

ICS 35.040

L 80

备案号



中华人民共和国密码行业标准

GM/T 0021—2012

动态口令密码应用 技术规范

One time password application of
cryptography algorithm

2012-11-22 发布

2012-11-22 实施

国家密码管理局 发布

GM/T 0021—2012

中华人民共和国密码

行业标准

动态口令密码应用

技术规范

GM/T 0021—2012

*

中国标准出版社出版发行

北京市朝阳区和平里西街甲2号(100013)

北京市西城区三里河北街16号(100045)

网址 www.spc.net.cn

总编室:(010)64275323 发行中心:(010)51780235

读者服务部:(010)68523946

中国标准出版社秦皇岛印刷厂印刷

各地新华书店经销

*

开本 880×1230 1/16 印张 0.00 字数 00 千字

2012年 月第一版 2012年 月第一次印刷

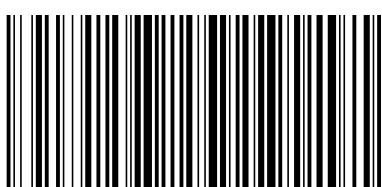
*

书号: 155066·2 定价 00.00 元

如有印装差错 由本社发行中心调换

版权专有 侵权必究

举报电话:(010)68510107



MG/T 0021-2012

目 次

前言
1 范围
2 规范性引用文件
3 术语
4 符号
5 动态口令系统
5.1 概述
5.2 总体框架
5.3 基本认证原理简述
6 动态口令生成方式
6.1 概述
6.2 算法使用说明
6.3 截位算法
7 动态令牌特性
7.1 令牌硬件要求
7.2 令牌安全特性
8 认证系统
8.1 系统说明
8.2 认证系统服务
8.3 认证系统管理功能
8.4 安全要求
9 密钥管理系统
9.1 概述
9.2 系统架构
9.3 功能要求
9.4 系统安全性设计
9.5 硬件密码设备接口说明
附录 A (资料性附录) 动态口令生成算法 C 语言实现用例
A.1 采用 SM3 的动态口令生成算法用例
A.2 采用 SM4 的动态口令生成算法用例
附录 B (资料性附录) 动态口令生成算法计算输入输出用例
B.1 采用 SM3 的动态口令生成算法输入输出用例
B.2 采用 SM4 的动态口令生成算法输入输出用例
附录 C (资料性附录) 运算参数与数据说明用例
附录 D (资料性附录) 认证系统接口

- D. 1 服务报文格式
- D. 2 服务标识
- D. 3 数据标识
- D. 4 返回码
- D. 5 应用接口

前　　言

本标准依据 GB/T 1.1—2009 给出的规则起草。

请注意本标准的某些内容可能涉及专利。本文件的发布机构不承担识别这些专利的责任。

本标准中的附录 A、附录 B、附录 C、附录 D 为资料性附录。

本标准由国家密码管理局提出并归口。

本标准起草单位：上海众人网络安全技术有限公司，上海复旦微电子股份有限公司，飞天诚信科技股份有限公司，北京集联网络技术有限公司，上海华虹集成电路有限责任公司，深圳同方电子设备有限公司，上海林果实业有限公司，上海格尔软件股份有限公司。

本标准主要起草人：詹榜华、谈剑锋、尤磊、柳逊、陈达、郭思建、张志茂、李阗、牛毅。

本标准凡涉及密码算法相关内容，按照国家有关法规实施。

动态口令密码应用 技术规范

1 范围

本标准规定了动态口令系统,动态口令生成方式,动态令牌特性,认证系统,密钥管理系统等的相关内容。

本标准适用于动态口令相关产品的研制、生产,也可用于指导相关产品的检测。

2 规范性引用文件

下列文件对于本文件的应用是必不可少的。凡是注日期的引用文件,仅所注日期的版本适用于本文件,凡是不注日期的引用文件,其最新版本(包括所有的修改单)适用于本文件。

- GB/T 2423.1—2008 电工电子产品环境试验 第2部分:试验方法 试验A:低温
- GB/T 2423.2—2008 电工电子产品环境试验 第2部分:试验方法 试验B:高温
- GB/T 2423.8—1995 电工电子产品环境试验 第2部分:试验方法 试验Ed:自由跌落
- GB/T 2423.9—2001 电工电子产品环境试验 第2部分:试验方法 试验Cb设备用恒定湿热
- GB/T 2423.10—2008 电工电子产品环境试验 第2部分:试验方法 试验Fc:振动(正弦)
- GB/T 2423.21—1991 电工电子产品基本环境试验规程:试验M:低气压试验方法
- GB/T 2423.22—2002 电工电子产品环境试验 第2部分:试验方法 N:温度变化
- GB/T 2423.53—2005 电工电子产品环境试验 第2部分:试验方法 试验Xb由手的摩擦造成标记和印刷文字的磨损
- GB/T 4208—2008 外壳防护等级(IP 代码)
- GB/T 17626.2—2006 电磁兼容试验和测量技术 静电放电抗扰度试验
- GB/T 18336.1—2008 信息技术 安全技术 信息技术安全性评估准则 第1部分:简介和一般模型
- GB/T 18336.2—2008 信息技术 安全技术 信息技术安全性评估准则 第2部分:安全功能要求
- GB/T 18336.3—2008 信息技术 安全技术 信息技术安全性评估准则 第3部分:安全保证要求
- GB/T 21079.1—2007 银行业务 安全加密设备(零售) 第1部分 概念、要求和评估方法
- GM/T 0002—2012 SM4 分组密码算法
- GM/T 0004—2012 SM3 密码杂凑算法
- GM/T 0005—2012 随机数检测规范

3 术语

下列术语和定义适用于本文件。

3.1

动态令牌 dynamic password token/one time password token

生成并显示动态口令的载体。

3.2

动态口令 dynamic password/one time password

由种子密钥与其它数据,通过特定算法,运算生成的一次性口令。

3.3

静态口令 static password

用户设置的,除非用户主动修改,否则不会发生变化的口令。

3.4

挑战码 challenge code

即挑战因子,可参与到动态口令生成过程中的一种数据。

3.5

UTC 时间 Universal Time Coordinated

协调世界时(Universal Time Coordinated)英文缩写,是由国际无线电咨询委员会规定和推荐,并由国际时间局(BIH)负责保持的以秒为基础的时间标度,是距 1970 年 1 月 1 日 00:00 时(格林尼治标准时间)的秒数。

3.6

种子密钥 seed key

即令牌种子密钥,计算动态口令的密钥。

3.7

大端 big-endian

数据在内存中的一种表示格式,规定左边为高有效位,右边为低有效位。数的高阶字节放在存储器的低地址,数的低阶字节放在存储器的高地址。

3.8

认证系统 authentication system

能够为应用系统提供动态口令身份认证服务的系统。

3.9

未激活 not activated

本状态为出厂时状态,成功激活后进入就绪状态。

3.10

就绪 ready

令牌为正常工作状态。

3.11

锁定 be locked

令牌因连续错误、重放攻击等原因被锁定后处于锁定状态。

3.12

挂起 hung up

令牌被人为挂起后,处于挂起状态。

3.13

作废 invalidate

令牌执行作废操作后,进入作废状态。

3.14

自动解锁 automatically unlock

令牌被锁定以后,经过一定时间,系统会将令牌的锁定状态解除。

3.15

密钥管理 key management

根据安全策略,对密钥的产生、登记、认证、注销、分发、安装、存储、归档、撤销、衍生和销毁等操作制定并实施一组确定的规则

3.16

硬件密码设备 hardware encryption device

一种用于密钥管理、加解密运算等功能的硬件载体。

3.17

密钥 key

控制密码变换操作的关键信息或参数。

3.18

服务报表 service list

系统提供的,对于令牌和系统不同时间段对应的状态和结果的统计报表。

3.19

接口 interface

两个不同系统(或子程序)交接并通过它彼此作用的部分。

3.20

大窗口 large window

用于令牌时间与系统时间同步的窗口,窗口大小不应超过±10。

3.21

中窗口 middle window

用于令牌时间与系统时间同步的窗口,窗口大小不应超过±5。

3.22

小窗口 small window

用于令牌时间与系统时间同步的窗口,窗口大小不应超过±2。

3.23

种子密钥加密密钥 encryption key for seed key

用于对种子密钥进行加密的密钥。

3.24

厂商生产主密钥 main key for manufacturer production

生成令牌生产时所需的种子密钥加密密钥。

3.25

主密钥 main key

系统的根密钥,可用于生成种子密钥,种子密钥加密密钥,厂商生产主密钥等。

3.26

厂商代码 manufacturer code

用于标识厂商的代码,可以是数字、英文字母、数字与英文字母的混合。

3.27

内部挑战认证 internal challenge authentication

一种由用户主动解除令牌异常状态时采用的挑战认证方式。

4 符号

IP44

防护等级第一特性4(防尘),防护等级第二特性4(防水)。

bitN	指一个字节当中的第 N+1 个比特位(从低位计起)
PIN	指用于使令牌工作并显示动态口令的一种口令,是一个至少 6 位长度的十进制数。
T ₀	是以 UTC 时间为标准的一个长度为 8 字节的整数
T	参与运算的时间因子
T _c	以秒为单位的口令变化周期
ID	杂凑及分组算法的输入信息
C	参与运算的事件因子
Q	认证双方通过协商输入的挑战因子
K	长度不少于 128 比特的种子密钥
S	算法函数输出结果
F()	算法函数
OD	输出结果
Truncate()	截位函数
N	令牌或其它终端显示口令的位数
K _m	主密钥
K _t	传输密钥
K _p	厂商生产主密钥
K _s	种子密钥加密密钥
ⁿ	幂运算符,即 2^n 代表 2 的 n 次方
%	求余运算,即 $5 \% 3 = 2$
<<	循环左移符号
	连接符,将两组数据根据左右顺序拼接
#	算术加符号,且不进位。

