



(51) International Patent Classification:
H04L 9/32 (2006.01)

(21) International Application Number:
PCT/GB2009/050814

(22) International Filing Date:
9 July 2009 (09.07.2009)

(25) Filing Language: English

(26) Publication Language: English

(30) Priority Data:
0812593.2 9 July 2008 (09.07.2008) GB

(71) Applicant (for all designated States except US): **THE QUEEN'S UNIVERSITY OF BELFAST** [GB/GB];
University Road, Belfast Antrim BT7 1NN (GB).

(72) Inventor; and

(75) Inventor/Applicant (for US only): **O'NEILL, Maire Patricia** [IE/GB]; Ecit, The Queen's University of Belfast,
Queen's Road, Queen's Island, Belfast Antrim BT3 9DT (GB).

(74) Agent: **MURGITROYD & COMPANY**; Scotland House, 165-169 Scotland Street, Glasgow Strathclyde G5 8PL (GB).

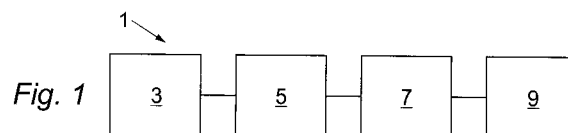
(81) Designated States (unless otherwise indicated, for every kind of national protection available): AE, AG, AL, AM, AO, AT, AU, AZ, BA, BB, BG, BH, BR, BW, BY, BZ, CA, CH, CL, CN, CO, CR, CU, CZ, DE, DK, DM, DO, DZ, EC, EE, EG, ES, FI, GB, GD, GE, GH, GM, GT, HN, HR, HU, ID, IL, IN, IS, JP, KE, KG, KM, KN, KP, KR, KZ, LA, LC, LK, LR, LS, LT, LU, LY, MA, MD, ME, MG, MK, MN, MW, MX, MY, MZ, NA, NG, NI, NO, NZ, OM, PE, PG, PH, PL, PT, RO, RS, RU, SC, SD, SE, SG, SK, SL, SM, ST, SV, SY, TJ, TM, TN, TR, TT, TZ, UA, UG, US, UZ, VC, VN, ZA, ZM, ZW.

(84) Designated States (unless otherwise indicated, for every kind of regional protection available): ARIPO (BW, GH, GM, KE, LS, MW, MZ, NA, SD, SL, SZ, TZ, UG, ZM, ZW), Eurasian (AM, AZ, BY, KG, KZ, MD, RU, TJ, TM), European (AT, BE, BG, CH, CY, CZ, DE, DK, EE, ES, FI, FR, GB, GR, HR, HU, IE, IS, IT, LT, LU, LV, MC, MK, MT, NL, NO, PL, PT, RO, SE, SI, SK, SM, TR), OAPI (BF, BJ, CF, CG, CI, CM, GA, GN, GQ, GW, ML, MR, NE, SN, TD, TG).

Published:

— without international search report and to be republished upon receipt of that report (Rule 48.2(g))

(54) Title: DATA SECURITY DEVICES AND METHODS



(57) Abstract: A hash value computation device for computing a hash value for a message using a hash algorithm that defines a key is provided, the device comprising a message schedule element having an input module which receives the message in 8-bit segments, a plurality of 8-bit memory modules which store the 8-bit segments of the message, an 8-bit message schedule value module which uses the 8-bit message segments and the hash algorithm to provide an 8-bit message schedule value for each 8-bit message segment, and an output module which outputs the 8-bit message schedule values, and a hash value computation element comprising a first input module which receives the 8-bit message schedule values, a second input module which receives 8-bit segments of the key defined by the hash algorithm, a plurality of 8-bit memory modules which store the 8-bit segments of the key, a hash value computation module which uses the 8-bit message schedule values and the 8-bit key segments to compute a hash value for the message, and an output module which outputs the hash value.



Data Security Devices and Methods

The invention relates to data security devices and methods that implement a hash algorithm.

5

Many devices and methods have been developed for application to data security. It is often desirable to provide a number of data security services, such as confidentiality (protecting the data from disclosure to unauthorised bodies), authentication (assuring that received data was transmitted by the body identified as the source), integrity (maintaining data consistency and ensuring that data has not been altered by unauthorised persons) and non-repudiation (preventing the originator of a message from denying transmission). Services such as these can be used, for example, in data communication networks, where each of the services may be provided, and in identification devices, such as radio frequency identification (RFID) tags, where authentication may be the most important service.

Various methodologies have been used to implement these security services. One such methodology is the use of hash functions, for example the suite of Secure Hash Algorithms (SHAs) devised by the US National Institute of Standard and Technology, which are particularly used to provide the authentication security service. In any device which is used to provide one or more of the security services, there are various criteria which need to be considered in order to give a device which can provide the or each security service to a required level in an efficient and realisable manner. One such criterion is the size of the device, and the area of material, for example, silicon which it uses. If the area is large, this has cost implications, and the device may be physically too big to be used in some applications, for example RFID tags.

30

According to a first aspect of the present invention there is provided a hash value computation device for computing a hash value for a message using a hash algorithm that defines a key, the device comprising

5 a message schedule element having an input module which receives the message in 8-bit segments, a plurality of 8-bit memory modules which store the 8-bit segments of the message, an 8-bit message schedule value module which uses the 8-bit message segments and the hash algorithm to provide an 8-bit message schedule value for each 8-bit message segment, and an output module which outputs the 8-bit message
10 schedule values, and

a hash value computation element comprising a first input module which receives the 8-bit message schedule values, a second input module which receives 8-bit segments of the key defined by the hash algorithm, a plurality of 8-bit memory modules which store the 8-bit segments of the
15 key, a hash value computation module which uses the 8-bit message schedule values and the 8-bit key segments to compute a hash value for the message, and an output module which outputs the hash value.

The hash value computation device may operate over n clock cycles to
20 compute the hash value for the message. The message schedule element may operate over the n clock cycles to calculate n message schedule values. The hash value computation module may operate over the n clock cycles to use the n message schedule values to compute the hash value for the message.

25

The hash value computation device may receive the message in a format appropriate for the hash algorithm used to compute the hash value. Alternatively, the hash value computation device may receive the message in a different format and change this to a format appropriate for
30 the hash algorithm used to compute the hash value. The hash value

computation device may comprise a message formatting module, which formats the message into a format appropriate for the hash algorithm used to compute the hash value. The message format may comprise N x-bit message blocks, where x is equal to or greater than 8, for example the message format may comprise N 512-bit message blocks. The message formatting module may format the message into N 512-bit message blocks, by padding the message, as appropriate, to a length of 448 mod 512, appending the length of the padded message as a 64-bit number to the padded message, and parsing the resultant message into N 512-bit data message blocks.

The input module of the message schedule element may receive the or each x-bit message block in 8-bit message segments. The input module may comprise a multiplexer. In each clock cycle of a first subset of the n clock cycles, the multiplexer may be configured to receive an 8-bit message segment and pass the 8-bit message segment to a first memory module of the plurality of 8-bit memory modules.

The plurality of 8-bit memory modules of the message schedule element may be arranged in a linear array, with a first 8-bit memory module connected to a second 8-bit memory module, the second 8-bit memory module connected to a third 8-bit memory module, etc. The first 8-bit memory module in the array may also be connected to the input module. The array of 8-bit memory modules may comprise a number of memory modules sufficient such that each memory module stores an 8-bit message segment of an x-bit message block of the message. In a first clock cycle of the first subset of the n clock cycles, the first 8-bit memory module in the array may receive a first 8-bit message segment, in a second clock cycle of the first subset of the n clock cycles, the first 8-bit memory module in the array may receive a second 8-bit message

segment, and the second 8-bit memory module in the array may receive the first 8-bit message segment, etc. for each of the clock cycles in the subset of clock cycles. Each of the plurality of 8-bit memory modules may comprise an 8-bit shift register.

5

The 8-bit message schedule value module may provide 8-bit message schedule values determined using modifications of a message schedule value definition specified by the hash algorithm. The modifications may change the message schedule value definition from a definition formulated to use 32-bit data segments to a definition formulated to use 8-bit data segments.

10

The modified message schedule value definition may comprise

$$15 \quad W_t = \begin{cases} Message_t & 0 \leq t \leq 63 \\ 32bitROT_{LEFT\ 1}(W_{t-12} \oplus W_{t-32} \oplus W_{t-56} \oplus W_{t-64}) & 64 \leq t \leq 319 \end{cases}$$

The 8-bit message schedule value module may use a first part of the modified message schedule value definition to provide a first set of 8-bit message schedule values which each comprise an 8-bit message segment of the message. The 8-bit message schedule value module may use a second part of the modified message schedule value definition to provide a second set of 8-bit message schedule values calculated using one or more functions carried out on one or more previously-determined message schedule values. The functions may comprise one or more logical functions, such as one or more XOR functions. The functions may comprise one or more rotate functions.

20

25

The 8-bit message schedule value module may comprises an 8-bit XOR module which carries out the one or more XOR functions of the second

part of the modified message schedule value definition. The XOR module may receive one or more previously-determined 8-bit message schedule values and carry out the one or more XOR functions on these, to provide one or more 8-bit XOR values.

5

The 8-bit message schedule value module may comprise a rotate module which carries out the one or more rotate functions of the second part of the modified message schedule value definition. The one or more rotate functions may comprise a rotate to the left by one place rotate function.

10 The rotate module may receive a plurality of 8-bit XOR values and carry out the one or more rotate functions on these, to provide a plurality of 8-bit message schedule values. The rotate module may receive at least one set of four 8-bit XOR values and carry out the one or more rotate functions on the set, to provide four 8-bit message schedule values. The four 8-bit
15 message schedule values may, on concatenation, give a 32-bit value which is the same as a 32-bit value obtained by an unmodified message schedule value definition of the hash algorithm carried out on a 32-bit value comprising the four 8-bit XOR values. The rotate module may comprise a data splitter, a first multiplexer, a register, a data combiner and
20 a second multiplexer.

