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

(19) World Intellectual Property Organization
International Bureau

PCT

(43) International Publication Date
27 March 2008 (27.03.2008)

(10) International Publication Number
WO 2008/035141 A2

(51) **International Patent Classification:**
G06F 21/00 (2006.01)

(21) **International Application Number:**
PCT/IB2006/053395

(22) International Filing Date:
20 September 2006 (20.09.2006)

(25) Filing Language: English

(26) Publication Language: English

(71) **Applicants and**
(72) Inventors: KAM-FU, CHAN [CN/CN]; Flat 2003, Block M, Allway Gardens, On Yat Street, Tsuen Wan, Hong Kong, Hong Kong 00000 (CN). **BEAN, LEE** [CN/CN]; c/o Flat 2003, Block M, Allway Gardens, On Yat Steet, Tsuen Wan, Hong Kong, Hong Kong 00000 (CN).

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

(84) Designated States *(unless otherwise indicated, for every kind of regional protection available):* ARIPO (BW, GH, GM, KE, LS, MW, MZ, NA, SD, SL, SZ, TZ, UG, ZM, ZW), Eurasian (AM, AZ, BY, KG, KZ, MD, RU, TJ, TM), European (AT, BE, BG, CH, CY, CZ, DE, DK, EE, ES, FI, FR, GB, GR, HU, IE, IS, IT, LT, LU, LV, MC, NL, PL, PT, RO, SE, SI, SK, TR), OAPI (BF, BJ, CF, CG, CI, CM, GA, GN, GQ, GW, ML, MR, NE, SN, TD, TG).

Published:
— *with declaration under Article 17(2)(a); without abstract; title not checked by the International Searching Authority*

WO 2008/035141 A2

(54) Title: FROM POLYMORPHIC EXECUTABLE TO POLYMORPHIC OPERATING SYSTEM

(57) Abstract:

Description

FROM POLYMORPHIC EXECUTABLE TO POLYMORPHIC OPERATING SYSTEM

Technical Field

[1] This invention relates to the protection of intellectual property, such as executable code, for running in device(s), including computer system(s) or computer-controlled device(s) or operating-system-controlled device(s) or system(s) that is/are capable of running executable code. Such device(s) is/are mentioned hereafter as Device(s).

[2] In particular, this invention relates to the method of creation, distribution and execution in Device(s) of executable code, such as boot code, programmes, applications, device drivers, or a collection of such executables constituting an operating system in the form of executable code embedded or stored into hardware, such as embedded or stored in all types of storage medium, including read-only or rewriteable or volatile or non-volatile storage medium (referred hereafter as the Storage Medium) in the form of virtual disk in physical memory or internal DRAM (Dynamic Random Access Memory) or hard disk or solid state flash disk or ROM (Read Only Memory), or read-only or rewriteable CD/DVD/HD-DVD/Blu-Ray DVD or hardware chip or chipset etc.; this invention being a method, capable of being implemented in executable instructions or programmes in Device(s), providing for running executable code in Device(s) in the form of Polymorphic Executable (PE) or Executable with Unexecutable Code (EUC) or the hybrid of these two, Polymorphic Executable with Unexecutable Code (PEUC), or Polymorphic Operating System (POS) containing PE, EUC and PEUC in an authenticated or authorized state.

[3] In this relation, it makes possible, in Device(s) capable of executing executable code, the phenomenon of executing PE, EUC and PEUC as well as POS in an authenticated or authorized state for the purpose of protecting intellectual property.

Background Art

[4] United States Patent Application No, 20050210274, entitled "Apparatus and method for intellectual property protection using the microprocessor serial number", describes a method for the creation, distribution and execution of software programmes with the use of apparatuses specially designated for encryption and decryption of the software programmes, using at least a part of a serial number or other identifying number stored in a processing unit as the encryption key, the processing unit being the execution environment for decrypting the encrypted software programmes before and then executing the software programmes thus decrypted.

[5] Protection of intellectual property for software programmes or executable code is

essential for innovation and the advance of society based on intellectual activities. Therefore there have been numerous efforts or methods designed for protecting the value of intellectual activities represented in software programmes or executable code; including the use of checking for correct password, checking for encrypted digital key file, checking for signature provided in hardware, checking for correct password through activation procedure over internet, etc. The use of such methods however is either easily bypassed or requires the deployment of specially designed hardware. These methods therefore are either simply ineffective or adding hardware costs and not suiting to general-purpose computing or processing environment.

[6] The problem encountered in the efforts for protecting intellectual activities expressed in the form of software programmes or executable code is therefore how such software programmes or executable code can be prevented from being simply copied or hacked and then executed in a general-purpose computing or processing environment. Uniquely identifiable information, such as a serial number or its derivative, retrieved from hardware used as a key for encryption/decryption of software programmes or executable code with hardly breakable or unbreakable algorithm can be a solution safeguarding such intellectual activities against those simple yet effective pirating activities involving copying and simple hacking.

[7] The patent application mentioned above is such an attempt for solving the problem. This however involves the use of specially designed apparatuses for such purpose that does not constitute a general-purpose computing or processing environment and the decryption procedure and the decrypted software programmes stored in the designed execution environment can also be copied out and pirated by people gaining access to such environment. So such environment has to be a highly secured environment, not to be accessed by the ordinary users, customers or clients of the general mass. In the way described in the aforesaid patent application, the encrypted software programmes cannot run without being decrypted; and for the encrypted software programmes to be executable, they have to be decrypted before they are run so that the decryted software programmes are simply executables which can be copied and runnable in any machines on the same architectural computing or processing platform with a processing unit provided with any serial number. The decrypted software programmes therefore simply become unprotected executables having all functionalities in an unencrypted form that can simply be copied and run without the need of even simple hacking.

Disclosure of Invention

Technical Problem

[8] The technical problem therefore boils down to how software programmes or executable code can be represented in polymorphic forms that are different and yet

runnable in different authenticated or authorized states, but unrunnable (for the features contained in the Polymorphic Code Section(s) as described below) in unauthenticated or unauthorized states in a general-purpose computing or processing environment so that intellectual property contained in such software programmes or executable code can be best protected. So the software programme or the executable code should be an executable, in an executable format, runnable in Device(s) in the general-purpose computing or processing environment, containing polymorphic code that is unique, different as well as runnable in different authenticated or authorized states; and such polymorphic code containing features or functionalities being unrunnable or un-accessible in unauthenticated or unauthorized states.

Technical Solution

[9] Simply put, the technical solution to the technical problem described above is to represent the executable code to be protected in the form of Polymorphic Executable (PE). Conventionally, an Executable is a body of executable code that can be loaded up in Device(s) and can execute or can be executed. In view of this invention, the definition of an Executable can be further refined as being a body of code that can be loaded up in Device(s), at least some part of which can execute or can be executed. A PE, being a subset of an Executable, is unique and different and yet runnable in different authenticated or authorized states; but the Polymorphic Code Section(s) (and the features or functionalities therein) contained within a PE is/are unexecutable in unauthenticated or unauthorized states in Device(s) in a general-purpose computing or processing environment. So the code in Polymorphic Code Section(s) is encrypted un-executable code when not in use, and this encrypted unexecutable code is different before it is decrypted for running in different authenticated or authroized states; and it is decrypted into executable code while the PE is running for its successful execution in the general-purpose computing or processing environment, performing the same function(s) as intended in the same authenticated or authorized states.

[10] The Polymorphic Code Section(s) in a PE is/are therefore best generated by hardly breakable or unbreakable alogrithm(s) of encryption/decryption; i.e. an encryption procedure with a corresponding decryption procedure of hardly breakable or un-breakable algorithm(s). And what encryption/decryption alogrithm(s) to be used is/are a matter of choice and can be updated with the advance of cryptographic science.

[11] So a PE is in an executable format that can be executed or run in Device(s) in a general-purpose computing or processing environment of any architectural hardware platform. In this format, a PE has the following sections:

[12] (A) General Code Section(s)

[13] The General Code Section(s) contain(s) executable code in an unencrypted format that is executable without the need for encryption/decryption. The General Code

Section(s), amongst performing other functionalities, contain(s):

[14] Retrieving Function(s) (RF), function(s) for retrieving, whether locally or through local network or internet, identifying information or identifier(s) from the general-purpose computing or processing environment for use in the Decrypting Function(s) (DF) in also the General Code Section(s), and

[15] Decrypting Function(s), which use(s) the identifying information or identifier(s) retrieved by the RF for decrypting the Polymorphic Code Section(s) corresponding to the Encryption Function(s) (EF) used for encrypting the Polymorphic Code Section(s).

[16] So there can be more than one pair of encryption/decryption algorithm used. Different Polymorphic Code Section(s) can be decrypted with different DF corresponding to different EF used for its/their encryption.

[17] Depending on need, the General Code Section(s) may also contain Exception Function(s) (ExF), which handle(s) exception(s) when the code of the Polymorphic Code Section(s) after being decrypted and placed back into the corresponding location in the system memory image of the PE (i.e. where the PE is loaded in the system memory) is not executable. For instance, the DF may call the the ExF after decrypting the corresponding Polymorphic Code Section(s) and placing the resulted code into the corresponding location in the system memory image of the PE. The ExF may then perform some verification activities, such as processing the corresponding decrypted Polymorphic Code Section(s) to determine whether the resulted code in the corresponding location in the system memory image of the PE is in the executable format as the code in the General Code Section(s). For example, the ExF may initiate other error-handling activities, such as reporting errors or doing preparation for exiting and c losing down the PE nicely if a certain signature is not found in the decrypted code. Such signature may be a checksum calculated from the Polymorphic Code Section execution code, or some other static data of choice appended to the beginning or the end of the Polymorphic Code Section(s) or inserted somewhere therein. And if after decryption, such signature is not correct, then the ExF may initiate other error-handling activities. If the signature is correct, the ExF does action that initiates the execution of the corresponding decrypted code.

[18] The functionalities of ExF so described may also be incorporated in DF, so separate ExF may not be necessary. ExF or such ExF functionalities as incorporated in DF are optional.

[19] (B) Polymorphic Code Section(s)

[20] Polymorphic Code Section(s) contain(s) encrypted code for those features or functions of intellectual activities to be protected. The encrypted code in the Polymorphic Code Section(s) is to be decrypted for execution by the DF corresponding to the EF that is used for encrypting the code.

[21] Upon execution, the PE is loaded up into the system memory and is executed. The General Code Section(s) of the PE in the system memory contain(s) executable code for execution in Device(s) in the general-purpose computing or processing environment without the need for decryption. For running the encrypted code in the Polymorphic Code Section(s), which are not executable as they are in encrypted format, the RF in the General Code Section(s) is executed and the RF passes the identifying information or identifier(s) retrieved from the general-purpose computing or processing environment to DF for use for decrypting the encrypted code of the corresponding Polymorphic Code Section(s).

[22] The DF, after decrypting the encrypted code of the corresponding Polymorphic Code Section(s), places back the decrypted code into the corresponding location in the system memory image where the PE is loaded up.

[23] So if the general-purpose computing or processing environment at the time is in an authenticated or authorized state, the decrypted code should be in the right executable format as the same as the code in General Code Section(s). So they can be executed correctly. The ExF or such ExF functionalities as incorporated in DF may first do the error-handling activites to detemine whether to initiate action to execute the decrypted code or not as described above.

[24] An authenticated or authorized state is a state in which the identifying information or identifier(s) as retrieved by the RF from the general-purpose computing or processing environment can be and is used by the DF to produce the decrypted code that is executable and performs the features or functionalities that are originally so designed. That is the encrypted code is rightly decrypted with the use of the identifying information or identifier(s) retrieved by the RF and passed to the DF for decryption for successful execution. An unauthenticated or unauthorized state is a state in which the identifying information or identifier(s) as retrieved by the RF from the general-purpose computing or processing environment, after being used by the DF, cannot yield decrypted code that is executable and the resulted code after decryption does not perform successfully the features or functionalities that are originally so designed. That is, the encrypted code is not rightly decrypted with the use of such identifying information or identifier(s) as retrieved by the RF and passed to the DF. In this case, the DF does not produce the right code for execution as originally designed.

[25] The identifying information or identifier(s) retrieved by the RF can be, but not limited to, the serial number(s) or the derivative(s) of such serial numbers of the Central Processing Unit(s) or a particular piece of hardware or a combination of pieces of hardware as found within or accessible to the general-purpose computing or processing environment through network including local network and internet. This identifying information or identifier(s) is/are unique to the purpose for which it is or

they are intended; for examples, such as unique in identifying a certain piece or a certain set or class of equipment produced by a manufacturer, or for identifying a certain manufacturer for all the equipment it makes. It can also be an identifier of an individual person or a target group of persons, such as fingerprint or any other biometric information that can be obtained from that individual person or a target group of persons through the general-purpose computing or processing environment, whether networked or not. Such identifying information or identifier(s) of course has/ have to be made accessible for the purpose of encryption in the stage for making a PE. This access of identifying information or identifier(s) for encryption purpose can be by transmission over local network or internet or through ordinary mails or by other means of transmission or other ways of delivery or distribution; such identifying information or identifier(s) is/are therefore to be transmitted from the general-purpose computing or processing environment where it is/they are collected and intended for use by the PE under concern to where it is/they are used for encryption for making the PE under concern.

