Linux Invention

Contact Information

Mr Rich Maggiani; rich.maggiani@solari.net

Solari Communication
137 Lost Nation, Suite 14
Essex, Vermont 05452
United States of America
802.879.9330

Title of the Invention

Method for Securely Transmitting Information Online When the Security Is Enabled by the Sender

Description of the Invention

Most digital data transmissions are not secure; most don't need to be. There are times, however, when a network client wants to secure a transmission, to transmit encrypted data (files, messages, or any other type of data). While methods exist to individually transmit secure messages, they can be cumbersome. This invention describes a method whereby Linux simplifies the process for a user to initiate a secure transmission while adding a deeper level of security to better ensure a successful transmission.

Problem or Opportunity

Users sometimes want to secure online transmissions that would otherwise not need to be secured (such as text transmissions in email messages). In order to do this, a user must implement their own secure transmissions and ensure that the receiver also has the necessary decryption process.

Users also want transactions that are regularly encrypted and sent securely to be more reliable and not subject to random identity theft.

A process that reliably and automatically secures all such transmissions would be most welcome.

Detailed Description of the Invention

Linux loads a java applet that creates a security button into the browsers of the sender and receiver of a transmission. The sender clicks the security button whenever a secure transmission was desired. This can be any type of transmission, from a text-only email to a complicated financial transaction.

Once the security button is clicked, Linux applies a security protocol to the transmission so that the message is sent securely. This security protocol includes a security key and a number of data packets.

Linux creates an asymmetrically encrypted security key for the transmission; the security key indicates the number of data packets to be sent. Linux creates two or more encrypted data packets. Only one data packet contains the actual data the sender wants to transmit; the remaining packets, while looking like they contain actual data, in fact contain only dummy data. The security key identifies which packet contains the actual data.

For example, Linux sends a security key, then sends four packets of secure data, the third of which contains the actual information to be decrypted.

Linux begins the actual transmission by passing the security key to the sender's browser, which then sends the key to the receiver's browser or to any online service that will process the transaction. Both must be running under a version of Linux. This transmission invokes the security button on the receiver's browser or contains an indication of a secure transaction if the transmission is not received by a remote browser. This causes the receiver's (remote) Linux to obtain the security key from the transmission. The remote Linux acknowledges receipt of this security key.

The sender's (local) Linux determines if the acknowledgement is clean (not corrupted in any way) or that its receipt has not timed out. If a problem occurs, the local Linux cancels the transmission and sends notification of this failure to the sender's browser.

Upon receiving a successful acknowledgement, the local Linux begins to transmit the data packets. The local Linux sends some of the packets to the receiver's browser, and some to a proprietary Linux database residing on the Internet. The remote Linux gathers the data packets in the transmission, then accesses the Linux database and downloads the remaining data packets. The remote Linux collects all the data packets and the security key, ensures that the entire transmission has been collected, locates the packet with the actual encrypted data (from information contained in the security key), decrypts it, and sends it to the receiver's browser. The browser then processes the data transaction.

If all the packets do not arrive as anticipated, then the transaction fails and an error message is returned to the local Linux.

Figure 1 depicts an overview of this sender-enabled encrypted transmission.