The 8-bit message schedule value module may comprise a control module which is used to control the operation of the rotate module. The control module may be used to control the operation of the first multiplexer and
25 the second multiplexer of the rotate module.

The hash value computation module may compute a hash value for the message comprising a concatenation of a plurality of hash variables, preferably five 32-bit hash variables. The hash value computation module
30 may compute the hash variables using modifications of hash variable

definitions specified by the hash algorithm. The modifications may change the hash variable definitions from definitions formulated to use 32-bit message schedule values and 32-bit key segments to definitions formulated to use 8-bit message schedule values and 8-bit key segments.

5

The modified hash variable definitions may comprise

$$T = ROT_{LEFT-5}(a) + F_t(b, c, d) + e + K_t + W_t$$

$$e = d$$

10
$$d = c$$

$$c = ROT_{LEFT-30}(b)$$

$$b = a$$

$$a = T$$

$$F_t(b, c, d) = \begin{cases} (b \text{ AND } c) \text{ OR } (\bar{b} \text{ AND } d) & 0 \leq t \leq 79 \\ b \oplus c \oplus d & 80 \leq t \leq 159 \\ (b \text{ AND } c) \text{ OR } (b \text{ AND } d) \text{ OR } (c \text{ AND } d) & 160 \leq t \leq 239 \\ b \oplus c \oplus d & 240 \leq t \leq 319 \end{cases}$$

15

where a, b, c, d and e are the hash variables.

The hash value computation module may compute the hash value using 8-bit key segments of initial values of the hash variables defined by the hash algorithm.

20

The hash value computation module may comprise a rotate hash variable b module and a T computation module. The T computation module may comprise a rotate hash variable a module, an F computation module, an add device and a carry register.

25

The hash algorithm may be a Secure Hash Algorithm (SHA, such as SHA-1, SHA-2, etc.

5 According to a second aspect of the invention there is provided a method of computing a hash value for a message using a hash algorithm that defines a key, comprising

using a message schedule element of the hash value computation device of the first aspect of the invention to determine a plurality of 8-bit message schedule values, and
10 using a hash value computation element of the hash value computation device of the first aspect of the invention to compute the hash value for the message using the message schedule values.

15 According to a third aspect of the invention there is provided a data security device comprising a hash value computation device according to the first aspect of the invention.

The data security device may comprise an RFID tag.

20 The hash value computation device of the invention has been designed as an 8-bit architecture and to operate using 8-bit data segments. Such an 8-bit architecture allows the use of less silicon for the components thereof, and thus achieves a small area and low cost architecture, in comparison to all previous 32-bit architectures.

25

An embodiment of the hash value computation device of the invention will now be described, by way of example only, with reference to the following drawings, in which:

Figure 1 is a schematic representation of a hash value computation device according to the invention;

5 Figure 2 is a schematic representation of a message schedule element of the hash value computation device of Figure 1;

Figure 3 is a schematic representation of a rotate function carried out by the message schedule element of Figure 2;

10 Figure 4 is a schematic representation of a hash value computation element of the hash value computation device of Figure 1;

Figure 5 is a schematic representation of a rotate function carried out by the hash value computation element of Figure 4, and

15

Figure 6 is a schematic representation of a further rotate function carried out by the hash value computation element of Figure 4.

Referring to Figure 1, the hash value computation device 1 comprises an
20 input element 3, a message schedule element 5, a hash value computation element 7 and an output element 9. The various elements are shown as being separate, however it will be appreciated that one or more of them could be combined, for example the input element 3 may be integral with the message schedule element 5, and the output element 9
25 may be integral with the hash value computation element 7. The hash value computation device receives a message, i.e. data, and uses this to calculate a hash value, employing, in this embodiment, a hash algorithm comprising the SHA-1. This algorithm comprises three principal steps: message pre-processing, message schedule value calculation and hash
30 value computation.

The message is received by the input element 3. The message may have already undergone the pre-processing step, to place it in a format appropriate for SHA-1 computation, or the input element 3 may receive a 'raw' message and perform the pre-processing step to place the message into the appropriate format. The pre-processing step may be performed using a software module. The pre-processing comprises padding the message, as appropriate, to a length of $448 \bmod 512$, appending the length of the padded message as a 64-bit number to the padded message, and parsing the resultant message into N 512-bit message blocks. Each 512-bit message block is fed into the message schedule element 5 of the hash value computation device 1, and used in the computation of a final hash value.

Each of the elements of the hash value computation device of the invention are designed to work on 8-bit data segments. SHA-1 specifies definitions for the determination of the message schedule values and the computation of the hash values, These definitions having been devised to be performed on 32-bit data segments. The hash value computation device of the invention implements modifications of these definitions, the modifications being such that the definitions can be performed using 8-bit data segments.

Referring to Figure 2, the message schedule element 3 comprises an input module comprising an input multiplexer 11, a plurality of 8-bit memory modules comprising an array 13 of sixty four 8-bit shift registers (W63 to W0), a message schedule value module comprising an XOR module 15, a rotate module 17 and a control module 19, and an output module 21. The message schedule element 5 receives the 512-bit message blocks, in turn, from the input element 3. The message schedule element 5 operates over 320 clock cycles, $t = 0$ to 319, to provide 320

message schedule values, W_t . The message schedule value module provides 8-bit message schedule values determined using modifications of a message schedule value definition specified by the SHA-1. The modifications change the message schedule value definition from a definition formulated to use 32-bit data segments to a definition formulated to use 8-bit data segments.

The modified message schedule value definition comprises

$$W_t = \begin{cases} Message_t & 0 \leq t \leq 63 \\ 32bitROT_{LEFT\ 1}(W_{t-12} \oplus W_{t-32} \oplus W_{t-56} \oplus W_{t-64}) & 64 \leq t \leq 319 \end{cases}$$

The message schedule values determined using the above definition are then output to the hash value computation element 7, where they are used in the computation of the hash value for the message.

15

The input multiplexer 11 comprises a first data input 111, a second data input 112, a control input 113 and an output 114. In each clock cycle, the input multiplexer 11 receives 8-bit data either on the first data input 111 or the second data input 112, according to the value of a control signal received from the control module 19 on the control input 113. In each clock cycle, 8-bit data is output by the input multiplexer 11 to the first shift register W63 of the array of shift registers 13. The shift registers in the array of shift registers 13 are connected in a linear array as shown in Figure 2. In each clock cycle, any 8-bit data contained in the shift registers W63, W62 and W60 to W1 is passed to shift registers W62, W61 and W59 to W0 respectively. For any 8-bit data contained in shift register W61, the seven MSBs are passed to a first input of shift register W60, and the LSB is passed to a multiplexer. The output of the multiplexer is fed to a second input of the shift register W60. The data received on the first

and second inputs of the shift register W60 is then concatenated. The shift registers W52, W32, W8 and W0 are each connected to the XOR module 15. In each clock cycle, any 8-bit data contained in these shift registers is passed to the XOR module 15. This comprises an 8-bit XOR module, and uses the 8-bit data received from the shift registers to calculate an 8-bit XOR value, according to the above equation, which is fed to the rotate module 17.

The rotate module 17 comprises a data splitter 171, a first multiplexer 172, a register 173, a data combiner 174 and a second multiplexer 175. In each clock cycle, an 8-bit XOR value received from the XOR module 15 is input into the data splitter 171. This splits the 8-bit value into the two parts. The first part comprises the seven least significant bits (LSBs) of the 8-bit value, and is fed to the data combiner 174. The second part comprises the most significant bit (MSB) of the 8-bit value, and is fed to the first multiplexer 172. The first multiplexer comprises a data input, a control input and two outputs as shown. The 1-bit data received by the data input is output either to the register 173 or the second multiplexer 175, according to the value of a control signal received from the control module 19 on the control input. The register 173 is connected to the data combiner 174. 1-bit data contained in the register is fed to the data combiner 174, where it is combined with the seven LSBs of the 8-bit XOR value received by the rotate module 17. The combined 8-bit data is fed to the second data input 112 of the input multiplexer 11. The second multiplexer 175 comprises a first data input connected to the shift register W61, a second data input connected to the first multiplexer 172, a control input connected to the control module 19, and an output connected to the shift register W60. . In each clock cycle, the second multiplexer 175 receives 1-bit data either on the first data input or the second data input, according to the value of a control signal received from the control module

19 on the control input. The 1-bit data received from the shift register W61 comprises the LSB of the 8-bit data contained in the register W62. The second multiplexer 175 outputs the 1-bit data to the shift register W60. The output module 21 is connected between the shift registers W60 and W59, as shown. In each clock cycle, 8-bit data passed to the register W59 from the register W60 is also passed to the output module 21, and output from the message schedule element 5 to the hash value computation element 7.

10 The operation of the message schedule element 5 to generate 320 message schedule values, W_t , will now be described in detail, with reference to Figures 2 and 3.

15 A first 512-bit message block is prepared for loading into the message schedule element 5, by splitting it into 64 8-bit segments. The message schedule element 5 is operated over 320 clock cycles to receive the 8-bit segments and use them to generate the 320 message schedule values. Prior to the start of the 320 clock cycles, the control module 19 is operated to send control signals to the input multiplexer 11, the first multiplexer 172 of the rotate module 17 and the second multiplexer 175 of the rotate module 17. The control signal received by the input multiplexer 11 causes it to receive data on the first data input 111. The control signal received by the first multiplexer 172 of the rotate module 17 causes this multiplexer to output data on its first data output to the register 173. The control signal received by the second multiplexer 175 of the rotate module 17 causes this multiplexer to receive data on its first data input from the shift register W61. The shift registers W63 to W0 are initialised, so that they comprise data equating to zero.

