**IoT设备中的分组密码研究**

**­**——以Chaskey、Simon和Speck为例

小组成员： 高亨利 1901210713

胡兆杰 1901210403

刘存展 1901210442

1. **背景**

思科IBSG的最新白皮书估计，到2020年，将有500亿个设备连接到互联网，这意味着在不远的将来，每个人都将被数十种传感器设备所包围，从互联网到物联网的这种演进将对我们的日常生活产生巨大影响，并改变我们与周围的物理世界的交互方式。 然而，显而易见的是，连接到Internet的500亿智能设备给其所有者或用户的安全和隐私带来了前所未有的挑战。众所周知，对称密码系统在物联网的安全领域中扮演着重要角色， 但是物联网硬件设备的计算和存储资源十分有限，这就需要对密码体系进行更加细致有效的设计，保证在物联网设备中有限的资源上能够稳定运行。

在本篇报告中，我们选取了3种在IoT设备中性能表现优异并且安全性较高的密码方案，即Chaskey、Simon和Speck，来进行简单的分析和介绍。

1. **密码算法介绍**

**2.1、 Chaskey**

随着IoT行业的蓬勃发展，在微控制器上的安全保密性能逐渐受到学界的重视，而相关的MAC算法也是重点发展的对象。目前使用的基于通用哈希函数的MAC算法通常会用到微控制器上的乘法运算。在某些微控制器上，执行一个整数乘法指令所需要的时钟周期时是与数据相关的，这使得算法容易受到计时攻击的影响。

对于那些基于哈希函数的MAC算法，它们的块大小都非常的大，以常见的MD5，SHA-1和SHA-2来说，消息通常都被分成大小为512比特的块进行处理。对于非常短的消息而言，这会导致过多的开销；对于较长的消息而言，一般的微控制器也无法处理如此大的消息块。因为将数据在有限的寄存器和RAM之间来回移动需要执行大量的装在和存储操作，这大大增加了MAC算法实现的时间、经历以及代码大小。

同样的问题也发生在基于分组密码的MAC算法上，尤其是在常用的AES以及三重DES算法。在典型的微控制器上，分组密码中密钥的存储增加了寄存器压力。这是因为在分组密码的实现上，轮密钥要么提前被计算好并且储存在RAM上，要么在算法过程中同步进行计算。此外，在32位的架构上，AES和三重DES中的S-box需要广泛的使用位屏蔽（bit masking）操作，这也使得算法速度受到负面影响。

在《Chaskey: a Lightweight MAC Algorithm for Microcontrollers》一文中，作者提出了Chasky算法,一种用于32为微控制器的有效的消息认证算法。主要适用于需要128比特安全性但是由于速度、能耗或是代码大小等要求而无法实施标准MAC算法的应用。Chaskey是基于排列的MAC算法，它使用Addition-Rotation-XOR设计方式。基于Even-Mansour分组密码体制的安全性，作者证明了Chaskey加密算法在标准模型中也是安全的。Chaskey在各种32位微控制器上表现良好。并且通过基准测试表明，在ARM Cortex-M3 / M4上，Chaskey加解密速度达到7.0 cycles/byte，而AES-128-CMAC为89.4 cycles/byte。对于ARM Cortex-M0，基准测试结果为Chaskey的速度16.9 cycles/byte和AES-128-CMAC的136.5cycles/byte。

下面我们对Chaskey算法进行具体的介绍：

**Chaskey：**

一种主要用于克服上述的问题的基于排列的MAC算法，。Chaskey算法中使用了长度为128bit的密钥K，并且将消息M分成大小为128bit的块进行置换（Permutation）π处理。该算法的置换过程是基于加法-旋转-异或（Addition-Rotation-XOR）设计方式设计的。

Chaskey算法具有以下的功能：

**专用设计：**

Chaskey是用于32位微控制器体系结构的专用设计。加法和异或操作都是对于32bit的字为单位执行的，并且它们在32位的体系结构下都只需要通过一条指令就可以实现。

**跨平台的多功能性：**

我们考虑到某些微控制器没有支持可变长度的位旋转和位平移。所以我们通过选择一些8的倍数作为旋转常数，使得这些操作可以通过寄存器上的8位或16位交换（swap）操作有效实现。

