软件学报 ISSN 1000-9825, CODEN RUXUEW
2025,36(11):4990−5007 [doi: 10.13328/j.cnki.jos.007368] [CSTR: 32375.14.jos.007368]

E-mail: jos@iscas.ac.cn
http://www.jos.org.cn
Tel: +86-10-62562563

后量子密码 Falcon 实现的形式化验证技术*

田新蕾[1], 董奕逸[1], 张济显[1], 王伟嘉[1,2]

[1](山东大学 网络空间安全学院, 山东 青岛 266237)

[2](泉城实验室, 山东 济南 250103)

通信作者: 王伟嘉, E-mail: wjwang@sdu.edu.cn

摘　要: Falcon 作为一种后量子数字签名算法, 被选为美国国家标准与技术研究院 (National Institute of Standards and Technology, NIST) 首批标准化方案之一. Falcon 的核心算法在实际实现中很容易出错, 引发可能的密码学误用问题. 对 Falcon 核心函数进行形式化验证, 以确保其正确性是非常重要的. 在过去 10 年中,计算机辅助密码学领域将形式化方法引入了密码学工程. 这使得高性能密码实现具有了强有力的函数正确性和特定实现安全性属性的形式化保证. 贡献包括: (1) 构建了完整的证明框架, 通过形式化验证方法, 成功弥合了 Falcon 的实际代码实现与数学描述之间的差距; (2) 在 EasyCrypt 证明体系中验证了 Falcon 中的 Montgomery 模乘、NTT 算法、FFT 算法的正确性, 并对整数高斯采样算法的正确性证明方法进行了探索; (3) 给出及优化了基于 Jasmin 混合编程的 Falcon 签名与验证实现.

关键词: 后量子密码; Falcon; 形式化验证; Jasmin; EasyCrypt

中图法分类号: TP311

中文引用格式: 田新蕾,董奕逸,张济显,王伟嘉.后量子密码Falcon实现的形式化验证技术.软件学报,2025,36(11):4990–5007.http://www.jos.org.cn/1000-9825/7368.htm

英文引用格式: Tian XL, Dong YY, Zhang JX, Wang WJ. Formal Verification Techniques for Implementing Post-quantum Cryptography Falcon. Ruan Jian Xue Bao/Journal of Software, 2025, 36(11): 4990–5007 (in Chinese). http://www.jos.org.cn/1000-9825/7368.htm

Formal Verification Techniques for Implementing Post-quantum Cryptography Falcon

TIAN Xin-Lei[1], DONG Yi-Yi[1], ZHANG Ji-Xian[1], WANG Wei-Jia[1,2]

[1](School of Cyber Science and Technology, Shandong University, Qingdao 266237, China)

[2](Quan Cheng Laboratory, Jinan 250103, China)

Abstract: Falcon, a post-quantum digital signature algorithm, has been selected as one of the first schemes standardized by the National Institute of Standards and Technology (NIST). Its core algorithms, however, are highly error-prone in practical implementations, raising risks of cryptographic misuse. Ensuring the correctness of Falcon through formal verification is therefore essential. In this work, this study introduces a comprehensive proof framework that bridges the gap between Falcon's mathematical specification and its real-world implementation. Within the EasyCrypt proof system, this study formally verifies the correctness of Falcon's Montgomery modular multiplication, NTT, and FFT algorithms, and further explores proof techniques for integer Gaussian sampling. Moreover, this study presents and optimizes Falcon's signing and verification implementations using Jasmin hybrid programming, thereby providing both formal correctness guarantees and practical efficiency.

Key words: post-quantum cryptography; Falcon; formal verification; Jasmin; EasyCrypt

---

* 基金项目: 国家重点研发计划 (2023YFA1009500, 2021YFA1000600); 国家自然科学基金 (62372273); 泉城实验室重点项目 (QCLZD202306)
收稿时间: 2024-07-25; 修改时间: 2024-10-08; 采用时间: 2024-10-29; jos 在线出版时间: 2025-08-20
CNKI 网络首发时间: 2025-08-21

1 引 言

2022 年 7 月, 美国国家标准与技术研究院 (National Institute of Standards and Technology, NIST) 宣布了首批后量子项目的标准化方案, 包括 3 个数字签名方案 (CRYSTALS-Dilithium[1]、Falcon[2]、SPHINCS+ [3]) 以及一个密钥封装机制 (KEM): CRYSTALS-Kyber[4]. 后量子密码算法的设计宗旨是在量子计算机时代到来后, 依然能够保持其安全性. 根据我们对现有的物理定律理解, 量子计算机理论上是可行的, 但要实现一个完全可操作的量子计算机, 仍需克服众多重大技术挑战. 一旦这样的量子计算机成为现实, 它将能够高效地破解基于数论的传统非对称加密和数字签名算法, 例如 RSA、DSA、Diffie-Hellman、ElGamal 及其椭圆曲线变体.

在这些方案中, Falcon 算法尤其引人注目. 它不仅因其创新的设计而受到关注, 还因为其在安全性、效率和实用性方面展现出的显著优势. Falcon 算法具备多项显著优点: 其内部采用的真正高斯采样器, 确保了密钥信息泄露的风险几乎可以忽略不计, 并且能够支持超过 2^{64} 个的海量签名. 得益于 NTRU 格点的应用, Falcon 的签名长度在保持相同安全等级的同时, 比其他格基签名方案更短, 而公钥大小则保持相近. 通过快速傅里叶采样技术, Falcon 实现了惊人的速度, 可以在普通计算机上每秒生成数千个签名, 验证速度快 5–10 倍. 其操作复杂度为 $O(n\log n)$, 这意味着在成本较低的情况下, Falcon 能够支持长期的安全参数. 此外, Falcon 的密钥生成算法经过增强, 使得其内存需求降至 30 KB 以下, 相比于以往的设计 (例如 NTRUSign[5]) 提高了百倍, 非常适合内存受限的小型嵌入式设备.

正是这些特性, 让 Falcon 算法在后量子密码学领域中独树一帜. 它基于专为格基签名方案设计的 GPV 理论框架[6], 并通过 NTRU 格点实例化这一框架. 此外, Falcon 算法采用了一种创新的陷门采样器——快速傅里叶采样, 进一步提升了其性能. Falcon 算法的核心难题在于求解 NTRU 格点上的短整数解 (SIS) 问题, 在当前的研究水平下, 即使有量子计算机的辅助, 也未能发现解决这一问题的高效算法.

Falcon 算法的这些创新特性不仅在理论上具有重要意义, 也为实际应用提供了强大的动力. 随着后量子密码学逐步由理论向实践发展, 我们可以对现有的加密软件进行革新性的改进. 在最近 10 年的发展中, 计算机辅助密码学领域[7]将形式化方法[8,9]引入了密码学工程, 这一变革极大地推动了加密技术的性能提升. NSS 库[10]和 BoringSSL 库[11]分别被 Firefox 浏览器和 Google 采用, 展示了高保障密码技术在实际应用中的潜力. 通过这种形式化的方法, 加密技术的实现不仅得到了功能性的正确性验证, 还确保了在具体部署时的安全性, 比如保障了内存的安全使用和防止了时间相关的侧信道攻击. 这些正式的保证机制为加密实现提供了额外的安全层, 增强了其在实际应用中的可靠性和安全性.

形式化验证是一种运用数学和逻辑工具来精确证实系统、程序或算法准确性的科学方法. 与传统的测试和验证方法不同, 形式化验证依靠严谨的数学定义和逻辑推理, 能够全面识别并排除潜在的缺陷和不确性. 通过构建数学模型来刻画系统行为, 运用逻辑推理和证明来确认系统符合其预定规范, 并且能够全面覆盖所有可能的输入和状态, 确保无遗漏.

现代的形式化验证工具能够自动产生证明, 减少人为操作, 提高效率和可靠性. 形式化验证在软件和硬件的验证中得到了广泛的应用, 特别是在那些对安全性要求极高的关键系统和敏感领域. 它不仅能够提供高度的可靠性, 发现深层次的缺陷, 而且形式化证明的过程本身也是一种精确的文档记录, 可以作为程序或系统的规范. 然而, 形式化验证也面临挑战, 比如其学习曲线较为陡峭, 且对复杂系统的验证需要投入大量的时间和精力.

在密码算法领域, 形式化验证技术的目标是设计出能够提供强大安全保障的密码系统, 这对现代通信和计算基础设施的安全至关重要. 通过形式化验证, 可以量化限制资源的对手破坏密码系统的可能性. 这种安全性证明通常基于归约法, 即通过构建一个受限的对手, 将其破坏密码构造的能力转化为破坏某个困难问题的能力. 这一过程称为可证明的安全性, 它理论上能够提供严格和详细的数学证明.

然而, 随着密码设计的复杂性增加, 验证过程变得更加容易出错且难以检查. 为了应对这些挑战, 开发支持构建和自动验证密码系统的机器校验框架成为一个充满希望的解决方案. 尽管在符号模型的密码学中已经存在许多这样的框架, 但在密码学家常用的计算模型中, 机器校验框架的研究仍然相对较少.

1.1 相关工作

1969 年, Hoare[12]提出了 Hoare 逻辑, 用于形式化证明程序的正确性. 在 20 世纪 80 年代, 模型检测 (model checking) 技术开始发展. Clarke 等人[13,14]引入了用时态逻辑进行模型检测的方法, 奠定了模型检测领域的基础. 这一时期, Church[15]的工作也为形式化语言和自动机理论的发展做出了贡献.

