description
*    Implementation of Find First API call.
*
*  Parameters
*    Index - Zero-based SCP session index.
*    pInfo - Pointer to object that receives file information.
*
*  Return value
*    >= 0                      - Success.
*    == SSH_ERROR_NO_MORE_FILES - Folder enumeration complete (i.e. empty folder).
*    <  0                      - Error status.
*/
static int _SSH_SCP_SOURCE_FS_FS_FindFirst(unsigned Index, SSH_SCP_SOURCE_FILE_INFO *pInfo) {
  SSH_SCP_SOURCE_FS_FS_CONTEXT * pContext;
  int                            Status;
  //
  pContext = &_SSH_SCP_SOURCE_FS_FS_Globals.aContext[Index];
  //
  // Close down any open handle; this is just belts and braces.
  //
  if (pContext->FindActive) {
    FS_FindClose(&pContext->FindData);
    pContext->FindActive = 0;
  }
  //
  // FS_FindFirstFile() and FS_FindNextFile() produce opposite "success" and "no
  // files" results.  FS_FindFirstFile() returns 0 for "found" and 1 for "not found".
  // FS_FindNextFile() returns 1 for "found" and 0 for "not found / error".
  //
  // We normalize FindFirstFile() to results of FindNextFile(), i.e. 1 is "found"
  // and 0 as "not found".
  //
  Status = FS_FindFirstFile(&pContext->FindData, pContext->aPath, &pContext-
>aFileName[0], sizeof(pContext->aFileName));
  if (Status == 0 || Status == 1) {
    Status = 1-Status;
  }
  while (Status == 1 && (!_SSH_SCP_SOURCE_Match(pContext->sPattern, pContext-
>FindData.sFileName) ||
                         (pContext-
>FindData.Attributes & (FS_ATTR_HIDDEN | FS_ATTR_SYSTEM | FS_ATTR_DIRECTORY)) != 0)) {
    Status = FS_FindNextFile(&pContext->FindData);
  }
  if (Status == 0) {
    Status = SSH_ERROR_NO_MORE_FILES;
  } else if (Status < 0) {
    Status = SSH_ERROR_CANT_READ_FOLDER;
```

```c
    } else {
      Status = _SSH_SCP_SOURCE_FS_ConstructData(pContext, pInfo);
    }
    if (Status >= 0) {
      pContext->FindActive = 1;
    }
    return Status;
}

/************************************************************************
*
*       _SSH_SCP_SOURCE_FS_FS_FindNext()
*
*  Function description
*    Implementation of Find Next API call.
*
*  Parameters
*    Index - Zero-based SCP session index.
*    pInfo - Pointer to object that receives file information.
*
*  Return value
*    >= 0                       - Success.
*    == SSH_ERROR_NO_MORE_FILES - Folder enumeration complete, no additional information.
*    <  0                       - Error status.
*/
static int _SSH_SCP_SOURCE_FS_FS_FindNext(unsigned Index, SSH_SCP_SOURCE_FILE_INFO *pInfo) {
  SSH_SCP_SOURCE_FS_FS_CONTEXT * pContext;
  int                            Status;
  //
  pContext = &_SSH_SCP_SOURCE_FS_FS_Globals.aContext[Index];
  //
  if (!pContext->FindActive) {
    return SSH_ERROR_NO_MORE_FILES;
  }
  //
  Status = FS_FindNextFile(&pContext->FindData);
  while (Status == 1 && !_SSH_SCP_SOURCE_Match(pContext->sPattern, pContext->aFileName)) {
    Status = FS_FindNextFile(&pContext->FindData);
  }
  if (Status == 0) {
    FS_FindClose(&pContext->FindData);
    pContext->FindActive = 0;
    return SSH_ERROR_NO_MORE_FILES;
  } else {
    return _SSH_SCP_SOURCE_FS_ConstructData(pContext, pInfo);
  }
}

/************************************************************************
*
*       _SSH_SCP_SOURCE_FS_FS_FindClose()
*
*  Function description
*    Implementation of Find Close API call.
*
*  Parameters
*    Index - Zero-based SCP session index.
*
*  Return value
*    >= 0 - Success.
*    <  0 - Error status.
*/
static int _SSH_SCP_SOURCE_FS_FS_FindClose(unsigned Index) {
  SSH_SCP_SOURCE_FS_FS_CONTEXT *pContext;
  //
  pContext = &_SSH_SCP_SOURCE_FS_FS_Globals.aContext[Index];
  //
  if (pContext->FindActive) {
    FS_FindClose(&pContext->FindData);
    pContext->FindActive = 0;
  }
  //
  return 0;
}

/************************************************************************
```

```
*
*         _SSH_SCP_SOURCE_FS_FS_OpenFile()
*
*  Function description
*    Implementation of Open File API call.
*
*  Parameters
*    Index - Zero-based SCP session index.
*    sName - Pointer to zero-terminated full path name to file.
*            Note that this is writable to allow in-place modification
*            of path separators.
*
*  Return value
*    >= 0 - Success.
*    <  0 - Error status.
*/
static int _SSH_SCP_SOURCE_FS_FS_OpenFile(unsigned Index, char *sName) {
  SSH_SCP_SOURCE_FS_FS_CONTEXT *pContext;
  //
  pContext = &_SSH_SCP_SOURCE_FS_FS_Globals.aContext[Index];
  //
  if (pContext->pFile != NULL) {
    FS_FClose(pContext->pFile);
  }
  //
  // Close down any open file; this is just belts and braces.
  //
  pContext->pFile = FS_FOpen(sName, "rb");
  if (pContext->pFile == NULL) {
    return SSH_ERROR_CANT_READ_FILE;
  } else {
    return 0;
  }
}

/**********************************************************************
*
*         _SSH_SCP_SOURCE_FS_FS_ReadFile()
*
*  Function description
*    Implementation of Read File API call.
*
*  Parameters
*    Index   - Zero-based SCP session index.
*    pData   - Pointer to object that receives the file data.
*    DataLen - Octet length of the object that receives the file data.
*
*  Return value
*    >  0 - Number of octets read from file (with more data to read from file).
*    == 0 - End of file indication (with no more data to read from file).
*    <  0 - Error status.
*/
static int _SSH_SCP_SOURCE_FS_FS_ReadFile(unsigned Index, U8 *pData, unsigned DataLen) {
  SSH_SCP_SOURCE_FS_FS_CONTEXT * pContext;
  unsigned                       L;
  //
  pContext = &_SSH_SCP_SOURCE_FS_FS_Globals.aContext[Index];
  //
  if (pContext->pFile == NULL) {
    return SSH_ERROR_CANT_READ_FILE;
  } else {
    L = FS_FRead(pData, 1, DataLen, pContext->pFile);
    if (FS_FError(pContext->pFile) != FS_ERRCODE_OK && FS_FError(pContext->pFile) !
= FS_ERRCODE_EOF) {
      return SSH_ERROR_CANT_READ_FILE;
    } else {
      return L;
    }
  }
}