**高效实现：**

我们将算法运行在ARM Cortex-M4上进行试验，对于长消息（≥128字节），Chaskey算法运行仅需要7.0cycles/byte，而对于短消息（16字节）则仅需要10.6cycles/byte并且只占用了402bytes的ROM。另一方面，Chaskey算法运行在Cortex-M0上的结果也非常好：长消息16.9cycles/byte，短消息21.3cycles/byte并且只消耗了414bytes的ROM。大致来说，我们可以认为运行周期数与能耗之间呈线性关系。因此，我们可以认为Chaskey算法非常节能。

**抵抗计时攻击：**

在所有的微控制器架构上知道，Chaskey的每条指令的执行时间都是固定的。且执行的时钟周期数仅取决于消息的长度。因此，Chaskey具有固有的抵御计时攻击的安全性。

PS：**计时攻击：**

在密码学中,时序攻击是一种侧信道攻击,攻击者试图通过分析加密算法的时间执行来推导出密码。每一个逻辑运算在计算机需要时间来执行,根据输入不同,精确测量执行时间,根据执行时间反推出密码。

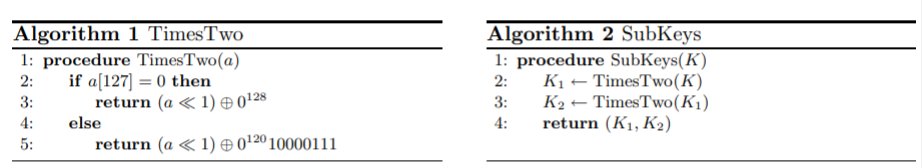
**密钥敏捷性：**

Chaskey没有密钥时间表，因为密钥只是简单地与状态进行异或。在Chaskey中更新密钥只需要生成一个新随机的128位密钥，并且在128位字上仅进行两次移位和两次条件异或就可以生成两个子密钥。

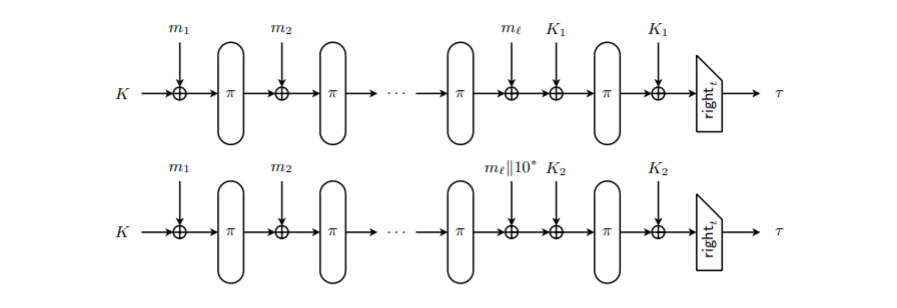
Chaskey的名称源自Chasqui，也称为Chaski。 Chasquis是快速跑者，负责在印加帝国传递信息。它们身材矮小，可以穿山越岭。

**Chaskey算法：**

1. **操作模式：**Chaskey使用n比特长度的密钥K为任意大小的消息M生成一个标签τ，并且标签τ的长度t小于等于n。我们通过算法1和2生成为密钥K生成两个子密钥K1和K2。

