(19)中华人民共和国国家知识产权局

(12)发明专利申请

(10)申请公布号 CN 105095698 A
(43)申请公布日 2015.11.25

(21)申请号 201510245908.3

(22)申请日 2015.05.14

(30)优先权数据
14/281,232 2014.05.19 US

(71)申请人 恩智浦有限公司
地址 荷兰艾恩德霍芬

(72)发明人 简·胡格布鲁格
W·P·A·J·米歇尔斯

(74)专利代理机构 中科专利商标代理有限责任公司 11021
代理人 王波波

(51)Int.Cl.
G06F 21/14(2013.01)

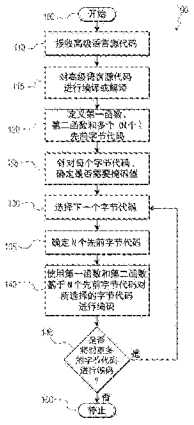
权利要求书2页 说明书8页 附图1页

(54)发明名称

基于最近执行的程序代码的程序代码模糊处理

(57)摘要

一种对包括多个指令的软件代码进行掩盖的方法,包括:由处理器确定针对当前指令的N个先前指令;基于第一函数、第二函数和所述N个先前指令对当前指令进行编码,其中所述第二函数基于所述N个先前指令,并且其中所述第一函数基于当前指令和所述第二函数的输出。



