Uniloc USA, Inc. et al v. NATIONAL INSTRUMENTS CORP. et al Doc. 207 Att. 1

Exhibit "A"

Dockets.Justia.com

UNITED STATES PATENT AND TRADEMARK OFFICE

UNITED STATES DEPARTMENT OF COMMERCE
United States Patent and Trademark Office
Address: COMMISSIONER FOR PATENTS
P.O. Box 1450
Alexandria, Virginia 22313-1450
www.uspto.gov

APPLICATION NO.	FILING DATE	FIRST NAMED INVENTOR	ATTORNEY DOCKET NO.	CONFIRMATION NO.
90/010,831	01/22/2010	5,490,216	2914.001REX0	2214

26111 7590 08/05/2011

STERNE, KESSLER, GOLDSTEIN & FOX P.L.L.C.
1100 NEW YORK AVENUE, N.W.
WASHINGTON, DC 20005

EXAMINER	
ART UNIT	PAPER NUMBER

DATE MAILED: 08/05/2011

Please find below and/or attached an Office communication concerning this application or proceeding.

UNITED STATES PATENT AND TRADEMARK OFFICE

Commissioner for Patents
United States Patent and Trademark Office
P.O. Box 1450
Alexandria, VA 22313-1450
www.uspto.gov

MAILED

AUG 05 2011

CENTRAL REEXAMINATION UNIT

DO NOT USE IN PALM PRINTER

(THIRD PARTY REQUESTER'S CORRESPONDENCE ADDRESS)

Kyle Rinehart
Klarquist Sparkman, LLP
One World Trade Center, Suite 1600
121 S.W. Salmon Street
Portland, OR 97204

EX PARTE REEXAMINATION COMMUNICATION TRANSMITTAL FORM

REEXAMINATION CONTROL NO. *90/010,831*.

PATENT NO. *5,490,216*.

ART UNIT *3992*.

Enclosed is a copy of the latest communication from the United States Patent and Trademark Office in the above identified *ex parte* reexamination proceeding (37 CFR 1.550(f)).

Where this copy is supplied after the reply by requester, 37 CFR 1.535, or the time for filing a reply has passed, no submission on behalf of the *ex parte* reexamination requester will be acknowledged or considered (37 CFR 1.550(g)).

Notice of Intent to Issue Ex Parte Reexamination Certificate	Control No.	Patent Under Reexamination	
	90/010,831	5,490,216	
	Examiner	Art Unit	
	MATTHEW HENEGHAN	3992	

-- The MAILING DATE of this communication appears on the cover sheet with the correspondence address --

1. ☒ Prosecution on the merits is (or remains) closed in this *ex parte* reexamination proceeding. This proceeding is subject to reopening at the initiative of the Office or upon petition. *Cf.* 37 CFR 1.313(a). A Certificate will be issued in view of
   - (a) ☒ Patent owner's communication(s) filed: *18 March 2011*.
   - (b) ☐ Patent owner's late response filed: _____.
   - (c) ☐ Patent owner's failure to file an appropriate response to the Office action mailed: _____.
   - (d) ☐ Patent owner's failure to timely file an Appeal Brief (37 CFR 41.31).
   - (e) ☐ Other: _____.

   Status of *Ex Parte* Reexamination:
   - (f) Change in the Specification: ☐ Yes ☒ No
   - (g) Change in the Drawing(s): ☐ Yes ☒ No
   - (h) Status of the Claim(s):
     - (1) Patent claim(s) confirmed: *1-20*.
     - (2) Patent claim(s) amended (including dependent on amended claim(s)): _____
     - (3) Patent claim(s) canceled: _____.
     - (4) Newly presented claim(s) patentable: _____.
     - (5) Newly presented canceled claims: _____.
     - (6) Patent claim(s) ☐ previously ☐ currently disclaimed: _____
     - (7) Patent claim(s) not subject to reexamination: _____.

2. ☒ Note the attached statement of reasons for patentability and/or confirmation. Any comments considered necessary by patent owner regarding reasons for patentability and/or confirmation must be submitted promptly to avoid processing delays. Such submission(s) should be labeled: "Comments On Statement of Reasons for Patentability and/or Confirmation."

3. ☒ Note attached NOTICE OF REFERENCES CITED (PTO-892).

4. ☐ Note attached LIST OF REFERENCES CITED (PTO/SB/08 or PTO/SB/08 substitute).

5. ☐ The drawing correction request filed on _____ is: ☐ approved ☐ disapproved.

6. ☐ Acknowledgment is made of the priority claim under 35 U.S.C. § 119(a)-(d) or (f).
   a)☐ All b)☐ Some* c)☐ None of the certified copies have
   - ☐ been received.
   - ☐ not been received.
   - ☐ been filed in Application No. _____.
   - ☐ been filed in reexamination Control No. _____.
   - ☐ been received by the International Bureau in PCT Application No. _____.

   * Certified copies not received: _____.

7. ☐ Note attached Examiner's Amendment.

8. ☐ Note attached Interview Summary (PTO-474).

9. ☐ Other: _____.

cc: Requester (if third party requester)

DETAILED ACTION

Reexamination

In response to the previous office action, the Patent Owner filed a Request for Reconsideration on 18 March 2011.

The patent owner is reminded of the continuing responsibility under 37 CFR 1.565(a) to apprise the Office of any litigation activity, or other prior or concurrent proceeding, involving Patent No. 5,490,216 throughout the course of this reexamination proceeding. The third party requester is also reminded of the ability to similarly apprise the Office of any such activity or proceeding throughout the course of this reexamination proceeding. See MPEP §§ 2207, 2282 and 2286.

Extensions of time under 37 CFR 1.136(a) will not be permitted in these proceedings because the provisions of 37 CFR 1.136 apply only to "an applicant" and not to parties in a reexamination proceeding. Additionally, 35 U.S.C. 305 requires that *ex parte* reexamination proceedings "will be conducted with special dispatch" (37 CFR 1.550(a)). Extensions of time in *ex parte* reexamination proceedings are provided for in 37 CFR 1.550(c).

Claims 1-20 have been examined.

Claim Construction

Claim 7 recites "said platform unique ID" in line 5. It is not clear to what this limitation refers. It is being presumed that this is a field produced by the platform unique ID generating means.

Claim 12 lacks a transitional phrase. It is being presumed that the limitations of the claim comprise all those beginning with "said registration system ..." In line 2 and the limitations have been recited in an open-ended manner.

