基于进程代数的 SP 网络结构密码形式化设计研究

张 磊, 许弘可, 肖超恩, 王建新, 郑玉峭

北京电子科技学院 北京 中国 100070

摘要 针对以往代换-置换(SP)网络结构密码算法的设计与实现主要依赖设计者的经验手工实现, 导致设计过程中不可避免的主观性和不一致性的问题, 本文提出了一种基于进程代数从组件层次描述 SP 网络结构密码的形式化设计方法。首先提出了面向密码算法开发的 MCL 元密码语言设计原则, 为后续形式化描述奠定基础。其次, 通过对 SP 网络结构密码的特点进行分析, 基于进程代数提出了四大组件, 对四大组件在设计 SP 网络结构密码算法中的使用进行了说明, 用于对 SP 结构密码进行形式化设计描述。最后, 通过该形式化设计方法, 建立了 TANGRAM 密码算法和 SM4 密码算法的形式化模型, 阐述了形式化模型设计在处理复杂的 SP 网络结构密码算法的关键难点和技术挑战, 基于形式化模型搭建了密码算法的 MCL 元密码模型, 同时在 MetaCrypto 平台下进行了正确性验证。验证结果表明, 基于本方法能实现轻量级 TANGRAM 加密算法的设计与实现; 同时, 针对 SM4 密码算法中的 SP 网络结构也能正确地建模与实现; 并与传统设计方法比较, 形式化设计方法在系统性、准确性、可维护性和适用性方面均优于传统设计方法。这一设计方法不仅为 SP 网络结构密码的设计过程提供了坚实的理论基础, 确保了设计的系统性和准确性, 还为密码算法的形式化设计提供了一种创新性的途径。

关键词 MCL 元密码语言; 形式化设计; SP 网络结构密码; 进程代数; TANGRAM

中图法分类号 TP309 **DOI 号** 10.19363/J.cnki.cn10-1380/tn.2026.03.08

Research on Formal Design of SP Network Cipher based on Process Algebra

ZHANG Lei, XU Hongke, XIAO Chaoen, WANG Jianxin, ZHENG Yuzheng

Beijing Electronic Science and Technology Institute, Beijing 100070, China

Abstract In view of the problem that subjectivity and inconsistency in the design and implementation of traditional Substitution-Permutation (SP) network structure cryptographic algorithms, which mainly rely on the designer's experience and manual implementation, this paper proposes a formal design method based on process algebra to describe the structure of SP network cipher algorithms from the component level. Firstly, the design principles of MCL (MetaCrypto Language) oriented to cryptographic algorithm development are proposed, which lays a foundation for the subsequent formal description. Secondly, through the analysis of the characteristics of SP network structure cryptography algorithm, four components based on process algebra are proposed based on process algebra, and the use of the four components in designing SP network structure ciphers is explained, which is used to formally design and describe SP structure ciphers. Finally, the formal models of TANGRAM cryptography algorithm and SM4 cryptography algorithm are established by using the formal design method, and the key difficulties and technical challenges of formal model design in dealing with complex SP network structure cryptographic algorithms are described. Based on the formal model, MCL model of cipher algorithm is built, which provides a solid theoretical support for the design of block cipher algorithm based on MCL. The correctness is verified by MetaCrypto platform. The verification results show that the design and implementation of lightweight TANGRAM encryption algorithm can be realized based on this method. At the same time, the SP network structure in SM4 cryptographic algorithm can be modeled and implemented correctly. Compared with traditional design methods, formal design methods are superior to traditional design methods in terms of systematicness, accuracy, maintainability and applicability. The proposed design method not only provides a solid theoretical foundation for the design process of SP network structure cryptography, ensuring systematic and accurate design, but also offers an innovative path for the formal design of cryptographic algorithms.

Key words MetaCrypto language; formal design; SP Network Cipher; process algebra; TANGRAM

通讯作者: 许弘可, 硕士, Email: 201720060827@ecut.edu.cn。

本课题得到中央高校基本科研业务费(No. 3282024009, No. 20230051Z0114, No. 20230050Z0114); 中央高校基本科研业务费(应用研究类)(No. 2023A03); 教育部新工科研究与实践项目(No. E-AQGABQ20202704); 北京高等教育"本科教学改革创新项目"(No. 202110018002); 北京电子科技学院一流学科建设项目(No. 20210064Z0401, No. 20210056Z0402)资助。

收稿日期: 2024-03-31; 修改日期: 2024-09-24; 定稿日期: 2026-01-21

1 引言

随着计算机网络进入快速发展的时代, 尤其是电子商务平台和电子政务系统在更大范围内应用, 更多的个人信息、商务信息、政务信息需要严格保密。保证信息传输过程中的安全性应是信息安全的核心与重点[1]。密码学是网络空间安全的基础理论, 能够为各种信息系统提供安全服务[2]。密码学的基本思想是按照密码算法的规律将信息进行变换, 以保护信息的安全性, 确保信息在传输和存储过程中不被未经授权的人或者系统访问和修改。密码算法是信息加密的核心, 在互联网时代, 保护用户的隐私和安全成为一项重要的任务, 密码算法设计与实现的重要性越来越突出。

然而, 以往密码算法的设计和验证主要依赖于设计者的经验, 采用人工编制模式[3]。设计者经过长期的经验积累, 掌握了设计的基本原则, 在此基础上进行设计。验证时, 利用人工检验文档和设计方案的形式, 并进行物理测试。这种设计和验证完全依赖于个人经验, 错误难以避免, 且验证过程周期长, 代价大, 很难适应用户的需求, 无论在时效性上还是完备性上都不能满足实际应用的需求。此外, 设计过程中的主观性因素也可能导致结果的不一致性, 这在追求算法设计的普适性和可靠性的现代密码学领域中, 显得尤为突出, 例如曲线加密算法中的参数选择问题, 体现了主观性的选择可能导致的安全性问题[4]。如果引入形式化的方法对密码算法进行设计和工程实现[5], 一方面设计人员能够高效地自动检测出密码算法设计中存在的缺陷, 从而改进设计; 另一方面, 密码模块的工程实现人员可以花费较少的人力物力验证模块是否满足设计要求。

苗洁君等人[6]提出将形式化方法用于对密码模块进行辅助设计及验证的研究方向, 但未就此方法进行实践; 彭昌勇等人[7]提出了形式化函数的分析方法, 但仅给出分析方法且为针对分组密码算法进行攻击的形式化方法, 并不适用分组密码算法设计; 张诗怡等人[8]提出形式化 MDS 矩阵的概念, 采用形式化设计方法对分组密码线性扩散层进行设计, 但只是将形式化方法用于密码算法设计的一部分, 未推出针对密码算法整体的设计方法; 毕兴[9]提出了针对安全协议的形式化方法, 对安全协议的分布式算法进行安全分析, 但本文主要针对安全协议, 未对安全协议中的密码算法进行形式化分析; 宋富[10]针对当前面向密码实现的形式化验证技术的不足进行阐述, 并指出未来构建高性能、高可信密码库的目标, 但仍未从密码算法设计的角度看待形式化方法的未来方向; Yasinsac[11]早在 1996 年提出采用形式化模型的方法对加密协约进行形式化定义, 以此来评估加密协议的安全性; Sebastien Briais[12]提出了基于 Coq 语言[13]的形式化方法, 对 SPI 演算进行分析, 建立了一个自动化验证工具; Zhou 等人[14]基于进化算法和 SVO 逻辑的自动化生成身份验证安全协议的形式化设计方法, 该方法可以自动生成大规模的身份验证协议, 但以上方法都未从密码算法设计的角度解决密码算法设计的问题。综上所述, 表明了传统密码算法设计方法依赖于设计者经验的问题, 并且指出了形式化方法在提高设计准确性和系统性方面的潜在价值。

代换-置换(Substitution-Permutation, SP)网络结构密码形式化设计方法的基本思路是将 SP 网络结构密码的工作流程、机理设计引入进程代数进行形式化设计, 然后使用 MCL 元密码语言根据形式化结果进行工程实现, 从而发现密码模块设计是否合理和安全性漏洞。与传统的设计方法相比, 形式化设计方法提高了建模的准确性、设计的正确性、完备性。

本文利用面向密码算法开发的领域专用语言(Domain-Specific Language, DSL)MCL 元密码语言(MetaCrypto Language), 基于进程代数理论, 提出了从组件层次描述 SP 网络结构密码结构的形式化设计方法。本文提出了 MCL 元密码语言的设计原则, 分析了 SP 网络结构密码的特点, 基于进程代数提出了四大组件, 对 SP 网络结构密码算法的各个结构进行了形式化描述, 并通过该形式化设计方法, 建立 TANGRAM 密码算法和 SM4 密码算法的形式化模型, 基于形式化模型搭建了 TANGRAM 密码算法和 SM4 密码算法的 MCL 元密码模型, 并进行正确性测试。

2 MCL 元密码语言设计原则

MCL 元密码语言是本文提出的一种专为密码算法设计和工程实现而开发的新兴领域专用语言。该语言旨在精确高效地描述密码算法, 并支持其在各种软硬件平台下的集成。MCL 元密码语言的核心设计理念是提供一种标准化工具, 在软件开发与硬件实现之间架起一座桥梁, 从而允许密码算法在多个平台上得到有效运用。

在密码算法中, 使用频次较高的运算操作分别为 S 盒运算操作、算术运算逻辑操作、移位运算操作[15], 这些操作通常处理的数据块为 64~128 bit, 而密钥长度则多为 64~256 bit。此外, 大部分 SP 网络结构密码的计算分支很少, 很少经过条件判断和跳转,

