MICROCHIP 2019 MASTERS

中国技术精英年会(第二十届)

C20L16 BLU15

如何实现基于 BLE 的 UART 安全透明传输

MICROCHIP

SMART | CONNECTED | SECURE

C20L16 BLU

前提条件

在参加本课程之前，学员应先参加以下课程：

- **C20H04**安全加密基础及如何使用**Microchip**安全芯片实现安全加密应用

课程目标

- 说明安全层添加到无线链路的重要性
- 演示如何使用库快速编译适用于**Android**和**iOS**智能手机应用程序以支持基于**BLE**的**UART**传输
- 展示如何设置和使用库的安全层来验证和加密通信

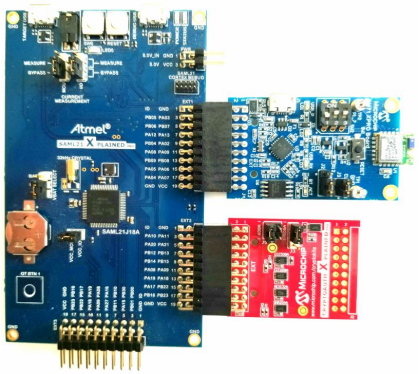
课程安排

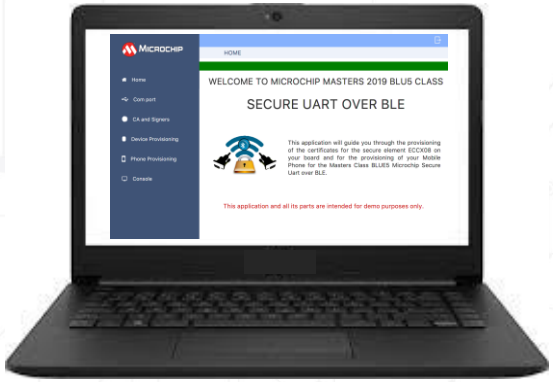
- 基本概念
- 基于BLE的UART安全传输
  - 蓝牙LE安全概述
  - 为什么要使用不同的方法？
  - 实现基于BLE的UART安全传输
  - ECCX08概述和使用
- 演示
  - 配置ECC和Android手机并建立安全连接