随着计算机性能的提高和算法的优化, 形式化验证工具开始逐步应用于实际系统. 模型检测工具如 SMV (symbolic model verifier)[16]和 SPIN (simple promela interpreter)[17]等开始出现, 并在工业界得到应用.

进入 21 世纪, 形式化验证技术在软件和硬件中的应用进一步扩展. 模型检测、定理证明和抽象解释等方法在复杂系统的验证中得到广泛应用. SAT 求解器 (boolean satisfiability problem solver) 和 SMT 求解器 (satisfiability modulo theories solver) 等工具在形式化验证中扮演重要角色.

现代形式化验证工具, 如 Coq、Isabelle、Z3 等, 具有更高的自动化程度和用户友好性, 降低了使用门槛, 提高了验证效率. 这些工具的进步极大地推动了形式化验证技术在各个领域的广泛应用和发展.

在密码学领域, Haselwarter 等人[18]开发了 SSProve, 这是一个在 Coq 中进行模块化密码证明的基础框架, 结合高层次的模块化证明和概率关系程序逻辑, 支持对组合协议的机器检查密码证明. Formosa Crypto 基于 Jasmin[19]和 EasyCrypt[20]开发了 Jasmin-EasyCrypt 工具链. 这是一个关于机器形式化验证密码学和高保障密码工程的项目. Bhargavan 等人[21]开发的 HACSPEC 是一种新兴的密码学规范语言, 连接了现有的工具, 如 Jasmin 语言和 SSProve 框架, 通过在 Coq 中进行形式验证, 确保密码学实现的高效性和安全性.

在 Jasmin 中实现高保证密码算法后, 可以对其进行特定实现安全性属性验证, 或进一步提取至 EasyCrypt 进行函数正确性和证密码构造安全性验证. Almeida 等人[22]在 Jasmin 编程语言中编写了对 SHA-3 哈希函数的高保障和高速度实现, 并使用 EasyCrypt 对其函数正确性、可证明安全性和时间攻击抵抗性进行了正式验证. Shivakumar 等人[23]在 Jasmin 框架中实现了一种针对 Spectre v1 攻击的高效密码实现方法, 保护了对称加密、椭圆曲线密码和 NIST 选定的基于格的 Kyber KEM. Firsov 等人[24]则在 EasyCrypt 中形式化了零知识协议及其安全性证明, 特别关注需要重绕技术的 sigma 协议.

对于后量子密码的形式化验证, Barbosa 等人[25]的工作迈出了重要的一步, 他们使用 EasyCrypt 完成了 Dilithium 在 ROM 中的 CMA 安全性的完全机械化证明. 随后, Almeida 等人[26]形式化验证了后量子密钥封装机制 Kyber, 以实现依赖于新一代高保障后量子密码软件的下一代密码原语的部署. 最后, Barbosa 等人[27]完成了对后量子签名方案 XMSS 的严格安全性证明, 并使用 EasyCrypt 证明体系正式验证.

1.2 本文工作

本文对在机器校验框架下形式化验证 Falcon 算法进行了初步探索, 并利用 Jasmin-EasyCrypt 证明体系实现, 本文工作的整体流程如图 1 所示. 首先, 本文用 Jasmin 语言实现了 Falcon 算法 (Falcon-512); 随后, 将函数提取至 EasyCrypt 端进行函数正确性验证, 确保函数正确实现了 Falcon 算法的数学描述; 之后, 我们给出及优化了基于 Jasmin 混合编程的实际实现代码 (x86 汇编); 最后, 我们对生成的汇编代码进行效率测试, 并与原始代码的效率进行比较. 这种方法不仅保证了算法实现的高效性, 还通过形式化验证确保了其正确性, 本文的最终目标是确保 Falcon 的实际代码实现与算法数学描述的一致性.

我们的贡献如下.

(1) 我们构建了一个完整的证明框架, 通过形式化验证方法, 成功弥合了 Falcon 的实际代码实现与数学理论描述之间的差距.

(2) 在 EasyCrypt 证明体系中验证了 Falcon 中的 Montgomery 模乘、NTT 算法、FFT 算法的正确性, 并对整数高斯采样算法的正确性证明方法进行了探索, 这些算法是构建 Falcon 签名的基石, 具有重要的数学价值.

(3) 给出及优化了基于 Jasmin 混合编程的 Falcon 签名与验证实现.

(4) 将项目中遇到的问题和建议反馈给开发者, 促使其在 EasyCrypt 上游代码中对加和函数的运算引理进行更新, 使其更适配 Falcon 的证明.

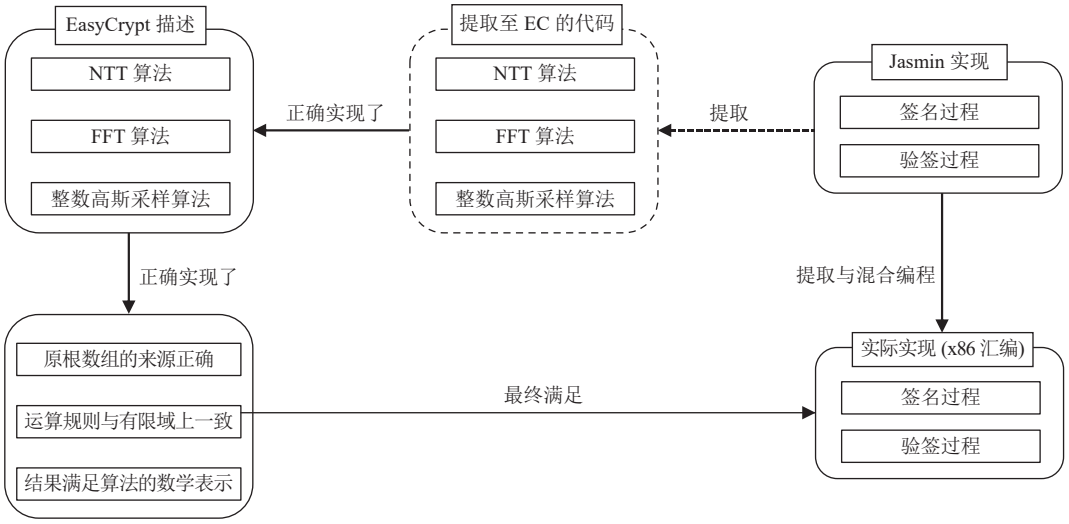

图 1　本文工作总体证明流程

这一形式化验证工作展示了实现高保证密码软件的可行性, 并为未来后量子密码算法的形式化验证提供了模板. 通过兼顾实现的效率和正确性, 我们的工作显著增强了 Falcon 在实际应用中的稳健性和可靠性.

本文第 2 节介绍与本文相关的预备知识. 第 3 节提出在 EasyCrypt 证明体系中验证的内容与关键方法或思路. 第 4 节详述用 Jasmin 框架实现 Falcon-512 的方式、难点以及生成实际代码 (x86 汇编) 的方法. 第 5 节中对生成的汇编代码进行效率测试, 并与原始代码的效率进行比较. 第 6 节进行总结与展望.

2　预备知识

2.1　Jasmin

Jasmin 是一种专为密码学领域设计的编程语言, 它巧妙地融合了高级语言的逻辑性和汇编语言的指令级控制. 这种独特的双重特性赋予开发者在编写高效紧凑、结构化代码的同时, 还能够精确掌控生成的汇编指令. Jasmin 的语法设计包含了结构化控制流元素, 例如循环和过程调用, 这不仅让代码更加简洁易懂, 还极大地简化了对安全性、功能正确性以及侧信道攻击防护的验证过程.

Jasmin 还是一个全面的形式化验证框架, 致力于推动快速且高度可靠的密码学软件的开发. 这个框架以 Jasmin 编程语言及其编译器为核心, 旨在增强程序的可移植性并简化验证任务. Jasmin 编译器专为输出可预测且高效的代码而设计, 目前支持 x64 平台, 并在 Coq 证明助理中进行了彻底的形式化验证.

Jasmin 编译器的高效代码生成是通过精心设计的编译过程实现的, 其中包括经过验证的编译步骤 (如函数内联、循环展开、常量传播) 以及翻译验证的过程 (如寄存器分配). 所有这些验证过程都在 Coq 中进行, 确保了编译器的准确性和生成代码的高效性. Jasmin 编译器本身也经过了验证, 涵盖了函数内联、循环展开和常量传播等关键过程, 并使用 Boogie[28]和 Z3 进行验证.

此外, Jasmin 框架还包括一系列高度自动化的工具, 这些工具能够证明内存安全性和恒定时间安全性, 从而防御基于缓存的时间攻击. 这些工具通过将 Jasmin 程序嵌入到 Dafny 验证工具[29]中, 生成验证条件并利用 Z3 求解器进行验证. 框架提供了一种将 Jasmin 程序嵌入 Dafny 的健全方法, 支持内存安全性、恒定时间安全性和函数正确性证明.

在性能方面, Jasmin 生成的代码与手工优化实现相当, 这一点通过使用 supercop 框架在代表性密码例程上的效率验证得到了证实. Jasmin 还开发了从 qhasm 到 Jasmin 的自动翻译器, 展示了 Jasmin 可以以类似 qhasm 的风格进行编程, 进一步证明了其实用性.

最后, Jasmin 利用 EasyCrypt 工具集进行形式化验证, 允许 Jasmin 程序被提取为相应的 EasyCrypt 程序, 以证明函数正确性、密码学安全性或对侧信道攻击 (常数时间) 的安全性. 通过这一全面的解决方案, Jasmin 为开发高速度、高保证的密码学软件提供了强有力的支持.