In the first clock cycle, $t = 0$, a first 8-bit segment, S_0 , of the first 512-bit message block is fed into the input multiplexer 11 and from there to the first shift register W63. The zero data in shift register W63 is fed to shift register W62, and the zero data in shift register W62 is fed to shift register W61. The seven MSBs of the zero data in shift register W61 are fed straight to the first input of the shift register W60, and the LSB of the zero data in shift register W61 is fed to the second multiplexer 175. This has been configured to output the LSB of the zero data to the shift register W60, and the seven MSBs and the LSB of the zero data are concatenated therein. The zero data in shift registers W60 to W1 is fed to shift registers W59 to W0. Data output from shift registers W52, W32, W8 and W0 is processed by the XOR module 15 and the rotate module 17, and is output to the second data input 112 of the input multiplexer 11. However, this multiplexer is configured to ignore data received on this input. The output module 21 receives zero data from the shift register W60, and outputs this to the hash value computation element 7, where it is ignored in the first clock cycle of this element.

In the second clock cycle, $t = 1$, a second 8-bit segment S_1 of the first 512-bit message block is fed into the input multiplexer 11 and from there to the first shift register W63. The first 8-bit segment S_0 in shift register W63 is fed to shift register W62, and the zero data in shift register W62 is fed to shift register W61. The seven MSBs of the zero data in shift register W61 are fed straight to the first input of the shift register W60, and the LSB of the zero data in shift register W61 is fed to the shift register W60 as before where they are concatenated. The zero data in shift registers W60 to W1 is fed to shift registers W59 to W0. Data output from shift registers W52, W32, W8 and W0 is processed as above. The output module 21 again receives zero data from the shift register W60, and outputs this to the hash

value computation element 7, where it is ignored in the second clock cycle of this element.

5 In the third clock cycle, $t = 2$, a third 8-bit segment S_2 of the first 512-bit message block is fed into the input multiplexer 11 and from there to the first shift register W63. The second 8-bit segment S_1 in shift register W63 is fed to shift register W62, and the first 8-bit segment S_0 in shift register W62 is fed to shift register W61. The seven MSBs of the zero data in shift register W61 are fed straight to the first input of the shift register W60, and
10 the LSB of the zero data in shift register W61 is fed to the shift register W60 as before where they are concatenated. The zero data in shift registers W60 to W1 is fed to shift registers W59 to W0. Data output from shift registers W52, W32, W8 and W0 is processed as before. The output module 21 again receives zero data from the shift register W60, and
15 outputs this to the hash value computation element 7, where it is ignored in the third clock cycle of this element.

In the fourth clock cycle, $t = 3$, a fourth 8-bit segment S_3 of the first 512-bit message block is fed into the input multiplexer 11 and from there to the
20 first shift register W63. The third 8-bit segment S_2 in shift register W63 is fed to shift register W62, and the second 8-bit segment S_1 in shift register W62 is fed to shift register W61. The seven MSBs of the first 8-bit segment S_0 in shift register W61 are fed straight to the first input of the shift register W60. The LSB of the first 8-bit segment S_0 in shift register
25 W61 is fed to the second multiplexer 175. This has been configured to output the LSB to the shift register W60, and the seven MSBs and the LSB of the first 8-bit segment S_0 are concatenated therein. The zero data in shift registers W60 to W1 is fed to shift registers W59 to W0. Data output from shift registers W52, W32, W8 and W0 is processed as before. The
30 output module 21 again receives zero data from the shift register W60,

and outputs this to the hash value computation element 7, where it is ignored in the fourth clock cycle of this element.

In the fifth clock cycle, $t = 4$, a fifth 8-bit segment S_4 of the first 512-bit message block is fed into the input multiplexer 11 and from there to the first shift register W63. The fourth 8-bit segment S_3 in shift register W63 is fed to shift register W62, and the third 8-bit segment S_2 in shift register W62 is fed to shift register W61. The seven MSBs of the second 8-bit segment S_1 in shift register W61 are fed straight to the first input of the shift register W60. The LSB of the second 8-bit segment S_1 in shift register W61 is fed to the second multiplexer 175. This has been configured to output the LSB to the shift register W60, and the seven MSBs and the LSB of the second 8-bit segment S_1 are concatenated therein. The first 8-bit segment S_0 in shift register W60 is fed to shift register W59. The zero data in shift registers W59 to W1 is fed to shift registers W58 to W0. Data output from shift registers W52, W32, W8 and W0 is processed as before. The output module 21 receives the first 8-bit segment S_0 from the shift register W60, and outputs this to the hash value computation element 7, where it is used to compute a final hash value.

The first 8-bit segment S_0 is the first message schedule value W_0 .

The above process is repeated until the sixty fourth clock cycle, $t = 63$, where all of the sixty four 8-bit message segments of the first 512-bit message block have been loaded into the message schedule element 5, the sixty fourth 8-bit message segment S_{63} being contained in the shift register W63, and so on, to the first 8-bit message segment S_0 contained in the shift register W0. Data output from shift registers W52, W32, W8 and W0 is processed by the XOR module 15 and the rotate module 17, as before. The output of the message schedule element 5 over these 64 clock cycles, is 0, 0, 0, 0, followed by the message segments S_0 to S_{59} ,

which comprise the message schedule values W_0 to W_{59} . These are passed to the hash value computation element 7.

- 5 The control module 19 comprises a clock, and therefore is able to ascertain when clock cycle $t = 63$ has been reached. When this occurs, the control module 19 sends a new control signal to the input multiplexer 11. The control signal received by the input multiplexer 11 causes it to switch to receiving data on its second data input 112.
- 10 In the sixty fifth clock cycle, $t = 64$, the sixty fourth 8-bit message segment S_{63} in shift register W63 is fed to shift register W62, and the sixty third 8-bit segment S_{62} in shift register W62 is fed to shift register W61. The seven MSBs of the sixty second 8-bit segment S_{61} in shift register W61 are fed straight to the first input of the shift register W60. The LSB of the sixty
- 15 second 8-bit segment S_{61} in shift register W61 is fed to the second multiplexer 175. This is still configured to output the LSB to the shift register W60, and the seven MSBs and the LSB of the sixty second 8-bit segment S_{61} are concatenated therein. The 8-bit segments, S_{60} to S_1 , in shift registers W60 to W1 are fed to shift registers W59 to W0. The output
- 20 module 21 receives the sixty first 8-bit segment S_{60} from the shift register W60, and outputs this to the hash value computation element 7, where it is used to compute a final hash value. The sixty first 8-bit segment S_{60} is the sixty first message schedule value W_{60} . Data which is output from shift registers W52, W32, W8 and W0 is also passed to the XOR module 15.
- 25 This data comprises the 8-bit message segments S_{52} , S_{32} , S_8 and S_0 , i.e. the message schedule values W_{52} , W_{32} , W_8 and W_0 . These message schedule values undergo XOR functions according to the equation given above, and the XOR result is output to the rotate module 17. The XOR result is operated upon by the rotate module 17, and the rotate result is
- 30 passed to the second data input 112 of the input multiplexer 11. As the

input multiplexer 11 is now configured to receive data on its second data input 112, this first rotate result is fed into the input multiplexer 11, and from there is fed to the first shift register W63.

- 5 In the sixty sixth clock cycle, $t = 65$, the first rotate result in shift register W63 is fed into shift register W62, and the sixty fourth 8-bit segment S_{63} in shift register W62 is fed to shift register W61. The seven MSBs of the sixty third 8-bit segment S_{62} in shift register W61 are fed straight to the first input of the shift register W60. The LSB of the sixty third 8-bit segment S_{62}
- 10 in shift register W61 is fed to the second multiplexer 175. This is still configured to output the LSB to the shift register W60, and the seven MSBs and the LSB of the sixty third 8-bit segment S_{62} are concatenated therein. The 8-bit segments, S_{61} to S_2 , in shift registers W60 to W1 are fed to shift registers W59 to W0. The output module 21 receives the sixty
- 15 second 8-bit segment S_{61} from the shift register W60, and outputs this to the hash value computation element 7, where it is used to compute a final hash value. The sixty second 8-bit segment S_{61} is the sixty second message schedule value W_{61} . Data which is output from shift registers W52, W32, W8 and W0 is also passed to the XOR module 15. This data
- 20 comprises the 8-bit message segments S_{53} , S_{33} , S_9 and S_1 , i.e. the message schedule values W_{53} , W_{33} , W_9 and W_1 . These message schedule values undergo XOR functions according to the equation given above, and the XOR result is output to the rotate module 17. The XOR result is operated upon by the rotate module 17, and the rotate result is
- 25 passed to the second data input 112 of the input multiplexer 11. The input multiplexer 11 is still configured to receive data on its second data input 112, this second rotate result is fed into the input multiplexer 11, and from there is fed to the first shift register W63.