Means Plus Function Limitations

Several means plus function limitations that are being treated under 35 U.S.C. 112, sixth paragraph appear in the claims of the '216 patent. They are support by the specification as follows:

local licensee unique ID generating means (claims 1, 19, 20): a hardware summer (see figure 10 and column 12, lines 62-65), including supporting software, with inputs (see column 12, lines 51-61), may be implemented in software, column 13, lines 42-48).

remote licensee unique ID generating means (claims 1,19, 20): a remote hardware summer (see figure 10 and column 13, lines 2-10), may be implemented in software, column 13, lines 42-48)

mode switching means (claims 1, 19, 20), mode-switching means (claim 17): two hardware gates and a comparator that determine software flow, controlled by a relay, which is driven by software (see column 13, lines 22-40, may be implemented in software, column 13, lines 42-48).

platform unique ID generating means (claim 7): code for creating the platform unique ID (see column 5, lines 57-64), read from a digital code reading device (see column 12, lines 46-50).

registration key generating means (claim 17): a hardware summer (see figure 10 and column 12, lines 62-65), with inputs (see column 12, lines 51-61), may be implemented in software, column 13, lines 42-48).

The term "third party means of operation" in claim 17 is not being treated as a 35 U.S.C. 112, sixth paragraph limitation because it does not have a function associated with the means, other than the broad term "operation."

Allowable Subject Matter

Claims 1-20 are confirmed.

STATEMENT OF REASONS FOR PATENTABILITY AND/OR CONFIRMATION

The following is an examiner's statement of reasons for patentability and/or confirmation of the claims found patentable in this reexamination proceeding:

During reexamination, claims are given the broadest reasonable interpretation consistent with the specification and limitations in the specification are not read into the claims (*In re Yamamoto*, 740 F.2d 1569, 222 USPQ 934 (Fed. Cir. 1984)). Where there exists a final decision by the Court of Appeals for the Federal Circuit regarding the construction of claims, an interpretation is not reasonable where it is inconsistent with that decision. The Patent Owner has persuasively argued that, based on such decisions regarding the '216 patent, Hellman cannot be reasonably construed as teaching to a local licensee unique ID generating means or a remote licensee unique ID generating means.

The licensee unique ID generated by the means recited in each of the claims must be derived from at least piece of information that is specific to the user, such as name, billing information, or product information unique to the instantiation entered by the user. The information cannot be specific to the computer or independently generated by the computer. Hellman's ID has four inputs: a computer-specific key (SK), a number of uses requested (N), a random number generated by the computer (R), and a hash of a code for the type of software package, which is general to all installations of that package (H). Since none of these are user-specific, Hellman's algorithm does not generated the claimed licensee unique ID.

It is also noted that the means itself must be an algorithm that, at least to some extent, must comprise a summation. The means provided by Hellman for combining the fields is DES, an encryption algorithm that does not involve summation; alternatively, Hellman suggests that digital signatures may be used (see Hellman, column 11, lines 42-47). Hellman further stresses that faster cryptographic alternatives could also be used (see column 7, line 67 to column 8, line 12). Given that there were a finite number of such algorithms available at the time of the Patent Owner's invention, it would have been obvious at that time to try any recognized alternative in the implementation of Hellman. As the Federal Circuit has pointed out, the MD5 algorithm (described in RFC 1321, attached to this action) could be such a means.

Of the other art of record, the only one that suggests that use of user-specific information in the computation of fields is Grundy. The Patent Owner has persuasively argued that the summation disclosed by Grundy is used in the context of merely verifying the correctness of information related to the user and is not being used to generate an ID per se. Since the information is not being used for the same purpose, one skilled in the art therefore would not use the algorithm of Grundy as part of the generation of the claimed licensee unique ID.

Any comments considered necessary by PATENT OWNER regarding the above statement must be submitted promptly to avoid processing delays. Such submission by the patent owner should be labeled: "Comments on Statement of Reasons for Patentability and/or Confirmation" and will be placed in the reexamination file.

Declarations

The declaration under 37 CFR 1.132 filed 18 March 2011 by Dr. Udo Pooch is sufficient to overcome the rejection of claims 1-20 based upon Dr. Pooch's argument that it would be improper to combine the references in the manner of the claimed invention.

The declaration under 37 CFR 1.132 filed 18 March 2011 by Dr. William R. Rosenblatt, in which secondary considerations for non-obviousness have been asserted, has been considered. However, we need not reach the issues raised in that declaration because the other evidence discussed above sufficiently supports a finding of non-obviousness.

Response to Arguments

Applicant's arguments, see Remarks, filed 18 March 2011, with respect to the rejections under 35 U.S.C. 102 and 103 have been fully considered and are persuasive. The rejections of the claims have been withdrawn.

Conclusion

All correspondence relating to this *ex parte* reexamination proceeding should be directed:
By Mail to: Mail Stop *Ex Parte* Reexam
Central Reexamination Unit
Commissioner for Patents
United States Patent & Trademark Office
P.O. Box 1450
Alexandria, VA 22313-1450

By FAX to: (571) 273-9900
Central Reexamination Unit

By hand: Customer Service Window
Randolph Building
401 Dulany Street
Alexandria, VA 22314

Registered users of EFS-Web may alternatively submit such correspondence via the electronic filing system EFS-Web, at https://sportal.uspto.gov/authenticate/authenticateuserlocalepf.html. EFS-Web offers the benefit of quick submission to the particular area of the Office that needs to act on the correspondence. Also, EFS-Web submissions are "soft scanned" (i.e., electronically uploaded) directly into the official file for the reexamination proceeding, which offers parties the opportunity to review the content of their submissions after the "soft scanning" process is complete.

Any inquiry concerning this communication should be directed to Examiner Matthew Heneghan at telephone number (571)272-3834.

/Matthew Heneghan/

Primary Examiner, USPTO AU 3992

Conferees:

/EBK/

JESSICA HARRISON
SUPERVISORY PATENT EXAMINER

Notice of References Cited	Application/Control No. 90/010,831	Applicant(s)/Patent Under Reexamination 5,490,216	
	Examiner MATTHEW HENEGHAN	Art Unit 3992	Page 1 of 1

U.S. PATENT DOCUMENTS

*		Document Number Country Code-Number-Kind Code	Date MM-YYYY	Name	Classification
	A	US-			
	B	US-			
	C	US-			
	D	US-			
	E	US-			
	F	US-			
	G	US-			
	H	US-			
	I	US-			
	J	US-			
	K	US-			
	L	US-			
	M	US-			

FOREIGN PATENT DOCUMENTS

*		Document Number Country Code-Number-Kind Code	Date MM-YYYY	Country	Name	Classification
	N					
	O					
	P					
	Q					
	R					
	S					
	T					

NON-PATENT DOCUMENTS

*		Include as applicable: Author, Title Date, Publisher, Edition or Volume, Pertinent Pages)
	U	Rivest, RFC 1321, "The MD% Message-Digest Algorithm," April 1992.
	V	
	W	
	X	

*A copy of this reference is not being furnished with this Office action. (See MPEP § 707.05(a).)
Dates in MM-YYYY format are publication dates. Classifications may be US or foreign.