Jasmin 中的模 q 加法如图 2 所示, 其语言风格接近高级语言, 但变量类型仍保持汇编风格.

```
/*模q加法, 操作数需要在0..q-1的范围*/
inline fn mq_add(reg u32 x, reg u32 y) -> reg u32
{
    reg u32 d; //定义寄存器变量
    d = x + y - 12289; //q=12289
    d += 12289 & -(d >> 31); //使用移位实现模q加法
    return d;
}
```

图 2　模 q 加法的 Jasmin 实现

2.2 EasyCrypt

EasyCrypt 是一个先进的交互式框架, 专门用于在计算模型中验证密码构造的安全性. 它采用了一种基于代码的方法, 通过构建概率程序来模拟安全目标和困难假设. EasyCrypt 利用程序验证和编程语言理论的工具, 以严格的方式证明密码学推理的正确性.

EasyCrypt 支持一种基于游戏的方法, 这种方法将复杂性证明分解成一系列易于理解和验证的小步骤. 在每一步中, 都涉及两个程序, 而 EasyCrypt 通过概率关系性 Hoare 逻辑 (pRHL) 来推理这些程序之间的关系. pRHL 的判断形式为 $[c_1 \sim c_2 : \Phi \Rightarrow \Psi]$, 其中, c_1 和 c_2 是概率程序, Φ 和 Ψ 是关系性断言. 通过有效的 pRHL 判断, 可以推导出概率声明, 这有助于为不同游戏中的事件概率建立等式和不等式, 从而为密码学证明提供了一种清晰和系统化的验证途径.

除 pRHL 外, EasyCrypt 还有 3 种逻辑: 概率霍尔逻辑——允许进行关于过程执行概率导致后置条件成立的证明; 普通 (可能性) 霍尔逻辑 (HL); 用于证明一般数学事实和连接其他逻辑中的判断的环境高阶逻辑.

一旦引理被明确表述, 开发者就可以利用 EasyCrypt 中内置的公理和引理, 结合证明策略来进行证明. 这些证明策略体现了一般的推理原则, 它们是一系列逻辑规则, 能够将当前引理 (或目标) 转换为一个或多个子目标, 这些子目标是原始引理成立的充分条件. 对于简单的环境逻辑目标, 可以利用 SMT 求解器进行自动化证明.

证明过程可以被结构化为一系列引理, EasyCrypt 的理论框架允许将相关的类型、谓词、运算符、模块、公理和引理有机地组合在一起, 形成一个连贯的证明体系.

除此之外, EasyCrypt 还提供了一个模块系统, 这个系统支持组合推理, 允许开发者构建分层和模块化的密码系统证明. 这种模块化的方法对于处理涉及大量游戏跳跃和不同级别减少的复杂证明尤其有用, 它有助于将复杂的证明任务分解为更小、更易于管理的部分, 从而使得证明过程更加结构化和高效. 通过这种方式, EasyCrypt 为开发和验证复杂的密码学系统提供了强大的支持.

EasyCrypt 中模 q 加法的证明引理如后文图 3 中 lemma mq_add_corr 所示. 对于函数执行结果的性质证明, 一般采用霍尔逻辑 (HL). JNTT.M.mq_add 为 Jasmin 提取成的 EasyCrypt 程序, 箭头前的条件的含义是函数的参数要满足 $0 \leqslant x, y \leqslant q-1$, 即 Falcon 源代码中要求的参数范围. 箭头后的语句表示函数返回值 res (32 位无符号值) 转换成有符号整数 (to_sint) 后, 结果与定义好的模 q 加法的数学含义描述 op mq_add_spec 一致. 在“proof.”与“qed.”间为证明策略, 逐步证明原函数与目标函数结果一致.

2.3 Falcon 的签名与验签过程

Falcon 签名算法主要由 3 个重要部分组成: GPV 框架、NTRU 格以及快速傅里叶采样 [2].

GPV 框架是一种基于格的签名方案, 它概述了一系列严格的步骤, 遵循这些步骤可以确保获得安全的签名结果.

- 公钥: 由满秩矩阵 $\boldsymbol{A} \in \mathbf{Z}_q^{n \times m}$ 生成的 q 元格 $\boldsymbol{\Lambda}$.
- 私钥: 由矩阵 $\boldsymbol{B} \in \mathbf{Z}_q^{m \times m}$ 生成的 q 元格 $\boldsymbol{\Lambda}_q^{\perp}$, 即 $\boldsymbol{B} \times \boldsymbol{A}^t = 0$.

● 签名: 给定消息 m, 其签名为满足 $s\boldsymbol{A}^t = H(m)$ 的短向量 s, 其中, $s \in \mathbf{Z}_q^m$, $H: \{0,1\}^* \to \mathbf{Z}_q^n$ 为哈希函数. 注意, 在 GPV 框架中的短向量指的是签名向量 s 的欧几里得范数 $\|s\|$ 较小.

● 验签: 已知公钥 $\boldsymbol{A}$, 验证签名 s 是否有效需要保证两点: 一是需要保证 $s\boldsymbol{A}^t = H(m)$, 另一点要保证 s 确实是一个短向量.

```
/*模q加法在EasyCrypt端的函数式定义*/
op mq_add_spec (x y: int) =
    if x + y < q then x + y
    else x + y - q.

/*模q加法在EasyCrypt端的证明引理*/
lemma mq_add_corr x0 y0:
hoare [JNTT.M.mq_add: x=x0 /\ y=y0 /\ W32.zero \sle x0
/\ x0 \sle W32.of_int(q-1) /\ W32.zero \sle y0 /\ y0
\sle W32.of_int(q-1) ==> to_sint res = mq_add_spec
(to_sint x0) (to_sint y0)].
proof.
    proc; wp; skip => &hr [#] /= H H0 H1 H2 H3 H4.
    rewrite /mq_add_spec.
    case (to_sint x0 + to_sint y0 < q).
    rewrite H H0 -/q => H5.
    (*Omit other proof tactics.*)
qed.
```

图 3　模 q 加法的证明引理

Falcon 使用 NTRU 格与快速傅里叶采样实例化 GPV 框架, NTRU 格包括 4 个满足 NTRU 方程 $fG - gF = q \mod \phi$ 的多项式 f, g, F, G, 其中, $\phi = x^n + 1$, n 是 2 的整数幂, $q \in \mathbf{Z}$. 假设 f 在模 q 下是可逆的, 定义多项式 $h = g \cdot f^{-1} \bmod q$, 则 h 为公钥, f, g, F, G 为私钥, 具体实例化过程如下.

● 公钥: $\boldsymbol{A} = [1|h^\star]$.

● 私钥: $\boldsymbol{B} = \begin{pmatrix} g & -f \\ G & -F \end{pmatrix}$, 可以验证 $\boldsymbol{B} \times \boldsymbol{A}^\star = 0 \mod q$.

● 签名: 对消息 m 的签名包括随机盐值 r 与通过快速傅里叶采样生成的一对多项式 (s_1, s_2), 满足 $s_1 + s_2 h = H(r\|m)$, 并将 (r, s_2) 作为签名.

● 验签: 在验证过程中, 与签名过程相同, 首先利用收到的盐值 r 和消息 m 重新计算哈希值. 然后, 使用公钥 h 计算 $s_1 = c - s_2 \cdot h \bmod q$. 接着, 检查 (s_1, s_2) 是否为一个短向量, 具体来说, 验证 $(s_1, s_2) \leqslant \beta$ 是否成立, 其中, β 是一个预定的阈值.

通过这种方式, Falcon 实现了高效且安全的签名和验证过程. 其基于 NTRU 格的结构和快速傅里叶采样技术, 使其在后量子密码学中具有显著优势. Falcon 的设计确保了在保持高效性的同时, 能够提供强大的安全保证, 适用于未来量子计算环境下的密码学应用.

3　验证函数实现的正确性

在 Falcon 签名方案的签名和验证过程中, 快速傅里叶采样扮演着至关重要的角色. Falcon 算法立足于格的短向量问题, 并涉及多项式乘法, 其中, 拉格朗日插值是关键步骤. 通过 FFT 与 NTT, 多项式乘法可以高效地转换为点值形式的乘法, 显著提升了签名生成和验证的效率. Falcon 签名方案还依赖于基于格的离散高斯采样技术, 以生成满足特定条件的签名向量. 这种高斯采样技术旨在从特定的高斯分布中抽取出符合条件的签名向量, 进而在 Falcon 中解决最近向量问题 (CVP), 即在格基上寻找一个接近目标点的向量.

正是这些核心操作的高效实现, 使得 Falcon 算法在签名生成和验证方面表现出色. 为了确保这些算法的可靠性, 本文特别针对 Falcon 签名和验签过程[2]中的 Montgomery 模乘、NTT、FFT 算法以及整数高斯采样算法中的部分关键函数的正确性进行了验证. 这些算法不仅是构建 Falcon 签名方案的基石, 也具有深远的数学意义和价值.

3.1　验证 Montgomery 模乘

我们首先验证了在 Falcon 中高效计算模乘的 Montgomery 模乘法. Montgomery 模乘法特别适合大整数运算

和密码学应用, 它的基本思想是将模乘运算转换为更简单的加法和移位操作, 以避免直接进行高开销的除法运算, Falcon 中的 Montgomery 模乘算法如算法 1 所示.

