对 Raviyoyla v1 的实际伪造攻击

姚 远[1),2)] 张 斌[1)] 吴文玲[1)]

[1)](中国科学院软件研究所可信计算与信息保障实验室 北京 100190)

[2)](中国科学院大学 北京 100190)

摘 要 随着移动互联网的兴起和大数据时代的来临,人们迫切需要安全高效的认证密码算法.2013 年,在 NIST 的赞助下,Bernstein 等人发起了名为 CAESAR 的认证密码竞选.对竞选算法的安全性评估已成为当前对称密码学研究领域的热点问题.Raviyoyla v1 是提交到 CAESAR 第 1 轮竞选的候选算法之一.它是建立在 eStream 计划的候选算法 MAG v2 的基础上的流密码算法,并采用带密钥的杂凑函数进行认证.虽然设计者声称 Raviyoyla v1 具有 128 比特的完整性,但是该文成功地构造了一种针对 Raviyoyla v1 的实际伪造攻击,从而说明该算法是极不安全的.具体地,通过在明文消息中引入特殊形式的差分,攻击者能够使算法的内部状态在输出认证标签时没有差分.而且,这种差分并不局限于某些具体值,从而可以利用同一个消息得到多个伪造.理论分析表明,该形式的差分有超过 0.307143 的概率使得内部状态发生碰撞.因此,平均而言只需要大约 3 次实验即可成功地进行伪造.特别地,若将差分限定到一些特殊值上,成功概率非常接近于 1.单机实验结果显示,攻击者能够在几秒钟之内成功地进行伪造.尽管设计者针对上述攻击提出了一种可能的改进方案,但文章的进一步分析表明改进并不是本质的,修改后的算法仍然不能抵抗基于差分的伪造攻击.针对设计者提出的各种可能的修正,该文都给出了实际可行的攻击.实验证实,这些攻击具有很高的成功概率且在单机上只需花费几秒钟的时间.文章最后列举了所有可能情形下的伪造示例.据我们所知,公开文献中尚无对 Raviyoyla v1 及其改进版的认证部分的分析,因此该文对 CAESAR 竞选有重要意义.

关键词 CAESAR;Raviyoyla v1;伪造攻击;差分分析

中图法分类号 TP309 **DOI 号** 10.11897/SP.J.1016.2016.00478

A Single Query Forgery Attack on Raviyoyla v1

YAO Yuan[1),2)] ZHANG Bin[1)] WU Wen-Ling[1)]

[1)](*Trusted Computing and Information Assurance Laboratory*, *Institute of Software*, *Chinese Academy of Sciences*, *Beijing* 100190)

[2)](*University of Chinese Academy of Sciences*, *Beijing* 100190)

Abstract Raviyoyla v1 is an authenticated encryption algorithm submitted for the first round of the CAESAR competition, which is a grand occasion launched in 2013 with the support of NIST to identify efficient, flexible and secure authenticated encryption primitives. Raviyoyla v1 is composed by an additive stream cipher motivated by the eStream candidate MAG v2 and a keyed hash function. While the designer declares 128 bit security for authentication, we propose a method to construct forgeries using a single query in this paper and the complexity is negligible. Indeed, we introduce a differential of a specific form to the public message and try to canceling it before outputting any authenticated tags. Specially, the differential is not restricted to any particular value and thus multiple forgeries may be made through a single query. Our theoretical analysis shows that the probability for a randomly selected differential of our form to be canceled out is at least 0.307143. Therefore, it is sufficient to have three trials to obtain a forgery. Moreover, the probability can

收到日期:2015-02-27;最终修改稿收到日期:2015-06-08.本文已被 AsiaCCS 2015 以长摘要的形式接收.本课题得到国家"九七三"重点基础研究发展规划项目基金(2013CB338002)资助.**姚 远**,男,1989 年生,博士研究生,主要研究方向为对称密码的分析.E-mail:yaoyuan@tca.iscas.ac.cn.**张 斌**,男,1976 年生,博士,研究员,博士生导师,主要研究领域为对称密码的分析与设计.**吴文玲**,女,1966 年生,博士,研究员,博士生导师,主要研究领域为分组密码的设计与分析、分组密码工作模式和可证明安全理论的研究.

approach one for some specialized values and the attack can be applied successfully within a few seconds based on our experiments on a PC. Furthermore, the revised Raviyoyla v1 is vulnerable from our attack as well and we provide several sample forgeries for possible revisions, which are found by negligible time complexity. As far as we know, no cryptanalysis on the authentication part of Raviyoyla v1 and its revision has been proposed in public. Therefore, our work is significant for the CAESAR competition.

Keywords　CAESAR; Raviyoyla v1; forgery attack; differential cryptanalysis

1　引　言

加密和认证是密码学的两大重要课题. 尽管在过去 20 年里，NESSIE 计划①和 eStream 计划②已经成功地选出了很多优秀的密码算法，而且已有不少认证模式被提出，但是随着移动互联网的兴起和大数据时代的来临，人们迫切需要更加安全、高效的认证密码算法. 由于认证模式通常采用分组密码作为内部的随机置换，其效率显然没有直接将加密和认证设计为一体的认证密码算法高. 这很大程度上也制约了 HTTPS 等密码协议的运用.

2013 年，在 NIST 的赞助下，Bernstein 等人发起了名为 CAESAR 的算法竞选③. 竞选旨在选出比 AES-GCM④更加高效且具有广泛的适应性的认证密码算法. 截止目前，共有 57 个算法被提交到第一轮的竞选当中，其中 9 个算法因为设计缺陷被撤回. 从密码学会议近期收录的文章(ASIACRYPT 2014 上发表的对 COBRA 和 POET 的伪造攻击[1]；SAC 2014上发表的对AEGIS的线性分析[2]，对 PAES 的伪造攻击和区分攻击[3]；INDOCRYPT 2014 上发表的对 FASER128/256 和 Trivia-ck 的线性区分攻击[4]；NSS 2014 上发表的对 Sablier 的猜测确定攻击[5]和 CANS 2014 上发表的对 PANDA 的伪造攻击[6])可以看出，对 CAESAR 竞选算法的安全性评估已然成为对称密码学研究领域的热点问题.

Raviyoyla v1⑤是提交到 CAESAR 第 1 轮的竞选算法之一. 它由异或流密码和带密钥的杂凑函数组成，采用数组作为内部数据结构，类似于 eStream 计划的候选算法 MAG v2. 由于 Raviyoyla v1 采用的运算都是简单的 CPU 指令，其软件效率非常高，加密速度远远超过 AES[7]. 而且，它还可以通过多个处理器进行并行运算. 因而，如果它是安全的，则将会是一个有力的竞选算法. 最后，其设计者声称算法具有 128 比特的消息完整性和相关数据完整性.

Raviyoyla v1 算法提交后不久，Samuel Neves 给出了一个密钥流区分器⑥，但 Raviyoyla v1 的认证部分的安全性仍然有待评估. 本文主要研究了 Raviyoyla v1 及其改进版的认证部分的差分性质，并在此基础上构造了单一询问的实际伪造攻击. 差分性质反映了算法的混淆和扩散效果，是差分分析的基础. 如果能够找到高概率的差分路径，则很容易通过修改消息的方法进行伪造攻击. 事实上，作为密码分析的一种重要手段和认证安全的主要威胁，差分分析已经被成功地应用于 MD5⑦ 和 SHA-1⑧ 等杂凑函数的攻击当中，例如文献[8-9].

针对 Raviyoyla v1 的分析表明，在两次明文吸取的间隔中，算法的混淆效果非常差. 具体地，如果在消息的两个字中引入差分，则该差分有很高的概率只能传播到有限的几个数组元素中. 利用后续的消息，攻击者可以很容易地抵消这些差分. 对于随机选取的差分值，有超过 0.307 143 的概率会发生这种现象，进而攻击者只需尝试 3 次即可成功 1 次. 利用差分值的自由度，攻击者甚至可以对 1 个询问进行多个伪造. 特别地，将差分限定到一些特殊值上，伪造成功的概率接近于 1. 单机实验证实，攻击者只需几秒即可攻击成功. 作为示例，文中给出了一个消息的多个伪造.