5 动态口令系统

5.1 概述

动态口令系统包含动态令牌和动态令牌认证系统,可以为应用系统提供动态口令认证服务。动态令牌认证系统由认证系统和密钥管理系统组成。

5.2 总体框架

动态令牌负责生成动态口令,认证系统负责验证动态口令的正确性,密钥管理系统负责动态令牌的密钥管理,应用系统负责将动态口令按照指定的协议(报文)发送至认证系统进行认证。动态口令系统架构图如图 1 所示:

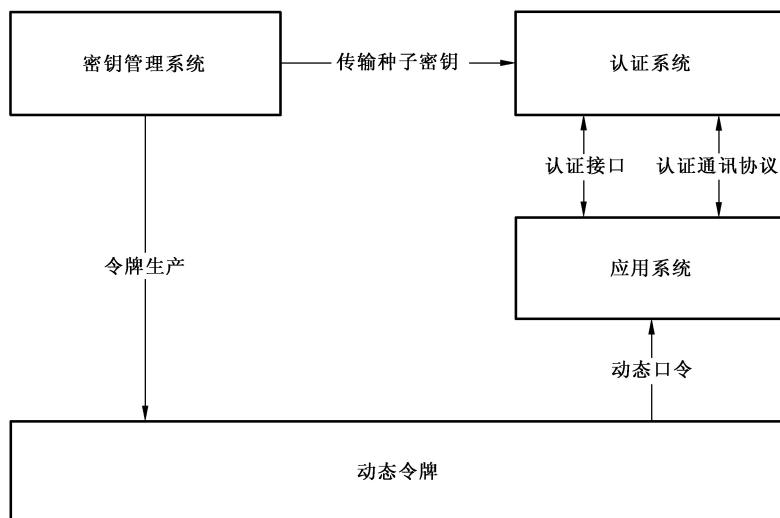


图 1 系统架构

5.2.1 动态令牌

动态令牌是一种生成动态口令的设备,每次生成的口令各不相同。

5.2.2 认证系统

认证系统用于执行动态口令认证和令牌同步,以及令牌相关状态的管理及配置等功能的服务集合。认证系统通过认证通讯协议与应用系统进行通讯或支持调用认证接口的方式,用以完成动态口令的认证、同步等功能。

5.2.3 密钥管理系统

密钥管理系统用于动态令牌种子密钥的生成、传输和存储的安全管理系统,包括系统登录认证模块、用户管理模块、保护密钥生成模块、种子密钥生成模块、令牌序号生成模块、时间同步模块(可选)、令牌生产配置模块、密码机接口模块和动态令牌读写接口等模块。

5.2.4 应用系统

应用系统是指集成了将动态口令按照认证通讯协议(或认证接口)与认证系统交互完成动态口令认证的应用程序集合,应用系统可以是软件系统,也可以是硬件设备,还可以是软件和硬件相结合的形式。

5.2.5 认证接口

认证接口是认证系统提供的用于连接应用系统与认证服务器的接口集合。开发语言包括但不限于C/C++、Java、php、ASP、ASP.NET、C#等。应用系统通过调用接口,可以完成动态口令认证、同步等功能。

5.2.6 认证通讯协议

认证通讯协议是认证服务通过标准的通讯协议和应用系统进行通讯的依据。应用系统通过发送报文的形式完成动态口令的认证、同步等功能。

5.3 基本认证原理简述

动态口令的基本认证原理,是通过用户端与认证服务提供端,以相同的运算因子,采用相同的运算

方法,生成相同的口令,并进行比对,来完成整个认证过程。通常,口令的比对是由认证服务提供端完成。具体如图 2 所示:

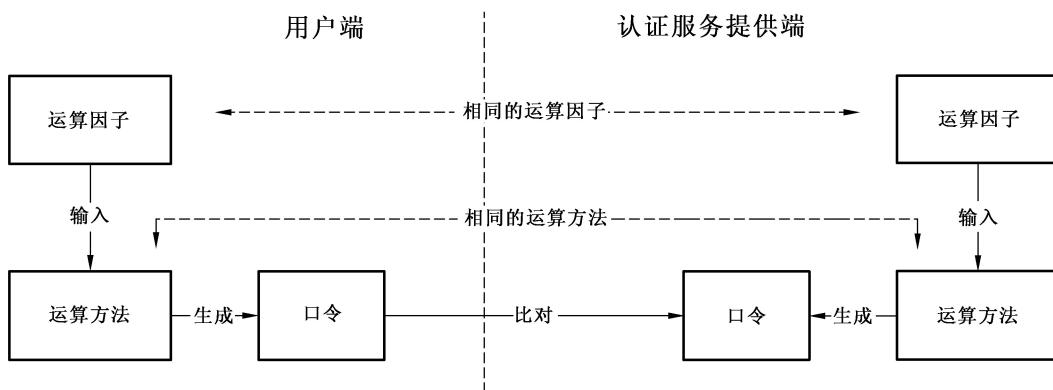


图 2 基本认证原理图

6 动态口令生成方式

6.1 概述

本标准的动态口令生成方式概述,如下所示:

$$T = T_0 / T_c$$

$$ID = \{T | C | Q\}$$

$$S = F(K, ID)$$

$$OD = \text{Truncate}(S)$$

$$P = OD \% (10^N)$$

T 是参与运算的时间因子,是一个 8 字节整数, T_0 是以 UTC 时间或用户选择的时间标准为计量标准的一个 8 字节整数, T_c 是以秒为单位的口令变化周期,最大长度为 60 秒; ID 是杂凑及分组算法的输入信息; C 是参与运算的事件因子,是一个 4 字节整数; Q 是认证双方通过协商输入的挑战因子或其它类型需要参与运算的因子,使用 ASCII 码表示,最小长度为 4 字节。 ID 最小长度为 128 比特,至少需要包含 T 、 C 的其中一个参数, Q 为可选参数,并按照 $T | C | Q$ 的顺序进行数据组装。未包含的参数位置,由下一个参数进行补充。如 ID 由 T 、 Q 组成,则数据组装方式为 $T | Q$ 。如 ID 由 C 、 Q 组成,则数据组装方式为 $C | Q$ 。如组成 ID 的数据不足 128 比特, ID 的数据末端填零至 128 比特。

K 是长度不少于 128 比特的种子密钥,只有认证双方持有; $F()$ 是算法函数,即 SM4 或 SM3,见 GM/T 0002—2012 和 GM/T 0004—2012。

S 是算法函数输出结果,SM4 算法的输出结果长度为 128 比特,SM3 算法的输出结果(即杂凑值)长度为 256 比特。

$\text{Truncate}()$ 是截位函数, OD 是其输出结果,长度为 32 比特。

N 是令牌或其它终端显示口令的位数, N 不小于 6。 P 是最终显示的动态口令。

6.2 算法使用说明

6.2.1 使用要求

本标准涉及动态口令生成方式的分组算法,应选用 SM4 分组算法,应满足 GM/T 0002—2012 的要求。分组长度为 128 比特,密钥长度为 128 比特,运算方式选用加密方式,加密结果长度为 128 比特。

本标准涉及动态口令生成方式的杂凑算法,应选用 SM3 杂凑算法,应满足 GM/T 0004—2012 的要求。本标准选用的杂凑值长度为 256 比特。

本标准涉及随机数生成的算法,应满足 GM/T 0005—2012 的要求。

6.2.2 SM3 杂凑算法使用说明

在 $S=F(K, ID)$ 环节,当使用 SM3 算法时,K|ID 为输入参数

6.2.3 SM4 分组算法使用说明

在 $S=F(K, ID)$ 环节,当使用 SM4 算法时,K 为运算密钥, ID 为输入参数,K 或 ID 大于 128 比特时,运算过程如下:

K 大于 128 比特时,只取其中从高位计起的前 128 比特数据,作为 K 参与运算。

ID 大于 128 比特时,在 ID 末端填零至 128 比特的整数倍长度。再将 ID 以 128 比特长度进行分组,高位在前,分别为 $ID_1, ID_2, ID_3, \dots, ID_m$ 。

运算过程如图 3 所示:

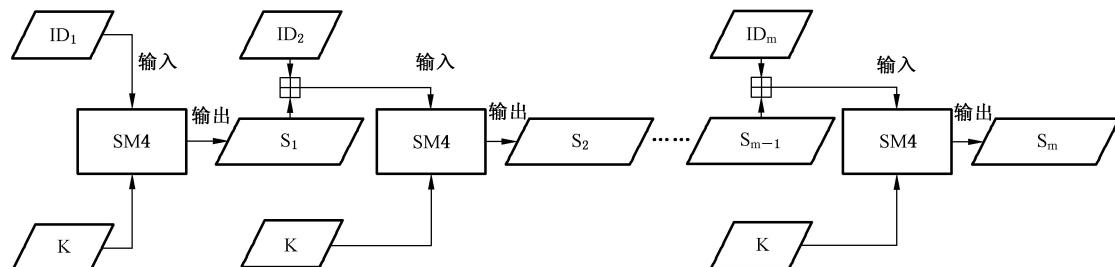


图 3 SM4 运算过程图

ID_1 和 K 作为第一个分组运算的输入,通过 SM4 运算生成 S_1 。将 S_1 与 ID_2 进行算术加运算(高位溢出舍去),其结果与 K 一起用于第二个分组的输入,通过 SM4 运算生成 S_2 。之后的运算以此类推。

如图 3 中所示, $S=S_m$ 。

6.2.4 数据格式

数据运算与存储,均采用大端(big-endian)格式。可参考附录 C。

6.3 截位算法

Truncate()是动态口令生成过程中,使用的截位函数。其实现分为杂凑结果截位和分组结果截位。杂凑结果截位实现方式如下:

定义 $S_1, S_2, S_3, S_4, S_5, S_6, S_7, S_8$,表示 8 个 4 字节整数,通过如下方法赋值:

```

S1=S[0]<<24|S[1]<<16|S[2]<<8|S[3]
S2=S[4]<<24|S[5]<<16|S[6]<<8|S[7]
S3=S[8]<<24|S[9]<<16|S[10]<<8|S[11]
S4=S[12]<<24|S[13]<<16|S[14]<<8|S[15]
S5=S[16]<<24|S[17]<<16|S[18]<<8|S[19]
S6=S[20]<<24|S[21]<<16|S[22]<<8|S[23]
S7=S[24]<<24|S[25]<<16|S[26]<<8|S[27]
  