/**********************************************************************
*
*         _SSH_SCP_SOURCE_FS_FS_CloseFile()
*
*  Function description
```

```
*      Implementation of Close File API call.
*
*  Parameters
*     Index - Zero-based SCP session index.
*
*  Return value
*     >= 0 - Success.
*     <  0 - Error status.
*/
static int _SSH_SCP_SOURCE_FS_FS_CloseFile(unsigned Index) {
  SSH_SCP_SOURCE_FS_FS_CONTEXT * pContext;
  //
  pContext = &_SSH_SCP_SOURCE_FS_FS_Globals.aContext[Index];
  //
  if (pContext->pFile) {
    FS_FClose(pContext->pFile);
  }
  pContext->pFile = NULL;
  //
  return 0;
}

/**********************************************************************
*
*       _SSH_SCP_SOURCE_FS_FS_DecodeStatus()
*
*  Function description
*     Decode error status.
*
*  Parameters
*     Status - Error status.
*
*  Return value
*     Pointer to zero-terminate string if error, or NULL if success.
*/
static const char * _SSH_SCP_SOURCE_FS_FS_DecodeStatus(int Status) {
  return Status == 0 ? NULL : "General failure";
}

/**********************************************************************
*
*       Public data
*
***********************************************************************
*/

const SSH_SCP_SOURCE_FS_API SSH_SCP_SOURCE_FS_FS = {
  _SSH_SCP_SOURCE_FS_FS_Config,
  _SSH_SCP_SOURCE_FS_FS_Init,
  _SSH_SCP_SOURCE_FS_FS_FindFirst,
  _SSH_SCP_SOURCE_FS_FS_FindNext,
  _SSH_SCP_SOURCE_FS_FS_FindClose,
  _SSH_SCP_SOURCE_FS_FS_OpenFile,
  _SSH_SCP_SOURCE_FS_FS_ReadFile,
  _SSH_SCP_SOURCE_FS_FS_CloseFile,
  _SSH_SCP_SOURCE_FS_FS_DecodeStatus,
};

/************************** End of file ***************************/
```

9.4.2.4   SSH-SCP-SINK for emFile

```
/**********************************************************************
*                   (c) SEGGER Microcontroller GmbH                  *
*                       The Embedded Experts                         *
*                          www.segger.com                            *
**********************************************************************

----------------------- END-OF-HEADER ----------------------------

File        : SSH_SCP_SINK_FS_FS.c
Purpose     : Adaptation of SSH SCP file system API to emFile.

*/

/**********************************************************************
*
*       #include Section
*
**********************************************************************
*/

#include "SSH.h"
#include "FS.h"
#include <stdlib.h>

/**********************************************************************
*
*       Defines, fixed
*
**********************************************************************
*/

#define ISLEAP(y)      (((((y) % 4) == 0 && ((y) % 100) != 0) || ((y) % 400) == 0)
#define EPOCH_YEAR     1970
#define SECSPERMIN     60L
#define MINSPERHOUR    60L
#define HOURSPERDAY    24L
#define SECSPERHOUR    (SECSPERMIN * MINSPERHOUR)
#define SECSPERDAY     (SECSPERHOUR * HOURSPERDAY)

/**********************************************************************
*
*       Local data types
*
**********************************************************************
*/

typedef struct {
  FS_FILE * pFile;
  char      acPath[SSH_SCP_CONFIG_PATH_MAX];
} SSH_SCP_FS_FS_CONTEXT;

/**********************************************************************
*
*       Static const data
*
**********************************************************************
*/

static const U8 _SSH_SCP_SINK_FS_FS_aMonLengths[2][12] = {
  {31, 28, 31, 30, 31, 30, 31, 31, 30, 31, 30, 31},
  {31, 29, 31, 30, 31, 30, 31, 31, 30, 31, 30, 31}
};

static const U16 _SSH_SCP_SINK_FS_FS_aYearLengths[2] = {
  365,
  366
};

/**********************************************************************
*
*       Static data
*
**********************************************************************
```

```c
*/

static struct {
  char                 acRoot  [SSH_SCP_CONFIG_PATH_MAX];
  SSH_SCP_FS_FS_CONTEXT aContext[SSH_SCP_CONFIG_MAX_SESSIONS];
} SSH_SCP_FS_FS_Globals;

/*********************************************************************
*
*       Static code
*
**********************************************************************
*/

/*********************************************************************
*
*       _SSH_SCP_SINK_FS_FS_PathNorm()
*
*  Function description
*    Normalize path to emFile directory delimiters.
*
*  Parameters
*    sPath - Pointer to zero-terminated path string.
*/
static void _SSH_SCP_SINK_FS_FS_PathNorm(char *sPath) {
  while (*sPath) {
    if (*sPath == '/' || *sPath == '\\') {
      *sPath = FS_DIRECTORY_DELIMITER;
    }
    ++sPath;
  }
}

/*********************************************************************
*
*       _SSH_SCP_SINK_FS_FS_PathAddX()
*
*  Function description
*    Append path data to length-limited string.
*
*  Parameters
*    sName   - Pointer to zero-terminated string.
*    NameLen - Capacity of the zero-terminated string.
*    sText   - Pointer to text to append to zero-terminated string.
*    TextLen - Octet length of the text to append to zero-terminated string.
*
*  Return value
*    == 0                      - Appended successfully.
*    == SSH_ERROR_PATH_TOO_LONG - String overflow.
*/
static int _SSH_SCP_SINK_FS_FS_PathAddX(char *sName, unsigned NameLen, const char *sText, unsigned TextLen)
  if (TextLen > 0 && sText[0] == '/' && sName[0] !
= 0 && sName[SSH_STRLEN(sName)-1] == '/') {
    ++sText;
    --TextLen;
  }
  if (SSH_STRLEN(sName) + TextLen + 1 >= NameLen) {
    return SSH_ERROR_PATH_TOO_LONG;
  }
  //
  sName = SSH_STRCHR(sName, '\0');
  SSH_MEMCPY(sName, sText, TextLen);
  sName[TextLen] = 0;
  return 0;
}

