第3章

开发准备

虽然第1章与第2章的内容有些枯燥，但理解安全算法与安全协议是开发基于TPM应用系统的前提条件。关于底层安全算法的开发确实不如编写Web应用程序或手机游戏让人兴奋，甚至是非常折磨人的，经常需要处理各种协议规则、编码、解码以及加解密等。但好的方面是，一旦掌握了其中的规律，对安全架构的理解就进入了全新的阶段。

为了照顾不同层次、不同经验的读者，本书不过多地介绍TPM的相关概念，而是直接准备搭建开发环境。有关TPM的理论知识，会逐步分散到本书各章中，这种编排结构降低了阅读理解的难度。

本章首先简要介绍TPM的发展历史，以及可供开发人员选择的应用程序接口(API)，随后开始搭建开发环境。

3.1 初识TPM

虽然本章不会深入介绍TPM的工作原理，但是有关TPM的一些基本术语以及API类型，有必要稍作了解。

3.1.1 什么是TPM

TPM是一种安全算法处理的国际标准，基于专用的硬件电路模块提供安全算法与密钥管理能力。

TPM主要实现如下功能：

(1) 安全运算：支持HASH、HMAC、AES等基础摘要或密码运算。

(2) 密钥管理：生成、加密、存储、导入、导出密钥。

(3) 存储容器：提供有限的安全存储空间，存储敏感数据。

(4) 随机生成：真正的硬件随机生成器，比软件算法更安全。

(5) 可信证据：生成证据链，为设备或系统提供状态完整性监测。

(6) 安全防护：基于硬件的安全防护，防篡改、防木马、防攻击。

3.1.2 TPM 历史

TPM 技术规范由可信计算组织(Trusted Computing Group,TCG)设计与编写。TCG 正式成立于 2003 年,其前身是可信计算平台联盟(Trusted Computing Platform Alliance,TCPA),最早可追溯到 1999 年,成员包括 Intel、AMD、IBM、Microsoft、Cisco 等公司。2011 年,TCG 发布了 TPM 1.2 版本规范; 2016 年,TPM 2.0 版本规范正式发布。

TPM 2.0 与 TPM 1.2 版本在架构层面有很大不同,本书将完全基于 TPM 2.0 版本进行介绍。如无特殊声明,本书将 TPM 2.0 统一简称为 TPM。

3.1.3 编程接口

TPM 软件栈(TPM Software Stack,TSS)是 TPM 官方发布的 API 实现规范,包含了从低级到高级的不同类型的 API 实现方式,例如最底层的 TPM Command API,中间层的 SAPI(System API)、ESAPI(Enhanced System API)和较高层的 FAPI(Feature API)。虽然 TSS API 分为多种类型,但它们的作用是类似的。API 越低级,实现方式就越接近原始的 TPM 技术规范,功能也越强大; API 越高级,封装的结构就越多,使用也越方便。

如果曾经尝试使用过 TPM 的原始 API(TPM Command API),就能深刻体会到这是一场噩梦,即需要手工构建命令请求、解析响应、创建策略会话以及验证数据完整性等,不仅复杂而且容易出错。FAPI 的出现让 TPM 的开发过程变得相对简单,它实现了大部分的 TSS 规范,封装了底层复杂的通信过程与数据结构,可以满足大多数应用系统的开发需求。本书将完全基于 FAPI 进行介绍。

Microsoft 公司基于 TSS 规范实现了一套较为完整的 FAPI,支持主流的 C++、C#、Java、Python 等语言。C++与 C#版本的 API 对于 TSS 规范的实现程度最为完整,而其他语言的 API 实现 TSS 规范的程度相对较弱。虽然本书的示例代码覆盖了 C++与 C#双语版本,但还是建议尽量选择 C++版本。毕竟,TPM 开发是面向底层硬件的交互过程,而 C++正是一种较为底层的编程语言。语言越高级,使用越方便,但损失的能力与性能也越多,灵活性也越弱。Microsoft 公司发布的这套 FAPI 已经实现了高度抽象与封装,同时提供了较强的灵活性。