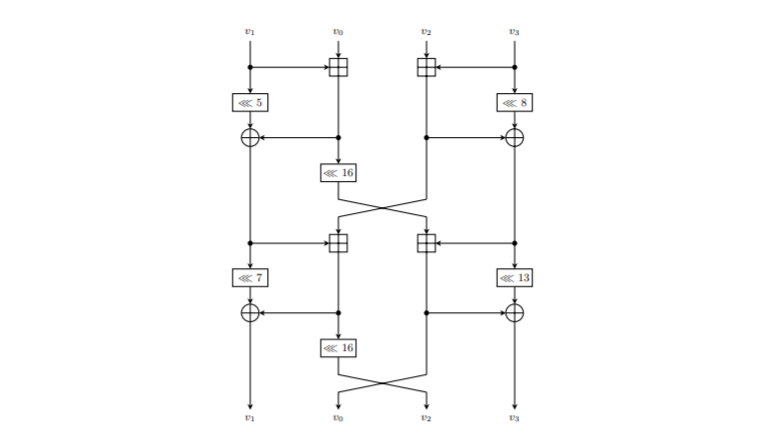


然后将消息M分成大小为n比特的块，需要注意的是消息的最后一个块可能不足长度n比特。然后我们以每n比特为单位对消息进行置换π的迭代，如下图所示：



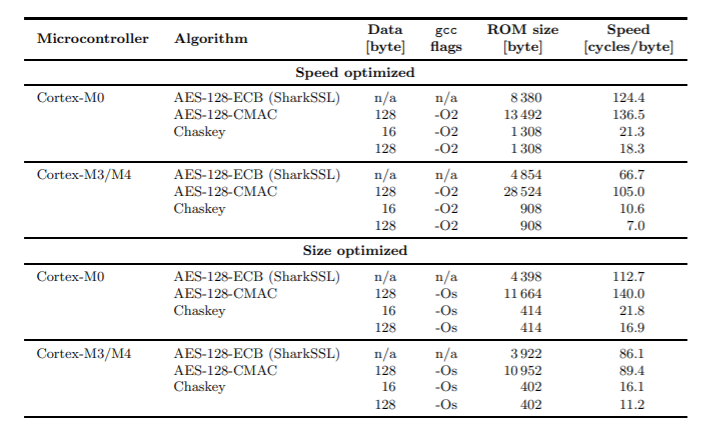
不难发现Chaskey算法在最后一次的置换π的前后都使用了相同的子密钥进行异或操作，这使得子密钥可以一直保留在寄存器当中，减少装载（load）和存储（store）操作的次数，这在操作成本较高的微控制器上显得十分重要。

1. **置换π：**Chaskey算法的置换π是由加法、位旋转和异或操作构成的。如下图所示，每一轮的置换π由八个操作过程组成。



值得注意的是，在一轮的置换π中有一半的位旋转常数都被规定为8的倍数，这是因为有些微控制器并不支持任意位长度的位旋转和平移。由于选择了特殊的常数，这使得Chaskey在8位和16位的微控制器比使用随机位旋转常数的算法具有更高的效率。

**算法应用结果：**下图为我们展示了Chaskey算法在不同的微控制器平台上与其他算法运行的结果对比



我们可以看出在不同的微控制器平台上，Chaskey算法无论在处理长消息或是短消息时都具有更快的运行速度，且速度差距从数倍到数十倍不等，并且只占用了更小的内存空间。综上可以看出Chaskey算法在微控制器上运行的明显优势。

**2.2、 Simon & Speck**

一些轻量级密码算法虽然在某些设备上取得了较好的效果，但美国国家安全局的研究人员发现，这些密码算法存在的一个最大的问题就是缺乏灵活性，即它们都是针对某个特定的平台设计，比如ASIC，不能很好地适用于所有平台。比如，PRESENT是一种效果非常好的轻量级密码算法，但它只在ASIC硬件实现上有良好的表现，在软件平台实现时或者在FPGA上性能表现并没有那么出色，因为它在设计时并没有考虑通用性，只是针对ASIC来进行设计和实现。考虑到这一点，研究人员希望设计一种适用于所有平台的密码算法，来应对未来物联网时代可能出现的各种各样不同架构的计算设备。因此，Simon和Speck算法应运而生。

Simon和Speck密码是一种轻量级的分组密码，由美国国家安全局在2013年发布。它不像之前的轻量级分组密码算法一样有固定的密钥长度和分组长度，而是提供了许多种分组长度以及每种分组可对应的密钥长度来实现灵活性。此外，为了让算法更加轻便，这两种算法没有使用S盒，因而它并不是一种置换-代换网络（SPN），它采用的是Feistel结构，可以在线性扩散和非线性混淆的操作之间很好的平衡。在扩散过程中，我们要让密文中的每一位受明文中尽可能多的位影响，需要采用位置换的手段。为了尽可能在硬件和软件上都达到良好的效果，减少开销，在Simon和Speck算法中使用旋转来完成位置换。

