计算机科学
COMPUTER SCIENCE

基于 FPGA 的高性能可扩展 SM4-GCM 算法实现

翟嘉琪，李斌，周清雷，陈晓杰

引用本文

翟嘉琪，李斌，周清雷，陈晓杰. 基于 FPGA 的高性能可扩展 SM4-GCM 算法实现[J]. 计算机科学, 2022, 49(10): 74-82.

ZHAI Jia-qi, LI Bin, ZHOU Qing-lei, CHEN Xiao-jie. Implementation of FPGA-based High-performance and Scalable SM4-GCM Algorithm[J]. Computer Science, 2022, 49(10): 74-82.

计算机科学 COMPUTER SCIENCE
http://www.jsjkx.com
DOI:10.11896/jsjkx.210900137

基于 FPGA 的高性能可扩展 SM4-GCM 算法实现

翟嘉琪[1]　李　斌[1]　周清雷[1,2]　陈晓杰[2]
1 郑州大学信息工程学院　郑州 450001
2 数学工程与先进计算国家重点实验室　郑州 450001
(zhaijiaqi777@163.com)

摘　要　在大数据和5G技术蓬勃发展的背景下,高速通信系统中的信息加密成为了新的研究热点,如何在保证数据高安全性的同时,提高数据吞吐率,降低加密算法适配不同应用场景的难度成为了重要的研究课题。针对传统软件实现的SM4-GCM算法吞吐率小、难以在多变的5G及大数据场景下应用的问题,文中基于FPGA可重构的特点,深入剖析SM4-GCM算法的特征,利用Mastrovito算法、Karatsuba算法、快速求余算法,设计了两种高性能、数控分离、可扩展的电路结构,分别采用全流水线技术和四度并行技术对SM4-GCM算法进行加速优化,在保证高安全性的同时,达到了较高吞吐率,并且可灵活移植于各种应用场景。实验结果表明,所提出的两种方案中的单个SM4-GCM模块的吞吐率分别达到了28.16Gbps和28.8Gbps,在性能、可扩展性等方面均优于同类已发表的设计。

关键词:SM4;伽罗华/计数器模式;FPGA;高吞吐率;可扩展
中图法分类号　TP309

Implementation of FPGA-based High-performance and Scalable SM4-GCM Algorithm

ZHAI Jia-qi[1],LI Bin[1],ZHOU Qing-lei[1,2] and CHEN Xiao-jie[2]
1 School of Information Engineering,Zhengzhou University,Zhengzhou 450001,China
2 State Key Laboratory of Mathematical Engineering and Advanced Computing,Zhengzhou 450001,China

Abstract　In the context of vigorous development of big data and 5G technology,information encryption in high-speed communication systems has become a new research hotspot. How to increase data throughput and reduce the difficulty of adapting encryption algorithms to different application scenarios while ensuring high data security has become important research topics. Aiming at the problem that traditional software's SM4-GCM algorithm has a low throughput rate and is difficult to apply in changing 5G and big data scenarios,this paper analyzes the characteristics of SM4-GCM algorithm based on the reconfigurable characteristics of FPGA,using Mastrovito,Karatsuba and fast remainder algorithms. Two high-performance,CNC-separated and expandable circuit structures are designed. Full-pipeline technology and four-degree parallel technology are used to accelerate the optimization of SM4-GCM algorithm. While ensuring high security,it can achieve a high throughput rate,and can be flexibly transplanted to various application scenarios. Experimental results show that the throughput rates of the proposed two solutions in this paper for a single SM4-GCM module have reach 28.16 Gbps and 28.8 Gbps,respectively,which are superior to similar published designs in terms of performance and scalability.

Keywords　SM4,Galois/Counter Mode,FPGA,High throughput rate,Scalable

1　引言

SM4是我国自主研发的商用密码算法,于2021年6月25日正式成为ISO/IEC国际标准,被广泛应用于加密通信、加密存储等领域[1]。仅对数据加密难以抵御重放和篡改攻击,而高安全性要求应用不仅需要验证信息的完整性和保密性,还需要对信息的来源进行认证,因此美国电气电子工程师学会提出了伽罗华/计数器模式(Galois/Counter Mode,GCM)高级加密标准[2]。随着大数据、人工智能、5G领域的发展,人们对网络加密数据的吞吐率有了更高的要求。新的应用场景下,数据加密吞吐率低下成为了数据传输的瓶颈。

当前已有较多针对SM4算法的研究。Fu等[3]基于CBC模式,提出了一种4轮合1的组合逻辑SM4电路,从而提高了吞吐率。Li等[4]设计了一种基于乒乓技术的双加密模块

到稿日期:2021-09-16　返修日期:2022-03-14
基金项目:国家自然科学基金(61702518);国家重点研发计划"公共安全风险防控与应急技术装配"重点专项(2018XXXXXXX01)
This work was supported by the National Natural Science Foundation of China(61702518) and National Key R&D Program "Public Safety Risk Preventionand Controland Emergency Technology Assembly" Key Special Project(2018XXXXXXX01).
通信作者:李斌(iebinli@zzu.edu.cn)

SM4-GCM 方案,在资源消耗较少的条件下实现了较高的吞吐率。Zheng 等[5]采用 CPU-GPU 异构计算模型对 SM4 算法进行并行化改进,在实现较高吞吐率的同时提高了程序的可移植性。Wang 等[6]针对 SM4 算法的 CTR 模式,提出了一种数据、控制分离的安全可扩展结构,吞吐率较高。

上述研究仍存在安全性低、吞吐率低、灵活性弱的问题。例如,基于 ASIC 的实现通用性不强;基于 GPU 的实现功耗高、稳定性较差[7-9];基于 FPGA 的实现单模块吞吐率不高、参数固定,难以满足多变的应用场景;仅采用 CTR 模式安全性不足,不能满足网络防篡改需求。

基于上述研究存在的问题,本文提出了基于 FPGA 的 SM4-GCM 算法并行实现,对 SM4-GCM 进行了全流水线的并行加速优化。首先采用流水线技术对 SM4 进行优化,然后分别采用 Mastrovito 乘法器和 Karatsuba 乘法器结合快速求余算法优化 GHASH 运算,最后采用松耦合架构,以异步 FIFO 连接各个模块,提高了 FPGA 的资源利用率和灵活性。实验结果表明,本文提出的全流水线型 SM4-GCM 优化实现和基于四度并行优化的 SM4-GCM 实现两种方案的单模块加密认证的吞吐率均优于其他方案。在加密吞吐率需求较高的场景下,扩展的 SM4-GCM 算法的吞吐率可达到 100 Gbps,满足 5G 应用场景的要求。

