(12) INTERNATIONAL APPLICATION PUBLISHED UNDER THE PATENT COOPERATION TREATY (PCT)

(19) World Intellectual Property Organization
International Bureau

(43) International Publication Date
16 January 2025 (16.01.2025)

WIPO | PCT

(10) International Publication Number
WO 2025/012631 A1

(51) International Patent Classification:
H04L 9/06 (2006.01) *H04L 9/00* (2022.01)

(21) International Application Number:
PCT/GB2024/051785

(22) International Filing Date:
09 July 2024 (09.07.2024)

(25) Filing Language: English

(26) Publication Language: English

(30) Priority Data:
2310704.8 12 July 2023 (12.07.2023) GB

(71) Applicant: **QUANTUM BLOCKCHAIN TECHNOLOGIES PLC** [GB/GB]; First Floor, 1 Chancery Lane, London Greater London WC2A 1LF (GB).

(72) Inventor: **NAIK, Rahul**; First Floor, 1 Chancery Lane, London Greater London WC2A 1LF (GB).

(74) Agent: **HARRIS, Thomas** et al.; D2 Knowledge Gateway, Nesfield Road, Colchester Essex CO4 3ZL (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, BN, BR, BW, BY, BZ, CA, CH, CL, CN, CO, CR, CU, CV, CZ, DE, DJ, DK, DM, DO, DZ, EC, EE, EG, ES, FI, GB, GD, GE, GH, GM, GT, HN, HR, HU, ID, IL, IN, IQ, IR, IS, IT, JM, JO, JP, KE, KG, KH, KN, KP, KR, KW, KZ, LA, LC, LK, LR, LS, LU, LY, MA, MD, MG, MK, MN, MU, MW, MX, MY, MZ, NA, NG, NI, NO, NZ, OM, PA, PE, PG, PH, PL, PT, QA, RO, RS, RU, RW, SA, SC, SD, SE, SG, SK, SL, ST, SV, SY, TH, TJ, TM, TN, TR, TT, TZ, UA, UG, US, UZ, VC, VN, WS, ZA, ZM, ZW.

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

Published:
— *with international search report (Art. 21(3))*
— *before the expiration of the time limit for amending the claims and to be republished in the event of receipt of amendments (Rule 48.2(h))*

(54) Title: REUSE OF MESSAGE SCHEDULING COMPUTATIONS FOR BITCOIN MINING

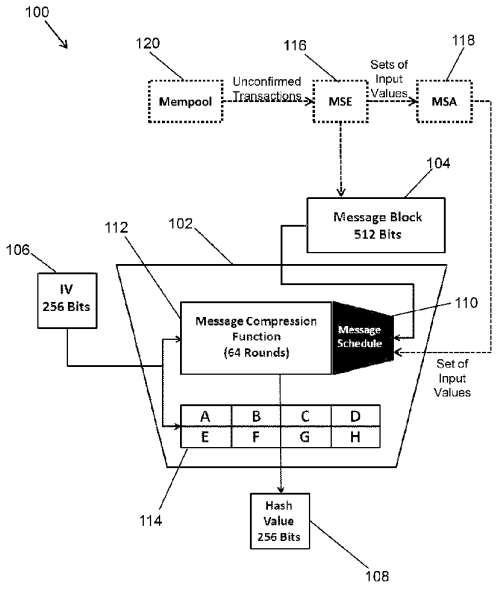

FIG 1

(57) Abstract: Cryptographic hashing is performed in respect of a group of plural different input messages (104). A message scheduling engine (116) pre-calculates a message schedule array (118) comprising plural different sets of input values for a cryptographic compression function. Each set of input values in the message schedule array (118) corresponds to a respective input message of the group of plural different input messages (104). After the message schedule array (118) has been pre-calculated by the message scheduling engine (116), a cryptographic compressor (112) performs the cryptographic compression function in respect of each input message of the group of plural different input messages (104). Performing the cryptographic compression function in respect of each input message comprises using the set of input values in the message schedule array (118) corresponding to that input message so as to generate a message digest (108) for that input message.

WO 2025/012631 A1

REUSE OF MESSAGE SCHEDULING COMPUTATIONS FOR BITCOIN MINING

FIELD

The present invention relates to message scheduling when performing cryptographic hashing in respect of a group of plural different input messages to be hashed. The present invention relates particularly, although not exclusively, to cryptographic hashing when mining for cryptocurrency, such as Bitcoin.

BACKGROUND

Bitcoin mining comprises applying three instances of a SHA256 cryptographic hashing function (referred to herein respectively as "$SHA256_0$," "$SHA256_1$" and "$SHA256_2$") to input messages, which are derived from a Bitcoin Block Header. $SHA256_0$ uses a standard initialisation vector IV and is applied to a first portion of the Bitcoin Block Header. $SHA256_1$ then uses the message digest H_0 of $SHA256_0$ as an initialisation vector and is applied to a second portion of the Bitcoin Block Header plus some padding. Finally, $SHA256_2$ again uses the standard initialisation vector IV and is applied to the message digest of $SHA256_1$ plus some padding.

The ultimate aim of Bitcoin mining is to find a $SHA256_2$ message digest or "hash" that is lower than or equal to the current Target. If the Target condition is met, then the Bitcoin Block Header in question is accepted by the Bitcoin Network and the Bitcoin miner receives a mining award. On the other hand, if the Target condition is not met, the Bitcoin miner will need to perform cryptographic hashing

in respect of one or more different Bitcoin Block Headers until the specified Target condition is met.

The different Bitcoin Block Headers are typically generated by incrementing a Nonce field in the second portion of the Bitcoin Block Header. In this case, since the first portion of the Bitcoin Block Header remains unchanged, the output of $SHA256_0$ is unchanged and there is no need to re-perform $SHA256_0$ for each new Nonce. Once all values for the Nonce have been exhausted, then a Bitcoin Block Header that references different transactions, by way of a different HashMerkleRoot, might be considered. Since the HashMerkleRoot spans both parts of the Bitcoin Block Header, $SHA256_0$ then needs to be re-performed to generate a new message digest H_0, and $SHA256_1$ then needs to be re-applied to the second portion of the Bitcoin Block Header using that new message digest H_0 as the initialisation vector. Further different Bitcoin Block Headers can again be generated by incrementing the Nonce field in the second portion of the Bitcoin Block Header.

Bitcoin mining can be extremely computationally expensive and various optimisations have therefore been proposed. For example, Rahul P. Naik, "*Optimising the SHA256 Hashing Algorithm for Faster and More Efficient Bitcoin Mining*", MSc Information Security Department of Computer Science UCL, 2013, describes various optimisations which make use of the fact that certain portions of the Bitcoin Block Header remain unchanged, or are slow to change, or are merely incremented, across plural different input messages. US 2018/0006808 A1 describes further optimisations which use hardwired values for the portions of input messages that always remain the same.

The above optimisations go some way to reducing the computational burden but further optimisations, particularly relating to message scheduling, are possible. In this regard, when performing the cryptographic hashing, a message scheduler is used to calculate, from distinct portions of the input message, a set of input values for compression by the cryptographic compression function. This process of calculating the input values can contribute significantly to the computational effort required to perform the cryptographic hashing.

EP 3 095 044 A1 describes a message scheduling optimisation commonly known as "ASICBoost" in which different input messages are used for $SHA256_0$ but in which the last 32 bits of the HashMerkleRoot in the Bitcoin Block Header, which are used to derive the input message for $SHA256_1$, are kept the same. This is achieved either by identifying multiple instances of HashMerkleRoot that collide in the last 32 bits (commonly known as "Covert ASICBoost") or by incrementing the Version in the Bitcoin Block Header (commonly known as "Overt ASICBoost"). The same message schedule for $SHA256_1$ can then be reused for multiple outputs of $SHA256_0$. Pham et al, "*Double SHA-256 Hardware Architecture With Compact Message Expander for Bitcoin Mining",* IEEE Access, Volume 8, 2020, pp.139634-139646, also describes avoiding certain message scheduler calculations for $SHA256_1$ and $SHA256_2$ where the input values will always have the value 0x00000000. GB 2 611 321 A1 describes ways in which to further reduce the number of message scheduling calculations that are needed by calculating a set of partially-calculated input values once using initial input message portions which will be invariable across plural different input messages and then, for each subsequent input message of the plural different input

messages, calculating a set of fully calculated input values using the pre-calculated set of partially-calculated input values and the variable portions of the subsequent input message.

It is desired to provide further improvements relating to message scheduling for cryptographic hashing.

SUMMARY

According to an aspect of the present invention there is provided a method of performing cryptographic hashing in respect of a group of plural different input messages, the method comprising:

causing a message scheduling engine to:

pre-calculate a message schedule array comprising plural different sets of input values for a cryptographic compression function, wherein each set of input values in the message schedule array corresponds to a respective input message of the group of plural different input messages; and

causing a cryptographic compressor to:

after the message schedule array has been pre-calculated by the message scheduling engine, perform the cryptographic compression function in respect of each input message of the group of plural different input messages, wherein performing the cryptographic compression function in respect of each input message comprises using the set of input values in the message schedule array corresponding to that input message so as to generate a message digest for that input message.