```

S8=S[28]<<24|S[29]<<16|S[30]<<8|S[31]

OD=(S1+S2+S3+S4+S5+S6+S7+S8) MOD 2^32

分组结果截位实现方式如下：

定义 S1,S2,S3,S4,表示 4 个 4 字节整数,通过如下方法赋值:

S1=S[0]<<24|S[1]<<16|S[2]<<8|S[3]

S2=S[4]<<24|S[5]<<16|S[6]<<8|S[7]

S3=S[8]<<24|S[9]<<16 || S[10]<<8|S[11]

S4=S[12]<<24|S[13]<<16|S[14]<<8|S[15]

OD=(S1+S2+S3+S4) MOD 2^32

7 动态令牌特性

7.1 令牌硬件要求

令牌硬件产品特性最低要符合以下严酷条件要求,产品提高严酷等级时试验方法要遵守国标规定和本标准定义的判断标准。

7.1.1 高温

使用 GB/T 2423.2—2008 中试验方法 Bb,严酷等级选择温度: +50 °C,持续时间: 2 h。

7.1.2 低温

使用 GB/T 2423.1—2008 中试验方法 Ab,严酷等级选择温度: -10 °C,持续时间: 2 h。

7.1.3 高低温冲击

使用 GB/T 2423.22—2002,严酷等级选择高温温度: +50 °C,低温温度: -10 °C,暴露试验时间: 10 min,转换时间:(2~3)min,循环数:3 个。

7.1.4 湿度

使用 GB/T 2423.9—2001,严酷等级选择温度: 30 °C ± 2 °C,相对湿度 93% ± 3%,试验时间: 2 h。

7.1.5 工作海拔

使用 GB/T 2423.21—1991,严酷等级选择气压: 55 kPa,持续时间: 2 h。

7.1.6 跌落

使用 GB/T 2423.8—1995 中方法一,严酷等级选择跌落高度: 1 000 mm。

7.1.7 防尘防水

遵守 GB/T 4208—2008 中 IP44 的要求。

7.1.8 标记和印刷

使用 GB/T 2423.53—2005,试验液体:人工合成汗液,力的大小:1 N ± 0.2 N,循环次数: 1 000 次。产品必须具备标记文字为:“序列号”和“有效期”。

7.1.9 振动

使用 GB/T 2423.10—2008,严酷等级选择频率范围:10 Hz 到 300 Hz,振动幅值:3.5 mm,持续时间:60 min。

7.1.10 静电放电

不低于 GB/T 17626.2—2006 中试验等级 3 的标准,即满足外壳端口接触放电±6 kV,空气放电±8 kV。

7.1.11 试验过程中试验样品是否处于工作状态的判断标准

样品带电运行,目视检查,提供最少 12 个显示相同动态口令的样品,6 个一组为参与试验样品,6 个一组为不参与试验样品,参与试验的样品显示动态口令与不参与试验样品一致即为合格。

7.2 令牌安全特性

7.2.1 种子密钥安全

种子密钥为令牌重要安全要素,令牌必须保证产品内的种子密钥的完整性,且种子密钥为单向导入令牌(或在令牌内生成),种子密钥不能被导出产品外部。

令牌必须拥有种子密钥的保护功能。令牌种子密钥加密、存储、使用在令牌芯片的安全区域内实现。

7.2.2 令牌芯片安全

本标准涉及的令牌芯片,应是国家密码管理局批准产品型号证书的安全芯片。

7.2.3 令牌物理安全

令牌具有低电压检测功能。

令牌完成种子密钥导入后,通讯 I/O 端口应失效,不能再输入或输出信息,包括但不限于内部参与口令生成运算的 K、T、C 信息。

令牌应防范通过物理攻击的手段获取设备内的敏感信息,物理攻击的手段包括但不限于开盖、搭线、复制等。

令牌芯片必须具备令牌掉电后,会自动销毁种子密钥的措施。

令牌芯片应保证种子密钥无法通过外部或内部的方式读出,包括但不限于:调试接口、改编的程序。

令牌芯片可防范通过物理攻击的手段获取芯片内的敏感信息,确保种子密钥和算法程序的安全性。

令牌芯片应具备抵抗旁路攻击的能力,包括但不限于:抗 SPA/DPA 攻击能力,抗 SEMA/DEMA 攻击能力。

在外部环境发生变化时,令牌芯片不应泄漏敏感信息或影响安全功能。外部环境的变化包含但不限于:高低电压、高低温、强光干扰、电磁干扰、紫外线干扰、静电干扰。

7.2.4 使用安全

具有数字和功能按键的令牌必须具有 PIN 保护功能,PIN 长度不少于 6 位,并具有 PIN 防暴力穷举功能。令牌可具有 PIN 找回功能,即令牌用户如果忘记令牌 PIN,可通过安全有效的环境与机制找回令牌 PIN,或对令牌 PIN 进行重新设置(如远程解 PIN)。

PIN 输入错误的次数不可超过 5 次,若超过,应至少等待 1 h 才可继续尝试。

PIN 输入超过最大尝试次数的情况不可超过 5 次,否则令牌应永久锁定,不可再使用。

令牌基本使用寿命为 3 年,令牌产品最长使用不超过 5 年。

室温环境下,令牌时间偏差小于 2 min/年。

7.2.5 安全评估

动态令牌产品安全评估遵守以下国标规定:

GB/T 18336.1—2008

GB/T 18336.2—2008

GB/T 18336.3—2008

GB/T 21079.1—2007

8 认证系统

8.1 系统说明

8.1.1 系统构成

认证系统是为应用系统提供动态令牌认证及管理的服务系统,由两个部分构成:认证服务子系统、管理子系统:

- 1) 认证服务子系统对应用提供认证和管理服务。
- 2) 管理子系统对认证系统的运行进行管理。

8.1.2 令牌的系统状态

令牌的系统状态为认证系统内保存的令牌工作状态,如表 1 所示:

表 1 令牌工作状态表

状态	描述	说明
未激活	令牌出厂时的状态,须激活后令牌才能进入就绪状态	未激活令牌不能提供正常的口令认证
就绪	令牌正常工作状态	就绪状态下令牌可用于口令认证
锁定	令牌因连续错误、重放攻击、人工方式等原因被锁定后处于锁定状态	锁定状态的令牌不能提供正常的口令认证
挂起	令牌被人为挂起后,处于挂起状态	挂起状态的令牌不能提供正常的口令认证
作废	令牌执行作废操作后,进入作废状态	作废的令牌不能提供正常的口令认证

8.1.3 令牌的系统数据

令牌的系统数据应包括:令牌序列号、密钥数据、令牌状态、上次使用时间、连续错误次数、令牌偏移量、其它配置参数等,其中:

- 1) 密钥数据应加密存放。
- 2) 其它系统数据应采用校验机制保证完整性。

8.1.4 令牌的同步

认证系统应提供对令牌的内部计数器与系统的令牌计数器之间的同步处理。对于时间型令牌,使

用双向时间窗口；对于事件型令牌，使用单向事件窗口。

时间型令牌根据不同的令牌同步需求，分别采用大窗口、中窗口、小窗口进行同步。

使用各种窗口时，要求如下（窗口大小的单位是认证周期）：

大窗口，窗口大小不应超过±10。使用大窗口同步时，要求下一个连续的动态口令匹配，同时调整系统的令牌偏移量。大窗口要求使用受限的同步服务，即进一步身份认证或需要更高权限才能执行大窗口的同步服务。大窗口可由认证系统的授权运维人员使用，并应与应用系统的验证码机制同时使用。

中窗口，窗口大小不应超过±5。使用中窗口同步时，要求下一个连续的动态口令匹配，同时调整系统的令牌偏移量。中窗口可由令牌用户或认证系统的运维人员使用，并应与应用系统的验证码机制同时使用。

小窗口，窗口大小不应超过±2。使用小窗口同步时，认证系统通过动态口令认证，同时调整系统的令牌偏移量。小窗口可由认证系统自动调用，在令牌用户使用动态口令进行身份认证时使用。

事件型令牌的大窗口、中窗口、小窗口的大小，可由用户与厂商协商制定，但应保证其认证的安全性和有效性。

8.1.5 自动锁定和自动解锁

令牌在使用过程中若连续多次验证错误超过最大次数后，认证系统会自动将该令牌状态修改为“锁定”。在超过设定的自动解锁时间后，认证系统会自动解除该令牌的锁定状态。

自动解锁只能解除被自动锁定的令牌。

8.2 认证系统服务

认证系统服务是由认证服务子系统提供的，分为2个类别：安全服务和管理服务。安全服务包括动态口令认证、挑战应答认证、挑战码生成等，管理服务包括对令牌的生命周期管理。

8.2.1 安全服务

安全服务用于支持应用系统对用户身份和交易内容进行验证。

8.2.1.1 动态口令认证

对提交的动态口令进行认证的服务，认证方式包括：静态口令+动态口令、动态口令。其中静态口令为与该动态令牌绑定的静态口令。

8.2.1.2 挑战应答认证

对提交的挑战应答码进行认证的服务，认证方式包括：挑战认证、内部挑战认证。

挑战认证是用户采用向令牌输入应用服务提供的挑战码的方式，获取相应的动态口令，完成认证。

内部挑战认证是用户通过向令牌输入PIN、静态口令等用户私有数据，获取相应的动态口令，完成认证。

8.2.1.3 产生挑战码

根据应用的请求产生挑战码，生成的挑战码格式包括：数字型、字符型、数字+字符型。其中数字为阿拉伯数字0~9，字符为英文字符或符号字符，区分大小写。挑战码的最小长度和最大长度可由认证系统进行设置。