2 SM4-GCM 算法原理分析

SM4-GCM 算法的工作流程分为认证加密和认证解密,认证加密流程包含 4 个输入和 2 个输出,均为一个比特串[10],如表 1 所列。

表 1 SM4-GCM 加密算法输入符号说明

Table 1 Description of SM4-GCM encryption algorithm input symbol

符号	说明
明文 P	表示需要进行加密的明文数据,长度范围是 $0\sim 2^{39}-256$
密钥 MK	表示所要使用的密钥
初始向量 IV	初始向量,它可以有 1 到 2^{64} 之间的任意比特数,96 比特的 IV 值在算法中更为有效
附加数据 A	附加数据,用于认证,不能被加密
密文 C	长度和加密前的明文长度相匹配
认证标签 TAG	它的长度可以是 64 至 128 之间的任意值,用于解密认证

SM4-GCM 算法的总体流程如图 1 所示。首先,利用 $E(K,0^{128})$ 计算出哈希算子 H,然后计数器改变初始向量 IV 值后与密钥进行 SM4 加密。输入的明文被分成了 n 组,每组数据 128 bit,最后一组可能不是 128 bit,但一定是 8 的倍数;其次,将 SM4 得到的结果和明文分组进行异或运算得到密文 C;然后把 C,A 和 H 传入 GHASH 模块,用于计算认证标签 TAG;最后将认证标签和密文一起发送给接收方,接收方根据密文和附加数据重新计算 TAG',若与 TAG 一致,则表示没有篡改。

SM4-GCM 认证解密算法和加密算法在结构上基本相同,区别是:1)在认证解密算法中,密文作为输入,明文作为输出;2)认证加密算法是先加密,后进行哈希函数运算,而认证解密算法是先使用哈希函数运算,后解密。

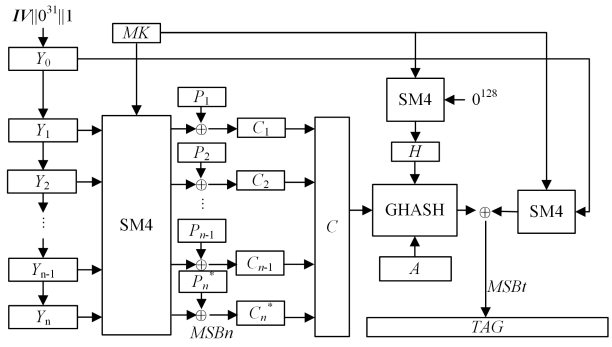

图 1 SM4-GCM 算法结构图

Fig. 1 SM4-GCM algorithm structure diagram

3 核心算法优化

将 SM4-GCM 算法应用到高速加密场景时,如何提高吞吐率和资源利用率一直是一个研究热点[11]。本文采用流水线和并行技术实现 SM4-GCM 算法的优化,并采用数据/控制分离的松耦合架构来提高电路的可扩展性[12]。

3.1 SM4 流水线优化

SM4 算法是一种分组对称密码算法,分组长度为128 bit,密钥长度也为 128 bit,主要包括密钥扩展算法、轮加解密算法。加密算法与密钥扩展算法均采用 32 轮非线性迭代结构。解密算法和加密算法结构相同,只是轮密钥的使用顺序相反[13]。

3.1.1 SM4 密钥扩展算法

SM4 密钥扩展算法的流程图如图 2 所示。加密过程中使用的轮密钥由加密密钥 $MK=(MK_0,MK_1,MK_2,MK_3)\in(Z_2^8)^4$ 生成,生成方法如式(1)和式(2)所示。其中,FK 为系统参数;CK 为固定参数;T' 为合成置换函数,如式(3)所示。它包括线性变换 L' 和非线性变换 τ,如式(4)和式(5)所示。

$$(K_0,K_1,K_2,K_3)=(MK_0\oplus FK_0,MK_1\oplus FK_1,MK_2\oplus FK_2,MK_3\oplus FK_3) \tag{1}$$

$$rk_i=K_{i+4}=K_i\oplus T'(K_{i+1}\oplus K_{i+2}\oplus K_{i+3}\oplus CK_i) \tag{2}$$

$$T'=L'(r(\cdot)) \tag{3}$$

$$L'(B)=B\oplus(B\ll 13)\oplus(B\ll 23) \tag{4}$$

$$(b_0,b_1,b_2,b_3)=\tau(A)=(Sbox(a_0),Sbox(a_1),Sbox(a_2),Sbox(a_3)) \tag{5}$$

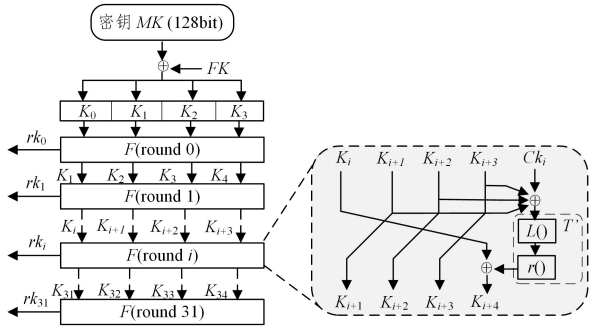

图 2 SM4 密钥扩展算法示意图

Fig. 2 Diagram of SM4 key expansion algorithm

3.1.2 SM4 加解密算法

SM4 加密算法的流程图如图 3 所示。轮加密算法由 32 次轮函数 F 运算和一次反序变换 R 组成。设明文输入为

$(X_0, X_1, X_2, X_3) \in (Z_2^{32})^4$，密文输出为 $(Y_0, Y_1, Y_2, Y_3) \in (Z_2^{32})^4$，轮密钥为 $rk_i \in Z_2^{32}$，$i=0,1,2,\cdots,31$。32 轮轮函数 F 运算如式(6)和式(7)所示，反序列变换如式(8)所示。SM4 加解密算法中的线性变换 L 与密钥扩展算法不同，如式(9)所示。

$$X_{i+4}=F(X_i,X_{i+1},X_{i+2},X_{i+3},rk_i),i=0,1,2,\cdots,31 \quad (6)$$

$$F(X_i,X_{i+1},X_{i+2},X_{i+3},rk_i)=X_i \oplus T(X_{i+1}\oplus X_{i+2}\oplus X_{i+3}\oplus rk_i),i=0,1,2,\cdots,31 \quad (7)$$

$$(Y_0,Y_1,Y_2,Y_3)=R(X_{32},X_{33},X_{34},X_{35})=(X_{35},X_{34},X_{33},X_{32}) \quad (8)$$