① NESSIE. https://www.cosic.esat.kuleuven.be/nessie/ (2015.6.5)
② eSTREAM. http://www.ecrypt.eu.org/stream/project.html (2015.6.5)
③ CAESAR: Competition for authenticated encryption: Security, applicability, and robustness. http://competitions.cr.yp.to/caesar-call.html (2015.6.5)
④ McGrew D, Viega J. The Galois/Counter mode of operation (GCM). http://siswg.net/docs/gcm_spec.pdf (2015.6.5)
⑤ Vuckovac R. Raviyoyla v1. http://competitions.cr.yp.to/round1/raviyoylav1.pdf (2015.6.5)
⑥ Neves S. Raviyoyla stream generation bugs. https://groups.google.com/forum/#!searchin/crypto-competitions/Raviyoyla/crypto-competitions/_8lWn7Zm26Q/SBzM7EetUeYJ (2015.6.8)
⑦ Rivest R. The MD5 message-digest algorithm. http://www.rfc-base.org/txt/rfc-1321.txt (2015.6.5)
⑧ FIPS 180-4. Secure Hash Standard-National Institute of Standards. http://csrc.nist.gov/publications/fips/fips180-4/fips-180-4.pdf (2015.6.5)

尽管设计者针对上述攻击提出了一种改进方案,但进一步分析发现这种改进并不是本质的.对设计者提出的各种可能的改进(因为设计者对于改进的描述不是很清晰),本文都进行了详细地分析并构造出实际可行的伪造攻击.通过单机实验,本文对每种情形下的成功概率进行了估计,且给出相应的伪造示例.因此,Raviyoyla v1 的改进版仍然不能抵抗基于差分的伪造攻击.文章的最后对如何抵抗差分分析进行了探讨.

本文第 2 节介绍一些概念和记号,并概述 Raviyoyla v1 的结构;第 3 节介绍攻击思路、攻击过程,并详细分析攻击的成功概率;第 4 节探讨对改进版的攻击及其成功概率;第 5 节进行总结并探讨进一步的改进方案.

2 基本介绍

2.1 消息认证码和安全性定义

一般认为,机密性保护就是密码算法的全部目标.但事实上,完整性保护可能比机密性保护更加重要,且机密性保护无法提供完整性保护.机密性保护使得消息不会泄露给未授权的接收方;完整性保护则保证了消息不会受到未授权的篡改.不妨假设消息中包含银行转账金额或账号,若接收者无法检测到来自未授权的第三方的篡改,则他很可能会因此蒙受巨大的经济损失.

记 M 为发送的消息,M' 为接收的消息.若 $M\neq M'$,则完整性保护要求接收者能够检测出 M' 是经过篡改的(如图 1 所示).

图 1 消息认证机制

为了达到这个目的,发送方通常会在发送 M 的同时,附上由消息认证码生成的标签 tag.相应地,接收方需要检测接收到的 M' 和 tag' 是否满足特定的要求,然后决定相应的操作(如图 2 所示).

图 2 消息认证码

形式上,消息认证码包含 3 个算法:密钥生成算法 $\mathcal{K}$、标签生成算法 MAC 和确定性的认证算法 VF.但实际上,标签生成算法和认证算法通常是一样的(即先对 M' 调用标签生成算法得到 tag'',然后检查 tag' 和 tag'' 是否相同),所以只需定义标签生成算法即可.

为了抵抗所有可能的敌手,标准的伪造攻击模型假设敌手拥有标签生成算法和认证算法的问答机(如图 3 所示),于是消息认证码的安全性按照如下方式进行定义.

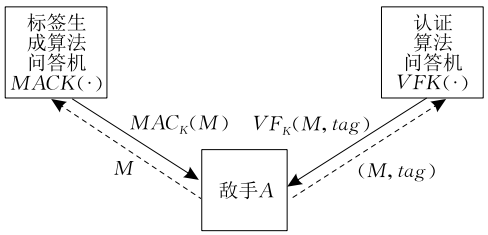

图 3 伪造攻击模型

定义 1[⑨]. 设 $\Pi=(\mathcal{K},MAC,VF)$ 是消息认证码,A 是敌手.考虑如下实验.

实验. $Exp_{\Pi}^{uf\text{-}cma}(A)$.

$K \xleftarrow{\text{随机选取}} \mathcal{K}$

运行 $A^{MAC_K(\cdot),VF_K(\cdot)}$

若 A 构造出 (M,tag) 使得

(1) 认证问答机返回合法

(2) A 没有向 MAC 询问过 M 的标签,且没有向 VF 询问过 (M,tag) 是否合法

则,返回成功;否则,返回失败.

于是,A 的 $uf\text{-}cma$ 优势定义为

$$Adv_{\Pi}^{uf\text{-}cma}(A)=\Pr\{Exp_{\Pi}^{uf\text{-}cma}(A)\text{ 返回真}\}$$

简而言之,如果敌手能够构造出未经询问的的合法的 (M,tag),则伪造成功.敌手成功概率越高,则敌手的 $uf\text{-}cma$ 优势越大,消息认证码的安全性越低.

2.2 CAESAR 竞选

目前,消息认证码大都通过分组密码结合认证模式生成.尽管其安全性已经得到证明,但随着人们对效率的要求越来越高,对认证加密算法的需求也变得越来越迫切.CAESAR 竞选旨在选出比 AES-GCM 更加高效且具有广泛的适应性的认证密码算法.

此外,CAESAR 竞选提出了两个新的概念作为算法的输入,分别是相关数据和秘密向量.相关数据

⑨ Goldwasser S, Bellare M. Lecture Notes on Cryptography. http://cseweb.ucsd.edu/~mihir/papers/gb.pdf, 2008: 156 (2015.6.5)

类似于消息，但不要求完整性；秘密向量类似于公开向量，且要求能够保密. 显然，相关数据可以用于一些公开参数的设定. 但是，秘密向量的用途则不太明确，因此并没有得到广泛的支持. 表 1 总结了 CAESAR 竞选对各个输入的安全性要求.

表 1　安全性要求[③]

	完整性	保密性	不可重复
消息	是	是	否
相关数据	是	否	否
秘密向量	是	是	是
公开向量	是	否	是

以下用 ad 表示相关数据，且不考虑秘密向量（因为 Raviyoyla v1 不支持秘密向量）. 数组下标从 0 开始计数，$a[i_1,\cdots,i_j]$ 表示从 i_1 到 i_j 的数组元素，也即 $a[i_1],\cdots,a[i_j]$. 若

$$a[i_1]=\cdots=a[i_j]=x,$$

则将其简记为 $a[i_1,\cdots,i_j]=x$. $\oplus$ 表示异或，$\Delta b=b\oplus b'$ 表示 b 与 b' 之间的差分.

2.3　Raviyoyla v1 概述

Raviyoyla v1 由同步流密码（即密钥流的生成与消息无关）和带密钥的杂凑函数构成，其密钥长度和 IV 长度均为 256 比特，标签长度为 512 比特. 设计者声称它对明文具有 256 比特机密性保护和 128 比特完整性保护，且对相关数据和公开向量具有 128 比特完整性保护. 目前，公开文献中尚无对 Raviyoyla v1 的认证部分的安全性分析，包括设计文档.

Raviyoyla v1 首先将密钥流与消息进行异或加密，然后将密文打入内部状态进行认证，最后输出标签. 其中使用的流密码类似于 eStream 计划的候选算法 MAG v2，都是基于数组结构的. 数组大小为 1028，元素大小为 64 比特. 算法的大致流程如图 4 所示. 由于本文的攻击不涉及密钥流生成，故只介绍认证相关的部分. 更多细节参见算法的设计文档[⑤].