[26] Such identifying information or identifier(s) collected from the general-purpose computing or processing environment for being used as a key for the purpose of applying encryption/decryption on the Polymorphic Code Section under concern, as will be described below, is/are best to be omitted in the encryption/decryption key holder within the Polymorphic Code Section for the best protection of the PE. If to be placed there, such identifying information or identifier(s) is/are best to be encrypted by using an encryption/decryption routine where the decryption routine for the purpose of decrypting such identifying information or identifier(s) should be made available, during the running of the PE, for decrypting such identifying information or identifier(s), which after decryption can then be used for decrypting the rest of the Polymorphic Code Section under concern.

[27] Furthermore, instead of using such identifying information or identifier(s) collected from the general-purpose computing or processing environment as the encryption/ decryption key, some other identifying information or identifier(s) can be supplied, during the encryption process or during the PE is in actual running, for being used as the encryption/decryption key for the encryption/decryption processing of the Polymorphic Code Section under concern. This type of "supplied" identifying information or identifier(s) is distinguished from the type of "collected" identifying information or identifier(s) in the sense that the "collected" type is usually static information or identifier(s) that the PE, while running, can collect by itself, such as the hardware serial numbers of the Device in which the PE is running. The "supplied" type is usually dynamic information or identifier(s) that has/have to be supplied to the PE by dynamic user input when the PE is running, such as the password emailed to the

registered user by the maker of the PE, or the fingerprint of the registered user, or the digital key file passed over internet to the Device and supplied to the PE for use while the PE is running, or a combination of these.

[28] In addition, such other identifying information or identifier(s) so supplied can also be used together with the identifying information or identifier(s) collected from the general-purpose computing or processing environment as the encryption/decryption key. All such identifying information or identifier(s), or either type of the "collected" or "supplied" identifying information or identifier(s) alone, can be placed or omitted in the encryption/decryption key holder within the Polymorphic Code Section. If to be so placed, as mentioned earlier, for the best protection, such all identifying information or identifiers is/are best to be encrypted by using an encryption/decryption routine where the decryption routine for the purpose of decrypting such identifying information or identifier(s) should be made available, during the running of the PE, for decrypting such identifying information or identifier(s), which after decryption can then be used for decrypting the rest of the Polymorphic Code Section under concern.

[29] And whether to place all or part of such identifying information or identifier(s) or not to place altogether in the encryption/decryption key holder within the Polymorphic Code Section and whether such identifying information or identifier(s) if so placed is/ are to be further encrypted/decrypted is up to the designer or maker of the PE according the purpose for which the PE is intended, the environment in which the PE is run and the degree of security and protection that the PE is intended to attend and achieve.

[30] How a PE in an executable format can be produced is illustrated by the following implementation steps and examples. The implementation steps and examples described outline in a broad way the method by which the PE is to be designed, programmed and Polymorphic-Section-encrypted. People skilled in the art can make similar implementation with slight variations of their choice and design.

[31] When the executable is written, it is not known where the executable will be loaded up in the system memory in runtime. Memory references to static variables of the executable are stored as offsets to the beginning of the run-time image. The executable contains a special section called relocation table that has pointers to these offsets. When the executable is loaded, the operating system will automatically add the run-time load address of the executable to these offsets, so that the memory references will be valid. However if such relocation, i.e. adding the run-time load address to offsets described above, occurs to a Polymorphic Code Section upon loading, it will destroy the integration of code and render it unexecutable after decryption. Therefore, such relocation must be eliminated in Polymorphic Code Section.

[32] Below is a description of how to achieve this (i.e. to eliminate such relocation by

using other programming techniques instead of those techniques which would have made such relocation) using the C progamming language, the same principle applies to using other programming languages as well.

[33]

[34] (a) Avoid making memory reference to global variable in Polymorphic Code Section. For example, the following function is not to be included in a Polymorphic Code Section.

[35]

[36] int global_variable;

[37]

[38] // Polymorphic Code Section starts here

[39]

[40] int foo()

[41] {

[42] return global_variable;

[43] }

[44] // Polymorphic Code Section ends here

[45]

[46] (b) Avoid making memory reference to static local variable in Polymorphic Code Section. For example, the following function is not to be included in a Polymorphic Code Section.

[47]

[48] // Polymorphic Code Section starts here

[49] int foo()

[50] {

[51] static int static_variable;

[52] return static_variable;

[53] }

[54] // Polymorphic Code Section ends here

[55]

[56] (c) Avoid using static string in Polymorphic Code Section. For example, the following function is not to be included in a Polymorphic Code Section.

[57]

[58] // Polymorphic Code Section starts here

[59] char* foo()

[60] {

[61] return "Hello, world";

[62] }

[63] // Polymorphic Code Section ends here

[64]

[65] (d) Avoid using pointers to function or global variable in Polymorphic Code Section. For example, the following referencing is not to be included in a Polymorphic Code Section.

[66]

[67] int global_variable;

[68]

[69] void foolO

[70] {

[71] }

[72]

[73] // Polymorphic Code Section starts here

[74] void foo2()

[75] {

[76] }

[77]

[78] int foo3()

[79] {

[80] void *pl,*p2,*p3;

[81]

[82] pl=&fool;

[83] p2=&foo2;

[84] p3=&global_variable;

[85] }

[86] // Polymorphic Code Section ends here

[87]

[88] In fact, it is possible to achieve the intended result of the above functions, as revealed by this invention, through using a wrapper function which is to be placed in the General Code Section(s) outside the Polymorphic Code Section under concern. For example, the following ways of coding are valid, where (e) to (h) correspond to (a) to (d) respectively:

[89]

[90] (e) Instead of making memory reference to global variable in Polymorphic Code Section, one can make memory reference to global variable with wrapper.

[91]

[92] int global_variable;

[93]

```
[94]    int global_variable_wrapper()
[95]    {
[96]    return global_variable;
[97]    }
[98]
[99]    // Polymorphic Code Section starts here
[100]   int fooO
[101]   {
[102]   return global_variabke_wrapper();
[103]   }
[104]   // Polymorphic Code Section ends here
[105]
```

[106] (f) Instead of making memory reference to static local variable in Polymorphic Code Section, one can replace static local variable with global variable, and use the method described in (e) above in making such memory reference.

[107]

[108] (g) Instead of using static string in Polymorphic Code Section, one can use static string with wrapper.

```
[109]
[110]   char* static_string_wrapper()
[111]   {
[112]   return "Hello, world";
[113]   }
[114]   // Polymorphic Code Section starts here
[115]   char* foo()
[116]   {
[117]   return static_string_wrapper() ;
[118]   }
[119]   // Polymorphic Code Section ends here
[120]
```

[121] (h) Instead of using pointers to function or global variable in Polymorphic Code Section, one can use pointers to function or global variable with wrapper.

```
[122]
[123]   int global_variable;
[124]
[125]   void foolO
[126]   {
[127]   }
```

```
[128]
[129]   void foo2(); // Forward declaration
[130]
[131]   void* function_pointer_wrapper 1()
[132]   {
[133]    return &fool;
[134]   }
[135]   void* function_pointer_wrapper2()
[136]   {
[137]   return &foo2;
[138]   }
[139]
[140]   void* global_variable_pointer_wrapper()
[141]   {
[142]   return &global_variable ;
[143]   }
[144]
[145]   // Polymorphic Code Section starts here
[146]   void foo2()
[147]   {
[148]   }
[149]
[150]   int foo3()
[151]   {
[152]    void *pl,*p2,*p3;
[153]
[154]   pi=function_pointer_wrapper 1();
[155]   p2=function_pointer_wrapper2();
[156]   p3=global_variable_pointer_wrapper() ;
[157]   }
[158]   // Polymorphic Code Section ends here
[159]
```

[160] The coding and encryption process for making Polymorphic Executable is described as follows:

[161] (1) Select the programming language of choice for writing the PE;

[162] (2) Decide and design what feature(s) or function(s) is/are to be included in General Code Section(s) or in Polymorphic Code Section(s);

[163] (3) Design and write General Code Section(s) as normally would be written for

normal code which does not require encryption/decryption and, if and where necessary, incorporating changes or revisions resulting from the need for providing for the writing of Polymorphic Code Section(s) according to the rules mentioned above and reproduced as in (4) a) to (4) d) below;

[164] (4) Design and write Polymorphic Code Section(s) according to the rules mentioned above and reproduced as follows:

[165] (4) a) Instead of making memory reference to global variable in Polymorphic Code Section, making memory reference to global variable with wrapper;

[166] (4) b) Instead of making memory reference to static local variable in Polymorphic Code Section, replacing static local variable with global variable, and using the step described in (4) a) in making such memory reference;

[167] (4) c) Instead of using static string in Polymorphic Code Section, using static string with wrapper;

[168] (4) d) Instead of using pointers to function or global variable in Polymorphic Code Section, using pointers to function or global variable with wrapper;

[169] (5) Add a Polymorphic Code Section Header function with a Polymorphic Code Section Header at the beginning of Polymorphic Code Section(s) when writing the programme. The header contains:

[170] (5) a) at least a header signature holder for holding a header signature. This holder and the header signature to be placed within should be long enough so that the header signature used can be easily distinguished and identified and will not be easily mistaken for normal execution code. For instance, a One-Byte header signature may be easily mistaken as normal execution code and may not be long enough for the purpose of clear-cut identification. The programme designer or maker can select a signature of choice for the header signature so that the PE can easily identify where the Polymorphic Code Section begins. This header signature is used for identifying where the encryption/decryption of the Polymorphic Code Section should begin. During the encryption process for making the PE, this header signature is used for making encryption of the Polymorphic Code Section. For the best protection, after encryption, this header signature can be scrambled at random or encrypted to prevent the Polymorphic Code Section from being easily located by hackers. However it is optional to scramble at random or encrypt it or not, depending on the purpose, the degree of security and protection that programme designer or maker wants to achieve. This header signature is not necessary for the PE, while executing, for identifying where the Polymorphic Code Section begins as the PE can know this by making reference to the Polymorphic Code Section Header function instead.

[171] (5) b) The encryption/decryption key holder for holding the identifying information or identifier(s) for encryption/decryption key. This holder can be omitted if the

encryption/decryption key contains only the identifying information or identifier(s) collected or supplied from the general-purpose computing or processing environment as mentioned above. If the encryption/decryption key holder contains identifying information or identifier(s), whether collected or supplied from the general-purpose computing or processing environment or supplied in the process of making and encrypting the PE, to be used for the encryption/decryption process, this holder should be long enough to hold such identifying information or identifier(s) as intended. If not omitted, this encryption/decryption key holder together with such identifying information or identifier(s), whether encrypted or not, should be placed just ahead of the header signature holder at (5) a) above; i.e. it is to be placed at the beginning of the Polymorphic Code Section. Such identifying information or identifier(s) placed in the encryption/decryption key holder can be unencrypted. Or if it is to be encrypted in the encryption process and decrypted for use for decrypting the Polymorphic Code Section under concern, such encryption/decryption should be done by using an encryption/decryption routine where the decryption routine for this purpose should be made available, during the running of the PE, for decrypting such identifying information or identifier(s), which after decryption can then be used for decrypting the rest of the Polymorphic Code Section under concern.

[172] (5) c) The error-checking signature holder for holding the error-checking signature for the purpose of checking whether the decrypted code is executable. The checksum calculated from the unencrypted executable code of the Polymorphic Code Section can be used for such purpose. And if such checksum is used, it is to be placed here before the Polymorphic Code Section is encrypted and goes through the encryption/decryption process as the rest of the Polymorphic Code Section as a whole. This holder may also contain some other static data of choice as described above for use in ExF error handling. This holder can be omitted if error-checking is not to be done.

[173] (6) Add a Polymorphic Code Section Footer function with a Polymorphic Code Section Footer at the end of the Polymorphic Code Section(s). The footer contains a footer signature holder for holding a footer signature. This holder and the footer signature to be placed within should be long enough so that the footer signature used can be easily distinguished and identified and will not be easily mistaken for normal execution code. For instance, a One-Byte footer signature may be easily mistaken as normal execution code and may not be long enough for the purpose of clear-cut identification. The programme designer or maker can select a signature of choice for the footer signature so that the PE can easily identify where the Polymorphic Code Section ends. This footer signature is used for identifying where the encryption/decryption of the Polymorphic Code Section should end. During the encryption process for making the PE, this footer signature is used for making encryption of the Polymorphic Code

Section. For the best protection, after encryption, this footer signature can be scrambled at random or encrypted to prevent the Polymorphic Code Section from being easily located by hackers. However it is optional to scramble at random or encrypt it or not, depending on the purpose, the degree of security and protection that programme designer or maker wants to achieve. This footer signature is not necessary for the PE, while executing, for identifying where the Polymorphic Code Section ends as the PE can know this by making reference to the Polymorphic Code Section Footer function instead.

[174] (7) Compile the programme, and generate the executable for the platform of general-purpose computing or processing environment in which the executable is intended to be run.

[175] (8) Use the executable generated in (7) or make a copy of such executable for such purpose, if necessary, and run an encryption programme of choice for encrypting such executable or its copy respectively. This encryption process can be done using as many and different encryption/decryption keys for as many and different authenticated or authorized states as required. It does the following process in sequence:

[176] (8) a) Scan the executable for the Polymorphic Code Section Header and Polymorphic Code Section Footer signatures of the Polymorphic Code Section(s) to determine where the encryption of the Polymorphic Code Section(s) begin(s) and end(s).