$$L(B)=B\oplus(B\ll 2)\oplus(B\ll 10)\oplus(B\ll 18)\oplus(B\ll 24) \quad (9)$$

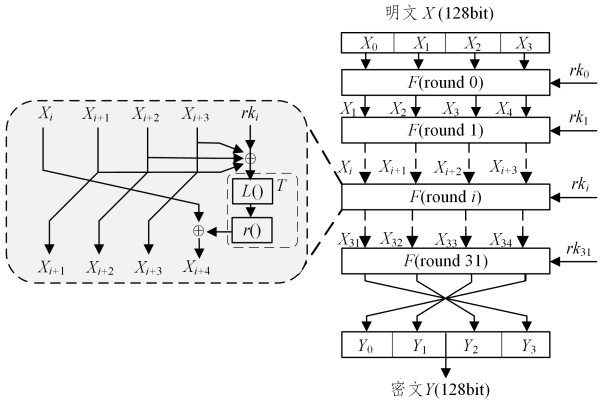

图 3 SM4 加密流程示意图

Fig. 3 Diagram of SM4 encryption process

3.1.3 SM4 算法流水线原理

SM4 的加密算法流水线实现主要包含两部分：密钥扩展计算和轮加密计算。其中，密钥扩展第 i 轮的计算过程描述如下。

输入：$K_i, K_{i+1}, K_{i+2}, K_{i+3}, CK_i$

输出：$rk_i = K_{i+4}$

运算公式如式(2)所示。

模块内以组合逻辑按照算法流程实现相应运算，模块间以时钟控制进行数据赋值和输出，数据在相邻模块依次传递。

轮加密第 i 轮的计算过程描述如下：

输入：$X_i, X_{i+1}, X_{i+2}, X_{i+3}, rk_i$

输出：$X_{i+1}, X_{i+2}, X_{i+3}, X_{i+4}$

运算公式如式(6)所示。

各个模块之间并行运算，当数据连续输入运算模块中，在 33 个时钟之后，每个时钟可产生一个输出[14]。SM4 的加密流水线实现思想和 FPGA 硬件结构分别如图 4 和图 5 所示。

采用流水线实现的 SM4 加解密算法对明文和密文的处理均划分了 32 个独立模块，忽略了流水线段间寄存器的延时，在填满流水线的情况下相比传统实现计算速率提高了 32 倍。

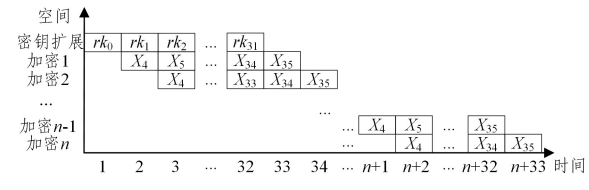

图 4 SM4 加密流水线原理

Fig. 4 Principles of SM4 Encryption pipeline

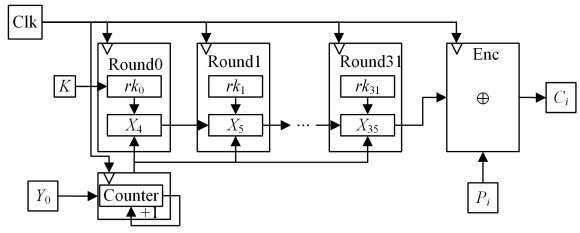

图 5 SM4 流水线硬件结构图

Fig. 5 Diagram of SM4 pipeline hardware structure

解密算法的流水线优化与加密算法同理，区别在于解密算法需将轮密钥全部计算完成后进行轮解密运算，因此数据在第 64 个时钟时填满整条流水线，之后每个时钟产生一个解密数据。

3.2 GHASH 优化

3.2.1 GHASH 算法分析

伽罗华/计数器模式(Galois/Counter Mode，GCM)是在二元伽罗华域(Galois Domain)上使用泛散列函数对加密进行认证的分组密码算法[15]。其采用 CTR 模式来实现对明文的加密，采用伽罗华域的 GHASH 函数进行认证，起到了身份验证及数据完整性检验的作用[16]。

GHASH 是一种基于伽罗华域的 128 比特乘法器的哈希操作。在 $GF(2^{128})$域中不可约多项式为：$f(x)=x^{128}+x^7+x^2+x+1$。GHASH 函数将附加数据 A、密文 C 和加密密钥 MK 作为输入，运算得到的运算结果记为 $X_i(i=0,1,\cdots,m+n+1)$，其中 X_i 的定义如式(10)所示[2]。

$$X_i=\begin{cases}0, & i=0\\ (X_{i-1}\oplus A_i)\times H, & i=1,2,\cdots,m-1\\ (X_{i-m}\oplus(A_m \parallel 0^{128-v}))\times H, & i=m\\ (X_{i-1}\oplus C_{i-m})\times H, & i=m+1,m+2,\cdots,m+n-1\\ (X_{m+n-1}\oplus(C_n \parallel 0^{128-u}))\times H, & i=m+n\\ X_{m+n}\oplus(len(A)\parallel len(C))\times H, & i=m+n+1\end{cases} \quad (10)$$

经过$(m+n+1)$个时钟后输出 X_{m+n+1}。将结果与 $E(Y_0)$异或得到认证标签 TAG。

式(11)给出了不同于式(10)的四度并行计算 X_{m+n+1}的方法。它将顺序乘加运算划分为多个并行计算，其中 Q_4, Q_3, Q_2, Q_1为可并行计算的中间变量，通过定义并行度 q，X_{m+n+1}可表示为 q 个子项 Q_i的异或和。其中输入变量 $I_i(i=1,2,3,\cdots,m+n+1)$ 的定义如式(12)所示。

$$\begin{aligned}&X_{m+n+1}=Q_1\oplus Q_2\oplus Q_3\oplus Q_4,\text{where}\\ &Q_1=(((I_1H^4\oplus I_5)H^4\oplus I_9)H^4\oplus\cdots)H^4\\ &Q_2=(((I_2H^4\oplus I_6)H^4\oplus I_{10})H^4\oplus\cdots)H^3\\ &Q_2=(((I_2H^4\oplus I_6)H^4\oplus I_{10})H^4\oplus\cdots)H^3\\ &Q_4=(((I_4H^4\oplus I_8)H^4\oplus I_{12})H^4\oplus\cdots)H\end{aligned} \quad (11)$$

$$(I_1I_2\cdots I_{m+n+1})=(A_1,\cdots,A_m^*\parallel 0^{128-v},C_1,\cdots,C_n^*\parallel 0^{128-u},len(A)\parallel len(C)) \quad (12)$$

SM4-GCM 电路结构的优劣主要取决于 GHASH 电路计算 X_i时所采用的计算方法[17-19]。本文以 Mastrovito 乘法器实现了 X_i的组合逻辑运算，以 Karatsuba 乘法器和快速求余算法结合的方法实现了 X_i的四度并行计算。

3.2.2　Karatsuba 乘法器

GHASH 模块内部需要大量的大数相乘，传统的乘法不仅计算效率低，而且需要消耗大量资源，因此本文采用 Karatsuba 乘法器[20]，其计算性能优越且硬件资源占用较少。

Karatsuba 算法的设计是基于分解相乘的思想，其本质是将位宽较大的数等分为几个位宽较小的数，在此基础上附加一定的移位和加法运算实现大数相乘。例如两个大整数 X 和 Y（分别为 n 比特位）相乘，已知 $X=A\parallel B$，$Y=C\parallel D$，则 XY 的算法如式(13)所示，计算流程如图 6 所示。

$$\begin{aligned}X\times Y&=(X^{\frac{n}{2}}\times A+B)(X^{\frac{n}{2}}\times C+D)\\&=X^{n}\times(AC)+X^{\frac{n}{2}}\times(AD+BC)+BD\end{aligned}\tag{13}$$