According to another aspect of the present invention there is provided an apparatus for performing cryptographic hashing in respect of a group of plural different input messages, the apparatus comprising:

a message scheduling engine having processing circuitry configured to:

pre-calculate a message schedule array comprising plural different sets of input values for a cryptographic compression function, wherein each set of input values in the message schedule array corresponds to a respective input message of the group of plural different input messages; and

a cryptographic compressor having processing circuitry configured to:

after the message schedule array has been pre-calculated by the message scheduling engine, perform the cryptographic compression function in respect of each input message of the group of plural different input messages, wherein performing the cryptographic compression function in respect of each input message comprises using the set of input values in the message schedule array corresponding to that input message so as to generate a message digest for that input message.

As will be appreciated, the present invention can greatly increase speed and efficiency when performing cryptographic hashing in respect of groups of plural different input messages. In particular, the Applicant has uniquely recognised that a message schedule array which comprises plural sets of input values can be pre-calculated in advance, i.e. asynchronously, for the plural different input messages and then later re-used many times over when

performing cryptographic hashing in respect of the plural different input messages. Surprisingly, the Applicant has recognised that it is worthwhile doing this even though the message schedule array may be very large. For example, the message schedule array can also be re-used for further input messages having the same content as the group of plural different input messages but being derived from different overall block headers. This can significantly reduce the number of calculations which need to be performed synchronously during the cryptographic hashing process. This in turn can significantly increase the processing speed and/or can significantly reduce the processing burden placed on the apparatus used to perform the cryptographic hashing. This is in contrast to the known techniques in which the set of input values for each given input message is typically calculated synchronously by a message scheduler just prior to that input message being hashed by the cryptographic compressor.

In embodiments, the cryptographic hashing may be performed when mining for cryptocurrency, such as (but not limited to) Bitcoin. The present invention is particularly beneficial in this context. However, the techniques disclosed herein may equally be applied in other cryptographic hashing contexts in which a group of plural different input messages are to be hashed.

In embodiments, the cryptographic compression function may be a SHA256 cryptographic compression function. Overall, the cryptographic hashing may comprise performing plural (e.g. three) instances of the cryptographic compression function. For example, the cryptographic hashing may comprise performing a first instance cryptographic compression function (referred to herein as "$SHA256_0$"), a second instance cryptographic compression function (referred

to herein as "$SHA256_1$"), and a third instance cryptographic compression function (referred to herein as "$SHA256_2$"). The message digest from one instance of the cryptographic compression function may be passed to a subsequent instance of the cryptographic compression function (in the case of the first instance ($SHA256_0$) and second instance ($SHA256_1$)) or may be compared to a Target (in the case of the third instance ($SHA256_2$)). An instance of the cryptographic compression function may use a standard initialisation vector as its initialisation vector (in the case of the first instance ($SHA256_0$) and third instance ($SHA256_2$)) or may use a message digest from the previous instance of the cryptographic compression function as its initialisation vector (in the case of the second instance ($SHA256_1$)). The message scheduling for at least one instance of the cryptographic compression function may be performed in the manner of the present invention. For example, the message scheduling for the second instance cryptographic compression function ($SHA256_1$) may be performed in the manner of the present invention. Thus, the cryptographic compression function may be a second instance cryptographic compression function ($SHA256_1$) of three instances of the cryptographic compression function ($SHA256_0$, $SHA256_1$, $SHA256_2$) performed in the cryptographic hashing.

In embodiments, the input messages of the group of plural different input messages may each be derived from a different block header. For example, the input messages of the group of plural different input messages may each be derived from a different Bitcoin Block Header. Each block header may comprise plural fields. For example, each block header may comprise a Version field, a HashPrevBlock field, a HashMerkleRoot field, a Timestamp field, a Target field,

and/or a Nonce field. The block header may be padded, e.g. with a Padding + Length field, for the purposes of the cryptographic hashing. In embodiments, one or more of the plural fields or parts thereof (e.g. a (second or last) part of the HashMerkleRoot field, the Timestamp field, the Target field and/or the Padding + Length field) may have the same value across (each and every one of) the headers corresponding to the input messages of the group of plural different input messages. One or more of the plural fields or parts thereof (e.g. the Version field, the HashPrevBlock field, a (first or initial) part of the HashMerkleRoot field, the Timestamp field and/or the Nonce field) may not have the same value across (each and every one of) the block headers corresponding to the input messages of the group of plural different input messages. For example, one or more of the fields or parts thereof (e.g. the Version field and/or the Nonce field) may be incremented across the block headers corresponding to the input messages of the group of plural different input messages, e.g. in an attempt to find a $SHA256_2$ message digest or "hash" that is lower than or equal to the Target. For another example, one or more of the fields or parts thereof (e.g. the HashPrevBlock field and/or a (first or initial) part of the HashMerkleRoot field) may change across the block headers corresponding to the input messages of the group of plural different input messages, e.g. in order to account for changes in the Bitcoin Network.

In embodiments, each block header may comprise a first portion and a second portion, wherein the input messages of the group of plural different input messages each include the second portion of the block header. Thus, the input messages of the group of plural different input messages may each comprise plural fields or parts thereof. For example, the input messages of the group of

plural different input messages may each comprise a (second or last) part of a HashMerkleRoot field, a Timestamp field, a Target field, a Nonce field, and/or a Padding + Length field. As indicated above, one or more of the plural fields or parts thereof (e.g. a (second or last) part of the HashMerkleRoot field, the Timestamp field, the Target field, and/or the Padding + Length field) may have the same value across (each and every one of) the input messages of the group of plural different input messages. As also indicated above, one or more of the plural fields or parts thereof (e.g. the Timestamp field and/or the Nonce field) may have a different value across (each and every one of) the input messages of the group of plural different input messages. For example, one or more of the plural fields or parts thereof (e.g. the Nonce field) may be incremented across the input messages of the group of plural different input messages.

In embodiments, each set of input values in the message schedule array may be calculated using a different field value of a set of field values that a particular (e.g. incremented) field (e.g. the Nonce field) of the input message can take. For example, where the particular field is an n-bit field having 2^n possible field values, each set of input values in the message schedule array may be calculated using a different field value of a set of the 2^n possible field values that the particular field of the input message can take. The set of field values may be a subset of, or the entire set of, possible values that the particular field of the input message can take. For example, when the set of field values is the entire set of possible values that the particular n-bit field of the input message can take, the message schedule array may comprise 2^n sets of input values, each set of input values corresponding to a different one of 2^n different input messages and

calculated using a respective one of the 2^n field values. The value of n may, for example, be 32. Thus, the message schedule array may comprise some of, or all of, 2^{32} sets of input values corresponding respectively to some of, or all of, 2^{32} different input messages having 2^{32} different (e.g. Nonce) field values. Alternatively, the message schedule array may comprise more than 2^{32} sets of input values corresponding respectively to different input messages having more than 2^{32} different combinations of (e.g. Nonce and/or Timestamp) field values.

In embodiments, each input message may be a 512-bit input message. Each set of input values in the message schedule array may comprise a set of 64 input values. Each input value of each set of input values may be a 32-bit input value. Thus, the message scheduling engine may expand each input message to a set of 64 32-bit input values totalling 2048 bits (64 x 32).

It will be appreciated that, in embodiments, a first portion of the block headers corresponding to the input messages (e.g. the Version field, the HashPrevBlock field and/or a (first or initial) part of the HashMerkleRoot field) may change freely between the block headers without affecting the validity of the message schedule array. For example, the Version field may be incremented without affecting the validity of the message schedule array. Furthermore, the HashPrevBlock field may be updated when there is a new previous block, without affecting the validity of the message schedule array. Furthermore, a (first or initial) part of the HashMerkleRoot field may be changed whilst a (second or last) part of the HashMerkleRoot field may be left unchanged, e.g. by selecting appropriate transactions, without affecting the validity of the message schedule array.

Thus, embodiments may comprise the cryptographic compressor using the message schedule array to perform the cryptographic compression function in respect of each input message of a first group of plural different input messages and then re-using the message schedule array to perform the cryptographic compression function in respect of each input message of a second group of plural different input messages, wherein the first and second groups of plural different input messages are identical but are derived from different first and second groups of block headers (e.g. having a different Version field, a different HashPrevBlock field and/or a different (first or initial) part of the HashMerkleRoot field).