/*********************************************************************
*
*       _SSH_SCP_SINK_FS_FS_PathAdd()
*
*  Function description
*    Append path data to length-limited string.
*
*  Parameters
*    sName   - Pointer to zero-terminated string.
*    NameLen - Capacity of the zero-terminated string.
```

```
*    sText   - Zero-terminated text to append to zero-terminated string.
*
*  Return value
*    == 0                      - Appended successfully.
*    == SSH_ERROR_PATH_TOO_LONG - String overflow.
*/
static int _SSH_SCP_SINK_FS_FS_PathAdd(char *sName, unsigned NameLen, const char *sText) {
  return _SSH_SCP_SINK_FS_FS_PathAddX(sName, NameLen, sText, SSH_STRLEN(sText));
}

/*********************************************************************
*
*       _SSH_SCP_SINK_FS_FS_PathTrim()
*
*  Function description
*    Remove all characters from final '/' or '\' to end.
*
*  Parameters
*    sName - Pointer to zero-terminated string.
*/
static void _SSH_SCP_SINK_FS_FS_PathTrim(char *sName) {
  char *s;
  //
  s = SSH_STRRCHR(sName, '/');
  if (s == NULL) {
    s = SSH_STRRCHR(sName, '\\');
  }
  if (s != NULL && s != sName) {
    *s = 0;
  }
}


/*********************************************************************
*
*       _SSH_SCP_SINK_FS_FS_Config()
*
*  Function description
*    Implementation of Config API call.
*
*  Parameters
*    sRoot - Pointer to zero-terminated virtual SCP root directory.
*
*  Return value
*    >= 0 - Success.
*    <  0 - Error status.
*/
static int _SSH_SCP_SINK_FS_FS_Config(const char *sRoot) {
  SSH_SCP_FS_FS_Globals.acRoot[0] = 0;
  return _SSH_SCP_SINK_FS_FS_PathAdd(SSH_SCP_FS_FS_Globals.acRoot,
                                     sizeof(SSH_SCP_FS_FS_Globals.acRoot),
                                     sRoot);
}

/*********************************************************************
*
*       _SSH_SCP_SINK_FS_FS_Init()
*
*  Function description
*    Implementation of Init API call.
*
*  Parameters
*    Index   - Zero-based SCP session index.
*    pPath   - Pointer to root-relative path string for this transfer.
*    PathLen - Octet length of root-relative path string for this transfer.
*
*  Return value
*    >= 0 - Success.
*    <  0 - Error status.
*/
static int _SSH_SCP_SINK_FS_FS_Init(unsigned Index, const char *sPath, unsigned PathLen) {
  SSH_SCP_FS_FS_CONTEXT * pSelf;
  int                    Status;
  //
  SSH_ASSERT(Index < SSH_SCP_CONFIG_MAX_SESSIONS);
  //
```

```
    pSelf = &SSH_SCP_FS_FS_Globals.aContext[Index];
    pSelf->pFile    = NULL;
    pSelf->acPath[0] = 0;
    //
    Status = _SSH_SCP_SINK_FS_FS_PathAdd(pSelf->acPath,
                                         sizeof(pSelf->acPath),
                                         SSH_SCP_FS_FS_Globals.acRoot);
    if (Status >= 0) {
      Status = _SSH_SCP_SINK_FS_FS_PathAddX(pSelf->acPath,
                                            sizeof(pSelf->acPath),
                                            sPath,
                                            PathLen);
      if (pSelf->acPath[0] == 0) {
        SSH_STRCPY(pSelf->acPath, "/");
      }
      _SSH_SCP_SINK_FS_FS_PathNorm(pSelf->acPath);
    }
    //
    return Status;
}

/**********************************************************************
*
*       _SSH_SCP_SINK_FS_FS_CvtStamp()
*
*  Function description
*    Convert Unix time to emFile time.
*
*  Parameters
*    UnixTime - Unix format timestamp, number of seconds since
*               1 Jan 1970.
*
*  Return value
*    Timestamp in emFile format.
*/
static U32 _SSH_SCP_SINK_FS_FS_CvtStamp(U32 UnixTime) {
  int        IsLeap;
  const U8 * pMonthLen;
  unsigned   Year;
  unsigned   Day;
  unsigned   Month;
  unsigned   Hour;
  unsigned   Minute;
  unsigned   Second;
  //
  Day    = UnixTime / SECSPERDAY;   UnixTime %= SECSPERDAY;
  Hour   = UnixTime / SECSPERHOUR;  UnixTime %= SECSPERHOUR;
  Minute = UnixTime / SECSPERMIN;   UnixTime %= SECSPERMIN;
  Second = UnixTime;
  //
  Year = EPOCH_YEAR;
  for (;;) {
    IsLeap = ISLEAP(Year);
    if (Day < _SSH_SCP_SINK_FS_FS_aYearLengths[IsLeap]) {
      break;
    }
    Day -= _SSH_SCP_SINK_FS_FS_aYearLengths[IsLeap];
    ++Year;
  }
  //
  pMonthLen = _SSH_SCP_SINK_FS_FS_aMonLengths[IsLeap];
  for (Month = 0; Day >= pMonthLen[Month]; ++Month) {
    Day -= pMonthLen[Month];
  }
  //
  // Convert to conventional day-of-month and month-of-year.
  //
  ++Day;
  ++Month;
  //
  // Now pack components.
  //
  if (Year < 1980) {
    return 0;
  } else {
    return ((Second / 2)  <<  0) +
```

```
                 (Minute          <<  5) +
                 (Hour            << 11) +
                 (Day             << 16) +
                 (Month           << 21) +
                 ((Year - 1980) << 25);
  }
}