Microsoft TSS API 可以从以下网址下载:

```
https://github.com/microsoft/TSS.MSR
```

3.2 准备工作

在正式开始 TPM 开发之前,还有一些准备工作需要完成。准备工作根据开发设备与开发语言有所不同。

3.2.1 TPM 芯片

TPM 是一种技术规范,不具有强制性。对于 TPM 规范的解读、实现方式以及实现程度,由各个芯片制造商自行决定。制造商出于成本或某些因素考虑,可能没有按照标准的

TPM 规范实现全部功能，或对规范的实现方式有局部调整。

确认开发设备是否安装有 TPM 芯片以及了解 TPM 芯片对于规范的实现程度，是准备工作的第一步。

在安装有 Windows 操作系统的计算机上，单击“开始”菜单，在弹出的菜单中选择“运行”命令，输入 tpm.msc，单击“确定”按钮，进入 TPM 管理控制台，如图 3-1 所示。在此界面可以进行如下操作：

(1) 初始化：TPM 在首次使用之前需要进行初始化，此过程通常由操作系统自动完成。

(2) 重置：清除 TPM 中存储的密钥与数据，类似恢复出厂设置。

(3) 查看：查看 TPM 芯片制造商与版本。

图 3-1　TPM 管理控制台

3.2.2　TPM 模拟器

TPM 模拟器以纯软件的方式完整地实现了 TPM 技术规范，可以在 x86 架构计算机上协助开发、测试、调试 TPM 的相关功能。

TPM 模拟器主要有以下特点：

(1) 支持跨平台开发，将模拟器部署在一台 Windows 服务器上，其他计算机(Linux、macOS 等)通过 TCP/IP 网络远程连接，进行远程开发。

(2) 运算速度比 TPM 芯片快很多。

(3) 易于恢复出厂设置。

(4) 支持通过源代码断点调试。

预编译的模拟器可以从以下网址下载：

```
https://www.microsoft.com/en-us/download/confirmation.aspx?id=52507
```

模拟器的使用方式非常简单，只需解压缩后双击即可。模拟器运行后将侦听 TCP 2321 端口，如图 3-2 所示，同时将在模拟器程序主目录生成名为 NVChip 的文件，用于模拟 TPM 的内存。如果想重置模拟器，只需简单地删除 NVChip 文件即可。

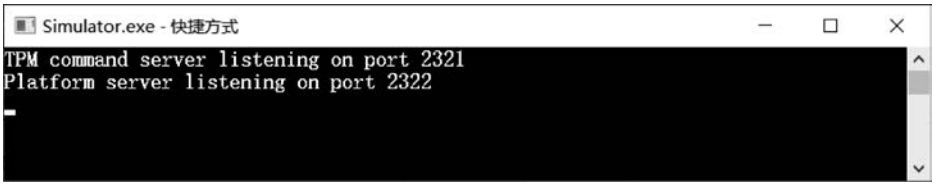

图 3-2　运行 TPM 模拟器

3.2.3　C++开发环境

如果选择 C++语言的 API，那么最低的 IDE 版本建议为 Visual Studio 2017。本节以 Visual Studio 2019 为例，按照如下步骤详细介绍 C++ API 的准备工作：

(1) 启动 Visual Studio 开发工具，打开从 GitHub 下载的 TSS. CPP 项目，TSS. CPP 解决方案包含 TSS. CPP 项目与 TSS. CPP Samples 项目；编译 TSS. CPP 项目，如果一切顺利，在 Debug 目录将看到新生成的 TSS. CPP. dll 与 TSS. CPP. lib 文件。