Network Working Group
Request for Comments: 1321

R. Rivest
MIT Laboratory for Computer Science
and RSA Data Security, Inc.
April 1992

The MD5 Message-Digest Algorithm

Status of this Memo

This memo provides information for the Internet community. It does not specify an Internet standard. Distribution of this memo is unlimited.

Acknowlegements

We would like to thank Don Coppersmith, Burt Kaliski, Ralph Merkle, David Chaum, and Noam Nisan for numerous helpful comments and suggestions.

Table of Contents

1. Executive Summary

This document describes the MD5 message-digest algorithm. The algorithm takes as input a message of arbitrary length and produces as output a 128-bit "fingerprint" or "message digest" of the input. It is conjectured that it is computationally infeasible to produce two messages having the same message digest, or to produce any message having a given prespecified target message digest. The MD5 algorithm is intended for digital signature applications, where a large file must be "compressed" in a secure manner before being encrypted with a private (secret) key under a public-key cryptosystem such as RSA.

The MD5 algorithm is designed to be quite fast on 32-bit machines. In addition, the MD5 algorithm does not require any large substitution tables; the algorithm can be coded quite compactly.

The MD5 algorithm is an extension of the MD4 message-digest algorithm 1,2]. MD5 is slightly slower than MD4, but is more "conservative" in design. MD5 was designed because it was felt that MD4 was perhaps being adopted for use more quickly than justified by the existing critical review; because MD4 was designed to be exceptionally fast, it is "at the edge" in terms of risking successful cryptanalytic attack. MD5 backs off a bit, giving up a little in speed for a much greater likelihood of ultimate security. It incorporates some suggestions made by various reviewers, and contains additional optimizations. The MD5 algorithm is being placed in the public domain for review and possible adoption as a standard.

For OSI-based applications, MD5's object identifier is

```
md5 OBJECT IDENTIFIER ::=
  iso(1) member-body(2) US(840) rsadsi(113549) digestAlgorithm(2) 5}
```

In the X.509 type AlgorithmIdentifier [3], the parameters for MD5 should have type NULL.

2. Terminology and Notation

In this document a "word" is a 32-bit quantity and a "byte" is an eight-bit quantity. A sequence of bits can be interpreted in a natural manner as a sequence of bytes, where each consecutive group of eight bits is interpreted as a byte with the high-order (most significant) bit of each byte listed first. Similarly, a sequence of bytes can be interpreted as a sequence of 32-bit words, where each consecutive group of four bytes is interpreted as a word.with the low-order (least significant) byte given first.

Let x_i denote "x sub i". If the subscript is an expression, we surround it in braces, as in x_{i+1}. Similarly, we use ^ for superscripts (exponentiation), so that x^i denotes x to the i-th power.

Let the symbol "+" denote addition of words (i.e., modulo-2^32 addition). Let X <<< s denote the 32-bit value obtained by circularly shifting (rotating) X left by s bit positions. Let not(X) denote the bit-wise complement of X, and let X v Y denote the bit-wise OR of X and Y. Let X xor Y denote the bit-wise XOR of X and Y, and let XY denote the bit-wise AND of X and Y.

3. MD5 Algorithm Description

We begin by supposing that we have a b-bit message as input, and that we wish to find its message digest. Here b is an arbitrary nonnegative integer; b may be zero, it need not be a multiple of eight, and it may be arbitrarily large. We imagine the bits of the message written down as follows:

m_0 m_1 ... m_{b-1}

The following five steps are performed to compute the message digest of the message.

3.1 Step 1. Append Padding Bits

The message is "padded" (extended) so that its length (in bits) is congruent to 448, modulo 512. That is, the message is extended so that it is just 64 bits shy of being a multiple of 512 bits long. Padding is always performed, even if the length of the message is already congruent to 448, modulo 512.

Padding is performed as follows: a single "1" bit is appended to the message, and then "0" bits are appended so that the length in bits of the padded message becomes congruent to 448, modulo 512. In all, at least one bit and at most 512 bits are appended.

3.2 Step 2. Append Length

A 64-bit representation of b (the length of the message before the padding bits were added) is appended to the result of the previous step. In the unlikely event that b is greater than 2^64, then only the low-order 64 bits of b are used. (These bits are appended as two 32-bit words and appended low-order word first in accordance with the previous conventions.)

At this point the resulting message (after padding with bits and with b) has a length that is an exact multiple of 512 bits. Equivalently, this message has a length that is an exact multiple of 16 (32-bit) words. Let M[0 ... N-1] denote the words of the resulting message, where N is a multiple of 16.

3.3 Step 3. Initialize MD Buffer

A four-word buffer (A,B,C,D) is used to compute the message digest. Here each of A, B, C, D is a 32-bit register. These registers are initialized to the following values in hexadecimal, low-order bytes first):

```
word A: 01 23 45 67
word B: 89 ab cd ef
word C: fe dc ba 98
word D: 76 54 32 10
```

3.4 Step 4. Process Message in 16-Word Blocks

We first define four auxiliary functions that each take as input three 32-bit words and produce as output one 32-bit word.

```
F(X,Y,Z) = XY v not(X) Z
G(X,Y,Z) = XZ v Y not(Z)
H(X,Y,Z) = X xor Y xor Z
I(X,Y,Z) = Y xor (X v not(Z))
```

In each bit position F acts as a conditional: if X then Y else Z. The function F could have been defined using + instead of v since XY and not(X)Z will never have 1's in the same bit position.) It is interesting to note that if the bits of X, Y, and Z are independent and unbiased, the each bit of F(X,Y,Z) will be independent and unbiased.

The functions G, H, and I are similar to the function F, in that they act in "bitwise parallel" to produce their output from the bits of X, Y, and Z, in such a manner that if the corresponding bits of X, Y, and Z are independent and unbiased, then each bit of G(X,Y,Z), H(X,Y,Z), and I(X,Y,Z) will be independent and unbiased. Note that the function H is the bit-wise "xor" or "parity" function of its inputs.

This step uses a 64-element table T[1 ... 64] constructed from the sine function. Let T[i] denote the i-th element of the table, which is equal to the integer part of 4294967296 times abs(sin(i)), where i is in radians. The elements of the table are given in the appendix.