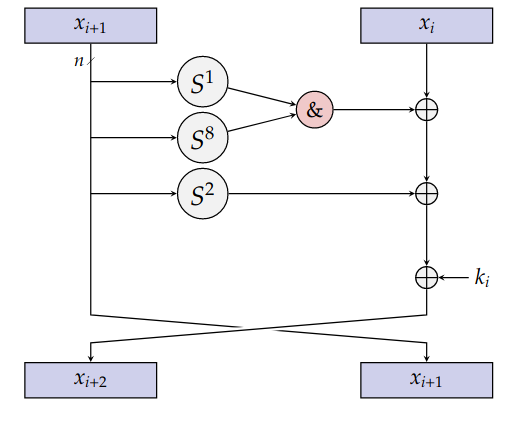
Simon和Speck算法的具体区别在于，Simon针对硬件实现进行优化，Speck针对软件实现进行了优化。为了在硬件上达到更好的效果，Simon使用了移位和与操作作为非线性函数，这两个指令所需的硬件开销非常少，但与此同时在安全性上相对较弱，所以Simon采用了更多的轮数来保证安全。Speck采用了模的加法作为非线性函数，在安全性上比Simon高，也更加适合于软件实现，但在硬件上的速度不如AND操作。

两个算法实现的具体过程如下。Simon算法的轮函数为：

其中

S为左循环移位，上标表示移位的位数。

轮函数如下图所示。

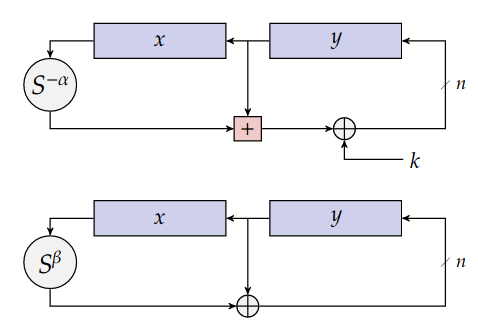


根据密钥大小与块大小之间的倍数，可将所有情况分为三类：（块大小/密钥长度）2倍（96/96，128/128），3倍（48/72, 64/96, 96/144, 128/192），4倍（32/64, 48/96, 64/128, 128/256）。这三类情况下，轮密钥的产生公式分别为：

其中，C、D、E都为常数，I是一个特殊的n×n矩阵，维度等于块大小。

Speck算法的轮函数为：

其中x和y的映射过程可以看做两个Feistel映射的组合，如下图所示。



Speck算法的轮密钥生成函数是基于轮函数得来的，公式如下：

在解密时，Simon的解密过程只需要交换密文，将轮密钥倒序使用，再将得到的明文交换即可，而Speck需要的步骤稍微多一些。且在生成轮密钥时，Speck依据轮函数来生成轮密钥，因此可以重复使用代码，减小代码体积。

1. **密码性能分析**

这里我们参考了文章《Triathlon of lightweight block ciphers for the Internet of things》。文章介绍了一个基准测试框架，用于对用于众多嵌入式平台的轻量级分组密码进行测试。测试框架能够评估执行时间，RAM占用空间以及二进制代码大小，并允许定义一个自定义的“性能指标”，据此可以对所有评估的候选人进行排名。

测试的分组密码包括Chaskey、Simon和Speck等19种分组密码，测试的平台为三个微控制器平台，分别为8位AVR，16位MSP430和32位ARM。基准测试框架接受以“纯” ANSI C或C语言编写的源代码，以及上述三种处理器架构的内联汇编代码。这是一种性能和可移植性之间的折衷方案。因为汇编代码能够保证密码方案运行的的性能，如果用纯C实现，则具有很高的可移植性，但总的来说，效率不如汇编代码实现。但是物联网由数十亿个异构设备组成，这些异构设备具有不兼容的处理器和不同的操作系统。 使用优化的汇编代码来支持大量平台既繁琐又容易出错，对于每种处理器体系结构，都需要单独编写，测试，调试然后维护单独的代码库。 鉴于这些压力，工程师可能更重视C代码的可移植性，而不是汇编代码的性能。并且作者收集了每个密码的两个至多达24个实现，以说明执行时间，RAM占用空间和代码大小之间的不同权衡。