计算流程倾向于线性执行, 这意味着其操作序列不受待处理数据值的影响。

基于上述 SP 网络结构密码特点, MCL 元密码语言的设计集成了软件语言的灵活性与硬件描述语言的规范性, 其语法结构采用简单明了的形式: <操作符> <操作数 1><操作数 2><操作数 3>, 其中操作符明确指定了当前执行的运算类型, 且每个操作符后跟随三个操作数, 分别表示两个输入和一个输出。该设计不仅反映了密码算法中操作的典型模式, 还允许对这些操作进行精确的表达与控制。特别地, 操作数在 MCL 元密码语言中被统一定义为 32 bit, 以适应广泛的密码学应用场景。

MCL 元密码语言提供了一组算术运算逻辑操作指令, 这些指令专门为设计和实现复杂的密码学算法而构建。每一种逻辑操作都遵循类似的模式, 其中包括基本逻辑运算操作(如 AND、OR、XOR、NOT), 以及组合逻辑运算操作(如 AND_XOR、OR_AND)。组合逻辑运算操作能够将基本逻辑运算操作的输出作为输入, 如在基础逻辑运算操作中:

AND *A*, *B*, *C*

表示操作数 *A* 和 *B* 执行逻辑与运算, 并将结果赋值给操作数 *C*, 即 *C*=*A*&*B*。

本文需要用的 MCL 元密码语言基本逻辑运算操作及相应功能如表 1 所示。

表 1 MCL 元密码语言基本逻辑运算操作及功能

Table 1 MetaCrypto Language Basic logic operations and functions

MCL 元密码语言	功能
XOR *A*,*B*,*C*	*C*=*A*^*B*
AND *A*,*B*,*C*	*C*=*A*&*B*
OR *A*,*B*,*C*	*C*=*A*|*B*
NOT *A*,NULL,*C*	*C*=~*A*
ROL32 *A*,*B*,*C*	*C*=*A*<<<*B*

其中, 操作符为 XOR,AND,OR,NOT 时, 上文已介绍其设计原则。当操作符为 ROL32 时, 表示执行向左循环移位运算操作:

ROL32 *A*,*B*,*C*

表示操作数 *A* 循环左移 *B* 比特, 并将结果赋值给操作数 *C*, 即 *C*=*A*<<<*B*。由于操作数的位宽统一定义为 32 bit, 如果要执行循环右移运算也能采用循环左移执行, 如此时需要执行循环右移 *m* bit, 相当于循环左移(32-*m*) bit。

在组合逻辑运算操作中, MCL 元密码语言采用隐式临时变量的设计原则:

AND *A*, *B*, NULL

AND_XOR *C*, *D*, *E*

第一句执行操作数 *A* 和 *B* 逻辑与运算, 由于操作数 3 为 NULL, 计算结果被赋予一个隐式的临时变量(假定为 temp), 即 temp=*A*^*B*。其结果再与隐式临时变量 temp 进行异或运算, 最终将结果赋给操作数 *E*。值得注意的是, 当操作数 3 非 NULL 时, 运算结果将赋给指定的操作数, 同时也会更新该隐式临时变量, 实现 *E*=(*A*&*B*)^(*C*&*D*)。类似地, MCL 中的其他基本运算也遵循此逻辑模式, 例如 XOR_XOR, XOR_AND, XOR_OR, AND_AND, AND_OR, AND_XOR, OR_AND, OR_OR, OR_XOR。

每一句 MCL 元密码语言对应一个 MCL 元密码模型, MCL 元密码语言的设计理念是通过可视化的方式进行密码算法设计, 采用直观交互的编程方式, 大大降低对专用编程语言知识的要求, 使得设计者可以更加专注于密码算法本身。通过使用 MCL 元密码模型搭建密码算法模型, 密码算法的设计和描述可以得到高效的实现, 再通过代码生成器将 MCL 元密码模型映射为 MCL 元密码语言, 如异或元模型 XOR 模块的 MCL 元模型可视化样式如图 1 所示。

图 1 XOR 的 MCL 元密码模型

Figure 1 MCL model of XOR

用户可以根据密码算法需求, 基于 MCL元密码模型实现密码算法可视化模块, 并可以通过拖拽连接这些功能模块, 快速地构建出复杂的密码算法模型。

分组密码算法通常将明文分为若干个固定长度的数据块, 每个数据块单独进行加密处理, 最终得到密文, 常见的分组密码算法有 DES[16]、3DES[17]、AES[18]等, 典型结构如图 2 所示。分组密码算法主要由若干轮相同的迭代组成, 另外还有前置变换、后置变换、密钥扩展等几部分。前置变换和后置变换只需要执行一次, 并不是出现在所有分组密码算法中, 如 AES 的前置变换就是预备轮变换, 也就是初始密钥加, 而 AES 并没有后置变换, 轮变换要在迭代控制下执行多次, 往往最后一轮结构稍有不同, 最后一轮变换列混合。轮变换由于执行次数多, 结构相对复杂, 是分组密码中最重要的结构。

不同分组密码算法具有不同的轮变换结构, 其中 Feistel 网络结构和 SP 网络结构是分组密码算法中最主要的两类结构。Feistel 网络结构的轮变换如图 3 所示, 轮变换的输入和输出都是长为 2ω 比特的数据。轮变换的输入数据分为左右两部分, 每部分长度

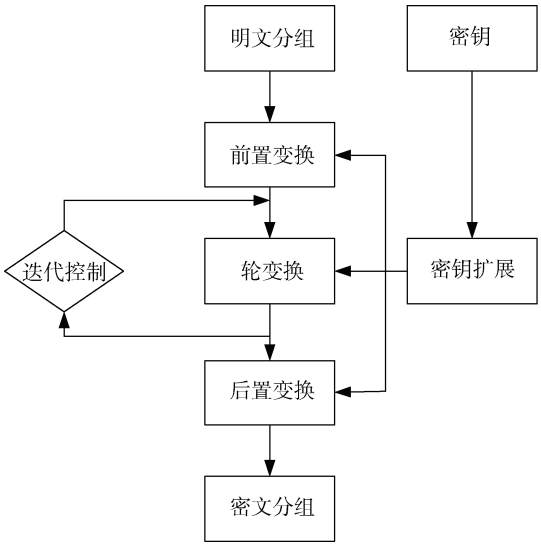

图 2　分组密码经典结构

Figure 2　Classical structure of block cipher

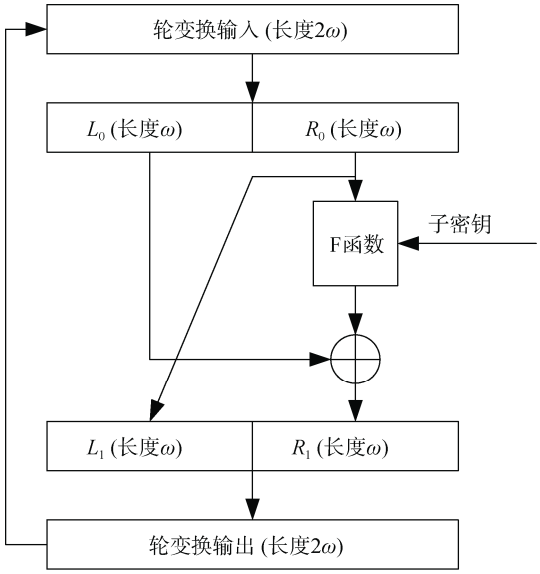

图 3　Feistel 网络结构

Figure 3　Feistel network structure

均为 ω 比特。数据的右半部分通过轮函数 F 处理, 处理结果与左半部分进行异或, 得到轮变换的右半部分输出。输入数据的右半部分同时直接输出到轮变换的左半部分。最后一轮迭代结束后, 需交换左右两部分, 目的就是使加密算法和解密算法具有相同的结构。加密和解密唯一不同的是子密钥输入的顺序刚好相反。Feistel 密码的解密过程与加密相同, 这是 Feistel 密码最大的特点之一。DES 算法和商密算法 SM4 都是典型的 Feistel 网络结构。

SP 网络结构基于密码学中替换和置换两个基本操作。替换经常称为 SBox, 用来对消息进行混淆, 一般是在子密钥的控制下进行。置换经常称为置换 (Permutation Box, PBox), 用来对消息进行扩散, 一般是可逆的线性变换。

SP 网络典型的轮变换如图 4 所示, 轮变换的输入首先送给一个由子密钥控制的可逆替换, 然后其结果送给一个置换或者可逆的线性变换。由于 SP 网络密码的轮变换结构比 Feistel 网络机构更加统一, 且轮变换输入分组各个子块处理过程都是一样的, 因此便于算法的并行实现。

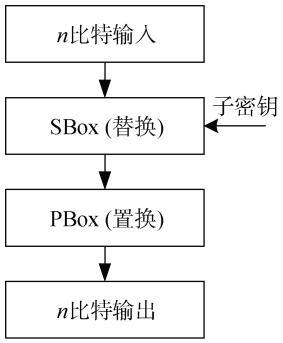

图 4　SP 网络典型的轮变换

Figure 4　Typical wheel transformation of SP network

3　SP 网络结构密码形式化描述

分组密码种类繁多, 结构复杂, 随着运算数量的增加, 基于进程代数[19]的形式化描述呈现指数级增长, 不利于研究与分析。本文从组件层次入手, 研究分组密码中的 SP 网络结构密码, 根据前文对 SP 网络结构密码结构的分析, 将 SP 网络结构密码分为四大组件, 分别为 KeyExpansion 组件, InitialTransformation 组件, Round 组件, FinalTransformation 组件。

3.1　KeyExpansion 组件

KeyExpansion 组件负责对初始密钥进行扩展, 从而生成每轮加/解密所需的轮密钥。该组件主要通过一系列算法操作, 生成一组子密钥, 其中具体操作包括初始密钥的输入、子密钥的生成和子密钥的输出。

