

INSTRUCTION TO ACCELERATE HASH-BASED MESSAGE AUTHENTICATION CODE PROCESSING

BACKGROUND

[0001] One or more aspects relate, in general, to cryptographic processing within a computing environment, and in particular, to cryptographic hash functions.

[0002] Cryptography is used for the protection of data. There are a number of cryptographic functions, including various cryptographic hash functions, such as SHA-2 (Secure Hash Algorithm 2) and SHA-3, as examples, that may be used to protect data. A cryptographic hash function may be used to provide a message authentication code, such as a hash-based message authentication code (HMAC), used to verify data integrity and authenticity of a message.

[0003] Hash-based message authentication code processing uses two passes of hash computation, in which prior to each pass, a confidential key is used to derive multiple keys, including an inner-key and an outer-key. In the first pass, an internal hash is derived from the message and the inner-key, and in the second pass, a final hash-based message authentication code is derived from the inner hash result and the outer-key.

SUMMARY

[0004] Shortcomings of the prior art are overcome, and additional advantages are provided through the provision of a computer program product. The computer program product includes a set of one or more computer-readable storage media and program instructions, collectively stored in the set of one or more computer-readable storage media, for causing at least one computing device to perform computer operations. The computer operations include executing an instruction to generate an authentication code. Executing the instruction includes performing a plurality of operations of the instruction to generate the authentication code. The plurality of operations includes performing a sequence of hash operations on a message obtained using the instruction to generate an intermediate message digest. The performing the sequence of hash operations uses an output chaining value generated based on performing an inner-key padding and hashing operation using a cryptographic key of the instruction. An outer-key padding and hashing operation is performed using the cryptographic key to generate another output chaining value to be used in generating a final output message digest based on a final input message digest produced using the intermediate message digest. The final output message digest being a resulting authentication code. The performing the sequence of hash operations and the outer-key padding and hashing operation are performed as part of a single invocation of the instruction.

[0005] In one or more aspects, a computer system is provided. The computer system includes at least one hardware accelerator to be used in executing an instruction to generate an authentication code. Executing the instruction includes performing a plurality of operations of the instruction to generate the authentication code. The

plurality of operations includes performing a sequence of hash operations on a message obtained using the instruction to generate an intermediate message digest. The performing the sequence of hash operations uses an output chaining value generated based on performing an inner-key padding and hashing operation using a cryptographic key of the instruction. An outer-key padding and hashing operation is performed using the cryptographic key to generate another output chaining value to be used in generating a final output message digest based on a final input message digest produced using the intermediate message digest. The final output message digest being a resulting authentication code. The performing the sequence of hash operations and the outer-key padding and hashing operation are performed as part of a single invocation of the instruction. According to one aspect, there is provided a computer program product comprising: a set of one or more computer-readable storage media; and program instructions, collectively stored in the set of one or more computer-readable storage media, for causing at least one computing device to perform computer operations including: executing an instruction to generate an authentication code, the executing the instruction including: performing a plurality of operations of the instruction to generate the authentication code, the plurality of operations including: performing a sequence of hash operations on a message obtained using the instruction to generate an intermediate message digest, the performing the sequence of hash operations using an output chaining value generated based on performing an inner-key padding and hashing operation using a cryptographic key of the instruction; and performing an outer-key padding and hashing operation using the cryptographic key to generate another output chaining value to be used in generating a final output message digest based on a final input message digest produced using the intermediate message digest, the final output message digest being a resulting authentication code, and wherein the performing the sequence of hash operations and the performing the outer-key padding and hashing operation are performed as part of a single invocation of the instruction.

[0006] According to one aspect, there is provided a computer program product comprising: a set of one or more computer-readable storage media; and program instructions, collectively stored in the set of one or more computer-readable storage media, for causing at least one computing device to perform computer operations including: executing an instruction to generate an authentication code, the executing the instruction including: performing a plurality of operations of the instruction to generate the authentication code, the plurality of operations including: performing a sequence of hash operations on a message obtained using the instruction to generate an intermediate message digest, the performing the sequence of hash operations using an output chaining value generated based on performing an inner-key padding and hashing operation using a cryptographic key of the instruction; and performing an outer-key padding and hashing operation using the cryptographic key to generate another output chaining value to be used in generating a final output message digest based on a final input message digest produced using the intermediate message digest, the final output message digest being a resulting authentication code, and wherein the performing the sequence of hash operations and the performing the outer-key padding and hashing operation are performed as part of a single invocation of the instruction.

[0007] According to another aspect, there is provided a computer system comprising: at least one computing device; a set of one or more computer-readable storage media; and program instructions, collectively stored in the set of one or more computer-readable storage media, for causing the at least one computing device to perform computer operations including: executing an instruction to generate an authentication code, the executing the instruction including: performing a plurality of operations of the instruction to generate the authentication code, the plurality of operations including: performing a sequence of hash operations on a message obtained using the instruction to generate an intermediate message digest, the performing the sequence of hash operations using an output chaining value generated based on performing an inner-key padding and hashing operation using a cryptographic key of the instruction; and performing an outer-key padding and hashing operation using the cryptographic key to generate another output chaining value to be used in generating a final output message digest based on a final input message digest produced using the intermediate message digest, the final output message digest being a resulting authentication code, and wherein the performing the sequence of hash operations and the performing the outer-key padding and hashing operation are performed as part of a single invocation of the instruction.

[0008] According to another aspect, there is provided a computer-implemented method comprising: executing an instruction to generate an authentication code, the executing the instruction including: performing a plurality of operations of the instruction to generate the authentication code, the plurality of operations including: performing a sequence of hash operations on a message obtained using the instruction to generate an intermediate message digest, the performing the sequence of hash operations using an output chaining value generated based on performing an inner-key padding and hashing operation using a cryptographic key of the instruction; and performing an outer-key padding and hashing operation using the cryptographic key to generate another output chaining value to be used in generating a final output message digest based on a final input message digest produced using the intermediate message digest, the final output message digest being a resulting authentication code, and wherein the performing the sequence of hash operations and the performing the outer-key padding and hashing operation are performed as part of a single invocation of the instruction.

[0009] According to another aspect, there is provided a computer system comprising: at least one hardware accelerator to be used in executing an instruction to generate an authentication code, the executing the instruction including: performing a plurality of operations of the instruction to generate the authentication code, the plurality of operations including: performing a sequence of hash operations on a message obtained using the instruction to generate an intermediate message digest, the performing the sequence of hash operations using an output chaining value generated based on performing an inner-key padding and hashing operation using a cryptographic key of the instruction; and performing an outer-key padding and hashing operation using the cryptographic key to generate another output chaining value to be used in generating a final output message digest based on a final input message digest produced using the intermediate message digest, the final output message digest being a resulting authentication code, and wherein the performing the sequence of hash operations and the performing the outer-key padding and hashing operation are performed as part of a single invocation of the instruction.

[0010] According to another aspect, there is provided a computer-implemented method comprising: executing an instruction, using at least one hardware accelerator, to generate an authentication code, the executing the instruction including: performing a plurality of operations of the instruction to generate the authentication code, the plurality of operations including: performing a sequence of hash operations on a message obtained using the instruction to generate an intermediate message digest, the performing the sequence of hash operations using an output chaining value generated based on performing an inner-key padding and hashing operation using a cryptographic key of the instruction; and performing an outer-key padding and hashing operation using the cryptographic key to generate another output chaining value to be used in generating a final output message digest based on a final input message digest produced using the intermediate message digest, the final output message digest being a resulting authentication code, and wherein the performing the sequence of hash operations and the performing the outer-key padding and hashing operation are performed as part of a single invocation of the instruction.

[0011] Computer-implemented methods, computer systems and computer program products relating to one or more aspects are described and claimed herein. Each of the embodiments of the computer program product may be embodiments of each computer system and/or each computer-implemented method and vice-versa. Further, each of the embodiments is separable and optional from one another. Moreover, embodiments may be combined with one another. Each of the embodiments of the computer program product may be combinable with aspects and/or embodiments of each computer system and/or computer-implemented method, and vice-versa. Further, services relating to one or more aspects are also described and may be claimed herein.

[0012] Additional features and advantages are realized through the techniques described herein. Other embodiments and aspects are described in detail herein and are considered a part of the claimed aspects.

BRIEF DESCRIPTION OF THE DRAWINGS

[0013] One or more preferred embodiments of the present invention will now be described, by way of example only, and with reference to the following drawings:

FIG. 1 depicts one example of a computing environment to incorporate and use one or more aspects of the present disclosure;

FIG. 2 depicts one example of further details of a processor of the processor set of FIG. 1, in accordance with one or more aspects of the present disclosure;

FIG. 3A depicts one example of hash-based message authentication code block processing;

FIG. 3B depicts further details of one example of hash-based message authentication code block processing;

FIG. 4A depicts one example of hash-based message authentication code processing code of FIG. 1, in accordance with one or more aspects of the present disclosure;

FIG. 4B depicts one example of the execute instruction code of the hash-based message authentication code processing code of FIG. 4A, in accordance with one or more aspects of the present disclosure;

FIG. 5A depicts one example of a format of a Compute Message Authentication Code instruction, in accordance with one or more aspects of the present disclosure;

FIGS. 5B-5C depict examples of contents of registers used by the Compute Message Authentication Code instruction of FIG. 5A, in accordance with one or more aspects of the present disclosure;

FIGS. 5D-5E depict examples of contents of general registers used by the Compute Message Authentication Code instruction of FIG. 5A, in accordance with one or more aspects of the present disclosure;

FIG. 5F depicts one example of a parameter block used by the Compute Message Authentication Code instruction of FIG. 5A, in accordance with one or more aspects of the present disclosure;

FIG. 5G depicts another example of a parameter block used by the Compute Message Authentication Code instruction of FIG. 5A, in accordance with one or more aspects of the present disclosure;

FIG. 6A depicts one example of operations of hash-based message authentication code processing, in accordance with one or more aspects of the present disclosure;

FIG. 6B depicts one example of operations of hash-based message authentication code processing for an empty block using a selected input message bit length, in accordance with one or more aspects of the present disclosure;

FIG. 6C depicts one example of operations of hash-based message authentication code processing for a partial block using another selected input message bit length, in accordance with one or more aspects of the present disclosure;

FIG. 6D depicts another example of operations of hash-based message authentication code processing for a partial block using a particular input message bit length, in accordance with one or more aspects of the present disclosure;

FIG. 6E depicts another example of operations of hash-based message authentication code processing for a partial block using a selected output message bit length, in accordance with one or more aspects of the present disclosure;

FIG. 7 depicts one example of hash-based message authentication code processing, in accordance with one or more aspects of the present disclosure;

FIG. 8 depicts an overview of using hardware to perform processing of a compute message authentication code instruction, in accordance with one or more aspects of the present disclosure;

FIG. 9 depicts one example of hardware to be used to perform hash-based message authentication code processing, in accordance with one or more aspects of the present disclosure;

FIG. 10 depicts one example of a SHA-2 engine used in accordance with one or more aspects of the present disclosure;

FIG. 11 depicts one example of a state diagram, in accordance with one or more aspects of the present disclosure; and

FIGS. 12A-12B depict another example of a computing environment to incorporate and use one or more aspects of the present disclosure.

DETAILED DESCRIPTION

[0014] In accordance with one or more aspects of the present disclosure, a capability is provided to facilitate processing within a computing environment by, for instance, accelerating hash-based message authentication code processing. In one or more aspects, hash-based message authentication code processing is accelerated by providing an instruction (e.g., a single architected instruction) to perform the hash-based message authentication code processing using parameters of the instruction. Further, in one or more aspects, hash-based message authentication code processing is accelerated by allowing the instruction to be interrupted and then resumed from where it was interrupted using saved state of the instruction. This is in contrast to repeating the hash-based message authentication code processing or having to separately determine, external to the instruction, where the hash-based message authentication code processing was interrupted and where it should be resumed.

[0015] In one or more aspects, a single instruction (e.g., a compute message authentication code instruction or other instruction) is provided that encodes parameters to be used for hash-based message authentication code processing. The single instruction is interruptible and includes the state to be used to resume hash-based message authentication code processing without compromising security. The single instruction is executed in hardware (e.g., using at least one hardware accelerator), in one example. In one or more aspects, interruptible hash-based message authentication code processing is implemented which saves state information, such as, e.g., a chaining value, to be used to resume interrupted processing.

[0016] In one or more aspects, a format of the instruction (e.g., compute message authentication code instruction) includes one or more parameters, such as, for instance: one or more keys (e.g., a cryptographic key), a message address, a message length, one or more control indicators (also referred to as flags), and/or state information. Additional, fewer and/or other parameters may be used. The instruction uses state information (e.g., a chaining or sequencing value) to allow interruption and resuming of the hash-based message authentication code processing.

[0017] In one or more aspects, hash-based message authentication code processing is accelerated by using, e.g., at least one hardware accelerator that is able to perform a plurality of operations of the instruction and/or the hash-based message authentication code processing. For instance, the at least one hardware accelerator is configured to: schedule various operations in hardware in a way to allow interruption and resuming of operations; and to maintain state information (e.g., chaining state information) in the hardware that can be read by firmware, enabling interruption and resuming of the processing.

[0018] Firmware includes, e.g., the microcode or millicode of a processor. It includes, for instance, the hardware-level instructions and/or data structures used in implementation of higher-level machine code. In one embodiment, it includes, for instance, proprietary code that is typically delivered as microcode or millicode that includes trusted software, microcode or millicode specific to the underlying hardware and controls operating system access to the system hardware.

[0019] In one or more aspects, a computer program product is provided. The computer program product includes a set of one or more computer-readable storage media and program instructions, collectively stored in the set of one or more computer-readable storage media, for causing at least one computing device to perform computer operations. The computer operations include executing an instruction to generate an authentication code. Executing the instruction includes performing a plurality of operations of the instruction to generate the authentication code. The plurality of operations includes performing a sequence of hash operations on a message obtained using the instruction to generate an intermediate message digest. The performing the sequence of hash operations uses an output chaining value generated based on performing an inner-key padding and hashing operation using a cryptographic key of the instruction. An outer-key padding and hashing operation is performed using the cryptographic key to generate another output chaining value to be used in generating a final output message digest based on a final input message digest produced using the intermediate message digest. The final output message digest is a resulting authentication code. The performing the sequence of hash operations and the performing the outer-key padding and hashing operation are performed as part of a single invocation of the instruction. Using a single instruction to perform a plurality of operations to generate an authentication code improves processing within the computing environment and reduces latency. Performance is higher and software processing overhead is reduced.

[0020] Additionally, or alternatively, in one or more embodiments, the plurality of operations further includes storing at least one chaining value generated based on execution of the instruction. A chaining value of the at least one chaining value is to be used in re-execution of the instruction based on execution of the instruction being interrupted. Saving the chaining value allows the instruction to be interrupted and re-executed without repeating previous processing, thereby improving processing. Further, by enabling the instruction to be interrupted, higher priority processes may be serviced, providing high responsiveness. Further, latency is reduced.

[0021] Additionally, or alternatively, in one or more embodiments, the computer operations further include determining that the instruction has been interrupted, and re-executing the instruction using the chaining value. Enabling the instruction to be interrupted, higher priority processes may be serviced, providing high responsiveness. Further, latency is reduced.

[0022] Additionally, or alternatively, in one or more embodiments, the performing the plurality of operations further includes performing the inner-key padding and hashing operation using the cryptographic key to generate the output chaining value. The performing the inner-key padding and hashing operation includes producing an inner-key based on performing a selected operation with the cryptographic key and an inner padding value, generating the output chaining value for the inner-key using a hash operation and an input chaining value, and storing the output chaining value that is generated in a parameter block that is input to the instruction. By performing the inner-key padding and hashing operation to generate the output chaining value as part of executing the single instruction, performance is improved.

[0023] Additionally, or alternatively, in one or more embodiments, the selected operation is an exclusive OR operation. Use of the exclusive OR operation to produce the inner-key facilitates processing and improves performance by enabling the instruction to generate a chaining value that may be used in executing other operations of the instruction, as well as in re-execution of the instruction based on the instruction being interrupted.