In the sixty seventh clock cycle, $t = 66$, the second rotate result in shift register W63 is fed into shift register W62, and the first rotate result in shift register W62 is fed to shift register W61. The seven MSBs of the sixty fourth 8-bit segment S_{63} in shift register W61 are fed straight to the first input of the shift register W60. The LSB of the sixty fourth 8-bit segment S_{63} in shift register W61 is fed to the second multiplexer 175. This is still configured to output the LSB to the shift register W60, and the seven MSBs and the LSB of the sixty fourth 8-bit segment S_{63} are concatenated therein. The 8-bit segments, S_{62} to S_3 , in shift registers W60 to W1 are fed to shift registers W59 to W0. The output module 21 receives the sixty third 8-bit segment S_{62} from the shift register W60, and outputs this to the hash value computation element 7, where it is used to compute a final hash value. The sixty third 8-bit segment S_{62} is the sixty third message schedule value W_{62} . Data which is output from shift registers W52, W32, W8 and W0 is also passed to the XOR module 15. This data comprises the 8-bit message segments S_{54} , S_{34} , S_{10} and S_2 , i.e. the message schedule values W_{54} , W_{34} , W_{10} and W_2 . These message schedule values undergo XOR functions according to the equation given above, and the XOR result is output to the rotate module 17. The XOR result is operated upon by the rotate module 17, and the rotate result is passed to the second data input 112 of the input multiplexer 11. The input multiplexer 11 is still configured to receive data on its second data input 112, this third rotate result is fed into the input multiplexer 11, and from there is fed to the first shift register W63.

25

In the sixty eighth clock cycle, $t = 67$, the control module 19 sends a control signal to the second multiplexer 175 of the rotate module 17, which control signal causes the second multiplexer 175 to receive data on its second data input, from the first multiplexer 173 of the rotate module 17. The third rotate result in shift register W63 is fed into shift register W62,

30

and the second rotate result in shift register W62 is fed to shift register W61. The seven MSBs of the first rotate result in shift register W61 are fed straight to the first input of the shift register W60. The LSB of the first rotate result is fed to the first data input of the second multiplexer 175.

5 This is now configured to ignore data received on its first data input, and to receive data on its second data input. The data received from the first multiplexer 173 on the second data input of the second multiplexer 175, is output to the shift register W60. The seven MSBs of the first rotate result and the data received from the first multiplexer 173 are concatenated in
10 the shift register W60. This forms the sixty fifth message schedule value, W_{64} . The 8-bit segments, S_{63} to S_4 , in shift registers W60 to W1 are fed to shift registers W59 to W0. The output module 21 receives the sixty fourth 8-bit segment S_{63} from the shift register W60, and outputs this to the hash value computation element 7, where it is used to compute a final hash
15 value. The sixty fourth 8-bit segment S_{63} is the sixty fourth message schedule value W_{63} . Data which is output from shift registers W52, W32, W8 and W0 is also passed to the XOR module 15. This data comprises the 8-bit message segments S_{55} , S_{35} , S_{11} and S_3 , i.e. the message schedule values W_{55} , W_{35} , W_{11} and W_3 . These message schedule values
20 undergo XOR functions according to the equation given above, and the XOR result is output to the rotate module 17. The XOR result is operated upon by the rotate module 17, and the rotate result is passed to the second data input 112 of the input multiplexer 11. The input multiplexer 11 is still configured to receive data on its second data input 112, this
25 fourth rotate result is fed into the input multiplexer 11, and from there is fed to the first shift register W63.

In the sixty ninth clock cycle, $t = 68$, the control module 19 sends a control signal to the second multiplexer 175 of the rotate module 17, which control
30 signal causes the second multiplexer 175 to swap back to receiving data

on its first data input, from the shift register W61. The fourth rotate result in shift register W63 is fed into shift register W62, and the third rotate result in shift register W62 is fed to shift register W61. The seven MSBs of the second rotate result in shift register W61 are fed straight to the first

5 input of the shift register W60. The LSB of the second rotate result is fed to the first data input of the second multiplexer 175. This is now configured to receive data on this first data input, and the LSB of the second rotate function is output to the shift register W60. The seven MSBs of the first rotate result and the LSB are concatenated in the shift

10 register W60. This forms the sixty sixth message schedule value, W_{65} . The sixty fifth message schedule value, W_{64} , and 8-bit segments, S_{63} to S_5 , in shift registers W60 to W1 are fed to shift registers W59 to W0. The output module 21 receives the sixty fifth message schedule value W_{64} from the shift register W60, and outputs this to the hash value computation

15 element 7, where it is used to compute a final hash value. Data which is output from shift registers W52, W32, W8 and W0 is also passed to the XOR module 15. This data comprises the 8-bit message segments S_{56} , S_{36} , S_{12} and S_4 , i.e. the message schedule values W_{56} , W_{36} , W_{12} and W_4 . These message schedule values undergo XOR functions according to the

20 equation given above, and the XOR result is output to the rotate module 17. The XOR result is operated upon by the rotate module 17, and the rotate result is passed to the second data input 112 of the input multiplexer 11. The input multiplexer 11 is still configured to receive data on its second data input 112, this fifth rotate result is fed into the input

25 multiplexer 11, and from there is fed to the first shift register W63.

The procedure for clock cycle $t = 68$, is repeated for clock cycles $t = 69$ and $t=70$. Thus it can be seen that when the first rotate result is transferred from shift register W61 to W60, the second data input of the

30 second multiplexer 175 is active, and the data received by the shift

register W60 is the seven MSBs of the first rotate result, and the data received on the second data input of the second multiplexer 175 (which is received from the first multiplexer 173). These are concatenated in the shift register W60, and form a message schedule value. When the
5 second, third and fourth rotate results are transferred from shift register W61 to W60, the first data input of the second multiplexer 175 is active, and the data received by the shift register W60 is the seven MSBs of the second, third or fourth rotate result, and the data received on the first data input of the second multiplexer 175, which is the LSB of the second, third
10 or fourth rotate result respectively. These are concatenated in the shift register W60, and form a message schedule value. This sequence of processing of the rotate results is repeated for each subsequent set of 4 rotate results.

15 The processing of the rotate results in the above manner is a consequence of using an 8-bit architecture for the calculation of the message schedule values W_{64} onwards. The SHA-1 has been formulated assuming that each 512-bit message block processed by the message schedule element is split up into sixteen 32-bit message segments. In the
20 invention, each 512-bit message block is split up into sixty four 8-bit message segments. The message schedule value calculations of the invention are carried out on 8-bit message segments, using a modified SHA-1 message schedule value definition. It is important that this modified SHA-1 message schedule value definition yields the same results
25 as would be achieved using an unmodified SHA-1 message schedule value definition on 32-bit message segments comprising four 8-bit message segments, i.e. for example that concatenation of message schedule values W_x to W_{x+3} calculated using four 8-bit message segments would be the same as message schedule value W_x calculated using a 32-
30 bit message segment comprising the four 8-bit message segments.

The calculation of message schedule values W_{64} onwards involves using the modified SHA-1 message schedule value definition, comprising XOR functions and a rotate to the left by one place function. When the XOR functions are carried out over four clock cycles on four 8-bit message segments or message schedule values or a combination of these, this yields the same overall result as if the XOR functions were carried out on a 32-bit segment comprising the four 8-bit message segments / schedule values. This is not the case for the rotate function. If a rotate to the left by one place function is carried out separately on each of four 8-bit XOR values, this will not yield the same result as carrying out a rotate to the left by one place function on a 32-bit value formed from concatenation of the four 8-bit XOR values. In order for this to be achieved, the message schedule element 5 of the invention is provided with the rotate module 17.

The operation of the rotate module 17 will be described with reference to Figure 3, which illustrates the operation of the rotate function on four 8-bit data segments as follows.

1110 0001 0110 0010 0110 0011 0110 0100

20

If these data segments are treated as a 32-bit segment, to rotate this one place to the left, each of the digits, apart from the MSB, are shifted one place to the left, and the MSB is placed at the end of the segment, i.e. becomes the LSB. The rotated data segment will be:

25

1100 0010 1100 0100 1100 0110 1100 1001

It can be seen that if each of the 8-bit data segments are individually rotated to the left by one place, the result will be:

30

1100 0011 1100 0100 1100 0110 1100 1000

which is not the same result achieved when the data segments are treated as a 32-bit data segment.

5

The rotate module 17 of the message schedule element 5 operates to achieve the same result as follows. As described above, when the sixty fifth clock cycle, $t = 64$, is reached, the message schedule values start to be calculated using the XOR and rotate functions of the SHA-1 equation.

10 The register 174 of the rotate module 17 is initialised to zero. The control module 19 sends a control signal to the first multiplexer 173 of the rotate module 17, and the control signal causes the first multiplexer 173 to output data to the register 174. In the sixty fifth clock cycle, $t = 64$, data which is output from shift registers W_{52} , W_{32} , W_8 and W_0 is passed to the XOR

15 module 15. This data comprises the 8-bit message segments S_{52} , S_{32} , S_8 and S_0 , i.e. the message schedule values W_{52} , W_{32} , W_8 and W_0 . These message schedule values undergo XOR functions according to the equation given above, and the XOR result, is output to the rotate module 17. In this example, this first XOR result comprises the first eight LSBs of

20 the 32-bit data segment given above (0110 0100). The first XOR result is passed to the data splitter of the rotate module 17, where it is split and the seven LSBs thereof (110 0100) are passed to the data combiner of the rotate module 17, and the MSB thereof (0) is passed to the first multiplexer 173. The first multiplexer 173 is configured to pass the MSB (0) to the

25 register 174. The MSB replaces the data (0) already stored in the register, and this data is passed to the data combiner, where it is combined with the seven LSBs of the first XOR function, to form the first rotate result (1100 1000) which is passed to the second data input 112 of the input multiplexer 11, and from there to shift register W_{63} . This is illustrated in

30 row $t = 64$ of Figure 3.