KeyExpansion 组件通过四种状态实现密钥扩展: 初始状态(Null)、轮变换状态(Running)、暂停状态(Paused)、停止状态(Stopped), 由密钥初始化(*start*)、暂停轮变换(*pause*)、继续轮变换(*continue*)、停止轮变换(*stop*)、第 n 轮轮变换(*ExpandKey_n*)5 个动作实现相应状态的转换, 描述如下:

密钥扩展初始化: $\text{KeyExpansion}^{\text{null}} \xrightarrow{start} \text{KeyExpansion}^{\text{running}}$;

密钥扩展停止: $\text{KeyExpansion}^{\text{running}} \xrightarrow{stop} \text{KeyExpansion}^{\text{stopped}}$;

密钥扩展暂停: $\text{KeyExpansion}^{\text{running}} \xrightarrow{pause} \text{KeyExpansion}^{\text{paused}}$;

密钥扩展暂停: $\mathrm{KeyExpansion}^{\mathrm{paused}} \xrightarrow{continue} \mathrm{KeyExpansion}^{\mathrm{running}}$;

密钥作为密钥扩展部分传递重要信息载体, 分为初始密钥和轮密钥。当 KeyExpansion 组件进行轮变换时, 密钥要进行初始化以及轮密钥变换, 即作为密钥扩展的输入和输出, 包括初始密钥的输入、轮密钥的输入、轮密钥的输出, 定义如下:

初始密钥的输入: $\overline{startKey}\langle x\rangle$ .KeyExpansion;

第 n 轮轮(子)密钥的输入: $\overline{RoundKey}\langle kn\rangle$ .KeyExpansion;

第 n 轮轮(子)密钥的输出: $RoundKey(kn)$ .KeyExpansion;

其中, kn 表示第 n 轮轮密钥。

当进行密钥扩展初始化时, 描述如下:

$\overline{startKey}\langle x\rangle$.$\mathrm{KeyExpansion}^{\mathrm{null}} \xrightarrow{start} \mathrm{KeyExpansion}^{\mathrm{running}}$;

密钥扩展的核心难点在于密钥扩展过程与后续的轮变换紧密相连, 在设计时还需考虑如何优化其计算复杂度, 以保证整个加/解密流程的效率和安全性, 因此采用 *ExpandKey_n* 动作对加/解密流程进行分解, 使 KeyExpansion 组件得出的每一轮密钥能平滑衔接后续的轮变换。

当进行轮密钥变换时, 描述如下:

$\overline{RoundKey}\langle k0\rangle$.$\mathrm{KeyExpansion}^{\mathrm{running}} \xrightarrow{ExpandKey\_1} RoundKey(k1)$.$\mathrm{KeyExpansion}^{\mathrm{running}}$;

该语句表示轮密钥 $k0$ 经过轮密钥变换 *ExpandKey*_1 得到轮密钥 $k1$

当进行 32 轮轮密钥变换时, 描述如下:

$\overline{RoundKey}\langle k0\rangle$.$\mathrm{KeyExpansion}^{\mathrm{running}} \xrightarrow{ExpandKey\_1} \overline{RoundKey}\langle k1\rangle$.$\mathrm{KeyExpansion}^{\mathrm{running}} \xrightarrow{ExpandKey\_2} \overline{RoundKey}\langle k2\rangle$.$\mathrm{KeyExpansion}^{\mathrm{running}} \xrightarrow{\cdots} \overline{RoundKey}\langle k31\rangle$.$\mathrm{KeyExpansion}^{\mathrm{running}} \xrightarrow{ExpandKey\_32} \overline{RoundKey}\langle k32\rangle$.$\mathrm{KeyExpansion}^{\mathrm{running}} \xrightarrow{stop} RoundKey(k32)$.$\mathrm{KeyExpansion}^{\mathrm{stopped}}$;

可将 32 轮轮密钥变换描述简化为:

$\overline{RoundKey}\langle k0\rangle$.$\mathrm{KeyExpansion}^{\mathrm{running}} \xrightarrow{\langle ExpandKey\_1, ExpandKey\_2, ExpandKey\_3, \cdots, ExpandKey\_32\rangle} \overline{RoundKey}\langle k32\rangle$.$\mathrm{KeyExpansion}^{\mathrm{running}} \xrightarrow{stop} RoundKey(k32)$. $\mathrm{KeyExpansion}^{\mathrm{stopped}}$;

若进行若干轮轮密钥变换, 对轮数进行计数, 未达到指定轮数则继续轮密钥变换, 以 *roundCount* 表示当前轮数, *roundOut* 表示指定轮数, 描述如下:

$\overline{RoundKey}\langle k0\rangle$.$\mathrm{KeyExpansion}^{\mathrm{running}} \xrightarrow{ExpandKey}$ Component|([$roundCount \le roundOut$] $\mathrm{KeyExpansion}^{\mathrm{running}}$+[$roundCount > roundOut$] $\mathrm{KeyExpansion}^{\mathrm{stopped}}$)

在 SP 网络结构密码算法的硬件设计中, 存在流水线设计方式, 需要每输出一轮轮(子)密钥就将该密钥加入加解密变换中, 因此, 需要暂停密钥扩展进程输出当前轮(子)密钥, 描述如下:

$\overline{RoundKey}\langle kn\rangle$.$\mathrm{KeyExpansion}^{\mathrm{running}} \xrightarrow{pause} RoundKey(kn)$.$\mathrm{KeyExpansion}^{\mathrm{paused}}$;

暂停完加入加解密变换后, 需要继续轮密钥变换, 描述如下:

$\overline{RoundKey}\langle kn\rangle$.$\mathrm{KeyExpansion}^{\mathrm{paused}} \xrightarrow{countinue} RoundKey(kn)$.$\mathrm{KeyExpansion}^{\mathrm{running}}$;

当一个密钥算法中存在两种密钥扩展方式时, 密钥输入后同时进行两种密钥扩展方式 KE1, KE2, 描述如下:

$\overline{RoundKey}\langle kn\rangle$. $RoundKey(kn+1)$ .KE1|| $\overline{RoundKey}\langle kn\rangle$. $RoundKey(kn+1)$ .KE2

如果密钥输入后, 先进行第一种密钥扩展方式 KE1, 再进行第二种密钥扩展方式 KE2, 描述如下:

$\overline{RoundKey}\langle kn\rangle$.$\mathrm{KE1}^{\mathrm{running}}$|| $\overline{startKey}\langle x\rangle$.$\mathrm{KE2}^{\mathrm{null}} \xrightarrow{ExpandKey\_1} \mathrm{KE1}^{\mathrm{stopped}}$|| $\{kn/x\}$.$\mathrm{KE2}^{\mathrm{running}}$;

可将上述描述简化, 进程代数中动作可简化为不在箭头上表示, 直接以行为迹的形式在组件表示当前组件执行的动作, *pause*、*start*、*stop* 可简化为在密钥扩展方式 KE1、KE2 前面表达, 描述如下:

((*pause*.$\mathrm{KE1}^{\mathrm{paused}}$).*start*.$\mathrm{KE2}^{\mathrm{running}}$).*stop*.$\mathrm{KE1}^{\mathrm{stopped}}$;

pause.$\mathrm{KE1}^{\mathrm{paused}}$ 表示 KE1 执行暂停(*pause*)动作进入暂停状态, 之后执行 KE2 的密钥初始化(*start*)动作, 开启 KE2 的 running 状态, 之后执行 stop 动作停止 KE1 状态, 此时 KE1 进入 stopped 状态。

3.2 Round 组件

Round 组件是整个加密算法的核心部分, 负责执行加/解密的轮变换。每一轮加/解密变换操作包括多个步骤, 通常涉及 S 盒替换和 P 盒置换, 以实现数据的混淆和扩散。轮操作通过一系列的密钥和数据运算, 将明文逐步转化为密文。

Round 组件同样具有四种状态: 初始状态(Null)、轮变换状态(Running)、暂停状态(Paused)、停止状态(Stopped), 由轮变换初始化(*startRound*)、暂停轮变换(*pause*)、继续轮变换(*continue*)、停止轮变换(*stop*)、第 n 轮轮变换(*Round_n*)5 个动作实现相应状态的转换, 每轮的输入是上一次轮变换的输出,

输出则作为下一轮的输入, 描述如下:

轮变换初始化, 判断是加密还是解密过程: $\text{Round}^{\text{null}} \xrightarrow{startRound} \text{Round}^{\text{running}}$;

轮变换停止: $\text{Round}^{\text{running}} \xrightarrow{stop} \text{Round}^{\text{stopped}}$;

轮变换暂停: $\text{Round}^{\text{running}} \xrightarrow{pause} \text{Round}^{\text{paused}}$;

轮变换暂停: $\text{Round}^{\text{paused}} \xrightarrow{continue} \text{Round}^{\text{running}}$;

明文经过前置变换处理作为加/解密轮变换的输入, 同时生成的轮密钥也将作为轮变换的输入。因此, Round 组件的输入有两个, 对两个值的输入定义如下:

根据前文分组密码结构分析, 明文在进行加密或解密时经历前置变换的处理, 随后方可输入至加/解密轮变换。为了明确定义这一处理过程, 需要规范明文经过前置变换后的值, 然后再输入到加密或解密的轮变换中进行进一步的处理。因此, 定义明文在经历前置变换后的数值状态, 作为输入到轮变换中的值: $\overline{startInput}\langle x\rangle$ .Round;

初始密钥输入: $\overline{startKey}\langle y\rangle$ .Round;

第 n 轮轮子密钥输入: $\overline{inputKey}\langle kn\rangle$ .Round;

为了区分明文在经历前置变换后的数值状态以及随后每一轮轮变换的输入, 需要规范每一轮轮变换的输入, 因此为了方便描述, 定义随后每一轮轮变换的输入为第 n 轮轮变换输入: $\overline{setRound}\langle pn\rangle$ .Round;