In embodiments, the message scheduling engine may be configured to (and caused to) select appropriate transactions for the headers of the first and/or second group of plural different input messages such that a (first or initial) part of the HashMerkleRoot field of the second group is different from that of the first group whilst a (second or last) part of the HashMerkleRoot field of the second group is the same as that of the first group. For example, a predetermined (second or last) part of the HashMerkleRoot field may be selected, e.g. in advance of and/or non-asynchronously with performing cryptographic hashing in respect of the first and second groups of plural different input messages. The message scheduling engine may, for example, be configured to (and caused to) select appropriate transactions which provide the predetermined (second or last) part of the HashMerkleRoot field to allow the same message schedule array to be used for both the first and second group of plural different input messages. Alternatively, the message scheduling engine may be configured to (and caused

to) store a (second or last) part of the HashMerkleRoot field for a first group of plural different input messages and then to select appropriate transactions which provide that same (second or last) part of the HashMerkleRoot field to allow the same message schedule array to be re-used for a second group of plural different input messages. The selection of appropriate transactions may be implemented in any desired and suitable way, for example the message scheduling engine may implement artificial intelligence which is trained to select the appropriate transactions.

It is also desirable to keep a message schedule array valid for as long as possible to reduce the need to pre-calculate a further message schedule array. Thus, the message scheduling engine may be configured to (and caused to) attempt to increase the amount of time between any changes to one or more of the fields or parts thereof (e.g. the HashMerkleRoot field). The message scheduling engine may, for example, be configured to (and caused to) select appropriate sets of transactions which are more likely to remain relevant for longer and/or for more blocks. Thus, the message scheduling engine may be configured preferentially to select transactions for forming block headers that are used to derive the group of plural different input messages which are more likely to remain relevant for a longer period of time and/or for more blocks when compared with other transactions. The selection may again be implemented by the message scheduling engine in any desired and suitable way, for example the message scheduling engine may implement artificial intelligence which is trained to select the appropriate transactions or may use a heuristic approach to select the appropriate transactions.

It will, however, be appreciated that the fields or parts thereof (e.g. a (second or last) part of the HashMerkleRoot field, the Timestamp field, and/or the Target field) which are the same for the group of plural different input messages for which the message schedule array was pre-calculated may not be the same for other groups of plural different input messages. These other groups of plural different input messages may make use of a different message schedule array which has been pre-calculated in the manner of the present invention.

Thus, in embodiments, the message scheduling engine may be configured to (and caused to), e.g. asynchronously, pre-calculate a further message schedule array for a further (e.g. subsequent) group of plural different input messages to be hashed. For example, the message scheduling engine may be configured to (and caused to) pre-calculate a further message schedule array if there is a change to one or more of the fields or parts thereof (e.g. a (second or last) part of the HashMerkleRoot field, the Timestamp field and/or the Target field) which were the same for the group of plural different input messages for which the (e.g. current) message schedule array was pre-calculated.

The further message schedule array may then be used by the message compressor when hashing the further group of plural different input messages in the manner of the present invention. Thus, the message compressor may be configured to (and caused to), after a further message schedule array has been pre-calculated, perform the cryptographic compression function in respect of each input message of the further group of plural different input messages, wherein performing the cryptographic compression function in respect of each input message of the further group of plural different input messages comprises

using the set of input values in the further message schedule array corresponding to that input message so as to generate a message digest for that input message.

In embodiments, for each different input message to be hashed, the pre-calculation of the set of input values may comprise performing one or more or all of the operations in a (e.g. SHA256) message scheduling calculation. For example, for t rounds of compression, the message scheduling calculation for an input value W[t] of a set of input values W, which is derived from a (e.g. distinct) input message portion M[t] of an input message M, may comprise performing:

$For\ 0 \le t \le 15,\ W[t] = M[t]$

$For\ 16 \le t \le 63,\ W[t] = \sigma_1(W[t-2]) + W[t-7] + \sigma_0(W[t-15]) + W[t-16]$

where:

$\sigma_1(x) = ROTR_{17}(x) \oplus ROTR_{19}(x) \oplus SHR_{10}(x)$

$\sigma_0(x) = ROTR_{7}(x) \oplus ROTR_{18}(x) \oplus SHR_{3}(x)$

$ROTR_i$ *is a bitwise rotate right for i bits*

SHR_i *is a bitwise shift right for i bits; and*

$\oplus$ *is a bitwise exclusive OR*

Thus, for each different input message to be hashed, the pre-calculation of the set of input values may comprise performing one or more or all of: an add operation; a bitwise rotate right (ROTR) operation; a bitwise shift right (SHR) operation; and a bitwise exclusive OR ($\oplus$) operation.

In embodiments, for each different input message to be hashed, the pre-calculation of the set of input values may also comprise adding a round-specific constant to each of the input values of the set of input values. Adding a round-specific constant may be performed by the message scheduling engine. Thus,

for each different input message to be hashed, the pre-calculation of the set of input values may also comprise the message scheduling engine adding a round-specific constant to each of the input values of the set of input values to generate a set of constant-modified input values. It may then be these constant-modified input values which form the message schedule array. Alternatively, adding a round-specific constant to each of the input values of the set may be performed by the message scheduling engine, a message scheduler or the message compressor after the message schedule array has been pre-calculated and/or, e.g. synchronously, prior to performing the cryptographic compression function in respect of the input message in question.

In embodiments, the message scheduling engine may be configured to (and caused to) store the message schedule array (e.g. in electronic storage of or for the apparatus, such as (but not limited to) registers and/or memory). The apparatus may comprise retrieval circuitry (e.g. of the message scheduling engine or a message scheduler) which is configured to (and caused to) retrieve, for a given input message, the corresponding set of input values for the input message in the message schedule array (e.g. from the electronic storage) and to provide that set of input values to the message compressor to be used when performing the cryptographic compression function in respect of that input message. Alternatively, the message compressor itself may comprise retrieval circuitry configured to (and caused to) retrieve, for a given input message, the corresponding set of input values for the input message in the message schedule array (e.g. from the electronic storage) prior to using that set of input values when

performing the cryptographic compression function in respect of that input message.

Embodiments can be implemented in any desired and suitable data processing apparatus, such as any suitably configured computer-based, processor-based, and/or circuitry-based, system. Similarly, the various functions of the apparatus described herein can be carried out in any desired and suitable manner. For example, the functions of the apparatus described herein may be implemented in hardware and/or software as desired. Thus, unless otherwise indicated, the various functional elements described herein may, for example, comprise a suitable processor or processors, controller or controllers, functional unit or units, circuitry, processing logic, microprocessor arrangements, etc., that are operable to perform the various functions, etc., such as appropriately dedicated hardware elements (processing circuitry) and/or programmable hardware elements (processing circuitry) that are configured to (and caused to) operate in the desired manner. In embodiments, the apparatus may comprise one or more integrated circuits, such as one or more ASICs (Application-Specific Integrated Circuits) and/or FPGAs (Field-Programmable Gate Arrays). In embodiments, the apparatus may comprise, and/or may be in communication with, one or more electronic storage devices that store the data (e.g. transaction data, header data, input message data, message schedule array data, round-specific constant data, initialisation vector data, message digest data, etc.) as described herein.

BRIEF DESCRIPTION OF DRAWINGS

By way of example only, embodiments of the present invention will now be described in detail with reference being made to the accompanying drawings in which:

Figure 1 illustrates an apparatus for performing cryptographic hashing according to an embodiment of the present invention;

Figure 2 illustrates a message scheduling operation as performed by the apparatus of Figure 1;

Figure 3 illustrates cryptographic hashing of a Bitcoin Block Header as performed by the apparatus of Figure 1;

Figure 4 illustrates generation of a hashMerkleRoot according to an embodiment of the present invention; and

Figure 5 illustrates cryptographic compression as performed by the apparatus of Figure 1.

DETAILED DESCRIPTION

Figure 1 shows an apparatus 100 comprising processing circuitry for performing cryptographic hashing. The apparatus 100 may, for example, be implemented using one or more ASICs. The apparatus 100 comprises one or more cryptographic hashing modules 102 (only one shown in Figure 1).

In this embodiment, the cryptographic hashing module 102 is configured to perform SHA256 to compress a 512-bit input message 104 based on a 256-bit initialisation vector 106 in order to produce a 256-bit message digest or hash 108. The cryptographic hashing module 102 comprises a message scheduler 110

which is configured to divide the input message 104 into distinct portions and use those distinct portions to derive a set of input values for t rounds of message compression performed by message compressor 112.