[177] (8) b) For each Polymorphic Code Section found, calculate the checksum, and fill it into the error-checking signature holder field of the Polymorphic Code Section Header. The checksum is calculated from the executable code of the Polymorphic Code Section, excluding information contained within the encryption/decryption key holder, the header signature holder, the footer signature holder and the error-checking signature holder, if any. After calculating this checksum in this way, the checksum is placed into the error-checking signature holder and is encrypted together for its protection also with the rest of the Polymorphic Code Section under concern except the encryption/decryption key holder, if any. If checksum is not used for such error-checking purpose and some other static data of choice is to be used for such purpose, such other static data has to be placed into the error-checking signature holder and is encrypted likewise with the rest of the Polymorphic Code Section under concern. This step can be omitted if checking is not intended to be done.

[178] (8) c) Encrypt the Polymorphic Code Section(s), using the encryption/decyption algorithm(s) of choice and using, as the encryption/decryption key, the identifying information or identifier(s) supplied, if any, by the programme designer or maker of the Polymorphic Code Section(s), and/or the identifying information or identifier(s) collected from the Device(s) in the general-purpose computing or processing en-

vironment and/or other identifying information or identifier(s) as supplied by the user for the encryption/decryption purpose through all means of transmission or delivery, including transmission over local network or internet or through ordinary mails or by other means of transmission or other ways of delivery or distribution. If the encryption/decryption key holder at the Polymorphic Code Section Header is omitted so that the identifying information or identifier(s) intended to be used as the encryption/decryption key is/are to be best protected, such identifying information or identifier(s) should be made available for encryption/decryption purpose and be collected from the general-purpose computing or processing environment or so supplied while the PE is in actual running. If the encryption/decryption key holder exists, the above mentioned identifying information or identifier(s) or part of which is/are then placed, according to the design of the PE or the purpose for which the PE is designed, into the encryption/decryption key holder at the Polymorphic Code Section Header. As mentioned earlier, the checksum, if any, should be encrypted as well so that it is not to be modified easily. After encryption, for the best protection, the header signature and footer signature should be scrambled or encrypted again so that these signatures are not to be or cannot be used for locating the Polymorphic Code Section(s) in the PE after encryption. However it is optional to scramble at random or encrypt these signatures or not, depending on the purpose, the degree of security and protection that programme designer or maker wants to achieve. The encryption/decryption key in the encryption/decryption key holder can be:

[179] (8) c) i) containing only the identifying information or identifier(s) collected or supplied from the general-purpose computing or processing environment as mentioned above. If so contained, such identifying information or identifier(s) is/are best to be encrypted; or

[180] (8) c) ii) containing some other identifying information or identifier(s) so supplied in the process of making or encrypting the PE, such as a random number or an identifier of the version number of the PE or the identifier of the maker of the PE, etc. If so contained, such identifying information or identifier(s) is/are best to be encrypted; or

[181] (8) c) iii) containing the identifying information or identifier(s) of (8) c) i) and (8) c) ii). If so contained, such identifying information or identifier(s) is/are best to be encrypted; or

[182] (8) c) iv) omitted if the identifying information or identifier(s) intended to be used as the encryption/decryption key is/are to be best protected, provided such identifying information or identifier(s) can be collected for encryption/decryption purpose from the general-purpose computing or processing environment or so supplied while the PE is in actual running.

[183] So in the implementation of the method revealed in this invention, each Polymorphic Code Section is marked by a Polymorphic Code Section Header function in the beginning and by a Polymorphic Code Section Footer at the end. It begins with the encryption/decryption key holder, if any. If no encryption/decryption key holder, the Polymorphic Code Section begins at the Polymorphic Code Section Header signature holder. The normal encrytion/decryption processing, i.e. excluding the encryption/decryption key if any, of the Polymorphic Code Section begins after the encryption/decryption key holder, if any, and begins at the Polymorphic Code Section Header signature holder and ends at and includes the Polymorphic Code Section Footer signature holder. As mentioned earlier, the encryption/decryption key, if any, for the Polymorphic Code Section is placed outside and ahead of the rest of the Polymorphic Code Section. The rest of the Polymorphic Code Section undergoes the normal encryption/decryption process, whereas the encryption/decryption key, if any, does not. The encryption/decryption key, if any, is either unencrypted; or encrypted/ decrypted as described earlier above by using an encryption/decryption routine where the decryption routine for this purpose should be made available, during the running of the PE, for decrypting the encryption/decryption key, which after decryption can then be used for decrypting the rest of the Polymorphic Code Section under concern.

[184] People skilled in the art may design implementation variations that can achieve the same effects according to the ideas of the method revealed in this invention. For instance, as mentioned earlier, it is optimal to have the encryption/decryption key holder and the error-checking signature holder in the Polymorphic Code Section Header or these two holders can be designed to be placed within the Polymorphic Code Section Footer instead. The error-checking and exception handling activities as embodied in ExF or in DF is also optional, depending whether such handling is required or not. Or People skilled in the art may do away with the use of Polymorphic Code Section Header function and Polymorphic Code Section Footer function altogether and design some other ways of identifying the Polymorphic Code Section(s) as long as the PE so designed can be executed and the Polymorphic Code Sections within can be identified correctly for encryption/decryption and execution purpose. In this invention later, description will also be given to implementation variations that convert a Polymorphic Executable into an Executable with Unexecutable Code or into a Polymorphic Executable with Unexecutable Code.

[185] The running of the PE and the related decryption process is described as follows:

[186] (9) When the PE is loaded, the Polymorphic Code Section(s) is/are loaded into memory just like normal code.

[187] (10) When the PE needs to call a function that is inside a Polymorphic Code Section. Through programme design, the PE is able to know which Polymorphic Code

Section that the function is supposed to be in. Before calling a function in a Polymorphic Code Section, the PE first locates Polymorphic Code Section by referencing the location of its corresponding Polymorphic Code Section Header function and Polymorphic Code Section Footer function. Using such locational information, the PE then decrypts the Polymorphic Code Section except the encryption/decryption key holder, if any, by using the corresponding DF as follows:

[188] (10) a) The PE must check whether the Polymorphic Code Section has been decrypted before doing decryption again. For instance, such checking can be done by implementing an encryption/decryption stack for the Polymorphic Code Section as described below in relation to multiple encryption/decryption. If the Polymorphic Code Section has been decrypted, the PE simply calls the function within the Polymorphic Code Section as intended and goes to take (10) f) below. If the Polymorphic Code Section has not been decrypted, then the PE goes to take (10) b) below;

[189] (10) b) If there is no encryption/decryption key holder in the Polymorphic Code Section Header of the Polymorphic Code Section, the DF uses the identifying information or identifier(s) as returned by the RF from the Device in the general-purpose computing or processing environment or such identifying information or identifier(s) so supplied for the purpose when the PE is now in actual running, or both sets of such identifying information or identifier(s) altogether, as the key for decryption and goes to take (10) c) below. If there is an encryption/decryption key inside the encryption key holder, the DF extracts the encryption/decryption key from the Polymorphic Code Section Header. If the encryption/decryption key so extracted has not been encrypted, the DF goes to take (10) c) below. And if for the best protection, and if such a key has been previously encrypted, the DF decrypts the key by calling the decryption routine that corresponds to the encryption routine that has been used for encrypting the key. In some implementation, this key is only a partial key and can be used together with the identifying information or identifier(s) as returned by the RF from the Device in the general-purpose computing or processing environment or such identifying information or identifier(s) so supplied for the purpose when the PE is now in actual running, or together with both sets of such identifying information or identifier(s) altogether, as the key for decryption. The DF then continues to take (10) c) below;

[190] (10) c) The DF decrypts the Polymorphic Code Section except the encryption/decryption key holder and the key so contained, using the key obtained in (10) b); and then continues to take (10) d) below;

[191] (10) d) If there is an error-checking signature holder holding a checksum or an identifying signature, the DF then calculates the checksum of the executable code and compares it to the value stored in the error-checking signature holder in the Polymorphic Code Section Header or verifies whether the identifying signature is

correct. If there is no match or the identifying signature is not correct, the code is corrupted and cannot be used. The DF jumpes to its error handling routines or calls ExF for such purpose. If there is a match or the identifying signature is correct or if there is no an error-checking signature holder, the DF returns and the PE continues to take (10) e) below;

[192] (10) e) The PE calls the funtion as it intends to call in the Polymorphic Code Section just decrypted. The PE then goes to take (10) f) below if required; if (10) f) below is not required to be taken, the function so called by the PE returns, and the PE continues with the programme logics immediately following the return of the function so called;

[193] (10) f) If extra security is required, the function so called by the PE should call the corresponding encryption algorithm to re-encrypt the Polymorphic Code Section before it returns. Or the function simply returns without calling the corresponding encryption algorithm for re-encyption. And the PE continues with the programme logics immediately following the return of the function so called.

[194] The implementation of the above method can be done in many programming languages and many operating systems. The programming language requirement is that the language used permits inserting raw data into the code segment, and that it permits marking a section of code. For example, in C language, raw data can be inserted into code section using inline assembly. Functions are compiled in sequence. Therefore it permits marking a section, for instance, by placing two functions, the section-header function at the beginning and the section-footer function at the end, for this purpose as described above in relation to the Polymorphic Code Section Header function and the Polymorphic Code Section Footer function. Assembly and many structure-programming-languages, such as C, C++ and Pascal, meet this requirement. The operating system requirement is that it permits executable to write to the code segment. Some operating system, like DOS, including for instances MSDOS and FreeDOS, does not protect the code segment, thus nothing needs to be done. Other operating systems such as those in the Microsoft Windows family, code segment cannot be usually written over. However, using special link options, the attribute of the code segment can be changed, making the code segment writeable.

[195] For further improvements, encryption/decryption scheme revealed in this invention can be modified or enhanced in the following ways:

[196] (11) One can apply multiple encryption/decryption algorithms of choice. Each section can go through multiple passes of encryption/decryption by using an encryption/decryption stack. Pushing to the stack means adding another pass of encryption, while popping from the stack means removing the outmost layer of encryption. Each section has a separate stack, which can be different to a stack that

belongs to another section.

[197] (12) At run-time, the PE can modify (push or pop) the encryption/decryption stack at will for any given Polymorphic Code Section, as long as the stack must be empty before calling a function in the section.

[198] (13) When a call to a function inside a Polymorphic Code Section returns, the PE, if needed, can recreate the encryption/decryption stack for the section. The algorithm and/or key for the new encryption/decryption in the stack may or may not be the same as the original one.

[199] (14) One can embed a Polymorphic Code Section inside another Polymorphic Code Section, each with its encryption/decryption stack. Depending on the level of security needed, multiple nesting or wrapping of these Polymorphic Code Sections can be implemented. Of course, to use the inner section, one must first decrypt its outer or wrapping section(s) or empty the corresponding encryption/decryption stack(s) for its outer or wrapping section(s).

[200] (15) An encryption/decryption routine can be hidden inside a Polymorphic Code Section. Of course, to use this routine, one must first decrypt the Polymorphic Code Section using another encryption/decryption routine that is not encrypted or has been decrypted previously.

[201] So the enhancements described above represent various methods for further protecting the function(s) or feature(s) embedded within the Polymorphic Code Section(s) of an executable. Simply put, they can be summarized as the method of applying the technique of nesting Polymorphic Code Section(s); the method of applying the technique of embedding hidden encryption/decryption algorithm(s) within nested Polymorphic Code Section(s); the method of applying the technique of implementing multiple passes of encryption/decryption; the method of applying the technique of implementing encryption/decryption with multiple encryption/decryption alogrithms; as well as the method of applying the technique of implementing dynamic single pass and/or multiple passes of encryption/decryption with or without the use of stacking, that is encryption/decryption and re-encryption/re-decryption of the same Polymorphic Code Section(s) while the executable is being executed, for the creation of any executable(s) using Polymorphic Code Section(s).

[202] Any executable or programme or executable code can be turned into a Polymorphic Executable. An operating system is a collection of executables, programmes or executable code. So any operating system can be turned into a Polymorphic Operating System consisting of its corresponding Polymorphic Executables. So this invention also applies to operating system in general, besides individual executable or programme or executable code. And a Polymorphic Operating System does not necessarily have to contain all executables in the form of Polymorphic Executables;

only those functions or features that the Polymorphic Operation System is designed to protect or the small number of installation files, start-up files or crucial kernel files that help to install, activate, authenticate and start-up the Polymorphic Operation System under concern are good candidates for putting into Polymorphic Executables.

[203] For convenience, Polymorphic Operating System is considered here an operating system made up of, in different combinations, ordinary Executables, Polymorphic Executables and the related variants. The related variants of PE are Executable with Unexecutable Code (EUC) and Polymorphic Executable with Unexecutable Code (PEUC).

[204] EUC and PEUC are better conceived as variants of PE that are used in Device(s) in the general-purpose computing or processing environment that is/are connected to network, whether local network or internet; or the executable code for replacing the unexecutable code in the Unexecutable Code Section(s) of EUC and/or PEUC is to be retrieved, whether through local network or internet, from somewhere within or accessible to the general-purpose computing or processing environment in which the corresponding EUC and/or PEUC is/are executed.