**执行时间**。执行时间通过执行块密码的四个基本操作（即加密，解密，加密密钥调度和解密密钥调度）中的每一个所需的时钟周期数来量化。 如前所述，我们的框架支持指令集模拟器以及从开发板获取实际测量值。

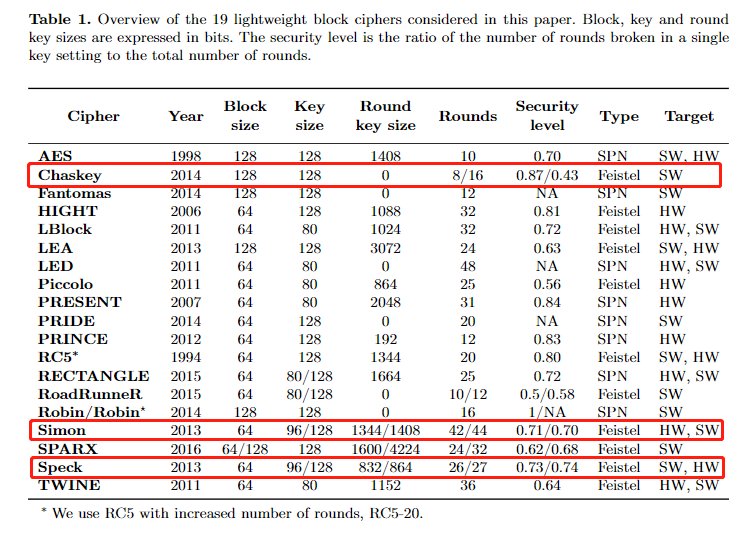
**RAM占用空间**。RAM占用空间由数据段的大小（包含初始化为非零值的所有静态变量）和最大堆栈消耗确定。数据部分占用的RAM量是借助GNU Binutils集合中的sizetool来确定的。使用标准方式评估堆栈的最大使用消耗情况，在操作的开始处（即，在该操作的函数调用之后），可用堆栈空间使用特定模式进行填充。然后，在函数执行结束时，将堆栈区域中的值与模式进行比较，来得出堆栈消耗量。

**代码大小**。代码大小以字节为单位，并量化在指定操作或使用情景下，在目标设备的非易失性存储器（例如闪存）中占用的存储量。通过对编译器生成的相关目标文件使用相关工具来对它进行评估。 该工具是GNU Binutils的一部分，可以列出目标文件或可执行文件不同部分的大小。为了获得整个代码的大小，框架只需添加相关对象文件的text段和data段部分的大小。 Text段部分包含编译器从源代码生成的可执行机器指令。 Data段部分包含所有用非零值初始化的静态变量。该节的内容在程序执行开始时从闪存加载到RAM中。bss段为空，因为测试中没有任何标准测试操作会涉及任何未初始化的静态变量。

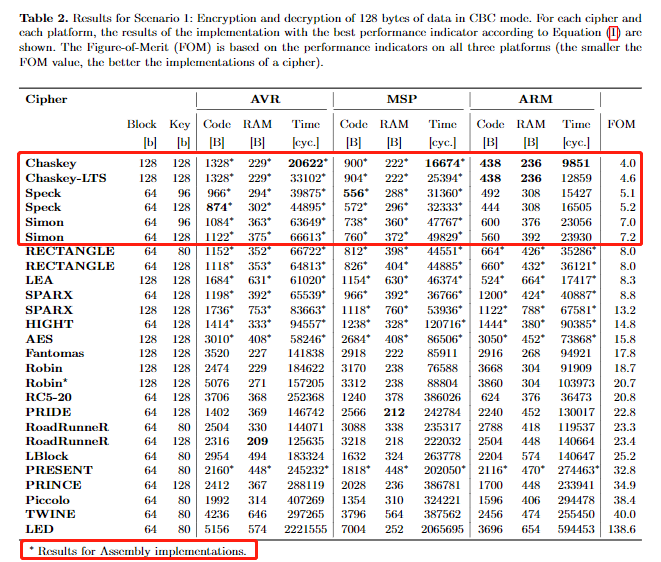
除了评估分组密码的四个基本操作（即加密，解密，加密密钥调度和解密密钥调度）外,测试还设计了一些与IoT实际相关的通用安全服务，并利用基本密码操作的真实使用场景。这样，就有可能获得在现实世界中有意义的基于现实基准的测试结果。