In the sixty sixth clock cycle, $t = 65$, data which is output from shift registers W52, W32, W8 and W0 is passed to the XOR module 15. This data comprises the 8-bit message segments S_{53} , S_{33} , S_9 and S_1 , i.e. the message schedule values W_{53} , W_{33} , W_9 and W_1 . These message

5 schedule values undergo XOR functions according to the SHA-1 equation given above, and the XOR result, is output to the rotate module 17. In this example, this second XOR result comprises the second eight LSBs of the 32-bit data segment given above (0110 0011). The second XOR result is passed to the data splitter of the rotate module 17, where it is split and the

10 seven LSBs thereof (110 0011) are passed to the data combiner of the rotate module 17, and the MSB thereof (0) is passed to the first multiplexer 173. The first multiplexer 173 is still configured to pass the MSB (0) to the register 174. The MSB replaces the data (0) already stored in the register, and this data (which is the MSB (0) of the first XOR result) is passed to the

15 data combiner, where it is combined with the seven LSBs of the second XOR function, to form the second rotate result (1100 0110) which is passed to the second data input 112 of the input multiplexer 11, and from there to shift register W63. The first rotate result (1100 1000) is passed from shift register W63 to shift register W62. This is illustrated in row $t =$

20 65 of Figure 3. It can be seen that the seven LSBs of the second XOR result are combined with the MSB of the first XOR result.

In the sixty seventh clock cycle, $t = 66$, data which is output from shift registers W52, W32, W8 and W0 is passed to the XOR module 15. This

25 data comprises the 8-bit message segments S_{54} , S_{34} , S_{10} and S_2 , i.e. the message schedule values W_{54} , W_{34} , W_{10} and W_2 . These message schedule values again undergo XOR functions according to the SHA-1 equation given above, and the XOR result, is output to the rotate module 17. In this example, this third XOR result comprises the third eight LSBs

30 of the 32-bit data segment given above (0110 0010). The third XOR result

is passed to the data splitter of the rotate module 17, where it is split and the seven LSBs thereof (110 0010) are passed to the data combiner of the rotate module 17, and the MSB thereof (0) is passed to the first multiplexer 173. The first multiplexer 173 is still configured to pass the MSB (0) to the register 174. The MSB replaces the data (0) already stored in the register, and this data (which is the MSB (0) of the second XOR result) is passed to the data combiner, where it is combined with the seven LSBs of the third XOR function, to form the third rotate result (1100 0100) which is passed to the second data input 112 of the input multiplexer 11, and from there to shift register W63. The second rotate result (1100 0110) is passed from shift register W63 to shift register W62, and the first rotate result (1100 1000) is passed from shift register W62 to shift register W61. This is illustrated in row $t = 66$ of Figure 3. It can be seen that the seven LSBs of the third XOR result are combined with the MSB of the second XOR result.

In the sixty eighth clock cycle, $t = 67$, the control module 19 sends a control signal to the first multiplexer 173 of the rotate module 17, which control signal causes the first multiplexer 173 to output data which it receives to the second multiplexer 175 of the rotate module 17. Data which is output from shift registers W52, W32, W8 and W0 is passed to the XOR module 15. This data comprises the 8-bit message segments S_{55} , S_{35} , S_{11} and S_3 , i.e. the message schedule values W_{55} , W_{35} , W_{11} and W_3 . These message schedule values undergo XOR functions according to the SHA-1 equation given above, and the XOR result, is output to the rotate module 17. In this example, this fourth XOR result comprises the fourth eight LSBs of the 32-bit data segment given above (1110 0001). The fourth XOR result is passed to the data splitter of the rotate module 17, where it is split and the seven LSBs thereof (110 0001) are passed to the data combiner of the rotate module 17, and the MSB thereof (1) is passed to the first multiplexer 173. The first multiplexer 173 is now

configured to pass the MSB (1) to the second multiplexer 175. The data already stored in the register, which is the MSB (0) of the third XOR result, is passed to the data combiner, where it is combined with the seven LSBs of the fourth XOR function, to form the fourth rotate result (1100 0010) which is passed to the second data input 112 of the input multiplexer 11, and from there to shift register W63. The third rotate result (1100 0100) is passed from shift register W63 to shift register W62, and the second rotate result (1100 0110) is passed from shift register W62 to shift register W61. The seven MSBs of the first rotate result (1100 100) in shift register W61 are fed straight to the first input of the shift register W60. The LSB of the first rotate result is fed to the first data input of the second multiplexer 175. This is now configured to ignore data received on its first data input, and to receive data on its second data input, from the first multiplexer 173. The data received from the first multiplexer 173 comprises the MSB (1) of the fourth rotate result, and this is output to the shift register W60. The seven MSBs of the first rotate result (1100 100) and the MSB (1) of the fourth rotate result are concatenated in the shift register W60, to form the sixty fifth message schedule value, W_{64} , (1100 1001). This is illustrated in row $t = 67$ of Figure 3.

20

Thus at the sixty eighth clock cycle, $t = 67$, the shift registers W63 to W60 contain 1100 0010, 1100 0100, 1100 0110 and 1100 1001, respectively. It can be seen that the original 32-bit data segment output in four 8-bit XOR results by the XOR module 15, has been rotated one place to the left, i.e. each of the digits, apart from the MSB, have been shifted one place to the left, and the MSB has been placed at the end of the segment, i.e. becomes the LSB, as a result of the rotate operations carried out by the rotate module 17. In subsequent clock cycles, these rotate results are output from the message schedule element 5, and form message schedule values W_{64} to W_{67} .

30

The process for clock cycles $t = 65$ to 67 is repeated for clock cycles $t = 69$ to 319 . Every fourth clock cycle, control signals are sent to the first multiplexer 173 causing it to send the MSB of the fourth rotate result to the second multiplexer 175. This multiplexer is configured to pass the MSB to shift register W60, where it is concatenated with the seven MSBs of the first rotate result. Thus every four 8-bit XOR results are treated as a 32-bit segment, and a 32-bit rotate one place to the left function is carried out, as required for the calculation of appropriate message schedule values.

At clock cycle $t = 319$, all of the first 512-bit data block has been fed into the message schedule element 5, and has been used to calculate the message schedule values. The message schedule element 5 has output 0,0,0,0 followed by message schedule values W_0 to W_{315} . Message schedule values W_{316} to W_{319} are still contained in shift registers W63 to W60. The clock then resets, and the entire process is started again. A second 512-bit data block is fed into the shift registers of the message schedule element 5, and used to calculate a second series of message schedule values. As the second data block is being fed into the shift registers, the message schedule values, W_{316} to W_{319} , for the first data block are output from the message schedule element 5. The message schedule values calculated for each 512-bit data block are passed to the hash value computation element 7, and used to compute a final hash value as follows.

The hash value computation element 7 receives 320 message schedule values for each 512-bit data block, and uses the message schedule values in the computation of a hash value for each data block. As each data block hash value is computed, it is used in the computation of the next data block hash value. The hash value for the last data block depends

upon the hash values for each of the preceding data blocks, and forms the final hash value.

For each set of message schedule values, W_t , and hence each 512-bit data block, the hash value comprises a concatenation of five 32-bit hash variables, a, b, c, d and e. The initial values of the hash variables used in the hash value computation for the first data block comprise the key defined by the SHA-1. The hash value computation module computes the hash variables using modifications of hash variable definitions specified by the SHA-1. The modifications change the hash variable definitions from definitions formulated to use 32-bit message schedule values and 32-bit key segments to definitions formulated to use 8-bit message schedule values and 8-bit key segments.

The modified hash variable definitions comprise

$$T = ROT_{LEFT-5}(a) + F_t(b, c, d) + e + K_t + W_t$$

$$e = d$$

$$d = c$$

$$c = ROT_{LEFT-30}(b)$$

$$b = a$$

$$a = T$$

$$F_t(b, c, d) = \begin{cases} (b \text{ AND } c) \text{ OR } (\bar{b} \text{ AND } d) & 0 \leq t \leq 79 \\ b \oplus c \oplus d & 80 \leq t \leq 159 \\ (b \text{ AND } c) \text{ OR } (b \text{ AND } d) \text{ OR } (c \text{ AND } d) & 160 \leq t \leq 239 \\ b \oplus c \oplus d & 240 \leq t \leq 319 \end{cases}$$

Referring to Figure 4, the hash value computation element 7 comprises an input multiplexer 710, twenty 8-bit shift registers 720, a rotate hash

variable b module 730 and a T computation module 740. The T computation module 740 comprises a rotate hash variable a module 750, an F computation module 760, an add device 770 and a carry register 780. The input multiplexer 710 is configured to receive data on a first data input. The initial values of each of the 32-bit hash variables a to e, set by the SHA-1, are input into the first data input of the input multiplexer 710, and from there into the shift registers 720, in four 8-bit segments for each hash variable. The hash variable segments are loaded into the shift registers 720 starting with hash variable segment e3, such that after 20 clock cycles, the shift registers 720 reading from left to right of Figure 4 contain the hash variable segments a0, a1, a2, a3, b0, b1, b2, b3, c0, c1, c2, c3, d0, d1, d2, d3, e0, e1, e2, e3. During these clock cycles, a direct connection is provided between shift register number 8 and shift register number 9, from the left, i.e. between the shift register which, at the end of the 20 clock cycles, contains b3 and the shift register which contains c0.