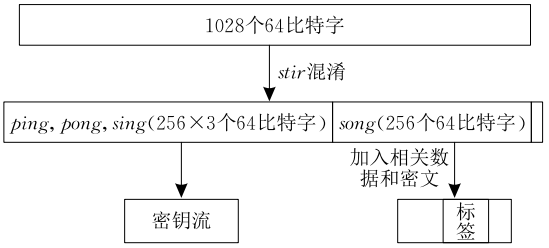

图 4　Raviyoyla v1 的算法流程

分别用 $adlen$ 和 $mlen$ 表示相关数据和消息的长度（以 64 比特为基本单位），则图 4 中加入相关数据和密文的过程由如下的 C 代码[⑩]定义.

```
1.  for (i=0; i<adlen*8; ++i){
        //加入相关数据
2.    ((char*)song)[i%2048]^=((char const*)ad)[i];
3.    if(i%2047==0 && i>0){
          //混淆
4.      revolve_all(song, 256, &carry);
5.    }
6.  }
7.  for(j=0; i<(adlen+mlen)*8; ++i, ++j){
        //加入密文
8.    ((char*)song)[i%2048]^=(char const*)c[j];
9.    if(i%2047==0){
          //混淆
10.     revolve_all(song,256,&carry);
11.   }
12. }
13. if (i%2047>0){
        //混淆
14.   revolve_all(song,256,&carry);
15. }
16. evolve(song,256);
```

其中 *revolve_all* 函数定义为

```
17. void revolve_all(uint64_t *song, int elen, uint64_t
    *carry){
18.   for(int i=0;i<elen; ++i){
19.     if(song[(i+2)%elen]>song[(i+3)%elen]){
20.       *carry^=song[(i+1)%elen];
21.     }else{
22.       *carry^=~song[(i+1)%elen];
23.     }
24.     song[i]^=*carry;
25.   }
26. }
```

注：Raviyoyla v1 设计文档中的实现[⑤,⑥]是有问题的：在输出若干个字节后密钥流全为零. 附录 A 给出了将问题修复之后的代码. 幸运地是，除了具体的伪造不一样以外，本文的攻击方法对两种实现都适用.

3　对 Raviyoyla v1 的伪造攻击

3.1　基本思想

由算法流程可以看出，设计者试图利用 *revolve_all* 函数进行混淆，以使标签的每个比特都依赖于相关数据和消息. 因为 *revolve_all* 函数使用的操作均受 CPU 指令的支持，所以它拥有很高的软件实现效率. 不幸的是，*revolve_all* 函数过于简单，远没有

⑩　注：为了叙述简单，这里给出的是经过修改之后的等价代码，而非原始代码.

达到应有的混淆效果.

事实上,*revolve_all* 的非线性操作只有 IF-ELSE 语句.因此,如果不考虑 IF-ELSE 语句或者 IF-ELSE 语句总是选择同样的分支进行操作,则 *song*、*carry* 与相关数据、密文之间的关系完全是线性的.在这种情况下,由密文(或相关数据)引入的差分的传播路径将很容易被预测,进而被后续的差分抵消.又设攻击者已经构造出合适的密文差分,由于 Raviyoyla v1 是一种异或流密码,攻击者很容易从一次询问中伪造出新的消息:只需将密文差分异或到原消息当中.

虽然比较运算依赖于变量的所有比特,且在随机情形下为真和假的概率都是 1/2,但它对不同比特的依赖性差异很大.显然,比较运算对高位的比特更敏感.因此,如果攻击者将状态的差分限制到低位上,则比较运算的结果很可能是一样的,即当 $\Delta a = a \oplus a'$, $\Delta b = b \oplus b'$ 很小时,比较运算 $a < b$ 的结果和 $a' < b'$ 的结果很可能相同.

下面介绍具体的攻击过程.

3.2 攻击过程

本节将给出一种单一询问的伪造攻击.假设 $M=(v,ad,m)$ 是攻击者向标签生成算法的问答机询问的一条数据,$M'=(v,ad',m')$ 是攻击者伪造的消息(其中 v 表示公开向量,ad,ad' 表示相关数据,m,m' 表示消息),则它们之间的差分为

$$\Delta ad = ad \oplus ad',\ \Delta m = m \oplus m'.$$

考虑 $adlen=0$ 或 $\Delta ad=0$(即没有相关消息或相关消息没有差分)的情形.因为代码段第 1～6 行不涉及明文或密文,故对差分的传播没有影响.若 Δm 只在 $4,5,256+3,256+4,256+5$ 上不为 0,且

$$\Delta m[4,5,256+3,256+4,256+5] = \delta \neq 0,$$

则 m 和 m' 的密文之间的差分为

$$\Delta c[4,5,256+3,256+4,256+5] = \delta.$$

由代码第 8 行可知,当 $0 \leq i \leq 2047$ 时(注意:*song* 是以 64 比特字为单位,而 C 代码是以 8 比特字节为单位,所以此时只有 2048/8=256 个密文被异或到 *song* 当中)有

$$\Delta song[4,5] = song[4,5] \oplus song'[4,5] = \delta,$$

又此时 $i = 0 \bmod 2047$,代码进入第 20 行,并调用 *revolve_all* 函数.因为 *song* 在 0,1,2,3 上没有差分,故 $i=0$ 时,两个消息在第 19 行的比较结果相同.若 $i=1,2,3$ 时,第 19 行的比较结果也相同,则易知 *revolve_all* 函数返回时有

$$\Delta song[3,4,5] = \delta,$$
$$\Delta carry = carry \oplus carry' = 0.$$

从而运行至第 16 行时有 $\Delta song = \Delta c = 0$,进而两个消息有同样的标签,故 $(v, ad, c \oplus \Delta c, tag)$ 有一定的概率是合法的(其中 c 是 m 对应的密文).图 5 展示了该差分的传播路径.

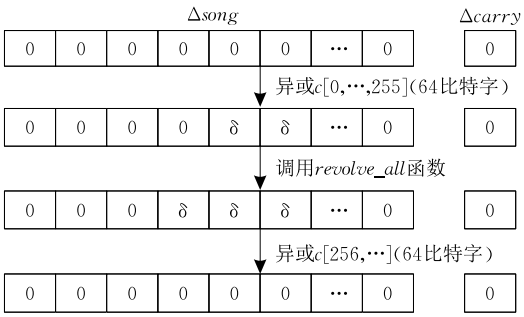

图 5 Raviyoyla v1 差分传播路径

3.3 成功概率

显然,只要 3.2 节中的假设

$$(song[i+2] > song[i+3]) = (song'[i+2] > song'[i+3]),\ i=1,2,3 \quad (1)$$

成立,则攻击可以成功.

定理 1. 设 $song[3,4,5,6]$,δ 独立均匀同分布,则式(1)成立的概率为 0.307143.

证明. 见附录 B.

因为式(1)只是攻击成功的一个充分条件,为了得到更准确的成功概率,我们利用如下的算法 1 进行实验验证.

算法 1. 估计成功概率.

```
输入:N(样本数量)
输出:成功概率
cnt=0
for(i=0; i<N; ++i){
  随机生成 Raviyoyla v1 的参数
  加密一个长度为 262 的随机消息
  随机生成 δ,并修改消息
  用同样的参数加密新消息
  if(两个消息的标签相同){
    ++cnt
  }
}
return cnt/N
```

实验显示,真实的成功概率确实比 0.307143 略大.而且对一些特殊的 δ,成功概率可接近于 1(附录 C 给出了不同的 δ 对应的成功概率).

显然,如果选取适当的 k 和 l,并将差分引入 Δm 的如下位置:

$$256k+l,\ 256k+l+1,$$
$$256(k+1)+l-1,$$

$256(k+1)+l$, $256(k+1)+l+1$,

则攻击仍然有效. 又因为加入相关消息和加入密文的过程类似,如果将差分引入相关消息的相应位置,则也可进行类似的攻击. 最后,表 2 给出上述攻击得到的伪造示例,其中

$$adlen=0,\ mlen=262,$$

$k[0,\cdots,31]=v[0,\cdots,31]=m[0,\cdots,31]=0$,

k 为密钥,且每个实现都给出了 3 个伪造(分别对应于不同的 δ). 寻找这些伪造所花费的时间可忽略,内存为 $O(1)$.

表 2　对 Raviyoyla v1 伪造示例

δ(十六进制)	tag(十六进制)
针对设计者的实现	
212c27dd2033b7ba	a8d917ee8265c832 1aaa669e542d000a 42c139dac83907af 8bc69fa9239419d4
396cb4981bc8be6f	432b4647caf15efe b1646e7d647ec2a0 959df1737c775ba7 fc8ed1e19e4f648e
973bbec0b5e56ec1	
针对附录 A 的实现	
1c34c2aef4fec1ba	ca211d2abdd1cc52 fd89247d9cc21839 00c7f0e784347f9b 2c29c35824693796
0d7ab02e032289bb	5a6b38df38950708 04e8a8ae036041ae bbc3d5cd5c8f7a14 30daef437f2f2cdc
c9722f3a9b122326	

4　对改进版的伪造攻击

4.1　改进方案

尽管设计者在网上论坛[11]中给出了针对上述攻击的改进方案(注意其中的 *state* 在本文中为 *song*):

"The remedy would be changing '*revolve_all*' indexing procedure. Instead of taking index from the loop counter, the next index of array element for processing is derived from previous element mod 256. That looks $j=state[j]\ \%$ 256 (of course loop counter for revolve all is now k). The j is initialized to zero and is not reinitialized afterwards. This tweak will be implemented on the first opportunity available."