/**********************************************************************
*
*       _SSH_SCP_SINK_FS_FS_CreateFile()
*
*  Function description
*    Implementation of Create File API call.
*
*  Parameters
*    Index   - Zero-based SCP session index.
*    Mode    - Unix file mode.
*    Len     - Length of file.
*    ModTime - Last modification time, Unix format, number of seconds
*              since 1 Jan 1970.  Zero indicates that time was not set.
*    AccTime - Last access time, Unix format, number of seconds since
*              1 Jan 1970.  Zero indicates that time was not set.
*    sName   - Zero-terminated full path name.
*
*  Return value
*    >= 0 - Success.
*    <  0 - Error status.
*/
static int _SSH_SCP_SINK_FS_FS_CreateFile(       unsigned Index,
                                                 unsigned Mode,
                                                 U32      Len,
                                                 U32      ModTime,
                                                 U32      AccTime,
                                           const char *   sName) {
  SSH_SCP_FS_FS_CONTEXT * pSelf;
  int                     Status;
  //
  SSH_ASSERT(Index < SSH_SCP_CONFIG_MAX_SESSIONS);
  //
  pSelf = &SSH_SCP_FS_FS_Globals.aContext[Index];
  //
  Status = _SSH_SCP_SINK_FS_FS_PathAdd(pSelf->acPath, sizeof(pSelf->acPath), "/");
  if (Status >= 0) {
    Status = _SSH_SCP_SINK_FS_FS_PathAdd(pSelf->acPath, sizeof(pSelf->acPath), sName);
    if (Status >= 0) {
      //
      // Create file.
      //
      _SSH_SCP_SINK_FS_FS_PathNorm(pSelf->acPath);
      Status = FS_FOpenEx(pSelf->acPath, "wb", &pSelf->pFile);
      if (Status >= 0) {
        //
        // Preallocate file data.
        //
        FS_FSeek(pSelf->pFile, Len, FS_SEEK_SET);
        Status = FS_SetEndOfFile(pSelf->pFile);
        FS_FSeek(pSelf->pFile, 0, FS_SEEK_SET);
        //
        //
        // If no Posix owner write permission, make read only.
        //
        if ((Mode & 0200) == 0) {
          (void)FS_ModifyFileAttributes(pSelf->acPath, FS_ATTR_READ_ONLY, 0);
        }
        //
        // If timestamps given, set them.
        //
        if (ModTime != 0) {
          FS_SetFileTimeEx(pSelf-
>acPath, _SSH_SCP_SINK_FS_FS_CvtStamp(ModTime), FS_FILETIME_CREATE);
          FS_SetFileTimeEx(pSelf-
>acPath, _SSH_SCP_SINK_FS_FS_CvtStamp(ModTime), FS_FILETIME_MODIFY);
        }
        if (AccTime != 0) {
```

```
            FS_SetFileTimeEx(pSelf-
>acPath, _SSH_SCP_SINK_FS_FS_CvtStamp(AccTime), FS_FILETIME_ACCESS);
          }
        }
        //
        _SSH_SCP_SINK_FS_FS_PathTrim(pSelf->acPath);
    }
  }
  //
  return Status;
}

/*********************************************************************
*
*       _SSH_SCP_SINK_FS_FS_WriteFile()
*
*  Function description
*    Implementation of Write File API call.
*
*  Parameters
*    Index   - Zero-based SCP session index.
*    pData   - Pointer to object to write.
*    DataLen - Octet length of the object to write.
*
*  Return value
*    >= 0 - Success.
*    <  0 - Error status.
*/
static int _SSH_SCP_SINK_FS_FS_WriteFile(unsigned Index, const U8 *pData, unsigned DataLen) {
  SSH_SCP_FS_FS_CONTEXT * pSelf;
  //
  pSelf = &SSH_SCP_FS_FS_Globals.aContext[Index];
  //
  return FS_FWrite(pData, 1, DataLen, pSelf->pFile);
}

/*********************************************************************
*
*       _SSH_SCP_SINK_FS_FS_CloseFile()
*
*  Function description
*    Implementation of Close File API call.
*
*  Parameters
*    Index - Zero-based SCP session index.
*
*  Return value
*    >= 0 - Success.
*    <  0 - Error status.
*/
static int _SSH_SCP_SINK_FS_FS_CloseFile(unsigned Index) {
  SSH_SCP_FS_FS_CONTEXT * pSelf;
  int                     Status;
  //
  SSH_ASSERT(Index < SSH_SCP_CONFIG_MAX_SESSIONS);
  //
  pSelf = &SSH_SCP_FS_FS_Globals.aContext[Index];
  if (pSelf->pFile != NULL) {
    Status = FS_FClose(pSelf->pFile);
    pSelf->pFile = NULL;
  } else {
    Status = 0;
  }
  //
  return Status;
}

/*********************************************************************
*
*       _SSH_SCP_SINK_FS_FS_EnterFolder()
*
*  Function description
*    Implementation of Enter Folder API call.
*
*  Parameters
*    Index   - Zero-based SCP session index.
```

```c
*     Mode    - Folder mode.
*     ModTime - Last modification time, Unix format, number of seconds
*               since 1 Jan 1970.  Zero indicates that time was not set.
*     AccTime - Last access time, Unix format, number of seconds since
*               1 Jan 1970.  Zero indicates that time was not set.
*     sName   - Zero-terminated folder name.
*
*  Return value
*     >= 0 - Success.
*     <  0 - Error status.
*/
static int _SSH_SCP_SINK_FS_FS_EnterFolder(      unsigned    Index,
                                                 unsigned    Mode,
                                                 U32         ModTime,
                                                 U32         AccTime,
                                            const char     * sName) {
  SSH_SCP_FS_FS_CONTEXT * pSelf;
  int                     Status;
  //
  SSH_USE_PARA(Mode);
  SSH_ASSERT(Index < SSH_SCP_CONFIG_MAX_SESSIONS);
  //
  pSelf = &SSH_SCP_FS_FS_Globals.aContext[Index];
  Status = _SSH_SCP_SINK_FS_FS_PathAdd(pSelf->acPath, sizeof(pSelf->acPath), "/");
  if (Status >= 0) {
    Status = _SSH_SCP_SINK_FS_FS_PathAdd(pSelf->acPath, sizeof(pSelf->acPath), sName);
    if (Status >= 0) {
      _SSH_SCP_SINK_FS_FS_PathNorm(pSelf->acPath);
      Status = FS_CreateDir(pSelf->acPath);
      if (Status == 1) {
        Status = 0;
        //
        // If timestamps given, set them.
        //
        if (ModTime != 0) {
          FS_SetFileTimeEx(pSelf-
>acPath, _SSH_SCP_SINK_FS_FS_CvtStamp(ModTime), FS_FILETIME_CREATE);
          FS_SetFileTimeEx(pSelf-
>acPath, _SSH_SCP_SINK_FS_FS_CvtStamp(ModTime), FS_FILETIME_MODIFY);
        }
        if (AccTime != 0) {
          FS_SetFileTimeEx(pSelf-
>acPath, _SSH_SCP_SINK_FS_FS_CvtStamp(AccTime), FS_FILETIME_ACCESS);
        }
      }
    }
  }
  //
  return Status;
}

/**********************************************************************
*
*       _SSH_SCP_SINK_FS_FS_ExitFolder()
*
*  Function description
*    Implementation of Exit Folder API call.
*
*  Parameters
*    Index - Zero-based SCP session index.
*
*  Return value
*     >= 0 - Success.
*     <  0 - Error status.
*/
static int _SSH_SCP_SINK_FS_FS_ExitFolder(unsigned Index) {
  SSH_ASSERT(Index < SSH_SCP_CONFIG_MAX_SESSIONS);
  //
  _SSH_SCP_SINK_FS_FS_PathTrim(SSH_SCP_FS_FS_Globals.aContext[Index].acPath);
  return 0;
}

