中国科学 : 信息科学 2025 年 第 55 卷 第 6 期: 1447–1470
SCIENTIA SINICA Informationis
论文

深度学习辅助密码分析的通用增强框架: 应用于 Speck, Simon 和 LEA 算法

申焱天[1,3], 陈怡[2], 于红波[1,3,4*]

1. 清华大学计算机科学与技术系, 北京 100084
2. 清华大学高等研究院, 北京 100084
3. 中关村实验室, 北京 100084
4. 清华大学密码与数字经济安全全国重点实验室, 北京 100084

* 通信作者. E-mail: yuhongbo@mail.tsinghua.edu.cn

收稿日期: 2025–01–14; 修回日期: 2025–03–18; 接受日期: 2025–04–16; 网络出版日期: 2025–06–10

国家密码科学基金 (批准号: 2025NCSF02014) 资助项目

摘要 在 CRYPTO 2019 上, Gohr 首次将深度学习技术应用于分组密码的安全性分析, 开辟了深度学习辅助密码分析的研究方向. 深度学习辅助密码分析的核心为基于深度神经网络构建的差分 – 神经区分器. 相比于经典差分区分器, 差分 – 神经区分器可以达到更高的准确率, 有利于降低密钥恢复攻击的数据复杂度. 然而, 差分 – 神经区分器的访问涉及大量浮点运算, 引入了额外的计算成本, 导致攻击的计算复杂度较高. 这一缺点极大地限制了深度学习辅助密码分析的性能. 本文提出一种通用的深度学习辅助密码分析增强框架, 可以在几乎不损失区分准确率的前提下, 将差分 – 神经区分器转换为存储复杂度可忽略的查找表, 克服了差分 – 神经区分器的缺点, 显著地增强深度学习辅助密码分析的能力. 在美国 NSA 设计的 Speck, Simon 以及 ISO/IEC 标准 LEA 三类分组密码上的实验, 充分地验证了该框架的有效性和通用性. 基于该框架, 本文改进了对 Speck32/64, Speck96 以及 Speck128 的深度学习辅助分析结果. 特别地, 本文给出了国内外第 1 个针对 13 轮 Speck32/64 完整验证的实际密钥恢复攻击, 同时给出 Speck96, Speck128 目前最长轮数的实际攻击. 这些攻击充分说明了本文框架对于增强深度学习辅助密码分析的应用价值.

关键词 深度学习, 对称密码分析, Speck, Simon, LEA

1 引言

分组密码算法是一类重要的对称密码原语, 用于保障数据机密性. 人们对分组密码算法安全性的信心来自于算法对已有密码分析方法的抵抗力. 常见的密码分析方法包括差分分析、线性分析等.

在 CRYPTO 2019 上, Gohr [1] 首次提出深度学习辅助的密码分析方法. Gohr 使用深度神经网络在 Speck32/64 算法上构建了差分约束下的密码区分器, 称为差分 – 神经区分器, 相比传统全差分分布

引用格式: 申焱天, 陈怡, 于红波. 深度学习辅助密码分析的通用增强框架: 应用于 Speck, Simon 和 LEA 算法. 中国科学: 信息科学, 2025, 55: 1447–1470, doi: 10.1360/SSI-2025-0024
Shen Y T, Chen Y, Yu H B. General enhancing framework for deep learning-aided cryptanalysis: applications to Speck, Simon and LEA ciphers. Sci Sin Inform, 2025, 55: 1447–1470, doi: 10.1360/SSI-2025-0024

表 DDT (differential distribution table) 区分器, 取得了更高的区分准确率. Gohr 进一步设计了一种差分 – 神经攻击框架, 并成功应用于对分组密码 Speck32/64 的密钥恢复中. 此外, Gohr 还设计了一种基于贝叶斯 (Bayes) 优化的高效密钥猜测策略, 大大降低了密钥猜测的复杂度. 最终, Gohr 给出了对 11 轮 Speck32/64 算法的密钥恢复攻击, 相比已有差分攻击结果在计算复杂度上有明显的改进.

Gohr 的工作开辟了深度学习辅助密码分析这一新的研究方向, 近年来在这一方向出现了许多差分 – 神经攻击结果 [2~5]. 在 ASIACRYPT 2022, Bao 等 [2] 扩展了 Gohr 攻击中使用的中性比特, 结合 8 轮差分 – 神经区分器, 首次给出对 13 轮 Speck32/64 算法的差分 – 神经攻击. Zhang 等 [3] 设计了新的差分 – 神经区分器的网络架构, 可接收多个密文对作为输入并取得了更高的区分准确率, 改进了 13 轮 Speck32/64 差分 – 神经攻击的复杂度, 并进一步将攻击扩展至 14 轮. 然而由于计算复杂度太高, 文献 [2,3] 只对 13 轮攻击的核心过程进行了实验验证[1)]. Chen 等 [4] 提出了一种用于大状态分组密码的神经辅助密钥恢复框架, 将差分 – 神经攻击扩展至大状态 Speck 成员. 同样由于攻击计算复杂度太高, 文献 [4] 对于 Speck96 和 Speck128 算法也分别只给出了 10 轮和 12 轮实际攻击.

上述攻击的成功依赖于差分 – 神经区分器作为核心的攻击组件. 然而, 差分 – 神经区分器在带来优异区分能力的同时, 也产生了额外的计算成本. 攻击中每次访问区分器均涉及神经网络大量的浮点数运算. 因此, 这些差分 – 神经攻击的计算复杂度较高, 通常只能进行较短轮攻击或对攻击核心过程的实验验证, 并且为提高计算效率, 差分 – 神经攻击往往依赖于 GPU 卡加速神经网络的处理. 上述缺点限制了深度学习辅助密码分析的性能和进一步的应用. 为克服这一缺点, 本文研究以下问题:

在深度学习辅助密码分析中, 能否对差分 – 神经区分器进行重建, 在保留其区分能力的同时, 去除深度神经网络的计算成本?

国内外已经有一些研究工作尝试对差分 – 神经区分器进行重建. 在 EUROCRYPT 2021, Benamira 等 [6] 通过在密文对上建立若干个部分输出分布表 (masked output distribution table, M-ODT), 并使用非神经网络的机器学习模型对各分布表的输出进行整合, 在 5~6 轮 Speck32/64 上实现了接近差分 – 神经区分器的准确率. 在 ASIACRYPT 2023, Bao 等 [5] 通过对 Speck 轮函数模加操作的研究, 设计算法改进 DDT 区分器, 甚至实现了优于差分 – 神经区分器的准确率. 然而, 文献 [5,6] 的工作侧重对差分 – 神经区分器的解释, 重建后区分器计算复杂度较高, 难以用于对差分 – 神经攻击的改进. 2024 年, 文献 [7] 基于多条差分 – 线性特征以及全连接网络实现了接近差分 – 神经区分器的准确率, 并首次将重建后的区分器用于改进对 13 轮 Speck32/64 的差分 – 神经攻击. 文献 [7] 中区分器仍需要全连接层的计算, 相较于差分 – 神经区分器在区分时间上的改进程度不高, 且其 13 轮攻击同样只验证了核心过程.

本文贡献. 本文工作表明: 在几乎不损失区分准确率的前提下, 完全去除差分 – 神经区分器的计算成本是可行的. 本文的主要贡献包括如下.

• 我们提出了一种通用的深度学习辅助密码分析的增强框架 NDAFL (neural distinguisher-aided feature location). 该框架通过在合适的输入形式下对差分 – 神经区分器进行比特敏感性测试, 可以将差分 – 神经区分器转化为存储成本很小的查找表, 同时几乎不损失区分准确率. 使用该框架, 可以在差分 – 神经攻击中摆脱对神经网络计算的依赖, 对区分器的访问只等价于一次查表操作, 由此大大降低了差分 – 神经攻击的计算复杂度.

• 我们将该框架成功地应用于 Speck, Simon 以及 LEA 算法中. 我们将各差分 – 神经区分器转换为查找表, 且区分准确率仍高于对应的全差分区分器. 这些结果验证了本文框架的有效性和通用性.

• 基于转换后的查找表, 我们改进了对 Speck32/64, Speck96 以及 Speck128 算法的差分 – 神经攻击, 这也是目前 3 个算法基于深度学习方法计算复杂度最低的攻击结果. 特别地, 我们大大降低了 Speck32/64 算法的 13 轮攻击复杂度, 并给出完整攻击过程的实验验证, 进一步说明已有差分 – 神经

1) 文献 [2] 对完整 13 轮攻击的 1/32 进行了实验验证, 最长运行时间约为 14.5 个 GPU 小时. 文献 [3] 对完整 13 轮攻击的 1/16 进行了实验验证, 平均运行时间约为 4 个 GPU 小时.

表 1　对 Speck32, Speck96 和 Speck128 算法不同攻击总结.

Table 1　Summary of different attacks on Speck32, Speck96 and Speck128.

Cipher	$\#R$	Method	Time	Data	Success rate	Ref.
Speck32/64	13/22	Differential-neural	$2^{51.07}$	2^{29}	0.82	[2]
		Differential	$2^{50.16}$	$2^{31.13}$	–	[8]
		Differential-neural	$2^{45.03}$	2^{27}	0.34	[3]
		Differential-neural	$2^{43.8}$	2^{31}	0.57	[7]
		Differential-neural	$2^{42.31}$	2^{29}	0.68	This work
		Differential-linear	2^{41}	2^{25}	–	[9]
	14/22	Differential-neural	$2^{60.93}$	2^{27}	0.34	[3]
		Differential	$2^{60.58}$	$2^{30.26}$	–	[10]
		Differential-neural	$2^{59.8}$	2^{31}	0.57	[7]
		Differential-neural	$2^{58.31}$	2^{29}	0.68	This work
		Differential-linear	2^{58}	2^{31}	–	[11]
		Differential-linear	2^{57}	2^{25}	–	[9]
Speck96/96	10/28	Differential-neural	$2^{32.41}$	2^{18}	0.98	[12]
		Differential-neural	$2^{33.30}$	2^{17}	0.81	[4]
	13/28	Differential-neural	$2^{52.61}$	$2^{36.60}$	0.81	[4]
	14/28	Differential-neural	$2^{43.03}$	2^{34}	0.82	This work
	20/28	Differential	$2^{95.75}$	$2^{93.17}$	–	[10]
Speck96/144	14/29	Differential-neural	$2^{100.61}$	$2^{38.60}$	0.81	[4]
	15/29	Differential-neural	$2^{91.03}$	2^{34}	0.82	This work
	21/29	Differential	$2^{143.13}$	$2^{93.61}$	–	[10]
Speck128/128	12/32	Differential-neural	$2^{35.49}$	$2^{17.32}$	0.52	[4]
	14/32	Differential-neural	$2^{38.63}$	$2^{27.32}$	0.78	This work
	17/32	Differential	2^{113}	2^{113}	–	[13]
		Differential-neural	$2^{78.98}$	$2^{61.28}$	0.52	[4]
		Differential-neural	$2^{72.36}$	$2^{61.62}$	0.78	This work
	23/32	Differential	$2^{124.95}$	$2^{122.37}$	–	[10]
Speck128/192	18/33	Differential	2^{177}	2^{113}	–	[13]
		Differential-neural	$2^{142.98}$	$2^{61.28}$	0.52	[4]
		Differential-neural	$2^{136.36}$	$2^{61.62}$	0.78	This work
	24/33	Differential	$2^{174.53}$	$2^{123.95}$	–	[10]
Speck128/256	19/34	Differential	2^{241}	2^{113}	–	[13]
		Differential-neural	$2^{206.98}$	$2^{61.28}$	0.52	[4]
		Differential-neural	$2^{200.36}$	$2^{61.62}$	0.78	This work
	25/34	Differential	$2^{238.53}$	$2^{123.95}$	–	[10]

攻击的有效性, 同时给出对 Speck96, Speck128 轮数最长的实际攻击. 这些攻击结果说明了我们框架的应用价值.

对 Speck32, Speck96 和 Speck128 已有攻击结果的总结见表 1[2~4, 7~13]. 表 1 中对 Speck128/128 的 12 轮、14 轮实际攻击同样适用于 Speck128/192 和 Speck128/256 算法. 此外文献 [12] 对其 Speck96/96 的 10 轮攻击, 没有提供具体的计算复杂度, 表 1 给出的计算复杂度数据由文献 [12] 中提供的攻击运行时间的关系换算得到. 本文的实验均在一台配有 NVIDIA RTX 3090 GPU 以及 Intel Xeon Gold 5218R

表 2　本文使用的缩写.

Table 2　Abbreviations used in this paper.

Abbreviation	Meaning	Abbreviation	Meaning
$\mathcal{IB}$	A set of informative bits	$\mathcal{DD}$	DDT-based distinguisher
$\mathcal{NDD}$	Neural differential distinguisher	$\mathcal{ND}$	Neural distinguisher
$\mathcal{LT}$	Lookup table distinguisher		

CPU 的服务器上进行测试. 本文的代码已开源至网站[2)].

本文后续内容组织如下: 第 2 节介绍必要的预备知识, 第 3 节介绍本文的 NDAFL 框架, 第 4 和 5 节分别介绍 NDAFL 在 3 类分组密码上的应用以及对 Speck 改进的差分 – 神经攻击, 最后, 第 6 节对工作进行总结.

2　预备知识

2.1　符号说明

n 个二进制位的无符号整数称为一个 n 比特字. 记 X 为一个 n 比特字, 本文用 $X[0]$ 表示其最低位, 用 $X[n-1]$ 表示其最高位. 本文使用 $\oplus$ 表示两个字按位异或, 使用 & 表示两个字按位与, 使用 $\boxplus$ 表示两个 n 比特字进行模 2^n 的加法运算, 使用 $\lll$ ($\ggg$) 表示 n 比特的循环左移 (循环右移), 使用 % 表示取模运算. 使用 $X||Y$ 表示两个字的拼接, 其中 X 为高位, Y 为低位. 对于一对 n 比特字 (X, X'), 使用 $\Delta X = X \oplus X'$ 表示两个字的差分. 本文使用 $\Delta_{[i]}$ 表示只有第 i 比特为 1, 其余比特为 0 的一比特差分约束. 后续使用到的一些缩写的含义在表 2 中列出.

2.2　Speck, Simon 和 LEA 分组密码算法

Speck 和 Simon [14] 是美国国家安全局 (National Security Agency, NSA) 于 2013 年提出的两个轻量级分组密码家族, 用于在资源受限的设备上实现高效加密. Speck 和 Simon 是一系列的算法, 支持分组长度 32∼128. 两类算法中一个分组均由左右两个 n 比特的字组成, 记为 (L, R), 主密钥由 m 个 n 比特字组成, 一个 Speck (Simon) 算法实例按照 “分组长度/主密钥长度” 的方式可记为 Speck2n/mn (Simon2n/mn).