第 n 轮轮变换输入在经过加解密变换后输出第 n 轮轮变换后的数值状态, 因此定义第 n 轮轮变换后的数值状态作为第 n 轮轮变换输出: $setRound(pn)$ .Round;

轮变换设计的主要挑战在于如何利用 Round 组件实现 S 盒替换和 P 盒置换, 使得数据在每一轮中既能充分混淆, 又不会导致算法过于复杂, 影响计算效率, 因此采用动作 *Round_n* 描述轮变换中的每一轮计算结果。

当轮变换进行初始化时, 动作 *startRound* 判断当前是加密还是解密过程, 描述如下:

$(\overline{startInput}\langle x\rangle \parallel \overline{startKey}\langle y\rangle)$.Round $^{\text{null}} \xrightarrow{startRound} setRound(p0)$ .Round $^{\text{running}}$;

当 Round 组件进行轮变换时, 描述如下:

$\overline{setRound}\langle p0\rangle$ .Round $^{\text{running}} \xrightarrow{Round\_1} setRound(p1)$ .Round $^{\text{running}}$;

当加解密轮变换进行 32 轮轮变换时, 描述如下:

$\overline{setRound}\langle p0\rangle$ .Round$^{\text{running}} \xrightarrow{\langle Round_1, Round_2, \cdots, Round_32\rangle} \overline{setRound}\langle p32\rangle$ .Round $^{\text{running}} \xrightarrow{stop} seyRound(p32)$ .Round$^{\text{stopped}}$;

若进行若干轮轮密钥变换, 对轮数进行计数, 未达到指定轮数则继续轮变换, 以 *roundCount* 表示当前轮数, *roundOut* 表示指定轮数, 描述如下:

$\overline{setRound}\langle p0\rangle$ .Round$^{\text{running}} \xrightarrow{Round}$ Component|([*roundCount* ≤ *roundOut*] Round$^{\text{running}}$+[*roundCount* > *roundOut*] Round$^{\text{stopped}}$)

3.3 InitialTransformation 组件

InitialTransformation 组件负责在加密轮变换之前对明文进行初步处理。该组件的作用是对输入的明文或数据进行预处理, 使得数据结构更适合后续的加密轮操作。在某些加密算法中, 前置变换是为了提高数据的混淆性, 使其在进入轮变换时更加复杂难解。

在某些密码算法中并不包含前置变换, 而在某些复杂的算法中, 前置变换的存在能够有效增强加密算法的安全性和抗攻击能力。由于前置变换只进行一次计算, 且输入值固定, 设置有初始状态(null)和运行状态(running), 设置前置变换初始化(*startInitial*)、停止前置变换(*stopInitial*)、前置变换(*Initial*)3 个动作组成, 描述如下:

前置变换初始化: $\text{InitialTransformation}^{\text{null}} \xrightarrow{startInitial} \text{InitialTransformation}^{\text{running}}$;

前置变换停止: $\text{InitialTransformation}^{\text{running}} \xrightarrow{stop} \text{InitialTransformation}^{\text{stopped}}$;

当 InitialTransformation 组件进行前置变换时, 明文要进行数学运算作为加/解密轮变换的输入, 定义如下:

明文输入: $\overline{startPlain}\langle x\rangle$ .InitialTransformation;

明文在经过前置变换后的输出数值状态将作为 Round 组件输入数值状态, 规范前置变换后的输出数值可借用 Round 组件输入数值状态 $\overline{startInput}\langle x\rangle$ .Round, 因此定义前置变换输出: $startInput(y)$ .InitialTransformation;

当明文进行前置变换时, 描述如下:

$\overline{startPlain}\langle x\rangle$ .InitialTransformation$^{\text{running}} \xrightarrow{Initial} startInput(y)$ .InitialTransformation$^{\text{running}}$;

3.4 FinalTransformation 组件

FinalTransformation 组件负责在所有轮变换结束后对数据进行最终的处理, 确保加/解密结果的正确性和一致性。在某些加密算法中, 最后一轮加密操作可能会与前几轮有所不同, 因此后置变换的设计显得尤为重要。

FinalTransformation 组件与 InitialTransformation 组件类似, 并不是每一个分组密码算法都存在, 同样设置有初始状态(null)和运行状态(running), 设置有后置变换初始化(*startFinal*)、停止轮变换(*stop*)、前置变换(*Final*)3 个动作组成, 描述如下:

后置变换初始化: $\text{FinalTransformation}^{\text{null}} \xrightarrow{startFinal} \text{FinalTransformation}^{\text{running}}$;

后置变换停止: $\text{FinalTran} \xrightarrow{stop} \text{sformation}^{\text{running}} \xrightarrow{stop} \text{FinalTransformation}^{\text{stopped}}$;

当 FinalTransformation 组件进行前置变换时, 加解密轮变换的输出要进行数学运算得到最终的密文, 定义如下:

后置变换输入: $\overline{setRound}\langle x\rangle$. FinalTransformation;

后置变换输出: *setResult*(*y*). FinalTransformation;

当进行后置变换时, 描述如下:

$\overline{setRound}\langle x\rangle$ .FinalTransformation$^{\text{running}} \xrightarrow{Final} setResult(y)$.FinalTransformation$^{\text{running}}$;

4 案例应用

本文采用形式化设计方法对 TANGRAM 密码算法和 SM4 密码算法进行应用, 相关代码已公布在码云 https://gitee.com/xu-dadan/implementation-code.git。

本文首先采用 TANGRAM 密码算法的加密过程进行形式化设计, 采用四大组件对 TANGRAM 加密算法进行形式化表达, 结合范畴理论将其形式化表达转换得到 TANGRAM 加密算法的源范畴图, 根据源范畴图将行为描述映射成 MCL 元密码语言的行为描述, 其后, 基于范畴图的行为描述及映射关系得到基于 MCL 元密码模型的 TANGRAM 加密算法实现, 设计流程如图 5 所示。

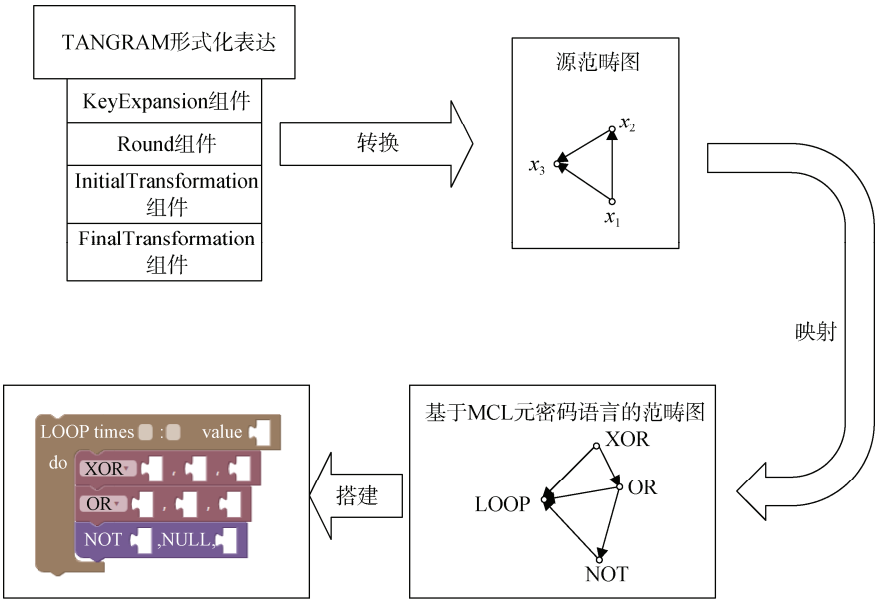

图 5 TANGRAM 密码算法的加密过程设计流程

Figure 5 TANGRAM cipher algorithm encryption process design flow

4.1 TANGRAM 密码算法

TANGRAM[20]是由张文涛等人设计的一种 SP 网络结构密码算法, 该算法采用比特切片方法来设计多个软硬件平台可用的系列分组密码算法。TANGRAM 算法即像七巧板一样, 能够适用于多种不同的应用场景, 这是其得名的原因。在 2019 年全国密码算法设计竞赛中, 该算法获得二等奖[21]。TANGRAM 系列密码算法共有三个版本, 分别为 TANGRAM-128/128 、 TANGRAM-128/256 、 TANGRAM-256/256, 本文以 TANGRAM-128/128 为参考应用形式化表达的方法实现其加密过程。

TANGRAM 系列密码算法采用 SP 网络结构, 总共包含 44 轮轮加密, 每个加密结构算法的最后增加一个子密钥异或操作, 不需要采用 S 盒。每一轮变换包含三个步骤: 轮密钥加 *AddRoundKey*(*ARK*), 列替换 *SubColumn*(*SC*), 行移位 *ShiftRow*(*SR*)。用 R_i 表示在轮子密钥 K_i 作用下轮函数变换的输出数据, m、c 分别表示明文和密文, TANGRAM-128/128 的算法结构如图 6 所示。

明文 m 经过 44 轮轮加密, 最后一轮增加一个轮子密钥 K_{44} 异或操作, 不需要采用 S 盒。最终得到密文 c。

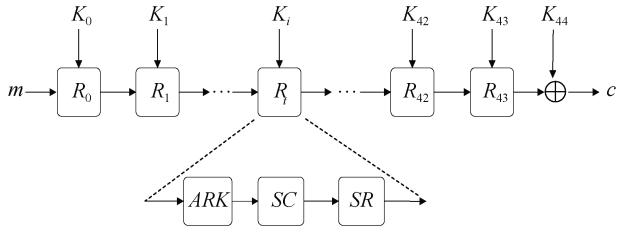

图 6 TANGRAM-128/128 的算法结构

Figure 6 The algorithm structure of TANGRAM-128/128

4.2 TANGRAM 加密算法的形式化表达

本文采用 TANGRAM 密码算法的加密过程进行形式化设计, 采用四大组件对 TANGRAM 加密算法进行形式化表达, 结合范畴理论将其形式化表达转换得到 TANGRAM 加密算法的源范畴图, 根据源范畴图将行为描述映射成 MCL 元密码语言的行为描述, 其后, 基于范畴图的行为描述及映射关系得到基于 MCL 元密码模型的 TANGRAM 加密算法实现。