算法 1. Falcon 中的 Montgomery 模乘算法.

输入: 有限域 $\mathbf{Z}_q$ 上的元素 x、y, q, q^{-1};
输出: z, 满足 $z = x \cdot y \cdot R^{-1} \bmod q$, 其中, $R = 2^{16}$.

1. $z \leftarrow x \cdot y$;
2. $w \leftarrow ((z \cdot q^{-1}) \& 0\text{xFFFF}) \cdot q$;
3. $z \leftarrow ((z + w) \gg 16) - q$;
4. IF $z < 0$ THEN
5. $\quad z \leftarrow z + q$;
6. END IF
7. RETURN z

注意到, 其对运算做了更符合机器运算的优化. 而在 EasyCrypt 中, 我们定义的 Montgomery 模乘的数学表示则更为直观, 更符合数学逻辑, 如算法 2 所示.

算法 2. EasyCrypt 中 Montgomery 模乘算法的数学表示.

输入: 有限域 $\mathbf{Z}_q$ 上的元素 x、y, q, q^{-1};
输出: z, 满足 $z = x \cdot y \cdot R^{-1} \bmod q$, 其中, $R = 2^{16}$.

1. $z \leftarrow x \cdot y$;
2. $m \leftarrow ((z \bmod R) \cdot q^{-1}) \bmod R$;
3. $z \leftarrow (z + m \cdot q)/R$;
4. IF $z \geqslant q$ THEN
5. $\quad z \leftarrow z - q$;
6. END IF
7. RETURN z

我们的目标即是证明二者之间的等价性, 即两种算法对于相同的参数返回的结果相等.

比较两个算法, 可以观察到二者在第 2 行有所不同, Falcon 中使用了与运算, 而数学描述则是取模. 这就是证明的关键引理之一, 即有限域 $\mathbf{Z}_q$ 上的取模运算 ($\bmod R$) 与实际实现中的与运算 (&0xFFFF) 等价. 而在第 3 行, 前者使用了移位运算, 后者则是除法运算. 类似地, 我们需要证明有限域 $\mathbf{Z}_q$ 上的除法运算 ($/R$) 与实际实现中的移位运算 ($\gg 16$) 等价. 二者均可使用其内置引理证明.

除此之外, 由于实际实现的变量类型为计算机类型 (32 位或 64 位无符号数等), 而数学描述所定义的变量类型为 $\mathbf{Z}_q$ 上的元素, 对于其他运算, 如加法、减法, 我们都需要做等价性证明, 如图 3 所示. 对于它们返回的结果, 由于数据类型不同, 我们也要做相应的转换, 将计算机类型转换成整数, 以进行比较.

最后, 对于条件分支, 我们需要对每种可能都分别讨论, 即使用策略“case (q <= REDC′(to_sint (x0 * y0)))”对中间结果与 q 的大小进行分情况讨论.

将上述步骤组合起来, 我们最终得到了两者之间的等价性证明.

3.2 验证 NTT 算法

NTT 是 FFT 在有限域 $\mathbf{Z}_q$ 上的类比, 其中, q 是一个满足 $q \equiv 1 \bmod 2n$ 的素数 (在 Falcon 中, $q = 12289 = 1 + 12 \times 1024$). 在这些条件下, 多项式 $\varphi = x^n + 1$ 在 $\mathbf{Z}_q$ 上恰好有 n 个根 ω_i, 任何多项式 $f \in \mathbf{Z}_q[x]/(\varphi)$ 都可以表示为这些

根的值 $f(\omega_i)$. 在 Falcon 中, NTT 与逆 NTT 算法是其主要计算负载[30].

多项式的系数表示和 NTT 表示可以在 $O(n\log n)$ 的运算时间内高效完成相互转换. 多项式的加法, 减法, 乘法和除法 (模 φ 和 q) 都可以用 NTT 表示进行. 通过使用 NTT 表示, 多个多项式运算 (如加法、减法、乘法和除法) 可以在每个坐标上独立地进行, 从而大大提高了计算效率.

在 Falcon 签名方案中, 数论变换 (NTT) 用于多个关键方面. 首先, 在公钥生成过程中, NTT 可以快速实现多项式的乘法和除法, 从而高效计算出公钥 $h = g/f \mod \phi \mod q$. 此外, NTT 在多项式运算中允许每个坐标独立处理, 显著提高了整体计算效率. NTT 的使用使得 Falcon 在保持高效性的同时, 提供了强大的安全保证, 使其在后量子密码学中具有显著优势.

Falcon 中的 NTT 算法如算法 3 所示.

算法 3. NTT 算法.

输入: 多项式的系数数组 a, 原根数组 GMb;
输出: 经由 NTT 变换后的数组 a.

```
1.  t ← N;
2.  FOR m ← 1 TO N DO
3.      ht ← t/2;
4.      FOR i1 ← 0 TO m DO
5.          j1 ← i1 · t;
6.          s ← GMb[m + i1];
7.          FOR j ← j1 TO j1 + ht DO
8.              x ← f[j];
9.              y ← s · f[j + ht];
10.             f[j] ← x + y;
11.             f[j + ht] ← x − y;
12.         END FOR
13.     END FOR
14.     t ← ht;
15.     m ← m × 2;
16. END FOR
```

完整的 NTT 算法包括 ntt 与其逆函数 intt, 本文所述证明均包含二者, 以下表述省略 intt.

3.2.1 证明流程与概述

NTT 的证明流程如图 4 所示.

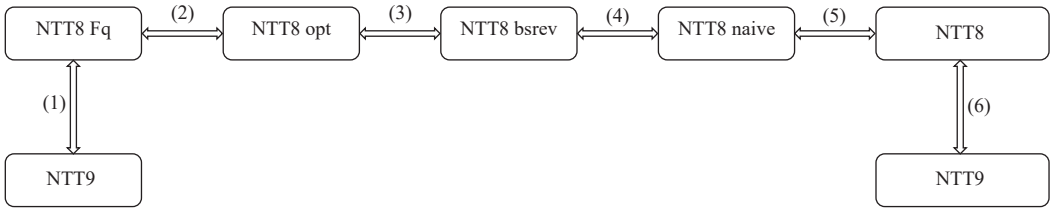

图 4 Falcon 中 NTT 的证明流程

本文验证的 Falcon-512 中的 NTT 的定义多项式则为 $\varphi = X^{512} + 1$, 对多项式 f 进行 NTT 运算可表示为:

$$\mathrm{NTT}_{\varphi}(f)=\sum_{j=0}^{511} f_j \omega^{(2\mathrm{br}_9(i)+1)j}.$$

在 Falcon-512 中, NTT 蝴蝶变换的层数为 9 层.

我们首先证明了 8 层 NTT 蝴蝶变换的正确性, 其公式表示为:

$$\mathrm{NTT}_8(f)=\hat{f}=\hat{f}_0+\hat{f}_1 X+\cdots+\hat{f}_{511}X^{511}.$$

其中,

$$\hat{f}_{2i}=\sum_{j=0}^{255} f_{2j}\omega^{(2\mathrm{br}_8(i)+1)j},$$

$$\hat{f}_{2i+1}=\sum_{j=0}^{255} f_{2j+1}\omega^{(2\mathrm{br}_8(i)+1)j}.$$

NTT8 Fq 将 8 层蝴蝶变换的数据类型由计算机类型 (32 位或 64 位无符号数等) 转换为数学类型 (定义在 $\mathbf{Z}_q$ 上). NTT8 specification 即为上述公式的函数式编程代码. NTT8 opt、NTT8 bsrev、NTT8 naive 则为证明 NTT8 Fq 与 NTT8 specification 等价的中间形式.

完成对 8 层 NTT 蝴蝶变换的正确性证明后, 我们只需在图 4 中的 (1) 处修改 Jasmin 中的 NTT 代码, 在 8 层 NTT 之后加上最后一层, 即为 9 层 NTT 源代码形式 NTT9 Jasmin. 同时, 在图 4 中的 (6) 处调用函数 ntt_8_to_9 (将 8 层 NTT 乘上蝴蝶变换最后一层的单位根, 变为 9 层), 并证明调用后的公式等价于 $\mathrm{NTT}_{\varphi}(f)$, 这样就最终证明了 NTT9 Jasmin 与 NTT9 specification 之间的等价性.

3.2.2 证明关键内容与方法

对于图 4 中的 (1), 首先要证明计算机类型之间的运算与 $\mathbf{Z}_q$ 上的运算 (包括域上的取模运算、加法、减法、除法、Montgomery 模乘) 结果相等. 证明第 1 步为定义运算在域上的数学含义, 之后对 Jasmin 所提取的函数建立条件与结论, 形成引理, 如图 3 所示. 证明策略包括将函数展开、分情况比较、类型转换、证明移位的性质等, 如第 3.1 节所述.

其次, 需要证明函数 ntt_9 是无损的 (lossless), 即满足下面的概率霍尔逻辑:

$$\mathrm{Phoare}[\mathrm{ntt\_9}: \mathrm{true} \Rightarrow \mathrm{true}] = 1\%\ r.$$

这个引理的含义是无论函数输入何种参数, 均会以概率 1 终止. 因此, 证明该逻辑只需证明 ntt_9 的 3 层嵌套循环会终止. 如算法 3 所示, 嵌套循环中的循环变量有依赖关系, 我们需要在证明中的 while 策略 (while 策略可用于证明循环会终止) 中指明 $m\cdot t=512$, 以及 m, t 均为 2 的幂次, 这样即可证明 m, i_1, j 均可达到某一上限, 并最终跳出循环.

