第 37 卷 第 24 期
Vol.37 No.24

计 算 机 工 程
Computer Engineering

2011 年 12 月
December 2011

• 安全技术 •

文章编号：1000—3428(2011)24—0136—02 文献标识码：A 中图分类号：N945

基于 COS 的 Hash 接口设计与实现

郑 斌，李 峥，王瑞蛟

(信息工程大学电子技术学院，郑州 450004)

摘 要：基于片上操作系统(COS)的 Hash 函数可扩展性较差。针对该问题，提出一种可重构的 Hash 接口方法。该方法引入面向对象的概念，由 Hash 算法接口与 Hash 算法设置接口 2 个部分组成，利用存储在 EEPROM 中的 Hash 算法设置接口对 Hash 算法接口进行实例化，使之具备密码服务功能。验证结果表明，该方法具有较强拓展性，能够达到预期设计目标。

关键词：Hash 算法；可重构；密码服务；算法接口

Design and Implementation of Hash Interface Based on COS

ZHENG Bin, LI Zheng, WANG Rui-jiao

(Electronic Technology Institute, Information Engineering University, Zhengzhou 450004, China)

【Abstract】To solve the problem of the Hash algorithm expansibility based on the Chip Operating System(COS), a flexible Hash interface is designed. The interface which takes the object-oriented thought is made up by Hash algorithm interface and Hash algorithm setting interface. The Hash algorithm interface is set by the Hash algorithm setting interface, which is stored in the EEPROM, to be an instance and has the capability to provide the cryptographic service. The results of experiment show that Hash interface has good expansibility to add other algorithms, and get the purpose to design it.

【Key words】Hash algorithm; reconfigurable; cryptographic service; algorithm interface

DOI: 10.3969/j.issn.1000-3428.2011.24.045

1 概述

片上操作系统(Chip Operating System, COS)由于其与硬件的良好兼容性、资源占用少、实现效率高等诸多优点被广泛应用于信息安全领域，可以为用户提供数据加解密、身份认证、数字签名等方便可靠的密码服务。但当前应用的 COS 中的密码服务不完善，过于关注于对密码算法函数的使用和调度，没有形成完善的服务体系设计，使密码算法的设置、使用、更新极为不便。

本文参照 ISO/IEC7816-4、ISO/IEC7816-8 标准[1-2]中密码服务系统，以面向对象和可重构思想，对密码服务体系设计方法进行探索，实现一种可重构的 Hash 接口，可以通过设置该算法接口方便的重构各种 Hash 算法，完成数据压缩和 HMAC 码计算等密码服务功能[3-4]。

2 接口的总体设计

通常基于 COS 的密码服务重点关注于密码算法函数的实现效率，但忽视了函数的使用环节。导致函数调用过程复杂、不够灵活，特别是影响密码算法的扩展性，使得密码系统功能局限于初始设计，不能根据用户需求进行算法函数的拓展，如文献[5]方案。针对当前 COS 密码服务存在的问题，本文以 Hash 算法为例进行了探索性研究。具体解决方案是在低层密码函数与上层应用之间加入了中间接口层，该接口主要是对算法函数接口和相关参数进行封装、设置，并为上层密码应用提供灵活的调用接口。该文对 Hash 密码服务中间层设计分为 2 个部分，Hash 算法接口结构和 Hash 设置接口结构，如图 1 所示。外部命令通过调用设置接口结构对算法结构进行实例化设置，使之具备不同的算法服务能力，为上层密码应用提供数据压缩和 HMAC 计算等密码服务。为增强算法的扩展功能，所有接口结构和相关参数均采用标准化设计，并将设置接口以函数数组的形式存放在 EEPROM 中。当底层算法更新或下载时，只要更新设置接口的函数数组就能够调用下载的新 Hash 算法，使 COS 的密码服务能力的扩展性极大提高。

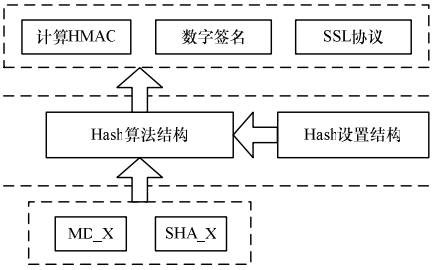

图 1 Hash 接口结构

3 Hash 算法接口设计

Hash 算法接口由算法结构和算法操作 2 个部分组成，按照面向对象的思想将 Hash 算法相关参数组织到算法结构中，并按照标准流程完成密码服务工作。下载更新的算法必须按照如下参数定义和算法流程实现，否则 COS 不能调用。

3.1 算法接口结构

Hash 算法接口结构主要有初始值 IV、分组长度和压缩函

基金项目：国家自然科学基金资助项目(61072047)；现代通信国家重点实验室基金资助项目(9140C1106021006)；郑州市创新型科技人才队伍建设工程基金资助项目(096SYJH21099)

作者简介：郑 斌(1985－)，男，硕士，主研方向：密码学，软件工程；李 峥，副教授、博士；王瑞蛟，硕士

收稿日期：2011-06-21 **E-mail：**countsinbad@163.com

数入口等参数，并加入操作状态等辅助变量。考虑到编译平台的兼容性问题采用函数指针的形式设计接口方法。具体结构如下：

```
typedef struct _Class_Hash{
INT          OprState;
INT          HshBlkLen;
INT          MsgBlkLen;
INT          HashValue[MAX_HashLEN];
DOUBLE           dwByteCounter;
INT          (*pHash)(INT byOpMode, const INT *pbyInData, INT byInDataLen );
} Class_Hash;
```