[205] EUC is an executable containing executable General Code Section(s) with executable code and unexecutable Unexecutable Code Section(s) with unexecutable code, whether in a pattern of all zeros or all ones or in one of the patterns ranging from all zeros to all ones or in a scramble-at-random pattern. PEUC is an executable containing executable General Code Section(s) with executable code, at least one or more Polymorphic Code Section(s) with encrypted unexecutable code when not in use and decrypted executable code when in use, as well as one or more unexecutable Unexecutable Code Section(s) with unexecutable code, whether in a pattern of all zeros or all ones or in one of the patterns ranging from all zeros to all ones or in a scramble-at-random pattern. This is so with the following qualification. That is, the Unexecutable Code Section(s) of EUC or PEUC may however contain some matching information (superimposed on the pattern described above) between the the encrypted Polymorphic Code Section(s) and the corresponding Unexecutable Code Section(s) for identifying where the Unexecutable Code Section(s) is/are located for its/their replacement by its/their corresponding encrypted Polymorphic Code Section(s) while the EUC or PEUC is running if the locational information, about where the encrypted Polymorphic Code Section(s) begin(s) or end(s) in the corresponding PE, is not or has not been saved or stored with the corresponding encrypted Polymorphic Code Section(s) to be saved or stored in the server or computer or machine or device administering such encrypted Polymorphic Code Section(s) and the corresponding locational information for the purpose of delivering or transmitting such encrypted Polymorphic Code Section(s) and the corresponding locational information for use

later by the EUC or PEUC running Device(s).

[206] The unexecutable code in the Unexecutable Code Section(s) of EUC and PEUC is to be replaced, while the EUC and PEUC are executing, by the corresponding Polymorphic Code Section(s) with encrypted unexecutable code when not in use and decrypted executable code when in use for the successful execution of EUC and PEUC in the general-purpose computing or processing environment in authenticated or authorized states. For receiving the corresponding Polymorphic Code Section(s) and the corresponding relational information, if any, for replacing the corresponding Un-executable Code Section(s), the EUC or the PEUC should have function(s) for such purpose; that is for the purpose of receiving the corresponding Polymorphic Code Section(s) and the corresponding relational information and replacing the Un-executable Code Section(s) by the corresponding Polymorphic Code Section(s). If the corresponding relational information for the corresponding Polymorphic Code Section(s) is not available, the EUC or the PEUC should have also function(s) for identifying where the corresponding Polymorphic Code Section(s) should go for replacing the Unexecutable Code Section(s). Such function(s) or functionalities can be placed in the General Code Section(s), for instance either separately as separate function(s) or incorporated into the Retrieving Function(s) (RF) or Decrypting Function(s) (DF) therein, or made available for use when such function(s) have to be used for the aforesaid purposes.

[207] The process of creating EUC or PEUC is as follows, where all Polymorphic Code Section(s) in a PE is/are converted to Unexecutable Code Section(s) for creating EUC and where only some, but not all, Polymorphic Code Section(s) in a PE is/are converted to Unexecutable Code Section(s) for creating PEUC:

[208] (16) a) creating a corresponding PE with Polymorphic Code Section(s) encrypted therein as described in (8) above;

[209] (16) b) extracting the encrypted Polymorphic Code Section(s) out of the corresponding PE. The encrypted Polymorphic Code Section(s) so extracted is/are to be saved or stored in the server or computer or machine or device administering such encrypted Polymorphic Code Section(s) for the purpose of delivering or transmitting such encrypted Polymorphic Code Section(s) for use later by the EUC or PEUC running in Device(s); or should be made available, for instance such as saved or stored in some accessible fixed or removeable storage medium thus installed or presented to the EUC or PEUC running in Device(s).

[210] (16) c) ascertaining and saving/storing the locational information, about where the encrypted Polymorphic Code Section(s) begin(s) or end(s) in the corresponding PE. This locational information is best to be saved or stored with the corresponding encrypted Polymorphic Code Section(s) in the server or computer or machine or device

administering such encrypted Polymorphic Code Section(s) and the corresponding locational information for the purpose of delivering or transmitting such encrypted Polymorphic Code Section(s) and the corresponding locational information for use later by the EUC or PEUC running in Device(s); or should be made available, for instance such as saved or stored in some accessible fixed or removeable storage medium thus installed or presented to the EUC running in Device(s). This step is optimal if it is certain that such locational information can be identified or re-constructed through matching the digital pattern of the encrypted Polymorphic Code Section(s) against the corresponding Unexecutable Code Section(s) by the EUC or PEUC while running. For instance, such identification or re-construction process can rely on some identifiable information that is left within the Unexecutable Code Section(s), such as encryption/decryption key(s), header signature, footer signature, checksum or error-checking signature, or other information left for such purpose; and such identifiable information can be matched with corresponding information within corresponding encrypted Polymorphic Code Section(s) for correct identification of where such encrypted Polymorphic Code Section(s) is/are to be placed for replacing the corresponding Unexecutable Code Section(s) within the EUC or PEUC;

[211] (16) d) converting the encrypted Polymorphic Code Section(s) into unexecutable code, thus creating the Unexecutable Code Section(s), whether in a pattern of all zeros or all ones or in one of the patterns ranging from all zeros to all ones or in a scramble-at-random pattern, and, if and where necessary, superimposing the pattern with such matching information required as in (16) c) such as if the locational information relating to the encrypted Polymorphic Code Section(s) is not available for use while the EUC or the PEUC is running.

[212] The process of distribution, execution and use of the PE, EUC and PEUC is described as follows:

[213] (17) So all these executables including Polymorphic Executables, Executables with Unexecutable Code and Polymorphic Executables with Unexecutable Code, as well as the Polymorphic Code Section(s) and the corresponding locational information, if any, extracted or created separately for replacing the corresponding Unexecutable Code Section(s) within the EUC and PEUC, can be distributed through transmission over local network or internet or through ordinary mails or by other means of transmission or other ways of delivery or distribution for use in Device(s). Other ordinary executables, if any required as in the case for distributing a Polymorphic Operating System, can be also distributed together as well.

[214] (18) As described in (9) and (10) above, when a PE is run up, identifying information or identifier(s) for use as encryption/decryption key(s) is/are be retrieved by the RF in the General Code Section(s) from the encryption/decryption key holder, if

any, within the Polymorphic Code Section and/or from the general-purpose computing or processing environment and/or supplied on run-time by the user using the PE and/or supplied over network, whether local network or internet, by a server or computer or machine or device administering such key(s) for use in encryption/decryption process of the Polymorphic Code Section(s) within the PE for successful execution in Device(s) in different authenticated or authorized states.

[215] (19) When an EUC is run up, it is loaded into memory just like normal code;

[216] (20) Upon execution, the EUC first locates the Unexecutable Code Section when it attempts to call a function therein;

[217] (21) The EUC then obtains, over network, whether local network or internet, from a server or computer or machine or device for such administration, the corresponding Polymorphic Code Section and the corresponding locational information, if any, or it identifies the corresponding matching locational information found within the corresponding pair of Polymorphic Code Section and Unexecutable Code Section, for replacing the Unexecutable Code Section within the EUC;

[218] (22) a) Either the EUC then decrypts the Polymorphic Code Section so obtained, using the corresponding encryption/decyption algorithm(s) and using, as the encryption/decryption key, the identifying information or identifier(s) embedded in the encryption/decryption key holder, if any, within the Polymorphic Code Section and/or the identifying information or identifier(s) supplied, if any, by the programme designer or maker of the Polymorphic Code Section(s), and/or the identifying information or identifier(s) collected from the Device(s) and/or other identifying information or identifier(s) as supplied by the user for the encryption/decryption purpose through all means of transmission or delivery, including transmission over local network or internet or through ordinary mails or by other means of transmission or other ways of delivery or distribution; and then replaces the corresponding Unexecutable Code Section with the Polymorphic Code Section just decrypted, using either the corresponding locational information or the corresponding matching locational information;

[219] (22) b) Or the EUC replaces the corresponding Unexecutable Code Section with the Polymorphic Code Section just obtained, using either the corresponding locational information or the corresponding matching locational information; and then decrypts the undecrypted Polymorphic Code Section now found within the EUC, using the corresponding encryption/decyption algorithm(s) and using, as the encryption/decryption key, the identifying information or identifier(s) embedded in the encryption/decryption key holder, if any, within the Polymorphic Code Section and/or the identifying information or identifier(s) supplied, if any, by the programme designer or maker of the Polymorphic Code Section(s), and/or the identifying information or identifier(s)

collected from the Device(s) and/or other identifying information or identifier(s) as supplied by the user for the encryption/decryption purpose through all means of transmission or delivery, including transmission over local network or internet or through ordinary mails or by other means of transmission or other ways of delivery or distribution;

[220] (23) The EUC now calls the funtion in the Polymorphic Code Section just replaced into the EUC;

[221] (24) If so designed, the function so called by the EUC in (23) calls the corresponding encryption algorithm to re-encrypt the Polymorphic Code Section before it returns. Or the function simply returns without calling the corresponding encryption algorithm for re-encyption. And the EUC continues with the programme logics immediately following the return of the function so called.

[222] (25) When a PEUC is run up, it is loaded into memory just like normal code;

[223] (26) Upon execution, the PEUC first locates the Unexecutable Code Section or Polymorphic Code Section when it attempts to call a function therein;

[224] (27) If the function is found to be in a Polymorphic Code Section, then the PEUC does just what a PE does as described in (10) above;

[225] (28) If the function is found to be in an Unexecutable Code Section, the PEUC then obtains, over network, whether local network or internet, from a server or computer or machine or device for such administration, the corresponding Polymorphic Code Section and the corresponding locational information, if any, or it identifies the corresponding matching locational information found within the corresponding pair of Polymorphic Code Section and Unexecutable Code Section, for replacing the Unexecutable Code Section within the PEUC;

[226] (29) a) Either the PEUC then decrypts the Polymorphic Code Section so obtained, using the corresponding encryption/decyption algorithm(s) and using, as the encryption/decryption key, the identifying information or identifier(s) embedded in the encryption/decryption key holder, if any, within the Polymorphic Code Section and/or the identifying information or identifier(s) supplied, if any, by the programme designer or maker of the Polymorphic Code Section(s), and/or the identifying information or identifier(s) collected from the Device(s) and/or other identifying information or identifier(s) as supplied by the user for the encryption/decryption purpose through all means of transmission or delivery, including transmission over local network or internet or through ordinary mails or by other means of transmission or other ways of delivery or distribution; and then replaces the corresponding Unexecutable Code Section with the Polymorphic Code Section just decrypted, using either the corresponding locational information or the corresponding matching locational information;

[227] (29) b) Or the PEUC replaces the corresponding Unexecutable Code Section with the Polymorphic Code Section just obtained, using either the corresponding locational information or the corresponding matching locational information; and then decrypts the undecrypted Polymorphic Code Section now found within the PEUC, using the corresponding encryption/decyption algorithm(s) and using, as the encryption/ decryption key, the identifying information or identifier(s) embedded in the encryption/decryption key holder, if any, within the Polymorphic Code Section and/or the identifying information or identifier(s) supplied, if any, by the programme designer or maker of the Polymorphic Code Section(s), and/or the identifying information or identifier(s) collected from the Device(s) and/or other identifying information or identifier(s) as supplied by the user for the encryption/decryption purpose through all means of transmission or delivery, including transmission over local network or internet or through ordinary mails or by other means of transmission or other ways of delivery or distribution;

[228] (30) The PEUC now calls the funtion in the Polymorphic Code Section just replaced into the PEUC;

[229] (31) If so designed, the function so called by the PEUC in (30) calls the corresponding encryption algorithm to re-encrypt the Polymorphic Code Section before it returns. Or the function simply returns without calling the corresponding encryption algorithm for re-encyption. And the PEUC continues with the programme logics immediately following the return of the function so called.

[230] In addition to the above description for executing the EUC and PEUC, the EUC or PEUC should have a function of making use of such locational information for successfully replacing the corresponding Unexecutable Code Section(s) with the corresponding Polymorphic Code Section(s) respectively. If such locational information is not saved or not stored or not available to the EUC or PEUC that is running, the EUC or PEUC should have another mechanizm or function(s) of identifying where to place the Polymorphic Code Section(s) so obtained for replacing the corresponding Unexecutable Code Section(s). Such identification process can rely on some identifiable information that is left within the Unexecutable Code Section(s), such as encryption/ decryption key(s), header signature, footer signature, checksum or error-checking signature, or other information left for such purpose; and such identifiable information can be matched with corresponding information within corresponding Polymorphic Code Section(s) so obtained for correct identification of where such Polymorphic Code Section(s) is/are to be placed for replacing the corresponding Unexecutable Code Section(s) within the EUC or PEUC.

[231] Of course, with such matching information left behind, the EUC or PEUC can be considered not as highly secure or protected as the PE for protecting the intellectual

property embedded therein.

[232] (32) If the EUC or PEUC is not in a network environment or only in standalone unconnected Device(s), the Polymorphic Code Section(s) and their locational information, if any, intended for use for replacing the corresponding Unexecutable Code Section(s) within the EUC or PEUC should be made available, for instance such as saved or stored in some accessible fixed or removeable storage medium thus installed or presented to the EUC or PEUC running in Device(s). The running of the EUC or PEUC then continues as in the same way as it is as described above for the Device(s) connected in a network environment as described above.

[233] The process of creating Polymorphic Operating System (POS), an operating system made up of ordinary Executables, PE, EUC and PEUC in different combinations or different degrees of mixture with as least one PE or one EUC or one PEUC as well as the Polymorphic Code Section(s) and the corresponding locational information, if any, so extracted or obtained from the corresponding PE, EUC and PEUC as described above, includes at least one of the following steps:

[234] (33) a) creating executable(s) as PE as described above;

[235] (33) b) creating executable(s) as EUC as described above; and

[236] (33) c) creating executable(s) as PEUC as described above;

[237] The process of executing POS includes at least one of the following steps:

[238] (34) a) executing PE as described above;

[239] (34) b) executing EUC as described above; and

[240] (34) c) executing PEUC as described above.