最后, 需要证明原根数组来源的正确性, 即代码中所提供的原根数组的值确实计算自原根的幂次, 并变换成 Montgomery 形式, 即 $zetas2[i]=\omega^{2br_9(2i)+1}$. 因为预置数组中的值的确来源合法, 使用 EasyCrypt 的计算功能证明即可. 此外, *zetas*2 数组经过顺序变换与 Montgomery 变换后最终形成 GMb 数组.

对于图 4 中的 (2)–(5), 我们需要证明几个中间形式的等价性, 具体方法如下.

首先, 为了便于整体证明, 我们将算法 3 中的循环顺序进行修改, 得到计算 8 层 NTT 的代码, 其伪代码如算法 4 所示.

算法 4. NTT8 opt 算法.

输入: 多项式的系数数组 *r*, 原根数组 *zetas*;
输出: 经由 NTT 变换后的数组 *r*.

1. *zetasctr* ← 0;
2. *len* ← 256;
3. WHILE *len* ⩽ 2 DO
4. *start* ← 0;

```
5.    WHILE start < 512 DO
6.      zetasctr ← zetasctr + 1;
7.      zeta_ ← zetas[zetasctr];
8.      j ← start;
9.      WHILE j < start + len DO
10.       t ← fqmul(zeta_, r[j + len]);
11.       r[j + len] ← r[j] − t;
12.       r[j] ← r[j] + t;
13.       j ← j + 1;
14.     END WHILE
15.   start = j + len;
16.   END WHILE
17.   len ← len/2;
18. END WHILE
```

注意, 实际上, 在 EasyCrypt 中, NTT8 opt 是 NTT 的 3 个嵌套循环拆解组合的形式, 即分别将每个循环写为一个独立的函数, 这样做可以分别对每个循环进行推理.

NTT8 bsrev 如算法 5 所示. NTT8 bsrev 将用于访问单位根数组的索引的增量计算替换为根据循环索引计算它的值, 并将最内层循环的索引变量在最内层循环结束后替换为其最终值, 其与 NTT8 opt 的等价性显而易见.

算法 5. NTT8 bsrev 算法.

输入: 多项式的系数数组 r, 原根数组 *zetas*;
输出: 经由 NTT 变换后的数组 r.

```
1.  len ← 256;
2.  WHILE len ⩽ 2 DO
3.    start ← 0;
4.    WHILE start < 512 DO
5.      zetasctr ← (512 + start) / (len × 2); // start 每次增加 2 × len, zetasctr 每次增加 1, 此表达式是为了满足初值
6.      zeta_ ← zetas[zetasctr];
7.      j ← start;
8.      WHILE j < start + len DO
9.        t ← fqmul(zeta_, r[j + len]);
10.       r[j + len] ← r[j] − t;
11.       r[j] ← r[j] + t;
12.       j ← j + 1;
13.     END WHILE
14.   start = start + len × 2; // j 初始值为 start, 在内层循环中 j 增加了 len
15.   END WHILE
16.   len ← len/2;
17. END WHILE
```

NTT8 naive 如算法 6 所示. NTT8 naive 与 NTT8 bsrev 的主要区别在于它们最内层循环对数组 r 中元素的计

算顺序不同, 接下来分析其等价性.

算法 6. NTT8 naive 算法.

输入: 多项式的系数数组 r, 原根数组 *zetas*;
输出: 经由 NTT 变换后的数组 r.

1. $len \leftarrow 1$;
2. WHILE $len \leqslant 128$ DO
3. $start \leftarrow 0$;
4. WHILE $start < len$ DO
5. $zetasctr \leftarrow br_9((start \times 2+1) \times (256/len))$;
6. $zeta\_ \leftarrow zetas[zetasctr]$;
7. $j \leftarrow 0$;
8. WHILE $j < 512$ DO
9. $t \leftarrow \text{fqmul}(zeta\_, r[br_9(j+len+start)])$;
10. $r[br_9(j+len+start)] \leftarrow r[br_9(j+start)] - t$; // br_9 为 9 位比特翻转函数
11. $r[br_9(j+start)] \leftarrow r[br_9(j+start)] + t$;
12. $j \leftarrow j + len \times 2$;
13. END WHILE
14. $start = start + 1$;
15. END WHILE
16. $len \leftarrow len \times 2$;
17. END WHILE

首先, 尽管这两种方式循环方向不同, 但它们都在处理相同的层级和块大小. 对于每一层, NTT8 bsrev 中 *len* 的值与 NTT8 naive 的 *len* 在逻辑上是反向的. 即, NTT8 bsrev 从大的 *len* 处理块开始, 而 NTT8 naive 从小的 *len* 块开始, 但两者最终遍历相同的处理块.

其次, 在 NTT8 bsrev 中, *start* 每次递 $len \times 2$ 而 j 在每次内层循环中递增 1, 表示每次处理一个蝶形块中的元素对; 在 NTT8 naive 中, *start* 每次递增 1, 而 j 在每次内层循环中递增 $len \times 2$, 即跳过 *len* 个元素, 进行下一个蝶形块的运算. 本质上, 两者在操作的对象 (数组中的元素) 和处理块是相同的, 只是控制变量的顺序不同.

最后, 在 NTT8 bsrev 中, *zetasctr* 通过公式 $(512+start)/(len \times 2)$ 计算, 在每一轮蝶形运算中, 它选择适当的原根系数 *zeta*; 而在 NTT8 naive 中, *zetasctr* 由公式 $br_9((start \times 2+1) \times (256/len))$ 计算. 两者的原根选择方式看似不同, 但实质上, 它们都在根据相同的比特翻转规则选择原根 *zeta*, 所以 *zetasctr* 在逻辑上是等价的.

在实际的 EasyCrypt 证明过程中, 我们通过对变量进行追踪分析来证明它们在每轮蝶形运算中选取的是相同的数组位置和原根值.

算法 7 在最内层循环中被算法 8 调用, 能够直接计算出目标数组每个元素的最终值, 而前述算法的计算顺序则不同. 这样做便于与公式的最终形式进行等价性证明. 对于其与 NTT8 naive 的等价性证明, 我们同样采用变量追踪的方式.

算法 7. partial_ntt 算法.

输入: 多项式的系数数组 p, 整数 *len*, *start*, *bsj*;
输出: NTT 变换的中间值 *sum*.

1. $sum \leftarrow 0$;
2. FOR $s \leftarrow 0$ TO len DO
3. $\quad index \leftarrow br_9(bsj \times len + s)$;
4. $\quad value \leftarrow \exp(\omega, (2 \times start + 1) \times br_9((s \mod len) \times 2)) \times p[index]$;
5. $\quad sum \leftarrow sum + value$;
6. END FOR
7. RETURN sum;

算法 8. ntt_spec 算法.

输入: 多项式的系数数组 p;
输出: 经由 NTT 变换后的数组 r.

1. FOR $start \leftarrow 0$ TO 256 DO
2. $\quad$ FOR $bsj \leftarrow 0$ TO 2 DO
3. $\quad\quad index \leftarrow br_9(bsj \times 256 + start)$;
4. $\quad\quad r[index] \leftarrow \text{partial\_ntt}(p, 256, start, bsj)$;
5. $\quad$ END FOR
6. END FOR

最后, 比较算法 8 与 $\hat{f}_{2i}$ 和 $\hat{f}_{2i+1}$, 可以发现其基本形式是一致的, 只需证明 for 循环与累加符号的等价性即可.

对于图 4 中的 (6), 即证明将 8 层 NTT 变为 9 层时的正确性, 我们需要证明的前置引理包括原根周期性, 即 $\omega^{1024} = 1$(注意到, 为了更好地适配 Falcon-512 与 Falcon-1024, Falcon 原根数组中存储的值实际上是其 2 048 次原根 $\omega' = 7$ 生成, 而在 Falcon-512 中, 我们使用 1 024 次原根 $\omega = 49$); 还需要证明比特反转函数 (BitReverse) 由 8 位转为 9 位的变化关系, 例如, 对于所有的整数 n, 如果 n 满足 $0 \leqslant n < 512$, 且 $n \bmod 2 = 0$, 则有 $\text{bsrev}_9(n) = \text{bsrev}_8\left(\frac{n}{2}\right)$; 以及累加函数奇偶相合的推导, 即:

$$\sum_{i=0}^{n-1}[F(2i) + F(2i+1)] = \sum_{j=0}^{2n-1} F(j).$$

上述引理均易于手写证明, 利用内置引理, 我们可以轻易地将其代码化.

通过上述引理, 可以证明以下公式:

$$\text{ntt}_9(p)[i] = \begin{cases} \text{ntt}_8(p)[i] + \text{ntt}_8(p)[i+1] \cdot \omega^{2br_9(i)+1}, & i \mod 2 = 0 \\ \text{ntt}_8(p)[i-1] + \text{ntt}_8(p)[i] \cdot \omega^{2br_9(i-1)+1+512}, & i \mod 2 = 1 \end{cases}.$$

此外, 我们还证明了 NTT 的可逆性 (即 ntt 与 intt 可以相互抵消) 以及 NTT 的加法同态与乘法同态.