The hash value computation element 7 then operates over 320 clock cycles, $t = 0$ to 319, to receive 320 message schedule values W_0 to W_{319} calculated using the first 512-bit data block, and to compute values for the hash variable segments a0 to e3 for the first data block. The input multiplexer 710 receives a control signal which causes it to accept data received on its second data input and to output this data to the shift registers 720. In the first clock cycle, $t = 0$, with the exception of shift register e3, the contents of each of the shift registers 720 are output to the shift register immediately to the right. The contents of shift register e3 is output to the add device 770 of the T computation module 740. The add device 770 also receives the first message schedule value W_0 , and a first value K_0 of a constant K_t specified by the SHA-1. The add device 770 further receives F_0 , which has been calculated according to the equation given above, using the outputs of shift registers b3, c3 and d3, which are

the initial values of b3, c3 and d3 respectively, set by the SHA-1. F_0 is calculated using 8-bit hash variable segments of 32-bit hash variables. As the calculation of F_0 involves only XOR, AND, OR and NOT functions, using 8-bit hash variable segments will achieve an equivalent result to that which would be achieved using 32-bit hash variables. The add device 770 further receives an output from the rotate 'a' module 750. The add device 770 then has all of the parameters necessary to compute T_0 using the equation for T given above. T_0 is computed by adding the 8-bit parameters which it receives, using modulo 2^{32} arithmetic. When performing this addition using 8-bit parameters, it is desired to obtain the same result as would be obtained if the addition were performed using 32-bit parameters. To achieve this, a carry value from the carry register 780 is used as appropriate. The computed T_0 is then output from the T computation module 740 to the second data input of the input multiplexer 710, and is input into the shift register a0.

The computation of T involves a rotate to the left by 5 places function operated on 8-bit hash variable segments a0 to a3. As explained above, it is desired to obtain the same result for the rotate function using 8-bit hash variable segments as would be obtained if a 32-bit hash variable were to be rotated. To achieve this, the rotate 'a' module 750 is used. This comprises a multiplexer 751 which has a first data input connected to the output of shift register a0, a second data input connected to the output of the shift register b0, and an output connected to a first input of a data combiner 752. A second input of the data combiner 752 is connected to the shift register a3, and an output of the data combiner is connected to the add device 770. The rotate 'a' module 750 operates over four clock cycles to perform the rotate function on the 8-bit hash variable segments a0 to a3, and to obtain the same result as rotating an equivalent 32-bit

hash variable, as illustrated in Figure 5. The first row of Figure 5 shows the initial values of the hash variable segments a0 to a3, i.e.

0110 0001 0110 0010 0110 0011 0110 0100

5

If this is treated as a single 32-bit hash variable and rotated 5 places to the left, the result will be

0010 1100 0100 1100 0110 1100 1000 1100

10

The same result is achieved over four clock cycles using the rotate 'a' module 750, as follows. In the first clock cycle, the contents of shift register a0 (i.e. a0 = 0110 0001) are placed in shift register a1, the five MSBs of a0 (0110 0) are output to the first input of the multiplexer 751, the contents of shift register a1 (i.e. a1 = 0110 0010) are placed in shift register a2, the contents of shift register a2 (i.e. a2 = 0110 0011) are placed in shift register a3, the contents of shift register a3 (i.e. a3 = 0110 0100) are placed in shift register b0, the three LSBs of a3 (100) are output to the first input of the data combiner 752, and the five MSBs of the contents of b0 are output to the second data input of the multiplexer 751. In this clock cycle, this multiplexer is configured to accept data received on its first data input, and to output this to the second data input of the data combiner 752. Thus the data combiner 752 receives the three LSBs of a3 (100) and the five MSBs of a0 (0110 0), and concatenates them to get the first byte of the rotate result (1000 1100), which is output to the add device 770.

In the second clock cycle, the contents of shift register a0 (T_0) is placed in shift register a1, the five MSBs of T_0 are output to the first input of the multiplexer 751, the contents of shift register a1 (i.e. a0 = 0110 0001) are

30

placed in shift register a2, the contents of shift register a2 (i.e. a1 = 0110 0010) are placed in shift register a3, the contents of shift register a3 (i.e. a2 = 0110 0011) are placed in shift register b0, the three LSBs of a2 (011) are output to the first input of the data combiner 752, and the five MSBs of the contents of b0 (i.e. the five MSBs of a3, 0110 0) are output to the second data input of the multiplexer 751. In this clock cycle, this multiplexer is configured to accept data received on its second data input, and to output this to the second data input of the data combiner 752. Thus the data combiner 752 receives the three LSBs of a2 (011) and the five MSBs of a3 (0110 0), and concatenates them to get the second byte of the rotate result (0110 1100), which is output to the add device 770.

In the third and fourth clock cycles, the same procedure as in the second clock cycle is followed, as can be seen from Figure 5, to get the third and fourth bytes of the rotate result, which are also output to the add device 770. Thus the rotate function is carried out on the four 8-bit hash variable segments a0 to a3.

The computation of hash variable segments c0 to c3 according to the SHA-1, involve a rotate to the left by 30 places function operating on the hash variable segments b0 to b3. Again, it is desired to obtain the same result for the rotate function using the 8-bit hash variable segments as would be obtained if a 32-bit hash variable were to be rotated. This is achieved using the rotate 'b' module 730. This comprises a multiplexer 731, a first register 732, a second register 733, a third register 734, a data splitter 735 and a data combiner 736. A first data input of the multiplexer 731 is connected to the output of the shift register b2, a second data input of the multiplexer 731 is connected to the output of the third register 734, and an output of the multiplexer is connected to a first data input of the data combiner 736. A data input of the data splitter 735 is connected to

the output of the shift register b3. A first output of the data splitter 735 is connected to an input of the first register 732, and a second data output of the data splitter 735 is connected to a second input of the data combiner 736. An output of the first register 732 is connected to an input of the second register 733, and an output of the second register 733 is connected to an input of the third register 734. The contents of the first, second and third registers 732, 733, 734 are initialised to zero. The rotate 'b' module 730 operates over four clock cycles to perform the rotate function on the 8-bit hash variable segments b0 to b3, and to obtain the same result as rotating an equivalent 32-bit hash variable, as illustrated in Figure 6. The first row of Figure 6 shows the initial values of the hash variable segments b0 to b3, i.e.

0110 0001 0110 0010 0110 0011 0110 0100

15

For a 32-bit data element, a rotate to the left by 30 places function is equivalent to a rotate to the right by 2 places. If the four 8-bit hash variable segments are treated as a single 32-bit hash variable and rotated 2 places to the right, the result will be

20

0001 1000 0101 1000 1001 1000 1101 1001

The same result is achieved over four clock cycles using the rotate 'b' module 730, as follows. In the first clock cycle, the contents of shift register b0 (i.e. b0 = 0110 0001) are placed in shift register b1, the contents of shift register b1 (i.e. b1 = 0110 0010) are placed in shift register b2, the contents of shift register b2 (i.e. b2 = 0110 0011) are placed in shift register b3, the two LSBs of b2 (11) are output to the first data input of the multiplexer 731, the contents of shift register b3 (i.e. b3 = 0110 0100) are output to the input of the data splitter 735, the two LSBs of

30

b3 (00) are output to the first register 732, the six MSBs of b3 (0110 01) are output to the second data input of the data combiner 736, the contents (0) of the first register 732 are output to the second register 733, the contents (0) of the second register 733 are output to the third register 734, and the contents (0) of the third register 734 are output to the second data input of the multiplexer 731. In this clock cycle, this multiplexer is configured to accept data received on its first data input, and to output this to the first data input of the data combiner 736. Thus the data combiner 736 receives the two LSBs of b2 (11) and the six MSBs of b3 (0110 01), and concatenates them to get the first byte of the rotate result (1101 1001), which is output to the shift register c0.

In the second clock cycle, the contents of shift register b0 are placed in shift register b1, the contents of shift register b1 (i.e. b0 = 0110 0001) are placed in shift register b2, the contents of shift register b2 (i.e. b1 = 0110 0010) are placed in shift register b3, the two LSBs of b1 (10) are output to the first data input of the multiplexer 731, the contents of shift register b3 (i.e. b2 = 0110 0011) are output to the input of the data splitter 735, the two LSBs of b2 (11) are output to the first register 732, the six MSBs of b2 (0110 00) are output to the second data input of the data combiner 736, the contents (the two LSBs of b3 (00) stored there in the first clock cycle) of the first register 732 are output to the second register 733, the contents (0) of the second register 733 are output to the third register 734, and the contents (0) of the third register 734 are output to the second data input of the multiplexer 731. In this clock cycle, this multiplexer is also configured to accept data received on its first data input, and to output this to the first data input of the data combiner 736. Thus the data combiner 736 receives the two LSBs of b1 (10) and the six MSBs of b2 (0110 00), and concatenates them to get the second byte of the rotate result (1001 1000), which is output to the shift register c0.

The same process is carried out in the third clock cycle, to get the third byte of the rotate result (0101 1000), which is output to the shift register c0.

5 In the fourth clock cycle, the contents of shift register b0 are placed in shift register b1, the contents of shift register b1 are placed in shift register b2, the contents of shift register b2 are placed in shift register b3, the two LSBs of the contents of shift register b2 are output to the first data input of the multiplexer 731, the contents of shift register b3 (i.e. b0 = 0110 0001)
 10 are output to the input of the data splitter 735, the two LSBs of b0 (01) are output to the first register 732, the six MSBs of b0 (0110 00) are output to the second data input of the data combiner 736, the contents (the two LSBs of b1 stored there in the third clock cycle) of the first register 732 are output to the second register 733, the contents (the two LSBs of b2 stored
 15 there in the second clock cycle) of the second register 733 are output to the third register 734, and the contents (the two LSBs of b3 (00) stored there in the first clock cycle) of the third register 734 are output to the second data input of the multiplexer 731. In this clock cycle, this multiplexer is configured to accept data received on its second data input,
 20 and to output this to the first data input of the data combiner 736. Thus the data combiner 736 receives the two LSBs of b3 (00) and the six MSBs of b0 (0110 00), and concatenates them to get the fourth byte of the rotate result (0001 1000), which is output to the shift register c0. Thus the rotate function is carried out on the four 8-bit hash variable segments b0 to b3 to
 25 give the 8-bit hash variable segments c0 to c3.