基于BLE的UART安全传输

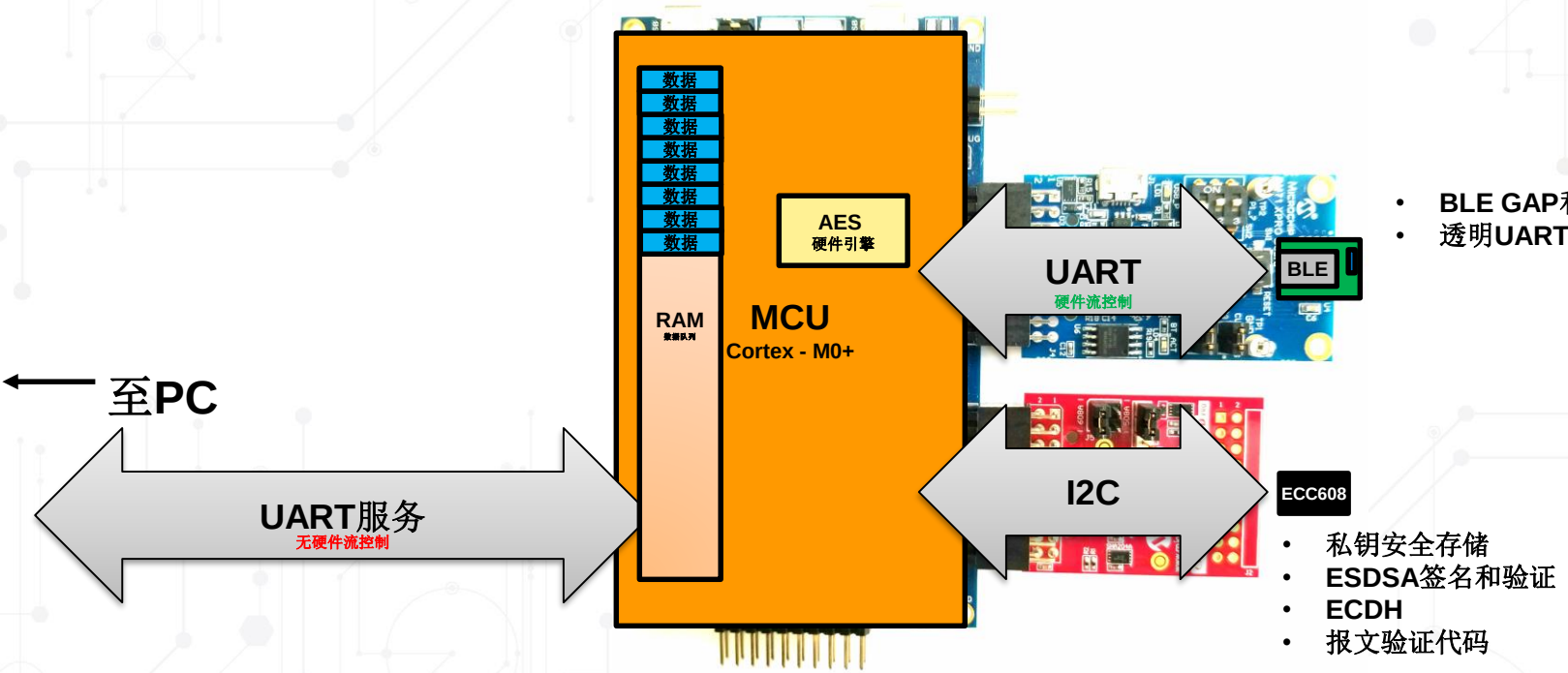
今天将用到：

- 评估板
  - SAML21 Xplained Pro
  - BM71 XPRO
  - ATCRYPTOAUTH-XPRO-B
- 手机
  - BLU Android 6.0
- **PC**
  - Windows 10，预装有
    SUOB Java应用程序