[0024] Additionally, or alternatively, in one or more embodiments, the performing the sequence of hash operations on the message obtained using the instruction includes processing a plurality of message blocks of the message. The processing the plurality of message blocks includes performing a plurality of block digest hash operations on the plurality of message blocks using the output chaining value as input to the processing of the plurality of message blocks to obtain the intermediate message digest. By using a single instruction to perform the sequence of operations rather than chaining back-to-back accelerator calls to perform the sequence of operations, performance is improved, and latency and overhead are reduced.

[0025] Additionally, or alternatively, in one or more embodiments, the performing the plurality of operations further includes performing an input message padding and hashing operation for the message. The performing the input message padding and hashing operation for the message includes performing a padding operation on a final message block of the message to produce a padded input message block, and performing a hash operation, using the intermediate digest, on the padded input message block to generate the final input message digest. By performing the input message padding and hashing operation for the message as part of executing the single instruction, performance is improved and overhead and latency are reduced.

[0026] Additionally, or alternatively, in one or more embodiments, the performing the outer-key padding and hashing operation includes producing an outer-key based on performing the selected operation with the cryptographic key and an outer padding value, and generating the another output chaining value for the outer-key using a selected hash operation and the input chaining value. By performing the outer-key padding and hashing operation to generate the other output chaining value as part of executing the single instruction, performance is improved.

[0027] Additionally, or alternatively, in one or more embodiments, the performing the plurality of operations further includes performing an output message padding and hashing operation. The performing the output message padding and hashing operation includes performing a final padding operation on the final input message digest to produce a padded output message block, and performing a final hashing operation, using the another output chaining value, on the padded output message block to generate the final output message digest. By performing the output message padding and hashing operation as part of executing the single instruction, performance is improved and overhead and latency are reduced.

[0028] Additionally, or alternatively, in one or more embodiments, the instruction includes a plurality of parameters to be used in performing the plurality of operations, the plurality of parameters being specified using one or more registers of the instruction. By using parameters of the instruction to perform the plurality of

operations, performance is improved, and latency is reduced by not requiring memory accesses to obtain information used by the instruction.

[0029] Additionally, or alternatively, in one or more embodiments, the plurality of parameters includes at least one chaining value, an input message bit length, and the cryptographic key. The at least one chaining value is used in re-execution of the instruction based on the instruction being interrupted. Use of a single instruction and a parameter set of the instruction enable lower software processing overhead and higher performance.

[0030] Additionally, or alternatively, in one or more embodiments, the instruction further includes as input an address of the message and one or more control indicators to control re-execution of the instruction based on the instruction being interrupted. Performance is improved and processing is facilitated by providing as inputs to the instruction one or more flags that may be used to control execution, as well as re-execution of the instruction. The use of the controls allows hardware to use software-provided parameters to resume a previously interrupted instruction to improve system responsiveness and lower latency.

[0031] Additionally, or alternatively, in one or more embodiments, at least the performing the sequence of hash operations on the message obtained using the instruction uses a hardware accelerator. Use of hardware improves performance. A hardware and firmware co-design enables computation of a hash-based message authentication code synchronously within a processor core pipeline, lowering latency.

[0032] Additionally, or alternatively, in one or more embodiments, the plurality of operations further includes performing the inner-key padding and hashing operation to generate the output chaining value, performing an input message padding and hashing operation for the message using the intermediate message digest to generate the final input message digest, and performing an output message padding and hashing operation using the final input message digest and the another output chaining value to produce the resulting authentication code. By performing the inner-key padding and hashing operation to generate the output chaining value, the input message padding and hashing operation for the message and the output message padding and hashing operation as part of executing the single instruction, performance is improved and overhead and latency are reduced.

[0033] In accordance with one or more aspects, each of the embodiments is separable and optional from one another. Further, embodiments may be combined with one another.

[0034] In one or more aspects, a computer system is provided. The computer system includes, for instance, at least one computing device, a set of one or more computer-readable storage media, and program instructions, collectively stored in the set of one or more computer-readable storage media, for causing the at least one computing device to perform computer operations. The computer operations include executing an instruction to generate an authentication code. Executing the instruction includes performing a plurality of operations of the instruction to generate the authentication code. The plurality of operations include performing a sequence of hash operations on a message obtained using the instruction to generate an intermediate message digest. The

performing the sequence of hash operations uses an output chaining value generated based on performing an inner-key padding and hashing operation using a cryptographic key of the instruction. An outer-key padding and hashing operation is performed using the cryptographic key to generate another output chaining value to be used in generating a final output message digest based on a final input message digest produced using the intermediate message digest. The final output message digest is a resulting authentication code. The performing the sequence of hash operations and the performing the outer-key padding and hashing operation are performed as part of a single invocation of the instruction. Using a single instruction to perform a plurality of operations to generate an authentication code improves processing within the computing environment and reduces latency. Performance is higher and software processing overhead is reduced.

[0035] Additionally, or alternatively, in one or more embodiments, the performing the plurality of operations further includes storing at least one chaining value generated based on execution of the instruction. The computer operations further include determining that the instruction has been interrupted, and re-executing the instruction using a chaining value of the at least one chaining value. Saving the chaining value allows the instruction to be interrupted and re-executed without repeating previous processing, thereby improving processing. Further, by enabling the instruction to be interrupted, higher priority processes may be serviced, providing high responsiveness. Further, latency is reduced.

[0036] Additionally, or alternatively, in one or more embodiments, the performing the plurality of operations further includes performing the inner-key padding and hashing operation using the cryptographic key to generate the output chaining value. The performing the inner-key padding and hashing operation includes producing an inner-key based on performing a selected operation with the cryptographic key and an inner padding value, generating the output chaining value for the inner-key using a hash operation and an input chaining value, and storing the output chaining value that is generated in a parameter block that is input to the instruction. By performing the inner-key padding and hashing operation to generate the output chaining value as part of executing the single instruction, performance is improved.

[0037] Additionally, or alternatively, in one or more embodiments, the performing the plurality of operations further includes performing an input message padding and hashing operation for the message. The performing the input message padding and hashing operation for the message includes performing a padding operation on a final message block of the message to produce a padded input message block, and performing a hash operation, using the intermediate message digest, on the padded input message block to generate the final input message digest. By performing the input message padding and hashing operation for the message as part of executing the single instruction, performance is improved and overhead and latency are reduced.

[0038] In accordance with one or more aspects, each of the embodiments is separable and optional from one another. Further, embodiments may be combined with one another.

[0039] In one or more aspects, a computer-implemented method is provided. The computer-implemented method includes, for instance, executing an instruction to generate an authentication code. Executing the instruction includes performing a plurality of operations of the instruction to generate the authentication code. The plurality of operations include performing a sequence of hash operations on a message obtained using the instruction to generate an intermediate message digest. The performing the sequence of hash operations uses an output chaining value generated based on performing an inner-key padding and hashing operation using a cryptographic key of the instruction. An outer-key padding and hashing operation is performed using the cryptographic key to generate another output chaining value to be used in generating a final output message digest based on a final input message digest produced using the intermediate message digest. The final output message is a resulting authentication code. The performing the sequence of hash operations and the performing the outer-key padding and hashing operation are performed as part of a single invocation of the instruction. Using a single instruction to perform a plurality of operations to generate an authentication code improves processing within the computing environment and reduces latency. Performance is higher and software processing overhead is reduced.

[0040] Additionally, or alternatively, in one or more embodiments, the performing the plurality of operations further includes storing at least one chaining value generated based on execution of the instruction. The computer operations further include determining that the instruction has been interrupted, and re-executing the instruction using a chaining value of the at least one chaining value. Saving the chaining value allows the instruction to be interrupted and re-executed without repeating previous processing, thereby improving processing. Further, by enabling the instruction to be interrupted, higher priority processes may be serviced, providing high responsiveness. Further, latency is reduced.

[0041] Additionally, or alternatively, in one or more embodiments, the plurality of operations further includes performing the inner-key padding and hashing operation using the cryptographic key to generate the output chaining value. The performing the inner-key padding and hashing operation includes producing an inner-key based on performing a selected operation with the cryptographic key and an inner padding value, generating the output chaining value for the inner-key using a hash operation and an input chaining value, and storing the output chaining value that is generated in a parameter block that is input to the instruction. By performing the inner-key padding and hashing operation to generate the output chaining value as part of executing the single instruction, performance is improved.

[0042] Additionally, or alternatively, in one or more embodiments, the performing the plurality of operations further includes performing an input message padding and hashing operation for the message. The performing the input message padding and hashing operation for the message includes performing a padding operation on a final message block to produce a padded input message block of the message, and performing a hash operation, using the intermediate digest, on the padded input message block to generate the final input message digest. By performing the input message padding and hashing operation for the message as part of executing the single instruction, performance is improved and overhead and latency are reduced.

[0043] Additionally, or alternatively, in one or more embodiments, the performing the plurality of operations further includes performing an output message padding and hashing operation. The performing the output message padding and hashing operation includes performing a final padding operation on the final input message digest to produce a padded output message block, and performing a final hashing operation, using the another output chaining value, on the padded output message block to generate the final output message digest. By performing the output message padding and hashing operation as part of executing the single instruction, performance is improved and overhead and latency are reduced.

[0044] In accordance with one or more aspects, each of the embodiments is separable and optional from one another. Further, embodiments may be combined with one another.

[0045] In one or more aspects, a computer system is provided. The computer system includes, for instance, at least one hardware accelerator to be used in executing an instruction to generate an authentication code. The executing the instruction includes performing a plurality of operations of the instruction to generate the authentication code. The plurality of operations includes performing a sequence of hash operations on a message obtained using the instruction to generate an intermediate message digest. The performing the sequence of hash operations uses an output chaining value generated based on performing an inner-key padding and hashing operation using a cryptographic key of the instruction. An outer-key padding and hashing operation is performed using the cryptographic key to generate another output chaining value to be used in generating a final output message digest based on a final input message digest produced using the intermediate message digest. The final output message is a resulting authentication code. The performing the sequence of hash operations and the performing the outer-key padding and hashing operation are performed as part of a single invocation of the instruction. Using a single instruction to perform a plurality of operations to generate an authentication code improves processing within the computing environment and reduces latency. Performance is higher and software processing overhead is reduced. Use of hardware improves performance. A hardware and firmware co-design enables computation of a hash-based message authentication code synchronously within a processor core pipeline, lowering latency.

[0046] In accordance with one or more aspects, each of the embodiments is separable and optional from one another. Further, embodiments may be combined with one another.

[0047] In one or more aspects, a computer-implemented method is provided. The computer-implemented method includes, for instance, executing an instruction, using at least one hardware accelerator, to generate an authentication code. The executing the instruction includes performing a plurality of operations of the instruction to generate the authentication code. The plurality of operations include performing a sequence of hash operations on a message obtained using the instruction to generate an intermediate message digest. The performing the sequence of hash operations uses an output chaining value generated based on performing an inner-key padding and hashing operation using a cryptographic key of the instruction. An outer-key padding and hashing operation is

performed using the cryptographic key to generate another output chaining value to be used in generating a final output message digest based on a final input message digest produced using the intermediate message digest. The final output message digest is a resulting authentication code. The performing the sequence of hash operations and the performing the outer-key padding and hashing operation are performed as part of a single invocation of the instruction. Using a single instruction to perform a plurality of operations to generate an authentication code improves processing within the computing environment and reduces latency. Performance is higher and software processing overhead is reduced. Use of hardware improves performance. A hardware and firmware co-design enables computation of a hash-based message authentication code synchronously within a processor core pipeline, lowering latency.

[0048] In accordance with one or more aspects, each of the embodiments is separable and optional from one another. Further, embodiments may be combined with one another.

[0049] In one or more aspects, a computer program product is provided. The computer program product includes a set of one or more computer-readable storage media and program instructions, collectively stored in the set of one or more computer-readable storage media, for causing at least one computing device to perform computer operations. The computer operations include executing an instruction to generate an authentication code. Executing the instruction includes performing a plurality of operations of the instruction to generate the authentication code. The plurality of operations includes performing a sequence of hash operations on a message obtained using the instruction to generate an intermediate message digest. The performing the sequence of hash operations uses an output chaining value generated based on performing an inner-key padding and hashing operation using a cryptographic key of the instruction. An outer-key padding and hashing operation is performed using the cryptographic key to generate another output chaining value to be used in generating a final output message digest based on a final input message digest produced using the intermediate message digest. The final output message digest is a resulting authentication code. The performing the sequence of hash operations and the performing the outer-key padding and hashing operation are performed as part of a single invocation of the instruction. The instruction includes a plurality of parameters to be used in performing the plurality of operations, the plurality of parameters being specified using one or more registers of the instruction. The plurality of parameters includes at least one chaining value, an input message bit length, and the cryptographic key. The at least one chaining value is used in re-execution of the instruction based on the instruction being interrupted. The instruction further includes as input an address of the message and one or more control indicators to control re-execution of the instruction based on the instruction being interrupted. The plurality of operations further includes performing the inner-key padding and hashing operation to generate the output chaining value, performing an input message padding and hashing operation for the message using the intermediate message digest to generate the final input message digest, and performing an output message padding and hashing operation using the final input message digest and the another output chaining value to produce the resulting authentication code. Using a single instruction to perform a plurality of operations to generate an authentication code improves processing within the computing environment and reduces latency. Performance is higher and software processing overhead is reduced.

By using parameters of the instruction to perform the plurality of operations, performance is improved, and latency is reduced by not requiring memory accesses to obtain information used by the instruction. Use of a single instruction and a parameter set of the instruction enable lower software processing overhead and higher performance. The use of the controls allows hardware to use software-provided parameters to resume a previously interrupted instruction to improve system responsiveness and lower latency. Performance is improved and processing is facilitated by providing as inputs to the instruction one or more flags that may be used to control execution, as well as re-execution of the instruction. By performing the inner-key padding and hashing operation to generate the output chaining value, the input message padding and hashing operation for the message and the output message padding and hashing operation as part of executing the single instruction, performance is improved and overhead and latency are reduced.

[0050] Computer-implemented methods, computer systems and computer program products relating to one or more aspects are described and claimed herein. Each of the embodiments of the computer program product may be embodiments of each computer system and/or each computer-implemented method and vice-versa. Further, each of the embodiments is separable and optional from one another. Moreover, embodiments may be combined with one another. Each of the embodiments of the computer program product may be combinable with aspects and/or embodiments of each computer system and/or computer-implemented method, and vice-versa.

[0051] One or more aspects of the present disclosure are incorporated in, performed and/or used by a computing environment. As examples, the computing environment may be of various architectures and of various types, including, but not limited to: personal computing, client-server, distributed, virtual, emulated, partitioned, non-partitioned, cloud-based, quantum, grid, time-sharing, cluster, peer-to-peer, wearable, mobile, having one node or multiple nodes, having one processor or multiple processors, and/or any other type of environment and/or configuration, etc. that is capable of executing a process (or multiple processes) that performs cryptographic processing including accelerated and/or interruptible hash-based message authentication code processing and/or one or more other aspects of the present disclosure. Aspects of the present disclosure are not limited to a particular architecture or environment.

[0052] Various aspects of the present disclosure are described by narrative text, flowcharts, block diagrams of computer systems and/or block diagrams of the machine logic included in computer program product (CPP) embodiments. With respect to any flowcharts, depending upon the technology involved, the operations can be performed in a different order than what is shown in a given flowchart. For example, again depending upon the technology involved, two operations shown in successive flowchart blocks may be performed in reverse order, as a single integrated step, concurrently, or in a manner at least partially overlapping in time.