但其根本缺陷仍然存在. 鉴于作者给出的改进方案不是特别明确(即哪些地方的下标需要修改),在后续的讨论中,我们将作出如下几个假设.

A. 若代码第 19 行被修改,则修改为

19. if(*song*[(*j*+2) % *elen*]>*song*[(*j*+3) % *elen*]){

B. 若代码第 20 行被修改,则修改为

20. **carry*^=*song*[(*j*+1) % *elen*];

C. 若代码第 22 行被修改,则修改为

22. **carry*^=~*song*[(*j*+1) % *elen*];

D. 若代码第 24 行被修改,则修改为

24. *song*[*j*]^=**carry*;

E. 第 20 行被修改当且仅当第 22 行被修改

F. j 的初始值为 0 且在第 18 行和第 19 行之间进行更新,更新方式为

18.5. $j=song[j]\ \%\ elen$;

4.2　攻击过程

类似 3.2 节,考虑没有相关消息或相关消息没有差分的情形.

4.2.1　第 20、22 行不改变时

此时 *revolve_all* 函数定义如下,其中 i',i''根据修改的具体情形等于 i 或 j.

```
17. void revolve_all(uint64_t *song, int elen, uint64_t
      *carry){
18.   for(int i=0, j=0; i<elen; ++i){
18.5.   j=song[j]%elen;
19.     if(song[(i'+2)%elen]>song[(i'+3)%elen]){
20.       *carry^=song[(i+1)%elen];
21.     }else{
22.       *carry^=~song[(i+1)%elen];
23.     }
24.     song[i'']^=*carry;
25.   }
26. }
```

对较小的 δ 满足 $\delta=0 \bmod elen$,设 Δm 只在 $254,255,256+pos,256+254,256+255$ 处不为 0(其中 pos 待定),且

$$\Delta m\begin{bmatrix}254,255,\\256+pos,256+254,256+255\end{bmatrix}=\delta,$$

则密文差分为

$$\Delta c\begin{bmatrix}254,255,\\256+pos,256+254,256+255\end{bmatrix}=\delta,$$

由代码第 8 行可知,当 $0\le i\le 2047$ 时

$$\Delta song[254,255]=\delta,$$

紧接着调用 *revolve_all* 函数.

---

[11] Vuckovac R. Revised Raviyoyla. https://groups.google.com/for um/#!searchin/crypto-competitions/Raviyoyla/crypto-competitions/x15PAQ_5uNk/pDCQsNIwBikJ (2015.6.5)

由

$$song[j]=song[j]\oplus\delta \bmod elen \quad (2)$$

可知⑫,对 m 和 m',*revolve_all* 函数中的 j 在不同时刻的值总是相同的,故 j 不受差分影响.

若在第 19 行时,

$$i'\in\{251,252,253\},$$

则 IF-ELSE 分支可能会受到差分 δ 的影响.显然,不论 i'等于 i 还是 j,都不会有太多的 i'落入该集合.具体地,若 $i'=i$,则恰好 3 次;若 $i'=j$ 且服从独立均匀分布,则发生 n 次的概率为$(1/256)^n$.但即便发生这样的情形,由定理 1 可知,比较结果也很可能不受差分的影响.对较小的 δ,这个概率更高.总之,对 m 和 m',IF-ELSE 分支的比较结果有很大的概率是相同的.

令 *pos* 为 $i=253$ 时第 24 行代码中 i''的值.综上所述,当 *revolve_all* 函数返回时,下式有很高的概率成立

$$\Delta song[pos,254,255]=\delta,\ \Delta carry=0,$$

故,在加入余下的密文后,状态的差分为零.进而,得到同样的标签.若 $i''=j$,则 *pos* 的值是未知的.但 $0\le pos\le 255$,故攻击者可以穷搜可能的值.总之,*pos* 的值是很容易确定的.

4.2.2 第 20、22 行改变时

此时 *revolve_all* 函数定义为

```
17. void revolve_all(uint64_t *song, int elen, uint64_t
      *carry){
18.   for(int i=0, j=0; i<elen;++i){
18.5.   j=song[j]%elen;
19.     if(song[(i'+2)%elen]>song[(i'+3)%elen]){
20.       *carry^=song[(j+1)%elen];
21.     }else{
22.       *carry^=~song[(j+1)%elen];
23.     }
24.     song[i'']^=*carry;
25.   }
26. }
```

其中 i',i''根据修改的具体情形等于 i 或 j.事实上,在这种情形下算法抵抗差分分析的能力更弱.令 Δm 只在 255 和 256+255 处不为 0 且

$$\Delta m[255,256+255]=\delta=0 \bmod elen.$$

同 4.2.1 节可知,j 不受差分的影响.若 *song*[255] 从未被异或到 *carry* 中(即 j 不取 254),则差分不会扩散到其它单元.故当 *revolve_all* 函数返回时,差分只存在于 $\Delta song[255]$.进而在加入余下的密文后,状态的差分为 0.若 j 服从独立均匀分布,则概率为$(255/256)^{256}\approx e^{-1}$,故该事件有很高的概率会发生.

此外,如果 $\delta\neq 0 \bmod elen$,该攻击仍可能成功.这是因为 j 由 $song[j]\%elen$ 进行更新,很可能永远不会等于 254,从而不受差分影响.若第 19 行的 IF-ELSE 语句也不受差分影响,则 *carry* 不受差分影响.

4.3 成功概率

由于可能的改进方案较多,因而为每种情形都推导其成功概率将会非常复杂和繁琐.故本文只对上述攻击实验估计其成功概率和代价.算法 2 是实验所使用的伪代码.显然,其消耗的内存是 $O(1)$.又单机实验表明攻击所需的时间为几秒,因而攻击是实际可行的且时间代价可忽略.表 3 给出了在不同情形下伪造成功的概率,伪造示例见附录 D.

表 3 对 Raviyoyla v1 改进版的攻击(各 1000 次实验)

修改方案	成功概率	
	对设计者的实现	附录 A 的实现
第 19 行改变	1.000	1.000
第 20、22 行改变	0.752	0.778
第 24 行改变	0.996	0.997
第 19、20、22 行改变	0.761	0.749
第 20、22、24 行改变	0.368	0.375
第 19、24 行改变	0.998	0.998
第 19、20、22、24 行改变	0.352	0.343

算法 2. 改进版的成功概率.

```
输入:N(样本数量)
输出:成功概率
δ=0000000000000100(十六进制)
cnt=0
for(i=0;i<N;++i){
  随机生成 Raviyoyla v1 改进版的参数
  加密一个长度为 512 的随机消息
  修改消息(若 pos 未知,则重复以下过程,直至 pos 从 0
          遍历到 255)
  用同样的参数加密新消息
  if(两个消息的标签相同){
    ++cnt
  }
}
return cnt/N
```

5 结 语

本文研究了 CAESAR 第 1 轮竞选算法 Raviyoyla v1 及其改进版的认证部分的差分性质,并利用其中

⑫ 注意:对一般的 $elen\neq 2^x$,式(2)并不一定成立.

存在的弱点构造伪造攻击.

Raviyoyla v1 在认证时的混淆操作主要由 *revolve_all* 函数完成，但由于它采用的操作过于简单，导致其在密文差分尚未完全扩散的情形下就加入下一段密文. 理论分析表明，对于文中构造的特殊形式的差分，其成立的概率至少为 0.307143. 对于特定的差分值，概率甚至接近于 1. 实验结果显示，攻击者能够在几秒之内成功地进行伪造. 利用差分的自由度，攻击者还可以对一个消息构造多个伪造. 作为示例，本文给出了一些伪造示例.