Do the following:

```
/* Process each 16-word block. */
For i = 0 to N/16-1 do

  /* Copy block i into X. */
  For j = 0 to 15 do
    Set X[j] to M[i*16+j].
  end /* of loop on j */

  /* Save A as AA, B as BB, C as CC, and D as DD. */
  AA = A
  BB = B
```

```
    CC = C
    DD = D

    /* Round 1. */
    /* Let [abcd k s i] denote the operation
         a = b + ((a + F(b,c,d) + X[k] + T[i]) <<< s). */
    /* Do the following 16 operations. */
    [ABCD  0  7  1]  [DABC  1 12  2]  [CDAB  2 17  3]  [BCDA  3 22  4]
    [ABCD  4  7  5]  [DABC  5 12  6]  [CDAB  6 17  7]  [BCDA  7 22  8]
    [ABCD  8  7  9]  [DABC  9 12 10]  [CDAB 10 17 11]  [BCDA 11 22 12]
    [ABCD 12  7 13]  [DABC 13 12 14]  [CDAB 14 17 15]  [BCDA 15 22 16]

    /* Round 2. */
    /* Let [abcd k s i] denote the operation
         a = b + ((a + G(b,c,d) + X[k] + T[i]) <<< s). */
    /* Do the following 16 operations. */
    [ABCD  1  5 17]  [DABC  6  9 18]  [CDAB 11 14 19]  [BCDA  0 20 20]
    [ABCD  5  5 21]  [DABC 10  9 22]  [CDAB 15 14 23]  [BCDA  4 20 24]
    [ABCD  9  5 25]  [DABC 14  9 26]  [CDAB  3 14 27]  [BCDA  8 20 28]
    [ABCD 13  5 29]  [DABC  2  9 30]  [CDAB  7 14 31]  [BCDA 12 20 32]

    /* Round 3. */
    /* Let [abcd k s t] denote the operation
         a = b + ((a + H(b,c,d) + X[k] + T[i]) <<< s). */
    /* Do the following 16 operations. */
    [ABCD  5  4 33]  [DABC  8 11 34]  [CDAB 11 16 35]  [BCDA 14 23 36]
    [ABCD  1  4 37]  [DABC  4 11 38]  [CDAB  7 16 39]  [BCDA 10 23 40]
    [ABCD 13  4 41]  [DABC  0 11 42]  [CDAB  3 16 43]  [BCDA  6 23 44]
    [ABCD  9  4 45]  [DABC 12 11 46]  [CDAB 15 16 47]  [BCDA  2 23 48]

    /* Round 4. */
    /* Let [abcd k s t] denote the operation
         a = b + ((a + I(b,c,d) + X[k] + T[i]) <<< s). */
    /* Do the following 16 operations. */
    [ABCD  0  6 49]  [DABC  7 10 50]  [CDAB 14 15 51]  [BCDA  5 21 52]
    [ABCD 12  6 53]  [DABC  3 10 54]  [CDAB 10 15 55]  [BCDA  1 21 56]
    [ABCD  8  6 57]  [DABC 15 10 58]  [CDAB  6 15 59]  [BCDA 13 21 60]
    [ABCD  4  6 61]  [DABC 11 10 62]  [CDAB  2 15 63]  [BCDA  9 21 64]

    /* Then perform the following additions. (That is increment each
       of the four registers by the value it had before this block
       was started.) */
    A = A + AA
    B = B + BB
    C = C + CC
    D = D + DD

  end /* of loop on i */
```

3.5 Step 5. Output

The message digest produced as output is A, B, C, D. That is, we begin with the low-order byte of A, and end with the high-order byte of D.

This completes the description of MD5. A reference implementation in C is given in the appendix.

4. Summary

The MD5 message-digest algorithm is simple to implement, and provides a "fingerprint" or message digest of a message of arbitrary length. It is conjectured that the difficulty of coming up with two messages having the same message digest is on the order of 2^64 operations, and that the difficulty of coming up with any message having a given message digest is on the order of 2^128 operations. The MD5 algorithm has been carefully scrutinized for weaknesses. It is, however, a relatively new algorithm and further security analysis is of course justified, as is the case with any new proposal of this sort.

5. Differences Between MD4 and MD5

The following are the differences between MD4 and MD5:

1. A fourth round has been added.

2. Each step now has a unique additive constant.

3. The function g in round 2 was changed from (XY v XZ v YZ) to (XZ v Y not(Z)) to make g less symmetric.

4. Each step now adds in the result of the previous step. This promotes a faster "avalanche effect".

5. The order in which input words are accessed in rounds 2 and 3 is changed, to make these patterns less like each other.

6. The shift amounts in each round have been approximately optimized, to yield a faster "avalanche effect." The shifts in different rounds are distinct.

References

[1] Rivest, R., "The MD4 Message Digest Algorithm", RFC 1320, MIT and RSA Data Security, Inc., April 1992.

[2] Rivest, R., "The MD4 message digest algorithm", in A.J. Menezes and S.A. Vanstone, editors, Advances in Cryptology - CRYPTO '90 Proceedings, pages 303-311, Springer-Verlag, 1991.

[3] CCITT Recommendation X.509 (1988), "The Directory - Authentication Framework."

APPENDIX A - Reference Implementation

This appendix contains the following files taken from RSAREF: A Cryptographic Toolkit for Privacy-Enhanced Mail:

global.h -- global header file

md5.h -- header file for MD5

md5c.c -- source code for MD5

For more information on RSAREF, send email to <rsaref@rsa.com>.

The appendix also includes the following file:

mddriver.c -- test driver for MD2, MD4 and MD5

The driver compiles for MD5 by default but can compile for MD2 or MD4 if the symbol MD is defined on the C compiler command line as 2 or 4.

The implementation is portable and should work on many different plaforms. However, it is not difficult to optimize the implementation on particular platforms, an exercise left to the reader. For example, on "little-endian" platforms where the lowest-addressed byte in a 32-bit word is the least significant and there are no alignment restrictions, the call to Decode in MD5Transform can be replaced with a typecast.

A.1 global.h

```
/* GLOBAL.H - RSAREF types and constants
 */

/* PROTOTYPES should be set to one if and only if the compiler supports
  function argument prototyping.
The following makes PROTOTYPES default to 0 if it has not already
```

```
  been defined with C compiler flags.
 */
#ifndef PROTOTYPES
#define PROTOTYPES 0
#endif

/* POINTER defines a generic pointer type */
typedef unsigned char *POINTER;

/* UINT2 defines a two byte word */
typedef unsigned short int UINT2;

/* UINT4 defines a four byte word */
typedef unsigned long int UINT4;

/* PROTO_LIST is defined depending on how PROTOTYPES is defined above.
If using PROTOTYPES, then PROTO_LIST returns the list, otherwise it
  returns an empty list.
 */
#if PROTOTYPES
#define PROTO_LIST(list) list
#else
#define PROTO_LIST(list) ()
#endif
```