(2) 新建项目，模板类型选择 C++\Windows\控制台应用，单击“下一步”按钮。

(3) 填写项目名称 TPMDemo 与存储路径，单击“创建”按钮。

(4) 将步骤(1)生成的 TSS. CPP. lib 文件复制到新项目的 lib 目录中，将 include 目录复制到新项目的根目录中。

(5) 打开项目属性，在“TPMDemo 属性页”对话框中选择 “链接器”→“输入”命令，然后在“附加依赖项”文本框新增 TSS. CPP. lib 文件的路径，例如 lib\tss. cpp. lib，如图 3-3 所示。

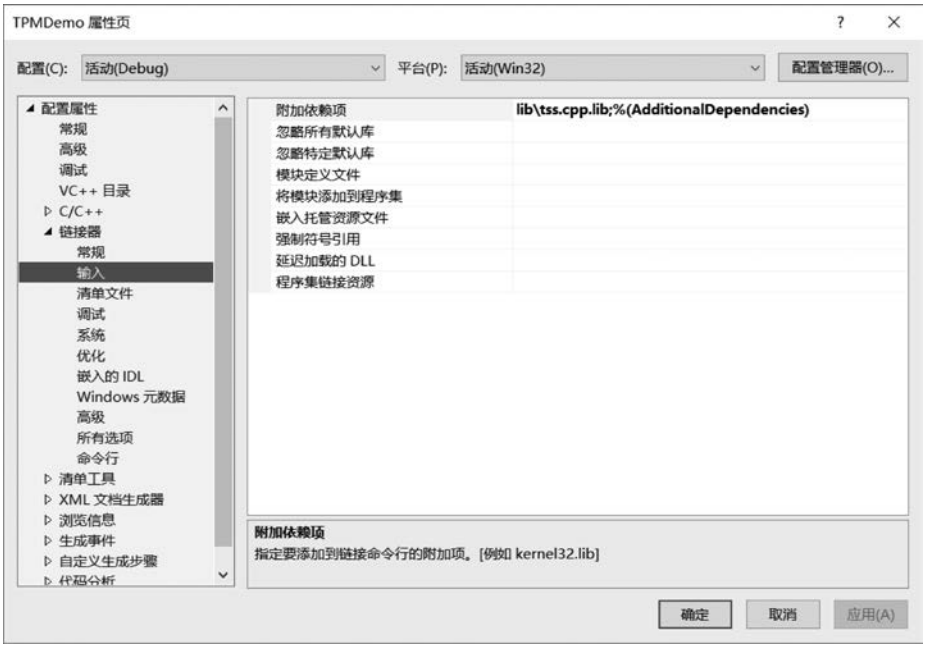

图 3-3　C++项目配置链接器

（6）在“TPMDemo 属性页”对话框中选择 C/C++→“常规”命令，然后在“附加包含目录”文本框新增 include 目录，如图 3-4 所示。

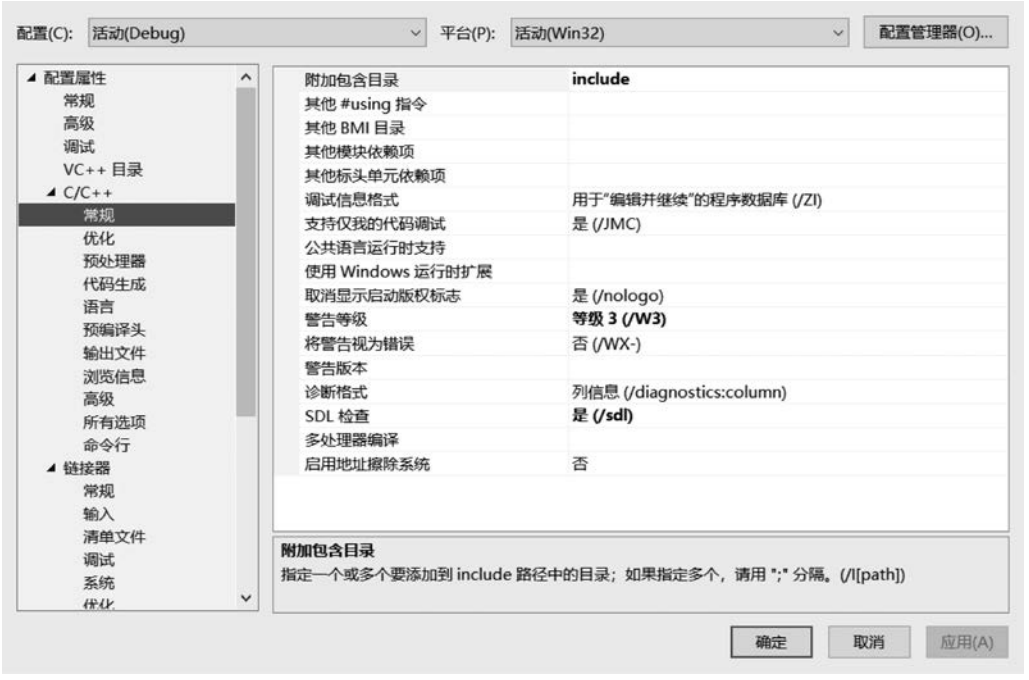

图 3-4　C++项目配置包含目录

（7）将默认的 cpp 文件替换为代码 3-1。

代码 3-1　使用 C++连接 TPM 模拟器

```
#include <iostream>
#include "Tpm2.h"
using namespace TpmCpp;

int main()
{
    TpmDevice* device = new TpmTcpDevice("127.0.0.1", 2321);
    if (!device || !device->Connect())
    {
        throw runtime_error("Can't connect to TPM.");
    }
