基于动态数据流的密码函数加解密过程分析

李继中，蒋烈辉，舒 辉，戴 理
（解放军信息工程大学 数学工程与先进计算国家重点实验室，郑州 450000）

摘 要：在密码函数识别的基础上，采用九元组的形式定义了加解密过程依赖图，提出了基于动态数据流分析的函数依赖图构建框架，并设计了构建算法；采用垃圾调用删除、循环归约的方式对依赖图进行化简；基于标准密码算法依赖图知识库，设计了相似性判别算法对依赖图集合进行判别，测试结果验证了方法的可靠性与高效性。

关键词：算法识别；密码函数；加解密过程；动态数据流；依赖图
中图分类号：TP309.7 **文献标志码**：A **文章编号**：1001-3695(2014)04-1185-04
doi:10.3969/j.issn.1001-3695.2014.04.055

Analysis of encryption and decryption process among crypto functions based on dynamic data-flow

LI Ji-zhong, JIANG Lie-hui, SHU Hui, DAI Li
(*State Key Laboratory of Mathematical Engineering & Advanced Computing, PLA Information Engineering University, Zhengzhou* 450000, *China*)

Abstract: On the result of crypto functions recognition, the paper adopted nine tuple to define the encryption/decryption process relation graph, and brought forward the relation graph constructing framework and algorithm. It also made use of rubbish-call deletion and loop reduction to simplify the relation graph. Lastly the paper applied the similarity decision algorithm to detect the standard algorithm from typical crypto libraries. The experiment verifies the reliability and high efficiency of above method.

Key words: algorithm recognition; crypto function; encryption and decryption process; dynamic data-flow; relation graph

近些年二进制代码中的密码算法逆向[1]已经成为恶意代码检测[2,3]、协议逆向分析[4,5]和密码破解[6,7]的重要研究内容，国内外针对密码算法的识别与定位已经开展了大量研究，分别采用静态识别技术和动态识别技术，针对密码算法软件实现时所体现的各类特征（关键数据特征、指令统计特征、输入与输出数据特征等）进行识别。软件加解密过程是基于密码算法识别的高层抽象，主要包括明密文获取、加解密过程初始化、密钥产生、密钥交换、密钥扩展和加解密操作等步骤，在海量代码中抽取出加解密过程可以有效地提高程序分析人员的代码逆向效率，但是国内外在该方面的研究并不多见。针对该问题本文提出了解决方案。

1 相关定义

密码算法在软件实现时通常采用函数化的设计思想，加解密过程中的函数之间具有一定的时序调用关系，阶段之间的耦合度比较低。为了直观展示软件的加解密过程，在密码函数识别与筛选的基础上，以输入的明/密文为污点数据，采用函数依赖图的方式对密码函数的控制依赖关系和数据依赖关系进行描述。

定义 1 加解密过程依赖图采用九元组的形式进行描述，$G=\{N, CN_i, \mathrm{Num}_i, E_{dd}, E_{cd}, \mathrm{Entry}, \mathrm{Exit}, \mathrm{Type}, \mathrm{Count}_{\mathrm{type}}\}$。其中各元素所代表的含义如下：

N 表示普通节点，代表一般函数调用。

CN_i 表示频繁调用节点，i 表示该节点在依赖图中的索引；在数据加解密过程中，密码函数调用往往会循环多次。

Num_i 表示 CN_i 被调用的次数。

E_{dd}是数据依赖边，表示数据依赖关系，数据依赖边的方向表明污点数据在节点间的传播方向。

E_{cd}是控制依赖边，表示控制依赖关系，控制依赖边的方向表明节点间的调用关系。

Entry 表示依赖图入口点。

Exit 表示依赖图出口。

Type 表示依赖图中所有节点的函数调用类型的统计信息。

$\mathrm{Count}_{\mathrm{type}}$表示依赖图中各类型函数调用出现的次数。

定义 2 节点属性指节点对应的函数调用在加解密过程中的功能，主要包括密钥生成节点、加解密处理节点、明文处理节点和其他函数节点四种，节点 N 的属性用 A_N 表示。

定义 3 过程依赖图统计特征差异度。给定两个非空依赖图 G 和 G'以及 G 和 G'中四种属性节点的个数 c_1、c_2、c_3、c_4 和 c'_1、c'_2、c'_3、c'_4，用 $\Delta\mathrm{Stat}(G,G')$来表示依赖图 G 和 G' 的统计特征差异度，其计算公式如下：

$$\Delta\mathrm{Stat}(G,G')=\frac{|c_1-c'_1|+|c_2-c'_2|+|c_3-c'_3|+|c_4-c'_4|}{\max(|G|,|G'|)}$$