/**********************************************************************
*
*       _SSH_SCP_SINK_FS_FS_DecodeStatus()
*
```

```
*  Function description
*    Decode error status.
*
*  Parameters
*    Status - Error status.
*
*  Return value
*    Pointer to zero-terminate string if error, or NULL if success.
*/
static const char * _SSH_SCP_SINK_FS_FS_DecodeStatus(int Status) {
  return Status == 0 ? 0 : FS_ErrorNo2Text(Status);
}

/**********************************************************************
*
*       Public data
*
**********************************************************************
*/

const SSH_SCP_SINK_FS_API SSH_SCP_SINK_FS_FS = {
  _SSH_SCP_SINK_FS_FS_Config,
  _SSH_SCP_SINK_FS_FS_Init,
  _SSH_SCP_SINK_FS_FS_CreateFile,
  _SSH_SCP_SINK_FS_FS_WriteFile,
  _SSH_SCP_SINK_FS_FS_CloseFile,
  _SSH_SCP_SINK_FS_FS_EnterFolder,
  _SSH_SCP_SINK_FS_FS_ExitFolder,
  _SSH_SCP_SINK_FS_FS_DecodeStatus,
};

/************************** End of file ***************************/
```

Chapter 10

Resource usage

This chapter covers the resource usage of emSSH. It contains information about the memory requirements in typical systems, which can be used to obtain sufficient estimates for most target systems.

10.1   Memory footprint

emSSH is designed to cater for many different embedded design requirements, from constrained microcontrollers to high performance microprocessors. Some features might be excluded from a build in order to construct a highly compact, minimal system. Note that the values are only valid for the given configuration.

10.1.1   Target system configuration

The following table shows the hardware and the toolchain details of a typical emSSH target system:

Detail	Description
CPU	Cortex-M4
Tool chain	SEGGER Embedded Studio with Clang version 3.7
Model	Thumb-2 instructions
Compiler options	Highest size optimization

10.1.2   ROM use

The following table shows the ROM requirement for each of emSSH's components:

Component	Size (approximate)
Public key algorithms	
DSA	0.5 KB
ECDSA	0.4 KB
RSA-PKCS1	0.5 KB
Hash and MAC functions	
SHA-1	0.5 KB
SHA-256 (including SHA-224)	0.9 KB
SHA-512 (including SHA-384)	2.3 KB
MD5	0.8 KB
HMAC-SHA1	0.2 KB
HMAC-SHA256	0.2 KB
HMAC-SHA384	0.2 KB
Cipher functions	
DES and 3DES	3.1 KB
AES	3.4 KB
AES-GCM (requires AES)	0.5 KB
Protocol support	
SSH core	10.5 KB
Shared supporting code	
MPI for RSA and ECC support	4.5 KB
Curve storage (for all curves)	4.4 KB
Curve arithmetic (requires MPI)	2.3 KB
Memory management	0.3 KB

10.1.3   RAM use

emSSH's RAM use can be partitioned as follows:

- *Static data requirement* — a fixed overhead incurred by using emSSH.
- *State and key material* — a variable overhead per connection that stores connection state and any key material required by the connection.
- *Public key memory* — a variable overhead for carrying out public key algorithms when negotiating a connection.
- *Protocol memory* — memory required to store protocol packets before encryption or decryption.

10.1.3.1   Static overhead

emSSH requires approximately 0.5 KB of RAM as a fixed overhead to manage its operation.

10.1.3.2   Connection state and key material

RAM is required to store the state of each active SSH connection. Part of this requirement is a variable amount of state that depends upon the cipher suite negotiated between the peers.

- Each SSH session object requires approximately 3 KB bytes to store its state, excluding negotiated contexts.
- Additional memory for cipher suite state varies with alogorithms negotiated, but typically about 500 bytes is used.

10.1.3.3   Key exchange and public key algorithms

Temporary memory is required to run appropriate public key algorithms when keys are exchanged. The amount of memory requires depends upon the public key algorithm and the key sizes. However, typically P-256 curves and 2048-bit RSA public keys require approximately 5 KB of memory to run.

10.1.3.4   Protocol memory

The number of bytes required per connection depends upon the packet size delivered from a server to a client, but this is under control of the client when and configured with `SSH_SESSION_ConfBuffers`. The SSH specifications say that an implementation should be capable of handling records of to 32 KB in size, but the record size is completely under control of the application and communicated to the peer; it can be reduced to the bare minimum required, but reducing it to very small values will cause the protocol to fail.

Chapter 11

Compatibility

The following sections detail the capabilities of some common clients applications and libraries that can connect with emSSH server. If you are using one of these clients, you can determine, in advance, which capabilities you would like to enable in emSSH.

- TeraTerm 4.84
- Putty 0.68
- OpenSSH 6.6.1p1
- Tectia Client 6.4.12.353
- libssh 0.7.0
- libssh2 1.7.0
- emSSH 2.52.0

The following tables make no distinction between what is enabled by default and what can be supported: it only describes what the client or library can be configured to provide.

11.1   Bugs in SSH clients

This section lists know interoperability issues, or bugs, in SSH clients.

11.1.1   Tera Term

Tera Term versions 4.84 and 4.90 are known to incorrectly implement the `blowfish-ctr` encryption algorithm, although `blowfish-cbc` is correctly implemented. Tera Term incorrectly uses a 128-bit key for SDCTR mode rather than the 256-bit key mandated by RFC 4344. PuTTY, the only other common client to implement `blowfish-ctr`, correctly uses a 256-bit key.

Confirmation of the 128-bit key issue is easily verified by altering the emSSH source code to reduce the key size in the definition of `SSH_ENCRYPTION_ALGORITHM_BLOWFISH_CTR`, after which Tera Term will connect to emSSH.

11.2   Key exchange algorithms

Method	TeraTerm	PuTTY	Dropbear	Tectia	OpenSSH	libssh	libssh2	emSSH
`rsa1024-sha1`		●						●
`rsa2048-sha256`		●						●
`diffie-hellman-group1-sha1`	●	●	●	●	●	●	●	●
`diffie-hellman-group14-sha1`	●	●	●	●	●	●	●	●
`diffie-hellman-group14-sha256`	●				●			●
`diffie-hellman-group15-sha512`	●				●			●
`diffie-hellman-group16-sha512`	●				●			●
`diffie-hellman-group17-sha512`	●				●			●
`diffie-hellman-group18-sha512`	●				●			●
`diffie-hellman-group-exchange-sha1`	●	●		●	●		●	●
`diffie-hellman-group-exchange-sha256`	●	●		●	●		●	●
`ecdh-sha2-nistp256`	●	●	●		●	●		●
`ecdh-sha2-nistp384`	●	●	●		●			●
`ecdh-sha2-nistp521`	●	●	●		●			●
`curve25519-sha256`		●	●					●
`curve448-sha512`								●
SSH Communications Security Extensions								
`diffie-hellman-group14-sha224@ssh.com`				●				●
`diffie-hellman-group14-sha256@ssh.com`				●				●