In SHA256, the message scheduler 110 divides the input message 104 into 16 x 32-bit words M[0]-M[15] and expands those 16 x 32-bit words to derive a set of 64 x 32-bit input values W[0]-W[63] for the message compressor 112 in accordance with the following:

$$\text{For } 0 \le t \le 15,\ W[t] = M[t]$$

$$\text{For } 16 \le t \le 63,\ W[t] = \sigma_1(W[t-2]) + W[t-7] + \sigma_0(W[t-15]) + W[t-16]$$

where:

$$\sigma_1(x) = ROTR_{17}(x) \oplus ROTR_{19}(x) \oplus SHR_{10}(x)$$

$$\sigma_0(x) = ROTR_{7}(x) \oplus ROTR_{18}(x) \oplus SHR_{3}(x)$$

$ROTR_i$ is a bitwise rotate right for i bits

SHR_i is a bitwise shift right for i bits; and

$\oplus$ is a bitwise exclusive OR

The 8 elements H_A-H_H of an initialisation vector 106 are also stored as working variables A-H in electronic storage 114 as follows:

$$A = H_A;\ B = H_B;\ C = H_C;\ D = H_D;\ E = H_E;\ F = H_F;\ G = H_G;\ H = H_H$$

The message compressor 112 then performs the 64 rounds of a SHA256 message compression function using the set of input values W[0]-W[63] provided by the message scheduler 110, together with the working variables A-H stored in electronic storage 114 and a set of 64 x 32-bit round-specific constants K[0]-K[63] in accordance with the following:

$T_1 = H + \Sigma_1(E) + Ch(E, F, G) + K[t] + W[t]$

$T_2 = \Sigma_0(A) + Maj(A, B, C)$

$H = G;\ G = F;\ F = E$

$E = D + T_1 = D + H + \Sigma_1(E) + Ch(E, F, G) + K[t] + W[t]$

$D = C;\ C = B;\ B = A$

$A = T_1 + T_2 = H + \Sigma_1(E) + Ch(E, F, G) + \Sigma_0(A) + Maj(A, B, C) + K[t] + W[t]$

where,

$Ch(E, F, G) = (E \wedge F) \oplus (\neg E \wedge G)$

$Maj(A, B, C) = (A \wedge B) \oplus (A \wedge C) \oplus (B \wedge C)$

$\Sigma_0(A) = ROTR_2(A) \oplus ROTR_{13}(A) \oplus ROTR_{22}(A)$

$\Sigma_1(E) = ROTR_6(E) \oplus ROTR_{11}(E) \oplus ROTR_{25}(E)$

After the 64 rounds the following calculations are performed:

$H_A = A + H_A$

$H_B = B + H_B$

$H_C = C + H_C$

$H_D = D + H_D$

$H_E = E + H_E$

$H_F = F + H_F$

$H_G = G + H_G$

$H_H = H + H_H$

With the final message digest (big-endian) being given by:

$H = H_A \,||\, H_B \,||\, H_C \,||\, H_D \,||\, H_E \,||\, H_F \,||\, H_G \,||\, H_H$

In SHA256, the standard initialisation vector IV is:

H_A = 0x6a09e667

H_B = 0xbb67ae85

H_C = 0x3c6ef372

H_D = 0xa54ff53a

H_E = 0x510e527f

H_F = 0x9b05688c

H_G = 0xlf83d9ab

H_H = 0x5be0cdl9

and the round-specific constants K[0]-K[63] are:

K[0] = 0x428A2F98	K[1] = 0x71374491	K[2] = 0xB5C0FBCF	K[3] = 0xE9B5DBA5
K[4] = 0x3956C25B	K[5] = 0x59F111F1	K[6] = 0x923F82A4	K[7] = 0xAB1C5ED5
K[8] = 0xD807AA98	K[9] = 0x12835B01	K[10] = 0x243185BE	K[11] = 0x550C7DC3
K[12] = 0x72BE5D74	K[13] = 0x80DEB1FE	K[14] = 0x9BDC06A7	K[15] = 0xC19BF174
K[16] = 0xE49B69C1	K[17] = 0xEFBE4786	K[18] = 0x0FC19DC6	K[19] = 0x240CA1CC
K[20] = 0x2DE92C6F	K[21] = 0x4A7484AA	K[22] = 0x5CB0A9DC	K[23] = 0x76F988DA
K[24] = 0x983E5152	K[25] = 0xA831C66D	K[26] = 0xB00327C8	K[27] = 0xBF597FC7
K[28] = 0xC6E00BF3	K[29] = 0xD5A79147	K[30] = 0x06CA6351	K[31] = 0x14292967
K[32] = 0x27B70A85	K[33] = 0x2E1B2138	K[34] = 0x4D2C6DFC	K[35] = 0x53380D13
K[36] = 0x650A7354	K[37] = 0x766A0ABB	K[38] = 0x81C2C92E	K[39] = 0x92722C85
K[40] = 0xA2BFE8A1	K[41] = 0xA81A664B	K[42] = 0xC24B8B70	K[43] = 0xC76C51A3

K[44] = 0xD192E819	*K[45] = 0xD6990624*	*K[46] = 0xF40E3585*	*K[47] = 0x106AA070*
K[48] = 0x19A4C116	*K[49] = 0x1E376C08*	*K[50] = 0x2748774C*	*K[51] = 0x34B0BCB5*
K[52] = 0x391C0CB3	*K[53] = 0x4ED8AA4A*	*K[54] = 0x5B9CCA4F*	*K[55] = 0x682E6FF3*
K[56] = 0x748F82EE	*K[57] = 0x78A5636F*	*K[58] = 0x84C87814*	*K[59] = 0x8CC70208*
K[60] = 0x90BEFFFA	*K[61] = 0xA4506CEB*	*K[62] = 0xBEF9A3F7*	*K[63] = 0xC67178F2*

Figure 2 shows details of the message scheduling operation, for example as performed by the message scheduler 110. In this embodiment, the message scheduling comprises causing a multiplexor 202 to progressively select, in respect of each input message, the input value W[t] either to be i) M[t] for $0 \leq t \leq 15$; or ii) $\sigma_1(W[t-2]) + W[t-7] + \sigma_0(W[t-15]) + W[t-16]$ for $16 \leq t \leq 63$. The selected input value W[t] is then progressively passed to circuitry 204 for addition with K[t] to produce the constant-modified input value W'[t]. Thus, in this embodiment, the message scheduling operation also involves performing the step of adding the set of round-specific constants K[0]-K[63] respectively to the set of input values W[0]-W[63] to generate a set of constant-modified input values W'[0]-W'[63], although in other embodiments the circuitry 204 could be omitted and the addition of the round-specific constants K[0]-K[63] could instead be performed by the message compressor 112. To enable the calculation of W[t] for $16 \leq t \leq 63$, the input values W[t] are also progressively moved through a set of 16 x 32-bit registers 206. Circuitry 208 is then used to perform the standard message scheduling calculation: $\sigma_1(W[t-2]) + W[t-7] + \sigma_0(W[t-15]) + W[t-16]$. The message

scheduler 110 operates in this conventional manner in respect of $SHA256_0$ and $SHA256_2$.

Referring again to Figure 1, the apparatus 100 further comprises a message scheduling engine (MSE) 116 for calculating a message schedule array (MSA) and electronic storage 118 for storing the MSA. The MSA comprises, for each one of plural different input messages derived from a respective block header, a corresponding set of pre-calculated input values W'[0]-W'[63], which are calculated in the same manner as the message scheduler 110 discussed above. Thus, the MSE 116 also includes the circuitry as shown in Figure 2 in order to calculate the sets of pre-calculated input values W'[0]-W'[63] for each of the input messages. As will be discussed in more detail below, in this embodiment, the MSE 116 only pre-calculates and stores an MSA in respect of $SHA256_1$ for later retrieval by the message scheduler 110.

Thus, in accordance with embodiments of the present invention, depending on which instance of SHA256 is being performed, the message scheduler 110 either operates in a conventional manner by synchronously calculating the constant-modified input values W'[0]-W'[63] or retrieves the asynchronously pre-calculated constant-modified input values W'[0]-W'[63] from the MSA in electronic storage 118. In either case, the message scheduler 110 provides those constant-modified input values W'[0]-W'[63] to the message compressor 112 to perform the message compression function. Using the MSA in respect of $SHA256_1$ can, for example, reduce the need to re-calculate the input values for $SHA256_1$ synchronously multiple times for many different groups of plural different input messages where those input values would remain the same.

Other cryptographic hashing modules 102 may also have access to the MSA in electronic storage 118 for parallel compression of the different input messages.

Figure 3 illustrates further details of cryptographic hashing of a Bitcoin Block Header 302. The Bitcoin Block Header 302 contains: a 32-bit Version field 304 which provides block version information; a 256-bit HashPrevBlock field 306 which provides the message digest of the previous block accepted by the Bitcoin Network as referenced by the current block; a 256-bit HashMerkleRoot field 308 which provides a message digest of the Merkle Root for the unconfirmed transactions that are selected by the MSA engine 116 to include in the new block; a 32-Bit Timestamp field 310 which provides the current timestamp in seconds since 1970-01-01 00:00 UTC; a 32-bit Target field 312 which provides the current Target to be met for the message digest of the new block (i.e. indicative of a particular minimum number of leading zeros that the message digest must have); and a 32-bit Nonce field 314 which is used to add variation to the block header so as to compute different message digests. A 384-bit Padding + Length field 316 is appended to provide two 512-bit input messages 318 and 320 of suitable size for cryptographic hashing using SHA256.