其中：$|G|$和$|G'|$分别表示 G 和 G'的节点数。

收稿日期：2013-05-17；**修回日期**：2013-07-23

作者简介：李继中（1983-），男，河南上蔡人，博士研究生，主要研究方向为二进制逆向、密码学、信息安全等（zhongzhong_hero@163.com）；蒋烈辉（1967-），男，教授，博士，主要研究方向为固件逆向、信息安全；舒辉（1974-），男，副教授，博士，主要研究方向为信息安全；戴理（1988-），男，硕士，主要研究方向为信息安全.

定义 4 数据依赖边等价。数据依赖边 E_{dd} 和 E'_{dd} 等价，当且仅当 E_{dd} 和 E'_{dd} 上传播的数据相同。

定义 5 控制依赖边等价。控制依赖边 E_{cd} 和 E'_{cd} 等价，当且仅当引起 E_{cd} 和 E'_{cd} 上的控制依赖关系的寄存器污点数据相同。

定义 6 依赖边集合相等。对于依赖边集合 $C_E = \{E_1, E_2, \cdots, E_n\}$ 和 $C'_E = \{E'_1, E'_2, \cdots, E'_m\}$，当同时满足以下条件时，称依赖边集合 C_E 和 C'_E 相等：a) $n = m$；b) 对于任意一条依赖边 $E \in C_E$，存在 $E' \in C'_E$ 与 E 等价；c) 对于任意一条依赖边 $E' \in C'_E$，存在 $E \in C_E$ 与 E' 等价。

定义 7 等价节点。节点 N 与 N' 等价，当且仅当：a) N 与 N' 的节点行为相同，即节点类型（普通节点或频繁调用节点）、参数个数和类型相同，当参数为常数时，要求它们的大小相等；b) 从 N 出发的每一条依赖边 E 都有一条对应的等价依赖边 E' 从 N' 出发，且 E 与 E' 的到达节点的节点行为相同。

定义 8 最大等价子图。给定过程依赖图 G_1 和 G_2，如果 G 是由 G_1 和 G_2 的所有等价节点构成的依赖图，则称 G 为 G_1 和 G_2 的一个最大等价子图，即 $G = \mathrm{mec}(G_1, G_2)$。

定义 9 过程依赖图的相似度。给定两个非空依赖图 G_1 和 G_2 以及它们的最大等价子图 $\mathrm{mec}(G_1, G_2)$，则 G_1 和 G_2 的相似度 $S(G_1, G_2)$ 可以定义为

$$S(G_1, G_2) = \frac{|\mathrm{mec}(G_1, G_2)|}{\max(|G_1|, |G_2|)}$$

2 基于动态数据流的加解密过程依赖图构建

2.1 加解密过程依赖图构建框架

加解密过程依赖图的构建以目标可执行程序的动态执行信息为基础，采用可回溯的污点传播方法将加解密数据标记为内存污点或寄存器污点（以字节为基本粒度），并制定污点传播规则，记录下改变污点状态的指令信息。将动态数据流分析过程划分为两个阶段，即数据关联性分析和数据流跟踪。数据关联性分析根据程序执行期间操作过所有的内存和寄存器信息，进行污点传播分析和污点管理，并生成全局性数据关联图[8]，图中标志出函数间和函数内的数据关联信息。该过程仅执行一次，当污点源发生改变后，无须重复污点传播分析。以全局数据关联图为基础，将加解密数据设置为污点源，进行污点传播路径和加解密过程的回溯，从而构建出加解密过程依赖图。加解密过程依赖图构建框架如图 1 所示。

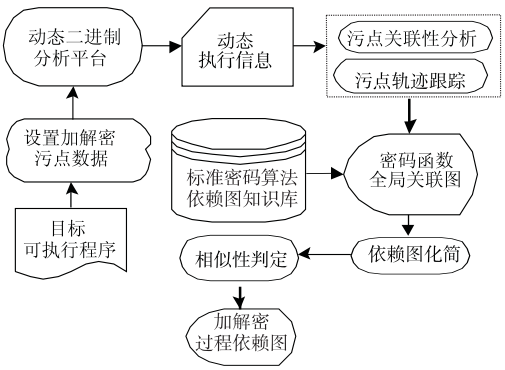

图1 加解密过程依赖图构建框架

2.2 加解密过程依赖图构建算法

依赖图的构建是针对数据关联图 G，主要操作包括初始节点添加、普通节点添加、依赖边添加、频繁调用节点添加以及构建完成判别等步骤，输出依赖图集合 C。构建算法的主要步骤如下：