观察式(13)可知，计算 XY 需要进行 4 次乘法运算和 3 次加法运算（移位运算时间复杂度忽略不计）。

$$\begin{array}{cccc}&&A&B\\&\times&C&D\\\hline&&AD&BD\\&AC&BC&\\\hline&AC&AD+BC&BD\end{array}$$

图 6　XY 的计算流程

Fig. 6　Calculation process of XY

对于 $AD+BC$ 部分，可以按式(14)进行转换，重复利用乘积结果 AC 和 BD，将两次乘法运算化简为 1 次乘法运算和 4 次加法运算，降低了计算复杂度。

$$\begin{aligned}AD+BC&=AC+AD+BC+BD-AC-BD\\&=(A+B)(C+D)-AC-BD\end{aligned}\tag{14}$$

经上述分析可知，利用 Karatsuba 算法计算两大数相乘只需 3 次乘法运算和 6 次加法运算，时间复杂度从 $4O(n^2)$ 降低至 $O(3n^2)$，消耗门电路个数从 $O(n^2)$ 降低至 $O(n^\delta)$，其中 $1<\delta<2$。这说明使用 Karatsuba 设计的乘法器可有效降低电路复杂度，尤其当 n 较大时，Karatsuba 乘法器可更明显地优化利用电路资源。但当分解次数继续增加时，电路延时也会随之增加，实验发现当选择 3 层 KOA（Karatsuba-Offman-Algorithm）将数据从 128 bit 分解到 16 bit 时，其相比其他 r-KOA 方案所需要的面积功耗更低。设计的 Karatsuba 乘法器如图 7 所示。

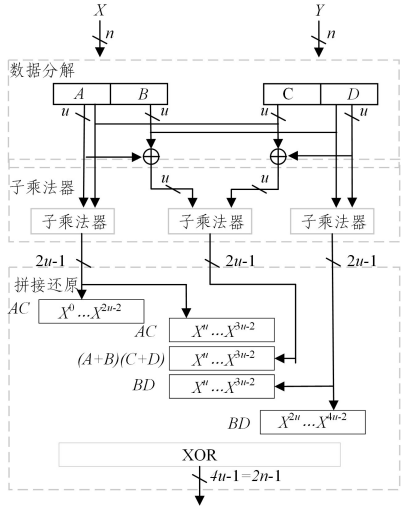

图 7　Karatsuba 乘法器电路的结构

Fig. 7　Structure of Karatsuba multiplier circuit

将 128 bit 数据 X 和 Y 在 3 个时钟周期内利用 Karatsuba 算法分解至 16 bit，然后进行乘法运算的数据分解流程如图 8 所示。

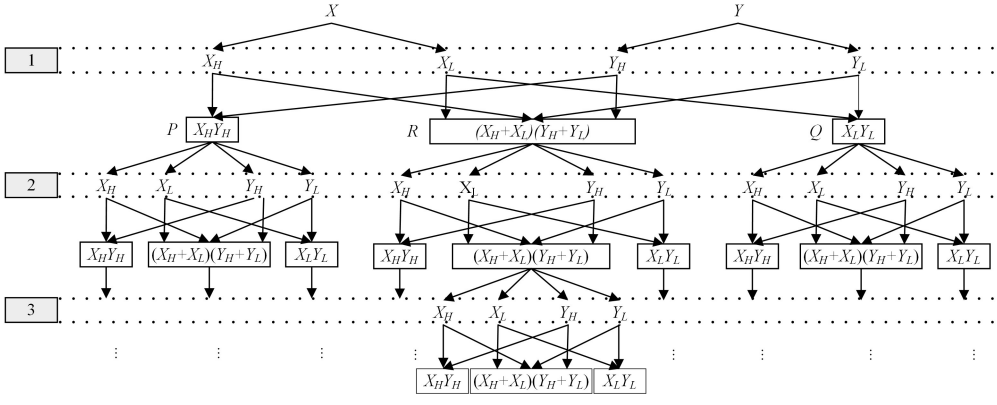

图 8　数据分解流程图

Fig. 8　Diagram of data decomposition

3.2.3　快速求余

Karatsuba 乘法器最后输出一个 256 位的结果，但 GF(2^{128})运算要求最终结果落入伽罗华域，因此需要对乘法结果求余。

传统的求余方法是将中间乘法结果与简化矩阵相乘。但这种运算需要消耗大量时间、资源进行矩阵乘法，且无法采用流水线优化，因此本文采用了文献[21]提出的快速求余方法。该方法采用几个时间复杂度为 O(1)的移位异或运算代替时间复杂度为 $O(n^2)$ 矩阵相乘，同时空间复杂度从 $O(n^2)$ 降至 $O(n)$，算法流程如算法 1 所示。

算法 1　快速求余算法

输入：乘法运算结果($X_3:X_2:X_1:X_0$)

输出：求余结果($X_1\oplus H_H:X_0\oplus H_L$)

Begin

1. $A=X_3\gg63$
2. $B=X_3\gg62$
3. $C=X_3\gg57$
4. $D=X_2\oplus A\oplus B\oplus C$
5. $E_H:E_L=X_3:D\ll1$
6. $F_H:F_L=X_3:D\ll2$
7. $G_H:G_L=X_3:D\ll7$

8. $H=H_H:H_L=(X_3\oplus E_H\oplus F_H\oplus G_H):(D\oplus E_L\oplus F_L\oplus G_L)$

End

3.2.4 Mastrovito 乘法器

传统的标准多项式基底乘法由一个多项式乘法后跟一个求模运算组成。定义 $C(x)=A(x)B(x) \bmod P(x)$，其中 $A(x),B(x),C(x)\in GF(2^m)$ 如式(15)－式(17)所示，多项式 $P(x)\in GF(2^{m+1})$ 如式(18)所示。

$$A(x)=\sum_{i=0}^{m-1}a_i x^i, a\in GF(2) \tag{15}$$

$$B(x)=\sum_{i=0}^{m-1}b_i x^i, b\in GF(2) \tag{16}$$

$$C(x)=\sum_{i=0}^{m-1}c_i x^i, c\in GF(2) \tag{17}$$

$$P(x)=\sum_{i=0}^{m}p_i x^i, p\in GF(2) \tag{18}$$

Mastrovito 算法[22]将 $GF(2^m)$ 运算分为 3 步，首先将 $A(x)$ 转置后生成矩阵 $\boldsymbol{U}$ 和 $\boldsymbol{V}$，再将 $P(x)$ 转换为简约矩阵 $\boldsymbol{R}$，最后利用公式 $C=(U+R^TV)B$ 计算出乘法结果。算法流程如算法 2 所示。

算法 2 Mastrovito 算法

输入：(A(x),B(x),P(x))

输出：C(x)