A.2 md5.h

```
/* MD5.H - header file for MD5C.C
 */

/* Copyright (C) 1991-2, RSA Data Security, Inc. Created 1991. All
rights reserved.

License to copy and use this software is granted provided that it
is identified as the "RSA Data Security, Inc. MD5 Message-Digest
Algorithm" in all material mentioning or referencing this software
or this function.

License is also granted to make and use derivative works provided
that such works are identified as "derived from the RSA Data
Security, Inc. MD5 Message-Digest Algorithm" in all material
mentioning or referencing the derived work.

RSA Data Security, Inc. makes no representations concerning either
the merchantability of this software or the suitability of this
software for any particular purpose. It is provided "as is"
without express or implied warranty of any kind.
```

```
These notices must be retained in any copies of any part of this
documentation and/or software.
 */

/* MD5 context. */
typedef struct {
  UINT4 state[4];                                   /* state (ABCD) */
  UINT4 count[2];        /* number of bits, modulo 2^64 (lsb first) */
  unsigned char buffer[64];                         /* input buffer */
} MD5_CTX;

void MD5Init PROTO_LIST ((MD5_CTX *));
void MD5Update PROTO_LIST
  ((MD5_CTX *, unsigned char *, unsigned int));
void MD5Final PROTO_LIST ((unsigned char [16], MD5_CTX *));
```