In order to perform the cryptographic hashing, a first input message 318 is provided by the MSE 116 to a first cryptographic hashing module 324 which operates in a conventional manner by synchronously calculating the constant-modified input values W'[0]-W'[63]. The first cryptographic hashing module 324 performs a first instance cryptographic hashing $SHA256_0$ of the first input message 318 using the standard initialisation vector IV 322. $SHA256_0$ produces a first message digest H_0 which is stored in electronic storage 326.

In accordance with the present invention, a second cryptographic hashing module 328 performs the SHA256 message compression function in respect of a second instance cryptographic hashing $SHA256_1$ using the first message digest H_0 as the initialisation vector together with a set of input values retrieved from the MSA, as calculated and stored by the MSE 116 in electronic storage 118.

As discussed above, in advance of performing $SHA256_1$ message compression in respect of plural different input messages 320, the MSE 116 pre-calculates and stores a respective set of input values W'[0]-W'[63] for each one of those plural different input messages 320. In this embodiment, each input message 302 will have the same last 32 bits of the HashMerkleRoot field 308, the same Timestamp field 310, the same Target field 312 and the same Padding + Length field 316, but an incremented Nonce field 314. Thus, the MSA can comprise up to 2^{32} sets of input values, with there potentially being one set of 64 x 32-bit input values for each value that the Nonce can take. Alternatively or additionally, a message scheduling array might be stored in the MSA based on different values for other fields, such as different values of the Timestamp field. In this case, the MSA can comprise fewer or more than 2^{32} sets of input values. Thus, the MSA may be up to 1TB in size, but the amount of data stored may be smaller or larger depending on whether or not input values are included corresponding to each and every Nonce value and/or whether or not input values are included corresponding to other fields values, such as different Timestamp field values. Although this can be a large amount of data, since the input values for the plural input messages 302 are already stored in the electronic storage 118, there is no need to wait for the message scheduler 110 to calculate those

input values synchronously while performing the $SHA256_1$ hashing algorithm, and this can significantly reduce the overall processing time by allowing asynchronous operation.

Furthermore, those input values can be reused, and thus do not need to be recalculated, for other Version fields 304, HashPrevBlock fields 306, and first 224-bits of the HashMerkleRoot field 308. For example, multiple instances of the HashMerkleRoot that have the same last 32 bits may be identified and/or the Version may be incremented. In either case, the last 32 bits of the HashMerkleRoot field 308 will remain unchanged and thus, as long as the Timestamp field 312 and Target field 312 also remain unchanged, the same MSA as previously pre-calculated and stored in the electronic storage 116, can be reused.

Figure 4 shows the process by which the HashMerkleRoot 412 may be derived. In this embodiment sixteen transactions are shown, although a much greater number of transactions could be used. First, a set 402 of unconfirmed transactions tx_A-tx_P are selected by the MSE 116 from the Bitcoin Mempool 120 (see Figure 1). Then, a first set 404 of hash values H_A-H_P are generated by hashing each transaction id in the set 402 of transactions tx_A-tx_P. Next, a second set 406 of hash values H_{AB}-H_{OP} are generated by hashing pairs of hash values in the first set 404 of hash values H_A-H_P. Next, a third set 408 of hash values H_{ABCD}-H_{MNOP} are generated by hashing pairs of hash values in the second set 406 of hash values H_{AB}-H_{OP}. Next, a fourth set 410 of hash values $H_{ABCDEFGH}$-$H_{ILKLMNOP}$ are generated by hashing pairs of hash values in the third set 408 of hash values

H_{ABCD}-H_{MNOP}. Finally, the HashMerkleRoot 412 is generated by hashing the pair of hash values in the fourth set 410 of hash values $H_{ABCDEFGH}$-$H_{ILKLMNOP}$.

As discussed above, the entire HashMerkleRoot 412 could be kept the same for plural different input messages. Alternatively or additionally, various techniques can be used to identify multiple instances of the HashMerkleRoot 412 that have a desired last 32 bits. For example, the MSE 116 could use artificial intelligence trained to select sets of transactions from the Mempool 120 that are more likely to result in a predetermined last 32 bits. Alternatively, the MSE 116 could store the last 32-bits of an initial HashMerkleRoot 412 for a first set of transactions and then use artificial intelligence trained to select other sets of transactions from the Mempool 120 that are more likely to result in the same last 32 bits. Various techniques can also be used to increase the amount of time that a given MSA, as stored in the electronic storage 118, remains valid. For example, an artificial intelligence or heuristic approach could be used to select sets of transactions which are more likely to remain relevant for more blocks by remaining in the Mempool 120 for a greater amount of time.

Referring again to Figure 3, the $SHA256_1$ hashing algorithm produces a second message digest H_1 330. The second message digest H_1 is combined with a Padding + Length 332 to create a third input message 334 of suitable size for cryptographic hashing using SHA256. The third input message 334 is provided to a third cryptographic hashing module 338, which again operates in a conventional manner by synchronously calculating the constant-modified input values W'[0]-W'[63]. The third cryptographic hashing module 338 performs a third instance cryptographic hashing $SHA256_2$ of the third input message 334 using

the standard initialisation vector IV 336. $SHA256_2$ produces a third, and final, message digest H_2 340.

If the third message digest H_2 meets the Target condition, then the Bitcoin Block Header used to create that message digest H_2 can be broadcast to the Bitcoin Network. On the other hand, if the third message digest H_2 does not meet the Target condition, then one or more different Bitcoin Block Headers will need to be considered. In this embodiment, this is achieved by the second cryptographic hashing module 328 using the same first message digest H_0 together with each subsequent set of input values in the MSA (which corresponds to an incremented Nonce value) until either H_2 meets the Target condition or all Nonce values are exhausted. At this point, the Version may be incremented, a new hashPrevBlock may be available and/or a new HashMerkleRoot with the same last 32 bits may be derived, such that the same MSA can be re-used again. It is also possible for the MSE 116 to asynchronously pre-calculate and store plural sets of input values in a new MSA for other groups of plural input messages, e.g. having a different last 32 bits of the HashMerkleRoot field 308, a different Timestamp field 310 and/or a different Target field 312.

Figure 5 shows the cryptographic compression in more detail. As is shown, a first message compressor 502 is used to perform the 64 rounds of compression in $SHA256_0$. The first message compressor 502 receives the set of constant-modified input values $W_0'[t]$ calculated synchronously by a message scheduler 110 and the standard initialisation vector IV 322. Also shown is circuitry 504 for adding the outputs of the first message compressor 502 to the elements of the standard initialisation vector IV 322 to produce the first message digest H_0. Also

shown is a second message compressor 506 for performing the 64 rounds of compression in $SHA256_1$. The second message compressor 506 receives the set of constant-modified input values $W_1'[t]$ retrieved from the MSA for the input message in question from the electronic storage 118 and H_0 as an initialisation vector. Also shown is circuitry 508 for adding the outputs of the second message compressor 506 to the elements of H_0 to produce the second message digest H_1. Also shown is a third message compressor 510 for performing the 64 rounds of compression in $SHA256_2$. The third message compressor 510 receives the set of constant-modified input values $W_2'[t]$ calculated synchronously by a message scheduler 110, which are based on H_1, and the standard initialisation vector IV 336. Also shown is circuitry 512 for adding the outputs of the third message compressor 510 to the elements of the standard initialisation vector IV 336 to produce the third, and final, message digest H_2.

CLAIMS

1. A method of performing cryptographic hashing in respect of a group of plural different input messages, the method comprising:

causing a message scheduling engine to:

pre-calculate a message schedule array comprising plural different sets of input values for a cryptographic compression function, wherein each set of input values in the message schedule array corresponds to a respective input message of the group of plural different input messages; and

causing a cryptographic compressor to:

after the message schedule array has been pre-calculated by the message scheduling engine, perform the cryptographic compression function in respect of each input message of the group of plural different input messages, wherein performing the cryptographic compression function in respect of each input message comprises using the set of input values in the message schedule array corresponding to that input message so as to generate a message digest for that input message.

2. A method as claimed in claim 1, wherein the cryptographic hashing is performed when mining for Bitcoin.

3. A method as claimed in claim 1 or 2, wherein the cryptographic compression function is a SHA256 cryptographic compression function.

4. A method as claimed in claim 1, 2 or 3, wherein the cryptographic compression function is a second instance cryptographic compression function

of three instances of the cryptographic compression function performed in the cryptographic hashing.

5. A method as claimed in any one of the preceding claims, wherein the input messages of the group of plural different input messages are each derived from a different block header.

6. A method as claimed in claim 5, wherein each block header comprises a Version field, a HashPrevBlock field, a HashMerkleRoot field, a Timestamp field, a Target field, and a Nonce field.

7. A method as claimed in claim 5 or 6, wherein each block header comprises a first portion and a second portion, and wherein the input messages of the group of plural different input messages each include the second portion of the block header.