[0053] A computer program product embodiment ("CPP embodiment" or "CPP") is a term used in the present disclosure to describe any set of one, or more, storage media (also called "mediums") collectively included in a set of one, or more, storage devices that collectively include machine readable code corresponding to instructions

and/or data for performing computer operations specified in a given CPP claim. A "storage device" is any tangible device that can retain and store instructions for use by a computer processor. Without limitation, the computer-readable storage medium may be an electronic storage medium, a magnetic storage medium, an optical storage medium, an electromagnetic storage medium, a semiconductor storage medium, a mechanical storage medium, or any suitable combination of the foregoing. Some known types of storage devices that include these mediums include: diskette, hard disk, random access memory (RAM), read-only memory (ROM), erasable programmable read-only memory (EPROM or Flash memory), static random access memory (SRAM), compact disc read-only memory (CD-ROM), digital versatile disk (DVD), memory stick, floppy disk, mechanically encoded device (such as punch cards or pits/lands formed in a major surface of a disc) or any suitable combination of the foregoing. A computer-readable storage medium, as that term is used in the present disclosure, is not to be construed as storage in the form of transitory signals per se, such as radio waves or other freely propagating electromagnetic waves, electromagnetic waves propagating through a waveguide, light pulses passing through a fiber optic cable, electrical signals communicated through a wire, and/or other transmission media. As will be understood by those of skill in the art, data is typically moved at some occasional points in time during normal operations of a storage device, such as during access, de-fragmentation or garbage collection, but this does not render the storage device as transitory because the data is not transitory while it is stored.

[0054] One example of a computing environment to perform, incorporate and/or use one or more aspects of the present disclosure is described with reference to FIG. 1. In one example, a computing environment 100 contains an example of an environment for the execution of at least some of the computer code involved in performing the inventive methods, such as accelerated hash-based message authentication code processing code 150 (also referred to herein as block 150). In addition to block 150, computing environment 100 includes, for example, computer 101, wide area network (WAN) 102, end user device (EUD) 103, remote server 104, public cloud 105, and private cloud 106. In this embodiment, computer 101 includes processor set 110 (including processing circuitry 120 and cache 121), communication fabric 111, volatile memory 112, persistent storage 113 (including operating system 122 and block 150, as identified above), peripheral device set 114 (including user interface (UI) device set 123, storage 124, and Internet of Things (IoT) sensor set 125), and network module 115. Remote server 104 includes remote database 130. Public cloud 105 includes gateway 140, cloud orchestration module 141, host physical machine set 142, virtual machine set 143, and container set 144.

[0055] Computer 101 may take the form of a desktop computer, laptop computer, tablet computer, smart phone, smart watch or other wearable computer, mainframe computer, quantum computer or any other form of computer or mobile device now known or to be developed in the future that is capable of running a program, accessing a network or querying a database, such as remote database 130. As is well understood in the art of computer technology, and depending upon the technology, performance of a computer-implemented method may be distributed among multiple computers and/or between multiple locations. On the other hand, in this presentation of computing environment 100, detailed discussion is focused on a single computer, specifically computer 101, to keep the presentation as simple as possible. Computer 101 may be located in a cloud, even though it is not shown

in a cloud in FIG. 1. On the other hand, computer 101 is not required to be in a cloud except to any extent as may be affirmatively indicated.

[0056] Processor set 110 includes one, or more, computer processors of any type now known or to be developed in the future. Processing circuitry 120 may be distributed over multiple packages, for example, multiple, coordinated integrated circuit chips. Processing circuitry 120 may implement multiple processor threads and/or multiple processor cores. Cache 121 is memory that is located in the processor chip package(s) and is typically used for data or code that should be available for rapid access by the threads or cores running on processor set 110. Cache memories are typically organized into multiple levels depending upon relative proximity to the processing circuitry. Alternatively, some, or all, of the cache for the processor set may be located "off chip." In some computing environments, processor set 110 may be designed for working with qubits and performing quantum computing.

[0057] Computer-readable program instructions are typically loaded onto computer 101 to cause a series of operational steps to be performed by processor set 110 of computer 101 and thereby effect a computer-implemented method, such that the instructions thus executed will instantiate the methods specified in flowcharts and/or narrative descriptions of computer-implemented methods included in this document (collectively referred to as "the inventive methods"). These computer-readable program instructions are stored in various types of computer-readable storage media, such as cache 121 and the other storage media discussed below. The program instructions, and associated data, are accessed by processor set 110 to control and direct performance of the inventive methods. In computing environment 100, at least some of the instructions for performing the inventive methods may be stored in block 150 in persistent storage 113.

[0058] Communication fabric 111 is the signal conduction paths that allow the various components of computer 101 to communicate with each other. Typically, this fabric is made of switches and electrically conductive paths, such as the switches and electrically conductive paths that make up buses, bridges, physical input/output ports and the like. Other types of signal communication paths may be used, such as fiber optic communication paths and/or wireless communication paths.

[0059] Volatile memory 112 is any type of volatile memory now known or to be developed in the future. Examples include dynamic type random access memory (RAM) or static type RAM. Typically, volatile memory 112 is characterized by random access, but this is not required unless affirmatively indicated. In computer 101, the volatile memory 112 is located in a single package and is internal to computer 101, but, alternatively or additionally, the volatile memory may be distributed over multiple packages and/or located externally with respect to computer 101.

[0060] Persistent storage 113 is any form of non-volatile storage for computers that is now known or to be developed in the future. The non-volatility of this storage means that the stored data is maintained regardless of whether power is being supplied to computer 101 and/or directly to persistent storage 113. Persistent storage 113

may be a read only memory (ROM), but typically at least a portion of the persistent storage allows writing of data, deletion of data and re-writing of data. Some familiar forms of persistent storage include magnetic disks and solid state storage devices. Operating system 122 may take several forms, such as various known proprietary operating systems or open source Portable Operating System Interface-type operating systems that employ a kernel. The code included in block 150 typically includes at least some of the computer code involved in performing the inventive methods.

[0061] Peripheral device set 114 includes the set of peripheral devices of computer 101. Data communication connections between the peripheral devices and the other components of computer 101 may be implemented in various ways, such as Bluetooth connections, Near-Field Communication (NFC) connections, connections made by cables (such as universal serial bus (USB) type cables), insertion-type connections (for example, secure digital (SD) card), connections made through local area communication networks and even connections made through wide area networks such as the internet. In various embodiments, UI device set 123 may include components such as a display screen, speaker, microphone, wearable devices (such as goggles and smart watches), keyboard, mouse, printer, touchpad, game controllers, and haptic devices. Storage 124 is external storage, such as an external hard drive, or insertable storage, such as an SD card. Storage 124 may be persistent and/or volatile. In some embodiments, storage 124 may take the form of a quantum computing storage device for storing data in the form of qubits. In embodiments where computer 101 is required to have a large amount of storage (for example, where computer 101 locally stores and manages a large database) then this storage may be provided by peripheral storage devices designed for storing very large amounts of data, such as a storage area network (SAN) that is shared by multiple, geographically distributed computers. IoT sensor set 125 is made up of sensors that can be used in Internet of Things applications. For example, one sensor may be a thermometer and another sensor may be a motion detector.

[0062] Network module 115 is the collection of computer software, hardware, and firmware that allows computer 101 to communicate with other computers through WAN 102. Network module 115 may include hardware, such as modems or Wi-Fi signal transceivers, software for packetizing and/or de-packetizing data for communication network transmission, and/or web browser software for communicating data over the internet. In some embodiments, network control functions and network forwarding functions of network module 115 are performed on the same physical hardware device. In other embodiments (for example, embodiments that utilize software-defined networking (SDN)), the control functions and the forwarding functions of network module 115 are performed on physically separate devices, such that the control functions manage several different network hardware devices. Computer-readable program instructions for performing the inventive methods can typically be downloaded to computer 101 from an external computer or external storage device through a network adapter card or network interface included in network module 115.

[0063] WAN 102 is any wide area network (for example, the internet) capable of communicating computer data over non-local distances by any technology for communicating computer data, now known or to be developed

in the future. In some embodiments, the WAN 102 may be replaced and/or supplemented by local area networks (LANs) designed to communicate data between devices located in a local area, such as a Wi-Fi network. The WAN and/or LANs typically include computer hardware such as copper transmission cables, optical transmission fibers, wireless transmission, routers, firewalls, switches, gateway computers and edge servers.

[0064] End user device (EUD) 103 is any computer system that is used and controlled by an end user (for example, a customer of an enterprise that operates computer 101), and may take any of the forms discussed above in connection with computer 101. EUD 103 typically receives helpful and useful data from the operations of computer 101. For example, in a hypothetical case where computer 101 is designed to provide a recommendation to an end user, this recommendation would typically be communicated from network module 115 of computer 101 through WAN 102 to EUD 103. In this way, EUD 103 can display, or otherwise present, the recommendation to an end user. In some embodiments, EUD 103 may be a client device, such as thin client, heavy client, mainframe computer, desktop computer and so on.

[0065] Remote server 104 is any computer system that serves at least some data and/or functionality to computer 101. Remote server 104 may be controlled and used by the same entity that operates computer 101. Remote server 104 represents the machine(s) that collect and store helpful and useful data for use by other computers, such as computer 101. For example, in a hypothetical case where computer 101 is designed and programmed to provide a recommendation based on historical data, then this historical data may be provided to computer 101 from remote database 130 of remote server 104.

[0066] Public cloud 105 is any computer system available for use by multiple entities that provides on-demand availability of computer system resources and/or other computer capabilities, especially data storage (cloud storage) and computing power, without direct active management by the user. Cloud computing typically leverages sharing of resources to achieve coherence and economies of scale. The direct and active management of the computing resources of public cloud 105 is performed by the computer hardware and/or software of cloud orchestration module 141. The computing resources provided by public cloud 105 are typically implemented by virtual computing environments that run on various computers making up the computers of host physical machine set 142, which is the universe of physical computers in and/or available to public cloud 105. The virtual computing environments (VCEs) typically take the form of virtual machines from virtual machine set 143 and/or containers from container set 144. It is understood that these VCEs may be stored as images and may be transferred among and between the various physical machine hosts, either as images or after instantiation of the VCE. Cloud orchestration module 141 manages the transfer and storage of images, deploys new instantiations of VCEs and manages active instantiations of VCE deployments. Gateway 140 is the collection of computer software, hardware, and firmware that allows public cloud 105 to communicate through WAN 102.

[0067] Some further explanation of virtualized computing environments (VCEs) will now be provided. VCEs can be stored as "images." A new active instance of the VCE can be instantiated from the image. Two familiar

types of VCEs are virtual machines and containers. A container is a VCE that uses operating-system-level virtualization. This refers to an operating system feature in which the kernel allows the existence of multiple isolated user-space instances, called containers. These isolated user-space instances typically behave as real computers from the point of view of programs running in them. A computer program running on an ordinary operating system can utilize all resources of that computer, such as connected devices, files and folders, network shares, CPU power, and quantifiable hardware capabilities. However, programs running inside a container can only use the contents of the container and devices assigned to the container, a feature which is known as containerization.

[0068] Private cloud 106 is similar to public cloud 105, except that the computing resources are only available for use by a single enterprise. While private cloud 106 is depicted as being in communication with WAN 102, in other embodiments a private cloud may be disconnected from the internet entirely and only accessible through a local/private network. A hybrid cloud is a composition of multiple clouds of different types (for example, private, community or public cloud types), often respectively implemented by different vendors. Each of the multiple clouds remains a separate and discrete entity, but the larger hybrid cloud architecture is bound together by standardized or proprietary technology that enables orchestration, management, and/or data/application portability between the multiple constituent clouds. In this embodiment, public cloud 105 and private cloud 106 are both part of a larger hybrid cloud.

[0069] Cloud computing services and/or microservices (not separately shown in FIG. 1): private and public clouds 106, 105 are programmed and configured to deliver cloud computing services and/or microservices (unless otherwise indicated, the word “microservices” shall be interpreted as inclusive of larger “services” regardless of size). Cloud services are infrastructure, platforms, or software that are typically hosted by third-party providers and made available to users through the internet. Cloud services facilitate the flow of user data from front-end clients (for example, user-side servers, tablets, desktops, laptops), through the internet, to the provider’s systems, and back. In some embodiments, cloud services may be configured and orchestrated according to as “as a service” technology paradigm where something is being presented to an internal or external customer in the form of a cloud computing service. As-a-Service offerings typically provide endpoints with which various customers interface. These endpoints are typically based on a set of APIs. One category of as-a-service offering is Platform as a Service (PaaS), where a service provider provisions, instantiates, runs, and manages a modular bundle of code that customers can use to instantiate a computing platform and one or more applications, without the complexity of building and maintaining the infrastructure typically associated with these things. Another category is Software as a Service (SaaS) where software is centrally hosted and allocated on a subscription basis. SaaS is also known as on-demand software, web-based software, or web-hosted software. Four technological sub-fields involved in cloud services are: deployment, integration, on demand, and virtual private networks.

[0070] The computing environment described above is only one example of a computing environment to incorporate, perform and/or use one or more aspects of the present disclosure. Other examples are possible. For instance, in one or more embodiments, one or more of the components/modules/blocks of FIG. 1 are not included

in the computing environment and/or are not used for one or more aspects of the present disclosure. Further, in one or more embodiments, additional and/or other components/modules/blocks may be used. Other variations are possible.

[0071] In one example, a processor (e.g., of processor set 110) includes a plurality of functional components (or a subset thereof) used to execute instructions. As depicted in FIG. 2, in one example, a processor 200 includes, for instance, an instruction fetch component 201 to fetch instructions to be executed; an instruction decode/operand fetch component 202 to decode the fetched instructions and to obtain operands of the decoded instructions; one or more instruction execute components 204 to execute the decoded instructions; a memory access component 206 to access memory for instruction execution, if necessary; and a write back component 208 to provide the results of the executed instructions. One or more of the components may access and/or use one or more registers 210 in instruction processing. Further, one or more of the components may access and/or use hash-based message authentication code processing code 150. Additional, fewer and/or other components may be used in one or more aspects of the present disclosure.

[0072] As indicated, hash-based message authentication code processing is to be performed and, in one or more aspects, such processing is accelerated. In one example, a hash-based message authentication code (HMAC) algorithm or technique includes:

[0073]
$$\text{HMAC}(K,m) = H((K' \oplus \text{opad}) \parallel H((K' \oplus \text{ipad}) \parallel m)),$$

[0074] where H is a cryptographic hash function; K is a confidential key; m is the message to be authenticated; K' is another confidential key, derived from the original key K (e.g., padding K to the right with extra zeros to the input block size of the hash function, or hashing K if it is longer than the block size, as examples); $\parallel$ denotes concatenation; $\oplus$ denotes exclusive OR (XOR); opad is the outer padding (0x5c5c5c...5c5c, one block-long hexadecimal constant); ipad is the inner padding (0x363636...3636, one block-long hexadecimal constant); for HMAC using SHA-256, the key is padded to 64 bytes; and the output per HMAC invocation is 32 bytes/256 bits. Other examples are possible.

[0075] In one example, as depicted in FIG. 3A, the message is broken up into blocks 300 (e.g., $m_0, m_1 \dots m_{n-1}$) and an iterative hash function (h) is performed over the message. In a first pass 310 of the hashing, in one example, an initial chaining value (IV) 312 and $k \oplus \text{ipad}$ 314 are input to a first hash function 316a of the first pass, and in a second pass 320 of the hashing, in one example, an initial chaining value (IV) 322 and $k \oplus \text{opad}$ 324 are input to the first hash function 326a of the second pass. Further, the output of the last hash function 316n of the first pass is input to a second hash function 326b of the second pass along with the output of the first hash function 326a of the second pass.

[0076] As further described in FIG. 3B, in one example, a key 340 is exclusive-ORed 342 with ipad 344 to produce i_key_pad 346. Similarly, a key 350 is exclusive-ORed 352 with opad 354 to produce o_key_pad 356. A

first pass 360 of the hash computation using, e.g., SHA-256 (362) produces an internal hash (hash_sum_1 (370)) derived from the inner-key (e.g., i_key_pad 346) and the message 366. The internal hash (hash_sum_1) 370 is input to a second pass 380 of the hash computation which uses SHA-256 (362) to produce a hash-based authentication code (e.g., hash-based message authentication code (HMAC) 390) derived from o_key_pad 356 and hash_sum_1 (370).

[0077] In one example, a SHA-2 accelerator is used to perform the first pass and the second pass. Thus, in one example, back-to-back SHA-2 accelerator calls are chained to perform the passes resulting in a large processing overhead. Further, each small hashing block (e.g., each hash 316a...316n of FIG. 3A) is performed by invocation of an instruction, resulting in increased processing overhead and latency.