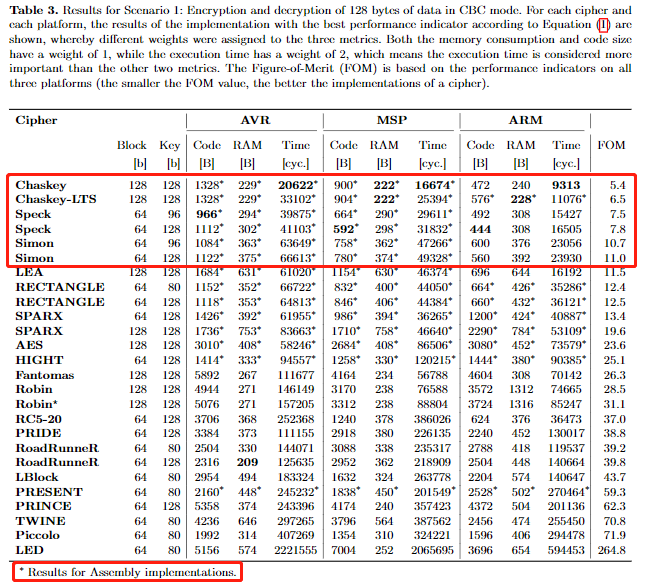
**场景1**：通信协议此方案涵盖了两个IoT设备之间安全通信的需求。场景2：质询响应身份验证。此场景的灵感来自于简单的身份验证协议，其中IoT设备通过使用分组密码对质询进行加密来证明它拥有私钥。

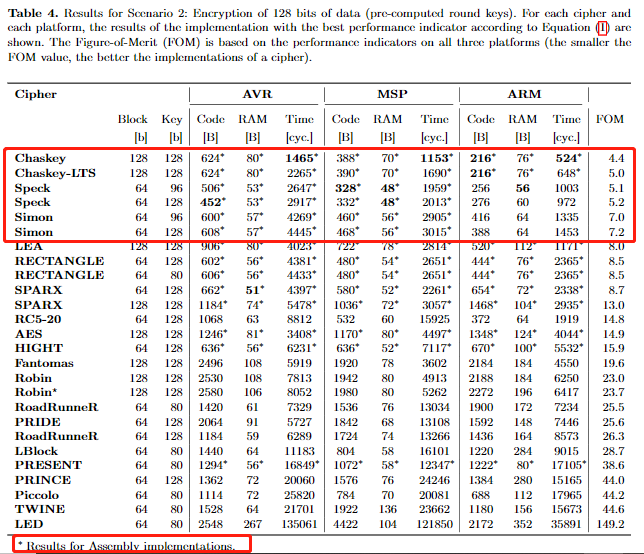
表1总结了参与测试的密码方案的一些主要特征。

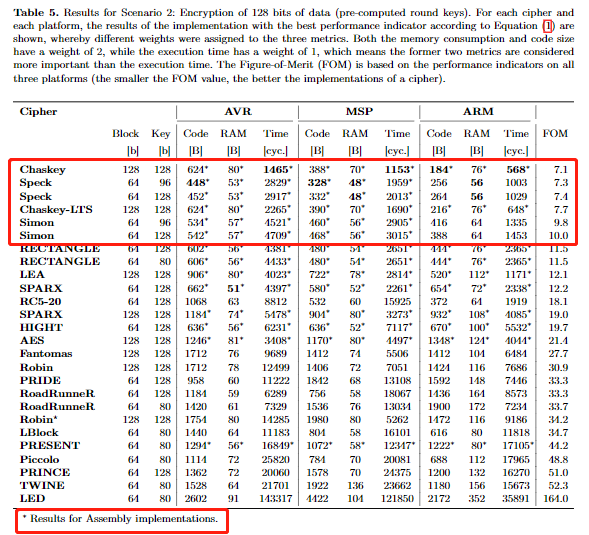


下面是在不同场景不同平台上，根据执行时间、RAM占用空间和代码大小之间的不同权衡比重，测试的在各种密码方案的不同实现的最终结果。









结果表明，最新的ARX和类似ARX的设计不仅非常快，而且在RAM空间占用量和代码大小方面也非常小。以执行时间、RAM占用空间和代码大小为基础的铁人三项比赛的总冠军是Chaskey，紧随其后的是Speck。 两者在两种使用情景下以及在所有三个平台上始终表现良好，这使它们成为保护物联网安全的轻量级密码的强大候选者。当执行时间，RAM占用空间和代码大小权重一样时，Simon等密码方案的FOM值也低于10.0，也获得了很好的结果。