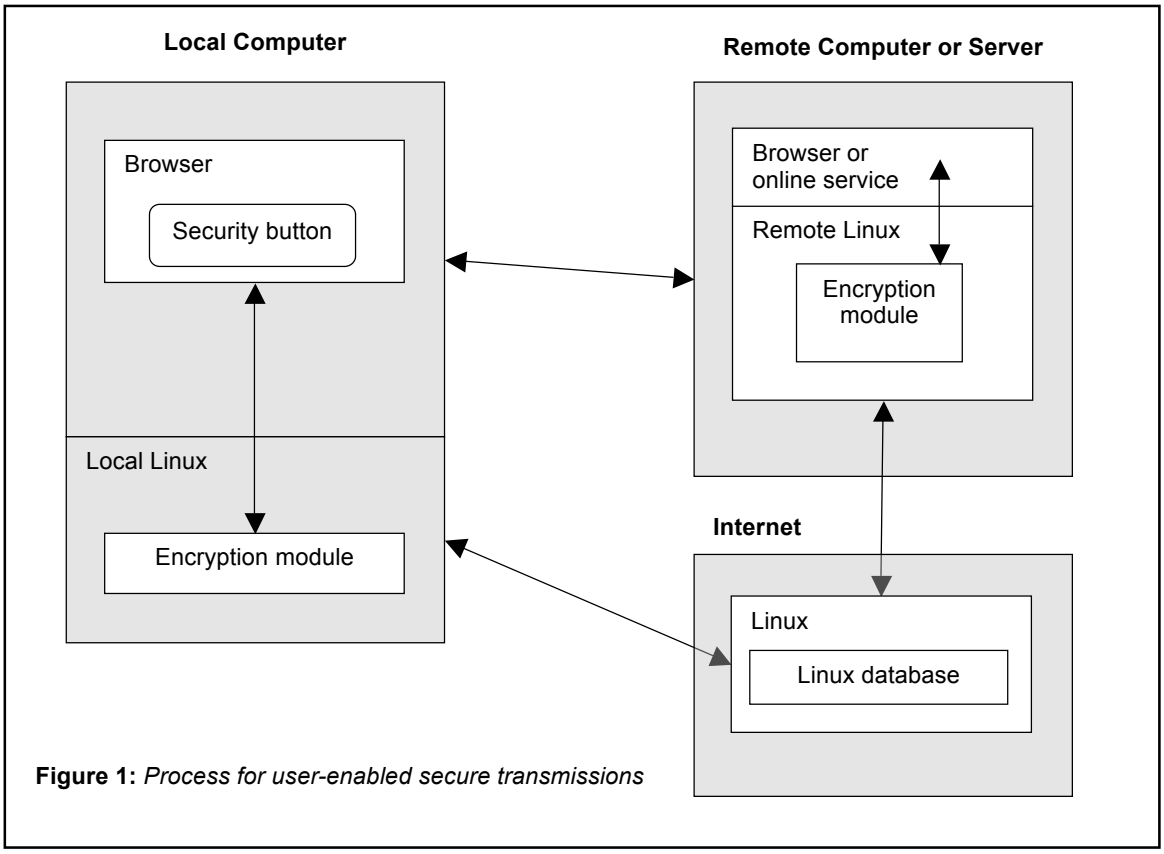

Figure 1: *Process for user-enabled secure transmissions*

The method starts with Linux establishing a proprietary method for a user to initiate a secure transaction whenever that user deems necessary.

In step 1, Linux loads a java applet that creates a security button into the browsers of the sender and receiver of a transmission.

In step 2, a user clicks the security button to initiate a secure transmission.

In step 3, Linux applies its proprietary security protocol to the transmission. Linux creates an asymmetrically encrypted security key, and two or more encrypted data packets only one of which contains the actual data to be transmitted. The security key identifies which packet contains the actual data.

In step 4, Linux passes the security key to the sender's browser.

In step 5, the sender's browser transmits the key to the receiver's browser or online service.

In step 6, the receiver's (remote) Linux recognizes the secure transmission. The remote Linux decodes the security key and returns an acknowledgement of the key's successful receipt (retracing the path of the key's original transmission).

In decision step 7, the sender's (local) Linux receives the acknowledgement. If the acknowledgement is corrupt, its transmission times out, or some other problem occurs, the method continues with step 8. If the acknowledgement is received successfully, the method proceeds to step 9.

In step 8, the local Linux cancels the transmission and sends notification of this failure to the sender's browser. The method ends.

In step 9, the local Linux, upon receiving a successful acknowledgement, begins to transmit the data packets. Linux sends some packets to the local browser and the remaining packets to a proprietary Linux database residing on the Internet together with information about the transmission.

In step 10, the local browser transmits its data packets to the remote browser or online service.

In step 11, the remote Linux accepts the transmission of the data packets, retrieves the remaining data packets from its Internet database, and attempts to reassemble the entire transmission: the security key and all the data packets.

In decision step 12, the remote Linux attempts to validate the entire transmission. If the transmission is invalid for some reason or cannot be reassembled, the method continues with step 13; if the entire transmission is successful, the method proceeds to step 15.

In step 13, the remote Linux sends a failure notice to its browser or online service.

In step 14, the browser or online service logs the error with any information about the transmission it can garner, and sends notification of the failure to the sender's browser. The method ends.

In step 15, the remote Linux locates the valid data packet and decrypts it, then sends the data to its browser or online service.

In step 16, the browser or online service processes the transmitted data.

The method ends.

Figure 2 depicts a flowchart of this sender-enabled encrypted transmission.