我们首先证明几何级数的性质, 即当几何级数的基数 z 大于 0 时, 其在区间 $[a,b]$ 上的和为:

$$S(z,a,b) = \begin{cases} 0, & \text{if } b < a \\ b - a, & \text{if } z = 1 \\ \dfrac{z^b - z^a}{z-1}, & \text{otherwise} \end{cases}.$$

多项式系数数组 p 经过 9 层 NTT 变换后, 所得数组 p' 中的元素 $p'[i]$ 可表示为 $p'[i] = \sum_{j=0}^{511} p[j] \times \omega^{(2br_9(i)+1)j}$, 逆 NTT 变换可以表示为 $p[i] = \frac{1}{512}\sum_{j=0}^{511} p'[j] \times \omega^{-(2br_9(j)+1)i}$.

将 NTT 的结果 $p'[j]$ 代入逆 NTT, 可以得到:

$$p[i] = \frac{1}{512}\sum_{j=0}^{511}\left(\sum_{k=0}^{511} p[k] \times \omega^{(2br_9(j)+1)k}\right) \times \omega^{-(2br_9(j)+1)i}.$$

将单位根部分组合, 得到:

$$p[i]=\frac{1}{512}\sum_{j=0}^{511}\sum_{k=0}^{511}p[k]\times\omega^{(2br_9(j)+1)(k-i)}.$$

利用单位根的正交性性质:

$$\sum_{j=0}^{511}\omega^{(2br_9(j)+1)(k-i)}=\begin{cases}512, & k=i\\ 0, & k\neq i\end{cases}.$$

可证等式成立.

实际证明过程调用了 EasyCrypt 实数、整数、环、域、比特运算、累加函数等库的大量内置引理, 对无法直接使用 SMT 求解的目标, 我们化整为零, 先作为前置引理进行证明, 再在最终目标处调用, 最终形成完整且完备的证明. 这些繁杂的工作使得我们最终可以桥接计算机代码与数学表示之间的巨大鸿沟.

3.3 验证 FFT 算法

在私钥操作和签名生成过程中, FFT 用于快速傅里叶采样. 这个过程需要频繁进行多项式的乘法和除法运算, 通过 FFT, Falcon 可以将这些运算转换到为点值进行, 从而减少计算量.

Falcon 中的 FFT 算法与 NTT 算法的整体结构相似, 不同之处在于原根 ζ 为 1 的 $2N$ 次单位根 $e^{i\pi/N}$. 值得注意的是, 由于原根与经过 FFT 变换后的值均为复数, 其实部与虚部需要在数组中以 64 位无符号类型分别存储. 如此一来, 数组中存储的值数目减半, 但是由于原根的周期性: $\zeta^{N-1-j}=\mathrm{conj}(\zeta^{2j+1})$, 其中, conj 表示复数的共轭, 最终缺少的一半数值也可以被计算出来, 这使得存储空间需求减少了一半.

由于 FFT 算法与 NTT 算法的高度相似性, 我们不需再做冗余的证明, 只需对原根数组的数值来源的正确性做证明即可.

3.4 验证整数高斯采样算法

在 Falcon 签名方案中, 整数高斯采样算法是实现安全和高效签名生成的关键部分. 具体来说, 涉及 3 个重要的采样器: SamplerZ、BaseSampler 和 BerExp.

SamplerZ 函数负责在给定均值和标准差的情况下, 生成一个接近高斯分布的整数样本. 其调用 BaseSampler 与 BerExp, 前者用于生成基本的样本 z_0, 根据预定义的分布 χ, 输出一个 0–18 之间的整数 z_0; 后者用于执行拒绝采样, 以近似概率 $ccs\times\exp(-x)$ 返回一个比特值 1.

本文证明了 BaseSampler 函数 (如算法 9 所示) 中的判断函数 $[\![u<\mathrm{RCDT}[i]]\!]$ 的正确性, 即如果双括号中的判断条件成立, 则返回 1, 否则返回 0. 其余运算多是对算法描述的直译, 故不作正确性证明.

算法 9. BaseSampler.

输入: 一;
输出: 整数 $z_0\in\{0,\ldots,18\}$, 且 $z\sim\chi$.

1. $u\leftarrow UinformBits(72)$;
2. $z_0\leftarrow 0$;
3. FOR $i\leftarrow 0$ TO 17 DO
4. $\quad z_0\leftarrow z_0+[\![u<\mathrm{RCDT}[i]]\!]$;
5. END FOR
6. RETURN z_0;

该函数为了保证精度, 将预计算的反向累积分布表 (RCDT) 的数值设置为 72 位, 随后将 72 位整数拆分为 3 个 24 位 (存储为 32 位) 存储在 w 数组中. 同时, 使用基于 SHAKE-256 的伪随机数生成器生成了一个 64 位伪随机数 lo 与一个 8 位伪随机数 hi, 并将组合成的 72 位伪随机数拆分为 3 个 24 位数, 每次与 RCDT 表中的 3 项进行比

较, 每项对应相减并设置进位, 如算法 10 所示. 我们将这个比较过程提取为 EasyCrypt 中的函数 cmp_24_merge, 并用两个引理证明了 $[\![u < \mathrm{RCDT}[i]]\!]$ 的正确性, 即当在 $v < w$ 时, 函数返回 1; 在 $w \leqslant v$ 时, 函数返回 0. 值得注意的是, 我们除了设置了必要的参数与 w 和 v 的大小比较, 还对 $\mathrm{to\_uint}(w_i + 1)$ 与 $\mathrm{to\_uint}\, v_i$ 的上限做了规定, 即 $w_i + 1$ 与 v_i 的数值必须是非负数 (小于 2^{31}), 以便排除负数移位的情况, 这显然成立, 因为它们实际上都是由 24 位数转换而来.

算法 10. Compare.

输入: 随机数 *lo*、*hi*, *GMb* 数组中的 3 个元素 w_0、w_1、w_2;
输出: 大小判断结果比特.

1. $v_0 \leftarrow lo$&0xFFFFFF;
2. $v_1 \leftarrow (lo >> 24)$&0xFFFFFF;
3. $v_2 \leftarrow (lo >> 48)|(hi << 16)$;
4. $cc \leftarrow (v_0 - w_0) >> 31$;
5. $cc \leftarrow (v_1 - w_1 - cc) >> 31$;
6. $cc \leftarrow (v_2 - w_2 - cc) >> 31$;
7. RETURN *cc*;

我们通过证明位运算的数学含义和进行详细的分类讨论来完成证明. 例如, 我们证明了对 24 位全 1 数 0xFFFFFF 进行按位与运算, 相当于对 2^{24} 取模. 详细的分类讨论, 例如对将 v 与 w 转换为无符号数后大小的讨论, 可以分为 3 种情况, 即分别对 v 与 w 的高中低 24 比特进行比较, 证明内容如图 5 所示.

```
op case1 (v w:W128.t) = (to_uint (split_128 v).`3 < to_uint (split_128
w).`3).
op case2 (v w:W128.t) = (to_uint (split_128 v).`3 = to_uint (split_128
w).`3) /\ (to_uint (split_128 v).`2 < to_uint (split_128 w).`2).
op case3(v w:W128.t) = (to_uint (split_128 v).`3 = to_uint (split_128
w).`3) /\ (to_uint (split_128 v).`2 = to_uint (split_128 w).`2) /\ (to_uint
(split_128 v).`1 < to_uint (split_128 w).`1).

lemma split_cmp(v w:W128.t):
    to_uint (v `&` (W128.of_int 4722366482869645213695)) < to_uint (w `&`
(W128.of_int 4722366482869645213695)) => case1 v w \/ case2 v w \/ case3
v w.
```

图 5　大小比较证明引理

本文对整数高斯采样算法的概率分布的性质证明研究比较初步, 仅提出证明思路, 具体证明代码有待后续完善.

验证高斯采样正确性的核心在于确立实际采样中得到的分布和理论模型中假定的高斯分布之间的等价性, 具体而言, 我们的目标是证明算法 9 所生成的输出分布与均值 $\mu = 0$, 标准差 $\sigma = 1.8205$ 的离散高斯分布 χ 匹配, 其中, χ 由以下公式给出:

$$P(X = k) = \frac{1}{\mathrm{Z}} \cdot \mathrm{e}^{-\frac{k^2}{2\sigma^2}},$$

其中, Z 是归一化常数, 定义为所有可能的 k 值的概率和, 用于确保总概率和为 1:

$$\mathrm{Z} = \sum_{k=0}^{18} \mathrm{e}^{-\frac{(k-\mu)^2}{2\sigma^2}}.$$

为了验证等价性, 我们需要计算理想分布和实际分布之间的汉明距离. 一般而言, 只要汉明距离之和小于预设的上限, 我们可以认为二者的分布是等价的. 对于算法 9, 由于只有 19 种可能的情况, 我们可以分别计算出每种情况的理想概率和算法 9 决定的实际产生的概率, 相减得出汉明距离. 通过这种方法, 我们能够定量地评估算法 9 产生的分布与理论分布之间的一致性, 从而验证高斯采样的正确性.

4 使用 Jasmin 体系实现 Falcon 算法

在本节中, 我们使用 Jasmin 体系实现 Falcon 算法的签名及验签过程.

首先, 我们使用 Jasmin 语言实现. Jasmin 语言与 Falcon 的 C 语言代码整体相似, 但对于 Jasmin 当前版本不支持或语法不同的部分, 如结构体, 指针, 递归函数, 循环等, 我们进行了必要的修改和调整. 同时, 我们尝试生成汇编代码. 对于密钥生成过程, 由于其大量使用指针算术访问数组元素, 我们难以显式指定每次数组访问的起始位置和长度, 并且难以对每个包含可变长度数组参数的函数, 为每一种长度都重定义一遍函数, 故并未实现此过程. 我们期待 Jasmin 后续版本对指针算术访问数组元素有更好的支持.