对 TANGRAM 加密算法的整体描述为:

($\overline{startInput}\langle x\rangle \parallel \overline{startKey}\langle y\rangle$).Round$^{\text{null}} \xrightarrow{startRound}$ $setRound(p0)$ .Round$^{\text{running}}$; //并行输入初始密钥 y 和明文 x, *startRound* 选择加密行为, 输出 $p0$ 作为轮变换的初始输入, 组件 Round 由 null 状态变为 running 状态, 由于 TANGRAM 加密算法没有前置变换, 此时 $p0$ 等于明文 x

$\overline{setRound}\langle p0\rangle$ .Round$^{\text{running}} \xrightarrow{\langle Round_1, Round_2, \cdots, Round_43\rangle}$ $\overline{setRound}\langle p43\rangle$ .Round$^{\text{running}} \xrightarrow{stop}$ $setRound(p43)$ .Round$^{\text{stopped}}$; //$p0$ 经过 43 轮轮变换后输出 $p43$

$\overline{setRound}\langle p43\rangle$ .FinalTransformation$^{\text{null}} \xrightarrow{startFinal}$ FinalTransformation$^{\text{running}} \xrightarrow{Final}$ $setResult(p44)$. FinalTransformation $^{\text{running}} \xrightarrow{stop}$ FinalTransformation$^{\text{stopped}}$; //进行置后变换得到密文 $p44$

第 2 句 43 轮轮变换可简化为:

$\overline{setRound}\langle p0\rangle$ .Round$^{\text{running}} \xrightarrow{Round_n}$ Component|([$n \le 43$] $setRound(pn)$ .Round$^{\text{running}}$+[$n > 43$] $setRound(p43)$ .Round$^{\text{stopped}}$)

对以上描述进行整合, 整体进程描述为:

($\overline{startInput}\langle x\rangle \parallel \overline{startKey}\langle y\rangle$).Round$^{\text{null}} \xrightarrow{startRound}$ $\overline{setRound}\langle p0\rangle$ .Round$^{\text{running}} \xrightarrow{Round_n}$ Component|([$n \le 43$] $seyRound(pn)$ .Round$^{\text{running}}$+[$n > 43$] $setRound(p43)$ .Round$^{\text{stopped}}$) $\xrightarrow{startFinal}$ FinalTransformation$^{\text{running}} \xrightarrow{Final}$ *setResult*($p44$).FinalTransformation $^{\text{running}} \xrightarrow{stop}$ FinalTransformation$^{\text{stopped}}$

TANGRAM 加密过程源范畴图, 如图 7 所示。其中, 组件 x, y 均为输入组件, 仅作为轮变换的输入, 组件 Round 的 Round_n、组件 FinalTransformation 的 Final 为组件内部行为, 组件内部行为不同于组件外部行为, 无法在范畴图中体现, 输出值的传输行为以及其他行为 *startRound*、*startFinal*、*startARK*、*startSC*、*startSR* 均由箭头表示。

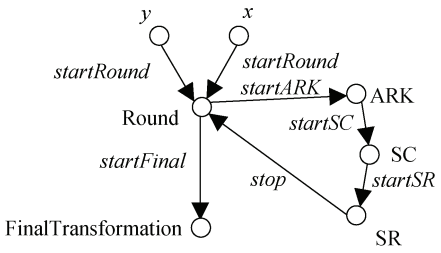

图 7 TANGRAM 加密过程源范畴图

Figure 7 Source category diagram of TANGRAM encryption process

其中, 轮变换组件 Round 包括 3 个步骤, 因此定义三个组件 ARK, SC, SR 以及动作集 $Round\_n$:=$\langle startARK, ARK, SC, SR\rangle$, 其中 *startARK* 指开始轮密钥加动作, *ARK* 指轮密钥加动作, *SC* 指列替换动作, *SR* 指行移位动作, 动作 *ARK* 为与组件 ARK 区分开统一用斜体表示, 动作 $Round\_n$ 是由 *startARK*, *ARK*, *SC*, *SR* 四个动作按顺序构成的动作集, 进程描述为: $\overline{setRound}\langle p0\rangle$ .Round$^{\text{running}} \xrightarrow{\langle startARK, ARK, SC, SR\rangle}$ Component|([$n \le 43$] $seyRound(pn)$ .Round$^{\text{running}}$+[$n > 43$] $setRound(p43)$ .Round$^{\text{stopped}}$)。

TANGRAM 算法中, 密钥扩展采用了特定的 S 盒操作来避免已知的攻击方式, 但如何选择通过四大组件设计出 TANRAM 算法仍然是一个挑战, 因此为满足基于 MCL 元密码语言的密码算法设计, 将 *ARK*, *SC*, *SR* 三个动作映射为 MCL 元密码语言动作分析。

(1) 轮密钥加(AddRoundKey)

将 128 bit 的轮子密钥逐比特与 128 bit 的明文异或, 设轮子密钥为 $SK = SK_3 \parallel SK_2 \parallel SK_1 \parallel SK_0$, 轮密钥加的输入为 $I = I_3 \parallel I_2 \parallel I_1 \parallel I_0$, 轮密钥加的结果为 $A = A_3 \parallel A_2 \parallel A_1 \parallel A_0$, 公式如下:

$$A_0 = I_0 \oplus SK_0 \tag{1}$$

$$A_1 = I_1 \oplus SK_1 \tag{2}$$

$$A_2 = I_2 \oplus SK_2 \tag{3}$$

$$A_3 = I_3 \oplus SK_3 \tag{4}$$

由于 MCL 元密码语言的位宽为 32 bit, 需要经过 4 个异或运算, 因此, 为了明确定义异或运算过程, 引入了一个名为 XOR 的组件定义组件 XOR。组件 XOR 的内部行为被规定为能够执行异或运算。组件 ARK 的动作 $ARK=\langle XOR, XOR, XOR, XOR\rangle$, 组件 ARK 的进程描述为:

$$\overline{startInput}\langle I\rangle \,\|\, \overline{startKey}\langle SK\rangle .\mathrm{Round}^{\mathrm{null}} \xrightarrow{startRound} \overline{startInput}\langle I\rangle \,\|\, \overline{startKey}\langle SK\rangle .\mathrm{Round}^{\mathrm{running}} \xrightarrow{startARK} \overline{startInput}\langle I\rangle \,\|\, \overline{startKey}\langle SK\rangle .\mathrm{ARK}^{\mathrm{running}} \xrightarrow{\langle XOR,XOR,XOR,XOR\rangle} setResult(A).\mathrm{ARK}^{\mathrm{running}} \xrightarrow{stop} \mathrm{ARK}^{\mathrm{stopped}}$$

(2) 列替换(SubColumn)

对每一列的 4bit 进行 S 盒替换, 形式如下:

$$\begin{pmatrix} a_{0,31} \\ a_{1,31} \\ a_{2,31} \\ a_{3,31} \end{pmatrix} \cdots \begin{pmatrix} a_{0,1} \\ a_{1,1} \\ a_{2,1} \\ a_{3,1} \end{pmatrix} \begin{pmatrix} a_{0,0} \\ a_{1,0} \\ a_{2,0} \\ a_{3,0} \end{pmatrix}$$

$$\downarrow S \quad \downarrow S \quad \downarrow S$$

$$\begin{pmatrix} b_{0,31} \\ b_{1,31} \\ b_{2,31} \\ b_{3,31} \end{pmatrix} \cdots \begin{pmatrix} b_{0,1} \\ b_{1,1} \\ b_{2,1} \\ b_{3,1} \end{pmatrix} \begin{pmatrix} b_{0,0} \\ b_{1,0} \\ b_{2,0} \\ b_{3,0} \end{pmatrix}$$

TANGRAM 的 S 盒设计很巧妙, 可以通过逻辑运算得到 S 盒变换的值, 详见参考文献, 本文不过多赘述。由于除了异或运算, 还需用到与、或、非运算, 因此, 定义组件 AND、OR、NOT, 组件内部行能进行相应的逻辑运算, 组件 SC 的动作定义为:

$SC=\langle XOR, AND, XOR, XOR, XOR, AND, XOR, OR, XOR, NOT, AND, XOR\rangle$

$A = A_3 \,\|\, A_2 \,\|\, A_1 \,\|\, A_0$ 为输入, $B = B_3 \,\|\, B_2 \,\|\, B_1 \,\|\, B_0$ 为输出, 组件 SC 的进程描述为:

$$\overline{startInput}\langle A\rangle .\mathrm{SC}^{\mathrm{null}} \xrightarrow{startSC} \overline{startInput}\langle A\rangle .\mathrm{SC}^{\mathrm{running}} \xrightarrow{\langle XOR,AND,XOR,XOR,XOR,AND,XOR,OR,XOR,NOT} \xrightarrow{AND,XOR\rangle} setResult(B).\mathrm{SC}^{\mathrm{running}} \xrightarrow{stop} \mathrm{SC}^{\mathrm{stopped}}$$

(3) 行移位(ShiftRow)