8.2.2 管理服务

管理服务用于对令牌进行生命周期管理，内容包括：激活、锁定、挂起、作废等。

8.2.2.1 激活

将未激活的令牌设为可用状态。激活时,要求验证令牌的动态口令。

- 1) 激活时验证动态口令的窗口使用大窗口。
- 2) 令牌成功激活后,状态设置为就绪。
- 3) 激活不成功,记录激活错误次数,但不锁定令牌。

8.2.2.2 锁定/解锁

锁定:将就绪状态的令牌设置为锁定状态。

- 1) 令牌被锁定后,可通过解锁服务回到就绪状态。
- 2) 令牌被锁定后,可通过废止服务设置为废止状态。

解锁:将锁定状态的令牌解锁,设置为就绪状态。

- 1) 解锁时,要求验证当前的动态口令。
- 2) 若设置了静态口令,要求验证静态口令。
- 3) 若静态口令的验证方式是内部挑战方式,使用内部挑战认证。
- 4) 若静态口令的验证方式是静动态混合方式,使用静态口令+动态口令认证。

8.2.2.3 挂起/解挂

挂起:将动态令牌设置为挂起状态。

- 1) 只有就绪或锁定状态的令牌可以被设置为挂起状态。
- 2) 令牌被挂起后,可通过废止服务设置为废止状态。

解挂:解除令牌的挂起状态。

- 1) 解挂成功后令牌的状态设置为就绪状态。
- 2) 要求验证当前的动态口令。
- 3) 若设置了静态口令,要求验证静态口令。
- 4) 若静态口令的验证方式是内部挑战方式,使用内部挑战认证。
- 5) 若静态口令的验证方式是普通方式,使用静态口令+动态口令认证。

8.2.2.4 设置静态口令

设置动态令牌绑定的静态口令。

- 1) 要求验证原有的静态口令。
- 2) 若静态口令的验证方式是内部挑战方式,使用内部挑战认证。
- 3) 若静态口令的验证方式是静动态混合方式,使用静态口令+动态口令认证。

8.2.2.5 远程解 PIN

认证系统可提供远程解 PIN 的功能(针对具有 PIN 保护的令牌)。根据应用请求,认证系统生成当前的远程解 PIN 密码,将该密码输入到令牌中,解除令牌因 PIN 而锁定的状态。

- 1) 解 PIN 的密码为 0-9 的数字串,长度最少为 6 位。
- 2) 解 PIN 的操作最大尝试次数不可超过 5 次,若超过最大尝试次数,应至少等待 1 h 才可继续尝试。
- 3) 超过最大尝试次数的情况不可超过 5 次,否则令牌应永久锁定,不可再使用。

8.2.2.6 同步与窗口要求

认证系统提供令牌的同步服务。

- 1) 在大窗口、中窗口内验证令牌的连续 2 个动态口令,若成功,调整令牌的系统偏移量。
- 2) 在小窗口内验证令牌的当前动态口令,若成功,调整令牌的系统偏移量。
- 3) 令牌的同步服务不改变令牌状态。

8.2.2.7 废止

令牌损坏或失效后,可使用认证系统的废止服务将其废止。废止的令牌不可再用于用户的身份认证和交易验证。系统仅保留该令牌的使用历史记录。

8.2.2.8 令牌信息查询

认证系统应提供令牌的信息查询服务,包括:令牌的当前状态、上次使用时间、当前累计错误次数等。

信息查询服务不改变令牌状态。

8.3 认证系统管理功能

8.3.1 权限管理

认证系统应对访问人员采取权限控制,不同角色的访问人员赋予不同的操作权限。

8.3.2 参数配置

认证系统应能对认证和管理功能参数进行配置。

8.3.3 日志管理

日志管理包括日志的写入,查询等功能,每条日志至少记录事件的日期和时间、事件类型、主体身份、事件的结果(成功或失效)、日志级别。以下事件应该记录日志:

- 1) 动态口令认证,同步的结果。
- 2) 令牌系统状态的变更。

8.3.4 服务报表

系统应可提供对令牌和系统不同时间段对应的状态和结果的统计报表。

8.3.5 种子导入

认证系统应能导入令牌的种子密钥,并设置令牌的初始状态。

8.3.6 备份恢复

认证系统应能对敏感信息进行备份和恢复。

8.4 安全要求

8.4.1 接入端控制

认证服务器应具有控制应用服务器安全接入的方法和措施。

8.4.2 通讯敏感字段加密

为了防止网络监听的形式对认证数据进行窃听和分析,应在认证服务器和应用服务器之间的通讯数据上做加密处理。

8.4.3 信息存储加密

认证系统中的种子密钥是加密存储的,当认证服务器接收到认证请求时,认证服务器应首先解密种子密钥加密密钥密文,然后通过种子密钥和时间因子等信息生成对应的动态口令,并与接收到的动态口令进行比较,从而完成动态口令身份认证。

8.4.4 令牌安全性控制

8.4.4.1 锁定及解锁

可以按照用户要求提供锁定机制,当一个令牌连续尝试认证失败次数累计达到上限,则对令牌进行锁定,同时提供人工解锁和自动解锁机制。

8.4.4.2 防重复认证

重复认证检测,对于已经通过认证的动态口令,认证服务器将予以作废,只要已经通过认证的动态口令,均不能再次通过认证。

对于特定的使用场景,可允许正确的动态口令在一定次数范围内重复使用,但认证系统必须具备保护此环境下身份认证安全性和有效性的机制。

8.4.4.3 日志安全

日志信息应具有校验码,只要用户对日志信息进行修改,就可通过校验码检查出来。

敏感数据应具有备份恢复机制。

认证系统针对日志访问应具备相应的访问控制策略,对日志的操作应有对应记录,以保证日志的完整性和安全性。

8.4.5 接入端控制

认证服务器应具有时间校准的处理方法和措施。

8.4.6 认证系统安全

认证系统安全必须符合目标应用服务或系统的安全需求,具体可参照目标应用服务或系统的相应规范或标准。

9 密钥管理系统

9.1 概述

密钥管理主要是指对种子密钥的生成、传输和存储的安全管理,种子密钥是否安全直接影响到整个认证系统是否安全。因此,保护种子密钥的安全至关重要,具体可以从以下几个方面来保护:

- 1) 种子密钥生成的安全性
- 2) 种子密钥传输的安全性
- 3) 种子密钥存储的安全性
- 4) 种子密钥使用的安全性

9.2 系统架构

9.2.1 系统组成

密钥管理系统主要由系统登录认证模块、用户管理模块、保护密钥生成模块、种子密钥生成模块、令

牌序号生成模块、时间同步模块(可选)、令牌生产配置模块、硬件密码设备接口模块和动态令牌读写接口模块组成,如图 4 所示:

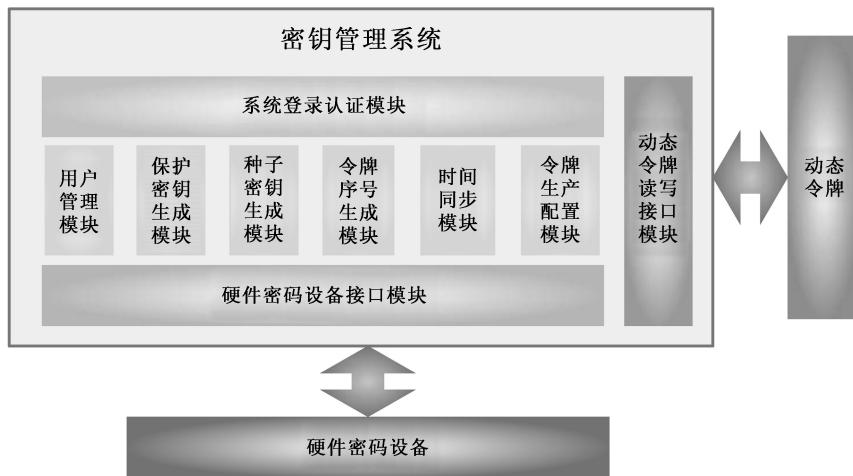


图 4 密钥管理系统组成图

9.2.1.1 系统登录认证模块

该模块负责密钥管理系统的用户登录认证。

9.2.1.2 用户管理模块

该模块负责密钥管理系统操作员的管理(新建、删除和修改)和权限分配。

9.2.1.3 保护密钥生成模块

该模块主要用于生成密钥管理系统的根密钥及传输密钥。

9.2.1.4 种子密钥生成模块

该模块的功能是调用硬件密码设备接口生成动态令牌的种子密钥,同时将生成的种子密钥加密后写入导出数据文件。

9.2.1.5 令牌序号生成模块

该模块的功能是根据序号生成规则生成动态令牌的唯一序列号。

9.2.1.6 时间同步模块(可选)

该模块的功能是同步密钥管理系统的系统时间,以保证种子密钥生成系统与动态口令认证系统的时间一致。

9.2.1.7 令牌生产配置模块

该模块的功能是将令牌序列号和种子密钥通过动态令牌读写接口模块写入动态令牌。

9.2.1.8 硬件密码设备接口模块

该模块的功能是完成软件与硬件密码设备之间的通讯。

9.2.1.9 动态令牌读写接口模块

该模块的功能是完成软件和令牌之间的通讯。

9.2.2 系统安全

系统安全的主要目标是保障网络、主机系统、应用系统及数据库运行的安全。应采取防火墙、病毒防治、漏洞扫描、入侵监测、数据备份、灾难恢复等安全防护措施。

9.3 功能要求

9.3.1 系统功能

密钥管理系统功能应包括：系统登录认证、用户管理、保护密钥管理、令牌序列号管理、种子密钥管理、令牌生产配置和时间同步。

9.3.1.1 系统登录认证

密钥管理系统的使用应在系统合法用户登录以后才能使用，且系统应对用户分权限管理，不同的用户授权使用不同的功能。用户登录系统一段时间未操作后，系统应能够自动锁定。

9.3.1.2 用户管理

用户管理的功能应包括但不限于：用户账号和角色的分配、修改和删除。系统角色应包括但不限于：系统管理员、密钥管理操作员、种子生成操作员和令牌生成操作员。密码管理系统的用户应使用包括但不限于 USB-KEY 的安全措施作为登录系统操作的凭证。