[241] The process of distributing and using POS is as follows:

[242] (35) So all executables, making up a POS, including, ordinary Executable(s), PE, EUC, PEUC, as well as the Polymorphic Code Section(s) and the corresponding locational information, if any, extracted or obtained from the corresponding PE, EUC and PEUC as described above, can be distributed through transmission over local network or internet or through ordinary mails or by other means of transmission or other ways of delivery or distribution for use in Device(s), facilitating the installation, activation, authentication and starting-up process, for instance, using the small number of installation files, start-up files or crucial kernel files that are put into the PE, EUC and PEUC formats, as well as protecting the intellectual property embedded in the PE, EUC, PEUC and the Polymorphic Operating System as a whole.

Advantageous Effects

[243] Using PE as an example for exposition, this invention differs from the method used by other encryption/decryption shells or wrappers commonly used in protecting executables in the following ways:

[244] (I) Using this invention, there is no visible signature inside the encrypted

executable. In other encryption/decryption shells or wrappers, there is usually some information inside the executable to indicate which encryption/decryption shell or wrapper is used so that a standard method can be devised for removing the encryption/decryption shell or wrapper under concern for all the executables processed by the respective shelling or wrapping.

[245] (II) Using this invention, the selection of section(s) to be used as Polymorphic Code Section(s) for applying encryption/decryption is at source level, and the choice is totally up to the programme designer or maker who can decide which section(s) of code is/are to be encrypted/decrypted. In other encryption/decryption shells or wrappers, the executable intended to be protected is encrypted/decrypted as a whole. It is therefore relatively easier to strip away the encryption/decryption shells or wrappers to obtain the original executable as a whole. However with the use of this invention, as the executable is not to be encrypted/decrypted as a whole and the General Code Section(s) and Polymorphic Code Section(s) in a PE can be designed to be laid out in a zebra-crossing or striped pattern. And each Polymorphic Code Section is not executable by itself. So hacking of a PE and reassembling the individual General Code Section(s) and Polymorphic Code Section(s) together through reverse-engineering is far more difficult than for those executables using the encryption/decryption shells or wrappers. This is so especially given the further enhancements described above in relation to multiple passes of encryption/decryption by using different encryption/decryption algorithms hidden within nested Polymorphic Code Section(s). Therefore, there is no standard way of doing hacking, reverse-engineering, re-constructing and re-assembling for PE designed with the use of this invention.

[246] This invention makes possible the running of Polymorphic Executable and its variants, EUC and PEUC, as well as Polymorphic Operating System(s) as mentioned above in Device(s) for use in a general-purpose computing or processing environment without the need for deploying specially designed apparatuses.

[247] For instance, every PE is different from every other PE running in different authenticated or authorized states. Piracy of intellectual property through simple copying or hacking of executable images stored on storage medium is almost possible. Reverse-engineering and reassembling of the running image of PE in system memory faces great difficulty of identifying where Polymorphic Code Section(s) begin(s) and end(s), breaking nested images of Polymorphic Code Section(s) hidden by multiple passes of different encryption/decryption algorithms and tracing programme logics flows passing through nested Polymorphic Code Sections.

[248] As there is no standard pattern in which General Code Sections and Polymorphic Code Sections are divided, placed, juxtaposed and nested and no standard way in which multiple passes of encryption/decryption of Polymorphic Code Sections are

applied with different encryption/decryption algorithms, the dynamic changing image of the PE in system memory may present tremendous hurdles, if not insurmountable, that hackers have to overcome to reverse-engineer, re-construct and re-assemble an executable running image with all the functionalities of the original programme that can nicely fit together to run successfully. The difficulties multiply if the programme under hacking consists of, not just one, but dozens of PE running simultaneously. If the time spent on hacking, reverse-engineering, re-constructing and re-assembling of the PE is more than the time for the hacker to re-write the programme from scratch, the hacker will give up.

[249] This invention therefore reveals a method providing a flexible and more effective schema for protecting intellectual activities as embodied in executable code. Depending on the level of security and protection for which the intellectual property as embodied in the executable code is required, the programme designer or maker can implement different combinations of the features outlined in this invention, from a PE with only one General Code Section and one Polymorphic Code Section to a PE with multiple General Code Sections and multiple nested Polymorphic Code Sections with multiple passes of encryption/decryption using different hidden encryption/decryption algorithms of choice.

[250] The degree of security and protection is further enhanced if the PE or EUC or PEUC is to be run in a network environment in which the Polymorphic Code Sections can be passed from a server or servers to its client machine, or from one machine to another machine and such Polymorphic Code Sections may have to be encrypted/ decrypted using uncertain encryption/decryption algorithms and unknown encryption/ decryption key(s) which are supplied only when the PE or EUC or PEUC is run. Polymorphic Code Sections not being used in run-time at any moment can be turned into scrambled-at-random Unexecutable Code Sections.

[251] So network distribution of Polymorphic Code Section(s), encryption/decryption key(s) and the PE or EUC or PEUC itself can be achieved. Based on this, the distribution, deployment and activation of Polymorphic Operating Systems is made possible. And as a Polymorphic Operating System does not need to have all its executables in PE or EUC or PEUC format, only some small number of installation files, start-up files or crucial kernel files need to be in these formats to make any operating system securely deployed and activated. So these small number of installation files, start-up files or crucial kernel files can be provided as PE or EUC or PEUC or a combination of these. These small number of installation files, start-up files or crucial kernel files can therefore be easily provided through internet for activating and running the Polymorphic Operating System. Such small number of installation files, start-up files or crucial kernel files can also be saved or stored permanently on the Device in

which the Polymorphic Operating System is intended to run. Such an operating system, or the small number of installation files, start-up files or crucial kernel files so saved or stored as just mentioned, when moved to another Device with the same architecture cannot be run up successfully. And if all the files in the Polymorphic Operating System are designed to be in PE or EUC or PEUC format, then the possibility of having the operating system copied to be used from one Device to another Device is almost non-existent. Incidentally, it is found that the upcoming Microsoft Windows Vista operating system is still not yet a Polymorphic Operating System even though it is claimed to be encrypted with advanced encryption algorithm for running, And if installed on a hard disk, Microsoft Windows Vista can be moved to run from one machine to another machine with the same architecture. And without having to move the hard disk installed, a duplicated hard disk does also work.

Description of Drawings

[252]

Best Mode

[253] The best and the simplest representation of this invention is the creation, distribution and execution in Device(s) of Polymorphic Executable. This provides the most secure or protected method for protecting the intellectual property embedded within the executable and avoids the complexity for retrieving Polymorphic Code Section(s) for replacing the Unexecutable Code Section(s) within EUC or PEUC for execution in Device(s), whether connected to network environment or not.

Mode for Invention

[254] A variation of implementing this method of protection for intellectual property in the form of software programmes is to, instead of creating, distributing and executing PE, create, distribute and execute EUC or PEUC in Device(s), whether connected to network environment or not.

[255] A further variation of implementing this method of protection for intellectual property in the form of software programmes is to create, distribute and executable Polymorphic Operation System(s) as mentioned above in Device(s), whether connected to network environment or not.

Industrial Applicability

[256] This invention therefore helps to protect intellectual property as embodied in executable code. Polymorphic Executable, Executable with Unexecutable Code or the hybrid of these two, Polymorphic Executable with Unexecutable Code, as well as Polymorphic Operating System can be designed, created and distributed, whether through network or not, for use in Device(s) in a general-purpose computing or processing environment under authenticated or authorized state.

[257] Operating systems can be designed with such principles as revealed in this invention in mind that new executable formats can be conceived and standardized. Programming and compilation tools for making executables, such as programming languanges, programming macros, compilers and pre-compilers or pre-processors, etc. can be designed and created to facilitate the making of PE, EUC and PEUC as well as Polymorphic Operating System.

[258] The prior art for the implementation of this invention includes encryption/ decryption algorithms and the knowledge of using such algorithms; progamming languages and compilers for making executable code and operating systems and the related knowledge; the hardware of any device(s), whether networked or standalone, including computer system(s) or computer-controlled device(s) or operating-system-controlled device(s) or system(s), capable of running executable code; and computer-executable or operating-system-executable instructions or programmes that help perform the steps for the method of this invention. In combination with the use of the technical features contained in the prior art stated above, this invention makes possible the creation, distribution and execution in Device(s) of executable code, such as boot code, programmes, applications, device drivers, or a collection of such executables constituting an operating system in the form of executable code embedded or stored into hardware, such as embedded or stored in all types of storage medium, including read-only or rewriteable or volatile or non-volatile storage medium in the form of virtual disk in physical memory or internal DRAM (Dynamic Random Access Memory) or hard disk or solid state flash disk or ROM (Read Only Memory), or read-only or rewriteable CD/DVD/HD-DVD/Blu-Ray DVD or hardware chip or chipset etc.; the executable code being in the form of Polymorphic Executable (PE), Executable with Unexecutable Code (EUC), Polymorphic Executable with Unexecutable Code (PEUC) and Polymorphic Operation System (POS) runnable in an authenticated or authorized state for the protection of intellectual property; and in this relation, is characterized by the following claims:

Sequence List Text

[259]

Claims

[1] A method, capable of being implemented in executable instructions or programmes in device(s), including computer system(s) or computer-controlled device(s) or operating-system-controlled device(s) or system(s), referred hereafter as Device(s), that is/are capable of running executable code, providing for the creation in Device(s) of executable code, such as boot code, programmes, applications, device drivers, or a collection of such executables constituting an operating system, in the form of executable code embedded or stored into hardware, such as embedded or stored in all types of storage medium, including read-only or rewriteable or volatile or non-volatile storage medium, such as in the form of virtual disk in physical memory or internal Dynamic Random Access Memory or hard disk or solid state flash disk or Read Only Memory, or read-only or rewriteable CD/DVD/HD-DVD/Blu-Ray DVD or hardware chip or chipset etc.; the executable code being in the form of Polymorphic Executable (PE) runnable in an authenticated or authorized state for the protection of intellectual property.

The method comprising at least one of the following steps for creating PE:

(1) Select the programming language of choice for writing the PE;

(2) Decide and design feature(s) or function(s) to be included in General Code Section(s) or in Polymorphic Code Section(s);

(3) Design and write General Code Section(s) as normally would be written for normal code which does not require encryption/decryption and, if and where necessary, incorporating changes or revisions resulting from the need for providing for the writing of Polymorphic Code Section(s) according to the rules in (4) a) to (4) d) below;

(4) Design and write Polymorphic Code Section(s) according to the rules as follows:

(4) a) Instead of making memory reference to global variable in Polymorphic Code Section, making memory reference to global variable with wrapper;

(4) b) Instead of making memory reference to static local variable in Polymorphic Code Section, replacing static local variable with global variable, and use (4) a) in making such memory reference;

(4) c) Instead of using static string in Polymorphic Code Section, using static string with wrapper;

(4) d) Instead of using pointers to function or global variable in Polymorphic Code Section, using pointers to function or global variable with wrapper;

(5) Add a Polymorphic Code Section Header function with a Polymorphic Code

Section Header at the beginning of Polymorphic Code Section(s). The header contains at least a header signature holder for holding a header signature;
(6) Add a Polymorphic Code Section Footer function with a Polymorphic Code Section Footer at the end of the Polymorphic Code Section(s). The footer contains at least a footer signature holder for holding a footer signature;
(7) Compile the programme thus designed and written above, and generate the executable;
(8) Use the executable generated in (7) or make a copy of such executable for such purpose, if necessary, and run an encryption programme of choice for encrypting such executable or its copy in the following sequence:
(8) a) Scan the executable for the Polymorphic Code Section Header and Polymorphic Code Section Footer signatures of the Polymorphic Code Section(s) to determine where the encryption of the Polymorphic Code Section(s) begin(s) and end(s);
(8) b) Encrypt the Polymorphic Code Section(s), using the encryption/decyption algorithm(s) of choice and using, as the encryption/decryption key, the identifying information or identifier(s) supplied, if any, by the programme designer or maker of the Polymorphic Code Section(s), and/or the identifying information or identifier(s) collected from the Device(s) and/or other identifying information or identifier(s) as supplied by the user for the encryption/decryption purpose through all means of transmission or delivery, including transmission over local network or internet or through ordinary mails or by other means of transmission or other ways of delivery or distribution.

[2] A method, capable of being implemented in executable instructions or programmes in device(s), including computer system(s) or computer-controlled device(s) or operating-system-controlled device(s) or system(s), referred hereafter as Device(s), that is/are capable of running executable code, providing for the distribution in Device(s) of executable code, such as boot code, programmes, applications, device drivers, or a collection of such executables constituting an operating system, in the form of executable code embedded or stored into hardware, such as embedded or stored in all types of storage medium, including read-only or rewriteable or volatile or non-volatile storage medium, such as in the form of virtual disk in physical memory or internal Dynamic Random Access Memory or hard disk or solid state flash disk or Read Only Memory, or read-only or rewriteable CD/DVD/HD-DVD/Blu-Ray DVD or hardware chip or chipset etc.; the executable code being in the form of Polymorphic Executable (PE) runnable in an authenticated or authorized state for the protection of intellectual property.

The method providing for distributing PE for use in Device(s) comprising at least one of the following steps:

(1) distributing through transmission over local network;

(2) distributing through transmission over internet;

(3) distributing through ordinary mails; and

(4) distributing by other means of transmission or other ways of delivery or distribution.