CN 105095698 A

1. 一种对包括多个指令的软件代码进行掩盖的方法，包括：

由处理器确定针对当前指令的 N 个先前指令；

基于第一函数、第二函数和所述 N 个先前指令对所述当前指令进行编码，

其中所述第二函数基于所述 N 个先前指令，以及

所述第一函数基于所述当前指令和所述第二函数的输出。

2. 根据权利要求 1 所述的方法，还包括：

确定在所述 N 个先前指令或所述当前指令中存在连接点；

当存在连接点时，针对所述连接点之前的每个先前指令产生掩码值，其中所述掩码是基于所述连接点之前的并行指令中的公共比特设置的；以及

将所述掩码与所述当前指令相关联。

3. 根据权利要求 1 所述的方法，还包括：

确定在所述 N 个先前指令或所述当前指令中存在连接点；

当存在连接点时，在所述连接点之前插入 I 个 NOOP 指令，其中 I 是所述连接点之前的先前指令的数量。

4. 根据权利要求 1 所述的方法，还包括：

确定在所述 N 个先前指令或所述当前指令中存在连接点；

当存在连接点时，将所述连接点下移 I 个指令，其中 I 是所述连接点之前的先前指令的数量。

5. 根据权利要求 1 所述的方法，还包括：

接收采用更高级语言的软件代码；以及

将所接收的软件代码编译成机器可执行指令。

6. 根据权利要求 1 所述的方法，还包括：

接收采用更高级语言的软件代码；以及

将所接收的软件代码解译成机器可执行指令。

7. 根据权利要求 1 所述的方法，还包括：产生密钥，其中所述密钥与针对第一当前指令的 N 个先前指令相对应。

8. 根据权利要求 1 所述的方法，其中，所述 N 个先前指令是 N 个先前经编码的指令。

9. 根据权利要求 1 所述的方法，还包括：

由处理器确定与当前指令之前的所述 N 个先前指令相关联的 M 个先前数据值；以及

其中对所述当前指令进行编码还基于所述 M 个数据值，其中所述第二函数还基于所述 M 个先前数据值。

10. 根据权利要求 9 所述的方法，还包括：产生密钥，其中所述密钥与针对第一当前指令的 N 个先前指令和 M 个先前数据值相对应。

11. 一种对包括多个经编码的指令的经掩盖的软件代码进行解码的方法，包括：

由处理器确定针对当前经编码的指令的 N 个先前经编码的指令；

基于第一函数、第二函数和所述 N 个先前经解码的指令对所述当前经编码的指令进行解码，

其中所述第二函数基于所述 N 个先前经解码的指令，以及

所述第一函数基于所述当前经编码的指令和所述第二函数的输出。

12. 根据权利要求 11 所述的方法，还包括：接收密钥，其中所述密钥与针对第一经编码的当前指令的 N 个先前指令相对应。

基于最近执行的程序代码的程序代码模糊处理

技术领域

[0001] 这里公开的各种示例性实施例一般地涉及基于最近执行的程序代码的程序代码模糊处理。

背景技术

[0002] 如今，软件应用广泛地用来向用户提供多种服务。这些软件应用可以寄宿在各种不同的设备上，比如移动电话、个人计算机、膝上型计算机、平板电脑、机顶盒等。可在许多由消费者使用的系统中或工业系统中找到软件应用。还可在智能卡和信用卡中找到软件应用。此外，软件应用还可实现在网络上，比如互联网，其中软件应用运行于服务器上，并且使用多种用户设备访问软件应用。许多这种软件应用需要使用安全协议来保护内容、信息、交易和隐私。许多软件应用运行于这样的环境中，即，攻击者具有对软件应用的操作的完全或一些控制，并且攻击者可以尝试对软件应用的代码进行反向工程以便获得对安全信息的访问，或者尝试甚至理解软件的操作以便重现或修改软件应用的功能。攻击者可使用多种反向工程工具，比如代码分析器和调试器，以获得与软件应用有关的信息。因此，已经开发了技术来使得攻击者难以对软件进行反向工程。一种用来使得对代码进行反向工程更加困难的方式是代码模糊处理。代码模糊处理试图创建经模糊处理的代码，该经模糊处理的代码是人类难以理解的。代码模糊处理可用来隐匿软件应用的目的或其逻辑，以防止对软件应用进行篡改或反向工程。

发明内容

[0003] 以下给出对多种示例性实施例的简要概述。可在以下概述中进行一些简化和省略，这是为了强调和介绍多个示例性实施例的某些方面，而不是为了限制本发明的范围。在随后的部分中将对足以使得本领域普通技术人员作出和使用创造性构思的示例性实施例进行具体描述。

[0004] 多种示例性实施例涉及一种对包括多个指令的软件代码进行掩盖的方法，包括：由处理器确定针对当前指令的N个先前指令；基于第一函数、第二函数和所述N个先前指令对当前指令进行编码，其中所述第二函数基于所述N个先前指令，并且其中所述第一函数基于当前指令和所述第二函数的输出。

[0005] 描述了多种实施例，还包括：确定在所述N个先前指令或当前指令中存在连接点；当存在连接点时，针对所述连接点之前的每个先前指令产生掩码值，其中所述掩码是基于所述连接点之前的并行指令中的公共比特设置的；以及将所述掩码与当前指令相关联。

[0006] 描述了多种实施例，还包括：确定在所述N个先前指令或当前指令中存在连接点；当存在连接点时，在所述连接点之前插入I个NOOP指令，其中I是所述连接点之前的先前指令的数量。

[0007] 描述了多种实施例，还包括：确定在所述N个先前指令或当前指令中存在连接点；当存在连接点时，将所述连接点下移I个指令，其中I是所述连接点之前的先前指令的数

量。

[0008] 描述了多种实施例，还包括：接收采用更高级语言的软件代码；以及将所接收的软件代码编译成机器可执行指令。

[0009] 描述了多种实施例，还包括：接收采用更高级语言的软件代码；以及将所接收的软件代码解译成机器可执行指令。

[0010] 描述了多种实施例，还包括：产生密钥，其中所述密钥与针对第一当前指令的N个先前指令相对应。

[0011] 描述了多种实施例，其中所述N个先前指令是N个先前经编码的指令。

[0012] 描述了多种实施例，还包括：由处理器确定与在当前指令之前的所述N个先前指令相关联的M个先前数据值；以及其中对当前指令进行编码还基于所述M个数据值，其中所述第二函数还基于所述M个先前数据值。

[0013] 描述了多种实施例，还包括：产生密钥，其中所述密钥与针对第一当前指令的N个先前指令和M个先前数据值相对应。

[0014] 此外，多种示例性实施例涉及一种对包括多个经编码的指令的经掩盖的软件代码进行解码的方法，包括：由处理器确定针对当前经编码的指令的N个先前经编码的指令；基于第一函数、第二函数和所述N个先前经解码的指令对当前经编码的指令进行解码，其中所述第二函数基于所述N个先前经解码的指令，并且其中所述第一函数基于当前经编码的指令和所述第二函数的输出。

[0015] 描述了多种实施例，还包括：接收密钥，其中所述密钥与针对第一经编码的当前指令的N个先前指令相对应。

[0016] 此外，多种示例性实施例涉及编码有用于由处理器执行的指令的非瞬时机器可读存储介质，所述指令用于对包括多个指令的软件代码进行掩盖，所述非瞬时机器可读存储介质包括：用于确定针对当前指令的N个先前指令的指令；用于基于第一函数、第二函数和所述N个先前指令对当前指令进行编码的指令，其中所述第二函数基于所述N个先前指令，并且其中所述第一函数基于当前指令和所述第二函数的输出。