Speck 遵循 ARX 结构, 轮函数由模加、循环移位以及按位异或 3 种操作组成, 而 Simon 轮函数由按位与、循环移位以及按位异或组成. 记加密中第 i 轮的输入输出分别为 (L_i, R_i) 和 (L_{i+1}, R_{i+1}), 使用到的轮密钥为 k_i, 则两算法的轮函数如图 1 和 2 所示. Speck 轮函数中 A 和 B 为与具体成员相关的常数, 在 Speck32/64 中为 $A = 7, B = 2$, 在其他成员中均为 $A = 8, B = 3$. 对于 r 轮的 Speck 和 Simon 算法, 明文记为 $(P_L, P_R) = (L_0, R_0)$, 密文记为 $(C_L, C_R) = (L_r, R_r)$. 特别地, 对 Speck 算法, 我们记最后一轮模加操作的右输入分支为 C_Y, 其取值可以由密文直接得到 $C_Y = (C_L \oplus C_R) \ggg B$.

LEA 分组密码算法 [15] 在 2013 年被提出并成为韩国的国家标准, 于 2019 年被 ISO 标准化. LEA 分组大小为 128, 密钥大小可以为 128, 192 和 256. LEA 为 ARX 类密码, 分组分为 4 个 32 位字 (X^0, X^1, X^2, X^3). 记第 i 轮输入为 $(X_i^0, X_i^1, X_i^2, X_i^3)$, 输出为 $(X_{i+1}^0, X_{i+1}^1, X_{i+1}^2, X_{i+1}^3)$, 其轮函数如图 3, 其中 $RK_i^0 \sim RK_i^5$ 为 6 个 32 位子密钥.

2.3　神经区分器

在文献 [1] 中, 对 Speck32/64 算法, Gohr 训练深度残差神经网络在差分约束下成功地构造出差分 – 神经区分器, 下文简称为神经区分器, 记为 $\mathcal{ND}$. 选定一个明文差分 Δ, 神经网络被训练来区分由满

2) https://github.com/nobosx/NDAFL.git.

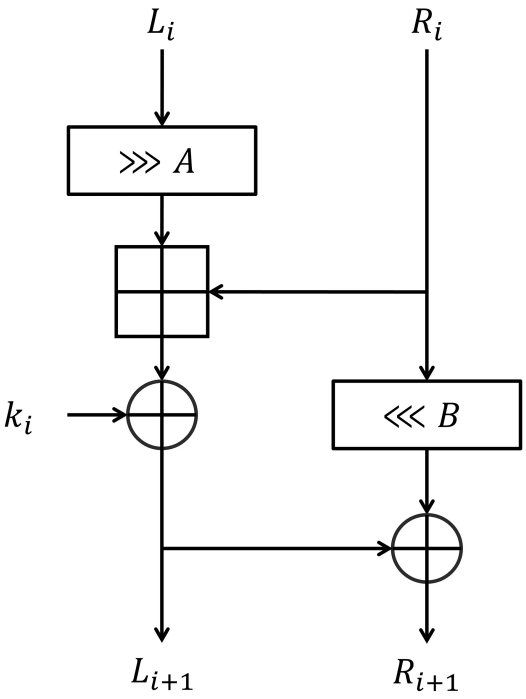

图 1　Speck 轮函数.

Figure 1　Speck round function.

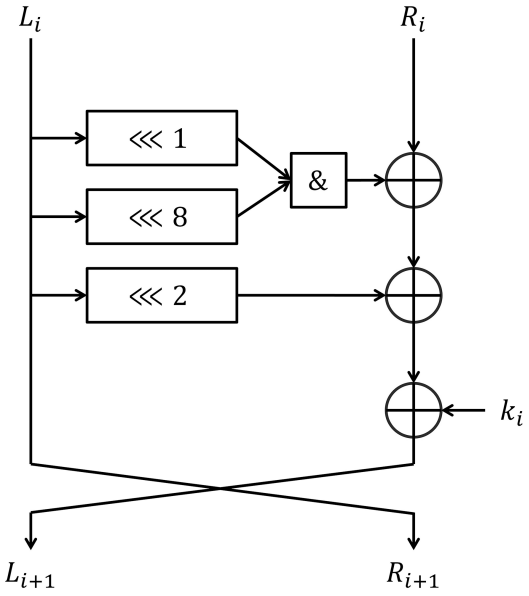

图 2　Simon 轮函数.

Figure 2　Simon round function.

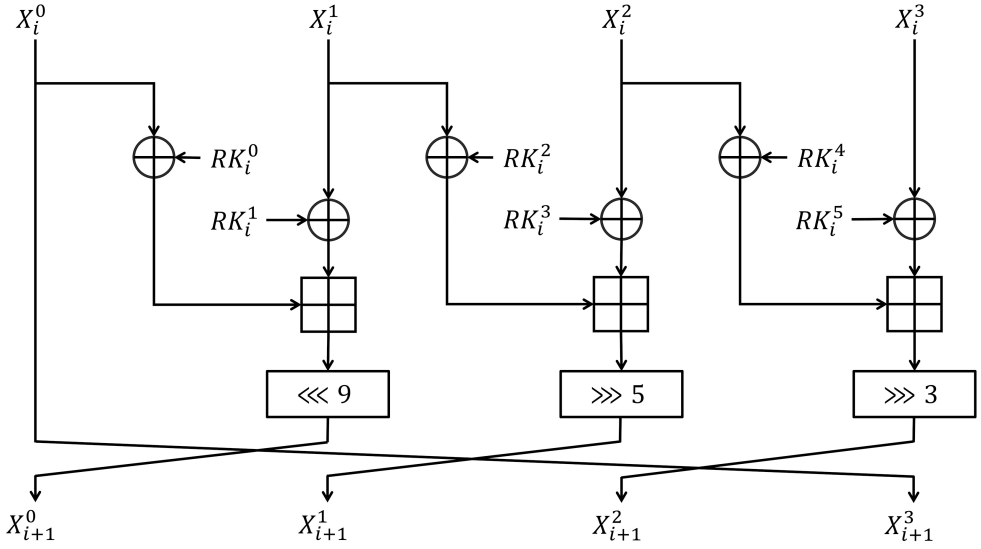

图 3　LEA 轮函数.

Figure 3　LEA round function.

足该差分的明文对加密得到的密文对以及随机密文对, 其输入为一对密文 $(C, C') = (C_L||C_R, C'_L||C'_R)$, 对应明文为 (P, P'). 对每个样本, 正例对应分类标签 $Y = 1$, 负例对应标签 $Y = 0$, 其中

$$Y = \begin{cases} 1, & P \oplus P' = \Delta, \\ 0, & P \oplus P' \text{ 满足随机分布}. \end{cases} \tag{1}$$

对于每个输入样例, 神经网络输出分类结果 $Z \in (0, 1)$. 如果 $Z > 0.5$ 认为样本来自正例, 否则认为样

本来自负例.

Gohr 使用的残差网络结构如图 4 所示. 网络输入为 4 个 n 比特字 $C_L||C_R||C'_L||C'_R$, 表示为 $4\times n$ 的矩阵. 输入通过转置变形, 依次经过初始卷积、残差卷积、全连接层后得到最终分类结果. 初始卷积包含 32 个核大小 $\mathrm{ks}=1$ 的一维卷积核, 用于对 $n\times 4$ 的输入在 n 比特字上逐比特提取特征. 残差卷积包含 m 个卷积块, 每个卷积块中依次进行 2 层 32 个核、核大小 $\mathrm{ks}=3$ 的一维卷积, 用于提取相邻比特间的特征. 同时每个卷积块的输入通过加法反馈到输出上, 形成残差结构, 有助于解决神经网络的退化现象. 全连接层包含两层宽度为 $d_1=d_2=64$ 的隐藏层, 最终连接至神经元数量为 1 的输出层, 经过 Sigmoid 激活函数得到取值位于 $(0,1)$ 区间的网络输出 Z. 每个卷积层以及全连接隐藏层后都会进行批标准化以及 ReLU 函数激活.

Gohr 分别使用大小为 10^7 的训练集以及 10^6 的验证集训练神经区分器, 共训练 200 个轮次. 训练过程使用循环学习率, 每 10 个训练轮次, 学习率由 2×10^{-3} 线性降至 10^{-4}. 每个训练轮次中数据以 5000 为一组进入神经网络. 训练结束后, 使用大小为 10^6 的测试集测试神经区分器的准确率, 若最终准确率大于 0.5, 则说明得到了一个有效的区分器. 以上构建的每个数据集均含有约一半正例以及一半负例. 后续实验中, 我们使用深度为 10 的网络以及与文献 [1] 相同的训练过程获得神经区分器.

Gohr 对 Speck32/64 算法得到了有效的 5~8 轮神经区分器. 与对应轮数下基于全差分分布表 DDT 构造的区分器 (记为 $\mathcal{DD}$) 相比, 神经区分器均达到了更高的区分准确率.

2.4　差分 – 神经攻击

基于神经区分器, Gohr 在文献 [1] 中构造了针对 Speck32/64 的密钥恢复攻击, 称为差分 – 神经攻击. 在 r_2 轮的 $\mathcal{ND}$ 前接概率为 p 的 r_1 轮经典差分路线, 在最后猜测一轮子密钥, 可以形成一个 r_1+r_2+1 轮的密钥恢复攻击. 特别地, 对于 Speck 算法, 由于其轮函数中密钥异或操作在非线性操作之后, 前置差分可以免费向前扩展一轮, 形成 $1+r_1+r_2+1$ 轮的攻击.

为增强区分信号, 差分 – 神经攻击中使用了中性比特技术 [16]. 使用前置差分中存在的 n 个中性比特, 一个明文对可以扩展为含有 2^n 个明文对的明文结构, 这些明文对可以一同以概率 p 通过前置差分. 这样一个通过前置差分的明文结构称为正确结构, 其所有明文对对应的密文对都为 $\mathcal{ND}$ 的正例.

进行密钥恢复时, 对于一个明文结构, 攻击者收集对应的密文结构并猜测最后一轮子密钥. 对于密钥猜测 kg, 攻击者将结构中所有密文对在 kg 下解密一轮并送入 $\mathcal{ND}$. 记密文结构中有 N 个密文对, 在 kg 下 $\mathcal{ND}$ 的 N 个输出为 $Z_i^{\mathrm{kg}}, i\in\{1,2,\ldots,N\}$, 使用如下公式计算密钥猜测 kg 的得分 v_{kg}:

$$v_{\mathrm{kg}}=\sum_{i=1}^{N}\log_2\left(\frac{Z_i^{\mathrm{kg}}}{1-Z_i^{\mathrm{kg}}}\right). \tag{2}$$

当明文结构为正确结构且 kg 猜测正确时, 对应 $\mathcal{ND}$ 的输入为正例, $\mathcal{ND}$ 的输出值 Z_i^{kg} 较高. 如果一个密钥猜测的得分 v_{kg} 超过某个阈值 c, 攻击者就认为 kg 为正确密钥.

2.5　比特敏感性测试

Chen 等 [17] 提出了比特敏感性测试 BST (bit sensitivity test) 的方法, 用于检测神经区分器输入的各比特是否含有区分信息. 其思想是去除 $\mathcal{ND}$ 的输入中与某一比特相关的分布信息, 如果发现 $\mathcal{ND}$ 在测试集上的准确率明显降低, 则说明该比特对区分影响较大, 即含有区分信息, 称这些比特为信息比特, 否则为非信息比特. BST 的具体过程见算法 1. 对于一个神经区分器, 给定一个敏感度阈值 t, 该算法会返回识别到的信息比特集合 $\mathcal{IB}$.

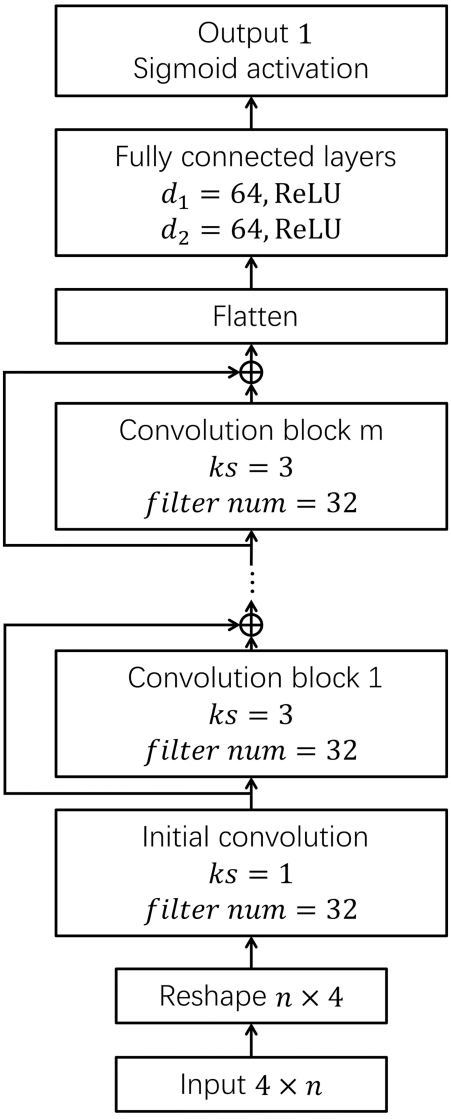

图 4　神经区分器网络结构.

Figure 4　Network architecture of neural distinguishers.

3　神经区分器辅助的特征检测框架

本节提出神经区分器辅助的特征检测框架 NDAFL (neural distinguisher-aided feature location). 我们首先介绍框架的总体思想, 之后分别介绍框架中两个关键的步骤.

3.1　总体思想

为回答第 1 节的问题, 一个很自然的想法是将神经网络的输入输出进行打表, 将神经区分器的访问方式由原来的计算式 (由神经网络计算得到输出) 转换为存储式 (由访问查找表得到输出). 但由于神经网络巨大的输入空间, 该途径无法直接实际实现.

为减少查找表的输入空间, 我们考虑使用比特敏感性测试先过滤掉神经网络输入中的部分冗余比特. 从对 Speck32/64 神经区分器的比特敏感性测试结果发现, 神经网络的输入密文对中, 一部分比特位置 (那些非信息比特) 的取值对神经网络的区分无影响 (或影响很小), 进而可以只考虑在信息比特上建立神经区分器的输入输出查找表. 然而, 即使只考虑那些信息比特, 对 Speck32/64 的神经区分器

算法 1 比特敏感性测试[17].

输入: 神经区分器 $\mathcal{ND}$; $\mathcal{ND}$ 网络输入比特集合 I; $\mathcal{ND}$ 测试集 $M=\{(X_j,Y_j), j\in\{1,\ldots,m\}\}$, 含有 m 个测例; 比特敏感度阈值 t.

输出: $\mathcal{ND}$ 的信息比特集合 $\mathcal{IB}$.

初始信息比特集合置空 $\mathcal{IB}\Leftarrow\{\}$;
测试 $\mathcal{ND}$ 在 M 上的准确率 acc;
for $\mathcal{ND}$ 的每个输入比特 $i\in I$ **do**
    构造新的测试集 $M'\Leftarrow\{\}$;
    for $j\in\{1,\ldots,m\}$ **do**
        生成一比特随机掩码 $\eta\in\{0,1\}$;
        对样例输入 X_j, 掩盖第 i 比特分布信息: $X'_j\Leftarrow X_j\oplus(\eta\ll i)$;
        $M'\Leftarrow M'\cup\{(X'_j,Y_j)\}$;
    end for
    测试 $\mathcal{ND}$ 在 M' 上的区分准确率 acc_i;
    计算比特 i 的敏感度为 $\mathrm{sen}_i\Leftarrow\mathrm{acc}-\mathrm{acc}_i$;
    if $\mathrm{sen}_i>t$ **then**
        $\mathcal{IB}\Leftarrow\mathcal{IB}\cup\{i\}$;
    end if
end for
返回 $\mathcal{IB}$.