4.1 自定义浮点类型

由于 Jasmin 不支持浮点类型, 我们采用了 Falcon 官方代码[2]中使用的 IEEE-754 “binary64”格式来自定义浮点类型和浮点数运算的方法. 与 C 语言不同的是, Jasmin 也不支持有符号类型, 因此, 我们对涉及有符号数的大小比较和类型转换进行了谨慎处理, 当有符号数的低位类型转为高位类型时, 我们先判断高位是否为 1, 然后再填充 1 或 0.

4.2 采样, 签名与验签

采样过程基本与 Falcon 官方的 C 语言代码相同, 而签名与验签过程则需进行大量修改.

首先, 我们要对结构体与指针这些复合数据类型进行改写. Jasmin 语言支持返回多个值, 因此我们可以将结构体解构, 并直接使用指针所指向的值, 最后在使用了这些数据类型的函数的参数或返回值处添加相应变量. 其次, Jasmin 也不支持递归函数, 我们另写函数, 将递归函数自调用改为调用其他函数. 最后, 由于递归函数涉及数组, 并且其每次自调用所用的数组长度均减半, C 代码中使用指针算术进行访问, 而在 Jasmin 中我们改为使用其支持的语法, 即数组切片进行处理. 相较于指针算术, 数组切片的灵活性较差, 并且需要显式指定切片的长度和起始位置, 需要对此进行推理.

4.3 生成实际实现代码 (x86_64 汇编)

实际上, 可以在 Jasmin 的函数前加上 inline 关键字或 export 关键字, inline 函数可用于执行或提取至 EasyCrypt, export 函数可用于生成汇编代码. 然而, export 函数支持的语法更少, 为了使其生成汇编代码, 我们要对代码进行低级化, 使其更加接近底层硬件级别的操作, 以确保生成的汇编代码能够正确执行.

在对代码低级化的过程中, 我们遇到了许多问题, 经过自行探索和同开发人员交流, 我们成功解决了其中的大部分.

我们遇到的一个最主要的问题是寄存器分配冲突与数目不足. Jasmin 中只有两种存储类型, 寄存器类型 (reg) 与栈类型 (stack), 均需先声明再赋值. 在 Jasmin 中, 函数中被分配到寄存器中的入参类似于 C 语言中的形参. 如果这些入参不在函数体中再次声明一次, 那么它们将被视为局部不可修改变量, 这种方式会增加寄存器的消耗. 考虑到 x86_64 架构中寄存器资源的稀缺性, 在程序规模较大时, 建议优先使用栈元素存储变量. 然而, 由于 Jasmin 本身的限制, 栈元素之间无法直接进行运算 (例如算数运算或寻址操作). 为了解决这个问题, 可以利用 Jasmin 另一特性——支持栈元素和寄存器元素之间的运算, 在函数体中定义寄存器变量来辅助栈元素进行这些运算. 这样既可以节省寄存器资源, 又能实现所需的运算操作.

另一个需要注意的问题是, RCX 寄存器在存储第 4 个参数和执行移位操作中存在潜在冲突. 在函数定义中, 若存在第 4 个参数且函数体中有寄存器移位操作, 调用函数和执行移位操作的指令会因为争夺 RCX 寄存器的使用权而引发寄存器冲突. 为了规避这一问题, 需要在定义变量时为它创建一个副本, 并在后续代码中使用这个副本来代替直接使用第 4 个参数. 否则, 会导致生成可执行的汇编代码失败.

此外, Jasmin 的函数调用采用内联方式, 因此没有像 C 语言中函数调用的出栈入栈过程. 在 Jasmin 中, 每个 while 循环至少需要占用一个寄存器, 以确保循环能够正常运行. 如果循环过多, 会导致寄存器资源消耗严重. 我们咨询了 Jasmin 的开发者, 核心思路是确保有栈元素可以在 while 循环开始前存入 while 循环中占用的寄存器值,

寄存器在 while 循环结束前取出栈存储的值.

在开发过程中, 当程序规模较大且函数调用层级较深时, 容易导致寄存器资源紧张. 然而, 将所有元素都存储在栈中也并非理想选择. do_sign 函数由于其调用层级过深, 导致生成可执行汇编代码速度显著下降. 为了解决此问题, 我们选择 C 与汇编的混合编程方式来实现 do_sign 函数. Vrfy 函数的调用深度较浅, 可以直接生成汇编代码. 在 Jasmin 中, 如果需要导出参数包含数组的函数, 需要为该函数额外编写一个包装器, 以便将数组的首地址传递给包装器, 从而方便后续的寻址. 这种设计思路基于 Jasmin 对 export 函数参数的限制, 即其参数必须是寄存器元素且不能是数组.

为了验证结果, 我们对比了汇编与 C 的结果以及 Jasmin 中利用 exec 指令计算的结果, 以确保它们一致.

在生成采样相关汇编文件时, 我们需要在终端中修改栈空间的大小, 否则会导致生成失败. 我们建议栈空间的大小为 30 720 KB.

5 效率测试

在本节中, 我们对由 Jasmin 生成的 Falcon 签名与验签函数的汇编代码进行了基准测试, 并与原始代码的效率进行比较, 进行基准测试所使用的操作系统为 Ubuntu 20.04.6 LTS, 系统内存为 9.7 GiB, 中央处理器为 12th Gen Intel® Core™ i7-12700 @ 2.10 GHz ×4, 编译器为 GCC 9.4.0. 表 1 展示了基准测试的结果.

表 1 C 和 Jasmin 生成的汇编代码的周期计数

实现	函数	周期计数
C_{ref}	sign_dyn	11 392 347
	verify_raw	62 454
$Jasmin_{ref}$	sign_dyn	25 430 571
	verify_raw	27 154

表 1 中的周期计数都是具有相同输入的 10 000 次运行中取得的中位数. 可以观察到 sign_dyn 函数的 Jasmin 参考实现比 C 参考实现要慢得多, 这在意料之中, 因为 Jasmin 编译器不会优化代码, 这是它与 C 编译器不同之处, 不过开发人员可以在后续工作中对代码进行优化设计. 我们的实现目标并不仅仅包括追求高性能, 同时也要为 EasyCrypt 的正确性证明提供便利.

值得注意的是, verify_raw 函数的 Jasmin 的参考实现比 C 的参考实现快了近 2.3 倍, 这有些出人意料. 在第 3.3 节, 我们提到, 当程序规模较大且调用层级较深会导致寄存器资源紧张, 因此我们通常会将大量元素存在栈中. 但是, 与 sign_dyn 相比, verify_raw 的程序规模非常小, 所有元素都可以存放在寄存器中. 此外, Jasmin 参考实现中所有函数均为内联函数, 并且在开发时我们对部分操作进行了合并优化, 这使得相比于 C 的参考实现, Jasmin 的参考实现少了许多出栈入栈操作. 这就是 Jasmin 的参考实现比 C 参考实现周期计数小的原因.

6 总结与展望

在本文中, 我们使用 Formosa Crypto 开发的 Jasmin-EasyCrypt 工具链对后量子密码算法 Falcon 的核心函数: Montgomery 模乘、NTT、FFT 和整数高斯采样算法的部分函数进行了形式化验证, 并对 Jasmin 生成的 Falcon 签名与验签函数的汇编代码进行了基准测试.

通过形式化验证, Falcon 中的 NTT、FFT 算法和整数高斯采样算法的部分函数的正确性得到了理论保证, 对于 Falcon 签名的后续部署意义重大, 进一步保证了现代通信和计算基础设施的安全性.

由于形式化证明工作复杂且充满挑战, 并且随着算法复杂程度的增加, 工作量也大幅增加, 而 Jasmin-EasyCrypt 工具链尚在开发和完善中, 我们目前无法提供对 Falcon 签名的完整形式化证明. 对于完整的整数高斯采样的概率正确性证明以及各函数证明连接的工作, 我们将在后续的研究中逐步进行探索和完善. 对于签名安全性的归约证明, 我们也将在后续工作中完成.