尽管设计者针对上述攻击提出了改进方案，但其根本缺陷仍未得到修正. 对于可能的改进，本文均给出了实际可行的高概率伪造攻击，且时间复杂度可忽略. 因而，Raviyoyla v1 远不是一个安全的认证流密码.

由于上述攻击并不涉及密钥流生成，下面讨论认证部分的两种可能的改进. 考虑到相关消息和密文的相似性，这里只考虑密文差分.

(1) 如果设计者坚持将密文异或到整个 *song*，则应当在充分混淆之后再加入下一段密文. 设第一段密文的差分为 Δ，则经过混淆之后它传播到任意差分的概率应该小于或等于 2^{-128}（假设 $\Delta carry=0$）. 否则，存在 Δ' 使得

$$\Pr(\Delta \rightarrow \Delta') > 2^{-128},$$

选取下一段密文的差分为 Δ'，则差分有可能被抵消. 此时，攻击复杂度小于 2^{128}. 若混淆之后 $\Delta carry\neq 0$，适当修改下一段消息的差分即可.

(2) 将密文异或到 *song* 的部分位置，采用海绵(sponge)结构[13]进行认证. 具体地，可以使用杂凑函数 Keccak[14] 进行认证. 若存在 m, m' 使得标签相同，相应的 c, c' 会导致 Keccak 的碰撞. 从目前 Keccak 的分析结果来看，这不太可能.

事实上，因为在加入下一段密文之前必须充分混淆以降低差分概率，Raviyoyla v1 采用的先加密后认证的方式在效率上难以提高. 作为认证加密算法，应该采用自然认证（密钥流依赖于消息）或边加密边认证的方式（参考 Grain 128a[10] 或 Sablier[15]）.

参　考　文　献

[1] Nandi M. Forging attacks on two authenticated encryption schemes COBRA and POET//Proceedings of the Advances in Cryptology—ASIACRYPT. Kaohsiung, China, 2014: 126-140

[2] Minaud B. Linear biases in AEGIS keystream//Proceedings of the Selected Areas in Cryptography—SAC. Montreal, Canada, 2014: 290-305

[3] Jean J, Nikolić I, Sasaki Y, et al. Practical cryptanalysis of PAES//Proceedings of the Selected Areas in Cryptography—SAC. Montreal, Canada, 2014: 228-242

[4] Xu Chao, Zhang Bin. Linear cryptanalysis of FASER128/256 and TriviA-ck//Proceedings of the Progress in Cryptology—INDOCRYPT. New Delhi, India, 2014: 237-254

[5] Feng X, Zhang F. Cryptanalysis on the authenticated cipher sablier//Proceedings of the Network and System Security. Xi'an, China, 2014: 198-208

[6] Sasaki Y, Wang Lei. Message extension attack against authenticated encryptions: Application to PANDA//Proceedings of the Cryptology and Network Security—CANS. Heraklion, Greece, 2014: 82-97

[7] Daemen J, Rijmen V. The Design of Rijndael: AES—The Advanced Encryption Standard. New York: Springer Science & Business Media, 2002

[8] Wang Xiao-Yun, Yu Hong-Bo. How to break MD5 and other hash functions//Proceedings of the Advances in Cryptology—EUROCRYPT. Aarhus, Denmark, 2005: 19-35

[9] Wang Xiao-Yun, Yin Yi-Qun Lisa, Yu Hong-Bo. Finding collisions in the full SHA-1//Proceedings of the Advances in Cryptology—CRYPTO. Santa Barbara, USA, 2005: 17-36

[10] Agren M, Hell M, Johansson T, et al. Grain-128a: A new version of Grain-128 with optional authentication. International Journal of Wireless and Mobile Computing, 2011, 5(1): 48-59

附录 A.　Raviyoyla v1 源码.

以下是修改之后的 C 代码，其中标注了与设计者提供的代码的不同之处.

```
#define CRYPTO_KEYBYTES 32
#define CRYPTO_NPRIVBYTES 0
#define CRYPTO_NPUBBYTES 32
#define CRYPTO_ABYTES 64

#include <stdint.h>
#include <string.h>
#include <stdlib.h>

void stir(uint64_t *state, int statelen){
  uint64_t mixer=6148914691236517205;
  uint64_t carry=1234567890123456789;
```

[13] Bertoni Guido, Daemen Joan, Peeters Michaël, et al. Cryptographic sponge functions. http://sponge.noekeon.org/CSF-0.1.pdf (2015.6.5)

[14] Bertoni Guido, Daemen Joan, Peeters Michaël, et al. The Keccak sponge function family. http://keccak.noekeon.org/(2015.6.5)

[15] Zhang Bin, Shi Zhenqing, Xu Chao, et al. Sablier v1. http://competitions.cr.yp.to/round1/sablierv1.pdf (2015.6.5)

```
  int i, j, rounds=8;
  for (i=0; i<rounds; i++){
    for (j=0; j<statelen; j++){
      if (state[(j+2) % statelen]>state[(j+3) %
                                        statelen])
        carry^=state[(j+1) % statelen];
      else
        carry^=~state[(j+1) % statelen];
      state[j]^=carry;
      carry+=mixer;
    }
  }
}
uint64_t revolve_step(uint64_t *state, int i, int elen,
                      uint64_t *carry){
  if (state[(i+2) % elen]>state[(i+3) % elen])
    *carry^=state[(i+1) % elen];
  else
    *carry^=~state[(i+1) % elen];
  state[i]^=*carry;
  return state[i];
}
void get_stream(uint64_t *ignite, int statelen, char
                *stream_random,
  unsigned long long mlen){
  uint64_t *ping, *pong, *sing;
  uint64_t ping_carry, pong_carry, sing_carry;
  uint64_t three, i;
  uint64_t end=mlen /8;
  ping=&ignite[0];
  pong=&ignite[256];
  sing=&ignite[512];
  ping_carry=ignite[statelen-4];
  pong_carry=ignite[statelen-3];
  sing_carry=ignite[statelen-2];
  for (i=0; i<end; i++){
    three=revolve_step(ping, i, (statelen/4)-1,
                       &ping_carry)
        ^revolve_step(pong, i, (statelen /4)-1,
                       &pong_carry)
        ^revolve_step(sing, i, (statelen /4)-1,
                       &sing_carry);
    //original
    //memmove(stream_random+(i/8), (uint64_t*)
              &three, 8);
    //revised
    memmove(stream_random+(i*8), (uint64_t*)
              &three, 8);
  }
  //original
  //if((end%8)>0){
  //revised
  if ((mlen%8)>0){
    three=revolve_step(ping, i, (statelen/4)-1,
                       &ping_carry)
        ^revolve_step(pong, i, (statelen/4)-1,
                       &pong_carry)
        ^revolve_step(sing, i, (statelen/4)-1,
                       &sing_carry);
    //original
    //memmove(stream_random+(i/8), (uint64_t*)
              &three, end%8);
    //revised
    memmove(stream_random+(i*8), (uint64_t*)
              &three, mlen%8);
  }
}
void revolve_all(uint64_t *state, int elen, uint64_t *carry){
  int i;
  for (i=0; i<elen; i++){
    if (state[(i+2) % elen]>state[(i+3) % elen]){
      *carry^=state[(i+1) % elen];
    }
    else{
      *carry^=~state[(i+1) % elen];
    }
    state[i]^=*carry;
  }
}
void evolve(uint64_t *state, int statelen){
  int i, j;
  for (i=0; i<statelen; i++){
    for (j=0; j<statelen; j++){
      if (state[(j+1) % statelen]>state[(j+3) % statelen])
        state[j % statelen]^=state[(j+1) % statelen];
      else
        state[j % statelen]^=~state[(j+1) % statelen];
      if (state[(j+2) % statelen]>state[(j+3) %
            statelen])
        state[j % statelen]=state[(j+2) % statelen];
      else
        state[j % statelen]^=~state[(j+2) % statelen];
      if (state[(j+3) % statelen] % 2==1)
        state[j % statelen]^=state[(j+3) % statelen];
      else
```

```
        state[j%statelen]^=~state[(j+3)%statelen];
      }
    }
  }
  int crypto_aead_encrypt(
    unsigned char *c, unsigned long long *clen,
    const unsigned char *nsec,
    const unsigned char *m, unsigned long long mlen,
    const unsigned char *ad, unsigned long long adlen,
    const unsigned char *npub, const unsigned char *k){
    int statelen=1028;
    uint64_t ignite[1028]={0};
    uint64_t *song;
    uint64_t song_carry;
    uint64_t i,j;
    char *stream_random, *song_char;
    memmove(&ignite[0],k,CRYPTO_KEYBYTES);
    memmove(&ignite[4],npub,CRYPTO_NPUBBYTES);
    ignite[statelen-1]=mlen;
    ignite[statelen-2]=adlen;
    stir(ignite, statelen);
    stream_random=(char*)calloc(mlen,sizeof(char));
    if (stream_random==NULL){
      printf("Error allocating requested memory");
      exit(1);
    }
    get_stream(ignite, statelen, stream_random, mlen);
    for (i=0; i<mlen; i++)
      c[i]=m[i]^stream_random[i];
    free(stream_random);
    song=&ignite[768];
    song_carry=ignite[statelen-1];
    song_char=(char*)song;
    for (i=0; i<adlen; i++){
      song_char[i%2048]^=ad[i];
    //original
    //if (i%2047==0&&i>0){
    //revised
      if((i+1)%2048==0){
        revolve_all(song, 256, &song_carry);
      }
    }
    for (i=adlen, j=0; i<(adlen+mlen);
         i++, j++){
      song_char[i%2048]^=c[j];
    //original
    //if (i%2047==0) {
    //revised
      if ((i+1)%2048==0){
        revolve_all(song, 256, &song_carry);
      }
    }
    //original
    //if (i%2047>0) {
    //revised
    if (i%2048>0){
      revolve_all(song, 256, &song_carry);
    }
    evolve(song, 256);
    memmove(&c[mlen], &song[256/2],
            CRYPTO_ABYTES);
    return 0;
  }
  int crypto_aead_decrypt(
    unsigned char *m, unsigned long long mlen,
    unsigned char *nsec,
    const unsigned char *c, unsigned long long clen,
    const unsigned char *ad, unsigned long long adlen,
    const unsigned char *npub, const unsigned char *k){
    /*…
    …the code for the cipher implementation goes here,
    …generating a plaintext m[0],m[1],…,m[mlen-1]
    …and secret message number nsec[0],nsec[1],…
    …from a ciphertext c[0],c[1],…,c[clen-1]
    …and associateddata ad[0],ad[1],…,ad[adlen-1]
    …and public message number npub[0],npub[1],…
    …and secret key k[0], k[1], …
    … */
    return 0;
  }
```