识别到的信息比特数量仍然很多 (不少于 32 比特), 无法以较低的成本进行建表.

为解决上述问题, 通过 3.2 小节的结果我们将说明, 对同一个密码算法, 通过改变神经网络的输入形式, 检测到的信息比特数量 (即查找表的输入空间) 可以进一步降低至可接受的水平, 而包含的区分信息量不会减少. 至此我们就可以在信息比特上以实际的复杂度建立神经网络的替代查找表区分器, 同时达到和原神经网络基本相当的区分能力.

由此我们提出神经区分器辅助的特征检测框架 NDAFL, 总体流程见算法 2. 值得注意的是, 整个框架可以以一种黑盒的方式工作, 分析者无需知道算法 E_r 的轮函数信息, 只需知道密码算法的接口, 即神经区分器输入的状态大小. 然而, 密码算法的轮函数结构信息可以帮助更好地设计神经区分器的输入形式 I. 整个框架成功的关键有两点, 一是要针对 E_r 训练出有效的神经区分器, 二是要设计合适的输入形式 I, 保证检测出的信息比特数量不会过多. 针对第 1 点需要说明的是, 训练出的神经区分器的准确率不是影响最终效果的最关键因素, 因为在该框架中我们只是利用神经区分器帮助识别信息比特同时排除掉输入中大量无关比特. 即使神经网络只经过简单的训练 (例如只训练了较少的几个轮次), 最终也可能得到较好的查找表区分器[3)]. 与已有的深度学习辅助密码分析一样, 训练神经区分器的明文差分约束需要仔细地选择, 因为这会极大地影响最终密文对分布的偏差程度. 对于第 2 点 I 的设计, 则可能有一定的技巧性, 本文会在 3.2 小节给出几条 I 的设计原则.

关于算法 2 第 5 行阈值 t 的设置, 则取决于 $\mathcal{ND}$ 的准确率. 如果 $\mathcal{ND}$ 的准确率较高 (例如大于 0.55), 我们可以适当忽略那些含有很少特征的比特而将 t 设置得大一些. 而当 $\mathcal{ND}$ 的准确率较低时 (例如对 Speck32/64 的 8 轮神经区分器, 其准确率只有 0.5142), 则可以设置较低的 t.

3.2　精简信息比特数量

保证检测出的信息比特数量不会过多是框架成功的关键. 我们这里以 Speck 算法为例说明, 设计合适的神经网络输入形式 I, 可以有效精简信息比特的数量.

对 Speck32/64 算法, 使用原始密文对 $I=C||C'=C_L||C_R||C'_L||C'_R$ 作为输入, 我们训练了 5∼8 轮神经区分器并检测信息比特, 训练这些区分器使用的明文差分约束与文献 [1] 中相同, 为 $\Delta=$

3) 例如, 我们针对 7 轮 Speck32/64 算法只训练一个轮次, 得到 $\mathcal{ND}$ 准确率为 0.579, 尚未达到收敛时的 0.611. 基于此构建出的查找表区分器准确率可以达到 0.595. 相关代码同样开源至代码仓库.

算法 2 神经区分器辅助的特征检测框架 NDAFL.

输入: r 轮加密算法 E_r; 对于 E_r 合适的神经网络输入形式 I; 允许的信息比特数量的上界 T.
输出: E_r 的查找表区分器 $\mathcal{LT}$ 或 Failure.
对算法 E_r, 构建具有输入形式 I 的神经区分器训练集 M_{train} 和测试集 M_{test};
使用数据集 $(M_{\text{train}}, M_{\text{test}})$ 训练一个神经区分器 $\mathcal{ND}$, 并测试准确率为 acc;
if acc ≈ 0.5 **then**
    返回 Failure;
end if
由 $\mathcal{ND}$ 的准确率 acc 确定比特敏感性阈值 t;
使用 $\mathcal{ND}, I, M_{\text{test}}, t$ 进行比特敏感性测试, 获得信息比特集合 $\mathcal{IB}$;
if $|\mathcal{IB}| > T$ **then**
    返回 Failure;
end if
对算法 E_r, 在信息比特 $\mathcal{IB}$ 上建立查找表区分器 $\mathcal{LT}$;
返回 $\mathcal{LT}$.

$(0x40, 0x0)$[4]. 之后我们尝试改变神经网络输入的格式 I, 聚焦于 Speck 最后一轮的模加, 将输入改为 $I = \Delta C_L||C_L||\Delta C_Y||C_Y$, 其中 $C_Y = (C_L \oplus C_R) \ggg B$ 即为模加右侧输入分支. 容易看出前后两种输入形式是线性等价的, 可以由异或和移位操作互相导出. 在新的输入形式下, 我们重新训练了神经区分器并检测信息比特. 这些神经区分器及比特敏感性测试结果见表 3. 比特敏感性测试在含有 10^7 个样本的测试集上进行. 对原始密文对, 由于 C 和 C' 是对称的, 故 C'_L (C'_R) 上的信息比特与 C_L (C_R) 上的信息比特相同, 表 3 只给出了 C_L 和 C_R 上的信息比特. 我们有如下发现.

(1) 两种输入形式下相同轮数的区分器准确率相当. 这验证了两种输入形式的线性等价性, 可以认为二者含有等量的区分特征.

(2) $\Delta C_L||C_L||\Delta C_Y||C_Y$ 输入下未在 C_L 字上检测到信息比特. 这是由于若将最后一轮未知的轮密钥视为均匀随机分布, 其与模加的输出异或后得到 C_L 字, 这只能保留 C_L 差分即 ΔC_L 上的特征, 而除差分信息外密文对中 C_L 本身的取值可以看作均匀随机分布, 不含任何其他区分信息.

(3) $\Delta C_L||C_L||\Delta C_Y||C_Y$ 输入下检测的信息比特数量明显少于原输入形式 $C_L||C_R||C'_L||C'_R$.

注意到改变输入后信息比特数量明显减少, 我们这里给出解释. 假设神经网络的区分中真正用到的字为最后一轮模加处的输出差分、右分支输入差分以及右分支输入真实值, 即 $(\Delta C_L, \Delta C_Y, C_Y)$ 3 个字[5]. 考虑以下两种情况.

(1) $\mathcal{ND}$ 利用到某一比特差分 $\Delta C_L[i]$ 或 $\Delta C_Y[i]$. 若 $\mathcal{ND}$ 利用了 $\Delta C_L[i]$, 由于 $\Delta C_L[i] = (C_L \oplus C'_L)[i]$, 原始密文对输入的神经网络中会有 $C_L[i]$ 和 $C'_L[i]$ 两比特被识别为信息比特. 若 $\mathcal{ND}$ 利用了 $\Delta C_Y[i]$, 设 n 为 Speck 算法字长, $j = (i + B)\%n$, 由于 $\Delta C_Y[i] = (C_L \oplus C'_L \oplus C_R \oplus C'_R)[j]$, 原始密文对输入的神经网络中会有 $C_L[j]$, $C'_L[j]$, $C_R[j]$ 和 $C'_R[j]$ 这 4 比特被识别为信息比特.

(2) $\mathcal{ND}$ 利用到 C_Y 真实值的某一比特 $C_Y[i]$. 仍设 $j = (i + B)\%n$, 由于 $C_Y[i] = (C_L \oplus C_R)[j]$, 原始密文对输入的神经网络中会有 $C_L[j]$ 和 $C_R[j]$ 两比特被识别为信息比特.

以上每种情况都会使得 $C_L||C_R||C'_L||C'_R$ 下信息比特数量多于 $\Delta C_L||C_L||\Delta C_Y||C_Y$. 由以上分析, 我们可以认为改变神经网络的输入形式实际上精简了密码算法区分特征的表示, 而总的可利用的特征数量不变. 至此, 信息比特的数量足够少, 我们可以在实际的复杂度下建立相应的查找表.

神经网络输入形式设计原则. 由上述分析, 我们给出一些神经网络输入形式 I 的设计原则.

(1) 由密文对尽可能地逆向解密, 将区分特征更清晰地暴露给神经网络[6]. 许多密码算法在轮函数

4) 此明文差分对应首轮模加处最高位的差分, 使得首轮差分可以确定性地传播.

5) 文献 [5] 中的研究表明, 神经区分器的区分机理很可能是使用已知的 C_Y 真实值修正差分传播的概率, 故这一假设是合理的.

6) 文献 [18] 在分析 Speck 和 Simon 算法时也使用了逆向计算密文的思想.

表 3　Speck32/64 不同神经区分器的信息比特.

Table 3　Informative bits of different neural distinguishers on Speck32/64.

$\#R$	Accuracy	t	$\mathcal{IB}$	$\lvert\mathcal{IB}\rvert$
5	0.9274	0.01	$C_L[0\sim5,8\sim14], C_R[2\sim5,9\sim14]$	46
	0.9287	0.01	$\Delta C_L[0\sim3,8\sim12], \Delta C_Y[0\sim3,7\sim12], C_Y[0\sim2,8\sim11]$	26
6	0.7861	0.01	$C_L[0\sim5,8\sim14], C_R[2\sim5,10\sim14]$	44
	0.7866	0.01	$\Delta C_L[0\sim3,8\sim12], \Delta C_Y[0\sim3,8\sim12], C_Y[0\sim2,8\sim11]$	25
7	0.6109	0.01	$C_L[2\sim5,9\sim14], C_R[4\sim5,11\sim14]$	32
	0.6116	0.01	$\Delta C_L[2\sim3,9\sim12], \Delta C_Y[2\sim3,9\sim12], C_Y[8\sim11]$	16
8	0.5139	0.001	$C_L[2\sim7,11\sim14], C_R[3\sim7,12\sim14]$	36
	0.5142	0.001	$\Delta C_L[2\sim5,11\sim12], \Delta C_Y[2\sim5,10\sim12], C_Y[1\sim4,10\sim11]$	19

算法 3 在信息比特 $\mathcal{IB}$ 上建立查找表区分器 $\mathcal{LT}$.

输入: r 轮加密算法 E_r; 神经区分器的明文差分约束 Δ; 神经区分器的信息比特 $\mathcal{IB}$; 平均表项计数 α.

输出: r 轮查找表区分器 $\mathcal{LT}$.

构建含有 $2^{|\mathcal{IB}|}$ 个表项的空列表 $\mathcal{LT} \Leftarrow []$;
for $i \in \{0,1,\ldots,2^{|\mathcal{IB}|}-1\}$ **do**
    $\mathcal{LT}[i] \Leftarrow 0$;
end for
$N \Leftarrow \alpha \times 2^{|\mathcal{IB}|}$;
for $i \in \{1,2,\ldots,N\}$ **do**
    随机产生一个明文对 $(P, P' \Leftarrow P \oplus \Delta)$;
    随机产生一个主密钥, 使用 E_r 将 (P,P') 加密 r 轮得到密文对 (C,C');
    提取 (C,C') 在 $\mathcal{IB}$ 上的取值 V;
    $\mathcal{LT}[V] \Leftarrow \mathcal{LT}[V]+1$;
end for
返回 $\mathcal{LT}$.

最后存在一些线性操作, 由原始密文在未知密钥的情况下可以逆向进行一部分解密计算, 这有助于减少区分特征的扩散.

(2) 在神经区分器输入中将差分和真实值分离. 将一对密文的输入形式 (C,C') 转换为密文差分加其中一个密文的输入 $(\Delta C = C \oplus C', C)$. 由已有工作 [1,5] 的分析, 神经区分器的区分利用的主要还是差分特征, 而非差分的特征可以被用来修正差分概率分布, 因此将差分信息从密文对中单独提供出来有助于精简区分特征在区分器输入中的分布.

3.3　建立替代查找表区分器

对一个神经区分器 $\mathcal{ND}$, 一旦检测到的信息比特数量足够少, 我们就可以建立替代神经网络的查找表区分器 $\mathcal{LT}$, 在后续的密码分析过程中摆脱对神经网络的依赖. 为构造 $\mathcal{LT}$, 我们仍采用黑盒的方式, 构造一定量的正例加密密文对, 只统计密文对在信息比特上的各取值的频率, 以此估计正例在信息比特上的取值概率分布, 进一步可进行正负例的区分. 由于信息比特含有 $\mathcal{ND}$ 利用到的绝大部分区分信息, 使用足量的加密数据, 我们期望 $\mathcal{LT}$ 可以达到接近 $\mathcal{ND}$ 的准确率.

建立 $\mathcal{LT}$ 的过程在算法 3 中给出. 查找表的空间复杂度为 $2^{|\mathcal{IB}|}$ 个计数表项, 若限制查找表的平均计数为 α, 则构建查找表所需的数据量 $N = \alpha \times 2^{|\mathcal{IB}|}$, 算法 3 的计算复杂度为 $O(\alpha \times 2^{|\mathcal{IB}|})$.

一旦查找表构建完成, 我们就可以使用 $\mathcal{LT}$ 估计正例在 $\mathcal{IB}$ 上的概率分布, 进而进行区分. 给定一个输入样例 (C,C'), 我们首先提取其在 $\mathcal{IB}$ 上的取值 V. 对正例数据, $\mathcal{LT}[V]$ 为取值 V 的计数频数, 而计数总数为 N, 故取值 V 在正例下的先验概率为 $\hat{\Pr}(V|Y=1) = \frac{\mathcal{LT}[V]}{N} = \frac{\mathcal{LT}[V]}{\alpha \times 2^{|\mathcal{IB}|}}$. 而对负例数据即

随机均匀分布, 取值 V 的先验概率为 $\Pr(V|Y=0)=\frac{1}{2^{|\mathcal{IB}|}}$. 我们认为正例和负例出现的概率均为 0.5 即 $\Pr(Y=1)=\Pr(Y=0)=0.5$, 因此可以估计输入样例属于正例的后验概率为

$$\begin{aligned}\Pr(Y=1|V)&=\frac{\Pr(V|Y=1)\times\Pr(Y=1)}{\Pr(V|Y=1)\times\Pr(Y=1)+\Pr(V|Y=0)\times\Pr(Y=0)}\\&=\frac{\Pr(V|Y=1)}{\Pr(V|Y=1)+\Pr(V|Y=0)}\\&\approx\frac{\hat{\Pr}(V|Y=1)}{\hat{\Pr}(V|Y=1)+\Pr(V|Y=0)}\\&=\frac{\mathcal{LT}[V]}{\mathcal{LT}[V]+\alpha}.\end{aligned}\tag{3}$$

如果上式计算得到 $\Pr(Y=1|V)>0.5$, 等价于满足 $\mathcal{LT}[V]>\alpha$, 则认为输入样例为正例并输出区分结果 $Y=1$, 否则输出 $Y=0$.