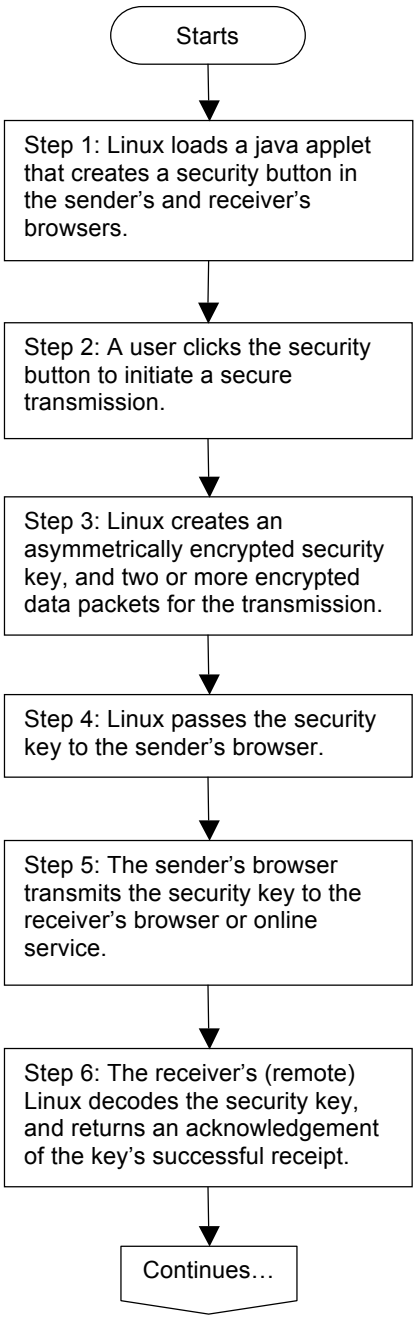

Figure 2. *Flowchart for user-enabled secure transmissions (part 1 of 3)*

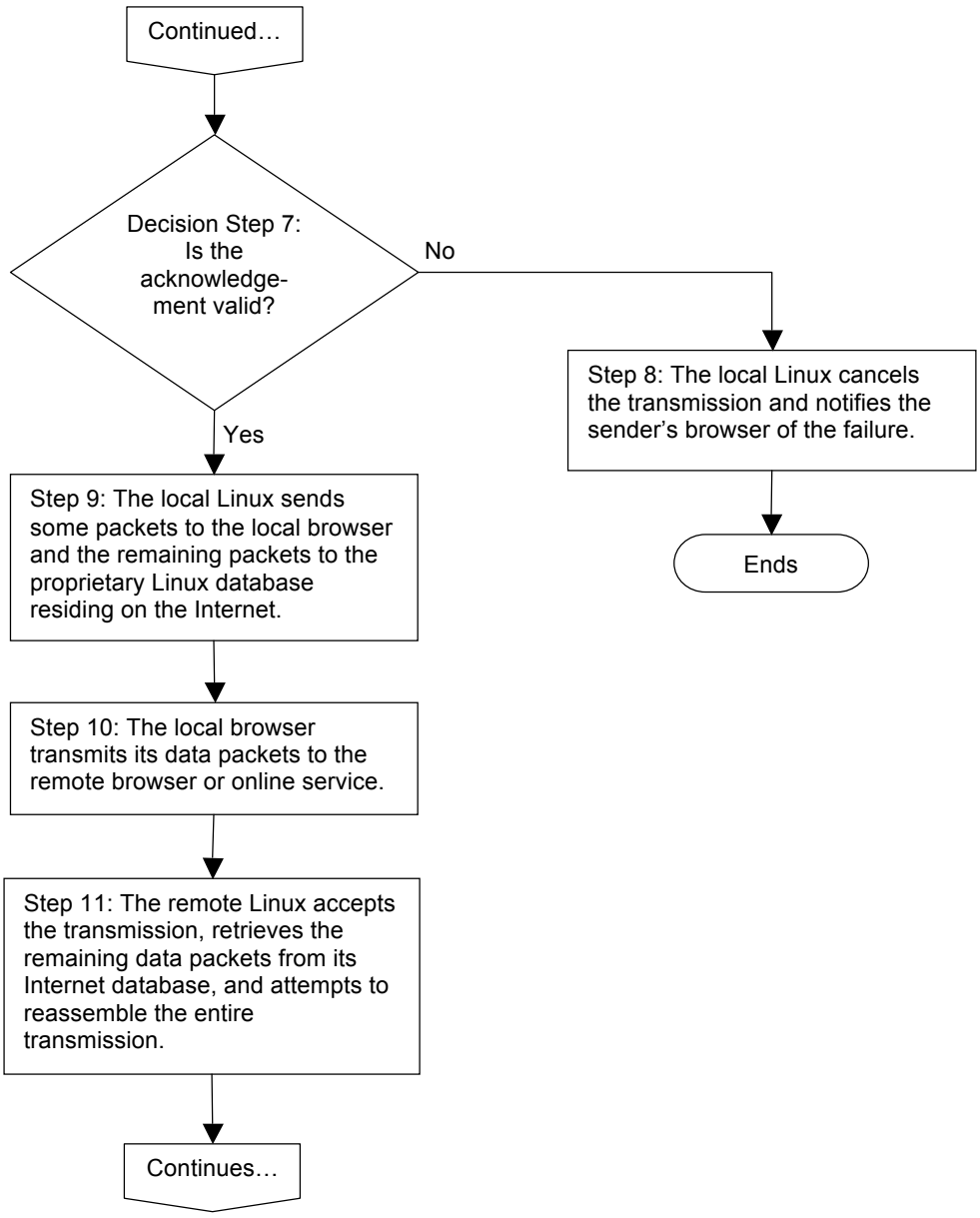