The above process of moving the contents of the shift registers 720 one place to the right and computing T and c, is carried out for the clock cycles $t = 0$ to 319, and results in final hash variable segments a0 to e3 for the
 30 first 512-bit data block, which have been correctly computed using the

message schedule values W_0 to W_{319} of the first 512-bit data block. The final hash variable segments a0 to e3 for the first 512-bit data block are then input into the input multiplexer 710 of the hash value computation element 7 in 20 clock cycles as before, and are used in the computation of the final hash variable segments a0 to e3 for the second 512-bit data block. This is repeated until final hash variable segments a0 to e3 have been computed for the last 512-bit data block. These final hash variable segments a0 to e3 for the last 512-bit data block are output from the hash value computation element 7, and are then concatenated to form a 160-bit hash value for the message. The hash value is output to the output module 9.

The 8-bit hash value computation device architecture has been implemented using the Faraday UMC 180 nm (L180 GII) and UMC 130 nm (L130 LL) CMOS libraries. It has been tested using Modelsim, synthesised using Synopsys Physical Compiler Version 2006.06 and its power consumption obtained from Synopsys PrimeTime PX Version 2007.06. The performance results of the 8-bit hash value computation device of the invention are an area of 5527 gates, power consumption of 2,32 μ W at 100 kHz, and timing of 344 cycles. The shift registers used to store the message schedule values in the hash value computation device of the invention account for 2560 gates and 46% of the overall architecture, when implemented on 130 nm technology. It is important that the power consumption of the architecture of the hash value computation device of the invention meets the limitations imposed by RFID tags when it is used in this application. The current consumption of a security architecture for implementation on RFID tags should not exceed 15 μ A. For 1.3 V and 1.8 V CMOS technologies, this is equivalent to 18 μ W and 27 μ W respectively. The proposed 8-bit hash value computation device meets these power constraints comfortably. In RFID tags the

silicon area requirement significantly impacts the cost and the cost per mm^2 of silicon is estimated at 4 cent. Therefore, it is vital that the silicon area overhead resulting from the inclusion of security on low-cost tags is kept to a minimum. The hash value computation device architecture of the invention is within close reach of current RFID tag deployment. This is achieved by using an 8-bit architecture, which reduces the previous 32-bit XOR, AND, NOT and OR functions to equivalent 8-bit functions with no logic overhead, and the previous 32-bit addition modulo 2^{32} and previous rotate functions to equivalent 8-bit functions with minimal control logic overhead. Using this design methodology, the overall saving is approximately 1200 gates. In relation to timing, an RFID tag must response to a reader's request within 32 μs in accordance with the ISO/IEC 18000 standard. An interleaved challenge- response protocol could be used, in which the response time is 18 ms. This corresponds to 1800 clock cycles when operating with a clock frequency of 100 kHz. The hash value computation device of the invention also satisfies this requirement.

CLAIMS

1. A hash value computation device for computing a hash value for a message using a hash algorithm that defines a key, the device comprising
- 5 a message schedule element having an input module which receives the message in 8-bit segments, a plurality of 8-bit memory modules which store the 8-bit segments of the message, an 8-bit message schedule value module which uses the 8-bit message segments and the hash algorithm to provide an 8-bit message schedule value for each 8-bit
- 10 message segment, and an output module which outputs the 8-bit message schedule values, and
- a hash value computation element comprising a first input module which receives the 8-bit message schedule values, a second input module which receives 8-bit segments of the key defined by the hash algorithm, a
- 15 plurality of 8-bit memory modules which store the 8-bit segments of the key, a hash value computation module which uses the 8-bit message schedule values and the 8-bit key segments to compute a hash value for the message, and an output module which outputs the hash value.
- 20 2. A hash value computation device according to claim 1, in which the hash value computation device comprises a message formatting module, which formats the message into a format appropriate for the hash algorithm used to compute the hash value.
- 25 3. A hash value computation device according to claim 1 or claim 2, in which the message schedule element operates over the n clock cycles to calculate n message schedule values, and the hash value computation module operates over the n clock cycles to use the n message schedule values to compute the hash value for the message.

4. A hash value computation device according to claim 3, in which the input module of the message schedule element comprises a multiplexer, and in each clock cycle of a first subset of the n clock cycles, the multiplexer is configured to receive an 8-bit message segment and pass the 8-bit message segment to a first memory module of the plurality of 8-bit memory modules.
5. A hash value computation device according to any preceding claim, in which the plurality of 8-bit memory modules of the message schedule element are arranged in a linear array.
6. A hash value computation device according to claim 5, in which in a first clock cycle of the first subset of the n clock cycles, the first 8-bit memory module in the array receives a first 8-bit message segment, in a second clock cycle of the first subset of the n clock cycles, the first 8-bit memory module in the array receives a second 8-bit message segment, and the second 8-bit memory module in the array receives the first 8-bit message segment, etc. for each of the clock cycles in the subset of clock cycles.
7. A hash value computation device according to any preceding claim, in which each of the plurality of 8-bit memory modules comprises an 8-bit shift register.
8. A hash value computation device according to any preceding claim, in which the 8-bit message schedule value module provides 8-bit message schedule values determined using modifications of a message schedule value definition specified by the hash algorithm.

9. A hash value computation device according to claim 8, in which the modifications change the message schedule value definition from a definition formulated to use 32-bit data segments to a definition formulated to use 8-bit data segments.

5

10. A hash value computation device according to claim 8 or claim 9, in which the modified message schedule value definition comprises

$$W_t = \begin{cases} Message_t & 0 \leq t \leq 63 \\ 32bitROT_{LEFT\ 1}(W_{t-12} \oplus W_{t-32} \oplus W_{t-56} \oplus W_{t-64}) & 64 \leq t \leq 319 \end{cases}$$

10

11. A hash value computation device according to claim 10, in which the 8-bit message schedule value module uses a first part of the modified message schedule value definition to provide a first set of 8-bit message schedule values which each comprise an 8-bit message segment of the message.

15

12. A hash value computation device according to claim 10 or claim 11, in which the 8-bit message schedule value module uses a second part of the modified message schedule value definition to provide a second set of 8-bit message schedule values calculated using one or more functions carried out on one or more previously-determined message schedule values.

20

13. A hash value computation device according to claim 12, in which the functions comprise one or more logical functions, comprising any of one or more XOR functions, one or more rotate functions.

25

14. A hash value computation device according to claim 13, in which the 8-bit message schedule value module comprises an 8-bit XOR module

which carries out one or more XOR functions of the second part of the modified message schedule value definition.

15. A hash value computation device according to claim 14, in which
5 the 8-bit XOR module receives one or more previously-determined 8-bit message schedule values and carries out the one or more XOR functions on these, to provide one or more 8-bit XOR values.

16. A hash value computation device according to claim 13, in which
10 the 8-bit message schedule value module comprises a rotate module which carries out one or more rotate functions of the second part of the modified message schedule value definition.

17. A hash value computation device according to claim 16, in which
15 the one or more rotate functions comprises a rotate to the left by one place rotate function.

18. A hash value computation device according to claim 16 or claim 17,
20 in which the rotate module receives a plurality of 8-bit XOR values and carries out the one or more rotate functions on these, to provide a plurality of 8-bit message schedule values.

19. A hash value computation device according to any of claims 16 to
25 18, in which the rotate module comprises a data splitter, a first multiplexer, a register, a data combiner and a second multiplexer.

20. A hash value computation device according to any of claims 16 to
30 19, in which the 8-bit message schedule value module comprises a control module which is used to control the operation of the rotate module.

21. A hash value computation device according to any preceding claim, in which the hash value computation module computes a hash value for the message comprising a concatenation of a plurality of hash variables.

5 22. A hash value computation device according to claim 21, in which the hash value computation module computes the hash variables using modifications of hash variable definitions specified by the hash algorithm.

23. A hash value computation device according to claim 22, in which
10 the modifications change the hash variable definitions from definitions formulated to use 32-bit message schedule values and 32-bit key segments to definitions formulated to use 8-bit message schedule values and 8-bit key segments.

15 24. A hash value computation device according to claim 22 or claim 23, in which the modified hash variable definitions comprises