[0078] Therefore, in accordance with one or more aspects, a hash-based message authentication code processing capability (also referred to as accelerated hash-based message authentication code processing) is provided in which a single invocation of an instruction is able to perform multiple (and potentially all) hashing blocks during the single invocation. Further, in one or more aspects, the instruction may be interrupted and re-executed, but in such a scenario, state of the instruction (e.g., a chaining value) is saved such that processing may be resumed from where it ended.

[0079] In one or more aspects, the single instruction is configured to perform multiple operations of the hash-based message authentication code processing including, for instance, at least, a sequence of hash operations on the message and an outer-key padding operation. Other operations may also be performed as part of the execution of the single instruction, including, but not limited to, an inner-key padding operation, an input message padding and hashing operation and/or an output message padding and hashing operation. These multiple operations are performed, for instance, as part of a single invocation of the instruction, thereby increasing processing speed and reducing latency. Further, in one or more aspects, one or more of the keys may be wrapped, thereby providing key protection.

[0080] In one or more aspects, the hash-based message authentication code processing uses hash-based message authentication code processing code (e.g., hash-based message authentication code processing code 150) to perform hash-based message authentication code processing, including accelerated and/or interruptible hash-based message authentication code processing. Hash-based message authentication code processing code (e.g., hash-based message authentication code processing code 150) includes code or instructions used to perform hash-based message authentication code processing, including accelerated and/or interruptible hash-based message authentication code processing, and/or perform other tasks, in accordance with one or more aspects of the present disclosure.

[0081] In one example, hash-based message authentication code processing code (e.g., hash-based message authentication code processing code 150) includes code to be used to perform hash-based message authentication code processing, including accelerated and/or interruptible hash-based message authentication code

processing. The code is, e.g., computer-readable program code (e.g., instructions) in computer-readable storage media, e.g., storage (persistent storage 113, cache 121, storage 124, other storage, as examples). The computer-readable storage media may be part of one or more computer program products and the computer-readable program code may be executed by and/or using one or more computing devices (e.g., one or more computers, such as computer(s) 101 and/or other computers; one or more servers, such as remote server(s) 104 and/or other remote servers; one or more devices, such as end user device(s) 103 and/or other end user devices; one or more processors or nodes, such as processor(s) or node(s) of processor set 110 (e.g., processor 200) and/or other processor(s) or node(s); processing circuitry, such as processing circuitry 120 of processor set 110 and/or other processing circuitry; one or more hardware accelerators separate and/or part of one or more processors and/or processing circuitry; and/or other computing devices, etc.). Additional and/or other computers, servers, devices, processors, nodes, processing circuitry, accelerators and/or computing devices may be used to execute the code and/or portions thereof. Many examples are possible.

[0082] One example of hash-based message authentication code processing code 150 is described with reference to FIG. 4A. In one example, hash-based message authentication code processing code 150 includes obtain instruction code 400 to obtain (e.g., receive, be provided, pull, retrieve, fetch, etc.) an instruction, such as a compute message authentication code instruction, to perform hash-based message authentication code processing, including accelerated and/or interruptible hash-based message authentication code processing; and execute instruction code 410 to execute the instruction.

[0083] Further details of execute instruction code 410 are described with reference to FIG. 4B. In one example, execute instruction code 410 includes obtain operands code 412 to obtain one or more operands and/or information of the obtained instruction; perform operations code 416 to perform hash-based message authentication code processing of the instruction, including accelerated and/or interruptible hash-based message authentication code processing; and provide result code 418 to provide a result of the instruction.

[0084] One example of an instruction to perform hash-based message authentication code processing, including accelerated and/or interruptible hash-based message authentication code processing, is a compute message authentication code instruction. In one example, a compute message authentication code instruction, such as a Compute Message Authentication Code instruction, is a single architected hardware machine instruction at the hardware/software interface. As an example, it is part of an instruction set architecture. One example of an instruction set architecture to incorporate and/or use a compute message authentication code instruction, other message authentication instructions, other instructions and/or aspects of the present disclosure is the z/Architecture® instruction set architecture offered by International Business Machines Corporation, Armonk, New York. One embodiment of the z/Architecture instruction set architecture is described in a publication entitled, "z/Architecture Principles of Operation," IBM Publication No. SA22-7832-13, Fourteenth Edition, May 2022, which is hereby incorporated herein by reference in its entirety. The z/Architecture instruction set architecture, however, is only one example architecture; other architectures and/or other types of computing environments of International

Business Machines Corporation and/or of other entities/companies may include and/or use one or more aspects of the present disclosure. z/Architecture and IBM are trademarks or registered trademarks of International Business Machines Corporation in at least one jurisdiction.

[0085] In one example, referring to FIG. 5A, a Compute Message Authentication Code instruction 500 has a format, referred to as a register and register with an extended operation code (opcode) format, having, e.g., 32 bits, and includes, for instance, an operation code field 502 (e.g., bits 0-15); one register field (R_1) 504 (e.g., bits 24-27); and another register field (R_2) 506 (e.g., bits 28-31). Although in this example there is one opcode field 502, in other examples, there may be more than one opcode field. For instance, there may be one opcode field at the beginning of the instruction format and one opcode field at the end of the instruction format. Further, in one example, the R_1 field is ignored; in other examples, the R_1 field is not included. Other examples are also possible.

[0086] In one example, referring to FIGS. 5A and 5B, register field (R_2) 506 specifies a register 520 (R_2) that includes a second operand address 522 of a second operand of the instruction. Referring to FIG. 5C, another register 530 ($R_2 + 1$) includes a length 532 of the second operand.

[0087] In one example, the R_2 field designates an even-odd pair of general registers and is to designate an even-numbered register other than, e.g., general register 0; otherwise, a specification exception is recognized, in one example. In other examples, other types of registers other than general registers may be used. Further, registers other than even-numbered registers may be designated. Many examples are possible.

[0088] In one example, the location of the leftmost byte of the second operand is specified by the contents of the R_2 general register. The number of bytes in the second operand location is specified in, e.g., general register $R_2 + 1$.

[0089] In one example, the second operand length is to be a multiple of the data block size when the designated function is a hash-based message authentication code function and an intermediate input message part flag (described herein) is, e.g., one; otherwise, a specification exception is recognized, in one example. As examples, for function codes 112 (hash-based message authentication code-SHA-224) and 113 (hash-based message authentication code-SHA-256), the data block size is 64 bytes; and for function codes 120 (hash-based message authentication code-encrypted-SHA-224) and 121 (hash-based message authentication code-encrypted-SHA-256), the data block size is 64 bytes. Other data block sizes are possible, as well as other functions/function codes.

[0090] When a hash-based message authentication code function is specified and the intermediate input message part flag is, e.g., zero, the second operand length can have any value, including zero. When a hash-based message authentication code function is specified and the intermediate input message part flag is, e.g., one and the inner-key padding flag is, e.g., initially zero and the second operand length is, e.g., initially zero, a specification exception is recognized, in one example. When a hash-based message authentication code function is specified

and a cryptography counter update pending flag (described herein) is, e.g., one and the second operand length is not, e.g., initially zero, the cryptography counter update pending flag is set to, e.g., zero. When a hash-based message authentication code function is specified and the cryptography counter update pending flag is, e.g., one and the intermediate input message part flag is, e.g., zero and the inner-key padding flag is, e.g., initially zero and the second operand length is, e.g., initially zero, the cryptography counter update pending flag is set to, e.g., zero.

[0091] As part of the operation, the address in general register R_2 is incremented by the number of bytes processed from the second operand, and the length in general register $R_2 + 1$ is decremented by the same number. The formation and updating of the addresses and length is dependent on, for instance, the addressing mode.

[0092] In, for instance, the 24-bit addressing mode, the contents of bit positions 40-63 of general register R_2 constitute the address of the second operand, and the contents of bit positions 0-39 are ignored; bits 40-63 of the updated address replace the corresponding bits in general register R_2 , carries out of, e.g., bit position 40 of the updated address are ignored, and the contents of bit positions 32-39 of general register R_2 are set to, e.g., zeros. In the 31-bit addressing mode, the contents of bit positions 33-63 of general register R_2 constitute the address of the second operand, and the contents of bit positions 0-32 are ignored; bits 33-63 of the updated address replace the corresponding bits in general register R_2 , carries out of, e.g., bit position 33 of the updated address are ignored, and the content of bit position 32 of general register R_2 is set to, e.g., zero. In the 64-bit addressing mode, the contents of bit positions 0-63 of general register R_2 constitute the address of the second operand; bits 0-63 of the updated address replace the contents of general register R_2 , and carries out of, e.g., bit position 0 are ignored. Other examples are possible.

[0093] In both the 24-bit and the 31-bit addressing modes, the contents of bit positions 32-63 of general register $R_2 + 1$ form a 32-bit unsigned binary integer which specifies the number of bytes in the second operand; and the updated value replaces the contents of bit positions 32-63 of general register $R_2 + 1$. In the 64-bit addressing mode, the contents of bit positions 0-63 of general register $R_2 + 1$ form a 64-bit unsigned binary integer which specifies the number of bytes in the second operand; and the updated value replaces the contents of general register $R_2 + 1$.

[0094] In the 24-bit or 31-bit addressing mode, the contents of bit positions 0-31 of general registers R_2 and $R_2 + 1$, remain unchanged, in one example.

[0095] In the access register mode, access registers 1 and R_2 specify the address spaces containing the parameter block and the second operand, respectively.

[0096] Further, in one example, the Compute Message Authentication Code instruction uses multiple implied general registers, such as general register 0 (GR0) and general register 1 (GR1). These registers are referred to as implied registers since they are not explicitly referenced by one or more fields of the instruction; however, they are used by the instruction. Examples of the registers are described below.

[0097] In one example referring to FIG. 5D, a general register 0 (540) includes, for instance:

[0098] Flags 544 (e.g., bits 48-55): In one example, bit positions 48-55 of general register 0 contain an 8-bit flags field controlling an operation of the function. The flags field and/or certain flags (also referred to as control indicators) is/are meaningful for selected function codes of the instruction, as described herein. One example format of the flags field is as follows:

[0099] Inner-Key Padding: In one example, bit 0 of the flags field indicates if the inner-key padding and hashing operation has been performed. The inner-key padding flag is meaningful when the function code in bits 57-63 of general register 0 designates, e.g., a hash-based message authentication code function (function codes 112-115 and 120-123).

[00100] When the inner-key padding flag is, e.g., zero, the inner-key padding and hashing operation has not been performed; otherwise, the inner-key padding and hashing operation has been performed.

[00101] Intermediate Input Message Part: In one example, bit 1 of the flags field indicates if operand 2 contains the intermediate input message part. The intermediate input message part flag is meaningful when, e.g., the function code in bits 57-63 of general register 0 designates a hash-based message authentication code function (e.g., function codes 112-115 and 120-123).

[00102] When the intermediate input message part flag is, e.g., zero, operand 2 contains the last input message part; otherwise, operand 2 contains the intermediate input message part.

[00103] In one example, when the intermediate input message part flag is, e.g., one, the second operand length is to be a multiple of the data block size. When the intermediate input message part flag is, e.g., zero and the last input message block contains a partial input message, the partial message block length (L), used in the final step of SHA padding, is set to the length of the last input message block; otherwise, L is set to, e.g., zero. If the operation ends due to partial completion, a specified condition code (e.g., condition code 3) is set.

[00104] Cryptography Counter Update Pending: In one example, bit 2 of the flags field indicates if a cryptography counter access exception has been detected. The cryptography counter update pending flag is meaningful when the function code in bits 57-63 of general register 0 designates a hash-based message authentication code function (e.g., function codes 112-115 and 120-123) and the second operand length is, e.g., zero. The program should set the cryptography counter update pending flag to, e.g., zero before the first issuance of the instruction and not update it in the subsequent re-drive of the same instruction; otherwise, the hash-based message authentication code algorithm may not be processed correctly.

[00105] When the cryptography counter update pending flag is, e.g., zero, the cryptography counter access exception either has not been detected or the cryptography counter access exception has been resolved; otherwise, the cryptography counter access exception has been detected.

[00106] When the cryptography counter update pending flag is, e.g., one and the second operand length is not, e.g., initially zero, the cryptography counter update pending flag is set to, e.g., zero. When the cryptography counter update pending flag is, e.g., one, the intermediate input message part flag is, e.g., zero, the inner-key padding flag is, e.g., initially zero, and the second operand length is, e.g., initially zero, the cryptography counter update pending flag is set to, e.g., zero. If the operation ends due to partial completion, a specified condition code (e.g., condition code 3) is set.

[00107] Reserved: In one example, bits 3-7 of the flags field are reserved and should contain, e.g., zeros; otherwise, the program may not operate compatibly in the future.

[00108] General 0 further includes, for instance:

[00109] Function code 546 (e.g., bits 57-63): In one example, bit positions 57-63 of general register 0 include the function code that specifies a function to be performed by the Compute Message Authentication Code instruction. The instruction is configured to specify and implement a plurality of functions and respective function codes. Example function codes to be used, in accordance with one or more aspects of the present disclosure, include, for instance, a function code (e.g., function code 112) that specifies a hash-based message authentication code-SHA-224 function; a function code (e.g., function code 113) that specifies a hash-based message authentication code-SHA-256 function; a function code (e.g., function code 120) that specifies a hash-based message authentication code-encrypted-SHA-224 function; and/or a function code (e.g., function code 121) that specifies a hash-based message authentication code-encrypted-SHA-256 function, etc. Although example functions and/or function codes may be specified, additional, fewer and/or other functions/function codes may be specified and/or used. Many examples are possible.

[00110] Further, in one example, one or more selected bits, e.g., bit 56 of general register 0 is to be, e.g., zero; otherwise, a specification exception is recognized, in one example. Bit positions 0-31 of general register 0 are ignored, in one example. When the function code in bits 57-63 of general register 0 of the Compute Message Authentication Code instruction specifies a hash-based message authentication code function (e.g., function code 112-113, 120-121 or other selected function codes) and the specified hash-based message authentication code function code is valid, bits positions 32-47 of general register 0 are reserved and should contain, e.g., zeros; otherwise, the program may not operate compatibly in the future.

[00111] In one example, when, for instance, message-security-assist extension 11 is installed, flags field 544, inner-key padding flag (e.g., bit 0 of the flags field), intermediate input message part flag (e.g., bit 1 of the flags field), and cryptography counter update pending flag (e.g., bit 2 of the flags field) are defined, and function codes

112, 113, 120 and 121 (and other selected function codes) are valid for the Compute Message Authentication Code instruction.

[00112] One example of general register 1 is depicted in FIG. 5E. In one example, a general register 1 (550) includes an address 552 of a parameter block in storage (e.g., memory, storage, etc.). For instance, address 552 is a logical address of, for instance, a leftmost byte of the parameter block in storage. In one example, the location of the address in the general register depends on the addressing mode. For instance, in the 24-bit addressing mode, the contents of bit positions 40-63 of general register 1 constitutes the address, and the contents of bit positions 0-39 are ignored. In the 31-bit addressing mode, the contents of bit positions 33-63 of general register 1 constitute the address and the contents of bit positions 0-32 are ignored. In the 64-bit addressing mode, the contents of bit positions 0-63 of general register 1 constitute the address. In the access register mode, access register 1 specifies the address space containing the parameter block. Other examples are possible.

[00113] One example of a parameter block used by the Compute Message Authentication Code instruction hash-based message authentication code SHA-224/SHA-256 functions is described with reference to FIG. 5F. In one example, a parameter block, e.g., parameter block 560, used by the Compute Message Authentication Code instruction hash-based message authentication code SHA-224/SHA-256 functions includes, for instance:

[00114] Chaining value 562 (called H fields) is in, e.g., byte offsets 0-31 of parameter block 560. In one example, the chaining value is formed by concatenating the H fields (e.g., H0-H7) together in order, starting with H0 on the left and ending with H7 on the right.

[00115] In one example, the initial chaining value (ICV) is the chaining value (CV) in the parameter block which is used as input to the SHA block digest algorithm, an example of which is described herein.

[00116] In one example, each SHA block digest algorithm contains a specific set of constants called the initial hash value ($H^{(0)}$). It is used as the initial chaining value of the first (leftmost) message block of a set of message blocks to be hashed.