8. A method as claimed in any one of the preceding claims, wherein the input messages of the group of plural different input messages each comprise a part of a HashMerkleRoot field, a Timestamp field, a Target field, a Nonce field and a Padding + Length field.

9. A method as claimed in claim 8, wherein the part of the HashMerkleRoot field, the Timestamp field, the Target field and/or the Padding + Length field have the same value across the input messages of the group of plural different input messages.

10. A method as claimed in claim 8 or 9, wherein the Nonce field has a different value across the input messages of the group of plural different input messages.

11. A method as claimed in claim 8, 9 or 10, wherein the Nonce field is incremented across the input messages of the group of plural different input messages.

12. A method as claimed in any one of claims 8-11, wherein each set of input values in the message schedule array is calculated using a different field value of a set of field values that the Nonce field of the input message can take.

13. A method as claimed in any one of the preceding claims, wherein the cryptographic compressor uses the message schedule array to perform the cryptographic compression function in respect of each input message of a first group of plural different input messages and then re-uses the message schedule array to perform the cryptographic compression function in respect of each input message of a second group of plural different input messages, wherein the first and second groups of plural different input messages are identical but are derived from different first and second groups of block headers.

14. A method as claimed in claim 13, wherein the message scheduling engine selects transactions for the block headers of the second group of plural different input messages such that a first part of a HashMerkleRoot field of the second group is different from that of the first group whilst a second part of the HashMerkleRoot field of the second group is the same as that of the first group.

15. A method as claimed in any one of the preceding claims, wherein the message scheduling engine preferentially selects transactions for forming block headers that are used to derive the group of plural different input messages which are more likely to remain relevant for a longer period of time and/or for more blocks when compared with other transactions.

16. A method as claimed in any one of the preceding claims, wherein the message scheduling engine pre-calculates a further message schedule array for a further group of plural different input messages to be hashed.

17. A method as claimed in claim 16, wherein the message compressor, after the further message schedule array has been pre-calculated, performs the cryptographic compression function in respect of each input message of the further group of plural different input messages, wherein performing the cryptographic compression function in respect of each input message of the further group of plural different input messages comprises using the set of input values in the further message schedule array corresponding to that input message so as to generate a message digest for that input message.

18. A method as claimed in any one of the preceding claims, wherein, for each different input message to be hashed, the pre-calculation of the set of input values comprises performing one or more or all of the operations in a message scheduling calculation.

19. A method as claimed in any one of the preceding claims, wherein, for each different input message to be hashed, the pre-calculation of the set of input values comprises adding a round-specific constant to each of the input values of the set of input values.

20. An apparatus for performing cryptographic hashing in respect of a group of plural different input messages, the apparatus comprising:

a message scheduling engine having processing circuitry configured to:

pre-calculate a message schedule array comprising plural different sets of input values for a cryptographic compression function, wherein

each set of input values in the message schedule array corresponds to a respective input message of the group of plural different input messages; and

a cryptographic compressor having processing circuitry configured to:

after the message schedule array has been pre-calculated by the message scheduling engine, perform the cryptographic compression function in respect of each input message of the group of plural different input messages, wherein performing the cryptographic compression function in respect of each input message comprises using the set of input values in the message schedule array corresponding to that input message so as to generate a message digest for that input message.

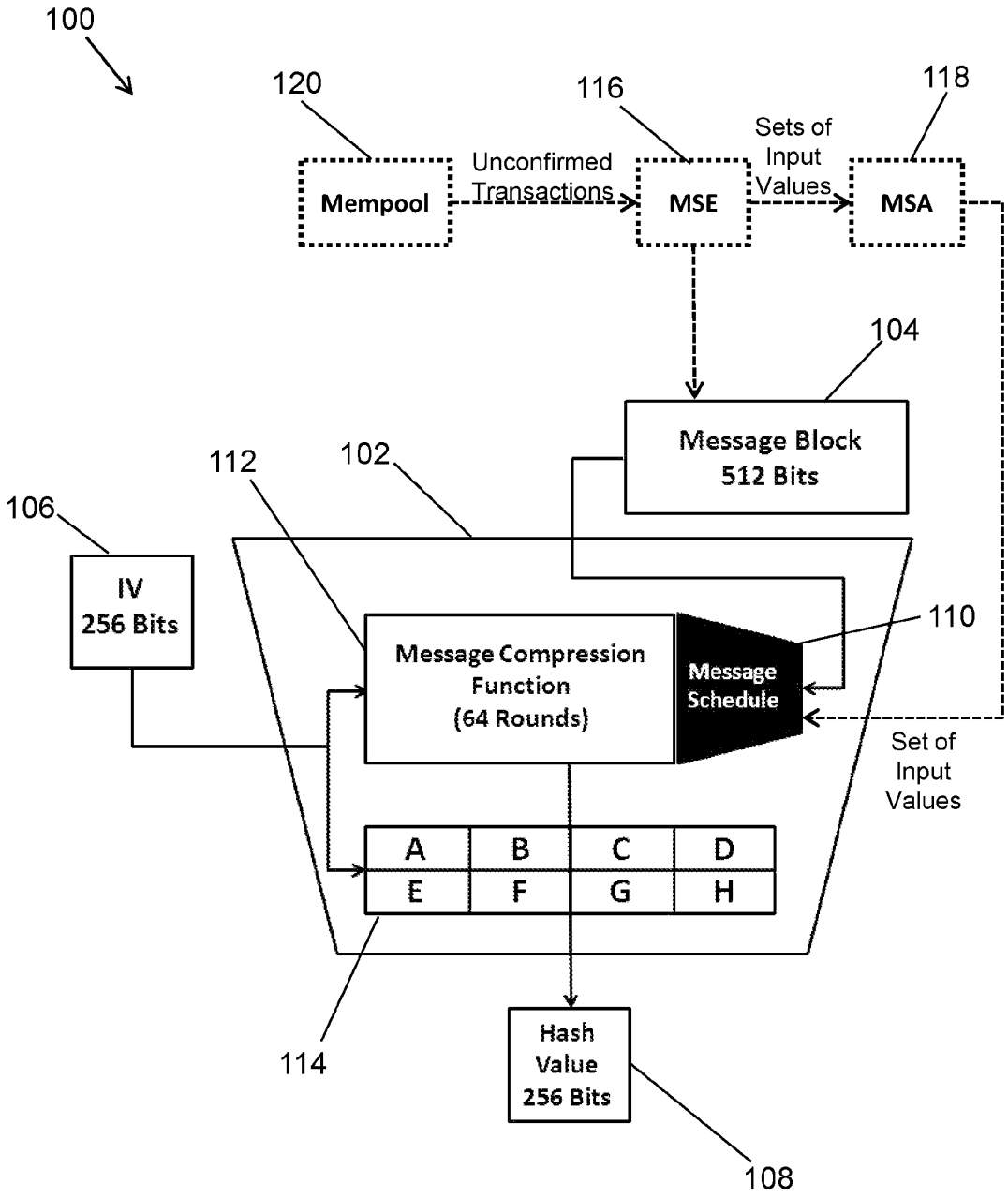
100
120
116
118
Mempool
Unconfirmed Transactions
MSE
Sets of Input Values
MSA
104
Message Block 512 Bits
112
102
106
IV 256 Bits
Message Compression Function (64 Rounds)
Message Schedule
110
Set of Input Values
A
B
C
D
E
F
G
H
114
Hash Value 256 Bits
108