    device->PowerCycle();
    std::cout << "Connected to TPM!\n";
    device->PowerOff();
    delete device;
}
```

（8）编译项目，将步骤(1)生成的 TSS.CPP.dll 文件复制到应用程序的输出目录中。

（9）运行模拟器，然后运行项目。如果一切顺利，将看到如图 3-5 所示的输出信息，表示已经成功连接 TPM 模拟器。

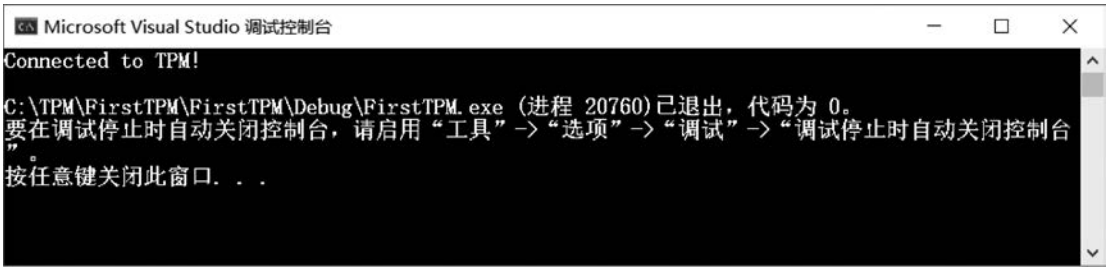

图 3-5　使用 C++连接 TPM 模拟器

(10) 如果连接硬件 TPM 芯片，可以使用代码 3-2。

代码 3-2　使用 C++连接 TPM 芯片

```
#include <iostream>
#include "Tpm2.h"
using namespace TpmCpp;

int main()
{
    TpmDevice* device = new TpmTbsDevice();
    if (!device || !device->Connect())
    {
        throw runtime_error("Can't connect to TPM.");
    }
    std::cout << "Connected to TPM!\n";
    delete device;
}
```

3.2.4　C#开发环境

如果选择 C#语言的 API，除了需要安装 Visual Studio 2017 或以上版本之外，还需要安装.NET Framework 4.7.2 或以上版本。本节以 Visual Studio 2019 为例，按照如下步骤详细介绍 C# API 的准备工作：