[00117] In one example, for SHA-256, the initial hexadecimal chaining value ($H^{(0)}$) constants are listed as follows:

[00118] H0 = 6A09 E667 H1 = BB67 AE85 H2 = 3C6E F372 H3 = A54F F53A

[00119] H4 = 510E 527F H5 = 9B05 688C H6 = 1F83 D9AB H7 = 5BE0 CD19

[00120] In one example, the SHA-224 algorithm is the same as the SHA-256 algorithm, except that the initial chaining value ($H^{(0)}$) constants and the final message digest lengths are different. The program may obtain the SHA-224 message digest using the SHA-256 functions with the following two actions:

[00121] a. The following initial hexadecimal chaining value ($H^{(0)}$) constants for SHA-224 are used:

[00122] $H_0 = C105\ 9ED8$ $H_1 = 367C\ D507$ $H_2 = 3070\ DD17$ $H_3 = F70E\ 5939$

[00123] $H_4 = FFC0\ 0B31$ $H_5 = 6858\ 1511$ $H_6 = 64F9\ 8FA7$ $H_7 = BEFA\ 4FA4$

[00124] b. The 224-bit message digest is obtained by truncating the final message digest to its leftmost 224 bits.

[00125] In one example, an output chaining value (OCV) is the output of the SHA block digest algorithm, which is stored into the chaining value of the parameter block, as described herein, in one example.

[00126] Input message bit length 564 is in, e.g., byte offsets 32-39 of parameter block 560. In one example, input message bit length 564 in parameter block 560 includes the bit length of the total input message to be hashed.

[00127] In one example, if the input message bit length is not, e.g., a multiple of 8, the program is to store, e.g., zeros in the unused bit positions of the last byte of the input message and round-up the input message bit length (IMBL) to a multiple of 8, in one example.

[00128] In one example, an output message bit length includes the bit length of the total output message to be hashed.

[00129] Cryptographic key (K) 568 is in, e.g., byte offsets 40-103.

[00130] In one example, if the program's cryptographic key is smaller than the cryptographic key in the parameter block, the program is to append zeros on the right side of the program's cryptographic key to enlarge it to the same size as the cryptographic key in the parameter block.

[00131] In one example, the program's original cryptographic key is not to be larger than the cryptographic key in the parameter block because $H^{(0)}$ (instead of the chaining value from the parameter block) is used when hashing the inner-padded key. That is, in one example, the program is not to use an appropriate SHA algorithm to reduce the original cryptographic key to the same size as the cryptographic key in the parameter block.

[00132] The parameter block may include additional, fewer and/or other information. Other examples and variations are possible.

[00133] One example of a parameter block used by the Compute Message Authentication Code instruction hash-based message authentication code encrypted SHA-224/SHA-256 functions is described with reference to FIG. 5G. In one example, a parameter block, e.g., parameter block 570, used by the Compute Message

Authentication Code instruction hash-based message authentication code encrypted SHA-224/encrypted SHA-256 functions includes, for instance:

[00134] Chaining value 572 (called H fields) is in, e.g., byte offsets 0-31 of parameter block 570. In one example, the chaining value is formed by concatenating the H fields (e.g., H0-H7) together in order, starting with H0 on the left and ending with H7 on the right.

[00135] Input message bit length 574 is in, e.g., byte offsets 32-39 of parameter block 570. In one example, input message bit length 574 in parameter block 570 includes the bit length of the total input message to be hashed.

[00136] Encrypted cryptographic key (K) 576 is in, e.g., byte offsets 40-103 of parameter block 570. It is a cryptographic key that has been encrypted.

[00137] AES wrapping key verification pattern 578 is in, e.g., byte offsets 104-135 of parameter block 570 and is used to verify the encrypted cryptographic key, in one example.

[00138] The parameter block may include additional, fewer and/or other information. Other examples and variations are possible.

[00139] For the Compute Message Authentication Code-Hash-Based Message Authentication Code (KMAC-HMAC)-Encrypted-SHA-224 and Compute Message Authentication Code-Hash-Based Message Authentication Code (KMAC-HMAC)-Encrypted-SHA-256 functions, the contents of byte offsets 104-135 of the parameter block (AES wrapping key verification pattern 578) are compared with the contents of an AES wrapping-key-verification-pattern register. If they mismatch, the parameter block location remains unchanged, and the operation is completed by setting, e.g., condition code 1. If they match, byte offsets 0-31 of the parameter block contain the chaining value (called H fields), byte offsets 32-39 of the parameter block contain the input-message-bit length (IMBL), and the contents of byte offsets 40-103 of the parameter block are deciphered using the AES wrapping key to obtain the 512-bit cryptographic key (K).

[00140] In operation, a function specified by the function code in general register 0 is performed. As examples herein, the function is a hash-based message authentication code function, such as a Compute Message Authentication Code-Hash-Based Message Authentication Code-SHA2-224/256 function or a Compute Message Authentication Code-Hash-Based Message Authentication Code-Encrypted-SHA-224/256 function. Additional, fewer and/or other functions are also possible including, but not limited to SHA-384 and SHA-512 functions, as well as other functions and/or different functions than described herein. Many examples are possible.

[00141] In one example, for the hash-based message authentication code functions, the inner-key padding and hashing operation is performed if the inner-key padding flag is, e.g., zero, and the full input message blocks

based on the second operand length are processed. If the intermediate input message part flag is, e.g., zero, the SHA padding and hashing operation is performed on the last (partial or empty) input message block, the output chaining value (OCV) is saved, the outer-key padding and hashing operation is performed, the SHA padding and hashing operation is performed on the saved output chaining value, and the result is stored into the chaining value (CV) field of the parameter block. The operation completes with either a selected condition code (e.g., condition code 0 (normal completion)) or a specified condition code (e.g., condition code 3 (partial completion)), as examples.

[00142] For the hash-based message authentication code functions, the result is obtained as if processing starts at the inner-key padding and hashing operation followed by the hashing of the input message blocks based on second operand length from, e.g., left to right, and if the intermediate input message part flag is, e.g., zero, then continuing with the input message padding and hashing operation followed by the outer-key padding and hashing operation and then ending with the output message padding and hashing operation. The authentication operation is ended when the source bytes (e.g., all source bytes) in the second operand have been processed if the intermediate input message part flag is, e.g., one or when the output message padding and hashing operation has been processed if the intermediate input message part flag is, e.g., zero, or when a central processing unit (CPU)-determined number of blocks that is less than the number of blocks of the entire process have been processed.

[00143] The CPU-determined number of blocks depends, for instance, on the model, and may be a different number each time the instruction is executed. The CPU-determined number of blocks is usually, e.g., nonzero. In certain unusual situations, this number may be, e.g., zero, and a specified condition code (e.g., condition code 3) may be set with no progress. However, the central processing unit protects against endless reoccurrence of this no-progress case.

[00144] For the hash-based message authentication code functions, if the central processing unit is enabled to update counters in the cryptography counter set and the appropriate cryptography counter is accessible, then subsequent to performing the inner-key padding and hashing operation and processing the bytes (e.g., all bytes) of the second operand followed by the input message block padding and hashing operation, the outer-key padding and hashing operation, and the output message block padding and hashing operation if the intermediate input message part flag is, e.g., zero, the appropriate cryptography counter is updated.

[00145] In one example, when the initial-chaining-value field overlaps any portion of the second operand, the result in the chaining value field is unpredictable. Normal completion occurs when the authentication operation has ended and, when applicable, a counter in the cryptography counter set has been updated. Partial completion occurs when a CPU-determined number of blocks that is less than the length of the second operand have been processed or when the central processing unit is enabled to update a counter in the cryptography counter set and the appropriate cryptography counter has not yet been updated.

[00146] For the hash-based message authentication code functions, when the operation ends due to normal completion, a selected condition code (e.g., condition code 0) is set, the inner-key padding flag is set to, e.g., one,

the value in $R_2 + 1$ is set to, e.g., zero, and the cryptography counter update pending flag is set to, e.g., zero. When the operation ends due to partial completion, a specified condition code (e.g., condition code 3) is set. In this case, if the central processing unit is enabled to update counters in the cryptography counter set, the inner-key padding and hashing operation has been performed, the second operand (e.g., all of the second operand) has been processed, the input message block padding and hashing operation has been performed, the outer-key padding and hashing operation has been performed, the output message block padding and hashing operation has been performed, and the appropriate counter is not accessible, then the inner-key padding flag is set to, e.g., one, the resulting value placed in general register $R_2 + 1$ is, e.g., zero, and the cryptography counter update pending flag is set to, e.g., one; otherwise the inner-key padding flag is set to, e.g., one if the inner-key padding and hashing operation has been performed, the resulting value placed in general register $R_2 + 1$ is, e.g., nonzero if the second operand (e.g., all of the second operand) has not been processed, or the resulting value placed in general register $R_2 + 1$ is, e.g., zero if the second operand (e.g., all of the second operand) has been processed.

[00147] For the hash-based message authentication code functions, when the second operand length is, e.g., initially zero, the following occurs, in one example: the second operand is not accessed, the parameter block is not accessed if the intermediate input message part flag is, e.g., one, and general registers R_2 and $R_2 + 1$ are not changed. The inner-key padding and hashing operation is performed if the inner-key padding flag is, e.g., zero. If the intermediate input message part flag is, e.g., zero and the cryptography counter update pending flag is, e.g., zero, the empty input message block SHA padding and hashing operation is performed followed by the outer-key padding and hashing operation and the final SHA padding and hashing operation and the result is stored into the chaining value (CV) field of the parameter block. If the central processing unit is not enabled to update counters in the cryptography counter set, then a selected condition code (e.g., condition code 0) is set and the cryptography counter update pending flag is set to, e.g., zero. If the central processing unit is enabled to update counters in the cryptography counter set, and the appropriate counter is accessible, then the appropriate cryptography counter is updated, the selected condition code (e.g., condition code 0) is set, and the cryptography counter update pending flag is set to, e.g., zero. If the central processing unit is enabled to update counters in the cryptography counter set and the appropriate counter is not accessible, then the specified condition code (e.g., condition code 3) is set and the cryptography counter update pending flag is set to, e.g., one if the cryptography counter update pending flag is, e.g., zero, or the condition code remains unchanged and an access exception is recognized for the location of the appropriate cryptography counter if the cryptography counter update pending flag is, e.g., one.

[00148] In one example, a program event recording (PER) storage-alteration event may be recognized for the portion of the parameter block that is stored. A PER zero-address-detection event may be recognized for the second operand location and for the parameter block. For the hash-based message authentication code functions, a PER zero-address-detection event may be recognized for the parameter block even when the second operand length is zero. When PER events are detected for more than one location, it is unpredictable which location is identified in the PER access identification (PAID) and PER ASCE ID (AI).

[00149] As observed by this central processing unit, other central processing units, and channel programs, references to the parameter block and storage operand may be multiple-access references, accesses to these storage locations are not necessarily block-concurrent, and the sequence of these accesses or references is undefined.

[00150] For functions that perform a comparison of the wrapping key verification pattern field in the parameter block with the wrapping key verification pattern register, it is unpredictable whether access exceptions and PER-zero-address-detection events are recognized for the second operand when the comparison results in a mismatch. For the hash-based message authentication code functions, the entire parameter block may be tested for store-type accesses even though part of it may not be stored.

[00151] In one or more aspects, access exceptions may be reported for a larger portion of the second operand than is processed in a single execution of the instruction; however, access exceptions are not recognized for locations beyond the length of the second operand nor for locations more than 4K bytes beyond the current location being processed.

[00152] In one example, when the inner-key padding flag is, e.g., zero (indicating that the inner-key padding and hashing operation has not been performed), the $H^{(0)}$ value of the SHA-224 block digest algorithm is used as the initial chaining value for the Compute Message Authentication Code-Hash-based Message Authentication Code-SHA-224 function and the Compute Message Authentication Code-Hash-based Message Authentication Code-Encrypted SHA-224 function, whereas the $H^{(0)}$ value of the SHA-256 block digest algorithm is used as the initial chaining value for the Compute Message Authentication Code-Hash-based Message Authentication Code-SHA-256 function and the Compute Message Authentication Code-Hash-based Message Authentication Code-Encrypted SHA-226 function.

[00153] When the inner-key padding flag is, e.g., zero (indicating that the inner-key padding and hashing operation has not been performed), $H^{(0)}$ is used (instead of the chaining value from the parameter block) as input to perform the inner-key padding and hashing operation. When the inner-key padding flag is, e.g., one (indicating that the inner-key padding and hashing operation has been performed), the inner-key padding and hashing operation is not performed.

[00154] In one example, when the inner-key padding flag is, e.g., zero (indicating that an inner-key padding and hashing operation has not yet been performed), an inner-key padding and hashing operation is performed. In one example, referring to FIG. 6A, to perform an inner-key padding and hashing operation, the 512-bit (64-byte) cryptographic key (K) 602 obtained, e.g., from the parameter block is exclusive-ORed 604 with an inner pad (ipad) 606 (e.g., 64 bytes of 36 hex) to produce a 64-byte inner-key 608. A 32-byte chaining value 1 (CV_1) 614 is generated for the 64-byte inner-key 608 using the SHA-256 block digest algorithm (bda) 610 with the 32-byte $H^{(0)}$ value 612 and the inner-key padding flag is set to, e.g., one. The generated chaining value 1 (OCV_1) 614, also

called the output chaining value (OCV), is stored into the chaining value (CV) field of a parameter block 618. If the operation ends due to partial completion, a specified condition code (e.g., condition code 3) is set.

[00155] In one example, a 32-byte intermediate message digest (IMD) 636 is generated for the 64-byte full input message blocks (M1, M2, ..., Mn) 620 in operand 2 632 using the SHA-256 block digest algorithm 634 with the 32-byte chaining value (ICV₁) 626 from the parameter block. If the intermediate input message part flag is, e.g., one or L is, e.g., zero, M_{LFB} 628 is the last input message block (Mn); otherwise, M_{LFB} is the input message block that immediately precedes the last (partial) input message block (Mn-1). The generated intermediate message digest (OCV₂) 636, also called the output chaining value (OCV), is stored into the chaining value (CV) field of parameter block 618. This operation repeats until the remaining input message is less than, e.g., 64 bytes or until a CPU-determined number of blocks have been stored. If the operation ends due to partial completion, a specified condition code (e.g., condition code 3) is set.

[00156] In one example, when the intermediate input message part flag is, e.g., one (indicating operand 2 includes the intermediate input message part), the following occurs, in one example: If the central processing unit is not enabled to update counters in the cryptography counter set, then a selected condition code (e.g., condition code 0) is set (indicating normal completion) and the cryptography counter update pending flag is set to, e.g., zero. If the central processing unit is enabled to update counters in the cryptography counter set and the appropriate counter is accessible, then the appropriate counter is updated, a selected condition code (e.g., condition code 0) is set (indicating normal completion), and the cryptography counter update pending flag is set to, e.g., zero. If the central processing unit is enabled to update counters in the cryptography counter set and the appropriate counter is not accessible, then a specified condition code (e.g., condition code 3) is set and the cryptography counter update pending flag is set to, e.g., one if the cryptography counter update pending flag is, e.g., zero, or the condition code remains unchanged and an access exception is recognized for the location of the appropriate cryptography counter if the cryptography counter update pending flag is, e.g., one.

[00157] When the intermediate input message part flag is, e.g., zero (indicating operand 2 includes the last input message part) and the operation ends due to partial completion, a specified condition code (e.g., condition code 3) is set.

[00158] The description in the following paragraphs apply to the functions when the intermediate input message part flag is, e.g., zero (indicating operand 2 includes the last input message part):

[00159] If the cryptography counter update pending flag is, e.g., one, the input message padding and hashing, the outer-key padding and hashing, and the output message padding and hashing operations are not performed. Instead, in one example, only the appropriate counter in the cryptography counter set is updated, assuming the counter is accessible.