TANGRAM 对每一行的 32 bit 做左循环的二次移位。第 0 行保持固定不动, 第 1 行左循环移动 1 bit, 第 2 行左循环移动 8 bit, 第 3 行左循环移动 11 bit, 如下所示:

$$(a_{0,31}\cdots a_{0,1}a_{0,0}) \xrightarrow{<<<0} (a_{0,31}\cdots a_{0,1}a_{0,0}) \tag{5}$$

$$(a_{1,31}\cdots a_{1,1}a_{1,0}) \xrightarrow{<<<1} (a_{1,30}\cdots a_{1,0}a_{1,31}) \tag{6}$$

$$(a_{2,31}\cdots a_{2,1}a_{2,0}) \xrightarrow{<<<8} (a_{2,23}\cdots a_{2,25}a_{2,24}) \tag{7}$$

$$(a_{3,31}\cdots a_{3,1}a_{3,0}) \xrightarrow{<<<11} (a_{3,20}\cdots a_{3,22}a_{3,21}) \tag{8}$$

式中, <<<x 表示左循环移动 x bit。

由于采用了 4 个移位运算, 因此, 定义组件 ROL32, 组件内部行能进行循环左移逻辑运算, 组件 SR 的动作定义为 $SR=\langle ROL32, ROL32, ROL32, ROL32\rangle$, $B = B_3 \,\|\, B_2 \,\|\, B_1 \,\|\, B_0$ 为输入, $C = C_3 \,\|\, C_2 \,\|\, C_1 \,\|\, C_0$ 为输出, 组件 SR 的进程描述为:

$$\overline{startInput}\langle B\rangle .\mathrm{SR}^{\mathrm{null}} \xrightarrow{startSR} \overline{startInput}\langle B\rangle .\mathrm{SR}^{\mathrm{running}} \xrightarrow{\langle ROL32,ROL32,ROL32,ROL32\rangle} setResult(C).\mathrm{SR}^{\mathrm{running}} \xrightarrow{stop} \mathrm{SR}^{\mathrm{stopped}}$$

现已对轮变换的三个步骤形式化, 转化为范畴图, 按照范畴图搭建 MCL 元密码模型, 如图 8 所示。

组件 XOR、NOT、ROL32 分别采用 XOR 元模型、NOT 元模型、ROL32 元模型按基于 MCL 元密码语言的范畴图搭建拼接成基于 MCL 元密码模型的 TANGRAM 模型。

现根据拼接而成的 TANGRAM 加密算法的 MCL 模型, 在由北京电子科技学院独立设计了一种基于模型驱动的密码算法可视化开发平台(即 MetaCrypto 平台)上运行, 在代码生成器一栏点击“Python”按钮, 映射为 Python 代码, 再点击“编译”按钮, 对映射生成的 Python 代码进行编译。TANGRAM 加密算法参考文档的测试向量如表 2 所示, 可见参考文献[20]。

生成的 Python 代码的编译结果如图 9 所示。

图 9 的具体输入和输出如下所示:

明文: 1D F2 09 F5 D5 5D 21 14 0A 55 D0 FC 33 77 3F D5

密钥: 1A A1 BD 18 7B 6E B9 D0 FA D1 24 EF A3 81 17 B9

加密值: 23 E1 85 44 6F 50 48 6E 0F 9C D6 84 F2 11 67 94

实验测试结果表明, 平台输出的结果与测试向量完全相同, MetaCrypto 平台下 TANGRAM 加密算法的设计与实现是正确的。

4.3 SM4 密码算法的形式化表达

SM4 密码算法是由中国国家密码管理局发布的一种对称分组加密算法, SM4 密码算法流程可见参考文献[22], 本文不再详细阐述。

SM4 密码算法是一个混合结构的分组密码算法, 其中也存在 SP 网络结构, 因此 SM4 密码算法分为三部分, 即密钥扩展算法、加密算法和解密算法, 本文分别对三部分进行形式化设计。

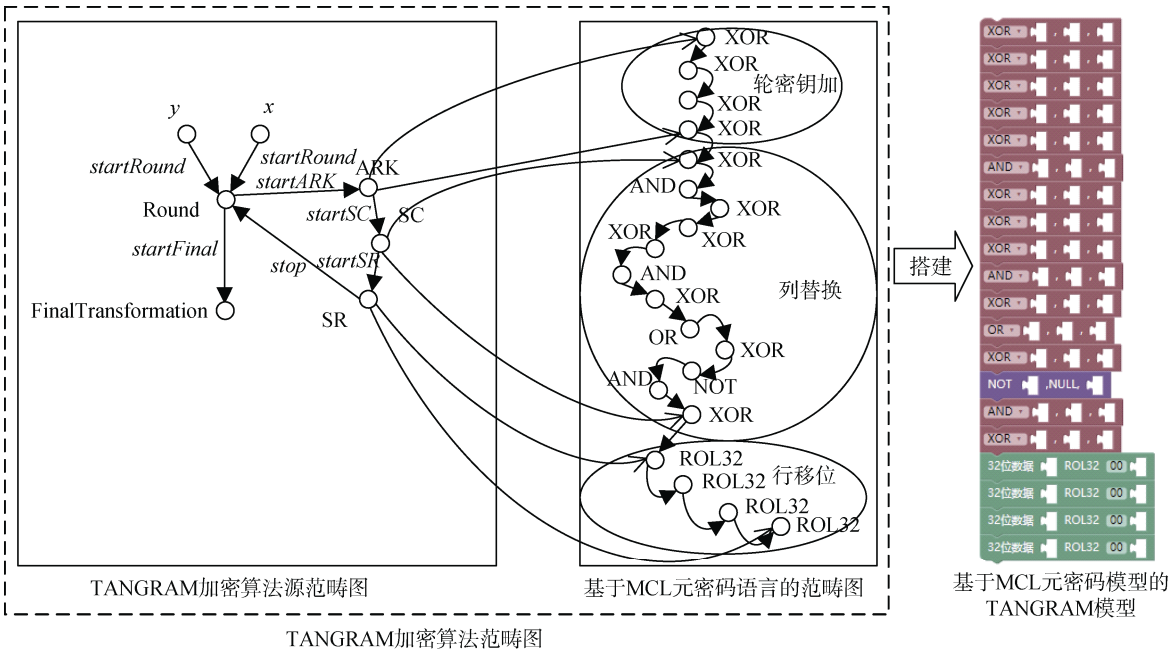

图 8　TANGRAM 加密过程范畴图

Figure 8　Category diagram of TANGRAM encryption process

表 2　TANGRAM 加密算法测试向量

Table 2　TANGRAM encryption algorithm test vector

映射语言	明文	密钥	密文
Python	0x1df209f5d55d2114a55d0fc33773fd5	0x1aa1bd187b6eb9d0fad124efa38117b9	0x23e185446f50486ef9cd684f2116794

(1) 密钥扩展算法

SM4 密码算法密钥扩展部分的整体描述为:

$\overline{startKey\langle x\rangle}$ .KeyExpansion $\xrightarrow{\text{null}}$ $\overline{start}$

$\overline{RoundKey\langle k0\rangle}$ .KeyExpansion $\xrightarrow{\text{running}}$ $\overline{ExpandKey\_n}$

Component|([$n \le 31$] $RoundKey(kn)$. KeyExpansion$^{\text{running}}$+[$n > 31$] $RoundKey(k31)$ .KeyExpansion$^{\text{stopped}}$);

其中, 组件 x 为输入组件, 作为初始密钥输入, 初始化密钥的行为由 *start* 表示, 32 轮轮密钥变换的行为由 *ExpandKey_n* 表示。

图 9　TANGRAM 加密算法的 Python 代码正确性验证

Figure 9　Verification of Python code correctness of TANGRAM encryption algorithm

为了更加明确运算过程, 引入名为 XOR_XOR、Sbox、ROL32、ROL32_XOR 定义组件 XOR_XOR、Sbox、ROL32、ROL32_XOR, 均执行 MCL 元密码语言设计原则中相应的逻辑运算, 轮变换进程描述为:

$\overline{RoundKey}\langle k0\rangle$ .KeyExpansion^{running} $\xrightarrow{\langle XOR,XOR\_XOR,Sbox,ROL32,ROL32\_XOR,ROL32\_XOR\rangle}$ $\xrightarrow{\langle XOR,ROL32\rangle}$ Component|([$n \le 31$] $RoundKey(kn)$. KeyExpansion^{running}+[$n>$ 31] $RoundKey(k31)$. KeyExpansion^{stopped})

现已对密钥扩展过程形式化, 转化为基于 MCL 元密码语言的密码算法范畴图。如图 10 所示。

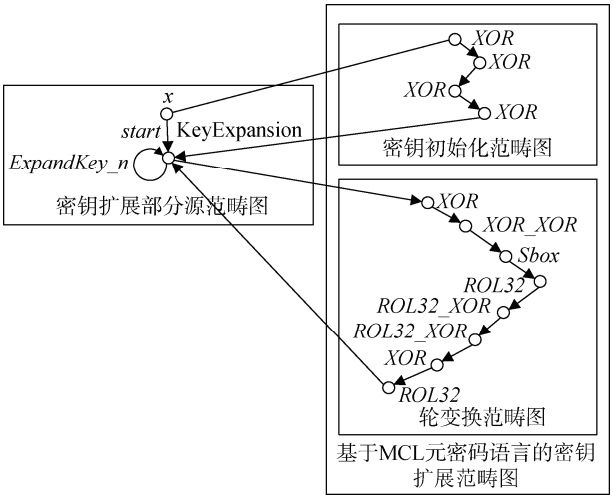

图 10 SM4 密码算法密钥扩展部分范畴图

Figure 10 SM4 Cryptographic algorithm key extension category diagram

(2) 加密算法

SM4 密码算法加密算法部分的整体描述为:

($\overline{startInput}\langle y\rangle$ || $\overline{startKey}\langle k0\rangle$).Round^{null} $\xrightarrow{startRound}$ $\overline{setRound}\langle p0\rangle$ .Round^{running} $\xrightarrow{Round\_n}$ Component|([$n \le 31$] $setRound(pn)$ .Round^{running}+[$n>$ 31] $setRound(p31)$. R-ound^{stopped}) $\xrightarrow{startFinal}$ FinalTransformation^{running} $\xrightarrow{Final}$ $setResult(m)$. FinalTransformation^{running} $\xrightarrow{stop}$ FinalTransformation^{stopped};

其中, 组件 y、$k0$ 为输入组件, 作为初始密钥和明文输入, 选择加密流程的行为由 *startRound* 表示, 32 轮轮密钥变换的行为由 *Round_n* 表示, 后置变换初始化由 *startFinal* 表示, 后置变换由 *Final* 表示, 均以箭头的形式呈现。

为了更加明确运算过程, 轮变换进程描述为:

$\overline{setRound}\langle p0\rangle$ .Round^{running} $\xrightarrow{\langle XOR,XOR\_XOR,Sbox,ROL32,ROL32\_XOR,ROL32\_XOR,ROL32\_XOR\rangle}$ $\xrightarrow{\langle ROL32\_XOR,XOR\rangle}$ Component|([$n \le 31$] $setRound(pn)$. Round^{running}+[$n>$ 31] $setRound(p31)$ .Round^{stopped});

后置变换描述为:

$\overline{setRound}\langle p31\rangle$ .FinalTransformation^{null} $\xrightarrow{startFinal}$ FinalTransformation^{running} $\xrightarrow{\langle ROL32,ROL32,ROL32,ROL32\rangle}$ $setResult(m)$.FinalTransformation^{running} $\xrightarrow{stop}$ FinalTransfor-mation^{stopped};

现已对加密过程形式化, 转化为基于 MCL 元密码语言的密码算法范畴图。如图 11 所示。

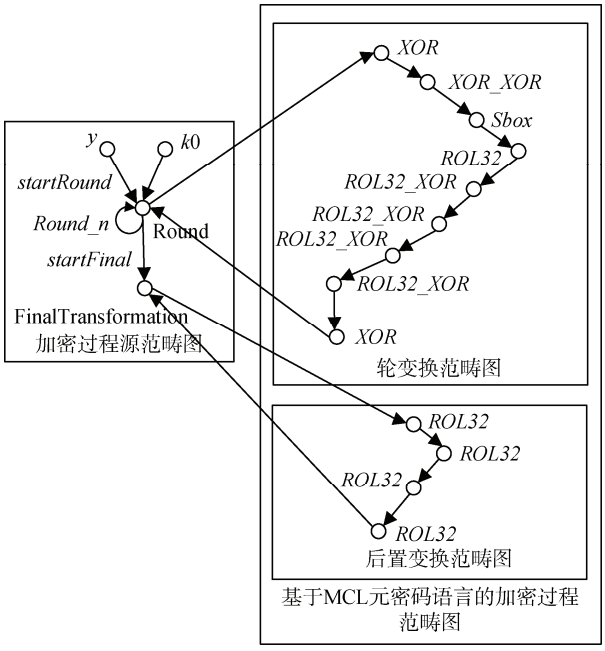

图 11 SM4 加密过程范畴图

Figure 11 SM4 Category diagram of the encryption process

(3) 解密算法

SM4 密码算法解密过程的整体描述为:

($\overline{startInput}\langle m\rangle$ || $\overline{startKey}\langle k31\rangle$).Round^{null} $\xrightarrow{startRound}$ $\overline{setRound}\langle p0\rangle$ .Round^{running} $\xrightarrow{Round\_n}$ Component|([$n \le 31$] $setRound(pn)$ .Round^{running}+[$n>$ 31] $setRound(p31)$ .Round^{stopped}) $\xrightarrow{startFinal}$ FinalTransformation^{running} $\xrightarrow{Final}$ $setResult(y)$.FinalTransformation^{running} $\xrightarrow{stop}$ FinalTransformation^{stopped};

整体描述中 startRound 选择为解密状态, 密文 m 和第 32 轮轮密钥 $k31$ 作为输入, 明文 y 作为输出。

解密变换与加密变换的流程基本相同, 但在解

密变换中所使用的轮密钥为加密变换的逆序。解密过程如图 12 所示。

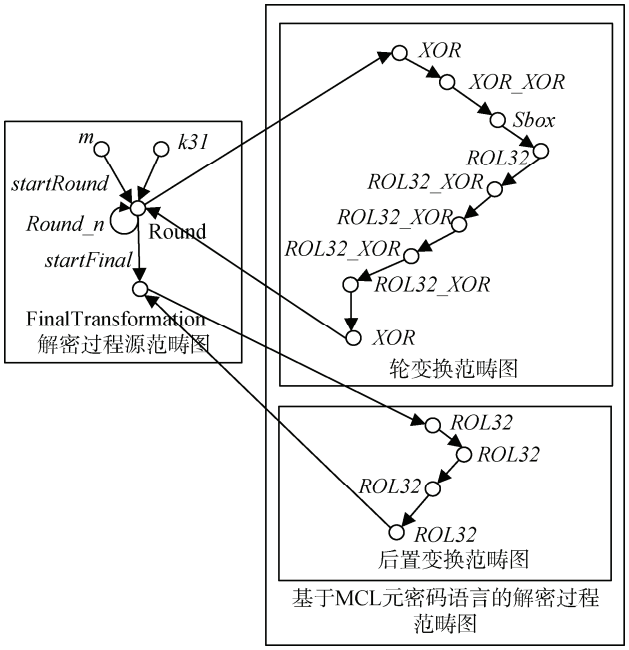

图 12　SM4 解密过程范畴图

Figure 12　SM4 decryption process category diagram

对 SM4 密码算法结构进行拆解形式化设计，并根据形式化设计的范畴图实现对应部分的可视化模型，最终将各个部分进行拼接。根据拼接而成的 SM4 密码算法的 MCL 模型在 MetaCrypto 平台上运行。

图 13 的具体输入和输出如下所示:

明文:　01 23 45 67 89 AB CD EF FE DC BA 98 76 54 32 10

密钥:　01 23 45 67 89 AB CD EF FE DC BA 98 76 54 32 10

加密值:　68 1E DF 34 D2 06 96 5E 86 b3 E9 4F 53 6E 42 46

解密值:　01 23 45 67 89 AB CD EF FE DC BA 98 76 54 32 10

实验测试结果表明，平台输出的结果与参考文献[21]的测试向量完全相同, MetaCrypto 平台下 SM4 密码算法的设计与实现是正确的。

5　与传统设计方法对比分析

综合参考文献[23-25]中传统设计方法，本文提出的形式化设计方法具有一定的优势与创新，具体情况如表 3 所示。

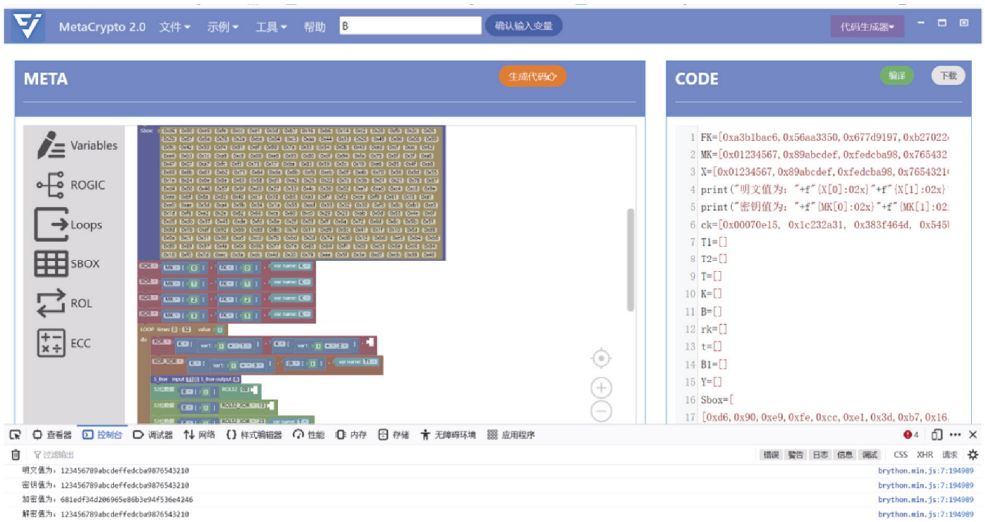

图 13　SM4 密码算法的 Python 代码正确性验证

Figure 13　Python code correctness verification of SM4 cryptographic algorithm

表 3　传统设计方法与形式化设计方法对比

Table 3　Comparison between traditional design methods and formal design methods

设计方法	系统性	准确性	依赖性	可维护性	适用性
传统设计方法			√		√
形式化设计方法	√	√		√	√

注:　“√”表示该设计方法具备的特性

在设计过程的系统性方面，传统方法主要依赖设计者的个人经验和手工实现，缺乏系统化和标准化，而形式化设计方法利用进程代数和 MCL 元密码语言提供了系统化和标准化的描述，显著提高了设计过程的一致性和可重复性。在设计准确性上，传统方法容易受到人为错误的影响，设计准确性难以保证，而形式化设计方法通过精确的语言描述和自动化工具验证，显著提高了设计的准确性。在依赖性方

面, 传统方法高度依赖设计者的个人经验和技能, 设计质量不稳定, 而形式化设计方法减少对个人经验的依赖, 通过系统化的语言和工具确保设计质量稳定。传统方法的设计文档和代码难以维护与扩展, 新设计者上手困难。在可维护性方面, 形式化设计方法通过标准化的描述语言和形式化的表达方式, 设计清晰易懂, 便于维护和扩展。此外, 传统方法的安全性验证需要大量的手工测试和分析, 难以覆盖所有潜在漏洞, 而形式化设计方法利用自动化验证工具 MetaCrypto 平台, 可以快速检查设计中的错误和漏洞。在适用性方面, 传统方法也具有一定的灵活性, 但较为有限, 通常针对特定的应用场景进行设计, 难以跨场景应用, 且每次设计新的密码算法都需要大量的定制化工作, 缺乏设计的复用性, 而形式化设计方法则通过使用标准化的语言和工具, 具有较高的灵活性, 能够适应不同的应用场景和需求。总体来看, 形式化设计方法在系统性、准确性、可维护性和适用性方面均优于传统方法。