[0017] 描述了多种实施例，还包括：用于确定在所述N个先前指令或当前指令中存在连接点的指令；用于当存在连接点时针对所述连接点之前的每个先前指令产生掩码值的指令，其中所述掩码是基于所述连接点之前的并行指令中的公共比特设置的；以及用于将所述掩码与当前指令相关联的指令。

[0018] 描述了多种实施例，还包括：用于确定在所述N个先前指令或当前指令中存在连接点的指令；用于当存在连接点时在所述连接点之前插入I个NOOP指令的指令，其中I是所述连接点之前的先前指令的数量。

[0019] 描述了多种实施例，还包括：用于确定在所述N个先前指令或当前指令中存在连接点的指令；用于当存在连接点时将所述连接点下移I个指令的指令，其中I是所述连接点之前的先前指令的数量。

[0020] 描述了多种实施例，还包括：用于接收采用更高级语言的软件代码的指令；以及用于将所接收的软件代码编译成机器可执行指令的指令。

[0021] 描述了多种实施例，还包括：用于接收采用更高级语言的软件代码的指令；以及用于将所接收的软件代码解译成机器可执行指令的指令。

[0022] 描述了多种实施例，还包括：用于产生密钥的指令，其中所述密钥与针对第一当前指令的 N 个先前指令相对应。

[0023] 描述了多种实施例，其中所述 N 个先前指令是 N 个先前经编码的指令。

[0024] 描述了多种实施例，还包括：用于由处理器确定与当前指令之前的所述 N 个先前指令相关联的 M 个先前数据值的指令；以及其中对当前指令进行编码还基于所述 M 个数据值，其中所述第二函数还基于所述 M 个先前数据值。

[0025] 描述了多种实施例，还包括：用于产生密钥的指令，其中所述密钥与针对第一当前指令的 N 个先前指令和 M 个先前数据值相对应。

[0026] 此外，多种示例性实施例涉及编码有用于由处理器执行的指令的非瞬时机器可读存储介质，所述指令用于对包括多个经编码的指令的经掩盖的软件代码进行解码，所述非瞬时机器可读存储介质包括：用于确定针对当前经编码的指令的 N 个先前经编码的指令的指令；用于基于第一函数、第二函数和所述 N 个先前经解码的指令对当前经编码的指令进行解码的指令，其中所述第二函数基于所述 N 个先前经解码的指令，并且其中所述第二函数基于当前经编码的指令和所述第二函数的输出。

[0027] 描述了多种实施例，还包括：用于接收密钥的指令，其中所述密钥与针对第一经编码的当前指令的 N 个先前指令相对应。

附图说明

[0028] 为了更好地理解各种示例性实施例，参考附图，其中：

[0029] 图 1 示出了对软件代码进行掩盖的方法。

[0030] 为了便于理解，使用相同的附图标记来指代具有实质上相同或相似的结构和 / 或实质上相同或相似的功能的元素。

具体实施方式

[0031] 说明书和附图示出了本发明的原理。因此，将理解的是，本领域技术人员将能够设想出实现本发明的原理并且包括在其范围内的各种布置（尽管这里并未明确地描述或示出）。此外，这里记载的所有示例主要特别旨在用于教导目的，以便辅助读者理解本发明的原理和发明人促进现有技术所贡献的构思，并且应被理解为不限于所特别描述的示例和条件。此外，除非另作指出（例如“或其它”或者“或在备选方式中”），本文中的术语“或”指代非排他性的“或”（即，和 / 或）。此外，本文描述的各个实施例并不必是互斥的，这是一些实施例可与一个或多个其它实施例组合，以形成新的实施例。

[0032] 可使用程序代码来实现软件应用。可采用更高级语言来写程序代码。然后，该程序代码可被编译或解译。经编译的代码可以是针对特定处理器的机器代码，其中经编译的代码使用来自针对特定处理器的指令集的指令。此外，所述代码可以被编译成字节代码。字节代码是表示各种操作的代码。然后，字节代码可被进一步解译，以在特定处理器上实现。这一技术可例如与虚拟机 (VM) 一起使用。然后，可以将 VM 实现为运行在各种处理器上，但对于将在任意处理器上实现的应用来讲，字节代码将是相同的，这是因为 VM 所操作的字节代码然后会被 VM 解译成针对特定处理器的机器代码。