[00160] After the full input message blocks of operand 2 (e.g., all the full blocks) have been processed, an input message padding and hashing operation is performed. For instance, the input message bit length is updated to include the input key bit length by, e.g., adding 512 (input key bit length) to the input message bit length and any carry is ignored. Then, a padding operation 640 is performed on a final message block of the message 642 (e.g., either on the last (partial) input message block or on a null block (M_{PBM}) in operand 2 (based on the calculated L value)) using the input message bit length in byte offsets 32-39 of the parameter block to produce the padded input message block. Next, a hashing operation 640 is performed on the padded input message block, and a final input message digest (FIMD) 648 is generated using the SHA-256 block digest algorithm with the 32-byte chaining value (ICV₂; e.g., the intermediate message digest) 644 from the parameter block.

[00161] In one example, the padding and hashing operation of the input message is performed as described with reference to FIGS. 6B-6D. For instance, if the length of the partial message block length (L) is, e.g., zero bytes, then the operation in FIG. 6B is performed; if the length of the partial message block length (L) is, e.g., between one byte and 55 bytes inclusive, then the operation in FIG. 6C is performed; and if the length of the partial message block length (L) is, e.g., between 56 bytes and 63 bytes inclusive, then the operation in FIG. 6D is performed.

[00162] Referring to FIG. 6B, a padding and hashing operation 1650 includes concatenating padding bytes 1652 (e.g., 56 padding bytes, in which the leftmost byte is 80 hex, other bytes are 00 hex) to an 8-byte value specifying a bit length of the total input message (IMBL) 1654, a result of which is input to a block digest algorithm 1656 (e.g., a SHA-256 bda). Using block digest algorithm 1656 and a 32-byte input chaining value 1658 a 32-byte output chaining value 1659 is generated.

[00163] Referring to FIG. 6C, a padding and hashing operation 1660 includes concatenating a value specifying a length 1662 of the M_{PBM} in operand 2 and padding bytes 1664 (e.g., 56-L padding bytes, in which the leftmost byte is 80 hex, other bytes are 00 hex) to an 8-byte value specifying a bit length of the total input message (IMBL) 1665, a result of which is input to a block digest algorithm 1666 (e.g., a SHA-256 bda). Using block digest algorithm 1666 and a 32-byte input chaining value 1668 a 32-byte output chaining value 1669 is generated.

[00164] Referring to FIG. 6D, a padding and hashing operation 1670 includes concatenating padding bytes 1672 (e.g., 56 padding bytes of zero) to an 8-byte value specifying a bit length of the total input message (IMBL) 1674, a result of which is a padding block 1675 that includes, e.g., 56 bytes of, e.g., zero followed by an 8-byte IMBL. Further, a value specifying a length 1676 of the M_{PBM} in operand 2 is concatenated with padding bytes 1678 (e.g., 64-L padding bytes, in which, the leftmost byte, in one example is, e.g., 80 hex and the other bytes are, e.g., 00 hex) to produce a 64-byte result that is input to a block digest algorithm 1680 (e.g., a SHA-256 bda). Using block digest algorithm 1680 and a 32-byte input chaining value 1682 a 32-byte result 1684 is generated. Result 1684 and padding block 1675 are input to a block digest algorithm 1686 (e.g., a SHA-256 bda) to generate a 32-byte output changing value 1688.

[00165] Returning to FIG. 6A, in one example, an outer-key padding and hashing operation is performed. For instance, the 512-bit (64-byte) cryptographic key (K) 602 is exclusive-ORed 650 with the outer pad (opad) 652 (e.g., 64 bytes of 36 hex) to produce the 64-byte outer-key 654. A 32-byte chaining value 3 (OCV₃) 660 is generated for the 64-byte outer-key 654 using the SHA-256 block digest algorithm 656 with the 32-byte H⁽⁰⁾ value 658.

[00166] After the chaining value 3 (OCV₃) 660 is generated, an output message padding and hashing operation is performed. For instance, a padding operation 662 is performed on the final input message digest (FIMD) 648, also called the (partial) output message block (M_{PMB}), using the final input message digest (FIMD) bit length to produce the padded output message block. For Compute Message Authentication Code-Hash-based Message Authentication Code-SHA-224 and Compute Message Authentication Code-Hash-based Message Authentication Code-Encrypted-SHA-224 functions, the leftmost 28 bytes of the final input message digest (FIMD) is used as the (partial) output message block (M_{PMB}), L is set to, e.g., 28, and output message bit length is set to, e.g., 736. For Compute Message Authentication Code-hash-based message authentication code-SHA-256 and Compute Message Authentication Code-hash-based message authentication code-Encrypted-SHA-256 functions, the entire final input message digest (FIMD) is used as the (partial) output message block (M_{PMB}), L is set to 32, and OMBL is set to, e.g., 768.

[00167] Next, a final hashing operation 662 is performed on the padded output message block and the final output message digest (OCV) 664 (also referred to as the resulting authentication code) is generated using the SHA-256 block digest algorithm with, e.g., the 32-byte chaining value 3 (OCV₃) 660. The entire final output message digest (OCV) 664 is stored into the chaining value (CV) field of the parameter block.

[00168] The final padding and hashing operation of the output message is further described with reference to FIG. 6E. In one example, a padding and hashing operation 1690 includes concatenating FIMD 1692 and padding bytes 1694 (e.g., 24 padding bytes, in which the leftmost byte is 80 hex, other bytes are 00 hex) to an 8-byte value specifying a bit length of the total output message (OMBL) 1695, a result of which is input to a block digest algorithm 1696 (e.g., a SHA-256 bda). Using block digest algorithm 1696 and a 32-byte output chaining value (OCV₃) 1698 a 32-byte output chaining value 1699 is generated.

[00169] Although in the example herein, certain byte sizes are described, other byte sizes may be used in other examples. Further, other size SHA block digest algorithms may be used, as well as other hash or hash-based techniques. Many examples are possible.

[00170] In one example, when the intermediate input message part flag is, e.g., zero (indicating the last input message part), the following occurs, in one example: If the central processing unit is not enabled to update counters in the cryptography counter set, then a selected condition code (e.g., condition code 0) is set (indicating normal completion) and the cryptography counter update pending flag is set to, e.g., zero. If the central processing unit is enabled to update counters in the cryptography counter set and the appropriate counter is accessible, then the appropriate counter is updated, a selected condition code (e.g., condition code 0) is set (indicating normal

completion), and the cryptography counter update pending flag is set to, e.g., zero. If the central processing unit is enabled to update counters in the cryptography counter set and the appropriate counter is not accessible, then a specified condition code (e.g., condition code 3) is set and the cryptography counter update pending flag is set to, e.g., one if the cryptography counter update pending flag is, e.g., zero, or the condition code remains unchanged and an access exception is recognized for the location of the appropriate cryptography counter if the cryptography counter update pending flag is, e.g., one.

[00171] In one example, a specification exception is recognized and no other action is taken if any of the following occurs:

[00172] • Bit 56 of general register 0 is not zero.

[00173] • Bits 57-63 of general register 0 specify an unassigned or uninstalled function code.

[00174] • The R_2 field designates an odd-numbered register or general register 0.

[00175] • The second operand length is not a multiple of the data block size when the designated function is either not a hash-based message authentication code function (e.g., function codes 112-115 and 120-123) or it is a hash-based message authentication code function but the intermediate input message part flag is, e.g., one.

[00176] • When a hash-based message authentication code function is specified and the intermediate input message part flag is, e.g., one and the inner-key padding flag is, e.g., zero and the second operand length is, e.g., zero.

[00177] Example resulting condition codes include, for instance: 0 Normal completion; 1 Verification-pattern mismatch; 2 --; 3 Partial completion.

[00178] Example program exceptions include, for instance:

[00179] • Access (fetch, operand 2, cryptographic key, input-message-bit length (IMBL), and wrapping-key verification pattern; fetch and store, chaining value, cryptography counter)

[00180] • Operation (if the message-security assist is not installed)

[00181] • Specification

[00182] • Transaction constraint

[00183] One example of execution priority is indicated below:

- [00184] 1.-6. Exceptions with the same priority as the priority of program-interruption conditions for the general case.
- [00185] 7.A Access exceptions for second instruction halfword.
- [00186] 7.B Operation exception.
- [00187] 7.C Transaction constraint.
- [00188] 8. Specification exception due to invalid function code or invalid register number.
- [00189] 9. Specification exception due to invalid operand length.
- [00190] 10. Access exceptions for an access to a cryptography counter and second operand length originally zero and the designated function is not a hash-based message authentication code function.
- [00191] 11. Condition code 0 (normal completion) due to second operand length originally zero and the designated function is not a hash-based message authentication code function.
- [00192] 12.A.1.A Access exceptions for an access to the entire parameter block when the designated function is a hash-based message authentication code function.
- [00193] 12.A.1.B Access exceptions for an access to the parameter block when the designated function is not a hash-based message authentication code function.
- [00194] 12.A.2. Condition code 1 due to verification pattern mismatch.
- [00195] 12.A.3.A. Condition code 3 due to the second operand length is not originally zero and the designated function is a hash-based message authentication code function and the cryptography counter update pending flag is one.
- [00196] 12.A.3.B. Condition code 3 due to the second operand length is originally zero and the designated function is a hash-based message authentication code function and the intermediate input message part flag is zero and the inner-key padding flag is originally zero and the cryptography counter update pending flag is one.
- [00197] 12.A.4. Condition code 3 due to the second operand length is originally zero and the designated function is a hash-based message authentication code function and the intermediate input message part flag is one and the inner-key padding flag is one and access exception condition is detected for an access to a cryptography counter.

- [00198] 12.A.5. Access exceptions for an access to a cryptography counter and the second operand length originally zero and the designated function is a hash-based message authentication code function and the intermediate input message part flag is one and the inner-key padding flag is one.
- [00199] 12.A.6. Condition code 0 (normal completion) due to the second operand length is originally zero and the designated function is a hash-based message authentication code function and the intermediate input message part flag is one and the inner-key padding flag is one.
- [00200] 12.B. Access exceptions for an access to the parameter block when the designated function is not a hash-based message authentication code function or second operand storage area.
- [00201] 13. Condition code 3 due to partial completion (second operand length still nonzero).
- [00202] 14.A. Condition code 3 due to second operand length stepped to zero and the designated function is either not a hash-based message authentication code function, or it is a hash-based message authentication code function and the intermediate input message part flag is one, and access-exception condition detected for an access to a cryptography counter.
- [00203] 14.B. Condition code 3 due to the second operand length stepped to zero and the designated function is a hash-based message authentication code function and the intermediate input message part flag is zero and the outer-key padding and hashing operation is performed and access exception condition is detected for an access to a cryptography counter.
- [00204] 14.C. Condition code 3 due to the second operand length originally zero and the designated function is a hash-based message authentication code function and the intermediate input message part flag is zero and the cryptography counter update pending flag is originally zero and the outer-key padding and hashing operation is performed and access exception condition is detected for an access to a cryptography counter.
- [00205] 15.A. Access exceptions for an access to a cryptography counter and the second operand length stepped to zero and the designated function is either not a hash-based message authentication code function, or it is a hash-based message authentication code function and the intermediate input message part flag is one.
- [00206] 15.B. Access exceptions for an access to a cryptography counter and the second operand length stepped to zero and the designated function is a hash-based message authentication code function and the intermediate input message part flag is zero and the outer-key padding and hashing operation is performed.
- [00207] 15.C. Access exceptions for an access to a cryptography counter and the second operand length originally zero and the designated function is a hash-based message authentication code function and the intermediate input message part flag is zero and the cryptography counter update pending flag is originally one.

[00208] 16.A. Condition code 0 (normal completion) due to the second operand length stepped to zero and the designated function is either not a hash-based message authentication code function, or it is a hash-based message authentication code function and the intermediate-input-message part flag is one and access exception condition is not detected for an access to a cryptography counter.

[00209] 16.B. Condition code 0 (normal completion) due to the second operand length stepped to zero and the designated function is a hash-based message authentication code function and the intermediate input message part flag is zero and the outer-key padding and hashing operation is performed and access exception condition is not detected for an access to a cryptography counter.

[00210] 16.C. Condition code 0 (normal completion) due to the second operand length originally zero and the designated function is a hash-based message authentication code function and the intermediate input message part flag is zero and the cryptography counter update pending flag is originally one and access exception condition is not detected for an access to a cryptography counter.

[00211] In one or more aspects, when condition code 3 is set, the second operand address and length in general registers R_2 and $R_2 + 1$, respectively, and the initial chaining value in the parameter block are usually updated such that the program can simply branch back to the instruction to continue the operation. For unusual situations, the central processing unit protects against endless reoccurrence for the no-progress case. Thus, the program can safely branch back to the instruction whenever condition code 3 is set with no exposure to an endless loop.

[00212] In one or more aspects, if the length of the second operand is nonzero initially and condition code 0 is set, the registers are updated in the same manner as for condition code 3; the initial chaining value in this case is such that additional operands can be processed as if they were part of the same chain.

[00213] In one or more aspects, before processing the first part of a message, the program is to set the initial values for the initial chaining value field. To comply with, e.g., ANSI X9.9 or ANSI X9.19, the initial chaining value shall be set to, e.g., all binary zeros.

[00214] In one or more aspects, the following applies to the input message bit length in the parameter block of the hash-based message authentication code functions, as examples:

[00215] • The program supplied input message bit length value includes only the bit length of the entire input message. The hash-based message authentication code function adds the bit length of the input key to the input message bit length when performing the input message padding and hashing operation.

[00216] • The input message bit length is completely independent of the second operand length in general register $R_2 + 1$.

[00217] • Regardless of whether the instruction ends with condition code 0 or 3, the input message bit length is not decremented by the number of bytes processed.

[00218] • In normal usage, the input message bit length is expected to be eight times the total size of the input message in bytes. If the program supplies an input message bit length that is not a multiple of eight, the results will be algorithmically correct, but may not be usable in any practical application.

[00219] • One example of a secure hash algorithm allows for input message bit lengths that are not multiples of eight. The cryptography counter update pending functions use input message bit lengths of a multiple of eight.

[00220] Other examples and/or variations are possible.

[00221] In one example, when computing a message digest, the program may not initially be aware of the total message bit length; for example, for a message being read from an I/O device, the message bit length may not be known until the final block is read. When computing a message digest for a message whose length is not known, or for a message where it is known that the last message block is not included in the calculation, a hash-based message authentication code function with intermediate input message part flag set to, e.g., one may be used. When computing a message digest for a message that includes the last block, a hash-based message authentication code function with intermediate input message part flag set to, e.g., zero is to be used.

[00222] In one example, the output chaining value is the output of the SHA block digest algorithm, which is stored into the chaining value of the parameter block.

[00223] In the description herein of a compute message authentication code instruction, such as Compute Message Authentication Code instruction 500, specific locations, specific fields and/or specific sizes of the fields may be indicated (e.g., specific bytes and/or bits). However, other locations, fields and/or sizes may be provided. Further, although the setting of a bit to a particular value, e.g., one or zero, may be specified, this is only an example. The bit, if set, may be set to a different value, such as the opposite value or to another value, in other examples. Many variations are possible.

[00224] In one embodiment, the fields of the instruction are separate and independent from one another; however, in other embodiments, more than one field may be combined. Further, although example types of registers are used, other types of registers may be used. Other examples are possible.

[00225] An instruction, such as a Compute Message Authentication Code instruction, may have additional, fewer and/or other fields. For instance, one or more fields of a message instruction, such as the Compute Message Authentication Code instruction, may be optional. Many variations are possible.

[00226] Although various examples are provided for one or more formats of the instruction, additional and/or other formats may be used. Further, the processing may be used for other purposes than described herein.

[00227] Further details of hash-based message authentication code processing of an instruction, such as a Compute Message Authentication Code instruction, are described with reference to FIG. 7. In one example, a hash-based message authentication code process 700 (also referred to as process 700) is executed by one or more computing devices (e.g., one or more computers, such as computer(s) 101 and/or other computers; one or more servers, such as remote server(s) 104 and/or other remote servers; one or more devices, such as end user device(s) 103 and/or other end user devices; one or more processors or nodes, such as processor(s) or node(s) of processor set 110 (e.g., processor 200) and/or other processor(s) or node(s); processing circuitry, such as processing circuitry 120 of processor set 110 and/or other processing circuitry; one or more hardware accelerators separate and/or part of one or more processors and/or processing circuitry; and/or other computing devices, etc.). Additional and/or other computers, servers, devices, processors, nodes, processing circuitry, accelerators, and/or computing devices may be used to execute the processing and/or aspects thereof. Many examples are possible.