附录 B.　定理 1 证明.

下面采用数学归纳法进行证明. 若 $x_{(n)}, y_{(n)}$ 是由 x, y 的低 n 比特构成的整数,定义

$$cmp_n(x,y)=\begin{cases}1, & \text{若 } x_{(n)}>y_{(n)}\\ 0, & \text{否则}\end{cases}$$

令

$$\begin{aligned}&a_i=song[i],\ a_i'=song'[i],\\ &p1_n=\Pr(cmp_n(a_3,a_4)=cmp_n(a_3',a_4'),\\ &\qquad cmp_n(a_4,a_5)=cmp_n(a_4',a_5'),\\ &\qquad cmp_n(a_5,a_6)=cmp_n(a_5',a_6'))\\ &p2_n=\Pr(cmp_n(a_3,a_4)=cmp_n(a_3,a_4'),\\ &\qquad cmp_n(a_4,a_5)=cmp_n(a_4',a_5'))\end{aligned}$$

$$p3_n=\Pr(cmp_n(a_3,a_4)=cmp_n(a_3,a_4'),\ cmp_n(a_5,a_6)=cmp_n(a_5',a_6)$$

$$p4_n=\Pr(cmp_n(a_4,a_5)=cmp_n(a_4',a_5'),\ cmp_n(a_5,a_6)=cmp_n(a_5',a_6))$$

$$p5_n=\Pr(a_{3(n)}\le a_{4(n)},a_{5(n)}>a_{6(n)},\ cmp_n(a_4,a_5)=cmp_n(a_4',a_5'))$$

$$p6_n=\Pr(a_{3(n)}\le a_{4(n)},a'_{5(n)}\le a_{6(n)}\ cmp_n(a_4,a_5)=cmp_n(a_4',a_5'))$$

$$p7_n=\Pr(a_{3(n)}>a_{4(n)},a'_{5(n)}>a_{6(n)},\ cmp_n(a_4,a_5)=cmp_n(a_4',a_5'))$$

$$p8_n=\Pr(a_{3(n)}>a_{4(n)},a_{5(n)}\le a_{6(n)},\ cmp_n(a_4,a_5)=cmp_n(a_4',a_5'))$$

$$p9_n=\Pr(cmp_n(a_3,a_4)=cmp_{n-1}(a_3,a_4'))$$

$$p10_n=\Pr(cmp_{n-1}(a_4,a_5)=cmp_{n-1}(a_4',a_5'))$$

$$p11_n=\Pr(cmp_{n-1}(a_5,a_6)=cmp_n(a_5',a_6))$$

$$p12_n=\Pr(x_{(n)}\le y_{(n)})$$

$$p13_n=\Pr(a_{5(n)}>a_{6(n)},cmp_n(a_4,a_5)=cmp_n(a_4',a_5'))$$

$$p14_n=\Pr(a_{3(n)}\le a_{4(n)},cmp_n(a_4,a_5)=cmp_n(a_4',a_5'))$$

$$p15_n=\Pr(a'_{5(n)}\le a_{6(n)},cmp_n(a_4,a_5)=cmp_n(a_4',a_5'))$$

$$p16_n=\Pr(a_{3(n)}>a_{4(n)},cmp_n(a_4,a_5)=cmp_n(a_4',a_5'))$$

故需要证明

$$p1_{64}=0.307143$$

首先,穷举 $a_i,a_i',i=3,4,5,6$ 的最低比特得

$$p9_1=p10_1=p11_1=p12_1=\frac{3}{4}$$

$$p1_1=p14_1=p15_1=\frac{9}{16}$$

$$p2_1=p3_1=p4_1=\frac{5}{8}$$

$$p13_1=p16_1=\frac{3}{16}$$

$$p5_1=p8_1=\frac{5}{32}$$

$$p6_1=\frac{7}{16}$$

$$p7_1=\frac{1}{16}$$

然后,对 $a_i,a_i',i=3,4,5,6$ 的第 n 比特进行归纳.

$$p1_n=\frac{1}{2}\left(\frac{2}{16}+\frac{2}{16}p1_{n-1}+\frac{2}{16}p2_{n-1}+\frac{2}{16}p3_{n-1}+\frac{2}{16}p4_{n-1}+\frac{2}{16}p9_{n-1}+\frac{2}{16}p10_{n-1}+\frac{2}{16}p11_{n-1}\right)+\frac{1}{2}\left(\frac{2}{16}p5_{n-1}+\frac{2}{16}p6_{n-1}+\frac{2}{16}p7_{n-1}+\frac{2}{16}p8_{n-1}\right)$$

$$p2_n=\frac{1}{8}+\frac{1}{8}p2_{n-1}+\frac{1}{8}p9_{n-1}+\frac{2}{8}p10_{n-1}$$

$$p3_n=\frac{2}{8}+\frac{1}{8}p3_{n-1}+\frac{1}{8}p9_{n-1}+\frac{1}{8}p11_{n-1}$$

$$p4_n=\frac{1}{8}+\frac{1}{8}p4_{n-1}+\frac{1}{8}p11_{n-1}+\frac{2}{8}p10_{n-1}$$

$$p5_n=\frac{1}{32}+\frac{1}{8}p5_{n-1}+\frac{1}{16}p10_{n-1}+\frac{1}{16}p13_{n-1}+\frac{1}{16}p14_{n-1}+\frac{1}{16}p12_{n-1}(1-p12_{n-1})$$

$$p6_n=\frac{1}{32}+\frac{1}{8}p6_{n-1}+\frac{1}{32}p10_{n-1}+\frac{1}{16}p12_{n-1}+\frac{1}{16}p14_{n-1}+\frac{1}{16}p15_{n-1}+\frac{1}{16}p12_{n-1}^2$$