[3] A method, capable of being implemented in executable instructions or programmes in device(s), including computer system(s) or computer-controlled device(s) or operating-system-controlled device(s) or system(s), referred hereafter as Device(s), that is/are capable of running executable code, providing for the execution in Device(s) of executable code, such as boot code, programmes, applications, device drivers, or a collection of such executables constituting an operating system, in the form of executable code embedded or stored into hardware, such as embedded or stored in all types of storage medium, including read-only or rewriteable or volatile or non-volatile storage medium, such as in the form of virtual disk in physical memory or internal Dynamic Random Access Memory or hard disk or solid state flash disk or Read Only Memory, or read-only or rewriteable CD/DVD/HD-DVD/Blu-Ray DVD or hardware chip or chipset etc.; the executable code being in the form of Polymorphic Executable (PE) runnable in an authenticated or authorized state for the protection of intellectual property.

The method comprising at least one of the following steps for executing PE:

(1) The PE is loaded and the Polymorphic Code Section(s) within is/are loaded into memory just like normal code; and

(2) a) Upon execution, the PE first locates Polymorphic Code Section when it attempts to call a function that is inside a Polymorphic Code Section;

(2) b) The PE then decrypts the Polymorphic Code Section on the first pass. In the later pass(es), the PE checks whether the Polymorphic Code Section has been decrypted. If decrypted, no decryption is necessary. If not decrypted, in these later pass(es), the PE decrypts the Polymorphic Code Section. The PE does such decryption, using the corresponding encryption/decyption algorithm(s) and using, as the encryption/decryption key, the identifying information or identifier(s) embedded in the encryption/decryption key holder, if any, within the Polymorphic Code Section and/or the identifying information or identifier(s) supplied, if any, by the programme designer or maker of the Polymorphic Code Section(s), and/or the identifying information or identifier(s) collected from the Device(s) and/or other identifying information or identifier(s) as supplied by the

user for the encryption/decryption purpose through all means of transmission or delivery, including transmission over local network or internet or through ordinary mails or by other means of transmission or other ways of delivery or distribution;

(2) c) The PE calls the funtion in the Polymorphic Code Section just decrypted in (2) b);

(2) d) If so designed, the function so called by the PE in (2) c) calls the corresponding encryption algorithm to re-encrypt the Polymorphic Code Section before it returns. Or the function simply returns without calling the corresponding encryption algorithm for re-encyption. And the PE continues with the programme logics immediately following the return of the function so called.

[4] A method, capable of being implemented in executable instructions or programmes in device(s), including computer system(s) or computer-controlled device(s) or operating-system-controlled device(s) or system(s), referred hereafter as Device(s), that is/are capable of running executable code, providing for the creation in Device(s) of executable code, such as boot code, programmes, applications, device drivers, or a collection of such executables constituting an operating system, in the form of executable code embedded or stored into hardware, such as embedded or stored in all types of storage medium, including read-only or rewriteable or volatile or non-volatile storage medium, such as in the form of virtual disk in physical memory or internal Dynamic Random Access Memory or hard disk or solid state flash disk or Read Only Memory, or read-only or rewriteable CD/DVD/HD-DVD/Blu-Ray DVD or hardware chip or chipset etc.; the executable code being in the form of Executable with Un-executable Code (EUC) runnable in an authenticated or authorized state for the protection of intellectual property.

The method comprising at least one of the following steps from (1) to (4) below, where all Polymorphic Code Section(s) in a PE is/are converted to Unexecutable Code Section(s):

(1) creating a PE with at least one of the following steps:

(1) a) Select the programming language of choice for writing the PE;

(1) b) Decide and design feature(s) or function(s) to be included in General Code Section(s) or in Polymorphic Code Section(s);

(1) c) Design and write General Code Section(s) as normally would be written for normal code which does not require encryption/decryption and, if and where necessary, incorporating changes or revisions resulting from the need for providing for the writing of Polymorphic Code Section(s) according to the rules in (1) d) i) to (1) d) iv) below;

(1) d) Design and write Polymorphic Code Section(s) according to the rules as follows:

(1) d) i) Instead of making memory reference to global variable in Polymorphic Code Section, making memory reference to global variable with wrapper;

(1) d) ii) Instead of making memory reference to static local variable in Polymorphic Code Section, replacing static local variable with global variable, and use the step described in (1) d) i) in making such memory reference;

(1) d) iii) Instead of using static string in Polymorphic Code Section, using static string with wrapper;

(1) d) iv) Instead of using pointers to function or global variable in Polymorphic Code Section, using pointers to function or global variable with wrapper;

(1) e)Add a Polymorphic Code Section Header function with a Polymorphic Code Section Header at the beginning of Polymorphic Code Section(s). The header contains at least a header signature holder for holding a header signature;

(1) f) Add a Polymorphic Code Section Footer function with a Polymorphic Code Section Footer at the end of the Polymorphic Code Section(s). The footer contains at least a footer signature holder for holding a footer signature;

(1) g) Compile the programme thus designed and written above, and generate the executable;

(1) h) Use the executable generated in (1) g) or make a copy of such executable for such purpose, if necessary, and run an encryption programme of choice for encrypting such executable or its copy in the following sequence:

(1) h) i) Scan the executable for the Polymorphic Code Section Header and Polymorphic Code Section Footer signatures of the Polymorphic Code Section(s) to determine where the encryption of the Polymorphic Code Section(s) should begin and end;

(1) h) ii) Encrypt the Polymorphic Code Section(s), using the encryption/ decyption algorithm(s) of choice and using, as the encryption/decryption key, the identifying information or identifier(s) supplied, if any, by the programme designer or maker of the Polymorphic Code Section(s), and/or the identifying information or identifier(s) collected from the Device(s) and/or other identifying information or identifier(s) as supplied by the user for the encryption/decryption purpose through all means of transmission or delivery, including transmission over local network or internet or through ordinary mails or by other means of transmission or other ways of delivery or distribution;

(2) Extract the encrypted Polymorphic Code Section(s) from the PE as created in (1). And save/store the encrypted Polymorphic Code Section(s) so extracted on storage medium in the server or computer or machine or device administering

such encrypted Polymorphic Code Section(s) for the purpose of delivering or transmitting such encrypted Polymorphic Code Section(s) for use later by the EUC in running Device(s); or save/store these encrypted Polymorphic Code Section(s) in some accessible fixed or removeable storage medium thus installed or presented to the EUC running in Device(s);

(3) Either ascertain and save/store the locational information, about where the encrypted Polymorphic Code Section(s) begin(s) or end(s) in the corresponding PE, on storage medium for use in the server in the server or computer or machine or device administering such locational information or on accessible fixed or removeable storage medium thus installed or presented to the EUC running in Device(s). Or, if necessary, leave some identifiable information within the Un-executable Code Section(s), such as encryption/decryption key(s), header signature, footer signature, checksum or error-checking signature, or other identifiable information for matching with corresponding information within corresponding encrypted Polymorphic Code Section(s) for correct identification of where such encrypted Polymorphic Code Section(s) is/are to be placed for replacing the corresponding Unexecutable Code Section(s) within the EUC;

(4) Convert the encrypted Polymorphic Code Section(s) into unexecutable code, thus creating the Unexecutable Code Section(s), whether in a pattern of all zeros or all ones or in one of the patterns ranging from all zeros to all ones or in a scramble-at-random pattern, and, if and where necessary, superimpose the pattern with such matching information required as in (3) such as if the locational information relating to the encrypted Polymorphic Code Section(s) is not available for use while the EUC is running.

[5] A method, capable of being implemented in executable instructions or programmes in device(s), including computer system(s) or computer-controlled device(s) or operating-system-controlled device(s) or system(s), referred hereafter as Device(s), that is/are capable of running executable code, providing for the distribution in Device(s) of executable code, such as boot code, programmes, applications, device drivers, or a collection of such executables constituting an operating system, in the form of executable code embedded or stored into hardware, such as embedded or stored in all types of storage medium, including read-only or rewriteable or volatile or non-volatile storage medium, such as in the form of virtual disk in physical memory or internal Dynamic Random Access Memory or hard disk or solid state flash disk or Read Only Memory, or read-only or rewriteable CD/DVD/HD-DVD/Blu-Ray DVD or hardware chip or chipset etc.; the executable code being in the form of Executable with Unexecutable Code (EUC) runnable in an authenticated or

authorized state for the protection of intellectual property.

The method providing for distributing EUC and the corresponding Polymorphic Code Section(s) and the corresponding locational information, if any, for use in Device(s) comprising at least one of the following steps:

(1) distributing through transmission over local network;

(2) distributing through transmission over internet;

(3) distributing through ordinary mails; and

(4) distributing by other means of transmission or other ways of delivery or distribution.

[6] A method, capable of being implemented in executable instructions or programmes in device(s), including computer system(s) or computer-controlled device(s) or operating-system-controlled device(s) or system(s), referred hereafter as Device(s), that is/are capable of running executable code, providing for the execution in Device(s) of executable code, such as boot code, programmes, applications, device drivers, or a collection of such executables constituting an operating system, in the form of executable code embedded or stored into hardware, such as embedded or stored in all types of storage medium, including read-only or rewriteable or volatile or non-volatile storage medium, such as in the form of virtual disk in physical memory or internal Dynamic Random Access Memory or hard disk or solid state flash disk or Read Only Memory, or read-only or rewriteable CD/DVD/HD-DVD/Blu-Ray DVD or hardware chip or chipset etc.; the executable code being in the form of Executable with Unexecutable Code (EUC) runnable in an authenticated or authorized state for the protection of intellectual property.

The method comprising at least one of the following steps of (1) to (6):

(1) When an EUC is run up, it is loaded into memory just like normal code;

(2) Upon execution, the EUC first locates the Unexecutable Code Section when it attempts to call a function therein;

(3) a) Either the EUC obtains the corresponding Polymorphic Code Section and the corresponding locational information, if any, either over network, whether local network or internet, from a server or computer or machine or device for such administration, or from accessible fixed or removeable storage medium thus installed or presented to the Device(s) in which the EUC is running, for replacing the Unexecutable Code Section within the EUC;

(3) b) Or the EUC obtains the corresponding Polymorphic Code Section, either over network, whether local network or internet, from a server or computer or machine or device for such administration, or from accessible fixed or removeable storage medium thus installed or presented to the Device(s) in which

the EUC is running, and identifies the corresponding matching locational information, if any and where necessary, found within the corresponding pair of Polymorphic Code Section and Unexecutable Code Section, for such corresponding locational information for replacing the Unexecutable Code Section within the EUC;

(4) a) Either the EUC then decrypts the Polymorphic Code Section so obtained, using the corresponding encryption/decyption algorithm(s) and using, as the encryption/decryption key, the identifying information or identifier(s) embedded in the encryption/decryption key holder, if any, within the Polymorphic Code Section so obtained and/or the identifying information or identifier(s) supplied, if any, by the programme designer or maker of the Polymorphic Code Section(s), and/or the identifying information or identifier(s) collected from the Device(s) and/or other identifying information or identifier(s) as supplied by the user for the encryption/decryption purpose through all means of transmission or delivery, including transmission over local network or internet or through ordinary mails or by other means of transmission or other ways of delivery or distribution; and then replaces the corresponding Unexecutable Code Section with the Polymorphic Code Section just decrypted, using either the corresponding locational information or the corresponding matching locational information;

(4) b) Or the EUC replaces the corresponding Unexecutable Code Section with the Polymorphic Code Section just obtained, using either the corresponding locational information or the corresponding matching locational information; and then decrypts the undecrypted Polymorphic Code Section now found within the EUC, using the corresponding encryption/decyption algorithm(s) and using, as the encryption/decryption key, the identifying information or identifier(s) embedded in the encryption/decryption key holder, if any, within the Polymorphic Code Section and/or the identifying information or identifier(s) supplied, if any, by the programme designer or maker of the Polymorphic Code Section(s), and/or the identifying information or identifier(s) collected from the Device(s) and/or other identifying information or identifier(s) as supplied by the user for the encryption/decryption purpose through all means of transmission or delivery, including transmission over local network or internet or through ordinary mails or by other means of transmission or other ways of delivery or distribution;

(5) The EUC now calls the funtion in the Polymorphic Code Section just replaced into the EUC;

(6) If so designed, the function so called by the EUC in (5) calls the corresponding encryption algorithm to re-encrypt the Polymorphic Code Section

before it returns. Or the function simply returns without calling the corresponding encryption algorithm for re-encyption. And the EUC continues with the programme logics immediately following the return of the function so called.

[7] A method, capable of being implemented in executable instructions or programmes in device(s), including computer system(s) or computer-controlled device(s) or operating-system-controlled device(s) or system(s), referred hereafter as Device(s), that is/are capable of running executable code, providing for the creation in Device(s) of executable code, such as boot code, programmes, applications, device drivers, or a collection of such executables constituting an operating system, in the form of executable code embedded or stored into hardware, such as embedded or stored in all types of storage medium, including read-only or rewriteable or volatile or non-volatile storage medium, such as in the form of virtual disk in physical memory or internal Dynamic Random Access Memory or hard disk or solid state flash disk or Read Only Memory, or read-only or rewriteable CD/DVD/HD-DVD/Blu-Ray DVD or hardware chip or chipset etc.; the executable code being in the form of Polymorphic Executable with Unexecutable Code (PEUC) runnable in an authenticated or authorized state for the protection of intellectual property.