[00228] Referring to FIG. 7, in one example, process 700 obtains 710 an instruction using, e.g., obtain instruction code 400 (FIG. 4A). For instance, process 700 obtains Compute Message Authentication Code instruction 500 (FIG. 5A) or another instruction. Process 700 executes 720 the instruction using, e.g., execute instruction code 410.

[00229] In one example, in executing the instruction, process 700 obtains 730 one or more operands and/or information of an encoding of the instruction. The operands and/or information obtained depends, for instance, on the function or operation to be performed. In one example, process 700 obtains an address of the message (e.g., using R_2), a length of the message (e.g., using $R_2 + 1$), one or more flags from flags field 544 of, e.g., general register 0, a function code from function code field 546 of, e.g., general register 0 and an address of a parameter block (e.g., using general register 1). One or more parameters may be obtained from the parameter block, such as one or more chaining values, an input message bit length and at least one key (e.g., a clear key, or an encrypted key and a wrapping key verification pattern); other examples are possible.

[00230] Using one or more of the operands, process 700 performs 740 a plurality of operations of the instruction, based, e.g., on the function code and using, e.g., perform operations code 416. For instance, process 700 performs 742 an inner-key padding and hashing operation to generate an output chaining value (e.g., OCV₁). For instance, as described with reference to FIG. 6A, to perform the inner-key padding and hashing operation, a cryptographic key (K) 602 (e.g., 512-bit (64-byte)) is exclusive-ORed 604 with an inner pad (ipad) 606 to produce an inner-key 608 (64-byte). A chaining value 1 (CV₁) 614 (e.g., 32-byte) is generated for inner-key 608 using, e.g., the SHA-256 block digest algorithm (bda) 610 with the $H^{(0)}$ 612 value (e.g., 32-byte) and the inner-key padding flag is set to, e.g., one. The generated chaining value 1 (OCV₁) 614, also called the output chaining value (OCV), is

stored into the chaining value (CV) field of parameter block 618. If the operation ends due to partial completion, a specified condition code (e.g., condition code 3) is set.

[00231] Returning to FIG. 7, process 700 determines 744 if the inner-key padding and hashing operation is complete. If it is not complete (e.g., has been interrupted due to partial completion of the instruction (e.g., CC=3)), process 700 saves 746 the instruction state (e.g., the chaining value (e.g., OCV₁)). This allows the instruction to be re-executed based on the instruction (e.g., the inner-key padding and hashing operation of the instruction) being interrupted.

[00232] Based on process 700 determining 744 that the inner-key padding and hashing operation is complete, process 700 performs 748 a sequence of hash operations on a message (specified by the instruction), using the chaining value (e.g., OCV₁), to generate an intermediate message digest (IMD, e.g., OCV₂). For instance, as described with reference to FIG. 6A, intermediate message digest (IMD) 636 (e.g., 32-byte) is generated for the full input message blocks (M1, M2, ..., Mn) 620 (e.g., 64-byte) in operand 2 632 using the SHA-256 block digest algorithm 634 with the chaining value (ICV₁) 626 (e.g., 32-byte) from the parameter block. If the intermediate input message part flag is, e.g., one or L is, e.g., zero, M_{LFB} 628 is the last input message block (Mn); otherwise, M_{LFB} is the input message block that immediately precedes the last (partial) input message block (Mn-1). The generated intermediate message digest (OCV₂ 636, also called the output chaining value (OCV)), is stored into the chaining value (CV) field of parameter block 618. This operation repeats until the remaining input message is less than, e.g., 64 bytes or until a CPU-determined number of blocks have been stored. If the operation ends due to partial completion, a specified condition code (e.g., condition code 3) is set.

[00233] Continuing with FIG. 7, process 700 determines 750 whether the calculation of the intermediate message digest (e.g., OCV₂) is complete. If process 700 determines that generation of the intermediate message digest is incomplete (e.g., has been interrupted due to partial completion of the instruction (e.g., CC=3)), process 700 saves 746 the instruction state (e.g., OCV₂). This allows the instruction to be re-executed based on the instruction (e.g., the generating the intermediate message digest operation of the instruction) being interrupted.

[00234] Based on process 700 determining 750 that the generating of the intermediate message digest is complete, process 700 performs 752 an input message padding and hashing operation using the intermediate message digest to generate a final input message digest. For example, the input message bit length is updated to include the input key bit length by, e.g., adding 512 (input key bit length) to input message bit length and any carry is ignored. Then, as described with reference to FIG. 6A, the padding operation 640 is performed either on the last (partial) input message block or on a null block (M_{PBM}) 642 in operand 2 (based on the calculated L value) using the input message bit length in byte offsets 32-39 of the parameter block to produce the padded input message block. Next, a hashing operation 640 is performed on the padded input message block. For instance, the final input message digest (FIMD) 648 is generated using, e.g., the SHA-256 block digest algorithm with chaining value (ICV₂) 644 (e.g., 32-byte) from the parameter block.

[00235] Returning to FIG. 7, process 700 also performs 754 an outer-key padding and hashing operation to generate another chaining value (OCV_3). For example, as described with reference to FIG. 6A, cryptographic key (K) 602 (e.g., 512-bit (64-byte)) is exclusive-ORed 650 with outer pad (opad) 652 to produce outer-key 654 (e.g., 64-byte). A chaining value 3 (OCV_3) (e.g., 32-byte) 660 is generated for outer-key 654 (e.g., 64-byte) using the SHA-256 block digest algorithm 656 with the $H^{(0)}$ value 658 (e.g., 32-byte).

[00236] Continuing with FIG. 7, process 700 performs 756 an output message padding and hashing operation using the final input message digest and OCV_3 to generate the final output message digest. For example, as described with reference to FIG. 6A, after the chaining value 3 (OCV_3) 660 is generated, a padding operation 662 is performed on the final input message digest (FIMD) 648, also called the (partial) output message block (M_{PMB}), using the final input message digest (FIMD) bit length to produce the padded output message block. For Compute Message Authentication Code-Hash-based Message Authentication Code-SHA-224 and Compute Message Authentication Code-Hash-based Message Authentication Code-Encrypted-SHA-224 functions, the leftmost 28 bytes of the final input message digest (FIMD) is used as the (partial) output message block (M_{PMB}), L is set to, e.g., 28, and output message bit length is set to, e.g., 736. For Compute Message Authentication Code-Hash-Based Message Authentication Code-SHA-256 and Compute Message Authentication Code-Hash-Based Message Authentication Code-Encrypted-SHA-256 functions, the entire final input message digest (FIMD) is used as the (partial) output message block (M_{PMB}), L is set to 32, and OMBL is set to, e.g., 768.

[00237] Next, a final hashing operation 662 is performed on the padded output message block. For instance, the final output message digest (OCV) 664 is generated using the SHA-256 block digest algorithm with, e.g., the chaining value 3 (OCV_3) (e.g., 32-byte). The entire final output message digest (OCV) 664 is stored into the chaining value (CV) field of the parameter block.

[00238] Returning to FIG. 7, based on performing the operations (e.g., performing the inner-key padding and hashing operation, generating the intermediate message digest, performing the input message padding and hashing operation, performing the outer-key padding and hashing operation and performing the output message padding and hashing operation), process 700 provides 780 a result (e.g., using provide result code 418) of the instruction. The result is, for instance, a final output message digest (e.g., resulting authentication code).

[00239] In one or more aspects, the operations, unless interrupted, are executed as part of a single execution of the instruction. In one or more aspects, the inner-key padding and hashing operation and the performing the sequence of operations to generate the intermediate message digest operation may be interrupted. If an operation of the instruction is interrupted, state of the instruction (e.g., an output chaining value (e.g., OCV_1 and OCV_2)) is saved, such that the instruction may be re-executed from where it was interrupted.

[00240] Further details of one or more aspects of the present disclosure are described with reference to FIG. 8. For instance, the use of software, firmware and hardware to perform aspects of the present disclosure is described with reference to FIG. 8. As shown, in one example, software creates or populates 800 a parameter

block 810 for hash-based message authentication code processing. Parameter block 810 includes, in one example, a plurality of parameters, such as chaining values 812 (e.g., H0-H7), a key 814 (either clear or encrypted) and a message length 816 (e.g., the length of the message in bytes). A parameter block may include additional, fewer and/or other parameters.

[00241] Example lengths of entries of the parameter block are shown in FIG. 8. For instance, the chaining value entries have a total length of 64*8 bits for SHA-512 and 32*8 bits for SHA-256; the key entry has a length of 64*16 bits for SHA-512 and 32*16 bits for SHA-256; and the message length entry has a length of 64 bits. The entries may be other lengths in other examples. Further, other hashing algorithms may be used.

[00242] In one example, firmware (e.g., millicode) translates 820 the parameters and provides the translated parameters in a hardware input buffer 830. Hardware input buffer 830 includes, in one example, a plurality of parameters, such as chaining values 832 (e.g., H0-H7), a key 834 (either clear or unwrapped) and a pointer to a message 836. A hardware input buffer may include additional, fewer and/or other parameters.

[00243] Example lengths of entries of the hardware input buffer include, for instance: the chaining value entries have a total length of 64*8 bits for SHA-512 and 32*8 bits for SHA-256; the key entry has a length of 64*16 bits for SHA-512 and 32*16 bits for SHA-256; and the message pointer entry has a length equal to the length of the message with zero padding plus the message length in bytes. The entries may be other lengths in other examples.

[00244] In one example, the hardware (e.g., at least one accelerator) computes 840 using one or more parameters of hardware input buffer 830 a hash-based message authentication code. One or more accelerator registers 850 are used to hold the hash-based message authentication code output. Firmware (e.g., millicode) provides 860 the result to software. Other examples and variations are possible.

[00245] In one or more aspects, an accelerator (e.g., a processor core accelerator) is used in performing hash-based message authentication code processing. One example of such an accelerator is described with reference to FIG. 9. As depicted, in one example, a co-processor 900 (e.g., an accelerator) includes an input buffer 910 (e.g., an input first in-first out (fifo) buffer) that receives input, such as operands, from a load/store data I/F 960. Further, co-processor 900 includes an output buffer 920 (e.g., an output first in-first out (fifo) buffer) that is used to provide results to be stored in a data cache, such as data cache 962. Each buffer is also coupled to one or more engines 930 of co-processor 900.

[00246] Engines 930 include, for instance, encryption/decryption engine(s) 932, one or more secure hash algorithm (SHA) engines 934, a random generator engine 936 and one or more other engines 938 to perform various operations including, but not limited to sort, compare, character encoding, etc. One or more of the engines are controlled by one or more controls 940 that receive, for instance, input for the firmware (e.g., millicode 950). One or more of the controls is, e.g., an instruction issue 955.

[00247] Further details of one example of a SHA-2 engine is described with reference to FIG. 10. As shown, in one example, a SHA-2 engine 1000 receives input from an input buffer 1002 coupled thereto and provides output via one or more registers 1004. There are, e.g., one or more latches 1006 used to store the key and/or hash. Other examples and variations are possible.

[00248] One example of a hardware state machine associated with the processing is depicted in FIG. 11. In one example, the hardware is initially in an idle state 1100 and when the processing begins 1102 (e.g., instruction is executed), one or more hardware registers are loaded 1104 with one or more parameters (e.g., key, chaining value, etc.). For instance, a key is loaded 1106 and the key is hashed 1110 with input padding, and a pause 1120 may be performed by the engine in case there are stalls in loading the input data. Based on ending the pause, the engine continues with computing the subsequent rounds 1121 in the hash function. Once the round computation is complete, the chaining values are updated 1122. The engine then checks if the input message buffer has data and goes back to 1104 in case it does. Since the key need not be processed on subsequent iterations, 1108 moves to 1109 to compute the initial hashing rounds on the message. Once the entire message has been processed, 1122 moves to 1130 where the key is loaded once again and hashed with output padding followed by the result of the message hash being fed to the hash round computations again. At the end of this operation, the result is returned in registers 1132 and the engine goes back to idle state.

[00249] In one or more aspects, a hash-based message authentication code processing capability is provided that includes an interruptible instruction to perform and accelerate hash-based message authentication code processing. The processing provides a message authentication code that is used to authenticate and/or verify data. It may be used in many situations, including but not limited to, authenticating email addresses during login, verifying other types of data, securing communications within a computing environment and/or over a network, etc.

[00250] In one or more aspects, the instruction may be executed a plurality of times in performing complex cryptographic operations, such as password-based key derivation functions, as one example. Other examples are possible.

[00251] Although one or more examples of a computing environment to incorporate and use one or more aspects of the present disclosure are described herein, FIGS. 12A-12B depict another embodiment of a computing environment to incorporate and use one or more aspects of the present disclosure.

[00252] Referring, initially, to FIG. 12A, in this example, a computing environment 36 includes, for instance, a native central processing unit (CPU) 37 based on one architecture having one instruction set architecture, a memory 38, and one or more input/output devices and/or interfaces 39 coupled to one another via, for example, one or more buses 40 and/or other connections.

[00253] Native central processing unit 37 includes one or more native registers 41, such as one or more general purpose registers and/or one or more special purpose registers used during processing within the

environment. These registers include information that represents the state of the environment at any particular point in time.

[00254] Moreover, native central processing unit 37 executes instructions and code that are stored in memory 38. In one particular example, the central processing unit executes emulator code 42 stored in memory 38. This code enables the computing environment configured in one architecture to emulate another architecture (different from the one architecture) and to execute software and instructions developed based on the other architecture.

[00255] Further details relating to emulator code 42 are described with reference to FIG. 12B. Guest instructions 43 stored in memory 38 comprise software instructions (e.g., correlating to machine instructions) that were developed to be executed in an architecture other than that of native CPU 37. For example, guest instructions 43 may have been designed to execute on a processor based on the other instruction set architecture, but instead, are being emulated on native central processing unit 37, which may be, for example, the one instruction set architecture. In one example, emulator code 42 includes an instruction fetching routine 44 to obtain one or more guest instructions 43 from memory 38, and to optionally provide local buffering for the instructions obtained. It also includes an instruction translation routine 45 to determine the type of guest instruction that has been obtained and to translate the guest instruction into one or more corresponding native instructions 46. This translation includes, for instance, identifying the function to be performed by the guest instruction and choosing the native instruction(s) to perform that function.

[00256] Further, emulator code 42 includes an emulation control routine 47 to cause the native instructions to be executed. Emulation control routine 47 may cause native central processing unit 37 to execute a routine of native instructions that emulate one or more previously obtained guest instructions and, at the conclusion of such execution, return control to the instruction fetch routine to emulate the obtaining of the next guest instruction or a group of guest instructions. Execution of the native instructions 46 may include loading data into a register from memory 38; storing data back to memory from a register; or performing some type of arithmetic or logic operation, as determined by the translation routine.

[00257] Each routine is, for instance, implemented in software, which is stored in memory and executed by native central processing unit 37. In other examples, one or more of the routines or operations are implemented in firmware, hardware, software or some combination thereof. The registers of the emulated processor may be emulated using registers 41 of the native central processing unit or by using locations in memory 38. In embodiments, guest instructions 43, native instructions 46 and emulator code 42 may reside in the same memory or may be disbursed among different memory devices.

[00258] An example instruction that may be emulated is the Compute Message Authentication Code instruction described herein, in accordance with one or more aspects of the present disclosure.

[00259] The computing environments described herein are only examples of computing environments that can be used. One or more aspects of the present disclosure may be used with many types of environments. The computing environments provided herein are only examples. Each computing environment is capable of being configured to include one or more aspects of the present disclosure. For instance, each may be configured to implement accelerated and/or interruptible hash-based message authentication code processing and/or to perform one or more other aspects of the present disclosure.

[00260] One or more aspects of the present disclosure are tied to computer technology and facilitate processing within a computer, improving performance thereof. For instance, processing speed is increased, and latency is reduced by using one instruction, e.g., one architected instruction, to perform hash-based message authentication code processing.

[00261] In one or more aspects, a hardware and firmware co-design is provided that enables the computing of a hash-based message authentication code synchronously within a processor core pipeline, providing very low latency. In one or more aspects, an instruction is configured and used to accelerate hash-based message authentication code generation in a processor core. Operations to generate a hash-based message authentication code are performed based on parameters within a processor core pipeline synchronously.

