Technical Note

TN245

Client-Side SSL Workaround Using Rabbit Server-Side SSL

Some Rabbit-based applications may require that the device initiate a connection to a server (with the Rabbit acting as a client), rather than the other way around (with the Rabbit acting as a server). The current Rabbit implementation of SSL does not support client-side SSL, so some applications will simply not be possible. However, there are a large number of applications that can use the server-side SSL implementation to emulate a client-side SSL setup (i.e., the Rabbit initiates the connection, but still uses SSL). The following process is one possible solution (other solutions are likely) that can be used to implement a "client-push" application using the Rabbit and the current SSL implementation, as long as the developer has access to the server-side (i.e., the PC) application.

> **NOTE:** Rabbit SSL does not have a published public API, so it will be necessary to use the HTTPS methods in the Dynamic C **`HTTP.LIB`** library to do SSL communication. All of these functions are in **`HTTP.LIB`**, begin with the prefix **`http_sock_`**, and are documented for the **<Ctrl-H>** help functionality in Dynamic C.

The process for setting up client-SSL emulation (or client callback) requires only one additional step when establishing the SSL session, and some software support on the server. The steps of the process are outlined below and in Figure 1 on the next page.

1. The Rabbit-based device initiates a standard TCP/IP connection to the server, and sends a plain-text packet with a unique identifier (***not*** an IP address – the unique identifier ensures that fraudulent attempts to mimic the Rabbit-based device are not possible, see Step 2).
2. The server (PC) stores a table* that associates the Rabbit's unique identifier to the Rabbit's IP address. This way, even if an attacker tried to mimic the Rabbit by sending a message with the same identifier, the server would only connect to the actual Rabbit-based device (assuming the address could not be spoofed, but the SSL authentication by the PC/server would fail if that happened — so the method is as secure as a standard Rabbit SSL session).
3. Upon receiving the "connect-to-me" message from the Rabbit, the server initiates a (client) connection to the Rabbit-based device and begins an SSL session. From here, the SSL protocol takes over. Once the SSL session is established, there is no functional difference between the client and the server, so the user is then free to use the SSL socket for any desired purpose.

* This table is part of the reason why the developer needs access to the server-side application.

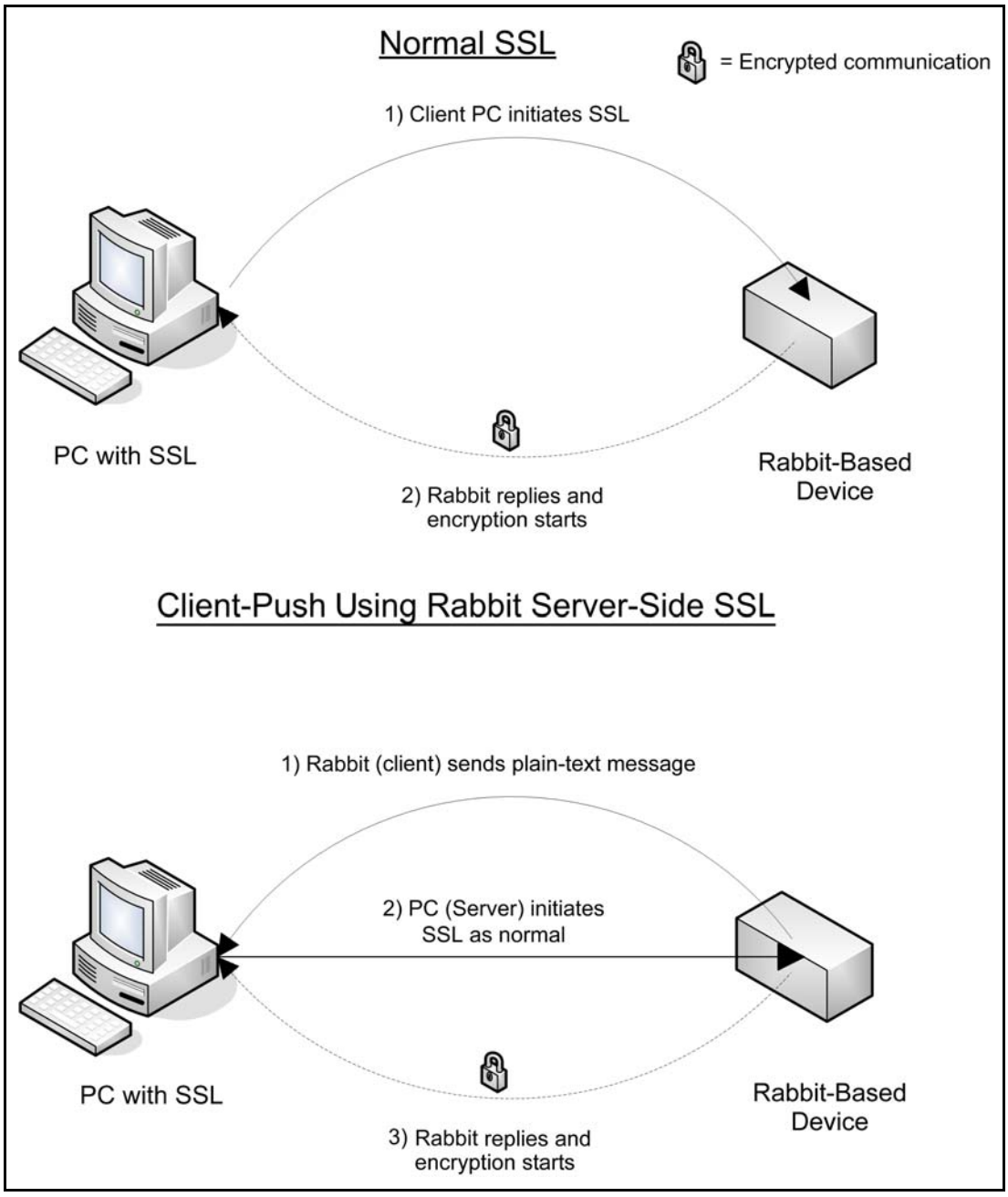

Figure 1. Steps to Set Up Client-SSL Emulation

The primary catch with this method is that if the Dynamic C **HTTP.LIB** library is used, the HTTP headers must be discarded by the server and by the Rabbit when reading from the socket during a non-Web transmission.* This is fairly simple to do in practice (see the References below on the HTTP protocol).

This strategy suffers from a potential denial-of-service attack, but limiting the number of connections to a single device in a specific period of time should alleviate some of this issue (this is only one possible solution to the DOS attack problem — generally this problem is difficult to deal with in any network situation and is not limited to Rabbit-based devices).

References

1. IETF RFC2616. Hypertext Transfer Protocol — HTTP v. 1.1.
2. World Wide Web Consortium (www.w3.org), Hypertext Transfer Protocol.
3. Internet Engineering Task Force (www.ietf.org).
4. Dynamic C TCP/IP User's Manual, Volume 2, Chapters 2 and 4.

* This is another reason why the developer needs access to the server.

Z-World, Inc.

2900 Spafford Street
Davis, California 95616-6809
USA

Telephone: (530) 757-3737
Fax: (530) 757-3792

www.zworld.com

Rabbit Semiconductor

2932 Spafford Street
Davis, California 95616-6809
USA

Telephone: (530) 757-8400
Fax: (530) 757-8402

www.rabbitsemiconductor.com