Begin

1. 将 A(x)转换成矩阵 **U**,**V**：

$$\mathbf{U}=\begin{pmatrix} a_0 & 0 & \cdots & 0 & 0 \\ a_1 & a_0 & \cdots & 0 & 0 \\ \vdots & \vdots & \ddots & \vdots & \vdots \\ a_{m-1} & a_{m-2} & \cdots & a_1 & a_0 \end{pmatrix},\mathbf{V}=\begin{pmatrix} 0 & a_{m-1} & \cdots & a_2 & a_1 \\ 0 & 0 & \cdots & a_3 & a_2 \\ \vdots & \vdots & \ddots & \vdots & \vdots \\ 0 & 0 & \cdots & 0 & a_{m-1} \end{pmatrix}$$

2. /* 将 P(x)转换成简约矩阵 **R** */

3. for k←0 to 127 do

4. if k=0 or k=k_1 or k=k_2 or k=k_t

5. **R**[0][k]=1

6. else

7. **R**[0][k]=0

8. for r←1 to 127 do

9. for c←0 to 127 do

10. **R**[r][(r+c) mod 128]=**R**[0][c]

11. $\mathbf{C}=(\mathbf{U}+\mathbf{R}^T\mathbf{V})\mathbf{B}=\begin{pmatrix} c_0 \\ c_1 \\ \vdots \\ c_{m-1} \end{pmatrix}$ /* 求矩阵 **C** */

12. $C(x)=\sum_{i=0}^{m-1}c_i x^i$.

End

对于多项式基底 $f(x)=x^{k\Delta}+x^{(k-1)\Delta}+\cdots+x^{\Delta}+1$，其中 $m=k\Delta$，Mastrovito 乘法器需要消耗 m^2 个与门和 $(m^2-\Delta)$ 个异或门。由于不需要对中间结果求余，整体门电路消耗低于传统算法，计算速度更快，且将其配置在 SM4-GCM 算法中时，可随着多项式基底的改变而灵活调整参数，提高了灵活性[10,23]。本文采用组合逻辑实现 Mastrovito 乘法器，一个时钟可计算出结果。

4 高速 SM4-GCM 密码电路结构设计

典型的 SM4-GCM 密码电路结构主要有串行式和并行式两种，本文设计了全流水线型 SM4-GCM 电路和基于四度并行优化的 SM4-GCM 电路，并在此基础上采用 Mastrovito 乘法器、Karatsuba 乘法器和快速求余算法结合的方法来提高 X_i 的计算速度，从而提升电路性能。

4.1 全流水线的 SM4-GCM 优化实现

方案一采用全流水线的 SM4-GCM 优化实现，由 SM4 加密模块和 Mastrovito 乘法器组成。SM4 全流水线实现每一个时钟都可输出一组结果，全流水线型 SM4-GCM 的结构图如图 9 所示。

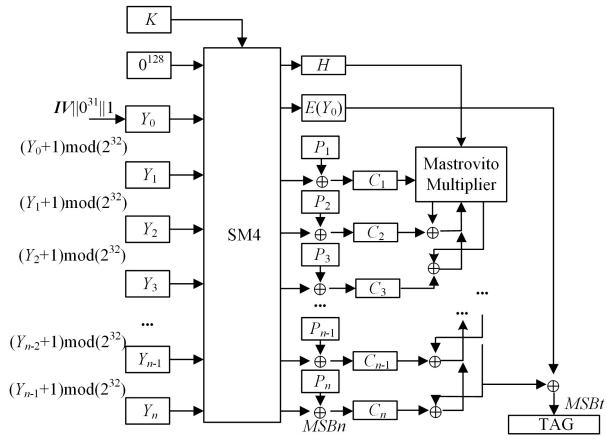

图 9 全流水线型 SM4-GCM 结构图

Fig. 9 Diagram of fully pipelined SM4-GCM structure

在该结构中，SM4 采用流水线实现，GHASH 由组合逻辑实现。当 SM4-GCM 电路运行时，配合 SM4 流水线，每加密一组密文可立即进行 GHASH 运算，从而得到有限域乘法结果。

首先，由控制模块调度计算出 H，并存入寄存器中，接着计算出 $E(Y_0)$；然后输入连续的明文分组，每组明文在进行 SM4 加密后配合 GHASH 计算出当前密文分组的有限域乘法结果；最后，当输入最后 1 组数据时，提前计算出整个明文长度，并和最后 1 组数据依次送入 GHASH，再结合 $E(Y_0)$，计算出 T。

显然，当 SM4-GCM 满负荷工作时，每个时钟只能处理 1 组数据。配合 SM4 流水线，当时钟频率为 f MHz 时，每秒可处理 f 组数据。因此，对于 m 组加密结果，计算出所有结果并输出所需时钟周期数如式(19)所示，得到的流水线的吞吐率如式(20)所示。

$$T=m \tag{19}$$

$$TP=128m/m\times f=128f \tag{20}$$

4.2 基于四度并行优化的 SM4-GCM 实现

根据式(11)，针对 GHASH 模块设计了基于四度并行优化的 SM4-GCM 电路，SM4 模块与方案一复用，GHASH 模块由 KOA 乘法器实现。其电路结构如图 10 所示。

该结构的 SM4 加密端时钟频率可综合至 300MHz，GHASH 认证端时钟频率最高可综合至 225 MHz，因此本文采用 FIFO 匹配两模块的计算速率。首先 SM4 模块计算哈希算子 H，然后使用 KOA 乘法器计算 $H^1\sim H^4$ 并存储结果，同时将明文 P 加密为密文 C，并存入 FIFO 中。将计算得出的 H 和 C 通过仲裁送入 KOA 乘法器，计算 X_{m+n+1}，计算

采用式(11)表述的四度并行方法，最后将 X_{m+n+1} 和 $E(Y_0)$ 异或后得到认证标签 TAG。四度并行运算的输入数据流如表 2 所列。

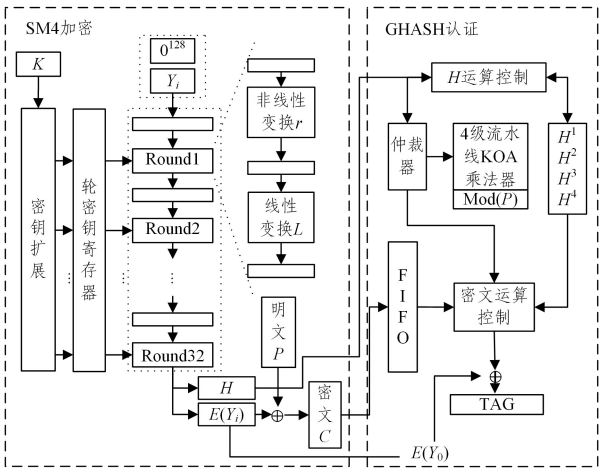

图 10 基于四度并行优化的 SM4-GCM 电路结构图

Fig. 10 Diagram of SM4-GCM circuit structure based on four-degree parallel optimization