[0033] 在许多情况中，软件应用必须防御尝试对代码进行反向工程的攻击者，如上面的

示例中所述的。攻击者可以使用复杂的工具来分析采用二进制形式的软件，以便理解软件正在做什么以及软件如何工作。代码模糊处理可用来实现软件应用，以便使得攻击者更难理解代码的操作、修改代码、移动代码或从代码中提取敏感信息。由于攻击者只能够访问可执行代码而不能访问源代码，所以模糊处理技术通常应用于可执行代码。

[0034] 一种用于对代码进行模糊处理的方式是使得对代码的解码（其对执行代码来讲是必要的）取决于最近执行的代码。以下实施例是通过使用在VM中实现的字节代码进行描述的。然而，在实施例中所描述的技术可以更广泛地应用于其它类型的代码和处理器。此外，除了对代码进行模糊处理之外，所述实施例还使应用免遭篡改。

[0035] 在许多情况中，可能想要在可以容易地观测到软件的执行的开放平台上运行软件，但同时想要隐藏软件实际上正在做什么。这可通过使用软件模糊处理来实现。这通常用于防止软件中所使用的知识产权被盗以及隐藏涉及安全性的软件的内部处理。

[0036] 一些工具可用来对软件进行模糊处理(ExeCryptor、VMProtect、Code Virtualizer、Thermida)。其中许多工具将要保护的软件转译成虚拟机(VM)，其中VM的指令以及如何在存储器中对其进行编码都是秘密。软件模糊工具还提供解译器，从而VM知道VM指令以及如何在存储器中对其进行编码。解译器与经转译的应用代码组合在一起形成经模糊处理的应用。

[0037] 一种用来创建快速解译器（从而代码模糊处理的开销较低）的非常流行的方法是使用字节代码。针对字节代码VM的程序包括多个字节，其中字节值指令将被执行的操作。例如，字节值12可表示乘法，字节值13可表示减法。可在作为将被VM执行的操作的输入的字节代码之间找到立即操作数。例如，字节代码14可表示将16比特的常量与某变量相加，其中常量是从跟随该字节代码的两个字节获得的。

[0038] 现在将描述实施例的概述。所描述的实施例对字节代码进行离线编码（当生成VM代码时）并且在解译期间对其进行解码。对字节代码的编码利用最近执行的字节代码。为了对字节代码B进行编码，并且在执行B之前未编码的字节代码是L_1，并且在L_1之前未编码的字节代码是L_2，等等，则针对B的经编码的值（称为B'）是：

[0039] $B' = f(B, g(L_1..L_n))$。

[0040] 为了提高安全性，$L_1..L_n$值可以是未编码的字节代码，但如果$L_1..L_n$包含经编码的字节代码，本发明同样是有效的。在执行期间，解译器按照下式对所获取的字节代码进行解码：

[0041] $B = f^1(B', g(L_1..L_n))$。

[0042] 其中，f^1是函数f的逆，满足：

[0043] $f^1(f(x,y),y) = x$。

[0044] 在执行了字节代码B之后，值$L_1..L_n$被移位一个位置，其中丢弃L_n，并且新的L_1变成B。换言之：

[0045] $L_n = L_{n1}$

[0046] $L_{n1} = L_{n2}$

[0047] ...

[0048] $L_2 = L_1$

[0049] $L_1 = B$。

[0050] 通过按所示的对字节代码进行编码，对其进行解码所需的信息变得只在执行代码期间可用。

[0051] 代码中的分支（其中特定代码位置可经由几个控制路径到达）意味着 $L_1..L_n$ 在代码中的这些点处不是唯一确定的。下文将描述如何处理这一问题。

[0052] 现在将描述两个实施例。第一实施例是针对 f、g 和 n 使用以下配置：

[0053] f(x，y) = x-y

[0054] g(x) = x

[0055] n = 1

[0056] 由于字节代码是使用其值和上一执行字节代码之差来编码的，所以该实施例可被称为差分字节代码。解译器通过应用 f^1（其是加法）来对经编码的字节代码进行解码。为了正确操作，f 中的减法和 f^1 中的加法应该在字节值范围 0..255 内回绕 (wrap around)。