在使用 $\mathcal{LT}$ 代替 $\mathcal{ND}$ 进行密钥恢复攻击时, 我们使用式 (3) 代替式 (2) 中的 Z_i^{kg} 以进行密钥得分的整合. 对于含有 m 个密文对的密文结构, 在密钥猜测 kg 下对各密文对进行解密, 得到 $\mathcal{LT}$ 信息比特上的取值 $V_1^{\mathrm{kg}},\ldots,V_m^{\mathrm{kg}}$, 并使用下列公式计算密钥得分:

$$v_{\mathrm{kg}}=\sum_{i=1}^{m}\log_2\left(\frac{\mathcal{LT}[V_i^{\mathrm{kg}}]}{\alpha}\right).\tag{4}$$

3.4　不同区分器复杂度比较

神经区分器 $\mathcal{ND}$ 与替代查找表区分器 $\mathcal{LT}$ 作为相同区分场景下的两类不同的区分器, 其本质均是识别并利用输入密文对中比特分布的偏差信息. 两类区分器的区别在于实现区分任务的方式. NDAFL 框架可以看作是对区分器计算复杂度与存储空间之间的权衡, 将区分器的区分方式由计算式转换为查表式, 通过增加不多的存储复杂度, 减少区分器的计算成本.

对于在 $\mathcal{IB}$ 上建立的参数为 α 的查找表 $\mathcal{LT}$, 其计数表项若使用 64 位无符号整数存储, 则查找表需要的存储空间为 $2^{|\mathcal{IB}|}$ 个 64 位表项, 即 $2^{|\mathcal{IB}|+3}$ Bytes 空间. 建立查找表的离线计算复杂度为 $O(\alpha\times 2^{|\mathcal{IB}|})$, 进行区分任务时所需的计算复杂度仅为一次查表操作, 这在攻击中相比解密操作的计算成本往往可忽略不计.

对于基于深度神经网络构建的神经区分器 $\mathcal{ND}$, 其所需存储空间以及计算复杂度则取决于具体的网络规模. 以 Speck32/64 的深度为 10 的残差网络区分器 [1] 为例, 其参数量约为 $2^{16.6}$. 若参数均使用 32 位浮点数存储, 所需存储约为 $2^{18.6}$ Bytes. 依据其网络结构, 一个样本的区分任务大约需要进行 $2^{19.96}$ 次 32 位浮点数乘法运算 (未计入浮点数加法以及激活函数的计算复杂度).

为直观地评估 NDAFL 框架的效果, 我们仍以 Speck32/64 为例, 在相同测试集上分别比较 $\mathcal{ND}$ 与第 4 节构建的 $\mathcal{LT}$ 的区分处理时间以及存储空间, 结果如表 4. 表 4 中的结果均在大小为 10^7 的测试集上测得, 所有实验均使用 Python 代码运行在单核 CPU 环境中. 表 4 中时间成本即为各区分器处理完所有 10^7 个样本所花费的时间. 对于查找表区分器, 时间成本包括从密文对中提取查找表输入值 V、访问查找表以及判断分类结果的时间. 对于分组状态更大的密码算法, $\mathcal{ND}$ 的参数量会更多, 因而计算及存储复杂度更大, 而 $\mathcal{LT}$ 的复杂度不会随密码算法状态增大而提高.

由表 4, 通过 NDAFL, Speck32/64 区分器的处理效率可以大约提高为原来的 $754.02/0.93=2^{9.66}$ 倍[7], 而文献 [7] 中重建的简化区分器 DL+FCNN, 仍需要后续全连接层计算开销, 区分效率的改进只有 $2^{1.35}$ 倍. 关于存储复杂度, 在表 4 对 Speck32/64 的查找表区分器中, 存储复杂度最大的区分器为

7) 表 4 使用深度为 10 的 $\mathcal{ND}$. 即使使用深度为 1 的 $\mathcal{ND}$, 本框架也可以将区分效率提升为至少原来的 $2^{9.66}/10=2^{6.34}$ 倍.

表 4　Speck32/64 不同区分器计算及存储复杂度比较.

Table 4　Comparisons of different distinguishers' time and memory complexities on Speck32/64.

Distinguisher	5 rounds		6 rounds		7 rounds		8 rounds	
	$\mathcal{ND}$	$\mathcal{LT}$	$\mathcal{ND}$	$\mathcal{LT}$	$\mathcal{ND}$	$\mathcal{LT}$	$\mathcal{ND}$	$\mathcal{LT}$
Time cost (s)	737.20	0.89	754.02	0.93	739.23	0.87	745.77	0.79
Memory (MBytes)	0.57	512	0.57	256	0.57	128	0.57	32

5 轮 $\mathcal{LT}$, 其对应 26 个信息比特, 所需要的存储也仅有 2^{26+3} Bytes 即 512 MB. 即使考虑本文构造的占用空间最大的查找表, 对应 29 个信息比特, 其存储复杂度为 2^{29+3} Bytes 即 4 GB, 这对于一台现代计算机来说是完全可以接受的. 而目前差分 – 神经攻击的瓶颈往往在于计算复杂度而非存储复杂度, 因此本文提出的增强框架有其实际的应用价值.

4　Speck, Simon 和 LEA 算法上的应用

本节我们将 NDAFL 框架应用于 Speck 系列算法、Simon 系列算法以及 LEA 算法. 对每个密码算法, 我们首先选择较好的输入差分, 训练区分轮数尽可能长的神经区分器, 之后应用本文的框架将对应的神经区分器转化为查找表区分器, 最后在相同测试集上评估各类区分器的区分能力. 为验证本框架的性能, 我们希望使用 DDT 作为 Baseline. 然而, 对于分组大于 32 比特的密码算法, 由于状态空间过大, 我们无法构建起实际的 DDT. 作为替代, 我们在对应密码算法上只使用密文对的差分 $C \oplus C'$ 作为输入训练神经区分器, 称这样只使用差分特征的神经区分器为神经差分区分器, 记为 $\mathcal{NDD}$ (neural differential distinguisher). 我们希望 $\mathcal{NDD}$ 能够较好地模拟 DDT 的分布, 并使用 $\mathcal{NDD}$ 作为 Baseline 进行后续实验. 我们在 10^7 的训练集中训练各神经区分器 10 个轮次. 对于每个区分器, 我们在含有 10^7 个样例的测试集上进行测试[8], 测试集约含一半正例及一半负例. 我们分别报告每个区分器的准确率、真阳性率以及真阴性率[9]. 对查找表区分器, 我们使用记号 $\mathcal{LT}_{\alpha}^{|\mathcal{IB}|}$ 说明建表的设置, 上下标分别为建立 $\mathcal{LT}$ 使用的信息比特数量以及 α 值.

NDAFL 应用于大状态分组密码. 对 Speck48, Speck64, Speck96 和 Speck128 这些分组状态更大的算法, 直觉上这些成员的神经区分器输入空间更大, 对应的信息比特数量应该更多. 事实上我们发现, 如果对这些算法成员训练轮数较长的神经区分器, 这些区分器的准确率不会很高, 这意味着虽然算法分组状态很大, 但神经网络可利用到的偏差较大的区分特征是有限的, 这些特征很可能也分布在较少的比特区间中, 因此我们仍可以使用 NDAFL 框架成功构建相应的查找表区分器.

4.1　Speck 算法上的应用

Speck 家族包含分组长度为 32, 48, 64, 96 和 128 的算法成员, 我们对每个分组长度的成员都进行了实验. 由 3.2 小节的分析, 我们使用 $I = \Delta C_L||C_L||\Delta C_Y||C_Y$ 作为神经网络的输入[10]. 对于输入差分, 我们选择首轮可确定性传播一轮的明文差分 $\Delta = (\Delta_{[A-1]}, 0)$.

对 Speck32/64, 我们使用差分约束 $(\Delta_{[A-1]}, 0) = (0x40, 0)$ 训练了 5∼8 轮神经区分器, 这些神经区分器的准确率以及信息比特在表 3 中已经给出. 使用 NDAFL 框架, 我们成功地对 5∼8 轮 Speck32/64 算法建立了相应的查找表区分器. 各个区分器的比较见表 5. 其中对于 7 轮和 8 轮查找表, 为提升准确率, 我们使用比表 3 中更多的信息比特. 表 5 中基于 DDT 的区分器来自文献 [1]. TPR 表示真阳性

8) 使用 10^7 大小的测试集, 能够保证本节报告准确率的标准差不超过 $\sqrt{0.5 \times (1-0.5)/10^7} < 0.00016$, 因此基本可以认为测试数据的小数点后 3 位是准确的.

9) 真阳性率为测试集中所有正例中被区分器正确分类的比例. 真阴性率为测试集中所有负例中被区分器正确分类的比例.

10) 由于 C_L 上没有信息比特, 使用 $\Delta C_L||C_L||\Delta C_Y||C_Y$ 与 $\Delta C_L||\Delta C_Y||C_Y$ 作为输入实验结果是相同的.

表 5　Speck32/64 不同区分器比较.

Table 5　Comparisons of different distinguishers on Speck32/64.

$\#R$	Type	Accuracy	TPR	TNR	$\#R$	Type	Accuracy	TPR	TNR
5	$\mathcal{DD}$	0.911	0.877	0.947	7	$\mathcal{DD}$	0.591	0.543	0.640
	$\mathcal{LT}_{2^6}^{26}$	0.9228	0.8977	0.9480		$\mathcal{LT}_{2^{10}}^{24}$	0.6063	0.5572	0.6554
	$\mathcal{ND}$	0.9285	0.9031	0.9540		$\mathcal{ND}$	0.6113	0.5501	0.6725
6	$\mathcal{DD}$	0.758	0.680	0.837	8	$\mathcal{DD}$	0.512	0.496	0.527
	$\mathcal{LT}_{2^6}^{25}$	0.7798	0.7191	0.8404		$\mathcal{LT}_{2^{16}}^{22}$	0.5137	0.5098	0.5177
	$\mathcal{ND}$	0.7864	0.7211	0.8518		$\mathcal{ND}$	0.5142	0.4970	0.5314

表 6　Speck 算法不同区分器比较.

Table 6　Comparisons of different distinguishers on Speck.

Cipher	$\#R$	Type	Accuracy	TPR	TNR	Cipher	$\#R$	Type	Accuracy	TPR	TNR
Speck48	7	$\mathcal{NDD}$	0.5452	0.4761	0.6142	Speck96	8	$\mathcal{NDD}$	0.5891	0.5711	0.6071
		$\mathcal{LT}_{2^9}^{29}$	0.5455	0.4705	0.6206			$\mathcal{LT}_{2^8}^{26}$	0.5985	0.5895	0.6077
		$\mathcal{ND}$	0.5516	0.4493	0.6539			$\mathcal{ND}$	0.6142	0.5726	0.6558
Speck64	7	$\mathcal{NDD}$	0.6164	0.5831	0.6497	Speck128	9	$\mathcal{NDD}$	0.6244	0.5598	0.6889
		$\mathcal{LT}_{2^8}^{28}$	0.6258	0.5970	0.6546			$\mathcal{LT}_{2^9}^{25}$	0.6513	0.6170	0.6856
		$\mathcal{ND}$	0.6371	0.5892	0.6850			$\mathcal{ND}$	0.6544	0.6024	0.7064

率, TNR 表示真阴性率.

对 Speck 其他成员, 我们也成功地将差分约束 $(\Delta_{[A-1]}, 0) = (0x80, 0)$ 下长轮数的神经区分器 $\mathcal{ND}$ 转换为对应的查找表区分器 $\mathcal{LT}$. 对于 Speck48, Speck64, Speck96 以及 Speck128 这 4 类算法, 我们最长分别可以训练出 7, 8, 8 以及 9 轮的神经区分器[11]. 各类区分器的比较见表 6. 对 Speck64 我们报告 7 轮区分器的结果.

表 5 和 6 中的结果符合预期. 对于 Speck 各个算法, 我们均建立了准确率高于 Baseline 且接近 $\mathcal{ND}$ 的 $\mathcal{LT}$. 由此可以说明, 对 Speck 算法, 我们成功地重建了各成员的神经区分器.

α 的选取. 我们以 Speck32/64 为例说明确定 α 的方法. 为确定合适的 α 值, 我们不断地增大 α (每次乘以 2) 并观察 $\mathcal{LT}$ 的准确率如何变化. 当准确率不再随 α 增大而提高时, 则说明找到了合适的 α 取值. 我们在表 7 中给出了建立查找表时尝试的 α 取值以及对应查找表区分器的准确率. 我们发现当 $\mathcal{ND}$ 的准确率较高 (区分轮数较低) 时, 使用很小的 α 值即可得到优于 DDT 区分器的查找表, 建表时间只需几十秒. 而对 8 轮 Speck32/64, 由于 $\mathcal{ND}$ 的准确率很接近 0.5, 说明正例概率分布很接近负例, 我们需要使用更大的 α 值即更多数据才能得到较好的 $\mathcal{LT}$.

4.2　Simon 和 LEA 算法上的应用

本小节我们将 NDAFL 框架应用于 Simon32, Simon64, Simon128 以及 LEA 算法中.

Simon 算法的区分器. 对 Simon 系列算法, 文献 [18] 已经证明, 在各轮密钥独立随机取值的假设下, 以一对密文作为输入, 一个 r 轮区分器的区分能力不会优于 $r-1$ 轮的 DDT. 即使如此, 我们仍然可以训练 r 轮的神经区分器以模拟 $r-1$ 轮 DDT. 特别是分组长度较大时, 建立实际的 DDT 是非常困难的, 此时可以训练对应的神经区分器以进行分析. 通过 NDAFL 框架, 我们希望可以方便地构建查找表以达到神经区分器的准确率, 进而模拟对应的 DDT.

11) 对 Speck64, 训练得到的 8 轮神经区分器准确率为 0.5140, 只有 3 个信息比特. 我们发现该神经区分器与一条相关性绝对值为 $2^{-4.157}$ 的差分 – 线性特征等价.

表 7　不同 α 设置下建立 $\mathcal{LT}_\alpha$ 的时间及其准确率.

Table 7　Constructing time and accuracy of $\mathcal{LT}_\alpha$ under different settings of α.

5-round Speck32/64			6-round Speck32/64			7-round Speck32/64			8-round Speck32/64		
α	Time cost (s)	Accuracy	α	Time cost (s)	Accuracy	α	Time cost (s)	Accuracy	α	Time cost (s)	Accuracy
2^0	7.92	0.8927	2^0	4.59	0.7134	2^0	2.11	0.5450	2^{10}	568.97	0.5123
2^1	15.31	0.9124	2^1	8.86	0.7465	2^2	7.72	0.5742	2^{12}	2243.00	0.5129
2^2	29.59	0.9189	2^2	17.23	0.7652	2^4	32.84	0.5960	2^{14}	8896.50	0.5137
2^3	53.70	0.9213	2^3	32.31	0.7736	2^6	137.70	0.6040	2^{16}	35143.61	0.5137
2^4	123.30	0.9222	2^4	67.66	0.7771	2^8	550.84	0.6057	2^{17}	70140.72	0.5138
2^5	238.09	0.9227	2^5	135.50	0.7792	2^{10}	2092.31	0.6063	2^{18}	130342.56	0.5138
2^6	446.28	0.9228	2^6	266.23	0.7798	2^{12}	3927.63	0.6062	–	–	–

表 8　Simon 和 LEA 算法不同区分器比较.

Table 8　Comparisons of different distinguishers on Simon and LEA.