[00262] In one or more aspects, one or more controls of the instruction enable the instruction to be interrupted and/or facilitate hash-based message authentication code processing. For instance, a flag bit (e.g., intermediate input message part flag) in the instruction parameter field allows firmware to interrupt the operation (e.g., at any time or selected times) to service higher priority interrupts, providing high responsiveness. As another example, a flag bit (e.g., intermediate input message part flag) in the instruction parameter field allows hash-based message authentication code processing to be performed with an incomplete parameter block (input chaining values to be used in hash-based message authentication code processing are saved in the parameter block as the processing progresses), reducing system latency. Further, in one example, a flag bit (e.g., an inner-key padding flag; an intermediate input message part flag and/or an operand 2 length) in one or more instruction fields allows hardware to use software-provided parameters to resume a previously interrupted operation or to initiate the standard's constant parameters, enabling system responsiveness and lower latency.

[00263] In one or more aspects, a single instruction and parameter set enable lower software processing overheads and higher performance. Reducing and/or eliminating the chaining of back-to-back accelerator calls to compute hash-based message authentication code results in lower overhead, improving performance.

[00264] In one or more aspects, a key used by the instruction may be protected by wrapping the key with a system key to increase security of the system.

[00265] In one or more aspects, a hash-based message authentication code operation is computed (generated) within a processor core pipeline synchronously based on parameters encapsulated by an instruction

(e.g., the Compute Message Authentication Code instruction). In one example, the instruction is a single instruction that encapsulates the parameters (e.g., all parameters) to trigger the hardware to perform operations of the instruction. In one or more aspects, a hash-based message authentication code operation is performed within a processor core pipeline based on one or more parameters of the instruction. In one or more aspects, the instruction is used to accelerate hash-based message authentication code computation in a processor core (instead of an external peripheral). The instruction has a format including, for instance, a key, a message address and a length, as parameters; state information or a chaining value to allow interruption and resuming of the operation (i.e., the instruction); a flag to allow early interruption for responsiveness (e.g., an intermediate input message part flag); a flag to allow a partial hash-based message authentication code (e.g., an intermediate input message part flag); a flag to allow hardware to initialize a hash at start (e.g., an inner-key padding flag); and allows use of keys protected from software and used by index tokens.

[00266] In one or more aspects, hardware and firmware (e.g., millicode) partitioning of hash-based message authentication code computation is provided. In one or more aspects, a hardware accelerator of a processor core is used that sequences back-to-back hashing operations to compute a hash-based message authentication code on a pipelined hashing accelerator engine. In one or more aspects: hardware computed key padding operations (e.g., ipad, opad) are performed; hardware is used to auto-initialize constants as per the operation based on a flag (e.g., inner-key padding flag); scheduling of key padding hash and message hash is performed in a way to allow interruption and resuming of operations by storing a single state value (e.g., use of intermediate input message part flag); protection of key latches on a scan dump is provided; computation of padding in firmware is performed prior to loading into hardware; and/or central processing unit interrupts are detected in firmware triggering hardware to stop hash-based message authentication code computation via one or more control bits (e.g., intermediate input message part flag).

[00267] In one or more aspects, hardware, firmware (millicode) partitioning of hash-based message authentication code generation such that message padding is handled in firmware but key padding in hardware and hardware returns context data to allow interruptability, as well as firmware detected interruptions. In one or more aspects, hardware-firmware partitioning is provided that supports interruptability. In one or more aspects, the hardware does not require external control signals to sequence different parts of the operation. Parameters of the instruction are used to control the operations.

[00268] In one or more aspects, interrupting includes terminating execution of the instruction prior to completion (e.g., at partial completion) with, e.g., a selected condition code (e.g., CC=3). Such an interrupted instruction may be re-executed from where it was interrupted, in accordance with one or more aspects, and avoids, for instance, a program interrupt exception that is typically handled by an interrupt handler.

[00269] Other and/or different aspects may be provided and/or included in processing of the single instruction. Processing within a processor, computer system and/or computing environment is improved.

[00270] Other aspects, variations and/or embodiments are possible.

[00271] In addition to the above, one or more aspects may be provided, offered, deployed, managed, serviced, etc. by a service provider who offers management of customer environments. For instance, the service provider can create, maintain, support, etc. computer code and/or a computer infrastructure that performs one or more aspects for one or more customers. In return, the service provider may receive payment from the customer under a subscription and/or fee agreement, as examples. Additionally, or alternatively, the service provider may receive payment from the sale of advertising content to one or more third parties.

[00272] In one aspect, an application may be deployed for performing one or more embodiments. As one example, the deploying of an application comprises providing computer infrastructure operable to perform one or more embodiments.

[00273] As a further aspect, a computing infrastructure may be deployed comprising integrating computer-readable code into a computing system, in which the code in combination with the computing system is capable of performing one or more embodiments.

[00274] Yet a further aspect, a process for integrating computing infrastructure comprising integrating computer-readable code into a computer system may be provided. The computer system comprises a computer-readable medium, in which the computer medium comprises one or more embodiments. The code in combination with the computer system is capable of performing one or more embodiments.

[00275] Although various embodiments are described above, these are only examples. For example, other instructions, instruction formats, operands and/or registers may be used. Further, other cryptographic algorithms may be used. Moreover, additional, less and/or other code may be used. Although particular code may be provided as an example of performing a particular operation or task, additional and/or other code may be used. Code may be combined and/or separated into code subsets. Many variations are possible.

[00276] Various aspects and embodiments are described herein. Further, many variations are possible without departing from a spirit of aspects of the present disclosure. It should be noted that, unless otherwise inconsistent, each aspect or feature described and/or claimed herein, and variants thereof, may be combinable with any other aspect or feature.

[00277] The terminology used herein is for the purpose of describing particular embodiments only and is not intended to be limiting. As used herein, the singular forms "a", "an" and "the" are intended to include the plural forms as well, unless the context clearly indicates otherwise. It will be further understood that the terms "comprises" and/or "comprising", when used in this specification, specify the presence of stated features, integers, steps, operations, elements, and/or components, but do not preclude the presence or addition of one or more other features, integers, steps, operations, elements, components and/or groups thereof.

[00278] The corresponding structures, materials, acts, and equivalents of all means or step plus function elements in the claims below, if any, are intended to include any structure, material, or act for performing the function in combination with other claimed elements as specifically claimed. The description of one or more embodiments has been presented for purposes of illustration and description but is not intended to be exhaustive or limited to in the form disclosed. Many modifications and variations will be apparent to those of ordinary skill in the art. The embodiment was chosen and described in order to best explain various aspects and the practical application, and to enable others of ordinary skill in the art to understand various embodiments with various modifications as are suited to the particular use contemplated.

CLAIMS

1. A computer program product comprising:
a set of one or more computer-readable storage media; and
program instructions, collectively stored in the set of one or more computer-readable storage media, for causing at least one computing device to perform computer operations including:
executing an instruction to generate an authentication code, the executing the instruction including:
performing a plurality of operations of the instruction to generate the authentication code, the plurality of operations including:
performing a sequence of hash operations on a message obtained using the instruction to generate an intermediate message digest, the performing the sequence of hash operations using an output chaining value generated based on performing an inner-key padding and hashing operation using a cryptographic key of the instruction; and
performing an outer-key padding and hashing operation using the cryptographic key to generate another output chaining value to be used in generating a final output message digest based on a final input message digest produced using the intermediate message digest, the final output message digest being a resulting authentication code, and wherein the performing the sequence of hash operations and the performing the outer-key padding and hashing operation are performed as part of a single invocation of the instruction.
2. The computer program product of claim 1, wherein the performing the plurality of operations further includes storing at least one chaining value generated based on execution of the instruction, wherein a chaining value of the at least one chaining value is to be used in re-execution of the instruction based on execution of the instruction being interrupted.
3. The computer program product of claim 2, wherein the computer operations further include:
determining that the instruction has been interrupted; and
re-executing the instruction using the chaining value.
4. The computer program product of claim 1, wherein the performing the plurality of operations further includes performing the inner-key padding and hashing operation using the cryptographic key to generate the output chaining value, wherein the performing the inner-key padding and hashing operation includes:
producing an inner-key based on performing a selected operation with the cryptographic key and an inner padding value;
generating the output chaining value for the inner-key using a hash operation and an input chaining value;
and
storing the output chaining value that is generated in a parameter block that is input to the instruction.

5. The computer program product of claim 4, wherein the selected operation is an exclusive OR operation.
6. The computer program product of claim 4, wherein the performing the sequence of hash operations on the message obtained using the instruction includes processing a plurality of message blocks of the message, the processing the plurality of message blocks including performing a plurality of block digest hash operations on the plurality of message blocks using the output chaining value as input to the processing of the plurality of message blocks to obtain the intermediate message digest.
7. The computer program product of claim 6, wherein the performing the plurality of operations further includes performing an input message padding and hashing operation for the message, the performing the input message padding and hashing operation for the message including:
 - performing a padding operation on a final message block of the message to produce a padded input message block; and
 - performing a hash operation, using the intermediate digest, on the padded input message block to generate the final input message digest.
8. The computer program product of claim 7, wherein the performing the outer-key padding and hashing operation includes:
 - producing an outer-key based on performing the selected operation with the cryptographic key and an outer padding value; and
 - generating the another output chaining value for the outer-key using a selected hash operation and the input chaining value.
9. The computer program product of claim 8, wherein the performing the plurality of operations further includes performing an output message padding and hashing operation, the performing the output message padding and hashing operation including:
 - performing a final padding operation on the final input message digest to produce a padded output message block; and
 - performing a final hashing operation, using the another output chaining value, on the padded output message block to generate the final output message digest.
10. The computer program product of claim 1, wherein the instruction includes a plurality of parameters to be used in performing the plurality of operations, the plurality of parameters being specified using one or more registers of the instruction.
11. The computer program product of claim 10, wherein the plurality of parameters includes at least one chaining value, an input message bit length, and the cryptographic key, and wherein the at least one chaining value is used in re-execution of the instruction based on the instruction being interrupted.

12. The computer program product of claim 11, wherein the instruction further includes as input an address of the message and one or more control indicators to control re-execution of the instruction based on the instruction being interrupted.

13. The computer program product of claim 1, wherein at least the performing the sequence of hash operations on the message obtained using the instruction uses a hardware accelerator.

14. The computer program product of claim 1, wherein the plurality of operations further includes:
performing the inner-key padding and hashing operation to generate the output chaining value;
performing an input message padding and hashing operation for the message using the intermediate message digest to generate the final input message digest; and
performing an output message padding and hashing operation using the final input message digest and the another output chaining value to produce the resulting authentication code.

15. A computer system comprising:
at least one computing device;
a set of one or more computer-readable storage media; and
program instructions, collectively stored in the set of one or more computer-readable storage media, for causing the at least one computing device to perform computer operations including:
executing an instruction to generate an authentication code, the executing the instruction including:
performing a plurality of operations of the instruction to generate the authentication code, the plurality of operations including:
performing a sequence of hash operations on a message obtained using the instruction to generate an intermediate message digest, the performing the sequence of hash operations using an output chaining value generated based on performing an inner-key padding and hashing operation using a cryptographic key of the instruction; and
performing an outer-key padding and hashing operation using the cryptographic key to generate another output chaining value to be used in generating a final output message digest based on a final input message digest produced using the intermediate message digest, the final output message digest being a resulting authentication code, and wherein the performing the sequence of hash operations and the performing the outer-key padding and hashing operation are performed as part of a single invocation of the instruction.

16. The computer system of claim 15, wherein the performing the plurality of operations further includes storing at least one chaining value generated based on execution of the instruction, and wherein the computer operations further include:

determining that the instruction has been interrupted; and
re-executing the instruction using a chaining value of the at least one chaining value.

17. The computer system of claim 15, wherein the performing the plurality of operations further includes performing the inner-key padding and hashing operation using the cryptographic key to generate the output chaining value, wherein the performing the inner-key padding and hashing operation includes:

producing an inner-key based on performing a selected operation with the cryptographic key and an inner padding value;

generating the output chaining value for the inner-key using a hash operation and an input chaining value;

and

storing the output chaining value that is generated in a parameter block that is input to the instruction.

18. The computer system of claim 17, wherein the performing the plurality of operations further includes performing an input message padding and hashing operation for the message, the performing the input message padding and hashing operation for the message including:

performing a padding operation on a final message block of the message to produce a padded input message block; and

performing a hash operation, using the intermediate message digest, on the padded input message block to generate the final input message digest.

19. A computer-implemented method comprising:

executing an instruction to generate an authentication code, the executing the instruction including:

performing a plurality of operations of the instruction to generate the authentication code, the plurality of operations including:

performing a sequence of hash operations on a message obtained using the instruction to generate an intermediate message digest, the performing the sequence of hash operations using an output chaining value generated based on performing an inner-key padding and hashing operation using a cryptographic key of the instruction; and

performing an outer-key padding and hashing operation using the cryptographic key to generate another output chaining value to be used in generating a final output message digest based on a final input message digest produced using the intermediate message digest, the final output message digest being a resulting authentication code, and wherein the performing the sequence of hash operations and the performing the outer-key padding and hashing operation are performed as part of a single invocation of the instruction.

20. The computer-implemented method of claim 19, wherein the performing the plurality of operations further includes storing at least one chaining value generated based on execution of the instruction, and wherein the computer operations further include:

determining that the instruction has been interrupted; and

re-executing the instruction using a chaining value of the at least one chaining value.

21. The computer-implemented method of claim 19, wherein the plurality of operations further includes performing the inner-key padding and hashing operation using the cryptographic key to generate the output chaining value, wherein the performing the inner-key padding and hashing operation includes:
- producing an inner-key based on performing a selected operation with the cryptographic key and an inner padding value;
 - generating the output chaining value for the inner-key using a hash operation and an input chaining value;
 - and
 - storing the output chaining value that is generated in a parameter block that is input to the instruction.
22. The computer-implemented method of claim 21, wherein the performing the plurality of operations further includes performing an input message padding and hashing operation for the message, the performing the input message padding and hashing operation for the message including:
- performing a padding operation on a final message block to produce a padded input message block of the message; and
 - performing a hash operation, using the intermediate digest, on the padded input message block to generate the final input message digest.
23. The computer-implemented method of claim 22, wherein the performing the plurality of operations further includes performing an output message padding and hashing operation, the performing the output message padding and hashing operation including:
- performing a final padding operation on the final input message digest to produce a padded output message block; and
 - performing a final hashing operation, using the another output chaining value, on the padded output message block to generate the final output message digest.
24. A computer system comprising:
- at least one hardware accelerator to be used in executing an instruction to generate an authentication code, the executing the instruction including:
 - performing a plurality of operations of the instruction to generate the authentication code, the plurality of operations including:
 - performing a sequence of hash operations on a message obtained using the instruction to generate an intermediate message digest, the performing the sequence of hash operations using an output chaining value generated based on performing an inner-key padding and hashing operation using a cryptographic key of the instruction; and
 - performing an outer-key padding and hashing operation using the cryptographic key to generate another output chaining value to be used in generating a final output message digest based on a final input message digest produced using the intermediate message digest, the final output message digest being a resulting

authentication code, and wherein the performing the sequence of hash operations and the performing the outer-key padding and hashing operation are performed as part of a single invocation of the instruction.

25. A computer-implemented method comprising:

executing an instruction, using at least one hardware accelerator, to generate an authentication code, the executing the instruction including:

performing a plurality of operations of the instruction to generate the authentication code, the plurality of operations including:

performing a sequence of hash operations on a message obtained using the instruction to generate an intermediate message digest, the performing the sequence of hash operations using an output chaining value generated based on performing an inner-key padding and hashing operation using a cryptographic key of the instruction; and

performing an outer-key padding and hashing operation using the cryptographic key to generate another output chaining value to be used in generating a final output message digest based on a final input message digest produced using the intermediate message digest, the final output message digest being a resulting authentication code, and wherein the performing the sequence of hash operations and the performing the outer-key padding and hashing operation are performed as part of a single invocation of the instruction.