表 2 四度并行运算的输入数据流

Table 2 Input data stream for four-degree parallel operation

时钟	I_k	H^k
1	I_0	H^4
2	I_1	H^3
3	I_2	H^2
4	I_3	H
5	I_4	H^3
6	I_5	H^2
7	I_6	H
8	Q_1	H^3
…	…	…

由表 2 所列的四度并行运算的输入数据流可知，前 4 组数据可连续输入，之后每 4 个时钟周期中，前 3 个时钟输入密文，最后 1 个时钟输入 Q_i，实现每 4 个时钟周期计算 3 组结果。因此，对于 m 组加密结果，有 $M=m+1$ 组数据传入 GHASH 中，计算出所有结果并输出所需时钟周期数、吞吐率，如式(21)和式(22)所示。

$$T=4+4\times(M-4)/3+4=4(M-4)/3+8=1.33M \quad (21)$$

$$TP=128M/(M+3)\times 0.75f=96f \quad (22)$$

4.3 其他优化技术

4.3.1 松耦合架构

在软件领域，“耦合”一般指软件组件之间的依赖程度。松耦合系统，由许多小巧且自给自足的小模块组成，这些自给自足的模块在程序迭代过程中可以单独迭代优化，只需要保持模块间接口统一即可。

方案一中，SM4 模块用于加解密运算，GHASH 模块用于生成认证标签，两模块之间的调度优化由控制逻辑模块实现。3 个模块功能划分明确、职责清晰，在接口不变的条件下，使用其他方法优化任意模块，对其他模块的正确执行不造成错误影响，方便了将来更进一步的优化。

在优化方案一采用的松耦合架构的基础上，利用 FIFO 匹配 SM4 模块和 GHASH 模块的计算频率，并编写控制逻辑协调两模块的运算。

优化方案一和方案二采用的松耦合架构分别如图 11 和图 12 所示。

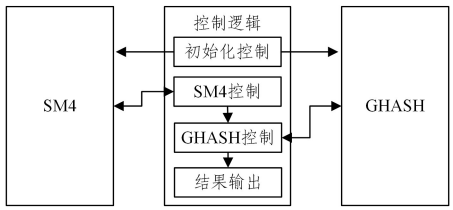

图 11 方案一的松耦合架构

Fig. 11 Loosely coupled architecture of project one

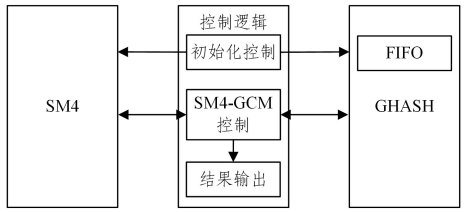

图 12 方案二的松耦合架构

Fig. 12 Loosely coupled architecture of project two

显然两种方案实现了数控分离和 SM4 流水线的复用。两种方案移植于具体应用时，可用 FIFO 封装顶层模块，将密钥、待加密数据直接送入 SM4-GCM 模块中加密。对于 100 GB 高速网络，可放置 4 个模块并行，从而满足更高速度的数据处理要求。

4.3.2 多级缓冲技术

多级缓存技术是实现高性能算法的重要手段之一。一方面，将多级缓存技术用于模块之间的通信数据，可以减小计算模块之间的耦合度，通过增加缓存寄存器或者存储器，建立模块之间的路由中断，来增加布线成功的效率；另一方面，将多级缓存技术用于数据信号对应的控制信号，使数据与控制分离，能够在数据处理后正确捕捉到对应的控制信号，增加算法运行时的可靠性。本文采用的多级缓冲技术如图 13 所示。

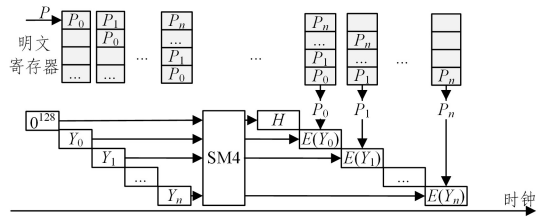

图 13 多级数据缓冲

Fig. 13 Multi-level data buffering

将 0^{128}，Y_0，Y_1，…，Y_n 依次输入到 SM4 模块中，每输入一个数据消耗一个时钟周期，每个数据花费 32 个时钟用于加密。第二个时钟周期时，获取 H 并存储，采用多级缓存技术存储明文有效信号及相应数据 P_0，P_1，…，P_n，这些数据在缓存中等待 32 个时钟后与相应的 $E(Y_i)$ 进行异或运算，得到密文 C。

4.3.3 面积功耗优化

在万物互联的背景下，加密和签名的绿色计算成为了安全加速设计中新的考核指标，下文针对本文的设计从面积

功耗优化方面做如下简要分析。

由 3.2 节中的分析可知，利用 Karatsuba 算法、快速求余算法和 Mastrovito 算法设计的乘法电路相比传统实现有效地减少了 SM4-GCM 电路的资源消耗，降低了硬件复杂度，提高了计算效率，实现了对 SM4-GCM 电路的整体优化。

不同综合策略和布线策略的选择会影响算法实现结果，资源占用和功耗也会随之改变。在满足布线时序时，本文选择面积和功耗最优的策略组合，即 Flow_AlternateRoutability 和 Performance-ExtraTimingOpt，从而通过综合策略的选择来实现进一步的面积和功耗优化。

综合布线结果表明：方案一占用 58 291 个资源，耗能为 2.554 w；方案二占用 53 380 个资源，耗能为 2.014 w。相比传统软件实现和 GPU 实现，基于 FPGA 实现的两种方案均在低功耗的条件下取得了很高的吞吐率，满足“节能降碳，绿色发展”的绿色计算要求，助力实现碳中和。

5 综合结果

5.1 两方案对比

5.1.1 性能对比

采用 Verilog HDL 描述两种优化方案，使用软件 Vivado 2019.2 仿真验证，在平台 Xilinx Zynq-7035 开发板中开发，使用芯片 ZYNQ XC7Z035-FFG676-2（Kintex-7 架构）实现。表 3 列出了本文设计的两种 FPGA 资源开销和性能评估。

表 3 两种方案资源消耗对比

Table 3 Comparison of resource consumption of two schemes

方案	器件	FFs	LUTs	RAMs	频率/MHz	吞吐率/Gbps
本文方案一	ZYNQ XC7Z035FFG676-2	12579	22856	0	220	28.16
本文方案二	ZYNQ XC7Z035FFG676-2	11528	20296	2	300	28.80

本文使用 R 表示占用资源数量、P 表示性能，使用 PR 代表性能资源比，这两种方案的性能资源比的计算式如式(23)和式(24)所示。方案一和方案二的性能资源比如式(25)和式(26)所示。

$$R=LUTs\times 2+FFs \tag{23}$$

$$PR=\frac{TP}{R} \tag{24}$$

$$PR1=\frac{TP1}{LUTs1\times 2+FFs1}=\frac{220\times 10^6\times 128}{22856\times 2+12579}\approx 4.83\times 10^5 \tag{25}$$

$$PR2=\frac{TP2}{LUTs2\times 2+FFs2}=\frac{300\times 10^6\times 96}{20926\times 2+11528}\approx 5.40\times 10^5 \tag{26}$$