`diffie-hellman-group15-sha256@ssh.com`				●				●
`diffie-hellman-group15-sha384@ssh.com`				●				●
`diffie-hellman-group16-sha384@ssh.com`				●				●
`diffie-hellman-group16-sha512@ssh.com`				●				●
`diffie-hellman-group18-sha512@ssh.com`				●				●
`diffie-hellman-group-exchange-sha224@ssh.com`				●				●
`diffie-hellman-group-exchange-sha384@ssh.com`				●				●
`diffie-hellman-group-exchange-sha512@ssh.com`				●				●
libssh Extensions								
`curve25519-sha256@libssh.org`			●		●	●		●

11.3   Public key algorithms

Method	TeraTerm	PuTTY	Dropbear	Tectia	OpenSSH	libssh	libssh2	emSSH
`ssh-dss`	●	●	●	●	●	●	●	●
`ssh-rsa`	●	●	●	●	●	●	●	●
`ecdsa-sha2-nistp521`	●		●		●	●		●
`ecdsa-sha2-nistp384`	●		●		●	●		●
`ecdsa-sha2-nistp256`	●		●		●	●		●
`ssh-ed25519`	●				●	●		●
Proposed standard								
`rsa-sha2-256`					●			●
`rsa-sha2-512`					●			●
OpenSSH Extensions								
`ecdsa-sha2-nistp521-cert-v01@openssh.com`					●			
`ecdsa-sha2-nistp384-cert-v01@openssh.com`					●			
`ecdsa-sha2-nistp256-cert-v01@openssh.com`					●			
`ssh-ed25519-cert-v01@openssh.com`					●			
`ssh-rsa-cert-v01@openssh.com`					●			
`ssh-rsa-cert-v00@openssh.com`					●			
`ssh-dss-cert-v01@openssh.com`					●			
`ssh-dss-cert-v00@openssh.com`					●			
SSH Communications Security Extensions								
`ssh-rsa-sha224@ssh.com`				●				●
`ssh-rsa-sha256@ssh.com`				●				●
`ssh-rsa-sha384@ssh.com`				●				●
`ssh-rsa-sha512@ssh.com`				●				●
`ssh-dss-sha256@ssh.com`				●				●
`x509v3-sign-dss-sha256@ssh.com`				●				
`x509v3-sign-rsa-sha256@ssh.com`				●				
Draft standards								
`x509v3-sign-dss`				●				
`x509v3-sign-rsa`				●				

11.4   Encryption algorithms

Method	TeraTerm	PuTTY	Dropbear	Tectia	OpenSSH	libssh	libssh2	emSSH
`aes256-ctr`	●	●	●	●	●	●	●	●
`aes256-cbc`	●	●	●	●	●	●	●	●
`aes192-ctr`	●	●		●	●	●	●	●
`aes192-cbc`	●	●		●	●	●	●	●
`aes128-ctr`	●	●	●	●	●	●	●	●
`aes128-cbc`	●	●	●	●	●	●	●	●
`camellia256-ctr`	●							●
`camellia256-cbc`	●							●
`camellia192-ctr`	●							●
`camellia192-cbc`	●							●
`camellia128-ctr`	●							●
`camellia128-cbc`	●							●
`3des-ctr`	●	●						●
`3des-cbc`	●	●		●	●	●	●	●
`twofish256-cbc`			●	●				●
`twofish256-ctr`								●
`twofish192-cbc`			●	●				●
`twofish192-ctr`								●
`twofish128-cbc`			●	●				●
`twofish128-ctr`								●
`twofish-cbc`			●	●				●
`blowfish-ctr`	●	●						●
`blowfish-cbc`	●	●		●	●	●	●	●
`arcfour256`	●	●			●			●
`arcfour128`	●	●			●		●	●
`arcfour`	●			●	●		●	●
`cast128-ctr`	●							●
`cast128-cbc`	●				●		●	●
`chacha20-poly1305`					●			●
OpenSSH Extensions								
`aes128-gcm@openssh.com`					●			●
`aes256-gcm@openssh.com`					●			●
`chacha20-poly1305@openssh.com`					●			●
SSH Communications Security Extensions								
`seed-cbc@ssh.com`				●				●
Other Extensions								
`rijndael-cbc@lysator.liu.se`		●			●		●	●

11.5   MAC algorithms

Method	TeraTerm	PuTTY	Dropbear	Tectia	OpenSSH	libssh	libssh2	emSSH
`hmac-sha2-512`	●		●	●	●	●	●	●
`hmac-sha2-256`	●	●	●	●	●	●	●	●
`hmac-sha1`	●	●	●	●	●	●	●	●
`hmac-sha1-96`		●	●	●	●		●	●
`hmac-md5`	●	●	●	●	●		●	●
`hmac-md5-96`				●	●		●	●
OpenSSH Extensions								
`hmac-sha2-512-etm@openssh.com`					●			●
`hmac-sha2-256-etm@openssh.com`					●			●
`hmac-sha1-etm@openssh.com`					●			●
`hmac-sha1-96-etm@openssh.com`					●			●
`hmac-md5-etm@openssh.com`					●			●
`hmac-md5-96-etm@openssh.com`					●			●
`hmac-ripemd160@openssh.com`	●				●		●	●
`hmac-ripemd160-etm@openssh.com`					●			●
`umac-64@openssh.com`					●			
`umac-64-etm@openssh.com`					●			
`umac-128@openssh.com`					●			
`umac-128-etm@openssh.com`					●			
SSH Communications Security Extensions								
`hmac-sha224@ssh.com`				●				●
`hmac-sha256@ssh.com`				●				
`hmac-sha256-2@ssh.com`				●				●
`hmac-sha384@ssh.com`				●				●
`hmac-sha512@ssh.com`				●				●

Chapter 12

Patents and export controls

This section tries to describe the patents and licenses you may require to deploy SSH *in general*, whatever SSH solution you choose, be it emSSH or some other product. It also describes export controls that may apply to your equipment.

Patents are granted, challenged, and struck down over time, in different geographical regions, and for different fields of use; these facts alone make it impossible to provide a factually-accurate, blanket statement regarding all end-customer equipment. Export controls change in the same way, but usually at a slower pace.

Export controls apply to emSSH itself and the end user equipment in order to restrict the export of strong cryptography. emSSH uses strong cryptography for signing and encryption. And as export controls depend upon what you are exporting, what the purpose of the cryptographic device is, and what type of cryptography your device has, it is impossible to provide statements that cover all situations and devices.

> **Note**
>
> We strongly advise you to conduct your own research and consult legal advisors on deployment of SSH (and ECC cryptography in particular) in your devices. This content of this section is provided without warranty of any kind. It is your sole responsibility to decide whether or not you wish to make use of ECC technology in your product.

To the best of our knowledge, having researched the issue, the following sections are guidelines for the deployment of emSSH in end-user equipment.

12.1   Using RSA signatures

The patents relating to RSA signatures are now all expired and, to the best of our knowledge, the use of RSA signatures requires no license from any patent holder.

12.2   Using DSA signatures

DSA signatures for SSL are not in common use, but are mandated as a requirement for SSH. Despite this, the patents relating to DSA signing are all expired and, to the best of our knowledge, the use of any DSA signature suite requires no license from any patent holder.

> **Note**
>
> DSA patents were assigned to United States of America and appropriate patents are made available worldwide on a royalty-free basis.

12.3   Using elliptic curve cryptography

ECC is the most problematic public key cryptosystem because of active patents assigned to Certicom (at the time of writing, June 1, 2022). There is no concise, clear statement relating to the implementation of elliptic curve cryptography from Certicom, nor is there a simple means of discovering whether an implementation scheme is covered by an ECC patent—you will need to conduct your own patent search in your geographic region.

To the best of our knowledge, the functions provided by emSSH do not infringe on any *implementation* patent.

It is required that anybody contributing to an IETF standard document disclose all IPR that they hold relating to the standard. To deploy ECC in your product, you may require a license from Certicom.