a) 在依赖图集合 C 中创建依赖图 $G_1 = \mathrm{NULL}$，并将当前依赖图 G' 置为 G_1。

b) 将加解密数据标记为污点，将产生污点的函数调用作为初始节点 Entry 加入当前依赖图 G'。

c) 从全局数据关联图 G 中按照程序执行顺序读取一条函数调用的记录 R 或区域单元（区域单元是指若干序号连续的指令组成的集合，一个或多个连续的区域单元构成一个区域）的记录 R。如果 R 进行污点传播则作为普通节点 N 加入当前依赖图 G' 中，并将当前节点 N' 置为 N，转步骤 d)；如果生成了新污点，则创建子依赖图 $G_n = \mathrm{NULL}$，并将当前依赖图 G' 置为 G_n，转步骤 b)；如果存在污点漂白（污点漂白是指非污点数据对污点数据进行修改或覆盖）操作，则转步骤 f)。

d) 对于一切 $N \in G'$ 且 N 与 N' 存在数据依赖关系的节点 N，在 N' 和 N 之间添加数据依赖边 E_{dd}，依赖边方向为污点传播的方向。

e) 如果 N' 中存在控制流跳转受寄存器污点影响的情况，则在 G' 中产生或传播该污点的节点和 N' 之间添加控制依赖边 E_{cd}，依赖边的终点为 N'，转步骤 c)。

f) 当污点数据被漂白时，将漂白污点的节点作为当前依赖图 G' 的出口节点 Exit，结束 G' 的构建，并将 G' 加入依赖图集合 C。

g) 如果 G' 存在父依赖图，则将当前依赖图 G' 置为其父依赖图，转步骤 c)；否则，置 $G' = \mathrm{NULL}$，转步骤 c)。

h) 当全局数据关联图中的最后一条函数调用记录或区域单元记录处理结束后，结束当前依赖图 G' 的构建，并将 G' 加入依赖图集合 C，依赖图集合 C 构建结束。

在依赖边的添加过程中，若污点数据从节点 A 传播到节点 B 没有被程序修改，则将数据依赖边 AB 标记为 stable，否则标记为 changed；此外，通过依赖图循环识别添加频繁调用节点。

使用 WinHttp 协议发送加密文件是常用的网络信息传送方式，采用加解密过程依赖图对该过程进行描述，如图 2 所示。图中的实线表示控制依赖关系（前驱节点的执行状态会对后继节点产生影响），虚线表示数据依赖关系（污点在两个节点之间发生了传播行为），箭头表明了污点传播的方向。

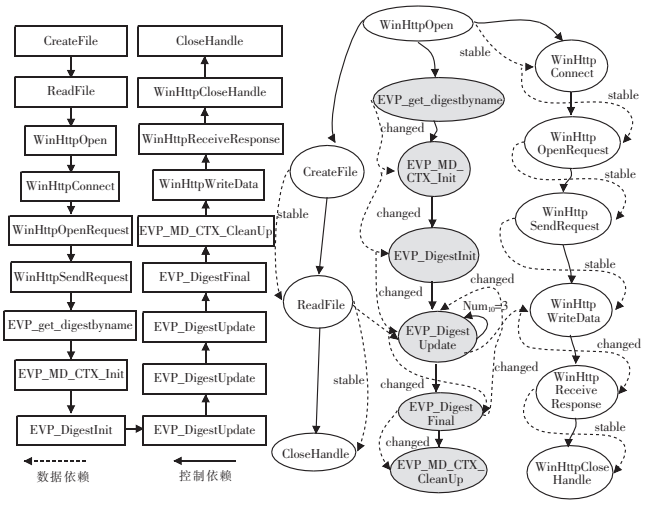

图 2 WinHttp 协议发送加密信息的过程描述

3 加解密过程依赖图化简

加解密过程依赖图集往往规模比较庞大，尤其是在对大型

应用软件进行分析时,会产生比较大的空间开销。通过对依赖图集的分析发现,在对污点数据进行回溯时,存在大量的内存整体移动、赋值操作,由这些操作所引起的数据依赖关系对于定位加解密过程的分析意义不大。为了缩小依赖图集的规模,提高相似性比对的效率,通过垃圾调用删除和循环归约来对原始依赖图集进行化简。