Figure 2. *Flowchart for user-enabled secure transmissions (part 2 of 3)*

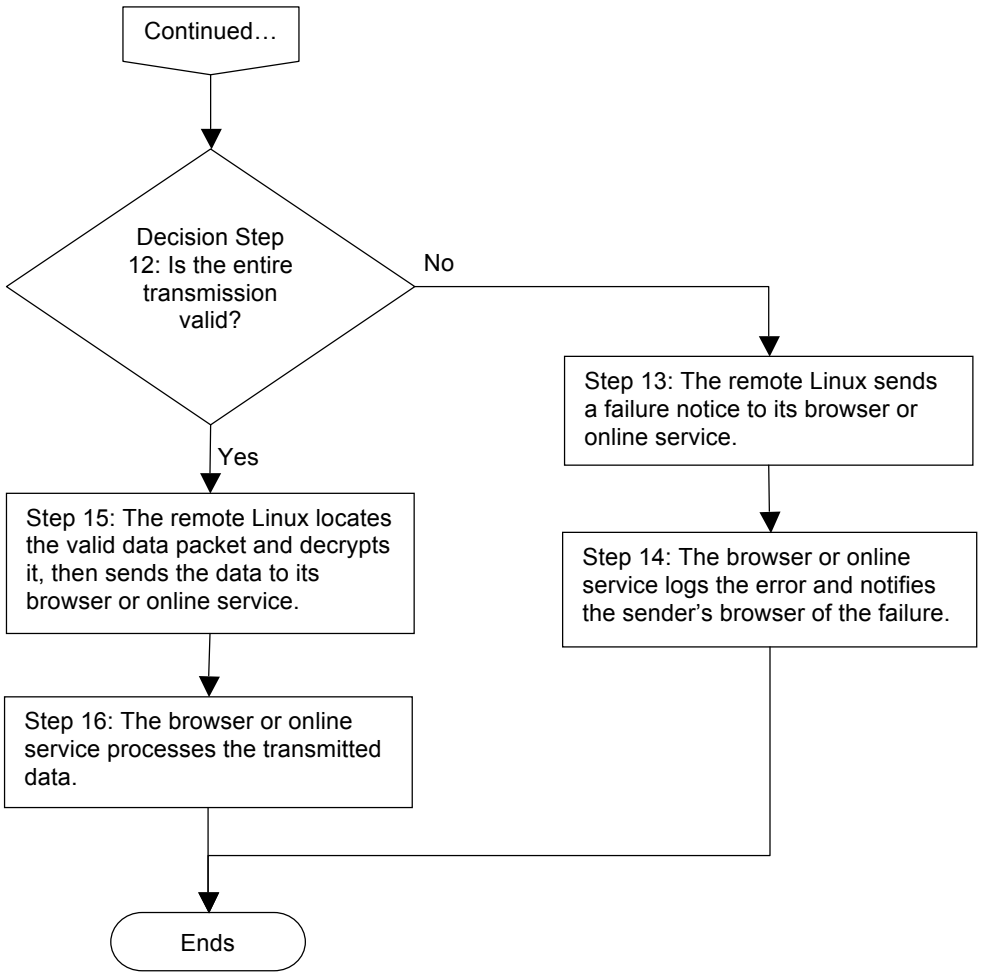

Figure 2. *Flowchart for user-enabled secure transmissions (part 3 of 3)*