Cipher	$\#R$	Type	Accuracy	TPR	TNR	Cipher	$\#R$	Type	Accuracy	TPR	TNR
Simon32	10	$\mathcal{DD}$	0.5203	0.5002	0.5404	Simon32	11	$\mathcal{LT}^{19}_{2^{15}}$	0.5179	0.5240	0.5118
								$\mathcal{ND}$	0.5178	0.4726	0.5630
Simon64	13	$\mathcal{LT}^{18}_{2^{12}}$	0.5175	0.4464	0.5887	Simon128	19	$\mathcal{LT}^{19}_{2^{11}}$	0.5203	0.3647	0.6759
		$\mathcal{ND}$	0.5192	0.4773	0.5610			$\mathcal{ND}$	0.5232	0.3841	0.6624
LEA	10	$\mathcal{NDD}$	0.5684	0.5178	0.6191	LEA	11	$\mathcal{NDD}$	0.5054	0.4407	0.5701
		$\mathcal{LT}^{27}_{2^9}$	0.5734	0.5493	0.5976			$\mathcal{LT}^{17}_{2^{15}}$	0.5071	0.4769	0.5372
		$\mathcal{ND}$	0.5881	0.5347	0.6415			$\mathcal{ND}$	0.5073	0.4545	0.5602

对 Simon 算法, 为执行 NDAFL 框架, 首先选择合适的输入形式 I, 为此使用类似文献 [2] 中 $(x, x', y \oplus y')$ 的形式. 给定一个 r 轮密文对 $(C, C') = (L_r||R_r, L'_r||R'_r)$, 由 Simon 算法的 Feistel 结构尽可能地进行逆向解密, 可以得到 $r-1$ 轮的左分支 $L_{r-1} = R_r, L'_{r-1} = R'_r$, 同时得到 $r-1$ 轮右分支的差分 $\Delta R_{r-1} = \Delta L_r \oplus F(R_r) \oplus F(R'_r)$, 其中 $F(x) = ((x \lll 1)\&(x \lll 8)) \oplus (x \lll 2)$ 为 Simon 的加密函数. 我们在实验中使用 $I = \Delta L_{r-1}||L_{r-1}||\Delta R_{r-1}$ 这 3 个字作为神经网络的输入. 对于明文差分约束, 基于 Simon 的轮函数结构, 我们选择在右分支字施加一比特差分, 使用 $\Delta = (0, \Delta_{[0]})$ 训练神经区分器[12]. 对于 Simon32, Simon64 以及 Simon128 这 3 类算法, 我们分别训练出 11 轮, 13 轮和 19 轮神经区分器. 各区分器在表 8 中给出. 表 8 中对 Simon32 基于 DDT 的区分器来自文献 [2].

LEA 算法的区分器. 对 LEA 算法, 我们仍然按照输入形式设计原则 1 由密文对进行逆向解密. 给定一个 r 轮加密的密文 $C = (X^0_r, X^1_r, X^2_r, X^3_r)$, 由 LEA 算法结构, 可以直接得到 $X^0_{r-1} = X^3_r$, 以及最后一轮 3 个模加的输出分别为 $X^0_r \ggg 9, X^1_r \lll 5, X^2_r \lll 3$. 再由设计原则 2 将真实值与差分分离, 最终在实验中使用的输入形式为

$$I = \Delta Y_0||\Delta Y_1||\Delta Y_2||\Delta Y_3||Y_0||Y_1||Y_2||Y_3, \tag{5}$$

$$Y_0 = X^0_{r-1}, Y_1 = X^0_r \ggg 9, Y_2 = X^1_r \lll 5, Y_3 = X^2_r \lll 3. \tag{6}$$

训练神经区分器时, 使用差分 $\Delta = (\Delta_{[31]}, \Delta_{[31]}, \Delta_{[31]}, \Delta_{[31]})$, 其可以确定性地传播一轮后变为一比特差分 $(0, 0, 0, \Delta_{[31]})$. 我们训练得到了 10 轮和 11 轮的区分器. 各区分器的比较见表 8.

对于 Simon 算法, $\mathcal{LT}$ 的准确率接近对应的 $\mathcal{ND}$, 且 Simon32 的 11 轮 $\mathcal{LT}$ 准确率接近 10 轮的 DDT, 这说明 r 轮 $\mathcal{LT}$ 可以较好地模拟 $r-1$ 轮的差分分布. 对于 LEA 算法, $\mathcal{LT}$ 均实现了高于

12) 事实上, Simon 加密具有旋转对称性, 在右分支字的任意位置上施加一比特差分都能训练出准确率基本相同的神经区分器.

Baseline 且接近 $\mathcal{ND}$ 的准确率. 特别地, 对 11 轮 LEA, 即使区分信号很弱 ($\mathcal{ND}$ 准确率只有 0.5073), 使用足够的数据, 我们的框架仍可以模拟出加密数据的分布偏差, 达到与 $\mathcal{ND}$ 相当的准确率.

5　改进 Speck 算法的差分 – 神经攻击

NDAFL 的一个直接应用即是改进已有的差分 – 神经攻击. 通过本文的框架, 将神经区分器的访问由原来的计算式转变为查表式, 避免了区分器带来的计算复杂度, 较长轮数的攻击也能以实际的时间成本进行实验验证. 本节对减轮的 Speck32/64, Speck96 以及 Speck128 算法给出改进的差分 – 神经攻击, 与已有的差分 – 神经攻击相比, 我们的攻击在计算复杂度方面均有明显降低.

5.1　改进 Speck32/64 的差分 – 神经攻击

Bao 等 [2] 基于 Gohr 的 8 轮和 7 轮神经区分器, 首次给出对 Speck32/64 的 13 轮差分 – 神经攻击. 此攻击在神经区分器前使用 3 轮前置差分路线, 并为前置差分找到了 12 个中性度较高的广义中性比特, 可以为准确率较低的 8 轮 $\mathcal{ND}$ 提供足够强的区分信号, 进而基于差分 – 神经攻击框架开展 $1+3+8+1=13$ 轮密钥恢复攻击. 此攻击使用 8 轮 $\mathcal{ND}$ 以及阈值 c_1 恢复倒数第 1 轮子密钥 k_{-1}, 进一步使用 7 轮 $\mathcal{ND}$ 以及阈值 c_2 恢复倒数第 2 轮子密钥 k_{-2}. 此外, 攻击分别使用贝叶斯密钥搜索算法以及信心上界 UCB 算法 [1] 以加速密钥搜索以及正确密文结构的识别过程.

我们注意到在此攻击中, 神经区分器的计算占据攻击总运行时间的很大一部分. 而在 4.1 小节中使用查找表成功地重建了对 Speck32/64 的 7 轮和 8 轮区分器, 因此我们考虑直接在文献 [2] 的攻击中使用表 5 中的 7 轮和 8 轮 $\mathcal{LT}$ 替换相应的 $\mathcal{ND}$, 得到 Speck32/64 改进的 13 轮差分 – 神经攻击. 7 轮和 8 轮查找表分别记为 $\mathcal{LT}_{7r}$ 和 $\mathcal{LT}_{8r}$.

我们的攻击基本与文献 [2] 中相同, 具体过程参见附录 A. 相比文献 [2], 我们的攻击做了一些调整.

(1) 由于构建的查找表与原神经区分器在区分性质上不完全相同, 我们将最后一轮密钥 k_{-1} 以及倒数第 2 轮密钥 k_{-2} 的密钥得分过滤阈值由 $c_1=18, c_2=-500$ 调整为 $c_1=17, c_2=-300$.

(2) 文献 [2] 中使用信心上界 UCB 算法推荐每次进行密钥搜索的密文结构. 我们不再使用 UCB 算法, 而是简单地对每个密文结构进行至多 4 次搜索尝试.

(3) 从表 3 中 7 轮 Speck32/64 的信息比特结果可以推导出 k_{-2} 的最高 2 比特不影响 $\mathcal{LT}_{7r}$ 的输入, 因此在我们的攻击中不再恢复 $k_{-2}[14\sim15]$, 只恢复 $k_{-2}[0\sim13]||k_{-1}[0\sim15]$.

(4) 原攻击中, 对于一个密钥猜测 kg, 在一个密文结构下使用式 (2) 得到 kg 的密钥得分 v_{kg}. 由于我们将区分器换为了查找表 $\mathcal{LT}$, 我们使用式 (4) 计算 v_{kg}.

实验结果. 我们将 13 轮 Speck32/64 的攻击使用 C++ 代码进行了实现, 并运行了完整攻击过程 100 次. 我们采用与文献 [2] 中攻击相同的标准, 当返回的密钥猜测 $k_{-2}[0\sim13]||k_{-1}[0\sim15]$ 与正确密钥相差不超过 2 比特时认为攻击成功. 在 100 次攻击实验中, 有 68 次攻击成功, 其中 67 次密钥完全恢复正确, 另外一次攻击中密钥猜测有一比特错误, 余下的 32 次攻击均因密文结构耗尽没有密钥幸存而失败. 在一个单核 CPU 上, 攻击的平均运行时间为 10722.59 s (约 3 h), 攻击的最长运行时间为 20335 s (约 5.65 h). 如果使用与已有差分 – 神经攻击相同的复杂度评估方式, 认为 Speck32/64 算法的加密效率为 2^{28} 次/s[13], 这样攻击的计算复杂度上界大约为 $20335\times2^{28}=2^{42.31}$. 攻击的数据复杂度为 2^{29}.

使用与文献 [3] 中相同的方式, 通过最后再额外遍历一轮子密钥, 可以将 13 轮攻击扩展至 14 轮, 攻击的数据和存储复杂度不变, 计算复杂度变为 $2^{42.31}\times2^{16}=2^{58.31}$.

13) 我们也在与攻击实验相同的环境中测试了 13 轮 Speck32/64 的软件实现效率, 结果只有 $2^{25.87}$ 次/s. 如果按照我们的实测结果, 13 轮攻击的计算复杂度应该为 $20335\times2^{25.87}=2^{40.18}$ 且 14 轮攻击的计算复杂度为 $2^{40.18+16}=2^{56.18}$.

表 9　13 轮 Speck32/64 的差分 – 神经攻击比较.

Table 9　Comparisons of differential-neural attacks on 13-round Speck32/64.

Running time	Time	Data	Success rate	Ref.
$32 \times 14.5 = 464$ GPU hours	$2^{51.07}$	2^{29}	0.82	[2]
$16 \times 4.05 = 64.8$ GPU hours	$2^{45.03}$	2^{27}	0.34	[3]
$8 \times 32 \times 4.08 = 1044.48$ CPU hours	$2^{43.8}$	2^{31}	0.57	[7]
5.65 CPU hours	$2^{42.31}$	2^{29}	0.68	This work

与已有 13 轮差分 – 神经攻击的比较. 已有对 13 轮 Speck32/64 的差分 – 神经攻击 [2,3,7] 均只运行了攻击的核心过程 (完整攻击计算复杂度的 1/32 或 1/16) 以评估攻击的复杂度和成功率, 而本文首次给出完整攻击的实验结果, 这也验证了已有攻击的有效性. 若运行完整的 13 轮攻击过程, 已有攻击的复杂度比较在表 9 中给出. 表 9 中各攻击运行时间与计算复杂度之间的关系存在不一致, 是因为不同攻击可能使用了不同的实验环境及复杂度评估方式. 得益于 NDAFL 框架, 本文攻击的运行时间有较大程度的改进.

与文献 [9] 中差分 – 线性攻击的比较. 文献 [9] 提出了对 Speck32/64 计算复杂度最低的 13 轮理论攻击, 复杂度为 2^{41}. 该攻击与本文攻击框架的区别仅是 8 轮区分器的不同. 文献 [9] 中的区分器为一条 8 轮差分 – 线性特征, 输入差分与本文相同, 为 $(0x40, 0x0)$, 输出的密文差分掩码为 $(0x7020, 0x6020)$, 差分 – 线性特征的相关性为 $-2^{-5.09}$. 使用文献 [2] 找到的 12 个中性比特中的 11 个, 由于 $11 > 2 \times 5.09$, 文献 [9] 表明可以在含有 2^{11} 个密文对的密文结构上将该差分 – 线性特征与随机分布区分, 进而恢复正确的轮密钥 k_{-1}. 我们首先通过实验估计得到该差分 – 线性特征的相关性为 $-2^{-5.5}$, 之后尝试复现此攻击. 我们发现即使使用全部 12 个中性比特, 线性近似统计量的标准差也较大[14], 不足以使得正确密钥猜测的计数排名有足够的统计优势, 相关实验结果见附录 B. 因此文献 [9] 给出攻击的有效性还有待检验, 而本文给出的 13 轮攻击已通过实际攻击结果进行验证. 另一方面, 比较两个 13 轮攻击中使用的 8 轮区分器, 两区分器具有相同的输入差分, 而文献 [9] 中 8 轮差分 – 线性特征的输出掩码 $(0x7020, 0x6020)$ 可以表示为线性近似

$$\Delta C_L[12] \oplus \Delta C_Y[3] \oplus \Delta C_Y[11] \oplus \Delta C_Y[12] = 0. \tag{7}$$

注意到式 (7) 中涉及的 4 个密文对比特均在建立 $\mathcal{LT}_{8r}$ 的信息比特中 (见表 3), 而 $\mathcal{LT}$ 本质即是估计密文对在信息比特上取值的概率分布偏差, 这意味着我们攻击的 $\mathcal{LT}_{8r}$ 已经包含了此差分 – 线性特征. 如果只使用式 (7) 的偏差构建 8 轮区分器, 则区分器的真阳性率为 $0.5 + 2^{-5.5}/2 = 0.5 + 2^{-6.5}$, 真阴性率为 0.5, 区分器准确率只能达到 $0.5 + 2^{-6.5}/2 = 0.5055$. 而我们攻击中使用的 $\mathcal{LT}_{8r}$ 可以达到 0.5137 的准确率, 这也说明我们的区分器利用到了该差分 – 线性近似以外的更多特征. 此外, 我们发现, $\mathcal{LT}_{8r}$ 可能利用了 (包含式 (7) 特征在内的) 多条输入差分均为 $(0x40, 0x0)$ 的 8 轮差分 – 线性特征. 相关验证实验见附录 C.

5.2　改进 Speck96 的差分 – 神经攻击

我们对 Speck96 算法使用 $(0x80, 0x0)$ 作为明文差分约束, 训练出了有效的 8 轮和 7 轮神经区分器, 通过本文的 NDAFL 框架, 成功将这些神经区分器转换为查找表. 在 8 轮区分器前接一条 4 轮前置差分

$$(0x800a080808, 0x800124a0848) \xrightarrow{2^{-22}} (0x80, 0x0), \tag{8}$$

14) 在大小为 $N = 2^{12}$ 的数据集上, 线性近似相关性统计量的标准差为 $2 \times \sqrt{\frac{0.5 \times (1-0.5)}{N}} = 2^{-6}$, 量级与该差分 – 线性特征相关性本身 $2^{-5.5}$ 相当. 此外, 由中性比特生成的明文结构中各数据之间不完全独立, 这会使得统计量的标准差更大.

表 10　14 轮 Speck96 的密钥恢复攻击设置.

Table 10　Key recovery attack settings on 14-round Speck96.