In order to avoid this, *do not add ECDH key exchange* and *do not add ECDSA public key algorithms* when initializing emSSH. In this manner, no ECC code is linked into your application.

We believe that the specialized reducers are not covered by an implementation patent because of prior art. If this situation causes you concern, you can avoid deploying the specialized reducers in your code and use only the slower, simple reduction scheme that use algorithms from antiquity.

> **Note**
>
> We stress again that patents are issued covering both different geographic domains and fields of use. It is your responsibility to ensure that your end products (that contain emSSH) do not infringe patents in the locations that you produce and sell that equipment.
>
> This contents of this section is provided without warranty of any kind. It is your sole responsibility to decide whether or not you wish to make use of ECC technology in your product and to seek independent legal advice.

12.4 Export controls

Strong cryptography is subject to export controls from many countries. Because emSSH supports strong cryptography and does not limit public key lengths, you must ensure that the equipment that you export complies with export controls in the country you export from and the country that you export into.

The European Union has a common dual-use goods list (including encryption items in Category 5, Part 2 "Information Security") defined by EC Regulation No 428/2009. Several member states have, in addition, regulations concerning the import, supply, use or export of encryption items. emSSH has been provided to you in accordance with the EC regulations and national laws of Germany. Any export or transfer of the software with a destination outside the European Union requires export permission.

The choice of cipher suites and public key algorithms is completely configurable in emSSH, allowing you to tailor the cryptographic capability of the device deployed with emSSH.

Note

We stress again that it is impossible to provide any warranties of statements made regarding export controls. It is your responsibility to ensure that your end products (that contain emSSH) conform to the export controls in place at the time and place of export.

This content of this section is provided without warranty of any kind. It is your sole responsibility to decide whether your product with cryptographic capability complies with appropriate export controls and to seek independent legal advice.

Chapter 13

Reference information

The SSH protocol is not defined by a single specification but as a range of specifications maintained by the Internet Engineering Task Force (IETF) as Requests for Comments (RFCs). All RFCs are made available online at the IETF website.

This section collates reference information relating to the implementation of SSH protocols by emSSH and should be considered the definitive specification when understanding emSSH's behavior. In addition to the RFCs mentioned above, reference is also made to national standards documents for many cryptographic algorithms. The references cited here does not constitute an exhaustive collection because it omits X.509 certificate specifications and the various ASN.1 standards.

Because information relating to the various SSH protocols and extensions is covered by so many standards documents, it is may be that we have not correctly understood or implemented all features of emSSH according to these standards. If you believe that emSSH does not comply with the appropriate standards, please contact us so we can investigate any discrepancy where you believe we fall short.

13.1   SSH specifications

emSSH implements SSH protocol version 2.

The base SSH specifications are spread across these RFCs:

The Secure Shell Protocol Architecture (RFC 4251)

https://datatracker.ietf.org/doc/html/rfc4251

The Secure Shell Authentication Protocol (RFC 4252)

https://datatracker.ietf.org/doc/html/rfc4252

The Secure Shell Transport Layer Protocol (RFC 4253)

https://datatracker.ietf.org/doc/html/rfc4253

The Secure Shell Connection Protocol (RFC 4254)

https://datatracker.ietf.org/doc/html/rfc4254

Generic Message Exchange Authentication for the Secure Shell Protocol (RFC 4256)

https://datatracker.ietf.org/doc/html/rfc4256

Diffie-Hellman Group Exchange for the Secure Shell Transport Layer Protocol (RFC 4419)

https://datatracker.ietf.org/doc/html/rfc4419

SHA-2 Data Integrity Verification for the Secure Shell (SSH) Transport Layer Protocol (RFC 6668)

https://datatracker.ietf.org/doc/html/rfc6668

13.2   Hardware acceleration

13.2.1   RSA and elliptic curve accelerators

Using a modular arithmetic accelerator will speed up RSA and elliptic curve operations over prime fields which reduces key exchange time considerably. Although emSSH contains a robust implementation of modular exponentiation, it is no match for hardware acceleration.

Unfortunately, devices that offer strong cryptography are subject to export controls. Worse still, many vendors restrict the documentation for both the devices themselves and the cryptographic accelerators they implement. Executing a non-disclosure agreement (NDA) with the silicon vendor to access the information and in order to provide support in emSSH still prevents delivering that implementation to customers unless the customer also has a non-disclosure agreement with the silicon vendor.

For example, the following devices have cryptographic modular arithmetic accelerators but their cryptographic abilities are covered by an NDA:

• Atmel SAMA5D4
• Maxim MAX32550
• Maxim MAXQ1103 and MAXQ1850

However, some devices are sufficiently open:

• Silicon Labs EFM32 (Pearl and Jade Gecko)

13.2.2   Hash and MAC accelerators

Hash and MAC algorithms cannot be used for encryption and, as such, export controls usually do not apply to them. Many modern devices that offer Ethernet hardware also offer a SHA accelerator in full expectation that implementations will make use of it.

Whilst the SHA accelerator will speed up the bulk encipherment process after a connection is established, it does little for the key agreement phase of a connection beyond speeding up signature generation and verification.

13.2.3   Bulk encipherment accelerators

The same devices that offer hash and MAC capabilities usually offer bulk encipherment with an AES accelerator for AES-128 or AES-256, or perhaps both. And fortunately, it seems that device datasheets and reference manuals that document how the particular AES accelerator functions are not under NDA.