9.3.1.3 保护密钥管理

密钥管理系统应具备主密钥、传输密钥的生成、备份和更新的功能。密钥的备用载体应使用 IC 卡或者 USB-KEY。

9.3.1.4 令牌序列号管理

令牌序列号管理应对令牌序列号模板的定义、修改和删除，令牌序列号的生成进行管理。

9.3.1.5 种子密钥生成

种子密钥管理功能应能实现种子密钥的生成和加密导出。

9.3.1.6 令牌生产配置

密钥管理系统应具备将种子密钥、序列号和时间(可选)写入动态令牌的功能。

9.3.1.7 时间同步

密钥管理系统应具备在生产时间同步令牌时，预先校对生产电脑时间的功能。

9.3.1.8 系统日志

密钥管理系统应具备记录密钥管理的所有操作及系统运行日志的功能。

9.3.2 系统兼容性

- 1) 系统算法兼容性：密钥管理系统应支持多种分组密码算法来对密钥的管理。

2) 硬件密码设备兼容性:密钥管理系统应兼容多种密码硬件设备,如密码机、密码卡、IC 卡等。

9.4 系统安全性设计

9.4.1 密钥管理的目的

密钥安全的主要目的是保障系统中所使用的密钥,在其生成、存储、使用、生命周期中的安全性。

9.4.2 密钥管理的原则

本系统中密钥安全设计遵循以下原则:

- 1) 采用国家密码管理局批准的硬件密码设备。
- 2) 主密钥的生成由专用硬件生成。
- 3) 主密钥的存储由专用硬件存储,并且无法导出。
- 4) 所有密钥的运算都在专用硬件中完成。
- 5) 种子加密密钥由主密钥对令牌序号分散得到。
- 6) 严格的主密钥管理机制和灾难恢复机制。

9.4.3 密钥管理的机制

系统中密钥简介

主密钥 K_m :在令牌应用服务商(如银行、电信公司、企业等)的硬件密码设备中生成与存储,使用时不离开该硬件密码设备。

种子密钥加密密钥 K_s :由主密钥 K_m 对序列号分散获得,存储在令牌应用服务商的硬件密码设备中。

厂商生产主密钥 K_p :由主密钥 K_m 对厂商代码分散获得,存储在令牌应用服务商的硬件密码设备中,可通过安全加密方式提供给厂商使用。

厂商种子密钥加密密钥 K_{ps} :由厂商生产主密钥 K_p 对序列号分散获得,存储在令牌应用服务商和令牌厂商的硬件密码设备中。

传输密钥 K_t :由令牌应用服务商的硬件密码设备随机生成,可通过分段导出的方式,交付给令牌厂商使用。

种子密钥:由硬件密码设备生成,生成时如采用了密码算法,则应采用国家密码管理局批准的密码算法。

具体密钥管理流程与机制,如图 5 所示:

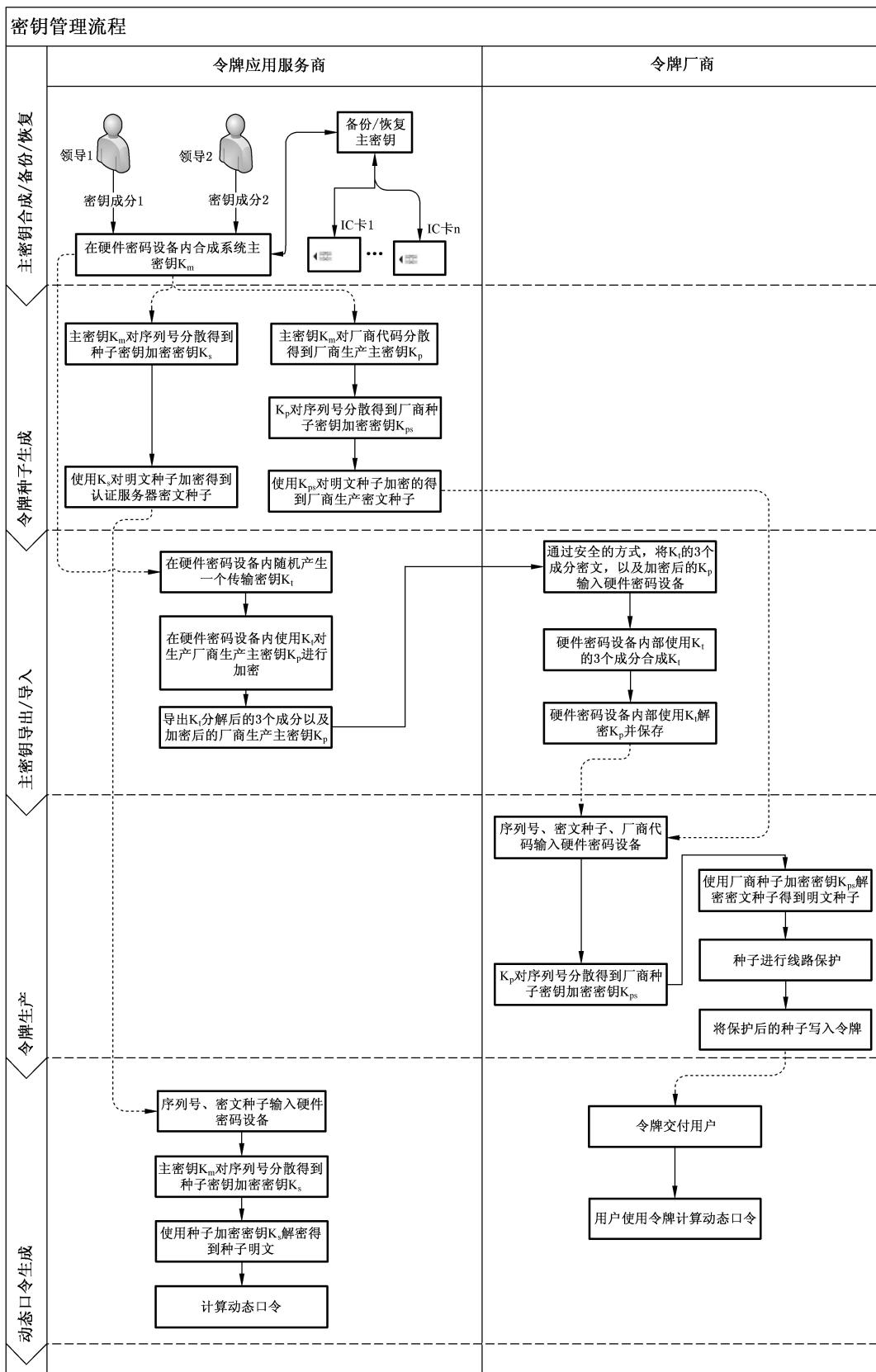


图 5 密钥管理机制

9.4.3.1 主密钥

主密钥 K_m 的生成是由多人分别输入各自的密钥成分,在硬件密码设备内部按照国家密码管理局批准的方法实现。

主密钥 K_m 的备份和恢复可有两种方式:

- 1) 每人分别将各自的密钥成分封存,以此作为恢复的依据
- 2) 由硬件密码设备管理员将打散后的主密钥因子分别备份在多张智能卡中,分开储存并建立严格的领用登记制度。上述智能卡作为恢复的依据。

9.4.3.2 厂商生产主密钥

厂商生产主密钥 K_p 由主密钥 K_m 分散厂商代码后得到,通过 K_t 加密提供给厂商使用,存储在令牌应用服务商的硬件密码设备中。 K_p 的导入、储存、备份和销毁基于令牌应用服务商的硬件密码设备完成,多人管理共同完成。

9.4.3.3 传输密钥 K_t

用于加密保护厂商生产主密钥 K_p 的交换,保障生产所用的硬件密码设备和认证所用的硬件密码设备之间厂商生产主密钥 K_p 交换的安全。

导出厂商生产主密钥 K_p 时,传输密钥 K_t 临时随机产生,输出其多个成份及传输密钥加密的厂商生产主密钥 K_p 密文;

导入厂商生产主密钥 K_p 时,输入传输密钥的多个成份及厂商生产主密钥 K_p 密文,内部合成传输密钥并解密出厂商生产主密钥 K_p 保存。

9.4.3.4 种子密钥加密密钥

系统中有两种类型的种子密钥加密密钥分别是认证硬件密码设备中的种子密钥加密密钥 K_s 和生产所用的硬件密码设备中的 K_{ps} 。 K_s 在硬件密码设备内部由主密钥 K_m 对令牌序号分散后生成。 K_{ps} 在硬件密码设备内部由厂商生产主密钥 K_p 对令牌序号分散后生成。硬件密码设备内部使用 SM4 算法负责密钥的分散过程,但分散出的种子密钥加密密钥不向硬件密码设备以外导出。种子密钥加密密钥实际上为临时密钥,不在硬件密码设备内部保存。

上述密钥的相关管理操作受一定的权限控制,未经授权无法操作硬件密码设备。

9.4.4 种子密钥的安全

9.4.4.1 种子密钥的生成安全

种子密钥由硬件密码设备生成,生成时如采用了密码算法,则应采用 SM4 算法。

9.4.4.2 种子密钥的传输安全

种子密钥的传输应采用加密方式,并采用安全方式传输。

9.4.4.3 种子密钥写入动态令牌过程的安全

种子密钥写入动态令牌时,该过程必须在安全的生产环境中进行,具有安全的管理机制。

种子密钥在写入令牌时应保证其写入线路的安全性。

安全的生产环境,是指用于安装密钥管理系统的电脑必须位于封闭、无网络连接的环境中。

安全的管理机制,是指在生产过程中的安全管理措施包括但不限于:

- 1) 生产环境中需安装监控设备,以监视进入生产环境的工作人员。
- 2) 种子密钥生成系统需同时两人输入用户名和密码才能启动,制作过程中一人操作,一人审核。
- 3) 限制 USB 存储设备的使用,只有获得允许才能使用。

9.4.4.4 种子密钥导入认证系统过程的安全

通过硬件传输方式(如光盘)导入到相关认证系统中。

数据文件命名规则:otp_[厂商代码]_yyyymmdd_[数量].data

数据文件内容格式如图 6 所示:

[令牌序列号], [密文种子], [密钥索引], [校验值], [创建时间]
.....
[令牌序列号], [密文种子], [密钥索引], [校验值], [创建时间]

图 6 数据文件的内部格式

9.4.4.5 种子密钥的存储安全

种子密钥在认证服务器中以 SM4 算法加密后的密文方式存储。种子密钥具体的加密流程如下:

- 1) 硬件密码设备使用主密钥 K_m 对令牌的序列号填充数据到 16 字节(填充数据为 0x00),然后使用 SM4 算法进行分散,得到种子密钥加密密钥 K_s 。
- 2) 对刚刚生成种子密钥进行填充到 16 字节的整数倍,填充数据的规则遵循 PKCS#5 中定义的规范(即:对输入数据 MSG,在 MSG 的右端添加 16-(||MSG|| % 16)个取值为 16-(||MSG|| % 10)的 PAD 字符;也就是说,对 MSG,最右端数据块,缺 n 个字节则补充 n 个数值 n,最少补 1 个 0x01,最多补 16 个 0x10(当 MSG 长度为 16 字节的整数倍时))。
- 3) 使用种子密钥加密密钥 K_s 对填充后的种子密钥数据进行 SM4 加密。
- 4) 加密完成后,将种子密钥加密密钥 K_s 和明文种子密钥数据销毁。

种子密钥存储时,应至少包含以下几个属性:

表 2 种子密钥存储格式

令牌序列号	密文种子	密钥索引	校验值	创建时间

9.4.4.6 种子密钥的使用安全

种子密钥的使用安全是指使用种子密钥计算动态口令过程中的安全。种子密钥的使用过程全部在硬件密码设备内完成,杜绝了种子密钥在使用过程中泄密的可能。具体使用流程如下:

- 1) 将令牌序号和种子密钥加密数据发送到硬件密码设备中,硬件密码设备对加密的种子密钥根据令牌序号进行“种子密钥存储”相同运算,即使用令牌序号由主密钥 K 分散得到种子密钥解密密钥 K_s 。
- 2) 对种子密钥密文进行 SM4 解密获得种子密钥的原文。
- 3) 使用明文种子密钥和其它相应的参数运算得到动态口令,对客户端发送的认证请求进行甄别。
- 4) 解密完成后,将种子密钥解密密钥 K_s 和明文种子密钥数据销毁。

9.4.4.7 其它

特定环境中,令牌厂商可与令牌应用服务商进行协商,在上述种子密钥的安全基础上,由令牌应用服务商将预先生成好的一批种子密钥,使用特定的一个种子密钥加密密钥进行加密,将种子密钥密文交

付令牌厂商用于生产。令牌厂商与令牌应用服务商应保证该过程的安全性。

特定环境中,令牌厂商可与令牌应用服务商进行协商,在上述种子密钥的安全基础上,种子密钥可由国家密码管理局批准的硬件随机数发生器生成。

9.5 硬件密码设备接口说明

9.5.1 硬件密码设备应用接口

动态口令计算

功能:使用种子密钥密文,计算动态口令。

9.5.2 认证系统密钥管理接口

1) 种子密钥的生成接口

功能:随机产生或运算产生指定长度的种子密钥,根据令牌序号产生种子密钥加密密钥,返回加密后的种子密钥密文。

2) 向令牌导入种子密钥

功能:将种子明文写入令牌中;

3) 合成主密钥接口

功能:根据密钥成分合成主密钥

4) 主密钥的备份接口

功能:将主密钥备份成多个密文密钥成分。

5) 主密钥的恢复接口

功能:将备份的密钥成分,在硬件设备内恢复成主密钥。

6) 传输密钥保护导出主密钥接口

功能:将主密钥从硬件密码设备中密文导出。

7) 传输密钥保护导入主密钥接口

功能:将密文主密钥导入硬件密码设备。

9.5.3 接口使用权限控制机制

建立接口使用权限机制,以达到控制管理接口使用权限的目的。

附录 A
(资料性附录)
动态口令生成算法 C 语言实现用例

A.1 采用 SM3 的动态口令生成算法用例

```
#include <stdio.h>
#include <string.h>
#include <math.h>
#include "sm3.h"
#include "sm_dpwd.h"

#ifndef __cplusplus
#define define INLINE inline
#else
#define define INLINE
#endif

INLINE bool IsBigEndian()
{
    union T
    {
        char c[2];
        short s;
    };
    T t;
    t.s=0x0031;
    if (t.c[1]==0x31)
    {
        return true;
    }
    return false;
}

INLINE bool IsLittleEndian()
{
    return ! IsBigEndian();
}

INLINE uint32 Reverse32(uint32 x)
```

```
{
    return ((x & 0x000000ff)<<24)
        | ((x & 0x0000ff00)<<8)
        | ((x & 0x00ff0000)>>8)
        | ((x & 0xff000000)>>24);
}
```

```
INLINE uint64 Reverse64(uint64 x)
{
    uint32 nTemp[3]={0};
    memcpy(nTemp+1,&x,sizeof(uint64));
    nTemp[0]=Reverse32(nTemp[2]);
    nTemp[1]=Reverse32(nTemp[1]);
    return *(uint64 *)nTemp;
}
```

```
INLINE sm_word ML(byte X,uint8 j)
{
    if (IsBigEndian())
    {
        return (sm_word)(X<<(j%32));
    }
    else
    {
        return Reverse32((sm_word)(X<<(j%32)));
    }
}
```

```
INLINE sm_word SUM(sm_word X,sm_word Y)
{
    if (IsBigEndian())
    {
        return (X+Y);
    }
    else
    {
        return Reverse32(Reverse32(X)+Reverse32(Y));
    }
}
```

```
int TruncateSM3(IN byte pSrc[32],IN int nSrcLen,OUT byte pDst[4],IN int nDstSize)
{
    if (nSrcLen != 32 || nDstSize<4)
```

```

{
    return -1;
}

memset(pDst,0,nDstSize);

byte * S=(byte *)pSrc;
sm_word S1=ML(S[ 0],24) | ML(S[ 1],16) | ML(S[ 2],8) | ML(S[ 3],0);
sm_word S2=ML(S[ 4],24) | ML(S[ 5],16) | ML(S[ 6],8) | ML(S[ 7],0);
sm_word S3=ML(S[ 8],24) | ML(S[ 9],16) | ML(S[10],8) | ML(S[11],0);
sm_word S4=ML(S[12],24) | ML(S[13],16) | ML(S[14],8) | ML(S[15],0);
sm_word S5=ML(S[16],24) | ML(S[17],16) | ML(S[18],8) | ML(S[19],0);
sm_word S6=ML(S[20],24) | ML(S[21],16) | ML(S[22],8) | ML(S[23],0);
sm_word S7=ML(S[24],24) | ML(S[25],16) | ML(S[26],8) | ML(S[27],0);
sm_word S8=ML(S[28],24) | ML(S[29],16) | ML(S[30],8) | ML(S[31],0);

sm_word OD=SUM(SUM(SUM(SUM(SUM(SUM(S1,S2),S3),S4),S5),S6),S7),
S8);
memcpy(pDst,&OD,sizeof(sm_word));

return 0;
}

#define SM_DPWD_KEY_LEN_MIN          (128/8)
#define SM_DPWD_CHALLENGE_LEN_MIN   (4)
#define SM_DPWD_LEN_MAX             (10)
#define SM_HASH_OUT_LEN              (32)

int SM3_DPasswd(IN byte * pKey,IN int nKeyLen,IN uint64 * pTime,IN uint64 * pInterval,IN
uint32 * pCounter,
                 IN char * pChallenge,IN int nGenLen,OUT char * pDynPwd,IN int nDynPwd-
Size)
{
    if (pKey == NULL || (pTime == NULL && pCounter == NULL && pChallenge ==
NULL)
        || pDynPwd == NULL || nKeyLen<SM_DPWD_KEY_LEN_MIN || nGenLen>SM_
DPWD_LEN_MAX
        || (pChallenge != NULL && strlen(pChallenge)<SM_DPWD_CHALLENGE_LEN_-
MIN)
        || nDynPwdSize<nGenLen+1)
    {
        return SM_DPWD_PARAM_ERROR;
    }
    memset(pDynPwd,0,nDynPwdSize);
}

```

```

// T=To/Tc
if (pTime !=NULL && pInterval !=NULL && * pInterval !=0)
{
    * pTime=( * pTime)/( * pInterval);
}

// Convert to big-endian.
if (! IsBigEndian())
{
    if (pTime !=NULL)
    {
        * pTime=Reverse64( * pTime);
    }
    if (pCounter !=NULL)
    {
        * pCounter=Reverse32( * pCounter);
    }
}

int      offset          =0;
byte *   sm_i            =NULL;
byte    sm_o[SM_HASH_OUT_LEN]={0};
int     sm_i_len         =0;
int     sm_o_len         =sizeof(sm_o);
uint32  pwd              ={0};

// ID(T|C|Q) Length at least 128 bits
sm_i_len=(pTime ? sizeof(uint64) : 0)+(pCounter ? sizeof(uint32) : 0)+(pChallenge ? strlen
(pChallenge) : 0);
if (sm_i_len<16)
{
    // Fill ID to 128 bits with 0 at the end.
    sm_i_len=16;
}
sm_i_len +=nKeyLen;

// Allocate IN-Data memory.
sm_i=new byte[sm_i_len];
if (sm_i ==NULL)
{
    return SM_DPWD_NO_MEMORY;
}