Stage	Distinguisher	Accuracy	$\mathcal{IB}$			Guessed key bits	
			ΔC_L	ΔC_Y	C_Y	k_{-1}	k_{-2}
1	$\mathcal{LT}_1$	0.5646	$8\sim10$	$6\sim10, 17\sim18$	$6\sim9$	$0\sim4, 11\sim12$	–
2	$\mathcal{LT}_2$	0.5948	$8\sim10, 15\sim18$	$8\sim10, 15\sim18, 24\sim26$	$7\sim9, 15\sim17$	$5\sim10, 13\sim20$	–
3	$\mathcal{LT}_3$	0.7437	$16\sim18$	$14\sim18, 22\sim26$	$15\sim17$	$21\sim23$	$8\sim20$
4	$\mathcal{LT}_4$	0.7890	$8\sim10, 15\sim18$	$8\sim10, 14\sim18, 23\sim26$	$7\sim9, 14\sim17$	$40\sim47$	$0\sim7$

我们可以构造对 Speck96 的 $1+4+8+1=14$ 轮实际密钥恢复攻击.

为降低攻击的计算复杂度, 我们考虑在神经区分器的部分信息比特上建立查找表, 采用猜测 – 过滤策略构建多阶段的密钥恢复攻击. 使用同一个明文差分约束 $(0x80, 0x0)$, 我们成功地在 8 轮和 7 轮神经区分器的部分信息比特上分别建立了两个 8 轮查找表 $\mathcal{LT}_1$ 和 $\mathcal{LT}_2$, 以及两个 7 轮查找表 $\mathcal{LT}_3$ 和 $\mathcal{LT}_4$, 这 4 个区分器依次用作每个攻击阶段密钥猜测的过滤, 形成一个 4 阶段的 14 轮密钥恢复攻击. 这些查找表的信息以及对应攻击各阶段要猜测的密钥比特在表 10 中给出. 攻击各阶段, 猜测相应轮密钥比特后可以计算出对应查找表区分器信息比特的取值, 进而得到查找表的输出. 对于一个含有 m 个密文对的密文结构, 设在某个密钥猜测 kg 下查找表的输入分别为 $V_1^{\mathrm{kg}}, \ldots, V_m^{\mathrm{kg}}$, 使用式 (4) 即可计算密钥猜测得分 v_{kg}. 若某个密钥猜测的得分大于特定阈值 c, 则保留该密钥猜测进入攻击下一阶段. 最终恢复 53 比特轮密钥 $k_{-1}[0\sim23, 40\sim47]||k_{-2}[0\sim20]$.

攻击过程. 在攻击的 4 轮前置差分中, 使用 10 个中性比特 $\mathcal{NB}=\{32, 33, 34, 35, 36, 87, 88, 89, 90, 91\}$, 这些中性比特的中性度均接近 1. 这样每个明文结构含有 $m=2^{10}$ 个明文对. 考虑到前置差分的概率为 2^{-22}, 我们限制一次攻击最多使用 $n_{\mathrm{cts}}=2^{23}$ 个明文结构. 攻击中取密钥得分阈值 $c=0$, 各阶段保留得分最高的密钥猜测数量 $T=10$. 攻击恢复倒数两轮子密钥共 53 比特 $k_{-1}[0\sim23, 40\sim47]||k_{-2}[0\sim20]$, 只有当返回的密钥完全正确时认为攻击成功. 攻击的具体过程如下所示.

(1) 随机生成 n_{cts} 个明文结构. 每个明文结构由一个满足式 (8) 中输入差分的状态对使用 10 个中性比特 $\mathcal{NB}$ 扩展而来. 对明文结构中每个明文对, 使用轮密钥 0 解密一轮后询问 14 轮 Speck96 加密得到对应密文对. 这样收集到 n_{cts} 个密文结构 $\{\mathcal{C}_1, \ldots, \mathcal{C}_{n_{\mathrm{cts}}}\}$.

(2) 对每个密文结构 $\mathcal{C}\in\{\mathcal{C}_1, \ldots, \mathcal{C}_{n_{\mathrm{cts}}}\}$.

(a) 依次执行 4 个攻击阶段. 在阶段 i ($i\in\{1, 2, 3, 4\}$).

(i) 对上阶段幸存的每个密钥猜测, 按照表 10 中设置猜测相关密钥比特并执行部分解密. 对每个密钥猜测 kg, 计算 $\mathcal{C}$ 中各密文对对应的查找表区分器输入值 V_j^{kg} ($j\in\{1, \ldots, m\}$), 由该阶段查找表区分器 $\mathcal{LT}_i$, 通过式 (4) 计算密钥得分 v_{kg}, 若 $v_{\mathrm{kg}}>c$, 将 $(\mathrm{kg}, v_{\mathrm{kg}})$ 加入幸存密钥列表 L_i.

(ii) 若 L_i 为空, 即没有密钥猜测幸存, 则返回第 2 步尝试下一个密文结构. 否则, 对列表 L_i 中元素按照密钥得分降序排序, 保留得分最高的 T 个密钥猜测进入下一阶段.

(b) 若阶段 4 执行完毕且 L_4 非空, 返回得分最高的密钥猜测作为正确密钥.

(3) 若密文结构耗尽仍没有正确密钥返回, 则返回攻击失败.

攻击结果. 攻击的数据复杂度为 $2^{23}\times2^{10}\times2=2^{34}$ 个选择明文. 存储空间主要为一个含 2^{26} 个 64 位整数表项的查找表, 即 2^{26+3} Bytes = 512 MB. 攻击计算复杂度主要来自阶段 1, 一旦阶段 1 输出了一个幸存密钥, 也意味着识别到了正确的密文结构, 后续 3 个攻击阶段只需要在这一个结构上执行即可, 因此后续复杂度可忽略. 阶段 1 的计算复杂度为使用 2^7 个密钥猜测对 2^{34} 个密文解密一轮. 通过 NDAFL 框架, 区分器的处理时间仅为一次查找操作, 相比于解密操作的时间可忽略, 故攻击的计算复杂度可理论给出, 为 $2^{34}\times2^7/14=2^{37.19}$. 为评估攻击成功率, 我们在单核 CPU 上执行了 100 次实际攻击实验, 其中 82 次成功, 攻击成功率为 0.82. 攻击的平均运行时间为 3.68 个 CPU 小时.

表 11　17 轮 Speck128 的密钥恢复攻击设置.

Table 11　Key recovery attack settings on 17-round Speck128.

Stage i	Distinguisher	Accuracy	$\mathcal{IB}$			Δ_{Si}	p_i	PNB_i	Guessed bits of k_{-1}
			ΔC_L	ΔC_Y	C_Y				
1	$\mathcal{LT}_1$	0.6155	$8 \sim 11$	$7 \sim 11, 16 \sim 19$	$6 \sim 10$	$\Delta_{[64]}$	2^{-45}	0.567	$0 \sim 13$
2	$\mathcal{LT}_2$	0.6208	$19 \sim 23$	$18 \sim 23, 28 \sim 31$	$18 \sim 22$	$\Delta_{[76]}$	2^{-46}	0.649	$14 \sim 25$
3	$\mathcal{LT}_3$	0.6190	$33 \sim 37$	$33 \sim 37, 42 \sim 45$	$32 \sim 36$	$\Delta_{[90]}$	2^{-45}	0.502	$26 \sim 39$
4	$\mathcal{LT}_4$	0.6208	$48 \sim 52$	$47 \sim 52, 57 \sim 60$	$47 \sim 51$	$\Delta_{[105]}$	2^{-45}	0.516	$40 \sim 54$
5	$\mathcal{LT}_5$	0.6192	$3 \sim 4, 58 \sim 60$	$0 \sim 4, 11 \sim 12, 56 \sim 60$	$1 \sim 3, 56 \sim 59$	$\Delta_{[113]}$	2^{-45}	0.512	$55 \sim 62$

恢复剩余密钥比特. 对 Speck96/96, 一旦上述 53 比特密钥恢复成功, 我们可以对 k_{-1}, k_{-2} 剩余的 43 比特进行暴力搜索, 由倒数两轮子密钥可推出各个轮密钥, 并使用一个明密文对进行验证. 后续计算复杂度为 2^{43}, 成功率为 1. 故完整 14 轮攻击的计算复杂度为 $2^{37.19} + 2^{43} = 2^{43.03}$, 成功率为 0.82. 对于 Speck96/144, 除猜测剩余 43 比特以外, 我们可以再多向后猜测一轮子密钥, 形成 15 轮密钥恢复攻击, 攻击计算复杂度为 $2^{43.03} \times 2^{48} = 2^{91.03}$.

5.3　改进 Speck128 的差分 – 神经攻击

Chen 等 [4] 给出了对 Speck128 的 17 轮理论攻击以及 12 轮实际攻击. 我们使用与其相同的攻击框架, 在不同的密文比特段建立 5 个 9 轮查找表区分器, 每个查找表区分器负责恢复最后一轮子密钥 k_{-1} 的若干比特, 改进对 17 轮 Speck128 的 5 阶段密钥恢复攻击的计算复杂度, 并给出 14 轮实际攻击结果.

本文攻击使用的区分器输入差分组合与文献 [4] 相比只有最后一阶段有所不同, 同时我们将所有区分器通过 NDAFL 变为查找表. 攻击中 5 个阶段使用的 9 轮查找表区分器分别记为 $\mathcal{LT}_1 \sim \mathcal{LT}_5$, 构建这 5 个区分器使用的明文差分约束分别为一比特的差分

$$\Delta_{[64]}, \Delta_{[76]}, \Delta_{[90]}, \Delta_{[105]}, \Delta_{[113]}. \tag{9}$$

在每个区分器前接一条 6 轮差分路线, 形成 $1+6+9+1=17$ 轮密钥恢复攻击. 由于各区分器的明文差分约束均为左侧字上的一比特差分, 这些前置差分路线具有旋转对称性, 由文献 [4] 中表 7 的基本差分路线

$$(0x4041041440401000, 0x024040240640d010) \xrightarrow{6r} (0x1, 0x0), \tag{10}$$

经循环左移得来, 差分概率为 2^{-45} 或 2^{-46}. 对于这些差分路线, 均可以找到多于 12 个中性度大于 0.7 的中性比特, 我们使用其中中性度最高的 10 个, 各攻击阶段使用中性比特的总中性度分别记为 $\mathrm{PNB}_1 \sim \mathrm{PNB}_5$. 每个攻击阶段的设置总结在表 11 中. 表 11 中 Δ_{Si} 表示每个区分器使用的差分约束, p_i 表示每个阶段使用前置差分路线的概率, PNB_i 表示 10 个中性比特的总中性度.

攻击过程. 攻击流程基本按照文献 [4] 中算法 1 串行执行的深度学习辅助多阶段密钥恢复攻击展开. 在攻击阶段 i, 前置差分概率为 p_i, 我们限制一次最多使用 $4/(p_i \times \mathrm{PNB}_i)$ 个明文结构, 这样期望有 4 个明文结构完全通过前置差分. 每个明文结构由一个明文对使用 10 个中性比特扩展得到, 含有 $m = 2^{10}$ 个明文对. 具体攻击过程为顺序执行 5 个攻击阶段, 在阶段 i ($i \in \{1,2,3,4,5\}$) 中.

(1) 随机生成 $4/(p_i \times \mathrm{PNB}_i)$ 个明文结构. 每个明文结构由一个满足前置差分输入差分约束的明文对使用 10 个中性比特扩展而来. 对明文结构中每个明文对, 使用轮密钥 0 解密一轮后, 询问 17 轮 Speck128 加密得到对应密文对. 这样收集到 $4/(p_i \times \mathrm{PNB}_i)$ 个密文结构 $\{\mathcal{C}_1, \ldots, \mathcal{C}_{4/(p_i \times \mathrm{PNB}_i)}\}$.

(2) 对每个密文结构 $\mathcal{C} \in \{\mathcal{C}_1, \ldots, \mathcal{C}_{4/(p_i \times \mathrm{PNB}_i)}\}$.

(a) 基于上阶段幸存的每个密钥猜测, 按照表 11 中设置猜测本阶段 k_{-1} 相关比特并对 $\mathcal{C}$ 中每个密文对执行一轮部分解密. 对每个密钥猜测 kg, 计算 $\mathcal{C}$ 中各密文对解密一轮后对应查找表区分器输入值 V_j^{kg} ($j \in \{1,\ldots,m\}$), 访问该阶段查找表 $\mathcal{LT}_i$, 由式 (4) 得到密钥得分 v_{kg}, 若 $v_{\mathrm{kg}} > c$, 将 $(\mathrm{kg}, v_{\mathrm{kg}})$ 加入幸存密钥列表 L_i.

(b) 若遍历所有密钥猜测取值后 L_i 为空, 则跳转至第 2 步处理下一个密文结构.

(c) 若遍历所有密钥猜测取值后 L_i 非空, 且当前不是最后一个攻击阶段 ($i \neq 5$), 则当前阶段攻击结束. 对 L_i 中元素按照密钥得分降序排列, 保留得分最高的 T 个密钥猜测, 跳转至第 1 步执行攻击阶段 $i+1$.

(d) 若遍历所有密钥猜测取值后 L_i 非空且 $i = 5$, 则本次攻击结束, 返回 L_i 中得分最高的密钥猜测作为正确密钥.

(3) 若攻击阶段 i 中处理完所有 $4/(p_i \times \mathrm{PNB}_i)$ 个密文结构后均无密钥猜测幸存, 则本次攻击结束, 返回攻击失败.

攻击中取密钥得分阈值 $c = 0$, 各阶段保留得分最高的密钥猜测数量 $T = 3$. 攻击恢复最后一轮子密钥的低 63 比特 $k_{-1}[0 \sim 62]$ (除最高位外所有比特). 当返回的密钥猜测错误不超过 2 比特时, 认为攻击成功. 正如文献 [4] 中指出的, 当上述攻击成功后, 各前置差分对应的正确明文结构也已找到. 为恢复剩余密钥比特, 后续可以遍历 $k_{-1}[0 \sim 62]$ 的两比特错误 $C_{63}^2 \times 2^2 = 2^{12.93}$ 个可能取值, 使用准确率更高的 8 轮区分器并复用已识别到的正确明文结构, 恢复 $k_{-1}[63]||k_{-2}[0 \sim 63]$. 后续攻击的复杂度可忽略不计.

攻击结果. 上述攻击进行一次最多消耗 $\sum_{i=1}^{5} 4/(p_i \times \mathrm{PNB}_i)$ 个明文结构, 数据复杂度为 $\sum_{i=1}^{5} 4/(p_i \times \mathrm{PNB}_i) \times 2^{11} = 2^{61.62}$. 存储空间主要来自一个含有 2^{24} 个 64 位整数表项的查找表, 即 2^{24+3} Bytes = 128 MB. 记各阶段猜测的密钥比特数量为 a_i, 上阶段最多幸存密钥数量为 β_i, 有 $a_1 = 14$, $a_2 = 12$, $a_3 = 14$, $a_4 = 15$, $a_5 = 8$ 和 $\beta_1 = 1$, $\beta_2 = \beta_3 = \beta_4 = \beta_5 = 3$. 攻击的计算复杂度为各阶段使用密钥猜测对所有密文对进行一轮解密操作. 同样地, 与解密操作相比, 区分器的访问成本可以忽略. 因此攻击总的计算复杂度为 $\sum_{i=1}^{5} 4/(p_i \times \mathrm{PNB}_i) \times 2^{11} \times 2^{a_i} \times \beta_i/17 = 2^{72.36}$.