Whilst the AES accelerator will speed up the bulk encipherment process after a connection is established, it does nothing for the key agreement phase of a connection or for signature generation and verification.

13.2.4   Vendor-optimized and certified libraries

It may well be that you would like to use vendor-optimized libraries provided for your device, or even NIST-certified "cryptograpic components" (i.e. libraries). It is possible to swap out parts of emSSH's cryptography and replace it with another implementation, but doing so is beyond the scope of this document.

Because there is no standard cryptography API, integrating alternative hardware and software libraries will require some effort. Please see the next section for how to proceed.

13.2.5   What to do if you require alternative cryptography

If you would like to replace any part of the standard emSSH cryptography implementation with either a hardware-accelerated or certified implementation that we do not already support, please contact us. It may well be that we have a particular hardware accelerator already implemented for emSSH and ready to go, or we may have already written the support code to transition from emSSH's APIs to other vendor library APIs.

Chapter 14

Glossary

3DES

Triple DES. A classical means to extend the 56-bit key space of DES to 112 bits by combining two 56-bit keys in three DES operations (two-key 3DES-EDE). 3DES is also known as TDES in standards documentation.

AEAD

Authenticated Encryption with Additional Data. A modern cipher mode that combines encryption with authentication where both can run in parallel and enhance throughput in hardware implementations. AES-GCM and AES-CCM are AEAD ciphers..

AES

Advanced Encryption Standard. A modern 128-bit block cipher, specified by NIST, that replaces the DES standard.

ASN.1

Abstract Syntax Notation 1. A specification of how to encode primitive data as octet streams.

CBC

Cipher Block Chaining. A cipher mode that uses the output of the previous block as an input to the following block to be encrypted.

DES

Data Encryption Standard. A retired 64-bit block cipher with 56-bit keys defined by NIST.

DH

Diffie-Hellman. A key agreement scheme based on discrete logarithm cryptography.

DHE

Ephemeral Diffie-Hellman. A key agreement scheme based on discrete logarithm cryptography where keys are generated once per connection and are unique for each connection. This guarantees Perfect Forward Secrecy (PFS).

DRBG

Deterministic Random Bit Generator. A random bit generator that will generate the same sequence of bits given the same seed as input, but the output is random using standard randomness tests.

DSA

Digital Signature Algorithm. The algorithm that signs a piece of data, specified in the Digital Signature Standard.

DSS

Digital Signature Standard. The NIST digital signature standard that specifies the Digital Signature Algorithm (DSA).

DTLS

Datagram Transport Layer Security. A scheme similar to TLS that transports TLS messages over UDP datagrams.

ECB

Electronic Code Book. An insecure mode for a block cipher where each block is encrypted in isolation and not chained.

ECC

Elliptic Curve Cryptography. Cryptography based on elliptic curves.

ECDH

Elliptic Curve Diffie-Hellman. The equivalent of Diffie-Hellman using elliptic curves.

ECDHE

Elliptic Curve Diffie-Hellman Ephemeral. As ECDH but using ephemeral keys. Provides perfect forward secrecy (PFS).

ECDSA

Elliptic Curve Digital Signature Algorithm. A standard for digital signatures signed using elliptic curve cryptography rather than discrete log cryptography. The elliptic curve analog of the discrete log signature scheme (DSA).

FIPS

Federal Information Processing Standard. A standard issued by NIST for Federal use and widely adopted throughout the world.

GCM

Galois Counter Mode. A modern mode for a block cipher where the authentication tag is computed using arithmetic in a Galois field, $GF(2^{128})$..

HMAC

Hashed Message Authentication Code. A MAC that is computed using a cryptographic hash function in combination with a secret key.

IANA

Internet Assigned Numbers Authority. IANA is responsible for the global coordination of the Internet protocol resources for TLS as part of its mandate.

IETF

Internet Engineering Task Force. The IETF produces high quality, relevant technical documents that influence the way people design, use, and manage the Internet.

MAC

Message Authentication Code. A small piece of information used to authenticate a message and to provide integrity and authenticity assurances about the content of the message..

MD5

Message Digest Algorithm 5. A MAC defined by RSA Data Security, Inc.

MPI

Multiprecision integer. An integer that can grow and shrink as required to represent cryptographic numbers.

NIST

National Institute of Standards and Technology. An organization in the USA responsible for the standardization of a wide range of technologies that pervade the IT industry.

PBKDF2

Password-based Key Derivation Function 2. An algorithm that takes a password and a salt and combines them to produce keying material (a *derived key*).

PFS

Perfect Forward Secrecy. A means to ensure that exposure of the session keys for one connection and its decryption does not expose other sessions to subsequent decryption using the recovered cryptographic material.

PKI

Public Key Infrastructure. A set of specifications and mechanisms that can provide confidence in and interoperability of public key cryptography systems.

PRF
Pseudorandom Function. A function defined in the TLS specifications used to generate various unpredictable internal data used by TLS connections.

PSK
Preshared Key. A shared private key held by two entities agreed in advance of communication.

RFC
Request For Comment. The standard means that IETF disseminates Internet standards.

RNG
Random Number Generator. A device that generates true random numbers.

RSA
Rivest, Shamir, Adleman. The name of the cryptosystem based on Integer Factorization problems defined by the three authors.

SHA
Secure Hash Algorithm. The standard set of one-way functions that provide a message digest, as specified by NIST.

SSH
Secure Shell. A protocol that provides a secure connection and login to a server.

SSL
Secure Sockets Layer. Previous name for Transport Layer Security (TLS).

TDES
Triple DES. See 3DES.

TLS
Transport Layer Security. The current name and standard definition that provides confidential and authenticated transmission of data over insecure channels.

Chapter 15

Indexes

15.1   Type index

15.2   Function index