接口结构中各参数含义如下：

(1)OprState 指示当前 Hash 函数操作状态。各比特位定义为：

1)OprState.b7b6：00 表示初始化状态，01 表示数据更新状态，10 表示结束状态；

2)OprState.b5：0 表示可链接模式，1 表示不可链接模式，用于指示 Hash 值长度不等于初始值的情况；

3)OprState.b4b3b2：由算法自行定义；

4)OprState.b1b0：用于将来使用。

(2)HshBlkLen 指示 Hash 函数输出值长度。

(3)MsgBlkLen 指示 Hash 函数输入消息分组长度。

(4)HashValue[MAX_HashLEN]为数据存储空间，大小根据硬件平台资源情况设定，按照所处状态不同，存储初始值、中间值和 Hash 值。

(5)dwByteCounter 用于记录当前函数所进行运算次数。

(6)(* pHash)(INT byOpMode, const INT *pbyInData, INT byInDataLen)为算法接口方法，即当前函数的指针。各参数定义如下：

1)byOpMode 指示操作模式，有 3 种类型：

①初始化：OpInit=0x01；

②中间数据块操作：OpUpdate=0x02；

③最后数据块操作：OpFinal=0x03。

2)*pbyInData 指示输入数据地址；

3)byInDataLen 指示输入数据长度。

3.2　算法接口操作流程

算法接口函数是整个算法接口结构的核心，是在对常用的 Hash 算法基本操作步骤分析总结的基础上而设计[6]。操作流程中定义一个全局空指针变量 gpThis，方便于对接口结构相关数据的操作。算法操作流程如下：

Step1 若 byOpMode==OpInit，则转向 Step1’；否则转向 Step2。

Step1’ 将(Class_Hash*)gpThis->pHash 算法的 IV 值赋予(Class_Hash*)gpThis->HashValue[]，并设置(Class_Hash *)gpThis -> OprState.b7b6=01。完成，返回成功；否则，返回失败。

Step2 若 byOpMode== OpUpdate，则转向 Step2.1；否则转向 Step3。

Step2.1 根据 InDataLen 的指示，消息长度为杂凑消息长度的整数倍，则转向 Step2.2；否则返回错误。

Step2.2 对输入消息进行运算，将计算的中间值存放在存储区(Class_Hash*)gpThis->HashValue[]中，且(Class_Hash*)gpThis->dwByteCounter 对操作进行计数，并设置(Class_Hash*)gpThis->OprState.b7bt6=10。完成，返回成功；否则，返回失败。

Step3 若 byOpMode==OpFinal，则转向 Step3’；否则返回错误。

Step3’ 若 InDataLen 长度不为(Class_Hash *) gpThis->MsgBlkLen，则按照 Hash 函数填充方法(一般是对消息进行 MD 增强)对数据进行最后的填充，并对填充后消息进行操作，将结果存放在(Class_Hash*)gpThis->HashBlkLen 长度的存储区(Class_Hash*)gpThis->HashValue[]中，且令(Class_Hash *)gpThis->OprState.bt7b6=00。完成返回成功；否则，返回失败。

4　Hash 算法设置接口设计

在 Hash 算法接口结构设计的基础上，按照可重构设计思想，设计了 Hash 算法设置结构和设置接口操作流程，用于对算法接口进行设置。每个算法拥有自己的 Hash 算法设置结构，但都要按照所述的标准流程进行实现和操作，并存储在 EEPROM 中由 COS 进行管理和维护。

4.1　设置接口结构

为了节省存储空间，设置接口结构进行简化处理，考虑到编译平台的兼容问题采用函数指针的形式设计接口。具体结构如下：

```
typedef     struct   _Intrfc_ Hash{
INT         byAttribute;
INT         (*pIntfcHash)(INT byIniMode，INT byAuxPara);
}   Intrfc_ Hash;
```

接口结构中各参数含义如下：

(1)byAttribute 指示接口属性。各比特位定义如下：

1)byAttribute.b7：1 表示算法存在，0 表示算法不存在；

2)byAttribute.b6：1 表示分组长度可变，0 表示分组长度不可变；

3)byAttribute.b5b4b3b2：将来使用；

4)byAttribute.b1b0：00 表示 COS 内部算法，01 表示下载算法，02 表示算法 IP 核。

(2)(* pIntfcHash)(INT byIniMode，INT byAuxPara)设置接口方法，即设置接口函数指针。各参数定义如下：

1)byIniMode：指示算法的应用类型。计算输入数据的 Hash 值：HashMode=0x00；验证算法正确性：SELFTST=0x01。

2)byAuxPara：辅助设置参数，由于算法自行解释。

4.2　设置接口操作流程

设置接口操作主要是将其对应的 Hash 算法设置为当前 Hash 接口方法，并设置其他相关参数。具体流程如下：

Step1 若 byIniMode == HashMode，则转向 Step1’；否则转向 Step2。

Step1’ 若算法相应模式存在，则令(Class_Hash*)gpThis->pHash=Hash 算法函数入口；(Clash_Hash*)gpThis->HshBlkLen=Hash 算法分组长度；(Class_Hash*)gpThis->OprState.b7b6=00 并根据 byAuxPara 指示及算法特点设置(Class_Hash*)gpThis->OprState 其他比特位；(Class_Hash*)gpThis -> dwByteCounter=0；

设置完成，返回成功；否则返回错误。

Step2 若 byIniMode==SELFTST，则转向 Step2’；否则返回错误。

Step2’ 算法自检。完成，返回成功；否则返回错误。

4.3　MD5 算法调用实例

将本文所设计的 Hash 算法接口在 AT90SC6464 芯片平台

(下转第 140 页)