$$T = ROT_{LEFT-5}(a) + F_t(b, c, d) + e + K_t + W_t$$

$$e = d$$

20 $d = c$

$$c = ROT_{LEFT-30}(b)$$

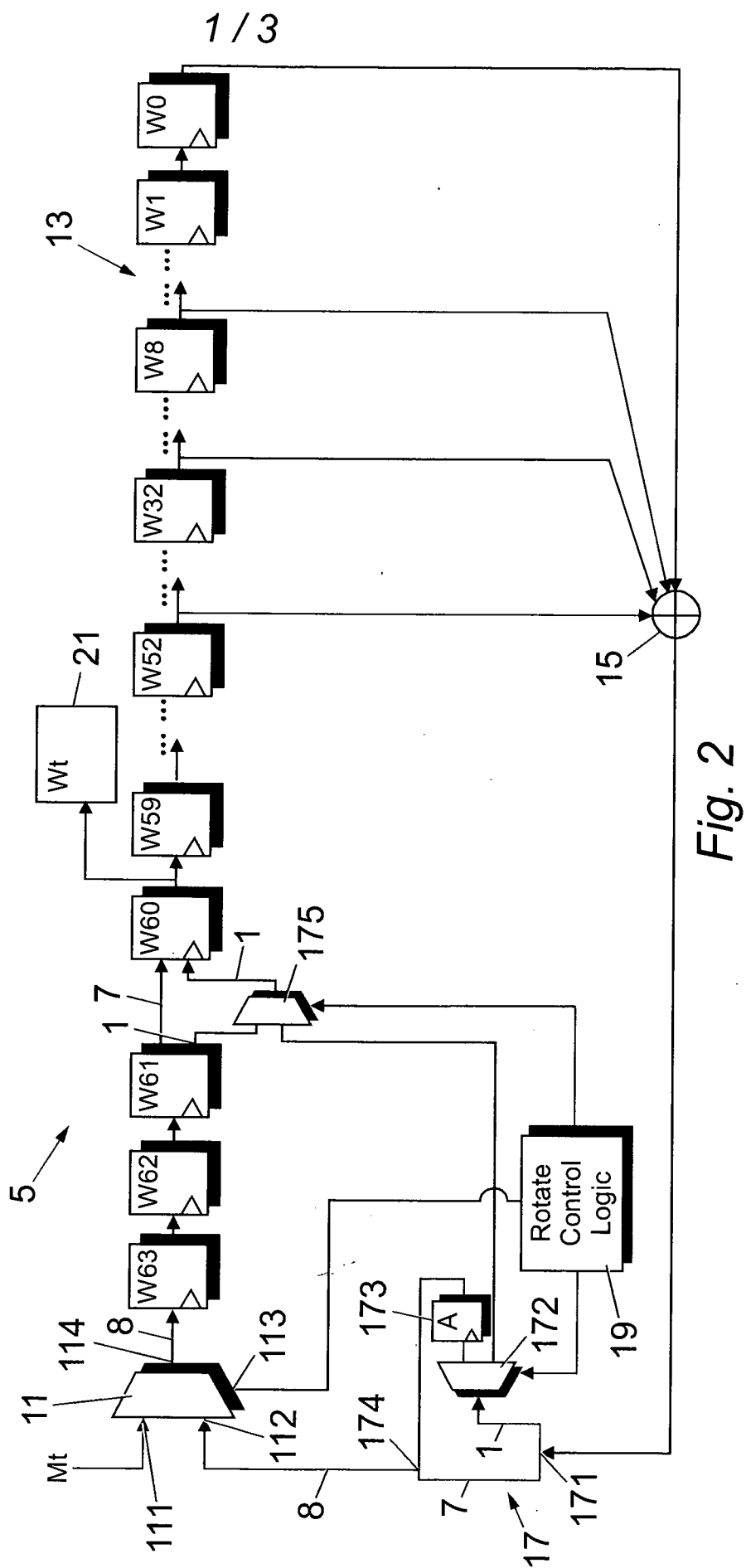
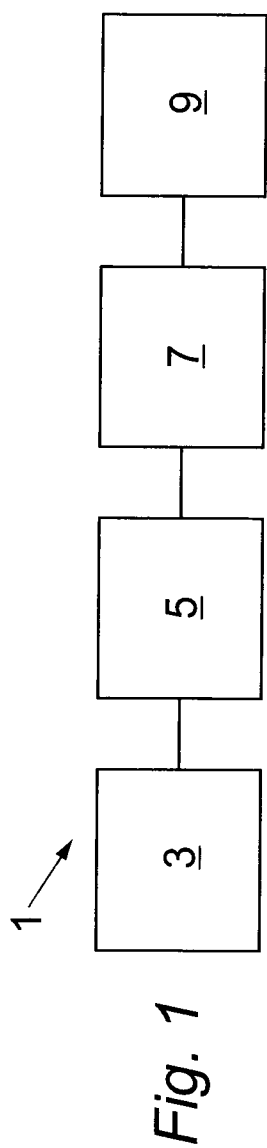
$$b = a$$

$$a = T$$

$$F_t(b, c, d) = \begin{cases} (b \text{ AND } c) \text{ OR } (\bar{b} \text{ AND } d) & 0 \leq t \leq 79 \\ b \oplus c \oplus d & 80 \leq t \leq 159 \\ (b \text{ AND } c) \text{ OR } (b \text{ AND } d) \text{ OR } (c \text{ AND } d) & 160 \leq t \leq 239 \\ b \oplus c \oplus d & 240 \leq t \leq 319 \end{cases}$$

25 where a, b, c, d and e are hash variables.

25. A hash value computation device according to any of claims 21 to 24, in which the hash value computation module computes the hash value using 8-bit key segments of initial values of the hash variables defined by the hash algorithm.
- 5
26. A hash value computation device according to claim 24, in which the hash value computation module comprises a rotate hash variable b module and a T computation module.
- 10
27. A hash value computation device according to claim 26, in which the T computation module comprises a rotate hash variable a module, an F computation module, an add device and a carry register.
- 15
28. A hash value computation device according to any preceding claim, in which the hash algorithm is a Secure Hash Algorithm.
29. A method of computing a hash value for a message using a hash algorithm that defines a key, comprising
- 20
- using a message schedule element of the hash value computation device of any of claims 1 to 28 to determine a plurality of 8-bit message schedule values, and
- using a hash value computation element of the hash value computation device of any of claims 1 to 28 to compute the hash value for the message using the message schedule values.
- 25
30. A data security device comprising a hash value computation device according to any of claims 1 to 28.
31. A data security device according to claim 30 which comprises an
- 30
- RFID tag.



	Input	W63	W62	W61	W60/Output
Cycle 1	x0: 0 110 0100	1100 1000	1100 1000	1100 1000	1100 1001
Cycle 2	x1: 0 110 0011	1100 0110	1100 0110	1100 0110	1100 0110
Cycle 3	x2: 0 110 0010	1100 0100	1100 0100	1100 0100	1100 0100
Cycle 4	x3: 1 110 0001	1100 0010	1100 0010	1100 0010	1100 0010

Fig. 3

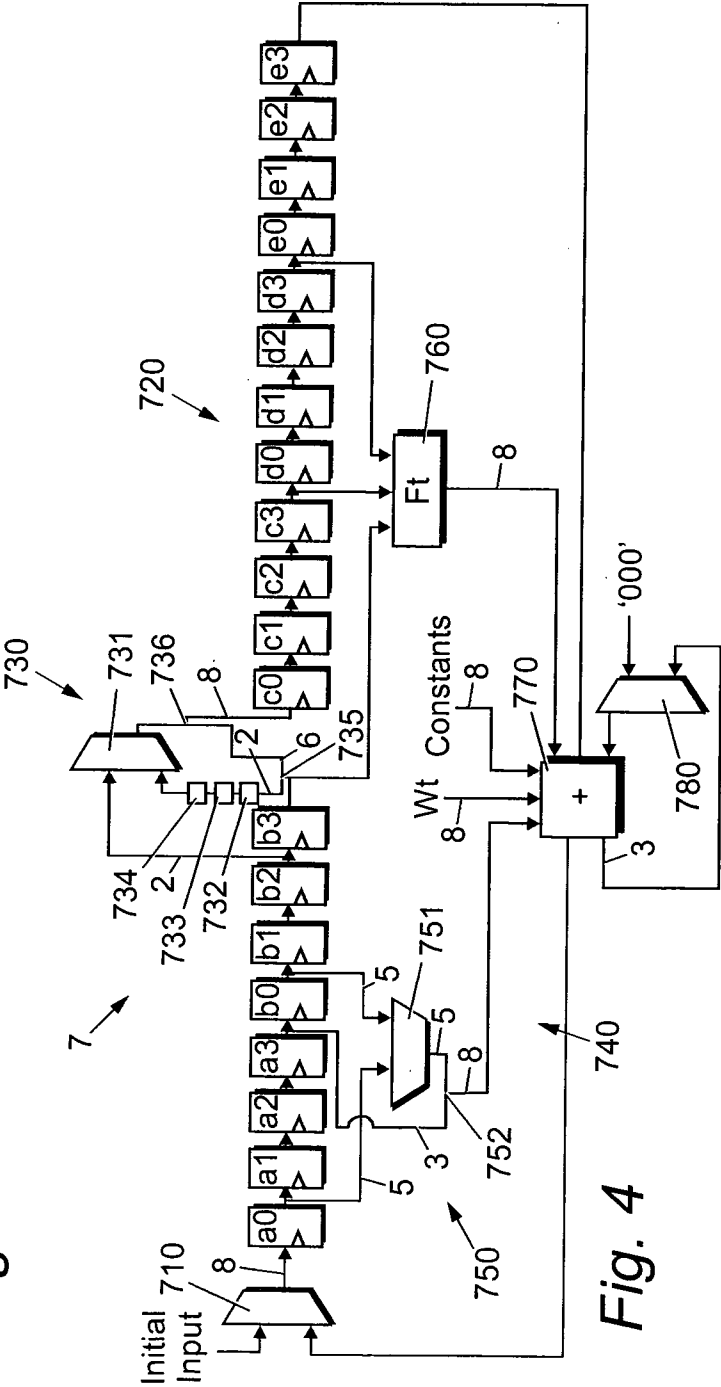


Fig. 4

Input	b0	b1	b2	b3	c0	c1	c2	c3
Cycle 1	-	0110 0001	0110 0011	0110 0100	1101 1001	-	-	-
Cycle 2	-	-	0110 0010	0110 0011	1001 1000	1101 1001	-	-
Cycle 3	-	-	-	0110 0001	0101 1000	1001 1000	1101 1001	-
Cycle 4	-	-	-	-	0001 1000	0101 1000	1001 1000	1101 1001

Fig. 6

Input	a0	a1	a2	a3	b0	Rot(a):byte 3	Rot(a):byte 2	Rot(a):byte 1	Rot(a):byte 0
Cycle 1	0110 0001	0110 0010	0110 0011	0110 0100	-	1000 1100	-	-	-
Cycle 2	-	-	0110 0001	0110 0010	0110 0011	0110 1100	1000 1100	-	-
Cycle 3	-	-	-	0110 0001	0110 0010	0100 1100	0110 1100	1000 1100	-
Cycle 4	-	-	-	-	0110 0001	0010 1100	0100 1100	0110 1100	1000 1100

Fig. 5