$$p7_n=\frac{1}{32}+\frac{1}{8}p7_{n-1}+\frac{1}{32}p10_{n-1}+\frac{1}{16}(1-p12_{n-1})+\frac{1}{16}p13_{n-1}+\frac{1}{16}p16_{n-1}+\frac{1}{16}(1-p12)_{n-1}^2$$

$$p8_n=\frac{1}{32}+\frac{1}{8}p8_{n-1}+\frac{1}{16}p10_{n-1}+\frac{1}{16}p15_{n-1}+\frac{1}{16}p16_{n-1}+\frac{1}{16}p12_{n-1}(1-p12_{n-1})$$

$$p9_n=\frac{1}{2}+\frac{1}{4}p9_{n-1}$$

$$p10_n=\frac{1}{4}+\frac{1}{2}p10_{n-1}$$

$$p11_n=\frac{1}{2}+\frac{1}{4}p11_{n-1}$$

$$p12_n=\frac{1}{4}+\frac{1}{2}p12_{n-1}$$

$$p13_n=\frac{1}{16}+\frac{1}{4}p13_{n-1}+\frac{1}{8}p10_{n-1}+\frac{1}{8}(1-p12_{n-1})$$

$$p14_n=\frac{1}{16}+\frac{1}{4}p14_{n-1}+\frac{1}{8}p10_{n-1}+\frac{1}{8}p12_{n-1}$$

$$p15_n=\frac{1}{16}+\frac{1}{4}p15_{n-1}+\frac{1}{8}p10_{n-1}+\frac{1}{8}p12_{n-1}$$

$$p16_n=\frac{1}{16}+\frac{1}{4}p16_{n-1}+\frac{1}{8}p10_{n-1}+\frac{1}{8}(1-p12_{n-1})$$

因此,

$$p1_n=\frac{1}{105}2^{-4n-2}(105\times2^{3n}+129\times2^{4n}-75\times2^{n+1}+35\times2^{2n+3}+56)$$

$$p6_n=\frac{1}{7}2^{-3n-3}(21\times2^n+21\times2^{2n}+2^{3n+3}+6)$$

$$p7_n=\frac{1}{7}2^{-3n-3}(-7\times2^n-7\times2^{2n}+2^{3n+3}+6)$$

$$p5_n=p8_n=\frac{1}{3}2^{-3n-3}(-3\times2^n+3\times2^{3n}+2^{2n+2}-4)$$

$$p2_n=p4_n=\frac{1}{21}2^{-3n}(7\times2^n+7\times2^{2n}+2^{3n+3}-1)$$

$$p3_n=\frac{1}{21}2^{-3n}(7\times2^{n+1}+5\times2^{3n+1}-3)$$

$$p9_n=p11_n=\frac{1}{3}2^{-2n}(2^{2n+1}+1)$$

$$p14_n=p15_n=2^{-2n-2}(2^n+1)^2$$

$$p10_n=p12_n=2^{-n-1}(2^n+1)$$

$$p13_n=p16_n=4^{-n-1}(4^n-1)$$

将 $n=64$ 代入 $p1_n$ 可知结论成立.

附录 C.　对 Raviyoyla v1 的攻击成功概率.

以下概率为随机实验 1000 次得到的成功概率.

δ(十六进制)	设计者的实现	附录 A 的实现
0000000000000001	1.000	1.000
0000000000000002	1.000	1.000
0000000000000004	1.000	1.000
0000000000000008	1.000	1.000
0000000000000010	1.000	1.000
0000000000000020	1.000	1.000
0000000000000040	1.000	1.000
0000000000000080	1.000	1.000
0000000000000100	1.000	1.000
0000000000000200	1.000	1.000
0000000000000400	1.000	1.000
0000000000000800	1.000	1.000
0000000000001000	1.000	1.000
0000000000002000	1.000	1.000
0000000000004000	1.000	1.000
0000000000008000	1.000	1.000
0000000000010000	1.000	1.000
0000000000020000	1.000	1.000
0000000000040000	1.000	1.000
0000000000080000	1.000	1.000
0000000000100000	1.000	1.000
0000000000200000	1.000	1.000
0000000000400000	1.000	1.000
0000000000800000	1.000	1.000
0000000001000000	1.000	1.000
0000000002000000	1.000	1.000
0000000004000000	1.000	1.000
0000000008000000	1.000	1.000
0000000010000000	1.000	1.000
0000000020000000	1.000	1.000
0000000040000000	1.000	1.000
0000000080000000	1.000	1.000
0000000100000000	1.000	1.000
0000000200000000	1.000	1.000
0000000400000000	1.000	1.000
0000000800000000	1.000	1.000
0000001000000000	1.000	1.000
0000002000000000	1.000	1.000
0000004000000000	1.000	1.000
0000008000000000	1.000	1.000
0000010000000000	1.000	1.000
0000020000000000	1.000	1.000
0000040000000000	1.000	1.000
0000080000000000	1.000	1.000
0001000000000000	1.000	1.000
0002000000000000	1.000	1.000
0004000000000000	1.000	1.000
0008000000000000	1.000	1.000
0010000000000000	0.999	1.000
0020000000000000	0.999	0.999
0040000000000000	0.999	0.997
0080000000000000	0.993	0.995
0100000000000000	0.986	0.987
0200000000000000	0.980	0.982
0400000000000000	0.958	0.959
0800000000000000	0.898	0.892
1000000000000000	0.810	0.787
2000000000000000	0.648	0.656
4000000000000000	0.436	0.440
8000000000000000	0.126	0.131

附录 D.　改进版 Raviyoyla v1 的伪造示例.

以下均假设

$$adlen=0,\ mlen=512$$

$$key[0,\cdots,31]=v[0,\cdots,31]=0$$

D.1　针对设计者的实现.

(1) 第 19 行改变时

m：全 0.

Δm：除

$$\Delta m\left[\begin{matrix}254,255,\\256+253,256+254,256+255\end{matrix}\right]=0000000000000100$$

其余全为 0.

tag：41 27 6b 9c aa f0 7d 0d 94 cf 73 fe e0 33 99 86 8e d5 59 5c 4d 54 3b 53 7b a4 5e 44 ad 0e c7 de bb 51 07 fa bd b0 87 6d 50 eb 50 0b b3 10 e7 57 44 d6 fe c3 41 5b fa f0 b1 7b 11 9d 35 f6 66 a7

(2) 第 20、22 行改变时

m：全 0.

Δm：除

$$\Delta m[255,256+255]=0000000000000100$$

其余全为 0.

tag：bc 23 d7 fd 8f 15 59 9e bc 5b f0 d6 76 5a b8 2b 18 08 fb d9 44 a6 83 4b 50 a3 4d d6 86 f5 2d 87 1a 24 c2 b8 4e d2 e7 46 4b 78 33 17 38 36 da bd 6d 70 92 72 f7 81 ee 9f 3d 91 98 04 ee 04 36 71

(3) 第 24 行改变时

m：全 0.

Δm：除

$$\Delta m\left[\begin{matrix}254,255,\\256+20,256+254,256+255\end{matrix}\right]=0000000000000100$$

其余全为 0.

tag：b5 90 08 b2 05 57 7e 9c 35 3a 4e 78 bc ff 55 2f cc c2 6a 84 ce 13 8d 69 74 41 7b 31 44 62 22 71 78 6c dc ff 31 83 f2 75 8a 24 88 02 8a 1e 76 26 ae ea b5 a2 57 1e 23 90 19 47 94 2c 48 fc e3 b4

(4) 第 19、20、22 行改变时

m：全 0.

Δm：除

$$\Delta m[255,256+255]=0000000000000100$$

其余全为 0.

tag：ca 6f ea 78 95 c5 ba 02 80 7a 45 eb 14 40 98 12 5f 0a fa be 49 5c 3a cb 8e f5 c5 32 74 b8 3e 87 93 f6 4a b7 47 cb 6e 0a 78 c4 89 98 73 ee f2 88 96 4d 98 23 60 d0 e0 08 8d c5 c8 04 c5 ab 16 30

(5) 第 20、22、24 行改变时

m：除

$$m[0]=9949a973ed6e86f3$$

其余全 0.