The method comprising at least one of the following steps from (1) to (4) below where according to design only some, but not all, Polymorphic Code Section(s) in a PE is/are converted to Unexecutable Code Section(s):

(1) creating a PE with at least one of the following steps:

(1) a) Select the programming language of choice for writing the PE;

(1) b) Decide and design feature(s) or function(s) to be included in General Code Section(s) or in Polymorphic Code Section(s);

(1) c) Design and write General Code Section(s) as normally would be written for normal code which does not require encryption/decryption and, if and where necessary, incorporating changes or revisions resulting from the need for providing for the writing of Polymorphic Code Section(s) according to the rules in (1) d) i) to (1) d) iv) below;

(1) d) Design and write Polymorphic Code Section(s) according to the rules as follows:

(1) d) i) Instead of making memory reference to global variable in Polymorphic Code Section, making memory reference to global variable with wrapper;

(1) d) ii) Instead of making memory reference to static local variable in Polymorphic Code Section, replacing static local variable with global variable, and use the step described in (1) d) i) in making such memory reference;

(1) d) iii) Instead of using static string in Polymorphic Code Section, using static

string with wrapper;
(1) d) iv) Instead of using pointers to function or global variable in Polymorphic Code Section, using pointers to function or global variable with wrapper;
(1) e)Add a Polymorphic Code Section Header function with a Polymorphic Code Section Header at the beginning of Polymorphic Code Section(s). The header contains at least a header signature holder for holding a header signature;
(1) f) Add a Polymorphic Code Section Footer function with a Polymorphic Code Section Footer at the end of the Polymorphic Code Section(s). The footer contains at least a footer signature holder for holding a footer signature;
(1) g) Compile the programme thus designed and written above, and generate the executable;
(1) h) Use the executable generated in (1) g) or make a copy of such executable for such purpose, if necessary, and run an encryption programme of choice for encrypting such executable or its copy in the following sequence:
(1) h) i) Scan the executable for the Polymorphic Code Section Header and Polymorphic Code Section Footer signatures of the Polymorphic Code Section(s) to determine where the encryption of the Polymorphic Code Section(s) should begin and end;
(1) h) ii) Encrypt the Polymorphic Code Section(s), using the encryption/ decyption algorithm(s) of choice and using, as the encryption/decryption key, the identifying information or identifier(s) supplied, if any, by the programme designer or maker of the Polymorphic Code Section(s), and/or the identifying information or identifier(s) collected from the Device(s) and/or other identifying information or identifier(s) as supplied by the user for the encryption/decryption purpose through all means of transmission or delivery, including transmission over local network or internet or through ordinary mails or by other means of transmission or other ways of delivery or distribution;
(2) Extract the encrypted Polymorphic Code Section(s) from the PE as created in (1). And save/store the encrypted Polymorphic Code Section(s) so extracted on storage medium for use in the server or computer or machine or device administering such encrypted Polymorphic Code Section(s) for the purpose of delivering or transmitting such encrypted Polymorphic Code Section(s) for use later by the PEUC in running Device(s); or save/store these encrypted Polymorphic Code Section(s) in some accessible fixed or removeable storage medium thus installed or presented to the PEUC running in Device(s);
(3) Either ascertain and save/store the locational information, about where the encrypted Polymorphic Code Section(s) begin(s) or end(s) in the corresponding PE, on storage medium for use in the server in the server or computer or machine

or device administering such locational information or on accessible fixed or removeable storage medium thus installed or presented to the PEUC running in Device(s). Or, if necessary, leave some identifiable information within the Unexecutable Code Section(s), such as encryption/decryption key(s), header signature, footer signature, checksum or error-checking signature, or other identifiable information for matching with corresponding information within corresponding encrypted Polymorphic Code Section(s) for correct identification of where such encrypted Polymorphic Code Section(s) is/are to be placed for replacing the corresponding Unexecutable Code Section(s) within the PEUC;
(4) Convert the encrypted Polymorphic Code Section(s) into unexecutable code, thus creating the Unexecutable Code Section(s), whether in a pattern of all zeros or all ones or in one of the patterns ranging from all zeros to all ones or in a scramble-at-random pattern, and, if and where necessary, superimpose the pattern with such matching information required as in (3) such as if the locational information relating to the encrypted Polymorphic Code Section(s) is not available for use while the PEUC is running.

[8] A method, capable of being implemented in executable instructions or programmes in device(s), including computer system(s) or computer-controlled device(s) or operating-system-controlled device(s) or system(s), referred hereafter as Device(s), that is/are capable of running executable code, providing for the distribution in Device(s) of executable code, such as boot code, programmes, applications, device drivers, or a collection of such executables constituting an operating system, in the form of executable code embedded or stored into hardware, such as embedded or stored in all types of storage medium, including read-only or rewriteable or volatile or non-volatile storage medium, such as in the form of virtual disk in physical memory or internal Dynamic Random Access Memory or hard disk or solid state flash disk or Read Only Memory, or read-only or rewriteable CD/DVD/HD-DVD/Blu-Ray DVD or hardware chip or chipset etc.; the executable code being in the form of Polymorphic Executable with Unexecutable Code (PEUC) runnable in an authenticated or authorized state for the protection of intellectual property.
The method providing for distributing PEUC and the corresponding Polymorphic Code Section(s) and the corresponding locational information, if any, for use in Device(s) comprising at least one of the following steps:
(1) distributing through transmission over local network;
(2) distributing through transmission over internet;
(3) distributing through ordinary mails; and
(4) distributing by other means of transmission or other ways of delivery or dis-

tribution.

[9] A method, capable of being implemented in executable instructions or programmes in device(s), including computer system(s) or computer-controlled device(s) or operating-system-controlled device(s) or system(s), referred hereafter as Device(s), that is/are capable of running executable code, providing for the execution in Device(s) of executable code, such as boot code, programmes, applications, device drivers, or a collection of such executables constituting an operating system, in the form of executable code embedded or stored into hardware, such as embedded or stored in all types of storage medium, including read-only or rewriteable or volatile or non-volatile storage medium, such as in the form of virtual disk in physical memory or internal Dynamic Random Access Memory or hard disk or solid state flash disk or Read Only Memory, or read-only or rewriteable CD/DVD/HD-DVD/Blu-Ray DVD or hardware chip or chipset etc.; the executable code being in the form of Polymorphic Executable with Unexecutable Code (PEUC) runnable in an authenticated or authorized state for the protection of intellectual property.
The method comprising at least one of the following steps of (1) to (3):
(1) When a PEUC is run up, it is loaded into memory just like normal code;
(2) Upon execution, the PEUC first locates the Unexecutable Code Section or Polymorphic Code Section when it attempts to call a function therein;
(3) (a) If the function is found to be in a Polymorphic Code Section, then the PEUC does just what a PE does as follows, using at least one of the following steps in (3) a) i) to (3) a) iii) for executing:
(3) a) i) The PEUC then decrypts the Polymorphic Code Section on the first pass and checks whether the Polymorphic Code Section has been decrypted before doing decryption again in the later pass(es), using the corresponding encryption/decyption algorithm(s) and using, as the encryption/decryption key, the identifying information or identifier(s) embedded in the encryption/decryption key holder, if any, within the Polymorphic Code Section and/or the identifying information or identifier(s) supplied, if any, by the programme designer or maker of the Polymorphic Code Section(s), and/or the identifying information or identifier(s) collected from the Device(s) and/or other identifying information or identifier(s) as supplied by the user for the encryption/decryption purpose through all means of transmission or delivery, including transmission over local network or internet or through ordinary mails or by other means of transmission or other ways of delivery or distribution;
(3) a) ii) The PEUC calls the funtion in the Polymorphic Code Section just decrypted in (3) a) i);

(3) a) iii) If so designed, the function so called by the PEUC in (3) a) ii) calls the corresponding encryption algorithm to re-encrypt the Polymorphic Code Section before it returns. Or the function simply returns without calling the corresponding encryption algorithm for re-encyption. And the PEUC continues with the programme logics immediately following the return of the function so called.
(3) b) If the function is found to be in an Unexecutable Code Section, the PEUC then does at least one of the following steps in (3) b) i) to (3) b) iv):
(3) b) i) 1) Either the PEUC obtains the corresponding Polymorphic Code Section and the corresponding locational information, if any, either over network, whether local network or internet, from a server or computer or machine or device for such administration, or from accessible fixed or removeable storage medium thus installed or presented to the Device(s) in which the PEUC is running, for replacing the Unexecutable Code Section within the PEUC;
(3) b) i) 2) Or the PEUC obtains the corresponding Polymorphic Code Section, either over network, whether local network or internet, from a server or computer or machine or device for such administration, or from accessible fixed or removeable storage medium thus installed or presented to the Device(s) in which the PEUC is running, and identifies the corresponding matching locational information, if any and where necessary, found within the corresponding pair of Polymorphic Code Section and Unexecutable Code Section, for such corresponding locational information for replacing the Unexecutable Code Section within the PEUC;
(3) b) ii) 1) Either the PEUC then decrypts the Polymorphic Code Section so obtained, using the corresponding encryption/decyption algorithm(s) and using, as the encryption/decryption key, the identifying information or identifier(s) embedded in the encryption/decryption key holder, if any, within the Polymorphic Code Section and/or the identifying information or identifier(s) supplied, if any, by the programme designer or maker of the Polymorphic Code Section(s), and/or the identifying information or identifier(s) collected from the Device(s) and/or other identifying information or identifier(s) as supplied by the user for the encryption/decryption purpose through all means of transmission or delivery, including transmission over local network or internet or through ordinary mails or by other means of transmission or other ways of delivery or distribution; and then replaces the corresponding Unexecutable Code Section with the Polymorphic Code Section just decrypted, using either the corresponding locational information or the corresponding matching locational information;

(3) b) ii) 2) Or the PEUC replaces the corresponding Unexecutable Code Section with the Polymorphic Code Section just obtained, using either the corresponding locational information or the corresponding matching locational information; and then decrypts the undecrypted Polymorphic Code Section now found within the PEUC, using the corresponding encryption/decyption algorithm(s) and using, as the encryption/decryption key, the identifying information or identifier(s) embedded in the encryption/decryption key holder, if any, within the Polymorphic Code Section and/or the identifying information or identifier(s) supplied, if any, by the programme designer or maker of the Polymorphic Code Section(s), and/or the identifying information or identifier(s) collected from the Device(s) and/or other identifying information or identifier(s) as supplied by the user for the encryption/decryption purpose through all means of transmission or delivery, including transmission over local network or internet or through ordinary mails or by other means of transmission or other ways of delivery or distribution;

(3) b) iii) The PEUC now calls the funtion in the Polymorphic Code Section just replaced into the PEUC;

(3) b) iv) If so designed, the function so called by the PEUC in (3) b) iii) calls the corresponding encryption algorithm to re-encrypt the Polymorphic Code Section before it returns. Or the function simply returns without calling the corresponding encryption algorithm for re-encyption. And the PEUC continues with the programme logics immediately following the return of the function so called.

[10] A method, capable of being implemented in executable instructions or programmes in device(s), including computer system(s) or computer-controlled device(s) or operating-system-controlled device(s) or system(s), referred hereafter as Device(s), that is/are capable of running executable code, providing for the creation in Device(s) of executable code, making up a collection of such executables constituting an operating system, in the form of executable code embedded or stored into hardware, such as embedded or stored in all types of storage medium, including read-only or rewriteable or volatile or non-volatile storage medium, such as in the form of virtual disk in physical memory or internal Dynamic Random Access Memory or hard disk or solid state flash disk or Read Only Memory, or read-only or rewriteable CD/DVD/HD-DVD/Blu-Ray DVD or hardware chip or chipset etc.; the executable code being in the form of Polymorphic Operation System (POS), an operating system made up of ordinary Executables, PE, EUC and PEUC in different combinations or different degrees of mixture with as least one PE or one EUC or one PEUC as well as the Polymorphic Code Section(s) and the corresponding locational information, if

any, so extracted or obtained from the corresponding PE, EUC and PEUC, runnable in an authenticated or authorized state for the protection of intellectual property.

The method comprising at least one of the following steps:

(1) creating executable(s) as PE as in Claim (1);

(2) creating executable(s) as EUC as in Claim (4); and

(3) creating executable(s) as PEUC as in Claim (7).

[11] A method, capable of being implemented in executable instructions or programmes in device(s), including computer system(s) or computer-controlled device(s) or operating-system-controlled device(s) or system(s), referred hereafter as Device(s), that is/are capable of running executable code, providing for the distribution in Device(s) of executable code, making up a collection of such executables constituting an operating system, in the form of executable code embedded or stored into hardware, such as embedded or stored in all types of storage medium, including read-only or rewriteable or volatile or non-volatile storage medium, such as in the form of virtual disk in physical memory or internal Dynamic Random Access Memory or hard disk or solid state flash disk or Read Only Memory, or read-only or rewriteable CD/DVD/HD-DVD/Blu-Ray DVD or hardware chip or chipset etc.; the executable code being in the form of Polymorphic Operation System (POS), an operating system made up of ordinary Executables, PE, EUC and PEUC in different combinations or different degrees of mixture with as least one PE or one EUC or one PEUC as well as the Polymorphic Code Section(s) and the corresponding locational information, if any, so extracted from the corresponding PE, EUC and PEUC, runnable in an authenticated or authorized state for the protection of intellectual property.

The method providing for distributing POS comprising at least one of the following steps:

(1) distributing through transmission over local network;

(2) distributing through transmission over internet;

(3) distributing through ordinary mails; and

(4) distributing by other means of transmission or other ways of delivery or distribution.