5.1.2 应用场景对比

方案一与方案二主要有以下不同：

(1)方案一中的 GHASH 模块采用 Mastrovito 乘法器实现，方案二采用 Karatsuba 乘法器配合快速求余算法实现。

(2)方案一中的 GHASH 模块采用组合逻辑实现，方案二采用四度并行流水线实现。

(3)方案一的功能模块参数可根据不同的多项式基底灵活配置，实用性更强[10]；方案二参数不可配置，但占用资源更少。

由于两种方案对于 GHASH 模块的优化策略不同，导致其在密钥是否变换方面存在性能上的差异。在密钥不断变换的情况下，方案一都可立即算出 H，并参与随后的计算，其速度不受密钥变化影响，因此可将方案一应用于服务器端，加速处理不同用户的数据。而方案二每次都需要消耗 12 个时钟重新计算 $H^1\sim H^4$，当密钥不断变换时，性能会略有下降。但当密钥不发生变换时，其综合性能更高。由于单个用户的密钥一般短时间内不会发生变化，因此可将方案二应用于客户端。

高速网络中可能存在窃听和假冒等安全威胁，攻击者窃取隐私数据或通过篡改数据假冒为合法用户[24]。而 SM4-GCM 通过数据加密、保证数据完整性和数据认证，来有效解决此类安全问题。再者，SM4 算法采用非平衡 Feistel 结构，抗分差攻击能力较强，由于其核心算法的碰撞阈值高，且轮函数安全，因此破解难度大。表 4 列出了核心算法的碰撞阈值。

表 4 核心算法的碰撞阈值

Table 4 Collision threshold of core algorithm

核心算法	碰撞阈值
SM4	$2^{128}\approx 3.4\times 10^{38}$
GHASH	$2^{64}\approx 1.8\times 10^{19}$

本文实现的两种方案的吞吐率分别达到了 28.16 Gbps 和 28.80 Gbps，完全能够满足 10 GB 高速网络加密认证的要求。针对 100 GB 高速网络，配合 PCIe，实例化 4 个方案一所述的 SM4-GCM 模块实现并行加速，以达到 100 GB 加速吞吐量需求，实验结果如表 5 所列。

表 5 100 GB 方案的实验结果

Table 5 Experimental results of 100 GB program

方案	器件	FFs	LUTs	RAMs	频率/MHz	吞吐率/Gbps
本文方案一 100 GB 实现	ZYNQ XC7Z035FFG676-2	55802	118656	60	220	112.64

其次，SM4-GCM 加密认证算法具有多种应用场合，包括 IEEE 802.1ae 安全协议、IPSec 加密认证等。本文设计的 SM4-GCM 电路可通过 PCIe 作为加密卡与计算机配合使用，使用 FIFO 互联，按地址将数据送入 SM4-GCM 模块，可对图片、文本等多种信息进行加密，也可将其结合 5G 场景应用，将 MAC 数据包送入 SM4-GCM 模块，通过 PCIe 配置密钥等参数，具有实际的应用价值。将方案一应用于两种场景的测试结果如表 6 所列。

表 6 两种应用场景的测试结果

Table 6 Test results of two application scenarios

方案	器件	FFs	LUTs	数据大小/GB	加密时间/s	吞吐率/Gbps
加密卡	ZYNQ XC7Z035FFG676-2	45 335	55 886	1.12	0.040 6	27.62
万兆网数据加密	ZYNQ XC7Z035FFG676-2	50 801	60 716	0.98	0.100 1	9.79

5.2 与其他方案对比

进一步地,将本文方案和其他方案进行了对比,结果如表 7—表 9 所列。本文使用 LUT 实现 S 盒,与文献[25]相比达到了相近的频率,均为 300 MHz 以上,验证结果表明本文方案的吞吐率大于文献[25]中的方案。与文献[26]和文献[27]相比,本文方案具有较高的频率和吞吐率。

从表 9 中可以看出,本文实现的 SM4-GCM 算法具有较高的频率和吞吐率。而对于文献[4],其以乒乓操作交替使用了两个 SM4 模块,获取了更高的吞吐率。本文为了获得更好的资源、性能、功耗间的平衡,仅使用了 1 个 SM4 模块,仍达到了较高的吞吐率。并且文献[4]的实现参数固定,难以适应多变的应用场景,本文提出的方案参数可配置,易于扩展移植。

表 7 SM4 模块与其他方案对比

Table 7 Comparison of SM4 module with other solutions

方案	器件	FFs	LUTs	RAMs	频率/MHz	吞吐率/Gbps
文献[25]中的方案	KINTEX-7 XC7K325TFFG900	7736	3213	33	345.00	42.10
文献[26]中的方案	KINTEX-7 XC7K325T-2	—	18643	—	250.00	31.50
文献[27]中的方案	Stratix IV EP4SE230F29I3	5438	7667	—	212.13	27.15
本文方案	ZYNQ XC7Z035FFG676-2	8483	14427	0	340.00	43.52

表 8 GHASH 模块与其他方案对比

Table 8 Comparison of GHASH module with other solutions

方案	器件	FFs	LUTs	RAMs	频率/MHz	吞吐率/Gbps
文献[28]中的方案	Virtex-5 xc5vlx50t	430	9405	0	120.17	15.38
本文方案一	ZYNQ XC7Z035FFG676-2	3972	8350	11	220.00	28.16
本文方案二	ZYNQ XC7Z035FFG676-2	1471	5406	0	301.93	28.98

表 9 SM4-GCM 与其他方案对比

Table 9 Comparison of SM4-GCM with other solutions

方案	算法	器件	FFs	LUTs	RAMs	频率/MHz	吞吐率/Gbps
文献[4]中的方案	SM4-GCM	Virtex-5 xc5vlx220	10609	10609	0	250.6	32.10
文献[29]中的方案	AES-GCM	Virtex-7 X485T	38241	—	0	119.0	15.24
本文方案一	SM4-GCM	ZYNQ XC7Z035FFG676-2	12579	22856	0	220.0	28.16
本文方案二	SM4-GCM	ZYNQ XC7Z035FFG676-2	11528	20296	2	300.0	28.80

注:文献[29]以 Slices 评估资源,对于本文方案一为 6062,方案二为 7210

结束语 本文针对数据吞吐率、灵活性要求高的应用场景,对 SM4-GCM 算法提出了两种优化方案,即全流水线型结构和基于四度并行优化的 SM4-GCM 结构,两种方案单个 SM4-GCM 模块的吞吐率均高于国内外已发表的同类设计。方案一消耗了 58 291 个资源,实现了 28.16 Gbps 的吞吐率,在密钥连续变化的场景下运算性能不会降低,适合应用在服务端,且方案一参数可配置,灵活性较高;方案二消耗了 52 120 个资源,实现了 28.8 Gbps 的吞吐率,在密钥不常变化的场景下整体吞吐率高于方案一,适合应用在客户端,具有较高的实际应用价值。

本文提到的两种方案均可在同一 FPGA 芯片上配置多个相同模块并行计算,进一步提高了吞吐率,用于适配 5G 大数据应用场景。下一步将针对本文提出的方案,持续研究降低 SM4-GCM 电路资源消耗及功耗的方法,在保证高吞吐率的同时进一步降低了计算成本。

参 考 文 献

[1] GB/T 32907-2016 Information Security Technology SM 4 Block Cipher Algorithm [S]. Beijing:China Standard Press,2016.