References:

[1] Ducas L, Kiltz E, Lepoint T, Lyubashevsky V, Schwabe P, Seiler G, Stehlé D. CRYSTALS-Dilithium: A lattice-based digital signature scheme. IACR Trans. on Cryptographic Hardware and Embedded Systems, 2018, 2018(1): 238–268. [doi: 10.13154/tches.v2018.i1.238-268]

[2] Fouque PA, Hoffstein J, Kirchner P, Lyubashevsky V, Pornin T, Prest T, Ricosset T, Seiler G, Whyte W, Zhang ZF. Falcon: Fast-Fourier lattice-based compact signatures over NTRU (Specification v1.2). 2020. http://falcon-sign.info/falcon.pdf

[3] Bernstein DJ, Hülsing A, Kölbl S, Niederhagen R, Rijneveld J, Schwabe P. The SPHINCS+ signature framework. In: Proc. of the 2019 ACM Conf. on Computer and Communications Security. London: ACM, 2019. 2129–2146. [doi: 10.1145/3319535.3363229]

[4] Bos J, Ducas L, Kiltz E, Lepoint T, Lyubashevsky V, Schanck JM, Schwabe P, Seiler G, Stehlé D. CRYSTALS-Kyber: A CCA-secure module-lattice-based KEM. In: Proc. of the 2018 IEEE European Symp. on Security and Privacy. London: IEEE, 2018. 353–367. [doi: 10.1109/EuroSP.2018.00032]

[5] Hoffstein J, Howgrave-Graham N, Pipher J, Silverman JH, Whyte W. NTRUSIGN: Digital signatures using the NTRU lattice. In: Topics in Cryptology—CT-RSA 2003. San Francisco: Springer, 2003. 122–140. [doi: 10.1007/3-540-36563-X_9]

[6] Gentry C, Peikert C, Vaikuntanathan V. Trapdoors for hard lattices and new cryptographic constructions. In: Proc. of the 40th Annual ACM Symp. on Theory of Computing. Victoria: ACM, 2008. 197–206. [doi: 10.1145/1374376.1374407]

[7] Barbosa M, Barthe G, Bhargavan K, Blanchet B, Cremers C, Liao K, Parno B. SoK: Computer-aided cryptography. In: Proc. of the 2021 IEEE Symp. on Security and Privacy. San Francisco: IEEE, 2021. 777–795. [doi: 10.1109/SP40001.2021.00008]

[8] Wang J, Zhan NJ, Feng XY, Liu ZM. Overview of formal methods. Ruan Jian Xue Bao/Journal of Software, 2019, 30(1): 33–61 (in Chinese with English abstract). http://www.jos.org.cn/1000-9825/5652.htm [doi: 10.13328/j.cnki.jos.005652]

[9] Wang SL, Zhan BH, Sheng HH, Wu H, Yi SC, Wang LT, Jin XY, Xue B, Li JH, Xiang SQ, Xiang Z, Mao BF. Survey on requirements classification and formalization of trustworthy systems. Ruan Jian Xue Bao/Journal of Software, 2022, 33(7): 2367–2410 (in Chinese with English abstract). http://www.jos.org.cn/1000-9825/6587.htm [doi: 10.13328/j.cnki.jos.006587]

[10] Zinzindohoué JK, Bhargavan K, Protzenko J, Beurdouche B. HACL*: A verified modern cryptographic library. In: Proc. of the 2017 ACM SIGSAC Conf. on Computer and Communications Security. Dallas: ACM, 2017. 1789–1806. [doi: 10.1145/3133956.3134043]

[11] Erbsen A, Philipoom J, Gross J, Sloan R, Chlipala A. Simple high-level code for cryptographic arithmetic—With proofs, without compromises. In: Proc. of the 2019 IEEE Symp. on Security and Privacy. San Francisco: IEEE, 2019. 1202–1219. [doi: 10.1109/SP.2019.00005]

[12] Hoare CAR. An axiomatic basis for computer programming. Communications of the ACM, 1969, 12(10): 576–580. [doi: 10.1145/363235.363259]

[13] Clarke EM, Emerson EA. Design and synthesis of synchronization skeletons using branching time temporal logic. In: Logics of Programs. Yorktown Heights: Springer, 1982. 52–71. [doi: 10.1007/BFb0025774]

[14] Clarke EM, Emerson EA, Sistla AP. Automatic verification of finite-state concurrent systems using temporal logic specifications. ACM Trans. on Programming Languages and Systems, 1986, 8(2): 244–263. [doi: 10.1145/5397.5399]

[15] Church A. An unsolvable problem of elementary number theory. American Journal of Mathematics, 1936, 58(2): 345–363. [doi: 10.2307/2371045]

[16] McMillan KL. Symbolic Model Checking. New York: Springer, 1993. 61–85. [doi: 10.1007/978-1-4615-3190-6]

[17] Holzmann GJ. The model checker SPIN. IEEE Trans. on Software Engineering, 1997, 23(5): 279–295. [doi: 10.1109/32.588521]

[18] Haselwarter PG, Rivas E, van Muylder A, Winterhalter T, Abate C, Sidorenco N, Hriţcu C, Maillard K, Spitters B. SSProve: A foundational framework for modular cryptographic proofs in Coq. ACM Trans. on Programming Languages and Systems, 2023, 45(3): 15. [doi: 10.1145/3594735]

[19] Almeida JB, Barbosa M, Barthe G, Blot A, Grégoire B, Laporte V, Oliveira T, Pacheco H, Schmidt B, Strub PV. Jasmin: High-assurance and high-speed cryptography. In: Proc. of the 2017 ACM SIGSAC Conf. on Computer and Communications Security. Dallas: ACM, 2017. 1807–1823. [doi: 10.1145/3133956]

[20] Barthe G, Dupressoir F, Grégoire B, Kunz C, Schmidt B, Strub PY. EasyCrypt: A tutorial. In: Foundations of Security Analysis and Design VII. Cham: Springer, 2014. 146–166. [doi: 10.1007/978-3-319-10082-1_6]

[21] Bhargavan K, Kiefer F, Strub PY. hacspec: Towards verifiable crypto standards. In: Proc. of the 4th Int'l Conf. on Security Standardisation Research. Darmstadt: Springer, 2018. 1–20. [doi: 10.1007/978-3-030-04762-7_1]

[22] Almeida JB, Baritel-Ruet C, Barbosa M, Barthe G, Dupressoir F, Grégoire B, Laporte V, Oliveira T, Stoughton A, Strub PY. Machine-checked proofs for cryptographic standards: Indifferentiability of sponge and secure high-assurance implementations of SHA-3. In: Proc. of the 2019 ACM SIGSAC Conf. on Computer and Communications Security. London: ACM, 2019. 1607–1622. [doi: 10.1145/3319535.

3363211]
[23] Shivakumar BA, Barthe G, Grégoire B, Laporte V, Oliveira T, Priya S, Schwabe P, Tabary-Maujean L. Typing high-speed cryptography against Spectre v1. In: Proc. of the 2023 IEEE Symp. on Security and Privacy. San Francisco: IEEE, 2023. 1094–1111. [doi: 10.1109/SP46215.2023.10179418]
[24] Firsov D, Unruh D. Zero-knowledge in EasyCrypt. In: Proc. of the 36th Computer Security Foundations Symp. Dubrovnik: IEEE, 2023. 1–16. [doi: 10.1109/CSF57540.2023.00015]
[25] Barbosa M, Barthe G, Doczkal C, Don J, Fehr S, Grégoire B, Huang YH, Hülsing A, Lee Y, Wu XD. Fixing and mechanizing the security proof of Fiat-Shamir with aborts and Dilithium. In: Proc. of the 43rd Annual Int'l Cryptology Conf. on Advances in Cryptology (CRYPTO 2023). Santa Barbara: Springer, 2023. 358–389. [doi: 10.1007/978-3-031-38554-4_12]
[26] Almeida JB, Barbosa M, Barthe G, Grégoire B, Laporte V, Léchenet JC, Oliveira T, Pacheco H, Quaresma M, Schwabe P, Séré A, Strub PY. Formally verifying Kyber—Episode IV: Implementation correctness. Trans. on Cryptographic Hardware and Embedded Systems, 2023, 2023(3): 164–193. [doi: 10.46586/tches.v2023.i3.164-193]
[27] Barbosa M, Dupressoir F, Grégoire B, Hülsing A, Meijers M, Strub PY. Machine-checked security for XMSS as in RFC 8391 and SPHINCS+. In: Proc. of the 43rd Annual Int'l Cryptology Conf. on Advances in Cryptology (CRYPTO 2023). Santa Barbara: Springer, 2023. 421–454. [doi: 10.1007/978-3-031-38554-4_14]
[28] Barnett M, Chang BYE, DeLine R, Jacobs B, Leino KRM. Boogie: A modular reusable verifier for object-oriented programs. In: Proc. of the 4th Int'l Symp. on Formal Methods for Components and Objects. Amsterdam: Springer, 2005. 364–387. [doi: 10.1007/11804192_17]
[29] Leino KRM. Dafny: An automatic program verifier for functional correctness. In: Proc. of the 16th Int'l Conf. on Logic for Programming, Artificial Intelligence, and Reasoning. Dakar: Springer, 2010. 348–370. [doi: 10.1007/978-3-642-17511-4_20]
[30] Zhou T, Zheng FY, Lin JQ, Wei R, Tang WX. On software implementations of post-quantum cryptography. Journal of Cryptologic Research, 2024, 11(2): 308–343 (in Chinese with English abstract). [doi: 10.13868/j.cnki.jcr.000681]

附中文参考文献:

[8] 王戟, 詹乃军, 冯新宇, 刘志明. 形式化方法概貌. 软件学报, 2019, 30(1): 33–61. http://www.jos.org.cn/1000-9825/5652.htm [doi: 10.13328/j.cnki.jos.005652]
[9] 王淑灵, 詹博华, 盛欢欢, 吴昊, 易士程, 王令泰, 金翔宇, 薛白, 李静辉, 向霜晴, 向展, 毛碧飞. 可信系统性质的分类和形式化研究综述. 软件学报, 2022, 33(7): 2367–2410. http://www.jos.org.cn/1000-9825/6587.htm [doi: 10.13328/j.cnki.jos.006587]
[30] 周天, 郑昉昱, 林璟锵, 魏荣, 唐文煦. 后量子密码算法的软件实现研究. 密码学报, 2024, 11(2): 308–343. [doi: 10.13868/j.cnki.jcr.000681]

田新蕾(2000－), 女, 硕士生, 主要研究领域为密码算法的形式化验证.

张济显(2002－), 男, 学士, 主要研究领域为博弈优化.

董奕逸(2001－), 女, 硕士生, 主要研究领域为密码算法的形式化验证.

王伟嘉(1988－), 男, 博士, 教授, 博士生导师, 主要研究领域为密码工程.