```

```

memset(sm_i,0,sm_i_len);

// 1. KEY|ID(T|C|Q)
memcpy(sm_i,pKey,nKeyLen);
offset=nKeyLen;
if (pTime !=NULL)
{
    memcpy(sm_i+offset,pTime,sizeof(uint64));
    offset +=sizeof(uint64);
}
if (pCounter !=NULL)
{
    memcpy(sm_i+offset,pCounter,sizeof(uint32));
    offset +=sizeof(uint32);
}
if (pChallenge !=NULL)
{
    memcpy(sm_i+offset,pChallenge,strlen(pChallenge));
}

// 2. SM3
SM3(sm_i,sm_i_len,sm_o,sm_o_len);

// 3. Truncate
TruncateSM3(sm_o,sm_o_len,(byte *)&pwd,sizeof(pwd));

#ifndef __SM_DBG_OUT
__DInit();
__DAdd("      K :[%s]\r\n",     __S2M(pKey,nKeyLen));
__DAdd("      T :[%016s]\r\n", __S2M(pTime,8));
__DAdd("      C :[%08s]\r\n",   __S2M(pCounter,4));
__DAdd("      Q :[%s]\r\n",     __S2M(pChallenge,pChallenge == NULL ? 0 : strlen(pChallenge)));
__DAdd("SM3-IN :[%s]\r\n",     __S2M(sm_i,sm_i_len));
__DAdd("SM3-OUT:[%s]\r\n",    __S2M(sm_o,sm_o_len));
__DAdd("      Cut :[%s]\r\n",   __S2M(&pwd,sizeof(pwd)));
#endif // __SM_DBG_OUT

// 4. MOD
if (! IsBigEndian())
{
    pwd=Reverse32(pwd);
}

```

```

pwd=pwd % (int)pow(10,nGenLen);

// Output
char szFmt[32]={0};
sprintf(szFmt,"%%0%dd",nGenLen);
sprintf(pDynPwd,szFmt,pwd);

delete [] sm_i;
return 0;
}

```

A.2 采用 SM4 的动态口令生成算法用例

```

#include <stdio.h>
#include <string.h>
#include <math.h>
#include "sm4.h"
#include "sm_dpwd.h"

#ifndef __cplusplus
#define INLINE inline
#else
#define INLINE
#endif

#ifndef max
#define max(a,b) (((a)>(b)) ? (a) : (b))
#endif
#ifndef min
#define min(a,b) (((a)<(b)) ? (a) : (b))
#endif

INLINE bool IsBigEndian()
{
    union T
    {
        char c[2];
        short s;
    };
    T t;
    t.s=0x0031;
    if (t.c[1]==0x31)

```

```
{  
    return true;  
}  
return false;  
}  
  
INLINE bool IsLittleEndian()  
{  
    return ! IsBigEndian();  
}  
  
INLINE uint32 Reverse32(uint32 x)  
{  
    return ((x & 0x000000ff)<<24)  
        | ((x & 0x0000ff00)<<8)  
        | ((x & 0x00ff0000)>>8)  
        | ((x & 0xff000000)>>24);  
}  
  
INLINE uint64 Reverse64(uint64 x)  
{  
    uint32 nTemp[3]={0};  
    memcpy(nTemp+1,&x,sizeof(uint64));  
    nTemp[0]=Reverse32(nTemp[2]);  
    nTemp[1]=Reverse32(nTemp[1]);  
    return *(uint64 *)nTemp;  
}  
  
INLINE sm_word ML(byte X,uint8 j)  
{  
    if (IsBigEndian())  
    {  
        return (sm_word)(X << (j%32));  
    }  
    else  
    {  
        return Reverse32((sm_word)(X << (j%32)));  
    }  
}  
  
INLINE sm_word SUM(sm_word X,sm_word Y)  
{  
    if (IsBigEndian())  
    {  
        return X + Y;  
    }  
    else  
    {  
        return (sm_word)((X << 24) | (Y >> 24));  
    }  
}
```

```

    {
        return (X+Y);
    }
    else
    {
        return Reverse32(Reverse32(X)+Reverse32(Y));
    }
}

int TruncateSM4(IN byte pSrc[16],IN int nSrcLen,OUT byte pDst[4],IN int nDstSize)
{
    if (nSrcLen != 16 || nDstSize<4)
    {
        return -1;
    }
    memset(pDst,0,nDstSize);

    byte * S=(byte *)pSrc;
    sm_word S1=ML(S[ 0],24) | ML(S[ 1],16) | ML(S[ 2],8) | ML(S[ 3],0);
    sm_word S2=ML(S[ 4],24) | ML(S[ 5],16) | ML(S[ 6],8) | ML(S[ 7],0);
    sm_word S3=ML(S[ 8],24) | ML(S[ 9],16) | ML(S[10],8) | ML(S[11],0);
    sm_word S4=ML(S[12],24) | ML(S[13],16) | ML(S[14],8) | ML(S[15],0);

    sm_word OD=SUM(SUM(SUM(S1,S2),S3),S4);
    memcpy(pDst,&OD,sizeof(sm_word));

    return 0;
}

#define SM_DPWD_KEY_LEN_MIN          (128/8)
#define SM_DPWD_CHALLENGE_LEN_MIN   (4)
#define SM_DPWD_LEN_MAX             (10)
#define SM_HASH_OUT_LEN              (32)

int SM4_DPasswd(IN byte * pKey,IN int nKeyLen,IN uint64 * pTime,IN uint64 * pInterval,IN
uint32 * pCounter,
                 IN char * pChallenge,IN int nGenLen,OUT char * pDynPwd,IN int nDynPwd-
Size)
{
    if (pKey == NULL || (pTime == NULL && pCounter == NULL && pChallenge ==
NULL)
        || pDynPwd == NULL || nKeyLen<SM_DPWD_KEY_LEN_MIN || nGenLen>SM_
DPWD_LEN_MAX

```

```

|| (pChallenge != NULL && strlen(pChallenge) < SM_DPWD_CHALLENGE_LEN_MIN)
|| nDynPwdSize < nGenLen + 1)
{
    return SM_DPWD_PARAM_ERROR;
}
memset(pDynPwd, 0, nDynPwdSize);

// T=To/Tc
if (pTime != NULL && pInterval != NULL && *pInterval != 0)
{
    *pTime = (*pTime) / (*pInterval);
}

// Convert to big-endian.
if (!IsBigEndian())
{
    if (pTime != NULL)
    {
        *pTime = Reverse64(*pTime);
    }
    if (pCounter != NULL)
    {
        *pCounter = Reverse32(*pCounter);
    }
}

int offset = 0;
byte * sm_buf = NULL;
byte * sm_k = NULL;
byte * sm_i = NULL;
byte sm_o[16] = {0};
int sm_k_len = 0;
int sm_i_len = 0;
int sm_o_len = sizeof(sm_o);
uint32 pwd = {0};

// If length of Key is not multiple of 128 bits, extend it to multiple of 128 with 0.
sm_k_len = nKeyLen;
if (sm_k_len % 16 != 0)
{
    sm_k_len += 16 - sm_k_len % 16;
}

```

```

// If length of ID(T|C|Q) is not multiple of 128 bits, extend it to multiple of 128 with 0.
sm_i_len=(pTime ? sizeof(uint64) : 0)+(pCounter ? sizeof(uint32) : 0)+(pChallenge ? strlen(pChallenge) : 0);
if (sm_i_len % 16 != 0)
{
    sm_i_len += 16 - sm_i_len % 16;
}

// Allocate SM4 buffer(KEY and ID) memory.
sm_buf=new byte[sm_k_len+sm_i_len];
if (sm_buf ==NULL)
{
    return SM_DPWD_NO_MEMORY;
}
memset(sm_buf,0,sm_k_len+sm_i_len);
sm_k=sm_buf;
sm_i=sm_buf+sm_k_len;

// KEY
memcpy(sm_k,pKey,nKeyLen);

// ID=T|C|Q
if (pTime !=NULL)
{
    memcpy(sm_i,pTime,sizeof(uint64));
    offset +=sizeof(uint64);
}
if (pCounter !=NULL)
{
    memcpy(sm_i+offset,pCounter,sizeof(uint32));
    offset +=sizeof(uint32);
}
if (pChallenge !=NULL)
{
    memcpy(sm_i+offset,pChallenge,strlen(pChallenge));
}

int k_cnt=sm_k_len/16;
int i_cnt=sm_i_len/16;
int _cnt=max(k_cnt,i_cnt);

#endif __SM_DBG_OUT

```

```

__Dump("      K :[%s]\r\n",     __S2M(pKey,nKeyLen));
__Dump("      T :[%016s]\r\n", __S2M(pTime,8));
__Dump("      C :[%08s]\r\n",   __S2M(pCounter,4));
__Dump("      Q :[%s]\r\n",     __S2M(pChallenge,pChallenge == NULL ? 0 : strlen(pChallenge)));
#endif //__SM_DBG_OUT

for (int i=0; i<_cnt; ++i)
{
    int rc=SM4_Encrypt(sm_k,16,sm_i,16,sm_o,sm_o_len);
    if (rc<0)
    {
        return rc;
    }

#endif //__SM_DBG_OUT

int j,k;
uint8 overflow;

// 'out'+next 16 bytes 'key'.
overflow=0;
k=min(i+1,k_cnt-1);
for (j=15; j >=0; -j)
{
    uint16 sum=sm_o[j]+sm_k[16 * k+j]+overflow;
    sm_k[j]=(uint8)sum;
    overflow=(uint8)(sum>>8);
}

// 'out'+next 16 bytes 'in'.
overflow=0;
k=min(i+1,i_cnt-1);
for (j=15; j >=0; -j)
{
    uint16 sum=sm_o[j]+sm_i[16 * k+j]+overflow;
    sm_i[j]=(uint8)sum;
    overflow=(uint8)(sum>>8);
}
}

```

```
TruncateSM4(sm_o,sm_o_len,(byte *)pwd,sizeof(pwd));  
  
#ifdef __SM_DBG_OUT  
__Dump("Cut :[%s]\r\n", __S2M(&pwd,sizeof(pwd)));  
#endif // __SM_DBG_OUT  
  
if (!IsBigEndian())  
{  
    pwd=Reverse32(pwd);  
}  
pwd=pwd % (int)pow(10,nGenLen);  
  
char szFmt[32]={0};  
sprintf(szFmt,"%0%dd",nGenLen);  
sprintf(pDynPwd,szFmt,pwd);  
  
delete [] sm_buf;  
return 0;  
}
```