(1) 启动 Visual Studio 开发工具，打开从 GitHub 下载的 TSS.NET 项目，TSS.NET 解决方案包含 TSS.NET 类库项目与 Samples 项目集；编译 TSS.NET 项目，如果一切顺利，在 Debug 目录将看到新生成的 TSS.NET.dll 文件。

(2) 新建项目，模板类型选择 C#\Windows\控制台应用(.NET Framework)，单击“下一步”按钮。

(3) 填写项目名称 TPMDemoNET 与存储路径，单击“创建”按钮。

(4) 将步骤(1)生成的 TSS.NET.dll 文件复制到新项目的根目录中。

(5) 在“解决方案资源管理”窗口右击“引用”，在弹出的快捷菜单中选择“添加引用”命令；在“引用管理器”对话框将 TSS.NET.dll 文件添加至项目引用。

(6) 将默认的 cs 文件替换为代码 3-3。

代码 3-3 使用 C# 连接 TPM 模拟器

```
using System;
using Tpm2Lib;
namespace TPMDemoNET
{
    class Program
    {
        static void Main(string[] args)
        {
            Tpm2Device device = new TcpTpmDevice("127.0.0.1", 2321);
            device.Connect();
            device.PowerCycle();
            Console.WriteLine("Connected to TPM");
        }
    }
}
```

(7) 运行模拟器,然后运行项目。如果一切顺利,将看到如图 3-6 所示的输出信息,表示已经成功连接 TPM 模拟器。

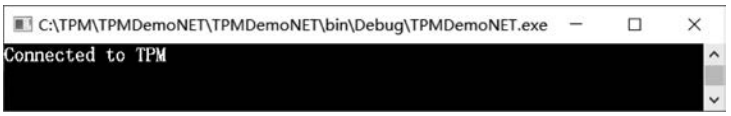

图 3-6 使用 C# 连接 TPM 模拟器

(8) 如果连接硬件 TPM 芯片,可以使用代码 3-4。

代码 3-4 使用 C# 连接 TPM 芯片

```
using System;
using Tpm2Lib;
namespace TPMDemoNET
{
    class Program
    {
        static void Main(string[] args)
        {
            Tpm2Device device = new TbsDevice();
            device.Connect();
            Console.WriteLine("Connected to TPM");
        }
    }
}
```

3.3 测试 TPM

如果没有完全理解 3.2 节的示例代码,也不用担心,第 4 章将正式讲解 TSS API 的相关代码,但在这之前,有必要确认 TPM 芯片能够支持哪些安全算法。

使用 C++查看 TPM 芯片的算法支持能力的过程见代码 3-5。

代码 3-5　使用 C++查看 TPM 芯片的算法支持能力

```
#include <iostream>
#include <iomanip>
#include "Tpm2.h"
using namespace TpmCpp;

int main()
{
    // 连接 TPM,略

    UINT32 index = 0;
    while (true)
    {
        auto resp = tpm.GetCapability(TPM_CAP::ALGS, index, 8);
        TPMU_CAPABILITIES* cabs = resp.capabilityData.get();
        TPML_ALG_PROPERTY* props = dynamic_cast<TPML_ALG_PROPERTY*>(cabs);
        vector<TPMS_ALG_PROPERTY> ps = props->algProperties;
        for (auto p = ps.begin(); p != ps.end(); p++)
        {
            string name = EnumToStr(p->alg);
            string prop = EnumToStr(p->algProperties);
            std::cout << setw(16) << name << ": " << prop << endl;
        }
        if (!resp.moreData)
            break;
        index = (ps[ps.size() - 1].alg) + 1;
    }
}
```