A.3 md5c.c

```
/* MD5C.C - RSA Data Security, Inc., MD5 message-digest algorithm
 */

/* Copyright (C) 1991-2, RSA Data Security, Inc. Created 1991. All
rights reserved.

License to copy and use this software is granted provided that it
is identified as the "RSA Data Security, Inc. MD5 Message-Digest
Algorithm" in all material mentioning or referencing this software
or this function.

License is also granted to make and use derivative works provided
that such works are identified as "derived from the RSA Data
Security, Inc. MD5 Message-Digest Algorithm" in all material
mentioning or referencing the derived work.

RSA Data Security, Inc. makes no representations concerning either
the merchantability of this software or the suitability of this
software for any particular purpose. It is provided "as is"
without express or implied warranty of any kind.

These notices must be retained in any copies of any part of this
documentation and/or software.
 */

#include "global.h"
#include "md5.h"

/* Constants for MD5Transform routine.
 */
```

```
#define S11 7
#define S12 12
#define S13 17
#define S14 22
#define S21 5
#define S22 9
#define S23 14
#define S24 20
#define S31 4
#define S32 11
#define S33 16
#define S34 23
#define S41 6
#define S42 10
#define S43 15
#define S44 21

static void MD5Transform PROTO_LIST ((UINT4 [4], unsigned char [64]));
static void Encode PROTO_LIST
  ((unsigned char *, UINT4 *, unsigned int));
static void Decode PROTO_LIST
  ((UINT4 *, unsigned char *, unsigned int));
static void MD5_memcpy PROTO_LIST ((POINTER, POINTER, unsigned int));
static void MD5_memset PROTO_LIST ((POINTER, int, unsigned int));

static unsigned char PADDING[64] = {
  0x80, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
  0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
  0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0
};

/* F, G, H and I are basic MD5 functions.
 */
#define F(x, y, z) (((x) & (y)) | ((~x) & (z)))
#define G(x, y, z) (((x) & (z)) | ((y) & (~z)))
#define H(x, y, z) ((x) ^ (y) ^ (z))
#define I(x, y, z) ((y) ^ ((x) | (~z)))

/* ROTATE_LEFT rotates x left n bits.
 */
#define ROTATE_LEFT(x, n) (((x) << (n)) | ((x) >> (32-(n))))

/* FF, GG, HH, and II transformations for rounds 1, 2, 3, and 4.
Rotation is separate from addition to prevent recomputation.
 */
#define FF(a, b, c, d, x, s, ac) { \
 (a) += F ((b), (c), (d)) + (x) + (UINT4)(ac); \
 (a) = ROTATE_LEFT ((a), (s)); \
```

```
 (a) += (b); \
  }
#define GG(a, b, c, d, x, s, ac) { \
 (a) += G ((b), (c), (d)) + (x) + (UINT4)(ac); \
 (a) = ROTATE_LEFT ((a), (s)); \
 (a) += (b); \
  }
#define HH(a, b, c, d, x, s, ac) { \
 (a) += H ((b), (c), (d)) + (x) + (UINT4)(ac); \
 (a) = ROTATE_LEFT ((a), (s)); \
 (a) += (b); \
  }
#define II(a, b, c, d, x, s, ac) { \
 (a) += I ((b), (c), (d)) + (x) + (UINT4)(ac); \
 (a) = ROTATE_LEFT ((a), (s)); \
 (a) += (b); \
  }

/* MD5 initialization. Begins an MD5 operation, writing a new context.
 */
void MD5Init (context)
MD5_CTX *context;                                        /* context */
{
  context->count[0] = context->count[1] = 0;
  /* Load magic initialization constants.
*/
  context->state[0] = 0x67452301;
  context->state[1] = 0xefcdab89;
  context->state[2] = 0x98badcfe;
  context->state[3] = 0x10325476;
}

/* MD5 block update operation. Continues an MD5 message-digest
  operation, processing another message block, and updating the
  context.
 */
void MD5Update (context, input, inputLen)
MD5_CTX *context;                                        /* context */
unsigned char *input;                                /* input block */
unsigned int inputLen;                     /* length of input block */
{
  unsigned int i, index, partLen;

  /* Compute number of bytes mod 64 */
  index = (unsigned int)((context->count[0] >> 3) & 0x3F);

  /* Update number of bits */
  if ((context->count[0] += ((UINT4)inputLen << 3))
```

```
   < ((UINT4)inputLen << 3))
 context->count[1]++;
  context->count[1] += ((UINT4)inputLen >> 29);

  partLen = 64 - index;

  /* Transform as many times as possible.
*/
  if (inputLen >= partLen) {
 MD5_memcpy
   ((POINTER)&context->buffer[index], (POINTER)input, partLen);
 MD5Transform (context->state, context->buffer);

 for (i = partLen; i + 63 < inputLen; i += 64)
   MD5Transform (context->state, &input[i]);

 index = 0;
  }
  else
 i = 0;

  /* Buffer remaining input */
  MD5_memcpy
 ((POINTER)&context->buffer[index], (POINTER)&input[i],
  inputLen-i);
}

/* MD5 finalization. Ends an MD5 message-digest operation, writing the
  the message digest and zeroizing the context.
 */
void MD5Final (digest, context)
unsigned char digest[16];                         /* message digest */
MD5_CTX *context;                                       /* context */
{
  unsigned char bits[8];
  unsigned int index, padLen;

  /* Save number of bits */
  Encode (bits, context->count, 8);

  /* Pad out to 56 mod 64.
*/
  index = (unsigned int)((context->count[0] >> 3) & 0x3f);
  padLen = (index < 56) ? (56 - index) : (120 - index);
  MD5Update (context, PADDING, padLen);

  /* Append length (before padding) */
  MD5Update (context, bits, 8);
```

```
  /* Store state in digest */
  Encode (digest, context->state, 16);

  /* Zeroize sensitive information.
*/
  MD5_memset ((POINTER)context, 0, sizeof (*context));
}

/* MD5 basic transformation. Transforms state based on block.
 */
static void MD5Transform (state, block)
UINT4 state[4];
unsigned char block[64];
{
  UINT4 a = state[0], b = state[1], c = state[2], d = state[3], x[16];

  Decode (x, block, 64);

  /* Round 1 */
  FF (a, b, c, d, x[ 0], S11, 0xd76aa478); /* 1 */
  FF (d, a, b, c, x[ 1], S12, 0xe8c7b756); /* 2 */
  FF (c, d, a, b, x[ 2], S13, 0x242070db); /* 3 */
  FF (b, c, d, a, x[ 3], S14, 0xc1bdceee); /* 4 */
  FF (a, b, c, d, x[ 4], S11, 0xf57c0faf); /* 5 */
  FF (d, a, b, c, x[ 5], S12, 0x4787c62a); /* 6 */
  FF (c, d, a, b, x[ 6], S13, 0xa8304613); /* 7 */
  FF (b, c, d, a, x[ 7], S14, 0xfd469501); /* 8 */
  FF (a, b, c, d, x[ 8], S11, 0x698098d8); /* 9 */
  FF (d, a, b, c, x[ 9], S12, 0x8b44f7af); /* 10 */
  FF (c, d, a, b, x[10], S13, 0xffff5bb1); /* 11 */
  FF (b, c, d, a, x[11], S14, 0x895cd7be); /* 12 */
  FF (a, b, c, d, x[12], S11, 0x6b901122); /* 13 */
  FF (d, a, b, c, x[13], S12, 0xfd987193); /* 14 */
  FF (c, d, a, b, x[14], S13, 0xa679438e); /* 15 */
  FF (b, c, d, a, x[15], S14, 0x49b40821); /* 16 */

 /* Round 2 */
  GG (a, b, c, d, x[ 1], S21, 0xf61e2562); /* 17 */
  GG (d, a, b, c, x[ 6], S22, 0xc040b340); /* 18 */
  GG (c, d, a, b, x[11], S23, 0x265e5a51); /* 19 */
  GG (b, c, d, a, x[ 0], S24, 0xe9b6c7aa); /* 20 */
  GG (a, b, c, d, x[ 5], S21, 0xd62f105d); /* 21 */
  GG (d, a, b, c, x[10], S22,  0x2441453); /* 22 */
  GG (c, d, a, b, x[15], S23, 0xd8a1e681); /* 23 */
  GG (b, c, d, a, x[ 4], S24, 0xe7d3fbc8); /* 24 */
  GG (a, b, c, d, x[ 9], S21, 0x21e1cde6); /* 25 */
  GG (d, a, b, c, x[14], S22, 0xc33707d6); /* 26 */
  GG (c, d, a, b, x[ 3], S23, 0xf4d50d87); /* 27 */
```

```
  GG (b, c, d, a, x[ 8], S24, 0x455a14ed); /* 28 */
  GG (a, b, c, d, x[13], S21, 0xa9e3e905); /* 29 */
  GG (d, a, b, c, x[ 2], S22, 0xfcefa3f8); /* 30 */
  GG (c, d, a, b, x[ 7], S23, 0x676f02d9); /* 31 */
  GG (b, c, d, a, x[12], S24, 0x8d2a4c8a); /* 32 */

  /* Round 3 */
  HH (a, b, c, d, x[ 5], S31, 0xfffa3942); /* 33 */
  HH (d, a, b, c, x[ 8], S32, 0x8771f681); /* 34 */
  HH (c, d, a, b, x[11], S33, 0x6d9d6122); /* 35 */
  HH (b, c, d, a, x[14], S34, 0xfde5380c); /* 36 */
  HH (a, b, c, d, x[ 1], S31, 0xa4beea44); /* 37 */
  HH (d, a, b, c, x[ 4], S32, 0x4bdecfa9); /* 38 */
  HH (c, d, a, b, x[ 7], S33, 0xf6bb4b60); /* 39 */
  HH (b, c, d, a, x[10], S34, 0xbebfbc70); /* 40 */
  HH (a, b, c, d, x[13], S31, 0x289b7ec6); /* 41 */
  HH (d, a, b, c, x[ 0], S32, 0xeaa127fa); /* 42 */
  HH (c, d, a, b, x[ 3], S33, 0xd4ef3085); /* 43 */
  HH (b, c, d, a, x[ 6], S34,  0x4881d05); /* 44 */
  HH (a, b, c, d, x[ 9], S31, 0xd9d4d039); /* 45 */
  HH (d, a, b, c, x[12], S32, 0xe6db99e5); /* 46 */
  HH (c, d, a, b, x[15], S33, 0x1fa27cf8); /* 47 */
  HH (b, c, d, a, x[ 2], S34, 0xc4ac5665); /* 48 */

  /* Round 4 */
  II (a, b, c, d, x[ 0], S41, 0xf4292244); /* 49 */
  II (d, a, b, c, x[ 7], S42, 0x432aff97); /* 50 */
  II (c, d, a, b, x[14], S43, 0xab9423a7); /* 51 */
  II (b, c, d, a, x[ 5], S44, 0xfc93a039); /* 52 */
  II (a, b, c, d, x[12], S41, 0x655b59c3); /* 53 */
  II (d, a, b, c, x[ 3], S42, 0x8f0ccc92); /* 54 */
  II (c, d, a, b, x[10], S43, 0xffeff47d); /* 55 */
  II (b, c, d, a, x[ 1], S44, 0x85845dd1); /* 56 */
  II (a, b, c, d, x[ 8], S41, 0x6fa87e4f); /* 57 */
  II (d, a, b, c, x[15], S42, 0xfe2ce6e0); /* 58 */
  II (c, d, a, b, x[ 6], S43, 0xa3014314); /* 59 */
  II (b, c, d, a, x[13], S44, 0x4e0811a1); /* 60 */
  II (a, b, c, d, x[ 4], S41, 0xf7537e82); /* 61 */
  II (d, a, b, c, x[11], S42, 0xbd3af235); /* 62 */
  II (c, d, a, b, x[ 2], S43, 0x2ad7d2bb); /* 63 */
  II (b, c, d, a, x[ 9], S44, 0xeb86d391); /* 64 */

  state[0] += a;
  state[1] += b;
  state[2] += c;
  state[3] += d;

  /* Zeroize sensitive information.
```

```
*/
  MD5_memset ((POINTER)x, 0, sizeof (x));
}

/* Encodes input (UINT4) into output (unsigned char). Assumes len is
  a multiple of 4.
 */
static void Encode (output, input, len)
unsigned char *output;
UINT4 *input;
unsigned int len;
{
  unsigned int i, j;

  for (i = 0, j = 0; j < len; i++, j += 4) {
 output[j] = (unsigned char)(input[i] & 0xff);
 output[j+1] = (unsigned char)((input[i] >> 8) & 0xff);
 output[j+2] = (unsigned char)((input[i] >> 16) & 0xff);
 output[j+3] = (unsigned char)((input[i] >> 24) & 0xff);
  }
}

/* Decodes input (unsigned char) into output (UINT4). Assumes len is
  a multiple of 4.
 */
static void Decode (output, input, len)
UINT4 *output;
unsigned char *input;
unsigned int len;
{
  unsigned int i, j;

  for (i = 0, j = 0; j < len; i++, j += 4)
 output[i] = ((UINT4)input[j]) | (((UINT4)input[j+1]) << 8) |
   (((UINT4)input[j+2]) << 16) | (((UINT4)input[j+3]) << 24);
}

/* Note: Replace "for loop" with standard memcpy if possible.
 */

static void MD5_memcpy (output, input, len)
POINTER output;
POINTER input;
unsigned int len;
{
  unsigned int i;

  for (i = 0; i < len; i++)
```

```
 output[i] = input[i];
}

/* Note: Replace "for loop" with standard memset if possible.
 */
static void MD5_memset (output, value, len)
POINTER output;
int value;
unsigned int len;
{
  unsigned int i;

  for (i = 0; i < len; i++)
 ((char *)output)[i] = (char)value;
}
```