通过 14 轮实际攻击评估成功率. 上述 17 轮攻击复杂度较高, 无法直接验证. 我们在区分器前使用轮数更短的差分路线可以得到复杂度较低的实际攻击. 在表 11 给出的 5 个区分器前分别接概率为 2^{-12} 的 3 轮差分路线, 可以得到 Speck128 的 14 轮实际攻击. 14 轮攻击与 17 轮攻击的密钥猜测过程完全相同, 攻击参数中只有前置差分概率以及中性比特中性度不同. 如果假设错误的明文结构不会对密钥恢复产生影响, 则两攻击的理论成功率完全相同. 14 轮攻击中, 前置差分概率均为 $p_i = 2^{-12}$, 10 个中性比特中性度均为 1, 攻击的计算复杂度为 $\sum_{i=1}^{5} 4/p_i \times 2^{11} \times 2^{a_i} \times \beta_i/14 = 2^{38.63}$, 数据复杂度为 $\sum_{i=1}^{5} 4/p_i \times 2^{11} = 2^{27.32}$. 我们运行实际攻击 100 次, 其中有 78 次成功, 平均运行时间为 2.7 个 CPU 小时. 因此本节给出的对 Speck128 的 17 轮理论攻击的成功率约为 0.78.

6　总结

本文提出了一种通用的深度学习辅助密码分析增强框架 NDAFL, 能够以一种黑盒的方式将神经区分器转换为查找表, 同时基本保留其区分能力. 使用此框架, 可以在差分 – 神经攻击中省去神经网络的计算成本, 大大降低攻击计算复杂度. 我们在 Speck 和 Simon 不同分组长度的成员以及 LEA 算法上验证了本框架的有效性. 通过 NDAFL 框架我们想要说明: 在密码分析任务中, 深度学习技术更多地应当作为一种工具, 帮助分析人员快捷地定位算法可利用的非随机特征、设计强力攻击, 而非作为攻击中的核心区分器使用. 一旦我们通过算法 3 成功地构建查找表区分器, 在后续的攻击过程中我们便可以完全摆脱对神经网络的依赖. 作为直接应用, 我们使用 NDAFL 框架改进了对 Speck32/64, Speck96, Speck128 算法已有的差分 – 神经攻击结果, 进一步说明了本框架的效果.

参考文献

1 Gohr A. Improving attacks on round-reduced speck32/64 using deep learning. In: Proceedings of Annual International Cryptology Conference, 2019. 150–179
2 Bao Z Z, Guo J, Liu M C, et al. Enhancing differential-neural cryptanalysis. In: Proceedings of International Conference on the Theory and Application of Cryptology and Information Security, 2022. 318–347
3 Zhang L, Wang Z L, Wang B C. Improving differential-neural cryptanalysis. IACR CiC, 2024, 1: 13
4 Chen Y, Bao Z Z, Shen Y T, et al. A deep learning-aided key recovery framework for large-state block ciphers. Sci Sin Inform, 2023, 53: 1348–1367 [陈怡, 包珍珍, 申焱天, 等. 用于大状态分组密码的深度学习辅助密钥恢复框架. 中国科学: 信息科学, 2023, 53: 1348–1367]
5 Bao Z Z, Lu J Y, Yao Y R, et al. More insight on deep learning-aided cryptanalysis. In: Proceedings of International Conference on the Theory and Application of Cryptology and Information Security, 2023. 436–467
6 Benamira A, Gerault D, Peyrin T, et al. A deeper look at machine learning-based cryptanalysis. In: Proceedings of Annual International Conference on the Theory and Applications of Cryptographic Techniques, 2021. 805–835
7 Lv G Q, Jin C H, Shi Z, et al. Approximating neural distinguishers using differential-linear imbalance. J Supercomput, 2024, 80: 26865–26889
8 Biryukov A, dos Santos L C, Teh J S, et al. Meet-in-the-filter and dynamic counting with applications to speck. In: Proceedings of International Conference on Applied Cryptography and Network Security, 2023. 149–177
9 Lv G Q, Jin C H, Cui T. A MIQCP-based automatic search algorithm for differential-linear trails of ARX ciphers. IACR Cryptol ePrint Archive, 2023, 2023: 259
10 Feng Z H, Luo Y, Wang C, et al. Improved differential cryptanalysis on speck using plaintext structures. In: Proceedings of Australasian Conference on Information Security and Privacy, 2023. 3–24
11 Bellini E, Gerault D, Grados J, et al. Fully automated differential-linear attacks against ARX ciphers. In: Proceedings of Cryptographers' Track at the RSA Conference, 2023. 252–276
12 Huang T R, Li Y Y, Fu Q G, et al. Improving differential-neural cryptanalysis for large-state SPECK. In: Proceedings of International Conference on Information and Communications Security, 2025. 40–57
13 Dinur I. Improved differential cryptanalysis of round-reduced speck. In: Proceedings of International Conference on Selected Areas in Cryptography, 2014. 147–164
14 Beaulieu R, Shors D, Smith J, et al. The SIMON and SPECK families of lightweight block ciphers. IACR Cryptol ePrint Archive, 2013, 2013: 404
15 Hong D, Lee J K, Kim D C, et al. LEA: a 128-bit block cipher for fast encryption on common processors. In: Proceedings of International Workshop on Information Security Applications, 2014. 3–27
16 Biham E, Chen R. Near-collisions of SHA-0. In: Proceedings of Annual International Cryptology Conference, 2004. 290–305
17 Chen Y, Shen Y, Yu H B. Neural-aided statistical attack for cryptanalysis. Comput J, 2023, 66: 2480–2498
18 Gohr A, Leander G, Neumann P. An assessment of differential-neural distinguishers. IACR Cryptol ePrint Archive, 2022, 2022: 1521

附录 A　Speck32/64 的 13 轮攻击设置

A.1　前置差分

本文的 13 轮攻击使用与文献 [2] 相同的前置差分设置. 在 8 轮区分器前接两条 3 轮前置差分 $(0x8020, 0x4101) \rightarrow (0x40, 0x0)$ 以及 $(0x8060, 0x4010) \rightarrow (0x40, 0x0)$. 这两条差分均只对 2^{63} 的弱密钥成立, 差分概率均为 2^{-11}.

为获得足够强的区分信号, 前置差分中使用共 11 个文献 [2] 给出的中性比特:

$$\mathcal{NB} = \{[22], [13], [20], [5, 28], [15, 24], [12, 19], [6, 29], [4, 27, 29], [14, 21], [0, 8, 31], [30]\}, \tag{A1}$$

以及 1 个邻接差分切换比特 [21]. $\mathcal{NB}$ 中包含 3 个条件中性比特, 相应地攻击中构造的明文数据需要满足 3 比特条件, 在攻击中需要猜测首轮子密钥 k_0 相关的 3 比特取值:

$$\begin{aligned} P_L[12] \oplus P_R[5] &= k_0[12] \oplus k_0[5] \oplus 1, \\ P_R[1] &= k_0[1], \\ P_L[11] \oplus P_R[4] &= k_0[11] \oplus k_0[4] \oplus 1. \end{aligned} \tag{A2}$$

前置差分中使用了 1 个邻接差分切换比特. 对于满足差分 $(0x8020, 0x4101) \rightarrow (0x40, 0x0)$ 的明文对, 同时翻转两明文的第 21 比特, 得到的明文对会满足另一条差分 $(0x8060, 0x4010) \rightarrow (0x40, 0x0)$, 反之亦然.

此外, 文献 [2] 中使用了数据复用技术以节省数据复杂度. 记 $\Delta_1 = (0x8020, 0x4101), \Delta_2 = (0x8060, 0x4010)$, 有 $\Delta_1 \oplus \Delta_2 = \Delta_{[22]}$, 而 [22] 又恰好为两差分路线共享的一个中性比特. 在攻击中, 构造满足 Δ_1 的明文对 $(P, P \oplus \Delta_1)$, 使用中性比特 [22] 扩展出另一个明文对 $(P \oplus \Delta_{[22]}, P \oplus \Delta_1 \oplus \Delta_{[22]})$, 这两个明文对构成一个满足 Δ_1 的明文结构. 对这 4 个明文重新组合便可额外得到一个满足 Δ_2 的明文结构 $\{(P, P \oplus \Delta_1 \oplus \Delta_{[22]} = P \oplus \Delta_2), (P \oplus \Delta_{[22]}, P \oplus \Delta_1 = P \oplus \Delta_{[22]} \oplus \Delta_2)\}$. 这样进行数据复用后生成一个明文对平均只需要一个选择明文.

A.2　攻击过程

对 Speck32/64 的 13 轮攻击的参数包括:

(1) $n_{\text{kg}} = 2^3$. 为攻击中需要猜测的式 (A2) 中 k_0 相关比特可能取值数;

(2) $n_{\text{cts}} = 2^{14}$. 攻击中限制使用的明文结构最大数量;

(3) $m = 2^{12}$. 每个密文结构中的密文对个数;

(4) $n_r = 4$. 对每个密文结构尝试搜索的次数;

(5) $c_1 = 17, c_2 = -300$. 分别为 k_{-1} 以及 k_{-2} 密钥猜测的得分过滤阈值;

(6) $n_{\text{iter1}} = 5, n_{\text{cand1}} = 32$ 以及 $n_{\text{iter2}} = 5, n_{\text{cand2}} = 32$. 分别为猜测 k_{-1} 和 k_{-2} 时运行贝叶斯密钥搜索的迭代次数和每次迭代的候选密钥数量;

(7) μ_1, σ_1 以及 μ_2, σ_2. 分别为 $\mathcal{LT}_{8r}$ 以及 $\mathcal{LT}_{7r}$ 的错误密钥响应文件 (wrong key response profile), 分别在 k_{-1} 和 k_{-2} 猜测的低 14 位上建立.

本文 13 轮攻击的完整过程如下.

(1) 猜测式 (A2) 中 k_0 相关的 3 比特共 n_{kg} 个取值. 在每种 k_0 取值下:

(a) 生成 $n_{\text{cts}}/2$ 个随机明文对 $(P_L||P_R, (P_L||P_R) \oplus \Delta_1)$, 且 P_L, P_R 满足式 (A2) 中的 3 比特条件;

(b) 将这 $n_{\text{cts}}/2$ 个明文对, 分别使用 $\mathcal{NB}$ 中 11 个中性比特进行扩展, 形成 $n_{\text{cts}}/2$ 个结构. 再使用邻接差分切换比特 [21] 将每个结构中明文对数量翻倍, 新的明文对是通过翻转原始对的 21 比特并将差分调为 Δ_2 获得. 这样每个结构中含有 m 个明文对;

(c) 以 0 为轮密钥对 $n_{\text{cts}}/2$ 个结构中所有数据进行一轮解密, 并使用解密后数据询问 13 轮 Speck32/64 加密, 收集密文. 这样得到 $n_{\text{cts}}/2$ 个密文结构, 每个密文结构含有 m 个密文对;

(d) 对每个密文结构中每组由中性比特 [22] 扩展出的新旧 2 个密文对, 按照之前介绍的数据复用方式交换 4 个密文的组合方式. 这样得到额外 $n_{\text{cts}}/2$ 个密文结构. 这样一共有 n_{cts} 个可用的密文结构, 记为 $\{\mathcal{C}_1, \ldots, \mathcal{C}_{n_{\text{cts}}}\}$;

(e) 遍历这 n_{cts} 个可用的密文结构至多 n_r 次. 对于每次遍历到的密文结构 $\mathcal{C} \in \{\mathcal{C}_1, \ldots, \mathcal{C}_{n_{\text{cts}}}\}$:

(i) 进行第 1 阶段攻击. 对密文结构 $\mathcal{C}$, 使用 8 轮查找表 $\mathcal{LT}_{8r}$ 以及参数 $n_{\text{iter1}}, n_{\text{cand1}}, \mu_1, \sigma_1$ 调用贝叶斯搜索算法 (算法 A1), 密钥猜测空间为 $k_{-1}[0 \sim 13]$. 算法返回 $n_{\text{iter1}} \times n_{\text{cand1}}$ 个候选密钥及得分列表 $L_1 = [(g_{1i}, v_{1i}), i \in \{1, \ldots, n_{\text{iter1}} \times n_{\text{cand1}}\}]$;

(ii) 对 L_1 中每个候选密钥 g_{1i}, 如果对应得分 $v_{1i} > c_1$:

– 使用 g_{1i} 作为 $k_{-1}[0 \sim 13]$, 并随机对 $k_{-1}[14 \sim 15]$ 赋值. 使用得到的 k_{-1} 对 $\mathcal{C}$ 中密文解密一轮, 得到 12 轮 Speck32/64 的密文结构 $\mathcal{C}'$;

– 进行第 2 阶段攻击. 对密文结构 $\mathcal{C}'$, 使用 7 轮查找表 $\mathcal{LT}_{7r}$ 以及参数 $n_{\text{iter2}}, n_{\text{cand2}}, \mu_2, \sigma_2$ 调用贝叶斯搜索算法 (算法 A1), 密钥猜测空间为 $k_{-2}[0 \sim 13]$. 算法返回 $n_{\text{iter2}} \times n_{\text{cand2}}$ 个候选密钥及得分列表 $L_2 = [(g_{2j}, v_{2j}), j \in \{1, \ldots, n_{\text{iter2}} \times n_{\text{cand2}}\}]$;

– 对 L_2 中每个候选密钥猜测 g_{2j}, 如果 $v_{2j} > c_2$: 将 g_{2j} 作为 $k_{-2}[0 \sim 13]$. 对目前的密钥猜测 $k_{-1}[0 \sim 15]||k_{-2}[0 \sim 13]$, 在 2 比特的汉明距离内使用区分器 $\mathcal{LT}_{7r}$ 进行验证搜索. 如果可以找到得分更高的密钥猜测, 则更新密钥猜测, 直到找不到更好的密钥猜测为止. 将最终的 $k_{-1}[0 \sim 15]||k_{-2}[0 \sim 13]$ 返回, 攻击结束.

(2) 若对于 n_{kg} 个 k_0 取值, 攻击均没有返回最终密钥猜测, 则返回攻击失败.

附录 B　Speck32/64 的 13 轮差分 – 线性攻击验证

B.1　13 轮差分 – 线性攻击设置

文献 [9] 给出了对 13 轮 Speck32/64 的差分 – 线性攻击, 计算复杂度大约为 2^{41}. 此攻击使用一条相关性为 $-2^{-5.09}$ 的 8 轮差分 – 线性特征 $(0x40, 0x0) \xrightarrow{8 \text{ rounds}} (0x7020, 0x6020)$ 作为区分器. 若 8 轮输出状态差分为 $(\Delta X_L, \Delta X_R)$, 该差分 – 线性特征对应的线性近似表达式为

$$\Delta X_L[12] \oplus \Delta X_Y[3] \oplus \Delta X_Y[11] \oplus \Delta X_Y[12] = 0, \quad \Pr = 0.5 - 2^{-6.09}, \tag{B1}$$

其中 $\Delta X_Y = (\Delta X_L \oplus \Delta X_R) \ggg B$. 在 8 轮差分 – 线性特征前使用与第 A.1 小节相同的 2 条 3 轮前置差分, 即可得到 $1+3+8+1=13$ 轮攻击. 为向差分 – 线性特征提供足够的数据进行区分, 攻击在 3 轮前置差分中使用了文献 [5] 找到的 11 个广义中性比特, 将一个明文对扩展为含 2^{11} 个明文对的明文结构, 这些明文对一同以 $2^{-11} \times \text{PNB} \approx 2^{-11.88}$ 的概率满足差分 – 线性特征的输入差分 $(0x40, 0x0)$. 由于 $2^{11} > (2^{5.09})^2$, 攻击者可以在一个明文结构中对差分 – 线性特

算法 A1 贝叶斯密钥搜索算法.