3.1　垃圾调用删除

垃圾调用是指从代码中移除后不改变整个加解密过程的调用,表现为进行了污点传播但在传播过程中没有引起程序执行状态的改变,或产生了污点但是没有进行传播两种情况。在加解密过程依赖图中,分别针对这两种情况进行垃圾调用删除操作。

情况 1　对于节点 N,存在以 N 为起点的数据依赖边 E_{dd},但不存在以 N 为起点的控制依赖边 E_{cd}。

情况 2　对于子依赖图 G,不存在数据依赖边 E_{dd}。

第一种情况的垃圾调用删除步骤为:

a)删除依赖图中以 N 为终点的控制依赖边 E_{cd}。

b)对于以 N 为终点的数据依赖边 E_{dd},记 E_{dd}对应的数据为 data,如果存在以 N 为起点的数据依赖边 E'_{dd},且 E'_{dd}对应的数据也是 data,则将 E_{dd}的终点更新为 E'_{dd}的终点,然后删除 E'_{dd}。

c)删除其他与 N 相连的数据依赖边。

d)删除节点 N。

针对第二种情况,直接将 G 从依赖图中删除,并删除所有与 G 相连的数据依赖边和控制依赖边。

3.2　循环归约

密码算法通常采用分组加密方式并且算法实现内部使用循环方式,使得加解密过程中会出现大量的循环结构,造成原始依赖图规模比较大。采用文献[9]提出的一种基于 DJ 图的循环识别方法,其中循环缩减与否取决于使用的污点内存是否连续和内存操作是否相同。令污点操作的范围 TaintEdge = {TaintAddress, TaintLength},F_n 上一个节点记为 pri(F_n),对于循环 LoopA,若 $F_n \in$ LoopA,则循环缩减为频繁调用节点的归并条件为

$$\text{dataUse}[F_n] \in (\text{Taint}, \text{pri}(F_n).\text{TaintAddress} + \text{pri}(F_n).\text{TaintLength} = F_n.\text{TaintAddress})$$

基于循环归约的频繁调用节点生成过程如图 3 所示。

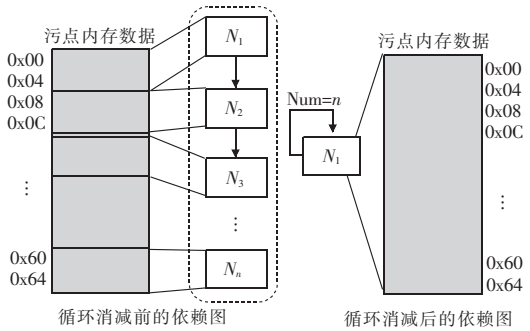

图 3　基于循环归约的频繁调用节点生成

4　基于相似性比对的加解密过程识别

以目标代码的加解密过程依赖图为基础,从知识库中选取相应算法的标准依赖图,根据目标依赖图的统计特征与标准依赖图的差异度,确定相似性比对的优先级,然后按照优先级依次执行相似性比对算法。

依赖图相似性比对的具体步骤描述如下:

a)根据密码算法识别的结果,选取相应算法的加解密过程标准依赖图 G_0。

b)遍历目标程序生成的依赖图集合,计算集合中所有依赖图与 G_0 的依赖图统计特征差异度,按统计特征差异度从小到大的顺序对依赖图集合进行排序,生成有序依赖图集合 $C = \{G_1, G_2, \cdots, G_n\}$。

c)依次从 C 中顺序取出依赖图 G_i($1 \leqslant i \leqslant n$),与 G_0 一起作为相似性比对算法的输入,执行相似性比对算法,输出 G_i 与 G_0 的相似度 $S(G_0, G_i)$。

d)重复步骤 c)直到集合 C 被遍历完毕,选取 C 中与 G_0 相似度最高的依赖图 $G_{\max}$作为相似性比对的结果。

目标依赖图 G_i 和标准依赖图 G_0 的相似性比对过程的具体描述如下:

a)以依赖图 G_i 和 G_0 的入口点 Entry_i 和 Entry_0 为相似性比对的起点,置等价节点计数器 Count = 0,置当前标准节点 N_0 为 Entry_0。