附录 B
(资料性附录)
动态口令生成算法计算输入输出用例

B. 1 采用 SM3 的动态口令生成算法输入输出用例

K 为种子密钥, T_c 为 1 秒, $T = T_c$ 为 UTC 时间表示的时间因子, C 为事件因子, Q 为挑战数据, P 为最终显示的 6 位长度动态口令。具体如表 B. 1:

表 B. 1 采用 SM3 的动态口令生成算法输入输出用例表

K	T	C	Q	P(SM3)
1234567890abcdef1234567890abcdef	1313998979	1234	5678	814095
1234567890abcdefabcdef1234567890	1313998995	5621	3698	959691
1234567890abcdef0987654321abcdef	1313999014	5621	3698	063014
1234567890abcdefabcdef0987654321	1313999047	2053	6984	302593
87524138025adcf2584376195abfedc	1313999067	2058	3024	657337
87524138025adcfabfedc2584376195	1313999098	2056	2018	345821
adcf287524138025abfedc2584376195	1313999131	2358	1036	629660
58ade3698fe280cb6925010dd236caef	1313999155	2547	2058	479821
58ade365201d80cbdd236caef6925010	1313999174	6031	2058	893826
65201d80cb58ade3dd236caef6925010	1313999189	6580	1047	607614

B. 2 采用 SM4 的动态口令生成算法输入输出用例

K 为种子密钥, T_c 为 1 秒, $T = T_c$ 为 UTC 时间表示的时间因子, C 为事件因子, Q 为挑战数据, P 为最终显示的 6 位长度动态口令。具体如表 B. 2:

表 B. 2 采用 SM4 的动态口令生成算法输入输出用例表

K	T	C	Q	P(SM4)
1234567890abcdef1234567890abcdef	1340783053	1234	5678	446720
1234567890abcdefabcdef1234567890	1340783416	5621	3698	049845
1234567890abcdef0987654321abcdef	1340783476	2584	2105	717777
87524138025adcfabfedc2584376195	1340783509	2053	6984	037000
87524138025adcf2584376195abfedc	1340783588	2058	3024	502206
1234567890abcdefabcdef0987654321	1340783624	2056	2018	692843
adcf287524138025abfedc2584376195	1340783652	2358	1036	902690

表 B.2 (续)

K	T	C	Q	P(SM4)
58ade3698fe280cb6925010dd236caef	1340783729	2547	2058	499811
58ade365201d80cbdd236caef6925010	1340783771	6031	2058	565180
65201d80cb58ade3dd236caef6925010	1340783815	6580	1047	724654

附录 C
(资料性附录)
运算参数与数据说明用例

采用 SM3 和 SM4 的运算参数与数据说明用例, K 为种子密钥, T_c 为 1 秒, $T = T_0$ 为 UTC 时间表示的时间因子, C 为事件因子, Q 为挑战数据, SM4 和 SM3 输出结果分别是 128 比特和 256 比特, 截位结果是截位运算后的输出数据。具体如表 C.1 和 C.2:

表 C.1 采用 SM3 的运算参数与数据说明用例表

SM3	输入	参与运算的值(HEX)
K	1234567890abcdef1234567890abcdef	12 34 56 78 90 ab cd ef 12 34 56 78 90 ab cd ef
T	1313655030	00 00 00 00 4E 4C C8 F6
C	1234	00 00 04 d2
Q	5678	35 36 37 38
SM3 输出结果		25 e0 b0 0d 75 0e b0 12 58 ef 7d b5 37 56 26 41 4a cf e7 fd 82 6a c0 7e 3e 5e b3 e8 d8 eb ba 4b
截位结果		0f ba 1a c3

表 C.2 采用 SM4 的运算参数与数据说明用例表

SM4	输入	参与运算的值(HEX)
K	1234567890abcdef1234567890abcdef	12 34 56 78 90 ab cd ef 12 34 56 78 90 ab cd ef
T	1340783053	00 00 00 00 4f ea b9 cd
C	1234	00 00 04 d2
Q	5678	35 36 37 38
SM4 输出结果		88 0d 6a e7 7e cf 8e e5 23 5c 71 98 e1 3f 15 9c
截位结果		0b 78 81 00

附录 D
(资料性附录)
认证系统接口

D.1 服务报文格式

D.1.1 报文格式

认证系统的服务报文由〈报头〉+〈报体〉构成。其中报头定义报文类型和选项，报体为具体的服务请求或服务响应数据。

报头的格式如表 D.1：

表 D.1 报头格式表

偏移量	名称	长度 (字节)	描述	说明
0	报头长度	1	标识报头长度	
1	报文类型	1	标识报文类型	x0：请求报文 x1：响应报文 0x：不需要校验 MAC 8x：需要校验 MAC
2	版本	1	标识报文版本	01：版本 1
3	调用者	8	标识调用者	为调用者分配的标识，缺省可为 8 个 00
11	调用号	8	标识本次调用/响应	由调用者产生的调用号，调用号应每次不同
19	报体长度	2	标识报体长度	不包含报头的报文长度
21	MAC	4	MAC 校验值	如果报文类型 bit7 设置为 1，即需要 MAC 校验，有本数据项。 将报头(不含本项)与报文，使用 SM3 算法运算，取低位的 4 个字节，作为 MAC 校验值。

D.1.2 服务请求报文

当报头的报文类型标识的 bit0 是 0 时，为服务请求报文。服务请求报文的格式为：〈请求头〉+〈数据体〉。

请求头的格式如表 D.2：

表 D.2 请求头格式表

偏移量	名称	长度 (字节)	描述	说明
0	服务标识	2	标识请求的认证系统服务	认证系统提供的服务。其中 1 000 以上为自定义服务
2	服务选项	2	标识服务请求的参数	根据具体服务的不同设置参数
4	数据项个数	1	标识服务请求的数据项个数	数据体包含的数据项个数

D.1.3 服务响应报文

当报头的报文类型标识的 bit0 是 1 时,为服务响应报文。服务响应报文的格式为:〈响应头〉+〈数据体〉。

响应头的格式如表 D.3:

表 D.3 响应头格式表

偏移量	名称	长度 (字节)	描述	说明
0	服务标识	2	标识响应的服务请求	认证系统提供的服务。其中 8 000 以上为自定义服务
2	结果标识	2	标识服务响应的结果	其中 8 000 以上标识服务请求错误。第 1 个字节标识具体的结果代码
4	数据项个数	1	标识服务响应的数据项个数	数据体包含的数据项个数

D.1.4 数据体及数据项格式

数据体由请求头或响应头设定的数据项个数的数据单元组成,其格式为:

〈数据项 1〉+〈数据项 2〉+…+〈数据项 n〉

每个数据项的格式如表 D.4:

表 D.4 数据项格式表

偏移量	名称	长度 (字节)	描述	说明
0	数据属性	1	标识数据属性	bit7 为 1,加密数据 bit7 为 0,明文数据
1	数据标识	2	标识数据的含义	数据项的含义,其中 8 000 以上为自定义数据项
4	数据长度	1	标识数据项的长度	
	数据内容	数据长度	由数据长度规定的数据内容	

D. 2 服务标识

认证系统的服务标识,如表 D. 5:

表 D. 5 服务标识表

名称	值	描述	说明
动态口令认证	0001	动态口令认证请求	
挑战应答认证	0002	挑战应答认证请求	
产生挑战码	0003	请求一个挑战码	
激活	0101	激活令牌	
锁定	0102	锁定令牌	
解锁	0103	解锁令牌	
挂起	0104	挂起令牌	
解挂	0105	解除令牌挂起	
设置静态口令	0106	设置令牌绑定的静态口令	
远程解 PIN	0107	请求远程解 PIN 密码	
同步	0108	强制令牌同步	
更新	0109	更新令牌密钥	
废止	010a	将令牌废止	
令牌信息查询	010b	查询令牌信息	

D. 3 数据标识

认证系统的数据标识,如表 D. 6:

表 D. 6 数据标识表

名称	标识	描述	说明
厂商标识	0001	令牌的厂商标识	
序列号	0002	令牌序列号	
动态口令	0003	动态口令	
下一口令	0004	下一个动态口令	
交易内容	0005	交易内容	用于计算挑战码或校验挑战应答口令
挑战码	0006	根据交易内容产生的挑战码	
应答码	0007	应答码	用于验证的应答码
PIN 码	0008	与令牌绑定的 PIN 码	
新 PIN 码	0009	设置的新 PIN 码	

表 D.6 (续)

名称	标识	描述	说明
更新码	000a	更新令牌的更新码	
初次激活时间	0101	初次激活时间	
最近使用时间	0102	最近使用时间	
过期时间	0103	过期时间	
错误次数	0104	错误次数	
令牌状态	0105	令牌状态	未激活、就绪、锁定、挂起、作废
令牌型号	0106	令牌型号	

D.4 返回码

认证系统的返回码,如表 D.7:

表 D.7 返回码表

名称	Byte1	Byte0	描述	说明
安全服务成功	00	xx	安全服务成功	
	00	01	动态口令认证成功	
	00	02	挑战应答认证成功	
	00	03	挑战码成功	
安全服务失败	80	xx	安全服务失败	
	80	01	要求下一个口令	
	80	02	动态口令错	
	80	03	PIN 码失败	
	80	04	口令已被验证过	
管理服务成功	01	xx	管理服务成功	xx 标识对应的管理服务
管理服务失败	81	xx	管理服务失败	
	81	01	要求下一个口令	
	81	02	动态口令错	
	81	03	PIN 码失败	
	81	04	口令已被验证过	
	81	05	同步失败	
令牌错误	84	xx	令牌错误	
	84	01	没有厂商号	
	84	02	没有这个令牌	
	84	03	令牌记录错误	

表 D.7 (续)

名称	Byte1	Byte0	描述	说明
	84	04	令牌被锁定	
	84	05	令牌被挂起	
	84	06	令牌未激活	
	84	07	令牌已被废止	
	84	08	令牌已过期	
报文错误	90	xx	报文错误	
	90	01	报文不正确	
	90	02	报文校验错	
	90	03	未授权的访问	
	90	04	没有指定的服务	
	90	05	服务参数错误	

D.5 应用接口

认证系统应提供 WEB Service 接口和 Socket 接口。