[0057] 第二实施例可使用：

[0058] f(x，y) = x xor y

[0059] $g(x_1, x_2) = S[x_1]$ xor $S[x_2]$

[0060] n = 2

[0061] 其中 xor 代表排他性的“或”，S[x] 是将字节映射到字节的替换操作。S[..] 映射可以不具有结构，从而攻击者难以分析是如何对代码进行编码和解码的。

[0062] 如上节所述，如果代码中存在连接点（其中使用跳 (jump) 来跳到代码中的不同点，这是频繁发生的），则 $L_1..L_n$ 不是唯一的。例如，考虑以下 4 个未编码的代码，其中在代码 B3 处存在跳条目目标：

[0063] B1

[0064] B2

[0065] J：B3

[0066] B4。

[0067] 根据 B3 是在 B1 和 B2 之后执行还是经由到连接点 J 的跳来执行，$L_1..L_n$ 中的执行历史将有所不同。存在不同的方式来解决这一问题。第一种方法是在使用每个先前字节代码进行编码或解码之前将其与应该应用于 L_i 上的 n 个掩码值相关联。掩码指示哪些比特在引导至字节代码的所有执行路径中具有相同值。即，针对在连接点之前与其它字节代码共享执行历史中的并行位置的任何字节代码，掩码基于在并行字节代码中找到的公共比特。编码和解码从而变成：

[0068] $B' = f(B, g(L_1 和 M_1..L_n 和 M_n))$

[0069] $B = f^1(B', g(L_1 和 M_1..L_n 和 M_n))$

[0070] 通过掩码 M_i，从 L_i 中移除不应当用作 g 的自变量的比特，这是因为它们不是恒定的。在存储器中，应该将掩码与字节代码相邻地存储。

[0071] 第二种方法是确保 $L_1..L_n$ 总是唯一的。通过将 n 个 noop（没有任何效果的字节代码）放置于标签 J 之前以及以 J 为目标的跳之前，总是能够实现这一点。从而，如果 n = 2，则上例变成：

[0072] B1

[0073] B2

[0074] NOOP

[0075] NOOP

[0076] J :B3

[0077] B4

[0078] NOOP

[0079] NOOP

[0080] 跳 J。

[0081] 第三种方法是下移标签 J 以及将 n 个字节代码拷贝到以 J 为目标的跳之前的位置。针对 n ＝ 1,上例变成 :

[0082] B1

[0083] B2

[0084] B3

[0085] J :B4

[0086] B3// 拷贝自原始 J 位置

[0087] 跳 J。

[0088] 在这一变换之后,针对 B4,L_1总是 B3。最初,代码将按照标签 J 所指示的跳到代码 B3。取而代之地,标签 J 下移一个指令至代码 B4,并且指令 B3 放置于跳指令之前具有跳指令的代码的其它部分中。如果 n 是 2,则跳目标将下移两个指令,并且两个指令将被放置于跳指令之前。

[0089] 在第二种方法和第三种方法中,通过使用 noop 以及拷贝字节代码,跳字节代码不应被用来使用其自己的字节代码来更新执行历史 $L_1..L_n$。

[0090] 以上实施例只描述了对字节代码(即指明要执行的操作的字节)进行编码而没有描述对位于典型 VM 中的字节代码之间的中间数据进行编码。将这一技术应用于中间数据是一种直接扩展。

[0091] 在一种对中间数据进行编码的实施例中,$L_1..L_n$值需要在执行经保护的代码之前的初始值。然后,该初始值还可充当程序的密钥。如果正确的值是不可用的,则不能正确地执行代码。

[0092] 利用高级抽象解译技术,攻击者可以尝试 $L_1..L_n$的值,并且当攻击者确信他们已经找到正确值时,攻击者借助抽象解译通过代码传播该信息以便对其进行解码。通过为 n 选择一个较大的值,可使得该任务更难,但是这将增加解译开销并增加处理代码中的连接点的技术的成本。另一种方法是生成将 $L_1..L_n$在代码中的特定位置处设置为计算恒定值的字节代码。例如,字节代码可将 R_0拷贝到 $L_1..L_n$,其中,R_0是虚拟机的寄存器 (register)。在执行该指令之前,寄存器 R_0已被计算例如 gcd(12341,3131)(最大公分母)的一段代码赋值。因此,抽象解译工具不能计算出所产生的恒定值。这可防止使用抽象解译在代码中的这些地方进行攻击。