A.4 mddriver.c

```
/* MDDRIVER.C - test driver for MD2, MD4 and MD5
 */

/* Copyright (C) 1990-2, RSA Data Security, Inc. Created 1990. All
rights reserved.

RSA Data Security, Inc. makes no representations concerning either
the merchantability of this software or the suitability of this
software for any particular purpose. It is provided "as is"
without express or implied warranty of any kind.

These notices must be retained in any copies of any part of this
documentation and/or software.
 */

/* The following makes MD default to MD5 if it has not already been
  defined with C compiler flags.
 */
#ifndef MD
#define MD MD5
#endif

#include <stdio.h>
#include <time.h>
#include <string.h>
#include "global.h"
#if MD == 2
#include "md2.h"
#endif
#if MD == 4
```

```
#include "md4.h"
#endif
#if MD == 5
#include "md5.h"
#endif

/* Length of test block, number of test blocks.
 */
#define TEST_BLOCK_LEN 1000
#define TEST_BLOCK_COUNT 1000

static void MDString PROTO_LIST ((char *));
static void MDTimeTrial PROTO_LIST ((void));
static void MDTestSuite PROTO_LIST ((void));
static void MDFile PROTO_LIST ((char *));
static void MDFilter PROTO_LIST ((void));
static void MDPrint PROTO_LIST ((unsigned char [16]));

#if MD == 2
#define MD_CTX MD2_CTX
#define MDInit MD2Init
#define MDUpdate MD2Update
#define MDFinal MD2Final
#endif
#if MD == 4
#define MD_CTX MD4_CTX
#define MDInit MD4Init
#define MDUpdate MD4Update
#define MDFinal MD4Final
#endif
#if MD == 5
#define MD_CTX MD5_CTX
#define MDInit MD5Init
#define MDUpdate MD5Update
#define MDFinal MD5Final
#endif

/* Main driver.

Arguments (may be any combination):
  -sstring - digests string
  -t       - runs time trial
  -x       - runs test script
  filename - digests file
  (none)   - digests standard input
 */
int main (argc, argv)
int argc;
```

```
char *argv[];
{
  int i;

  if (argc > 1)
 for (i = 1; i < argc; i++)
   if (argv[i][0] == '-' && argv[i][1] == 's')
     MDString (argv[i] + 2);
   else if (strcmp (argv[i], "-t") == 0)
     MDTimeTrial ();
   else if (strcmp (argv[i], "-x") == 0)
     MDTestSuite ();
   else
     MDFile (argv[i]);
  else
 MDFilter ();

  return (0);
}

/* Digests a string and prints the result.
 */
static void MDString (string)
char *string;
{
  MD_CTX context;
  unsigned char digest[16];
  unsigned int len = strlen (string);

  MDInit (&context);
  MDUpdate (&context, string, len);
  MDFinal (digest, &context);

  printf ("MD%d (\"%s\") = ", MD, string);
  MDPrint (digest);
  printf ("\n");
}

/* Measures the time to digest TEST_BLOCK_COUNT TEST_BLOCK_LEN-byte
  blocks.
 */
static void MDTimeTrial ()
{
  MD_CTX context;
  time_t endTime, startTime;
  unsigned char block[TEST_BLOCK_LEN], digest[16];
  unsigned int i;
```

```
  printf
 ("MD%d time trial. Digesting %d %d-byte blocks ...", MD,
  TEST_BLOCK_LEN, TEST_BLOCK_COUNT);

  /* Initialize block */
  for (i = 0; i < TEST_BLOCK_LEN; i++)
 block[i] = (unsigned char)(i & 0xff);

  /* Start timer */
  time (&startTime);

  /* Digest blocks */
  MDInit (&context);
  for (i = 0; i < TEST_BLOCK_COUNT; i++)
 MDUpdate (&context, block, TEST_BLOCK_LEN);
  MDFinal (digest, &context);

  /* Stop timer */
  time (&endTime);

  printf (" done\n");
  printf ("Digest = ");
  MDPrint (digest);
  printf ("\nTime = %ld seconds\n", (long)(endTime-startTime));
  printf
 ("Speed = %ld bytes/second\n",
  (long)TEST_BLOCK_LEN * (long)TEST_BLOCK_COUNT/(endTime-startTime));
}

/* Digests a reference suite of strings and prints the results.
 */
static void MDTestSuite ()
{
  printf ("MD%d test suite:\n", MD);

  MDString ("");
  MDString ("a");
  MDString ("abc");
  MDString ("message digest");
  MDString ("abcdefghijklmnopqrstuvwxyz");
  MDString
 ("ABCDEFGHIJKLMNOPQRSTUVWXYZabcdefghijklmnopqrstuvwxyz0123456789");
  MDString
 ("12345678901234567890123456789012345678901\
2345678901234567890123456789012345678901");
}

/* Digests a file and prints the result.
```

```
 */
static void MDFile (filename)
char *filename;
{
  FILE *file;
  MD_CTX context;
  int len;
  unsigned char buffer[1024], digest[16];

  if ((file = fopen (filename, "rb")) == NULL)
 printf ("%s can't be opened\n", filename);

  else {
 MDInit (&context);
 while (len = fread (buffer, 1, 1024, file))
   MDUpdate (&context, buffer, len);
 MDFinal (digest, &context);

 fclose (file);

 printf ("MD%d (%s) = ", MD, filename);
 MDPrint (digest);
 printf ("\n");
  }
}

/* Digests the standard input and prints the result.
 */
static void MDFilter ()
{
  MD_CTX context;
  int len;
  unsigned char buffer[16], digest[16];

  MDInit (&context);
  while (len = fread (buffer, 1, 16, stdin))
 MDUpdate (&context, buffer, len);
  MDFinal (digest, &context);

  MDPrint (digest);
  printf ("\n");
}

/* Prints a message digest in hexadecimal.
 */
static void MDPrint (digest)
unsigned char digest[16];
{
```

```
  unsigned int i;

  for (i = 0; i < 16; i++)
 printf ("%02x", digest[i]);
}
```