[12] A method, capable of being implemented in executable instructions or programmes in device(s), including computer system(s) or computer-controlled device(s) or operating-system-controlled device(s) or system(s), referred hereafter as Device(s), that is/are capable of running executable code, providing for the execution in Device(s) of executable code, making up a collection of such executables constituting an operating system, in the form of executable code

embedded or stored into hardware, such as embedded or stored in all types of storage medium, including read-only or rewriteable or volatile or non-volatile storage medium, such as in the form of virtual disk in physical memory or internal Dynamic Random Access Memory or hard disk or solid state flash disk or Read Only Memory, or read-only or rewriteable CD/DVD/HD-DVD/Blu-Ray DVD or hardware chip or chipset etc.; the executable code being in the form of Polymorphic Operation System (POS), an operating system made up of ordinary Executables, PE, EUC and PEUC in different combinations or different degrees of mixture with as least one PE or one EUC or one PEUC as well as the Polymorphic Code Section(s) and the corresponding locational information, if any, so extracted from the corresponding PE, EUC and PEUC, runnable in an authenticated or authorized state for the protection of intellectual property.

The method comprising at least one of the following steps:

(1) executing PE as in Claim (3);

(2) executing EUC as in Claim (6); and

(3) executing PEUC as in Claim (9).

[13] The product, a Polymorphic Executable, created and embedded or stored into hardware, such as embedded or stored in all types of storage medium, including read-only or rewriteable or volatile or non-volatile storage medium, such as in the form of virtual disk in physical memory or internal Dynamic Random Access Memory or hard disk or solid state flash disk or Read Only Memory, or read-only or rewriteable CD/DVD/HD-DVD/Blu-Ray DVD or hardware chip or chipset etc., using Claim (1).

[14] The Use in device(s), including computer system(s) or computer-controlled device(s) or operating-system-controlled device(s) or system(s), that is/are capable of running executable code, of the product, a Polymorphic Executable, created using Claim (1).

[15] Device(s), including computer system(s) or computer-controlled device(s) or operating-system-controlled device(s) or system(s), that is/are capable of running executable code, using the product, a Polymorphic Executable, created using Claim (1).

[16] The product, an Executable with Unexecutable Code, created and embedded or stored into hardware, such as embedded or stored in all types of storage medium, including read-only or rewriteable or volatile or non-volatile storage medium, such as in the form of virtual disk in physical memory or internal Dynamic Random Access Memory or hard disk or solid state flash disk or Read Only Memory, or read-only or rewriteable CD/DVD/HD-DVD/Blu-Ray DVD or hardware chip or chipset etc., using Claim (4).

[17] The Use in device(s), including computer system(s) or computer-controlled device(s) or operating-system-controlled device(s) or system(s), that is/are capable of running executable code, of the product, an Executable with Un-executable Code, created using Claim (4).

[18] Device(s), including computer system(s) or computer-controlled device(s) or operating-system-controlled device(s) or system(s), that is/are capable of running executable code, using the product, an Executable with Unexecutable Code, created using Claim (4).

[19] The product, a Polymorphic Executable with Unexecutable Code, created and embedded or stored into hardware, such as embedded or stored in all types of storage medium, including read-only or rewriteable or volatile or non-volatile storage medium, such as in the form of virtual disk in physical memory or internal Dynamic Random Access Memory or hard disk or solid state flash disk or Read Only Memory, or read-only or rewriteable CD/DVD/HD-DVD/Blu-Ray DVD or hardware chip or chipset etc., using Claim (7).

[20] The Use in device(s), including computer system(s) or computer-controlled device(s) or operating-system-controlled device(s) or system(s), that is/are capable of running executable code, of the product, a Polymorphic Executable with Unexecutable Code, created using Claim (7).

[21] Device(s), including computer system(s) or computer-controlled device(s) or operating-system-controlled device(s) or system(s), that is/are capable of running executable code, using the product, a Polymorphic Executable with Un-executable Code, created using Claim (7).

[22] The product, a Polymorphic Operating System, created and embedded or stored into hardware, such as embedded or stored in all types of storage medium, including read-only or rewriteable or volatile or non-volatile storage medium, such as in the form of virtual disk in physical memory or internal Dynamic Random Access Memory or hard disk or solid state flash disk or Read Only Memory, or read-only or rewriteable CD/DVD/HD-DVD/Blu-Ray DVD or hardware chip or chipset etc., using at least one of the following steps:
(1) creating executable(s) as PE as in Claim (1);
(2) creating executable(s) as EUC as in Claim (4); and
(3) creating executable(s) as PEUC as in Claim (7).

[23] The Use in device(s), including computer system(s) or computer-controlled device(s) or operating-system-controlled device(s) or system(s), that is/are capable of running executable code, of the product, a Polymorphic Operating System, created using at least one of the following steps:
(1) creating executable(s) as PE as in Claim (1);

(2) creating executable(s) as EUC as in Claim (4); and
(3) creating executable(s) as PEUC as in Claim (7).

[24] Device(s), including computer system(s) or computer-controlled device(s) or operating-system-controlled device(s) or system(s), that is/are capable of running executable code, using the product, a Polymorphic Operating System, created using at least one of the following steps:
(1) creating executable(s) as PE as in Claim (1);
(2) creating executable(s) as EUC as in Claim (4); and
(3) creating executable(s) as PEUC as in Claim (7).

[25] Programming and/or compilation tools converting source code of a programme according to at least one of the following rules for making executable code in Polymorphic Code Section(s) of executable(s) :
(1) Converting source code making memory reference to global variable in Polymorphic Code Section to making memory reference to global variable with wrapper;
(2) Converting source code making memory reference to static local variable in Polymorphic Code Section by replacing static local variable with global variable, and using the step described in (1) above in making such memory reference;
(3) Converting source code using static string in Polymorphic Code Section to using static string with wrapper; and
(4) Converting using pointers to function or global variable in Polymorphic Code Section to using pointers to function or global variable with wrapper.

[26] The method of applying the technique of nesting Polymorphic Code Section(s) for the creation of any executable(s) in Polymorphic Executable format or in Executable with Unexecutable Code format or in Polymorphic Executable with Unexecutable Code format, in the form of executable code embedded or stored into hardware, such as embedded or stored in all types of storage medium, including read-only or rewriteable or volatile or non-volatile storage medium, such as in the form of virtual disk in physical memory or internal Dynamic Random Access Memory or hard disk or solid state flash disk or Read Only Memory, or read-only or rewriteable CD/DVD/HD-DVD/Blu-Ray DVD or hardware chip or chipset etc.; this method being capable of being implemented in executable instructions or programmes in device(s), including computer system(s) or computer-controlled device(s) or operating-system-controlled device(s) or system(s), that is/are capable of running executable code.

[27] The method of applying the technique of embedding hidden encryption/ decryption algorithm(s) within nested Polymorphic Code Section(s) for the creation of any executable(s) in Polymorphic Executable format or in Executable

with Unexecutable Code format or in Polymorphic Executable with Un-executable Code format, in the form of executable code embedded or stored into hardware, such as embedded or stored in all types of storage medium, including read-only or rewriteable or volatile or non-volatile storage medium, such as in the form of virtual disk in physical memory or internal Dynamic Random Access Memory or hard disk or solid state flash disk or Read Only Memory, or read-only or rewriteable CD/DVD/HD-DVD/Blu-Ray DVD or hardware chip or chipset etc.; this method being capable of being implemented in executable instructions or programmes in device(s), including computer system(s) or computer-controlled device(s) or operating-system-controlled device(s) or system(s), that is/are capable of running executable code.

[28] The method of applying the technique of implementing multiple passes of encryption/decryption for the creation of any executable(s) in Polymorphic Executable format or in Executable with Unexecutable Code format or in Polymorphic Executable with Unexecutable Code format, in the form of executable code embedded or stored into hardware, such as embedded or stored in all types of storage medium, including read-only or rewriteable or volatile or non-volatile storage medium, such as in the form of virtual disk in physical memory or internal Dynamic Random Access Memory or hard disk or solid state flash disk or Read Only Memory, or read-only or rewriteable CD/DVD/HD-DVD/Blu-Ray DVD or hardware chip or chipset etc.; this method being capable of being implemented in executable instructions or programmes in device(s), including computer system(s) or computer-controlled device(s) or operating-system-controlled device(s) or system(s), that is/are capable of running executable code.

[29] The method of applying the technique of implementing encryption/decryption with multiple encryption/decryption alogrithms for the creation of any executable(s) in Polymorphic Executable format or in Executable with Un-executable Code format or in Polymorphic Executable with Unexecutable Code format, in the form of executable code embedded or stored into hardware, such as embedded or stored in all types of storage medium, including read-only or rewriteable or volatile or non-volatile storage medium, such as in the form of virtual disk in physical memory or internal Dynamic Random Access Memory or hard disk or solid state flash disk or Read Only Memory, or read-only or rewriteable CD/DVD/HD-DVD/Blu-Ray DVD or hardware chip or chipset etc.; this method being capable of being implemented in executable instructions or programmes in device(s), including computer system(s) or computer-controlled device(s) or operating-system-controlled device(s) or system(s), that is/are

capable of running executable code.

[30] The method of applying the technique of implementing dynamic single pass and/or multiple passes of encryption/decryption with or without the use of stacking, that is encryption/decryption and re-encryption/re-decryption of the same Polymorphic Code Section(s) while the executable is being executed, for the creation of any executable(s) in Polymorphic Executable format or in Executable with Unexecutable Code format or in Polymorphic Executable with Un-executable Code format, in the form of executable code embedded or stored into hardware, such as embedded or stored in all types of storage medium, including read-only or rewriteable or volatile or non-volatile storage medium, such as in the form of virtual disk in physical memory or internal Dynamic Random Access Memory or hard disk or solid state flash disk or Read Only Memory, or read-only or rewriteable CD/DVD/HD-DVD/Blu-Ray DVD or hardware chip or chipset etc.; this method being capable of being implemented in executable in-structions or programmes in device(s), including computer system(s) or computer-controlled device(s) or operating-system-controlled device(s) or system(s), that is/are capable of running executable code.

DECLARATION OF NON-ESTABLISHMENT OF INTERNATIONAL SEARCH REPORT

(PCT Article 17(2)(a),Rules 13 fer.l(c) and 39)

Applicant's agent's reference PCTPOLY	IMPORTANT DECLARATION	Date filing (day/month/year) 12 APR 2007 (12.04.2007)
International application No. PCT/IB2006/053395	International filing date (day/month/year) 20 Sep. 2006 (20. 09. 2006)	(Earliest)Priority date(#a>/7noB//!/year,)
International Patent Classification (IPC) or both national classification and IPC G06F21/00 (2006. 01) i		
Applicant KAM-FU, CHAN et al		

This International Searching Authority hereby declares, according to Article 17(2)(a), that no international search report will be established on the international application for the reasons indicated below.

1. ☒ The subject matter of the international application relates to:
   - a. D scientific theories
   - b. □ mathematical theories
   - c. D plant varieties
   - d. □ animal varieties
   - e. ☐ essentially biological processes for the production of plants and animals, other than microbiological processes and the products of such processes
   - f. D schemes, rules or methods of doing business
   - g. ☒ schemes, rules or methods of performing purely mental acts
   - h. ☐ schemes, rules or methods of playing games
   - i. ☐ methods for treatment of the human body by surgery or therapy
   - j. ☐ methods for treatment of the animal body by surgery or therapy
   - k. ☐ diagnostic methods practised on the human or animal body
   - 1. D mere presentations of information
   - m. ☐ computer programs for which this International Searching Authority is not equipped to search prior art

2. [U The failure of the following parts of the international application to comply with prescribed requirements prevents a meaningful search from being carried out:
   [U the description ☐ the claims HH the drawings

3. ☐ A meaningful search could not be carried out without the sequence listing; the applicant did not, within the prescribed time limit:
   - ☐ furnish a sequence listing on paper complying with the standard provided for in Annex C of the Administrative Instructions, and such listing was not available to the International Searching Authority in a form and manner acceptable to it.
   - [U furnish a sequence listing in electronic form complying with the standard provided for in Annex C of the Administrative Instructions, and such listing was not available to the International Searching Authority in a form and manner acceptable to it.
   - [U pay the required late furnishing fee for the furnishing of a sequence listing in response to an invitation under Rule 13ter.l(a) or (b).

4. [H A meaningful search could not be carried out without the tables related to the sequence listings; the applicant did not, within the prescribed time limit, furnish such tables in electronic form complying with the technical requirements provided for in Annex C-bis of the Administrative Instructions, and such tables were not available to the International Searching Authority in a form and manner acceptable to it.

5. Further comments: The applicant suggests in the whole application that when protecting intellectual property, selects programming language, decides, designs, writes and implements the program code by the programmer's physically or mentally changing the original data types and the running environment of the executable code. Although there are some steps like encrypting or decrypting, but the running of these kinds of steps are choose and used by the programmer's will. Thus the solutions of the present invention are based on the mental acts of the programmer who writes the program code. So it seems that the whole application is a scheme, a rule or a method oi performing purely mental acts.

Name and mailing address of the ISA/CN	Authorized officer
The State Intellectual Property Office, the P.R.China 6 Xitucheng Rd., Jimen Bridge, Haidian District, Beijing, China 100D88 Facsimile No. 86- 10-620 1945 1	ZHANQJjYjtanflfJ Telephone No. 86-10-620849028

Form PCT/ISA/203(April.2005)