[0093] 虽然本文所描述的实施例的目标是对程序代码进行模糊处理,但这些实施例还可用于使代码免遭篡改。如果攻击者尝试计算出代码中关键字节代码的位置并尝试用另一字节代码来替代该字节代码以便实现改变代码的操作的目标,则攻击者还将改变对跟随经修改的字节代码的字节代码的解译。因此,跟随的代码将成为“垃圾”代码。

[0094] 如上所述，本文所述的用于对字节代码进行模糊处理的实施例可应用到其它类型的软件程序。这些实施例还可应用到例如使用（硬件）解码功能进行扩展的 ARM 处理器内核，其中在针对 ARM 处理器的编译器中实现编码。

[0095] 本文描述的实施例可实现在编译器中，该编译器将更高级语言编译成用于在处理器上执行的机器代码。此外，本文所述的实施例可以实现在解译器中，该解译器将程序指令解译成用于在处理器上执行的机器代码。此外，可将实施例应用于现有机器或其它可执行代码，以掩盖对该机器代码的操作。

[0096] 图 1 示出了一种掩盖软件代码的方法。方法 100 开始于 105。接下来，方法可以接收高级语言源代码 (110)。然后，方法 100 可编译或解译高级语言源代码 (115)。接下来，方法 100 可定义第一函数和第二函数（比如上述 f 和 g）以及将在对每个字节代码进行编码时使用的先前字节代码的数量 (120)。然后，方法 100 可以针对每个字节代码确定是否需要掩码值 (125)。备选地，在该步骤处，可在作为跳指令的目的地的任意字节代码之前添加 n 个 NOOP 字节代码。此外，取而代之地，可按上文所述下移任意跳目的地。然后，选择下一个要处理的字节代码 (130)。接下来，方法 100 可以确定 n 个先前字节代码 (135)。然后，可按上文所述使用前 n 个字节代码和第一和第二函数对当前字节代码进行编码 (140)。接下来，方法 100 可以确定是否存在更多字节代码要编码 (145)。如果是的话，则方法选择下一个用于处理的字节代码 (130) 并重复步骤 135 和 140。如果不是的话，则方法 100 在 150 处结束。

[0097] 所述方法还可以包括按照上文所述使用中间数据值来对任意给定的字节代码进行编码。当编译代码时，该方法可全部在编译器中执行。此外，在已经编译了代码之后，可以独立于编译器来应用其中的许多步骤。在上文所述的实施例中对方法 100 中的步骤的多个方面进行了讨论。此外，该方法可运行于机器代码或其它类型的计算机指令，以便对其操作进行模糊处理。

[0098] 一种根据本发明的实施例的方法可作为计算机实现方法实现在计算机系统上。针对根据本发明的方法的可执行代码可存储在计算机程序介质上。计算机程序介质的示例包括存储器设备、光存储设备、集成电路、服务器、在线软件等。这种计算机系统还可包括其他硬件元件，包括存储器、用于与外部系统以及在计算机系统的元件之间传输数据的网络接口。

[0099] 在本发明的实施例中，计算机程序可包括适于在计算机程序运行于计算机上时执行根据本发明的方法的所有步骤的计算机程序代码。优选地，计算机程序实现在非瞬时计算机可读介质上。

[0100] 一种根据本发明创建经掩盖的代码的方法可作为计算机实现方法实现在计算机上。针对根据本实施例的方法的可执行代码可存储在计算机程序介质上。在这种方法中，计算机程序可包括适于在计算机程序运行于计算机上时执行方法的所有步骤的计算机程序代码。计算机程序实现在非瞬时计算机可读介质上。

[0101] 运行于处理器上的用来实现本发明的实施例的特定软件的任意组合构成特定专用机器。

[0102] 本文使用的术语“非瞬时机器可读存储介质”将被理解为排除瞬时传播信号，但包括所有形式的易失性和非易失性存储器。此外，本文使用的术语“处理器”将被理解为涵盖

多种类型的设备，比如微处理器、现场可编程门阵列 (FPGA)、专用集成电路 (ASIC) 和其它类似的处理设备。当在处理器上实现软件时，组合变成单个特定机器。

[0103] 虽然特别地参照多种示例性实施例的某些示例性方面对多种示例性实施例进行了具体描述，但应该理解的是，本发明能够具有其它实施例并且能够在多个明显的方面对其细节进行修改。对于本领域技术人员显而易见的是，在保持在本发明的精神和范围内的同时，可以进行各种变形和修改。从而，前述公开、说明书和附图只是为了说明，而不以任何方式限制本发明，本发明仅由权利要求限定。