A.5 Test suite

The MD5 test suite (driver option "-x") should print the following results:

```
MD5 test suite:
MD5 ("") = d41d8cd98f00b204e9800998ecf8427e
MD5 ("a") = 0cc175b9c0f1b6a831c399e269772661
MD5 ("abc") = 900150983cd24fb0d6963f7d28e17f72
MD5 ("message digest") = f96b697d7cb7938d525a2f31aaf161d0
MD5 ("abcdefghijklmnopqrstuvwxyz") = c3fcd3d76192e4007dfb496cca67e13b
MD5 ("ABCDEFGHIJKLMNOPQRSTUVWXYZabcdefghijklmnopqrstuvwxyz0123456789") =
d174ab98d277d9f5a5611c2c9f419d9f
MD5 ("12345678901234567890123456789012345678901234567890123456789012345678
78901234567890") = 57edf4a22be3c955ac49da2e2107b67a
```

Security Considerations

The level of security discussed in this memo is considered to be sufficient for implementing very high security hybrid digital-signature schemes based on MD5 and a public-key cryptosystem.

Author's Address

Ronald L. Rivest
Massachusetts Institute of Technology
Laboratory for Computer Science
NE43-324
545 Technology Square
Cambridge, MA 02139-1986

Phone: (617) 253-5880
EMail: rivest@theory.lcs.mit.edu

Search Notes	Application/Control No.	Applicant(s)/Patent Under Reexamination
	90010831	5,490,216
	Examiner	Art Unit
	MATTHEW HENEGHAN	3992

SEARCHED

Class	Subclass	Date	Examiner

SEARCH NOTES

Search Notes	Date	Examiner
Litigation Search	3/8/10	MH
Review of Prosecution History	4/2/10	MH
Updated Litigation Search	9/8/10	MH
Review for NIRC (Withdrawn)	12/15/10	MH
Updated Litigation Search	4/7/11	MH
Review for NIRC	7/31/11	MH

INTERFERENCE SEARCH

Class	Subclass	Date	Examiner

Reexamination	Application/Control No.	Applicant(s)/Patent Under Reexamination
	90/010,831	5,490,216
	Certificate Date	Certificate Number
		C1

Requester **Correspondence Address:** ☐ **Patent Owner** ☒ **Third Party**

Kyle Rinehart
Klarquist Sparkman, LLP
One World Trade Center, Suite 1600
121 S.W. Salmon Street
Portland, OR 97204

LITIGATION REVIEW ☒	**/MH/** (examiner initials)	**7/31/11** (date)
Case Name		Director Initials
6:11cv33 (OPEN), 6:10cv636 (OPEN)		JR for RG
6:10cv591 (OPEN), 6:10cv472 (OPEN)		JR for RG
6:10cv471 (OPEN)		JR for RG
6:10cv373 (OPEN)		JR for RG
2:08cv3574, 8:08cv203, 1:03cv440, 8:10cv1483, 6:10cv69, 6:10cv18, 6:09cv538 (CLOSED)		JR for RG

COPENDING OFFICE PROCEEDINGS	
TYPE OF PROCEEDING	**NUMBER**
1. None.	
2.	
3.	
4.	

Issue Classification

Application/Control No.	Applicant(s)/Patent Under Reexamination
90010831	5,490,216
Examiner	**Art Unit**
MATTHEW HENEGHAN	3992

ORIGINAL	
CLASS	SUBCLASS
705	59

CROSS REFERENCE(S)

CLASS	SUBCLASS (ONE SUBCLASS PER BLOCK)				
705	56				

INTERNATIONAL CLASSIFICATION					
CLAIMED					NON-CLAIMED
G	0	6	F	1 / 00 (2006.0)	
G	0	6	F	21 / 00 (2006.0)	
G	0	6	Q	30 / 00 (2006.0)	

☒ Claims renumbered in the same order as presented by applicant ☐ CPA ☐ T.D. ☐ R.1.47

Final	Original	Final	Original	Final	Original	Final	Original	Final	Original	Final	Original	Final	Original	Final	Original
1	1	17	17												
2	2	18	18												
3	3	19	19												
4	4	20	20												
5	5														
6	6														
7	7														
8	8														
9	9														
10	10														
11	11														
12	12														
13	13														
14	14														
15	15														
16	16														

NONE		**Total Claims Allowed:**	
(Assistant Examiner)	(Date)	20	
/MATTHEW HENEGHAN/ Primary Examiner.Art Unit 3992	7/31/11	O.G. Print Claim(s)	O.G. Print Figure
(Primary Examiner)	(Date)	1	1

UNITED STATES PATENT AND TRADEMARK OFFICE

UNITED STATES DEPARTMENT OF COMMERCE
United States Patent and Trademark Office
Address: COMMISSIONER FOR PATENTS
P.O. Box 1450
Alexandria, Virginia 22313-1450
www.uspto.gov

Bib Data Sheet

CONFIRMATION NO. 2214

SERIAL NUMBER	FILING OR 371(c) DATE	CLASS	GROUP ART UNIT	ATTORNEY DOCKET NO.
90/010,831	01/22/2010 RULE	705	3993	6620-83067-01

APPLICANTS

5,490,216, Residence Not Provided;
UNILOC (SINGAPORE) PRIVATE LIMITED (OWNER), SINGAPORE, SINGAPORE;
KLARQUIST SPARKMAN, LLP (3RD.PTY.REQ.), PORTLAND, OR;
KLARQUIST SPARKMAN, LLP, PORTLAND, OR

**** CONTINUING DATA ***************************

This application is a REX of 08/124,718 09/21/1993 PAT 5,490,216

**** FOREIGN APPLICATIONS *********************

Foreign Priority claimed	STATE OR COUNTRY	SHEETS DRAWING	TOTAL CLAIMS	INDEPENDENT CLAIMS
☒ yes ☐ no			20	5
35 USC 119 (a-d) conditions met ☒ yes ☐ no ☐ Met after Allowance				
Verified and Acknowledged Examiner's Signature 7/5/11 Initials				

ADDRESS

ABELMAN, FRAYNE & SCHWAB
150 EAST 42nd STREET
NEW YORK, NY10017-5612

26111

TITLE

SYSTEM FOR SOFTWARE REGISTRATION

FILING FEE RECEIVED	FEES: Authority has been given in Paper	
2520	No. _____ to charge/credit DEPOSIT ACCOUNT No. _____ for following:	☐ All Fees ☐ 1.16 Fees (Filing) ☐ 1.17 Fees (Processing Ext. of time) ☐ 1.18 Fees (Issue) ☐ Other _____ ☐ Credit