FIG 1

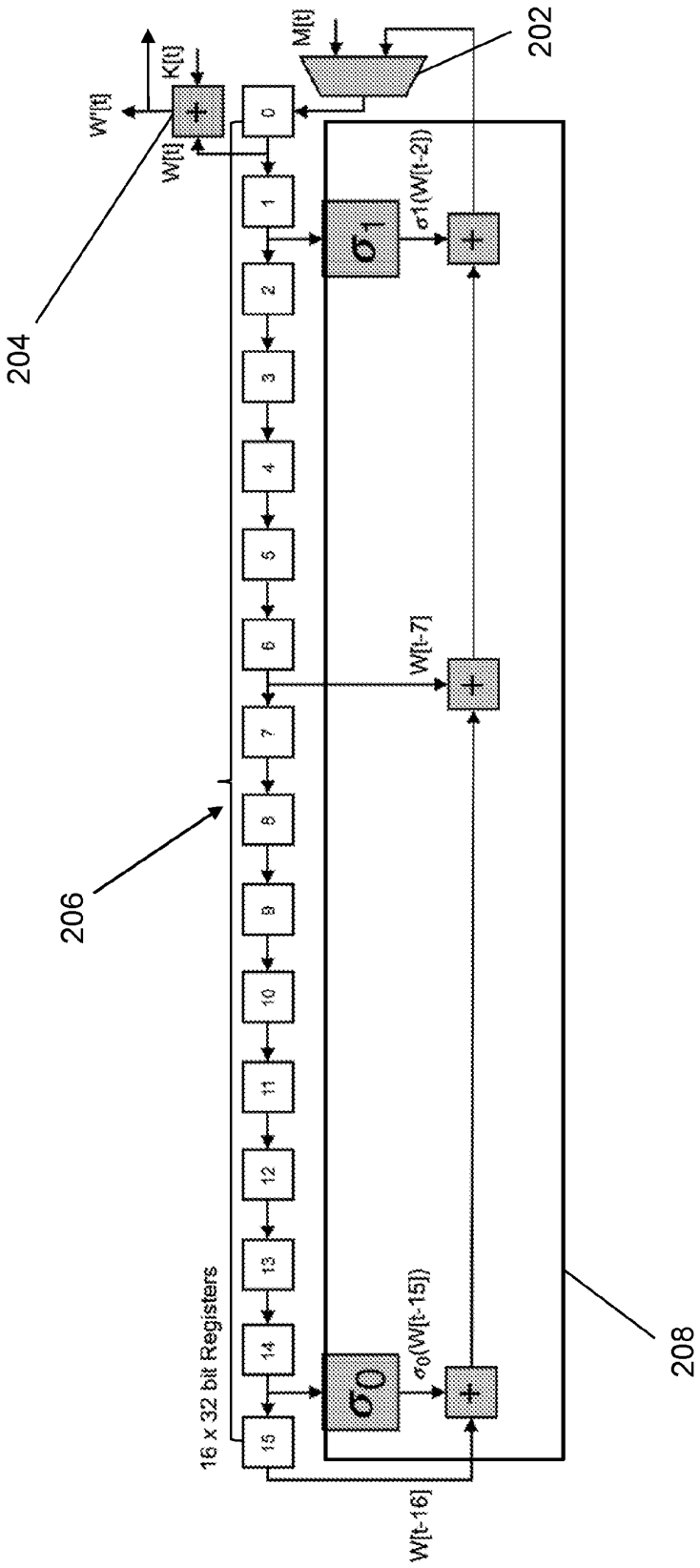
M[t]
K[t]
W'[t]
W[t]
σ1
σ0
σ1(W[t-2])
W[t-7]
σ0(W[t-15])
W[t-16]
16 x 32 bit Registers
202
204
206
208

FIG 2

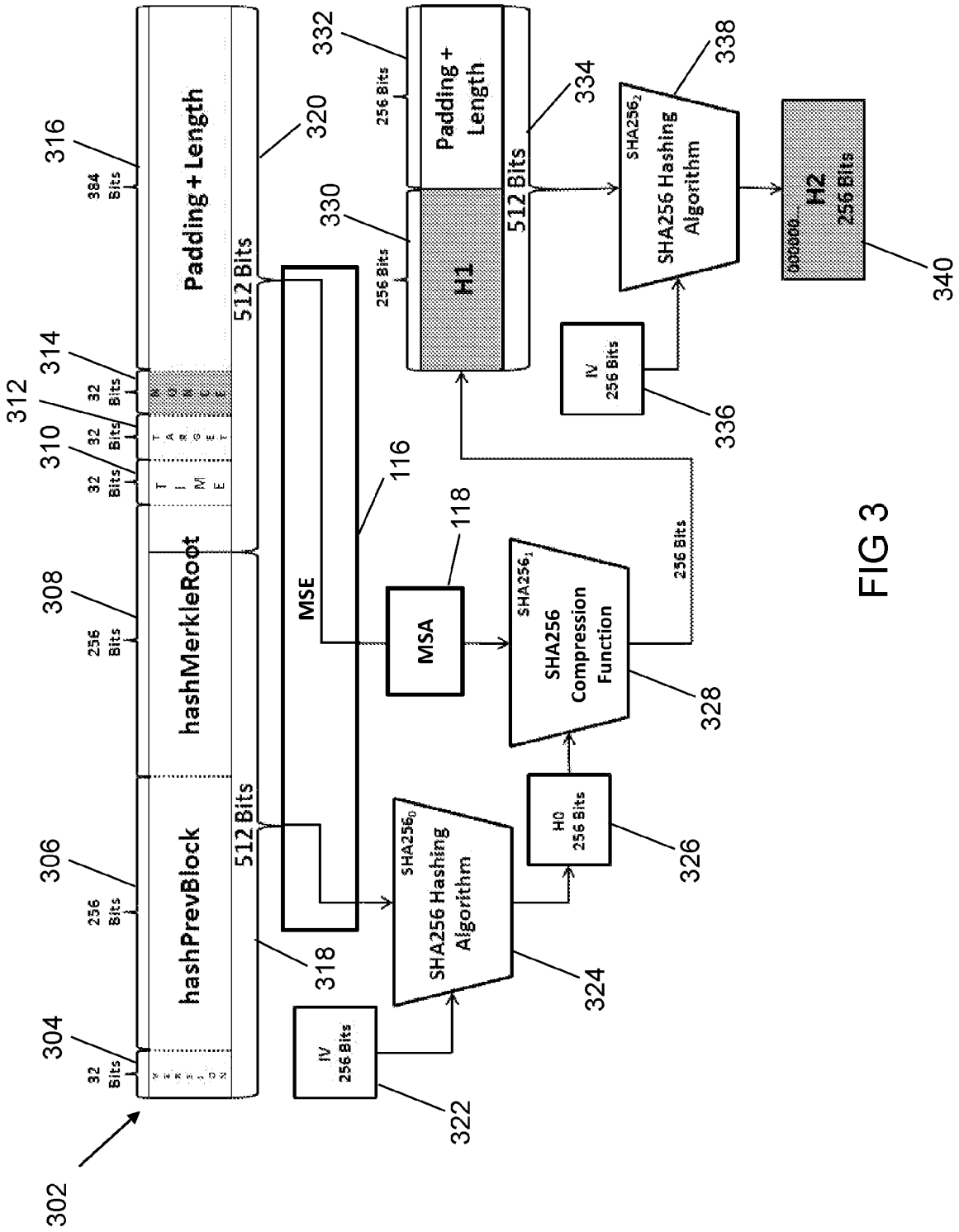
302
304
32 Bits
VERSION
306
256 Bits
hashPrevBlock
308
256 Bits
hashMerkleRoot
310
32 Bits
TIME
312
32 Bits
TARGET
314
32 Bits
NONCE
316
384 Bits
Padding + Length
512 Bits
318
320
512 Bits
322
IV
256 Bits
324
SHA256 Hashing Algorithm
SHA256D
326
H0
256 Bits
328
SHA256 Compression Function
SHA256₁
256 Bits
116
MSE
118
MSA
330
256 Bits
H1
332
256 Bits
Padding + Length
512 Bits
334
336
IV
256 Bits
338
SHA256 Hashing Algorithm
SHA256₂
340
H2
256 Bits

FIG 3

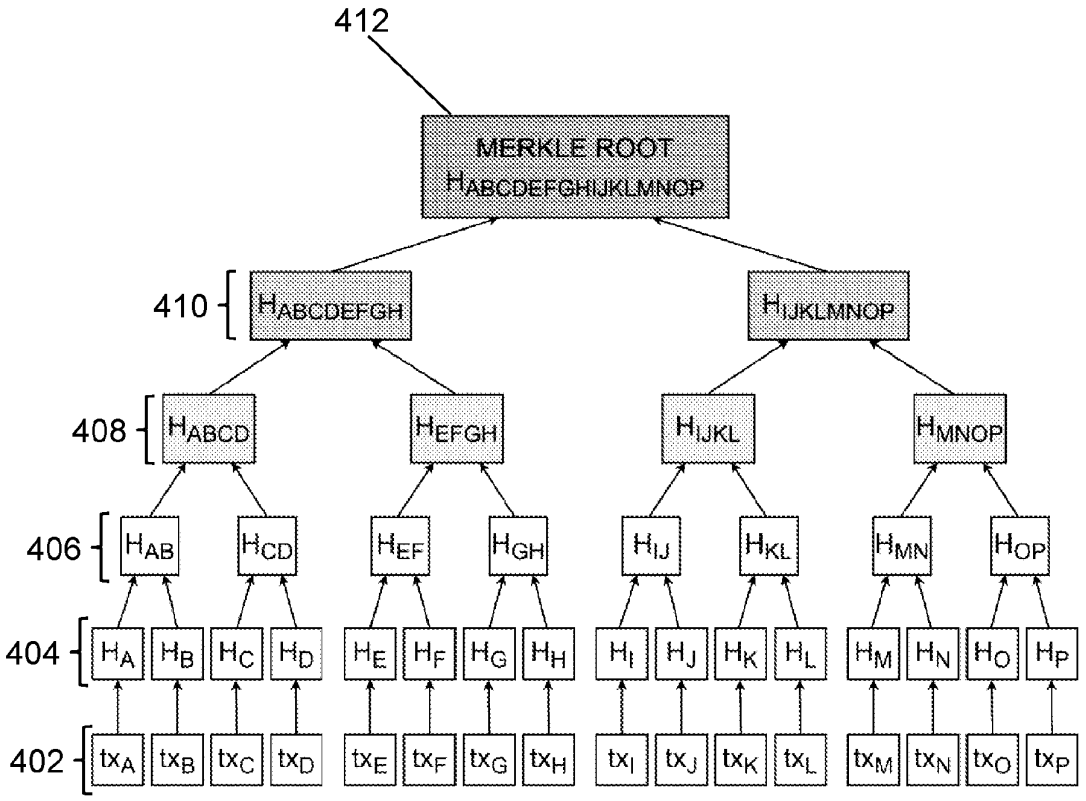
412
MERKLE ROOT
HABCDEFGHIJKLMNOP
410
HABCDEFGH
HIJKLMNOP
408
HABCD
HEFGH
HIJKL
HMNOP
406
HAB
HCD
HEF
HGH
HIJ
HKL
HMN
HOP
404
HA
HB
HC
HD
HE
HF
HG
HH
HI
HJ
HK
HL
HM
HN
HO
HP
402
txA
txB
txC
txD
txE
txF
txG
txH
txI
txJ
txK
txL
txM
txN
txO
txP