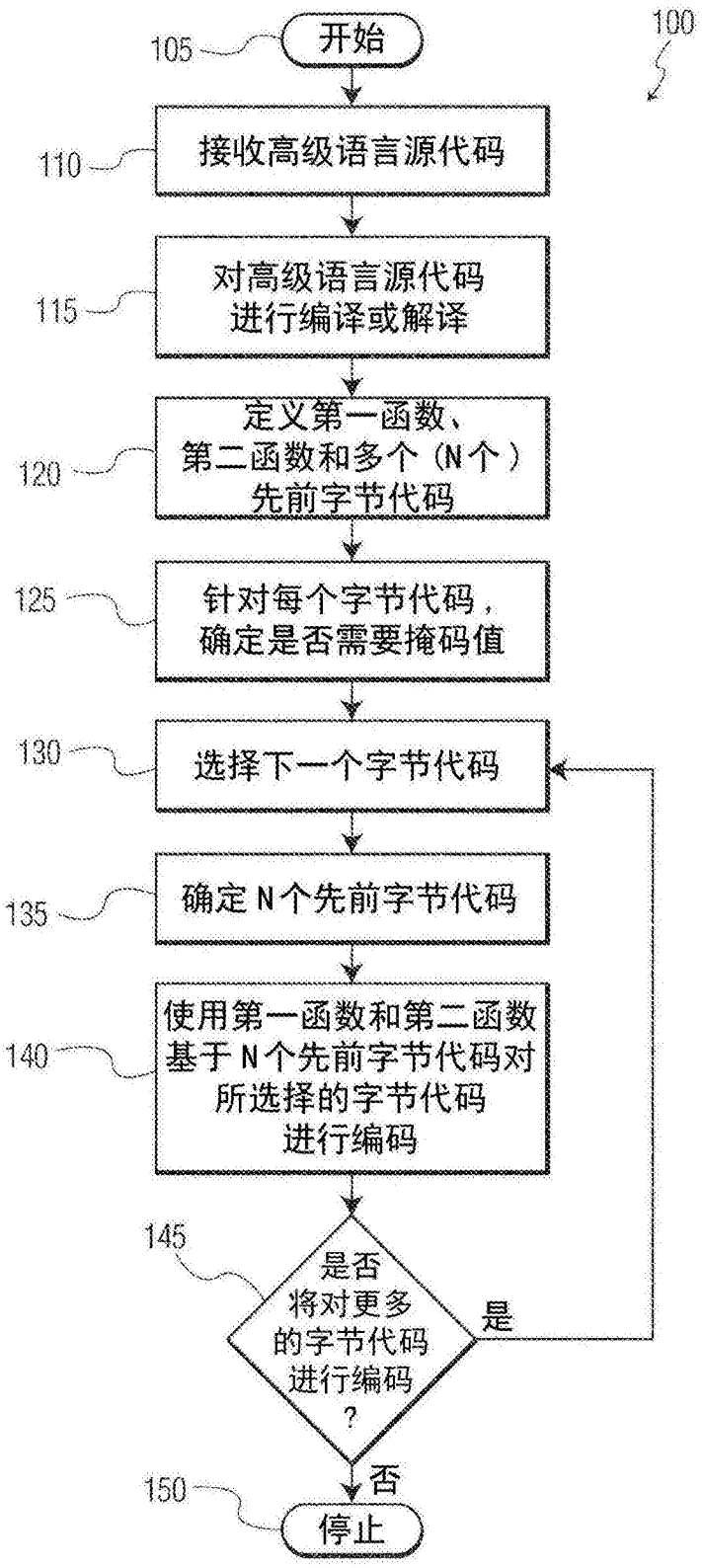

100
105
开始
110
接收高级语言源代码
115
对高级语言源代码
进行编译或解译
120
定义第一函数、
第二函数和多个（N个）
先前字节代码
125
针对每个字节代码，
确定是否需要掩码值
130
选择下一个字节代码
135
确定N个先前字节代码
140
使用第一函数和第二函数
基于N个先前字节代码对
所选择的字节代码
进行编码
145
是否
将对更多
的字节代码
进行编码
？
是
否
150
停止

图 1