输入: 密文结构 $\mathcal{C}=\{(C_i,C_i'),i=1,\ldots,m\}$; 查找表区分器 $\mathcal{LT}$; 密钥猜测空间 $\mathcal{K}$; 迭代次数 n_{iter}; 每轮迭代候选密钥数 n_{cand}; $\mathcal{LT}$ 的错误密钥响应 μ,σ.

输出: 密钥猜测及得分列表 L.

从 $\mathcal{K}$ 中不重复地随机选择 n_{cand} 个密钥猜测 $S\Leftarrow\{k_1,\ldots,k_{n_{\text{cand}}}\}$;
目前已尝试的密钥猜测及得分列表置空 $L\Leftarrow\{\}$;
for $j\in\{1,2,\ldots,n_{\text{iter}}\}$ **do**
  for 每个候选密钥猜测 $k\in S$ **do**
    对 $\mathcal{C}$ 中每个密文对使用 k 进行一轮解密 $(X_i,X_i')\Leftarrow(D_k(C_i),D_k(C_i'))$, $i\in\{1,\ldots,m\}$;
    由解密后伪密文对 (X_i,X_i') 提取 $\mathcal{LT}$ 信息比特上的输入值 V_i, $i\in\{1,\ldots,m\}$;
    计算密钥猜测得分 $v_k\Leftarrow\sum_{i=1}^{m}\log_2\frac{\mathcal{LT}[V_i]}{\alpha}$;
    $L\Leftarrow L||[(k,v_k)]$;
    计算区分器响应均值 $m_k\Leftarrow\frac{1}{m}\sum_{i=1}^{m}\log_2\frac{\mathcal{LT}[V_i]}{\alpha}$;
  end for
  for 每个密钥猜测 $k'\in\mathcal{K}$ **do**
    计算 $\lambda_{k'}\Leftarrow\sum_{k\in S}(m_k-\mu[k'\oplus k])^2/(\sigma[k'\oplus k])^2$;
  end for
  对 $\mathcal{K}$ 中所有密钥猜测 k', 依据 $\lambda_{k'}$ 升序排序, 取排名最靠前的 n_{cand} 个密钥猜测更新 S;
end for
返回 L.

征对应线性近似的偏差进行区分. 一旦有明文结构通过了前置差分且密钥猜测正确, 密文结构中所有解密一轮的状态计算得到线性近似的相关性统计量应该达到最低 (线性近似偏差绝对值最高). 为计算式 (B1) 涉及的解密一轮后的状态值, 攻击需要猜测最后一轮子密钥低 15 比特 $k_{-1}[0\sim14]$. 此外, 攻击中需猜测 k_0 的 2 比特取值以满足 2 个条件中性比特, 同时额外猜测 k_0 的 3 比特以提前满足前置差分的 3 比特条件 (这样 3 轮前置差分的概率由 2^{-11} 提高为 2^{-8}).

完整的攻击过程如下.

(1) 猜测 5 比特 k_0 的取值. 对每个 k_0 的取值:

(a) 使用 11 个广义中性比特生成 2^9 个明文结构, 收集对应密文结构. 每个密文结构含有 $N=2^{11}$ 个密文对;

(b) 猜测 $k_{-1}[0\sim14]$. 在每个密钥猜测下, 对所有的密文结构解密一轮, 在每个结构上统计使式 (B1) 成立的伪密文对数量 t 并计算相关性统计量 $\hat{c}=(2t-N)/N$, 记录所有密文结构中 $\hat{c}$ 的最小值作为该密钥猜测的得分.

(2) 将所有密钥猜测按得分由低到高排名, 返回得分最低的密钥猜测为正确密钥.

B.2　攻击的实验验证

首先, 我们发现攻击中为计算式 (B1) 的线性近似, 只需要知道最后一轮子密钥的低 14 位即 $k_{-1}[0\sim13]$, 无需猜测 $k_{-1}[14]$. 此外, 我们通过 2^{26} 的加密数据估计得到文献 [9] 使用的 8 轮差分 – 线性特征相关性为 $-2^{-5.50}$, 其绝对值比文献 [9] 报告的结果更小. 对于一个密文结构中 $N=2^{11}$ 个数据, 假设所有数据相互独立, 最终计算的相关性统计量 $\hat{c}$ 的标准差为 $\sigma=2\times\sqrt{0.5\times(1-0.5)/N}=2^{-5.5}$, 这与该特征的相关性绝对值量级相同. 同时正如文献 [4] 指出, 使用中性比特会在一定程度上破坏掉明文结构中各明文对的独立性, 这往往会使得统计量的标准差更大. 因此, 我们认为统计量分布较大的标准差很难使得正确密钥得分排名具有明显的优势.

我们尝试复现文献 [9] 中的 13 轮攻击. 为加速攻击的验证, 我们只运行核心过程. 具体地, 我们只运行攻击中 k_0 取值猜测正确的情况, 每次攻击共使用 2^9 个明文结构. 此外, 我们只尝试从 2^8 个密钥猜测 (而非 2^{14} 个密钥猜测) 中恢复正确密钥 (每次攻击的密钥猜测空间包括 1 个正确密钥以及 255 个随机生成的 $k_{-1}[0\sim13]$), 这使得正确密钥更容易排名靠前. 我们记录每次攻击中正确密钥得分的排名, 以及攻击返回的密钥猜测与正确密钥的平均汉明距离 (差异比特数目), 一次攻击中正确密钥排名为 0 或者返回密钥与正确密钥相差汉明距离为 0 说明攻击成功. 我们使用文献 [9] 中的 11 个广义中性比特以及文献 [2] 中发现的 12 个广义中性比特分别进行了 100 次攻击实验, 结果如表 B1 所示. 表 B1 中 $|\mathcal{NB}|$ 表示攻击中使用的中性比特数目. 对于无效的区分器 (相当于随机猜测密钥), 需要从 2^8 个 $k_{-1}[0\sim13]$ 中找到正确密钥, 期望正确密钥排名为 $(0+255)/2=127.5$, 期望返回密钥与正确密钥汉明距离为 $14/2=7$, 期望成功率为 $1/256=0.0039$. 表 B1 中结果说明文献 [9] 给出的 8 轮差分 – 线性特征的确存在区分效果, 但区分优势较弱. 即使使用文献 [2] 给出的 12 个广义中性比特, 文献 [9] 中的攻击过程也很难直接形成一个强力的密钥恢复攻击 (也许需要设计更精细的攻击过程).

表 B1　13 轮差分 – 线性攻击实验结果.

Table B1　Experimental results of the 13-round differential-linear attack.

$\lvert\mathcal{NB}\rvert$	Success rate	Average rank	Average hamming distance
11	0.00	123.29	6.76
12	0.02	107.58	6.88

表 C1　Speck32/64 的 8 轮差分 – 线性特征利用实验结果.

Table C1　Results of the 8-round differential-linear characteristic using experiment on Speck32/64.

Γ	Linear approximation	Correlation	Remaining accuracy
0x203100	$\Delta C_L[12] \oplus \Delta C_Y[3] \oplus \Delta C_Y[11] \oplus \Delta C_Y[12] = 0$	$-2^{-5.51}$	$0.5 + 1.76 \times 10^{-3}$
0x302100	$\Delta C_L[11] \oplus \Delta C_L[12] \oplus \Delta C_Y[3] \oplus \Delta C_Y[12] = 0$	$-2^{-5.51}$	$0.5 + 1.80 \times 10^{-3}$
0x302130	$\Delta C_L[11] \oplus \Delta C_L[12] \oplus \Delta C_Y[3] \oplus \Delta C_Y[12] \oplus C_Y[10] \oplus C_Y[11] = 0$	$2^{-6.52}$	$0.5 + 4.57 \times 10^{-4}$
0x202930	$\Delta C_L[12] \oplus \Delta C_Y[3] \oplus \Delta C_Y[10] \oplus \Delta C_Y[12] \oplus C_Y[10] \oplus C_Y[11] = 0$	$-2^{-6.51}$	$0.5 + 4.71 \times 10^{-4}$
0x203130	$\Delta C_L[12] \oplus \Delta C_Y[3] \oplus \Delta C_L[11] \oplus \Delta C_L[12] \oplus C_Y[10] \oplus C_Y[11] = 0$	$-2^{-6.51}$	$0.5 + 4.77 \times 10^{-4}$
0x202180	$\Delta C_L[12] \oplus \Delta C_Y[2] \oplus \Delta C_Y[3] \oplus \Delta C_Y[12] = 0$	$2^{-6.57}$	$0.5 + 2.03 \times 10^{-4}$

算法 C1 随机化区分器测试集 M, 只保留单条线性特征 Γ.

输入: 区分器输入形式 I; 区分器测试集 M, $M = \{(M[i].X, M[i].Y), i = 1, \ldots, m$: 样例 X 满足输入形式 $I\}$; I 上的线性掩码 Γ; Γ 在 I 上涉及的比特下标集合 L_Γ; 返回 1 比特随机值的函数 Rand_bit().

输出: 只保留 Γ 线性近似特征的测试集 M_Γ.

$M_\Gamma \Leftarrow M$;
for $i \in \{1, 2, \ldots, m\}$ **do**
　for 比特下标 $j \notin L_\Gamma$ **do**
　　$M_\Gamma[i].X \Leftarrow M_\Gamma[i].X \oplus (\text{Rand\_bit}() \ll j)$;
　end for
　Γ 相关的比特位置数 $t = |L_\Gamma|$;
　生成 t 个随机比特, $S : S[j] \Leftarrow \text{Rand\_bit}(), j \in \{0, \ldots, t-1\}$;
　for $j \in \{0, \ldots, t-1\}$ **do**
　　$M_\Gamma[i].X \Leftarrow M_\Gamma[i].X \oplus ((S[j] \oplus S[(j+1)\%t]) \ll L_\Gamma[j])$;
　end for
end for
返回 M_Γ.

附录 C　Speck32/64 的 8 轮查找表的多差分 – 线性特征利用实验

本小节, 我们通过实验以验证 Speck32/64 的 13 轮攻击中使用的 8 轮查找表区分器 $\mathcal{LT}_{8r}$ 使用到了多条差分线性特征进行区分. 为此, 我们首先寻找输入差分为 $(0x40, 0)$ 的高偏差的 8 轮差分 – 线性特征. 在建立 $\mathcal{LT}_{8r}$ 的 22 个信息比特位置, 穷尽搜索所有 2^{22} 个线性掩码, 并通过实验测试各线性掩码对应线性近似的偏差, 我们找到了 6 条偏差较大的线性掩码, 各线性表达式以及相关性见表 C1, 其中式 (7) 中线性近似对应表中第 1 条线性掩码.

接着我们分别验证 $\mathcal{LT}_{8r}$ 利用到每一条线性近似的偏差进行区分. 记一条线性掩码为 Γ (对应 8 轮差分 – 线性特征 $(0x40, 0) \xrightarrow{8 \text{ rounds}} \Gamma$), 其线性近似表达式在密文形式 $I = \Delta C_L||C_L||\Delta C_Y||C_Y$ 中涉及到的密文比特下标列表为 L_Γ. 我们构造 $\mathcal{LT}_{8r}$ 的测试集 M, 并可以通过算法 C1, 在 M 中只保留每个密文对涉及 Γ 对应线性表达式的值, 同时随机化该线性特征以外的所有分布信息, 形成随机化后的测试集 M_Γ. 这样对 M_Γ 中正负例的区分只能使用到差分线性特征 $(0x40, 0) \xrightarrow{8 \text{ rounds}} \Gamma$. 若 $\mathcal{LT}_{8r}$ 可以在随机化后的测试集 M_Γ 上取得稳定大于 0.5 的准确率 (即仍存在有效的区分能力), 则说明 $\mathcal{LT}_{8r}$ 利用到了该差分 – 线性特征. 我们对搜索到的 6 条线性掩码均进行了实验, 随机化测试集后准确率结果如表 C1. 表 C1 中, Γ 是形式为 $I = \Delta C_L||C_L||\Delta C_Y||C_Y$ 对应的线性掩码, 原始 $\mathcal{LT}_{8r}$ 的准确率为 0.5137. 实验中使用的测试集 M 含有 10^9 个测例, 约含一半正例及一半负例. 10^9 的数据集带来的标准差约为 $\sigma = \sqrt{0.5 \times 0.5/10^9} = 1.58 \times 10^{-5}$. 若 $\mathcal{LT}_{8r}$ 在 M_Γ 上没有区分能力, 观测到的区分准确率不应大于 $0.5 + 3\sigma < 0.5 + 0.5 \times 10^{-4}$. 因此表 C1 准确率的显著程度可以验证, $\mathcal{LT}_{8r}$ 在区分时利用了这些差分 – 线性特征.

General enhancing framework for deep learning-aided cryptanalysis: applications to Speck, Simon and LEA ciphers

Yantian SHEN[1,3], Yi CHEN[2] & Hongbo YU[1,3,4*]

1. *Department of Computer Science and Technology, Tsinghua University, Beijing 100084, China*
2. *Institute for Advanced Study, Tsinghua University, Beijing 100084, China*
3. *Zhongguancun Laboratory, Beijing 100084, China*
4. *State Key Laboratory of Cryptography and Digital Economy Security, Tsinghua University, Beijing 100084, China*

* Corresponding author. E-mail: yuhongbo@mail.tsinghua.edu.cn

Abstract In CRYPTO 2019, Gohr first applied the deep learning techniques to the security analysis of block ciphers, opening the research direction of deep learning-aided cryptanalysis, the core of which is the differential-neural distinguisher based on deep neural networks. Compared with classical differential distinguishers, differential-neural distinguishers can achieve higher accuracy, which can help reduce the data complexity of a key recovery attack. However, the access of a differential-neural distinguisher involves a large number of floating-point operations, bringing an additional computational burden and resulting in a high computational complexity of the attack. This drawback significantly limits the performance of deep learning-aided cryptanalysis. This paper proposes a general enhancing framework for deep learning-aided cryptanalysis, which can convert a differential-neural distinguisher into a lookup table with negligible memory complexity while sacrificing little accuracy, overcoming the shortcoming of the differential-neural distinguisher and enhancing the power of deep learning-aided cryptanalysis. Experiments on Speck and Simon block ciphers designed by the National Security Agency of the USA, as well as the ISO/IEC standard LEA block cipher, have fully verified the effectiveness and generality of this framework. With the help of this framework, in this paper, the deep learning-aided cryptanalysis results on Speck32/64, Speck96 and Speck128 are improved. In particular, this paper presents the first fully verified practical key recovery attack on 13-round Speck32/64, as well as the longest practical attacks on Speck96 and Speck128. These attacks fully demonstrate the application value of this framework for enhancing deep learning-aided cryptanalysis.

Keywords deep learning, symmetric cryptanalysis, Speck, Simon, LEA