[2] IEEE Std 802.1AEbn. IEEE Standard for Local and Metropolitan Area Networks-Media Access Control (MAC) Security Amendment 1: Galois Counter Mode-Advanced Encryption Standard-256(GCM-AES-256) Cipher Suite, September 2011, [OL]. http://www.ieee802.org/1/pages/802.1aebn.html.

[3] FU T S,LI S G. A High-throughput ASIC implementation of SM4 algorithm CBC mode[J]. Microelectronics and Computer, 2016,33(10):13-18.

[4] LI L,YANG F,PAN Y M,et al. An implementation method for SM4-GCM on FPGA[C]// 2017 IEEE 2nd Advanced Information Technology,Electronic and Automation Control Conference (IAEAC). IEEE,2017:1921-1925.

[5] CHENG W Z,ZHENG F Y,PAN W Q,et al. High-performance symmetric cryptography server with GPU acceleration[C]// International Conference on Information and Communications Security. Cham:Springer,2017:529-540.

[6] WANG Z F,TANG Z J. High-throughput ASIC implementation of SM4 algorithm CTR mode[J]. Electronic Devices, 2019, 42(1):173-177.

[7] QIU S,BAI G Q. Power analysis of a FPGA implementation of

SM4[C]// Fifth International Conference on Computing, Communications and Networking Technologies (ICCCNT). IEEE, 2014: 1-6.
[8] LI J, XIE W B, LI L C, et al. Parallel Implementation and Optimization of SM4 Based on CUDA[C]// EAI International Conference on Applied Cryptography in Computer and Communications. Cham: Springer, 2021: 93-104.
[9] OSCAR F, SRINIVASAN S, RAMESH C, et al. A Survey on High-Throughput Non-Binary LDPC Decoders: ASIC, FPGA, and GPU Architectures[J]. IEEE Communications Surveys & Tutorials, 2021, 24(1): 524-556.
[10] LIU J J, SHI J J, ZHANG D J, et al. Hardware implementation and application of SM4 algorithm in wireless communication [J]. Computer Engineering and Applications, 2016, 52(17): 118-122.
[11] XU J F, YANG Y H. Parallel mapping of SM4 algorithm on coarse-grained array platform [J]. Application of Electronic Technology, 2017, 43(4): 39-42.
[12] ZHANG X, ZHOU Q L, LI B. Research and Implementation of Reconfigurable SM4 Cipher Algorithm Based on HRCA[J]. Journal of Network and Information Security, 2020, 6(5): 101-109.
[13] ZHANG J, WU W L. Authenticated encryption algorithm based on SM4 round function design [J]. Acta Electronica Sinica, 2018, 46(6): 1294-1299.
[14] SA'ED A, REEM J, BASSAM J M, et al. Performance evaluation of the SM4 cipher based on field-programmable gate array implementation[J]. IET Circuits, Devices & Systems, 2021, 15(2): 121-135.
[15] MOZAFFARI-KERMANI M, REYHANI-MASOLEH A. Efficient and high-performance parallel hardware architectures for the AES-GCM[J]. IEEE Transactions on Computers, 2011, 61(8): 1165-1178.
[16] SANDHYA K, AMITABH D, KESHAB K P. FPGA implementation and comparison of AES-GCM and Deoxys authenticated encryption schemesp[C]// 2017 IEEE International Symposium on Circuits and Systems(ISCAS). 2017: 1-4.
[17] ZHANG Z, WANG X, HAO Q, et al. High-efficiency parallel cryptographic accelerator for real-time guaranteeing dynamic data security in embedded systems [J]. Micromachines, 2021, 12(5): 560-584.
[18] KARIM M A, ROSELYNE C A HABIB M, et al. AES-GCM and AEGIS: Efficient and High Speed Hardware Implementations[J]. Journal of Signal Processing Systems, 2017, 88(1): 1-12.
[19] AHMAD, NABIHAH, LIM M W, et al. Advanced Encryption Standard with Galois Counter Mode using Field Programmable Gate Array[J]. Journal of Physics: Conference Series, 2018, 1019(1): 1-7.
[20] LI Y, MA X P, ZHANG Y, et al. Mastrovito form of non-recursive Karatsuba multiplier for all trinomials[J]. IEEE Transactions on Computers, 2017, 66(9): 1573-1584.
[21] GUERON S, KOUNAVIS M. Efficient implementation of the Galois Counter Mode using a carry-less multiplier and a fast reduction algorithm [J]. Information Processing Letters, 2010, 110(14): 549-553.
[22] SUNAR B, KOC C K. Mastrovito multiplier for all trinomials [J]. IEEE Transactions on Computers, 1999, 48(5): 522-527.
[23] HALBUTOGULLARI A, KOC C K. Mastrovito multiplier for general irreducible polynomials[J]. IEEE Transactions on Computers, 2000, 49(5): 503-518.
[24] SKOWYRA R, XU L, GU G F, et al. Effective topology tampering attacks and defenses in software-defined networks[C]// 2018 48th Annual IEEE/IFIP International Conference on Dependable Systems and Networks(DSN). IEEE, 2018: 374-385.
[25] HE S Y, LI H, LI F H. The FPGA optimization implementation method of SM4 algorithm [J]. Journal of Xidian University, 2021, 48(3): 155-162.
[26] YANG G Q, DING H C, ZOU J, et al. A big data security scheme based on high-performance cryptography[J]. Computer Research and Development, 2019, 56(10): 2207-2215.
[27] GUAN Z, LI Y, SHANG T, et al. Implementation of SM4 on FPGA: Trade-off analysis between area and speed[C]// 2018 IEEE International Conference on Intelligence and Safety for Robotics(ISR). IEEE, 2018: 192-197.
[28] QU S X. Research and implementation of GCM encryption authentication algorithm based on FPGA[D]. Beijing: Beijing University of Posts and Telecommunications, 2010.
[29] VLIEGEN J, REQARAZ O, MENTENS N. Maximizing the throughput of threshold-protected AES-GCM implementations on FPGA[C]// 2017 IEEE 2nd International Verification and Security Workshop(IVSW). IEEE, 2017: 140-145.

ZHAI Jia-qi, born in 1998, postgraduate. His main research interests include high-performance computing and information security.

LI Bin, born in 1986, Ph.D, lecturer. His main research interests include high-performance computing and information security.

(责任编辑:何杨)