b)置待比对节点 N 为 Entry_i。

c)比较 N 和 N_0 的行为是否相同。如果不相同,则 N 和 N_0 不等价,转 d);否则,转 f)。

d)将 N 标记为 visited。

e)选取一个与 N 存在数据依赖关系的未被标记的非等价节点,将其置为待比对节点 N,转 c);如果不存在这样的节点,选取一个与 N 存在控制依赖关系的未被标记的非等价节点,将其置为待比对节点 N,转 c);如果不存在这样的节点,则从 G_i 中选取一个未被标记为 visited 的非等价节点,将其置为待比对节点 N,转 c);如果 G_i 中所有节点均被标记为 visited,转 i)。

f)判断与 N 相连的依赖边集合$\{E\}$和与 N_0 相连的依赖边集合$\{E_0\}$是否相等。如果相等,转 g);否则 N 和 N_0 不等价,转 d)。

g)对于$\{E\}$和$\{E_0\}$中每一对等价依赖边,例如 $E(N, N') = E(N_0, N'_0)$,其中 N'和 N'_0 分别表示与 N 和 N_0 存在依赖关系的节点,判断 N'和 N'_0 的行为是否相同。如果存在不相同的情况,则 N 和 N_0 不等价,转 d);否则 N 和 N_0 为等价节点,转 h)。

h)标记 N 和 N_0 为等价节点。如果 N 为普通节点,计数器 Count 加 1;如果 N 为频繁调用节点,计数器 Count 加上 N 的调用次数。

i)清除 G_i 中所有的 visited 标记,将当前标准节点 N_0 标记为 visited,选取一个与 N_0 存在数据依赖关系的未被标记的非等价节点,将其置为当前标准节点 N_0,转 e);如果不存在这样的节点,选取一个与 N_0 存在控制依赖关系的未被标记的非等价节点,将其置为当前标准节点 N_0,转 e);如果不存在这样的节点,则从 G_0 中选取一个未被标记为 visited 的非等价节点,将其置为当前标准节点 N_0,转 e);如果 G_0 中所有节点均被标记为 visited,转 j)。

j)根据计数器 Count 计算 G_i 和 G_0 的相似度 $S(G_i, G_0)$。

5　测试与结论

5.1　测试环境(表 1)

5.2　功能测试

选取局域网通信软件 FeiQ 对加解密过程定位模块进行功

能测试。FeiQ 在进行网络通信时采用分组密码算法 BlowFish 对通信数据进行加密；此外，FeiQ 采用的是 MFC 编程框架，API 调用较多，会对加解密过程定位产生干扰。测试共生成加解密过程行为依赖图 73 张，经过与依赖图知识库中的 BlowFish 算法加密过程基准依赖图进行相似性比对，相应测试集中依赖图相似度达到了 93.79%，如表 2 所示。

表 1 软硬件测试环境

环境	项目	配置信息
硬件环境	处理器	Intel® Core™ i7-2600K CPU @ 3.40 GHz
	内存	DDR3 8 GB
	硬盘	SATA II 2TB
软件环境	操作系统	Windows7 Ultimate SP1 x64
	动态二进制分析平台（自研平台）	PIN-2.12-55942
	静态二进制分析工具（商业软件）	IDA-6.2.0111006

表 2 FeiQ 加解密过程依赖图相似性比对结果

应用程序	FeiQ 2012 测试版
模块数/个	81
线程数/个	11
加解密过程依赖图集规模/个	73
相似性比对测试集规模/个	13
依赖图精简化平均时间/ms	451.3
相似性比对平均时间/ms	1 535.7
依赖图平均相似度/%	93.79

相似性判定得到的加解密过程依赖图在内存中以二进制形式存储，采用 DOT 语言对依赖图进行可视化处理，FeiQ2012 加解密过程依赖图如图 4 所示。

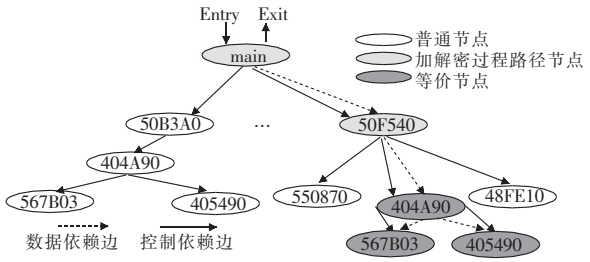

图 4 FeiQ2012 测试版加解密过程依赖图

5.3 性能测试

考虑到测试用例中的密码算法类别、编程框架、程序体积等问题，分别选取 WinRAR、QQ、Foxit Reader（福昕阅览器）、HashCalc 等软件作为用例，性能测试主要包括时间和空间两个方面，分别针对不加载 Pin 插件、加载空插件和加载全功能记录插件时的运行时间作了比较，结果如图 5 所示。无插件与全功能插件的时间差在密码计算软件中的差别不大，而针对功能较多的商业软件，由于记录信息量比较大导致该时间差较大。