我们用来对轻量级分组密码进行排名的FOM分数仅基于效率指标，并未考虑任何（密码分析）安全性方面。在这种情况下，应该指出，Chaskey或在超过32位上运行的Speck版本都不提供针对线性或差分密码分析的可证明的安全性。 与安全性也相关的是，我们发现密钥大小会对轻量级密码的整体效率产生巨大影响。特别是Simon和Speck的结果表明，当密钥大小从128位减少到96位时，FOM指标增加了一些百分比，但是很难证明会产生相应的安全损失。

测试所提供的结果可以给IoT安全工程师 在选择轻量级密码以满足应用程序的需求和目标设备的约束时带来帮助。 此外，结果也与密码的设计者有关，因为当密码在软件中实现并在微控制器上执行时，可以推断出基本设计决策与其性能和占用空间之间的一些联系。我们尤其建议密码设计者将重点放在简单的舍入函数上，因为这些函数可以使用尽可能少的操作，并在多次迭代后可以达到良好的安全性。为了在不同字长的架构上获得最佳性能，密码的字长应与所用架构上可用的最大寄存器的大小相匹配。这样，寄存器的内容可有效地用于具有最大字长的平台上，而不会影响具有较小寄存器的体系结构上的性能。只要密码状态可以保存在可用寄存器中，上述有效操作就不需要访问内存。 最后，应避免使用任何大小的查找表，因为它们会增加代码大小和RAM占用空间，并且还需要额外的加载指令。

1. **总结**

报告通过对三种适用于IoT设备的分组密码Chaskey、Simon和Speck进行分析总结，对它们的设计构造方案和安全性能有了进一步的认识。并且通过对它们的性能测试部分的分析，分别得出了在资源受限的IoT设备中，三种密码方案的运行性能情况和对比其他密码方案的优势。也给我们在以后的学习和研究带来了一些启发，在设计密码方案的时候，要尽可能考虑在实际运行环境下兼顾安全性和运行性能。

1. **参考文献**

1. Beaulieu R, Treatman-Clark S, Shors D, et al. The SIMON and SPECK lightweight block ciphers[C]//2015 52nd ACM/EDAC/IEEE Design Automation Conference (DAC). IEEE, 2015: 1-6.

2. Beaulieu R, Shors D, Smith J, et al. SIMON and SPECK: Block Ciphers for the Internet of Things[J]. IACR Cryptology ePrint Archive, 2015, 2015: 585.

3. Daniel D , Le C Y , Dmitry K , et al. Triathlon of lightweight block ciphers for the Internet of things[J]. Journal of Cryptographic Engineering, 2018.

4. D. Evans. The Internet of Things: How the Next Evolution of the Internet is Changing Everything. Cisco IBSG white paper, available for download at http://www.cisco.com/web/about/ac79/docs/innov/IoT\_IBSG\_0411FINAL.pdf, Apr. 2011.

5.  N. Mouha, B. Mennink, A. Van Herrewege, D. Watanabe, B. Preneel, I. Verbauwhede, "Chaskey: an efficient mac algorithm for 32-bit microcontrollers", International Workshop on Selected Areas in Cryptography, pp. 306-323, 2014

6. Dworkin, M.: Recommendation for Block Cipher Modes of Operation: The CMAC Mode for Authentication. NIST special publication 800-38b, National Institute of Standards and Technology (NIST) (May 2005), <http://csrc.nist.gov/publications/nistpubs/800-38B/SP_800-38B.pdf>

7. Bellare, M., Canetti, R., Krawczyk, H.: Keying Hash Functions for Message Authentication. In: Koblitz [47], pp. 1–15