使用 C# 查看 TPM 芯片的算法支持能力的过程见代码 3-6。

代码 3-6　使用 C# 查看 TPM 芯片的算法支持能力

```
using System;
using Tpm2Lib;
namespace TPMDemoNET
{
    class Program
    {
        static void Main(string[] args)
        {
            // 连接 TPM,略

            ICapabilitiesUnion caps;
            tpm.GetCapability(Cap.Algs, 0, 1000, out caps);
            var algs = (AlgPropertyArray)caps;
            foreach (var alg in algs.algProperties)
            {
                string s = string.Format("{0} | {1}",
                    alg.alg.ToString().ToUpper(),
                    alg.algProperties);
```

```
                Console.WriteLine(s);
            }
        }
    }
}
```

程序运行结果如图 3-7 所示，通过控制台窗口看到这款 TPM 芯片没有完全支持 TPM 规范中的全部算法，例如 SHA-256 算法得到了支持，但是缺少 SHA-384 或 SHA-512 算法。如果执行以下代码，将遇到错误代码为 131 的异常。

```
tpm.Hash(data, TPM_ALG_ID::SHA384, TPM_RH_NULL);
```

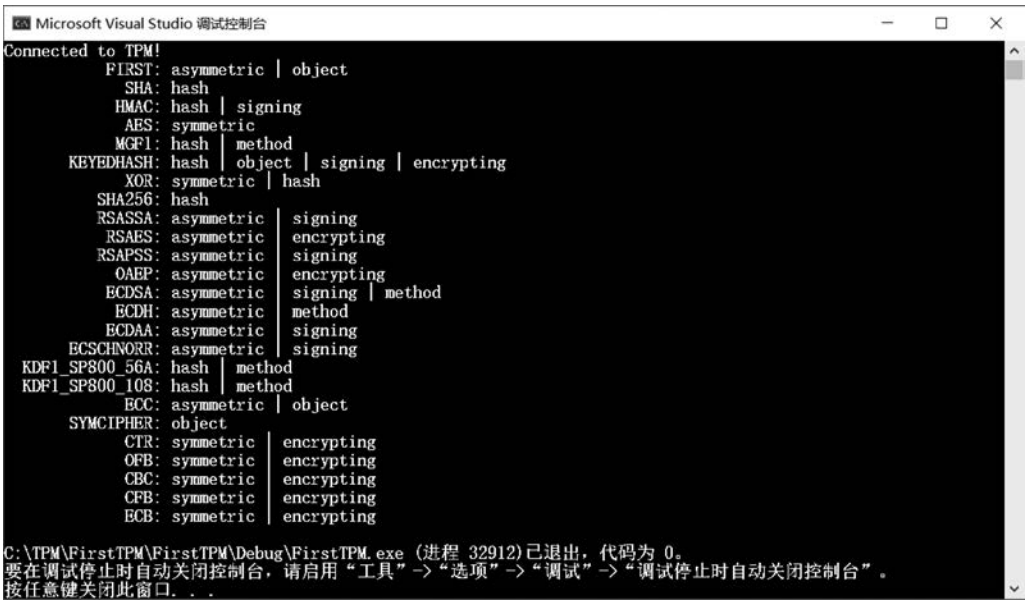

图 3-7 查看 TPM 芯片的算法支持能力

在设计应用系统时，应当首先确定目标设备的 TPM 芯片制造商与算法支持能力，尽可能选择符合安全需求同时适用于多数硬件平台的安全算法。

3.4 本章小结

本章首先简要回顾了 TPM 的发展历史，介绍了一些常用的 TSS API 类型以及各自的特点。之所以选用 FAPI 作为本书的重点讲解对象，是因为其兼顾了功能性与便捷性。随后介绍了 TPM 模拟器的使用方式，并分别基于 C++与 C#语言演示了与 TPM 建立连接的方法。最后演示了如何查看 TPM 支持的算法集，这对于系统设计来说至关重要。

尽管 TPM 模拟器使用起来十分方便，但为了最大程度贴近真实的生产环境，如无特殊说明，本书的示例程序将默认基于硬件 TPM 芯片编写。