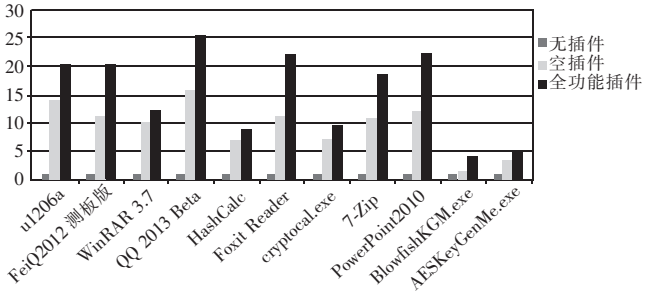

图 5 数据流记录时间性能柱状图

针对测试用例的数据流记录时间和加解密定位时间如表 3 所示，数据流记录平均时间达到 353 922 ms，加解密定位平均时间达到 10 716 ms，其中复杂网络通信软件 u1206a、压缩工具和办公工具 PowerPoint2010 的数据记录与加解密工具过程定位的时间比较长，而密码算法小工具 BlowfishKGM 和 AESKeyGenMe 的耗时则较少。

表 3 性能测试结果

测试用例	软件大小/KB	时间/ms	
		数据流记录	加解密过程定位
u1206a.exe	1 886	611 227	26 073
FeiQ2012	15 007	30 320	1 987
WinRAR 3.7	4 603	873 410	33 527
QQ2013 Beta	4 873	170 536	3 841
HashCalc	444	502	377
Foxit Reader	35 696	286 250	2 627
cryptocal.exe	780	612	456
7-Zip	4 560	603 245	12 572
PowerPoint2010	2 113	1 316 321	36 322
BlowfishKGM	100	313	52
AESKeyGenMe	108	412	51

6 结束语

本文在密码函数识别的基础上，设计了加密过程依赖图构建算法来进行垃圾调用删除、循环归约，以消减依赖图规模；根据已知的标准密码算法依赖图，采用相似性比对的方法对目标加解密过程依赖图进行判别，并选取通信软件 FeiQ2012、压缩软件 WinRAR 3.7 等工具分别进行了功能测试和性能测试。

加解密过程依赖关系分析是密码编制重构的重要步骤，对于目标代码的安全性分析具有重要作用。随着应用软件规模的不断增大，使得所构造的加解密过程依赖图不断膨胀，如何进一步归约依赖图是下一步研究的重要工作。

参考文献：

[1] LI Ji-zhong, JIANG Lie-hui, YIN Qin, *et al*. Hybrid method to analyze cryptography in software[C]//Proc of the 4th International Multimedia Information Networking and Security. 2012: 930-933.

[2] LEDER F, MARTINI P, WICHMANN A. Finding and extracting crypto routines from malware[C]//Proc of the 28th International Performance Computing and Communications Conference. 2009: 394-401.

[3] CALVET J, FERNANDEZ J M, MARION J Y. Aligot: cryptographic function identification in obfuscated binary programs[C]//Proc of ACM Conference on Computer and Communications Security. New York: ACM Press, 2012: 169-182.

[4] WANG Zhi, JIANG Xu-xian, CUI Wei-dong, *et al*. ReFormat: automatic reverse engineering of encrypted messages[C]//Lecture Notes in Computer Science, vol 5789. Berlin: Springer-Verlag, 2009: 200-215.

[5] LUTZ N. Towards revealing attacker's intent by automatically decrypting network traffic[D]. [S.l.]: ETH Zuerich, 2008.

[6] ZHAO Ruo-xu, GU Da-wu, LI Juan-ru, *et al*. Detection and analysis of cryptographic data inside software[C]//Lecture Notes in Computer Science, vol 7001. Berlin: Springer-Verlag, 2011: 182-196.

[7] GREBERT F, WILLEMS C, HOLZ T. Automated identification of cryptographic primitives in binary programs[C]//Lecture Notes in Computer Science, vol 6961. Berlin: Springer-Verlag, 2011: 41-60.

[8] 李洋. 数据流分析在密码算法识别中的应用[D]. 郑州：解放军信息工程大学，2012.

[9] SREEDHAR V C, GAO G, LEE Y F. Identifying loops using DJ graphs[J]. ACM Trans on Programming Languages and Systems, 1996, 18(6): 649-658.