

[0155] In some embodiments the decryption algorithm (e.g., 3DES) is performed over the fields represented by line 920, namely, the 3DES IV 926 and the secure code 928.

[0156] In some embodiments the authentication (e.g., HMAC-SHA1) may be performed over the fields represented by line 918: {a 4 byte firmwareHash 922| a 4 byte logical address 924| an 8 byte IV 926| a 256 byte secure code block 928}. The line 914 represents that the secure code from the memory 902 is provided to the cryptographic operations. The firmware hash field 922 from the secure code descriptor prevents an attacker from mixing secure code blocks from different secure code load revisions. The 4 byte logical address 924 points to the base address of the 256 byte block and may be used to determine whether an attacker has moved any of the code blocks in the flash memory.

[0157] As represented by line 916, the authentication generated as a result of the HMAC-SHA1 operation over the fields 918 is verified against the code authentication 908 for the corresponding 256 byte block stored in the flash memory. After successful authentication, the cache line may be marked as valid and the data may be returned to the instruction multiplexer block. On a failure, the cache line may be marked as invalid and the instruction cache controller may raise an error signal to the instruction multiplexer.

[0158] Referring now to FIGS. 7 and 8, one embodiment of operations that may be performed to update secure code will be discussed. FIG. 7 relates to operations that may be performed by, for example, a manufacturer to create and deliver new code to a target system (e.g., a TPM) that has been installed in the field. FIG. 8 relates to operations that may be performed by the target system (hereafter referred to for convenience as the TPM) to install the new code.

[0159] At block 702 in FIG. 7 the manufacturer creates any new functions that are needed and modifies any functions that need to be changed. Typically, this will involve a programmer rewriting a portion of the last release of the code and/or adding new sections to that code. As the instruction ROM code and the on-chip function table stored in a masked instruction ROM may not be modified after the TPM has been taped-out (as part of the integrated circuit design and manufacturing process), any changes to the code are stored in the flash memory. However, since there may be significant overhead associated with fetching data (e.g., code) from flash memory it may be desirable to store as much of the code in on-chip ROM as possible.

[0160] To modify code, an entire function may be replaced. Depending on whether code has been previously loaded into the flash memory, modifying a function may involve creating a modified version of an original function (e.g., one stored in an internal memory) or modifying a function that was previously loaded into the flash memory. In some embodiments the only functions that cannot be replaced are secure code functions and low-level functions that access the flash.

[0161] In some embodiments the guidelines that follow may be used to more effectively update the code. First, existing global structures may not be modified. Since global structures may be used by multiple functions any change in the global structures may cause a large number of functions to be updated. Second, fields may be added to the end of

volatile structures. Adding fields to the end of volatile structures may cause the heap pointer to be moved at boot. Third, memory may be allocated for adding fields to non-volatile structures.

[0162] At block 704 the compiler generates the machine code that will be stored in the external memory. Here, the compiler may determine which functions have changed or have been added. For example, a script (e.g., provided as an extension of the compiler) may compare (e.g., perform a “diff” operation) the newly compiled code with a previously saved copy of the prior version of the code. In some embodiments, the compiler may advantageously be configured to compile only the new and/or modified code.

[0163] As represented by block 706, the order of functions stored in flash memory is determined. The order of the functions may be determined, for example, by the programmer or automatically by the compiler. In some embodiments functions are ordered to maximize locality. The order of functions in the original flash function table is not changed.

[0164] As represented by block 708, the code may be defined and/or modified to support function calls to code stored in either the internal memory or the external memory. This may be accomplished as discussed herein. For example, a script may be defined that searches the code for a specific type of function call and re-writes that function call to support a table offset variable.

[0165] As represented by block 710 the manufacturer generates a new external function table that contains, for example, function pointers for each function. Again, this may be performed by a script (e.g., an extension of the compiler). For any original functions that have not been modified, the function table points to the internal memory. For any modified version of an original function or any new function, the flash function table points to the external memory. In the latter case, the corresponding function pointers are either modified or added in the external flash function table.

[0166] As represented by block 712, the system also generates information for a new flash secure code descriptor. As discussed above, this information may include, for example, a version number and a verification key.

[0167] As represented by block 714, the secure environment (e.g., a FIPS level 3 environment such as a hardware security module as discussed in conjunction with FIG. 1) may generate a new verification key (and optionally a new decryption key) for the TPM. Replacing the public verification key allows the manufacturer to prevent older secure code loads from replacing the latest firmware revision. In addition, if the private signing key is ever compromised, this mechanism allows the manufacturer to release a secure code image with a new signing/verification key. The new public keys may be provided to the TPM by incorporating them into the new secure code descriptor.

[0168] As represented by block 716, the compiled code may be sent to the secure environment. Here, cryptographic processor(s) in the secure environment may use the private verification key (and optionally the private encryption key) to cryptographically sign (and optionally encrypt) the secure code information. As discussed herein, the private key corresponds to a public key maintained by the TPM. The secure code information may include, for example, the