FIG 4

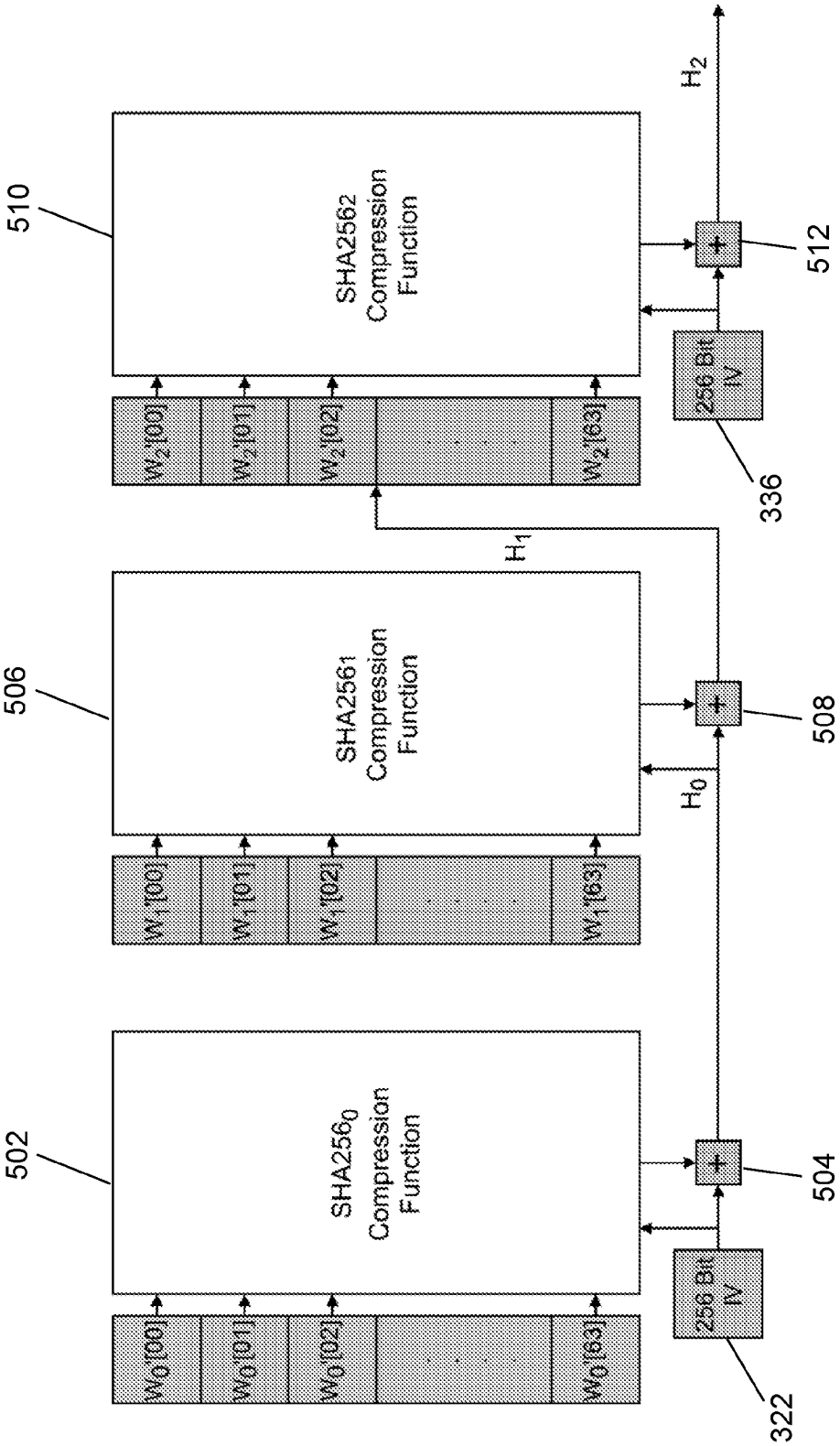
502
SHA256$_0$ Compression Function
$W_0[00]$
$W_0[01]$
$W_0[02]$
$W_0[63]$
322
256 Bit IV
504
506
SHA256$_1$ Compression Function
$W_1[00]$
$W_1[01]$
$W_1[02]$
$W_1[63]$
H_0
508
510
SHA256$_2$ Compression Function
$W_2[00]$
$W_2[01]$
$W_2[02]$
$W_2[63]$
336
256 Bit IV
H_1
512
H_2

FIG 5

INTERNATIONAL SEARCH REPORT

International application No
PCT/GB2024/051785

A. CLASSIFICATION OF SUBJECT MATTER
INV. H04L9/06 H04L9/00
ADD.

According to International Patent Classification (IPC) or to both national classification and IPC

B. FIELDS SEARCHED

Minimum documentation searched (classification system followed by classification symbols)
H04L

Documentation searched other than minimum documentation to the extent that such documents are included in the fields searched

Electronic data base consulted during the international search (name of data base and, where practicable, search terms used)
EPO-Internal

C. DOCUMENTS CONSIDERED TO BE RELEVANT

Category*	Citation of document, with indication, where appropriate, of the relevant passages	Relevant to claim No.
X	US 2017/300877 A1 (MANN OMRI [IL] ET AL) 19 October 2017 (2017-10-19) paragraphs [0040] - [0043]; figure 2 -----	1-20
X	Rubin Jeremy: "The Problem with ASICBOOST", , 11 April 2017 (2017-04-11), XP093198885, Retrieved from the Internet: URL:https://www.mit.edu/~jlrubin/public/pdfs/Asicboost.pdf pages 3-5 ----- -/--	1-20

[x] Further documents are listed in the continuation of Box C.

[x] See patent family annex.

* Special categories of cited documents :

"A" document defining the general state of the art which is not considered to be of particular relevance

"E" earlier application or patent but published on or after the international filing date

"L" document which may throw doubts on priority claim(s) or which is cited to establish the publication date of another citation or other special reason (as specified)

"O" document referring to an oral disclosure, use, exhibition or other means

"P" document published prior to the international filing date but later than the priority date claimed

"T" later document published after the international filing date or priority date and not in conflict with the application but cited to understand the principle or theory underlying the invention

"X" document of particular relevance;; the claimed invention cannot be considered novel or cannot be considered to involve an inventive step when the document is taken alone

"Y" document of particular relevance;; the claimed invention cannot be considered to involve an inventive step when the document is combined with one or more other such documents, such combination being obvious to a person skilled in the art

"&" document member of the same patent family

Date of the actual completion of the international search	Date of mailing of the international search report
22 October 2024	05/11/2024

Name and mailing address of the ISA/	Authorized officer
European Patent Office, P.B. 5818 Patentlaan 2 NL - 2280 HV Rijswijk Tel. (+31-70) 340-2040, Fax: (+31-70) 340-3016	Billet, Olivier

2

Form PCT/ISA/210 (second sheet) (April 2005)

INTERNATIONAL SEARCH REPORT

International application No
PCT/GB2024/051785

C(Continuation). DOCUMENTS CONSIDERED TO BE RELEVANT

Category*	Citation of document, with indication, where appropriate, of the relevant passages	Relevant to claim No.
X	Hanke Timo: "AsicBoost-ASpeedupforBitcoinMining", arXiv.org, arXiv:1604.00575, 31 March 2016 (2016-03-31), XP055451703, Retrieved from the Internet: URL:https://arxiv.org/ftp/arxiv/papers/1604/1604.00575.pdf [retrieved on 2018-02-15] section 3 -----	1-20

INTERNATIONAL SEARCH REPORT

Information on patent family members

International application No
PCT/GB2024/051785

Patent document cited in search report	Publication date	Patent family member(s)	Publication date
US 2017300877 A1	19-10-2017	EP 3198539 A1	02-08-2017
		US 2017300877 A1	19-10-2017
		WO 2016046820 A1	31-03-2016