Δm：除

$$\Delta m[255,256+255]=0000000000000100$$

其余全为 0.

tag:16 de 71 62 c0 d7 26 93 ae b2 dd cc bf 5b f9 fe d3 48 ac 9e 21 8d 0c 7b 0b 40 1a cc b5 79 65 9b a6 8e 8a 77 7e bf 33 ef 32 d2 49 58 96 cf 21 ce cf ef ae a9 ad 80 d1 f5 d8 e1 13 58 b9 58 4e 8a

(6)第 19、24 行改变时

m:全 0.

Δ*m*:除

$$\Delta m\left[\begin{matrix}254,255,\\256+180,256+254,256+255\end{matrix}\right]=0000000000000100$$

其余全为 0.

tag:6a 07 a9 a5 03 12 c3 81 e7 a0 d7 3a cc ee 75 22 21 b2 36 9e 9d 0a ca c5 13 00 d9 7a 36 e8 82 c4 f6 ff 0d 66 6a 69 08 78 01 c9 af 1b 52 a9 ac 23 bb 04 80 60 2b 59 bc 7c cb 41 c1 1a 1c c2 1c ab

(7)第 19、20、22、24 行改变时

m:全 0.

Δ*m*:除

$$\Delta m[255,256+255]=0000000000000100$$

其余全为 0.

tag:ab 45 3b d8 c2 2f e5 14 d9 aa f3 ac 1a 80 96 fd 48 a3 23 9e b9 15 a0 21 fc 98 80 91 a3 ed 1c 77 72 b4 db 77 6c e1 d1 cc 7c dc fe 25 4a 66 8f 6b 01 80 4e 9b 6b 06 76 82 5d 33 81 7c 3c 99 bc 69

D.2 针对附录 A 的实现.

(1)第 19 行改变时

m:全 0.

Δ*m*:除

$$\Delta m\left[\begin{matrix}254,255,\\256+253,256+254,256+255\end{matrix}\right]=0000000000000100$$

其余全为 0.

tag:48 c4 9c 79 c8 81 85 9f 4a a7 0e d3 00 f2 d9 92 a4 e2 33 05 d8 95 04 59 cd 20 ad 56 d9 7e fe c3 1c 78 f1 47 84 41 68 d0 cd 80 61 3a 42 e2 d2 9a 4d 38 4a e3 2b 99 c3 1d ba 73 b9 36 9d 4a 25 8b

(2)第 20、22 行改变时

m:除

$$m[0]=64ddd1507189f6cb$$

其余全 0.

Δ*m*:除

$$\Delta m[255,256+255]=0000000000000100$$

其余全为 0.

tag:9f 09 be 63 b0 92 d7 af 43 06 49 a5 82 6d d7 1a ec 95 99 a7 73 c6 8a 55 13 e0 20 4d bf 5b 1c 01 8a 3c 35 ce cd 1b 74 ff de 10 1d 2b 01 b6 5c f8 1f d5 91 9b d5 c1 6d 9c eb ff 06 9a 0f e3 69 66

(3)第 24 行改变时

m:除

$$m[0]=ec0dd164d408daea$$

其余全 0.

Δ*m*:除

$$\Delta m\left[\begin{matrix}254,255,\\256+253,256+254,256+255\end{matrix}\right]=0000000000000100$$

其余全为 0.

tag:15 f6 e0 7c a7 99 78 7b a5 0e 07 33 4d e2 ef 54 06 544b 11 5e 26 67 9a 6d ca 6b 0d c9 82 58 2c b3 15 c6 fc 4f d4 26 23 a8 35 9b 3f 0a 23 5f 8d 46 35 18 30 f2 c8 0f ad 1c 95 4c 2d 6b 94 aa de

(4)第 19、20、22 行改变时

m:全 0.

Δ*m*:除

$$\Delta m[255,256+255]=0000000000000100$$

其余全为 0.

tag:73 38 0a 3a 5d 53 51 1d 41 93 41 26 8a f8 b9 0f df 5e c0 5d 3c 3f a1 c3 9a 5b ba 44 86 e0 f4 39 6e 98 55 a8 12 74 e4 e6 c4 bf 73 1b 5e bc dc c0 1e a5 c9 64 d3 cc 9d 33 4b 22 e4 01 c7 53 d7 54

(5)第 20、22、24 行改变时

m:全 0.

Δ*m*:除

$$\Delta m[255,256+255]=0000000000000100$$

其余全为 0.

tag:30 91 d4 ac 56 bd 14 c6 e0 cd 4d d8 d4 e2 98 c8 2e b1 3b 70 66 cf 61 5c 18 b5 fd c8 57 49 f7 da 05 b7 7a 63 b7 f4 4f 19 f6 1c 7d 9c 75 46 07 f1 97 89 e5 37 d1 4c 94 28 28 2c 85 76 77 20 cd cc

(6)第 19、24 行改变时

m:除

$$m[0]=872495e4eadfae9d$$

其余全 0.

Δ*m*:除

$$\Delta m\left[\begin{matrix}254,255,\\256+253,256+254,256+255\end{matrix}\right]=0000000000000100$$

其余全为 0.

tag:f0 0f bc d9 e9 03 29 55 9f 2b f3 8a c3 ae f3 b7 c5 af 0b 4b 40 42 65 3a 44 35 76 6c 96 a5 de 5e 64 59 5b 4b c7 5c 17 5b 20 88 93 de 4e 9b 7f c1 62 b6 1e a7 b0 5c 8e 63 ed eb a2 93 3a 52 65 f2

(7)第 19、20、22、24 行改变时

m:除

$$m[0]=066b7bcfd2c72199$$

其余全 0.

Δ*m*:除

$$\Delta m[255,256+255]=0000000000000100$$

其余全为 0.

tag:69 ae e4 9f bc 60 10 3c ca 65 b6 ec 7c 2a dd a7 b2 b3 75 92 57 db 44 a7 dd f5 19 02 53 49 ca 19 d9 11 d0 a4 ff 8d 6b d9 c8 24 65 9b b3 4f 17 34 7e 12 8c a6 88 8e 3e 24 49 84 50 b5 6e cf c4 24

YAO Yuan, born in 1989, Ph. D. candidate. His research interest is cryptanalysis of symmetric ciphers.

ZHANG Bin, born in 1976, Ph. D., professor, Ph. D. supervisor. His research interest is design and cryptanalysis of symmetric ciphers.

WU Wen-Ling, born in 1966, Ph. D., professor, Ph. D. supervisor. Her research interests include design and cryptanalysis of block ciphers, modes of operation for block ciphers and the theory of provable security.

Background

Privacy and authenticity are primary goals of cryptography. However, privacy does not necessarily imply authenticity which is far more important in certain scenes. Nowadays, authenticity is usually guaranteed by tags generated by block ciphers together with authentication modes. Although, it has been proved to be a secure way in both theory and reality, the performance of this method has greatly limited its use.

With the spread of limited computing devices and the coming period of big data, the demand for efficient authenticated encryption algorithms becomes more and more significant. The CAESAR competition, announced by Bernstein et al. in 2013, intends to identify such primitives offering advantages over AES-GCM and suitable for widespread adoption. 57 algorithms were submitted for the first round in total and then 9 algorithms were withdrawn because of various weaknesses. And it has taken center stage in symmetric cryptanalysis to evaluate the security of the remaining proposals in the last few months.

Raviyoyla v1 is one of the remaining proposals. It was first analyzed by Samuel Neves at March 22nd, 2014 on an online discussion group of CAESAR competition, and then we presented the basic idea of the first attack in this paper at April 23rd, 2014 on the same group. As a response, the designer, Rade Vuckovac, proposed a revision on the *revolve_all* function. In the poster accepted by AsiaCCS 2015 of this paper, we proposed the first forgery attack on the revised Raviyoyla, without detailed explanations.

In this paper, we theoretically analyze the success probability of the first attack and present the corresponding experimental results. Moreover, we provide full details of attacks against the revised Raviyoyla v1 as well as their experimental results. As the best of our knowledge, this is the only cryptanalytic result on the authentication part of Raviyoyla v1 in public.

Based on the results of this paper, it is easily concluded that Raviyoyla v1 is extremely valuable to the differential cryptanalysis and is far from secure. Thus, it is unsuitable for the second round of CAESAR competition without further modifications.

This work is supported by the National Basic Research Program (973 Program) of China under Grant No. 2013CB338002.