6 结论

本文通过设计一种新的领域专用语言, 即 MCL 元密码语言(MetaCrypto Language), 有效解决了 SP 网络结构密码算法设计和实现中的关键问题。MCL 元密码语言结合软件开发与硬件实现的特点, 专门针对密码算法开发而设计, 体现了对于密码算法设计的高效性和易用性, 为密码算法设计者提供了强大的工具。通过对分组密码算法分析, 采用基于进程代数的形式化方法详细描述了 SP 网络结构密码的 4 大组件, 该方法从组件层次提供了一种新的构建 SP 网络结构密码算法的范畴图, 建立了 TANGRAM 分组密码算法和 SM4 密码算法的形式化模型, 为基于 MCL 元密码语言的 SP 网络结构密码算法设计提供了坚实的理论支撑, 不仅改善了传统上依赖设计者编码经验的问题, 还提高了 SP 网络结构密码设计的效率, 且形式化设计方法在系统性、准确性、可维护性和适用性方面均优于传统方法。最后, TANGRAM 加密算法验证结果表明, TANGRAM 加密算法和 SM4 密码算法的形式化设计与实现是可行的。

参考文献

[1] Liu J W, Lin J Q, Huang X Y, et al. Preface of Special Issue on Security Applications of Cryptography[J]. *Journal of Cryptologic Research*, 2020, 7(3): 285-289.
(刘建伟, 林璟锵, 黄欣沂, 等. 密码应用安全专刊序言(中英文)[J]. *密码学报*, 2020, 7(3): 285-289.)

[2] Xiong H, Qu Z, Huang X, et al. Revocable and Unbounded Attribute-Based Encryption Scheme with Adaptive Security for Integrating Digital Twins in Internet of Things[J]. *IEEE Journal on Selected Areas in Communications*, 2023, 41(10): 3306-3317.

[3] Elbirt A J. Understanding and Applying Cryptography and Data Security[M]. Boca Raton, Fla.: CRC Press, 2009.

[4] Koblitz N, Menezes A. The brave new world of bodacious assumptions in cryptography[J]. *Notices of the American Mathematical Society*, 2010, 57(3): 357-365.

[5] Vulpe A, Andrei R, Brumaru A, et al. Lightweight Cryptographic Algorithm Implementation in a Microcontroller System[J]. *STRATEGIES XXI - Command and Staff College*, 2021, 17(1): 260-264.

[6] Miao Jiejun, Wang Ke. Research on Formal Design and Verification of Cryptographic module [C]. *Computer Security Professional Committee of China Computer Society, Computer Engineering and Application Society of China Electronics Society Computer Security Group. Proceedings of the 21st National Computer Security Academic Exchange Conference*, 2006: 3.
(苗洁君, 王克. 密码模块的形式化设计和验证研究[C]. *中国计算机学会计算机安全专业委员会, 中国电子学会计算机工程与应用学会计算机安全保密学组. 第二十一次全国计算机安全学术交流会论文集*, 2006: 3.)

[7] Peng C Y, Zhu C Y, Huang L, et al. Formal Function Cryptanalysis of Block Cipher and Its Application[J]. *Acta Electronica Sinica*, 2013, 41(11): 2314-2316.
(彭昌勇, 朱创营, 黄莉, 等. 对分组密码的形式化函数分析及其应用[J]. *电子学报*, 2013, 41(11): 2314-2316.)

[8] Zhang S Y, Wang Y J, Wang L, et al. Construction and Application of 4×4 Formalized MDS Matrices[J]. *Journal of Cryptologic Research*, 2018, 5(6): 680-694.
(张诗怡, 王永娟, 王磊, 等. 4×4 形式化 MDS 矩阵的构造与应用[J]. *密码学报*, 2018, 5(6): 680-694.)

[9] Bi X. Research on Protocol Security Analysis Technology Based on Formal Method[D]. Changsha: National University of Defense Technology, 2020.
(毕兴. 基于形式化方法的协议安全性分析技术研究[D]. 长沙: 国防科技大学, 2020.)

[10] Song F. Status and Prospects of Formal Verification for Security of Cryptographic Implementations[J]. *Science and Technology Foresight*, 2023, 2(1): 90-105.
(宋富. 密码实现安全形式化验证发展现状与展望[J]. *前瞻科技*, 2023, 2(1): 90-105.)

[11] Yasinsac A F. A formal semantics for evaluating cryptographic protocols[M]. University of Virginia, 1996.

[12] Briais S. Theory and tool support for the formal verification of cryptographic protocols[R]. EPFL, 2008.

[13] Pierce B C, Casinghino C, Gaboardi M, et al. Software foundations[J]. *Webpage: http://www. cis. upenn. edu/bcpierce/sf/current/index. html*, 2010: 16.

[14] Zhou Y J, Guan H M. Research of Formal Design of Authentication Protocols[C]. *2009 IEEE International Conference on Intelligent Computing and Intelligent Systems*, 2009: 25-28.

[15] Yamamoto D, Itoh K, Yajima J. Compact Architecture for ASIC and FPGA Implementation of the KASUMI Block Cipher[J]. *IEICE Transactions on Fundamentals of Electronics, Communications and Computer Sciences*, 2011, E94-A(12): 2628-2638.

[16] Simanjuntak H A, Aspriyono H, Rohmawan E P. Network-Based Text Message Security System Using the Data Encryption Standard (DES) Algorithm[J]. *Jurnal Komputer*, *Informasi Dan Teknologi*, 2022, 2(2): 613-620.

[17] Furkan Altınok K, Peker A, Tezcan C, et al. GPU Accelerated 3DES Encryption[J]. *Concurrency and Computation*: *Practice and Experience*, 2022, 34(9): e6507.

[18] Fathurrahmad E. Development and Implementation of the Rijndael Algorithm and Base-64 Advanced Encryption Standard (AES) for Website Data Security[J]. *International Journal of Scientific & Technology Research*, 2020, 9: 6-9.

[19] Ju W G. A New Process Algebra More Suitable for Formal Specification[C]. *2021 4th International Conference on Computer Science and Software Engineering*, 2021: 103-106.

[20] Zhang W T, Ji F L, Ding T Y, et al. TANGRAM: A Bit-Slice Block Cipher Suitable for Multiple Platforms[J]. *Journal of Cryptologic Research*, 2019, 6(6): 727-747.
(张文涛, 季福磊, 丁天佑, 等. TANGRAM: 一个基于比特切片的适合多平台的分组密码[J]. *密码学报*, 2019, 6(6): 727-747.)

[21] Wang J X, Xu H K, Zheng Y Z, et al. Implementation of TANGRAM Block Cipher Algorithm Based on FPGA[J]. *Application Research of Computers*, 2024, 41(1): 260-265.
(王建新, 许弘可, 郑玉峭, 等. 基于 FPGA 的 TANGRAM 分组密码算法实现[J]. *计算机应用研究*, 2024, 41(1): 260-265.)

[22] Zheng Dong, Chang Xiaoyang, Yang Zhonghuang. Efficient implementation of SM4 state cryptographic algorithm in Android kernel in assembly language[J]. *Journal of XI'AN university of posts and telecommunications*, 2021 ,26(6): 66-71.
(郑东, 常晓阳, 杨中皇. SM4 国密算法在 Android 内核的汇编语言快速实现[J]. *西安邮电大学学报*, 2021, 26(6): 66-71.)

[23] Heys H M. A Tutorial on the implementation of block ciphers: software and hardware applications [EB/OL]. (2020-12-11) [2023-7-9]. https://eprint.iacr.org/2020/1545.pdf.

[24] Geek. Block cipher design principles [EB/OL]. (2023-5-18) [2024-1-9]. https://www.geeksforgeeks.org/block-cipher-design-principles/ref=lbp.

[25] Raza A R, Mahmood K, Amjad M F, et al. On the Efficiency of Software Implementations of Lightweight Block Ciphers from the Perspective of Programming Languages[J]. *Future Generation Computer Systems*, 2020, 104: 43-59.

张磊 于 2007 年在中国农业大学车辆工程专业获得博士学位。现任北京电子科技学院教授。研究领域为信息安全、密码工程。研究兴趣包括: 信息安全、密码工程。Email: zhanglei@besti.edu.cn.

许弘可 于 2021 年在东华理工大学电子信息工程专业获得学士学位。现在北京电子科技学院电子信息专业攻读硕士学位。研究领域为电子信息工程、形式化语义、EDA 技术。研究兴趣包括: 图形化编程、EDA 工具设计。Email: 201720060827@ecut.edu.cn.

肖超恩 于 2010 年在北京航空航天大学电子信息专业获得博士学位。现任北京电子科技学院讲师。研究领域为嵌入式系统安全。研究兴趣包括: 嵌入式系统安全。Email: xce@besti.edu.cn.

王建新 于 2007 年在天津科技大学轻装备研发专业获得博士学位。现任北京电子科技学院正高级工程师。研究领域为 EDA 技术、密码工程。研究兴趣包括: EDA 工具设计、密码工程。Email: wangjianxin@besti.edu.cn.

郑玉峭 于 2021 年在烟台大学电子信息专业获得学士学位。现在北京电子科技学院学校新一代电子信息技术(含量子技术等)专业攻读硕士学位。研究领域为信息安全。研究兴趣包括: 信息安全。Email: 1002150413@qq.com.