基于BLE的UART安全传输



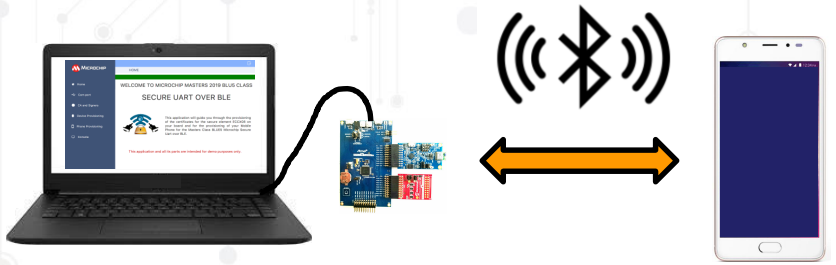
- BLE GAP和GATT
- 透明UART服务

- 私钥安全存储
- ESDSA签名和验证
- ECDH
- 报文验证代码

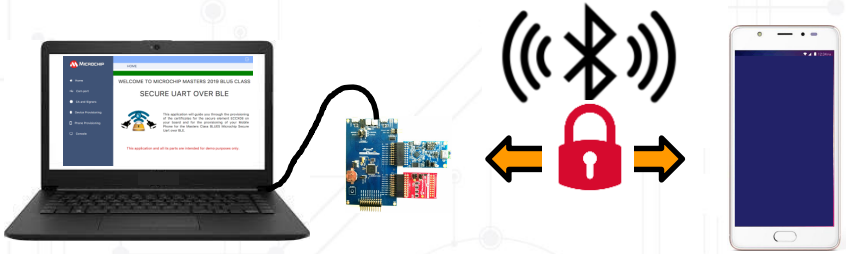
基于BLE的UART安全传输

我们想要实现的目标

- 按照与普通UART上一样的方式在评估板和智能手机应用程序之间传输字节



- 对手机和评估板进行验证并在评估板和智能手机应用程序之间以加密方式传输字节



基本概念

Bluetooth® LE

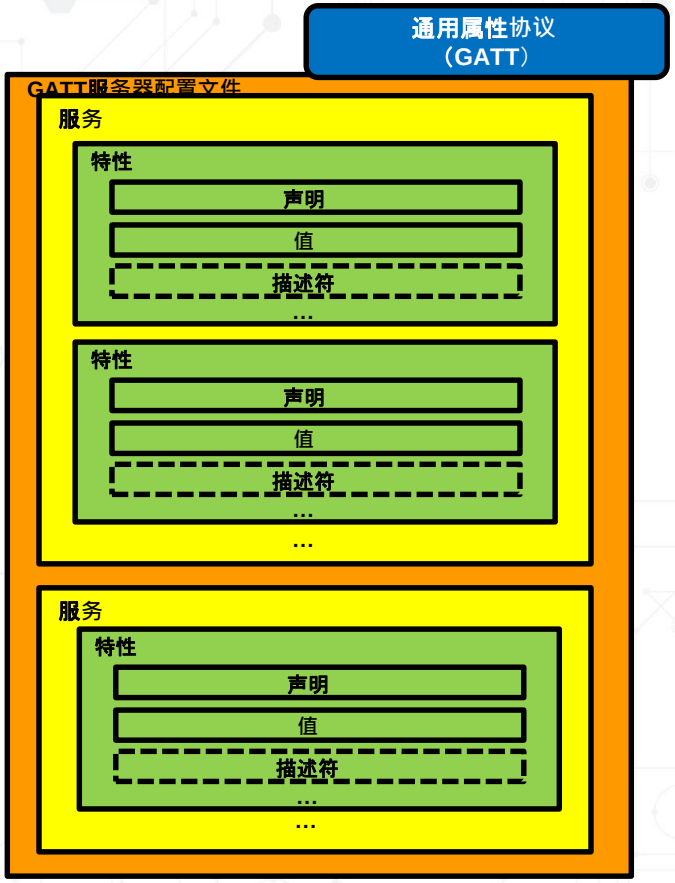
- 通用访问配置文件（GAP）
  - 通告和连接
  - 中央设备和外设
- 通用属性配置文件（GATT）
  - 数据传输
  - 客户端和服务器
  - 服务和特性

通用属性配置文件（GATT）

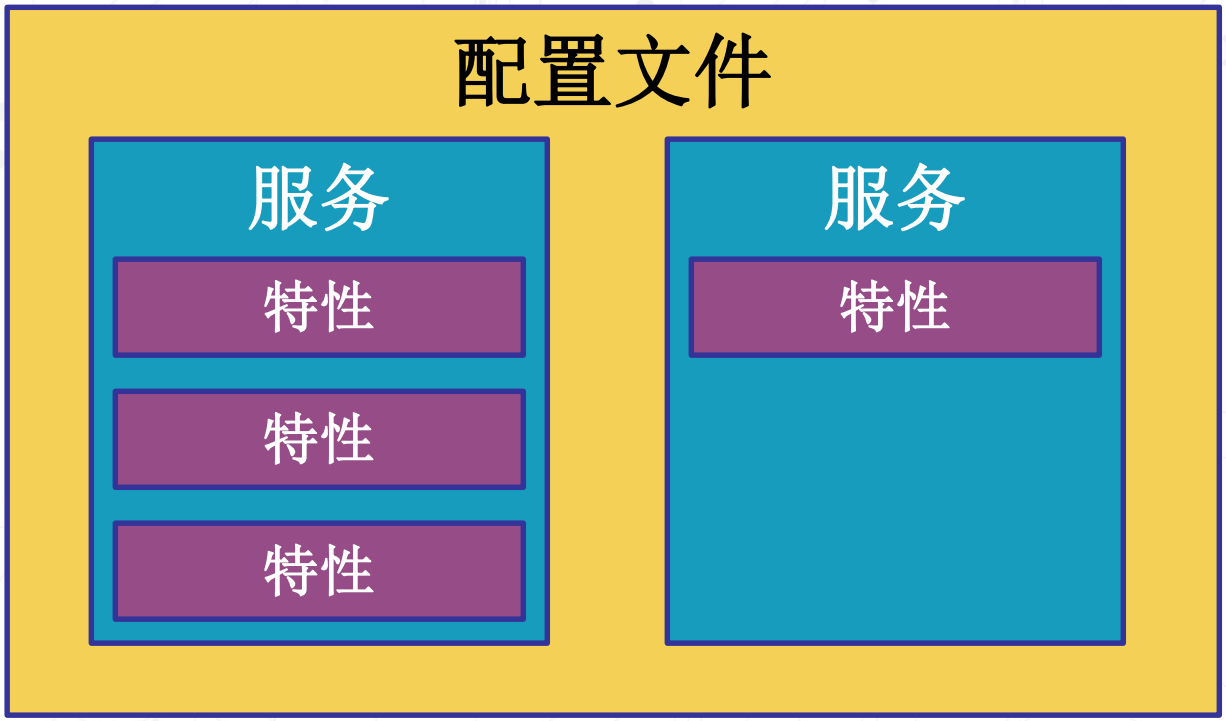
- 配置文件是服务的集合
- 一个服务包含一个或多个名为“特性”的属性
- 一个特性就是通过Bluetooth®发送的单个数据元素
- 由UUID标识
  - 通用惟一标识符

通用属性协议

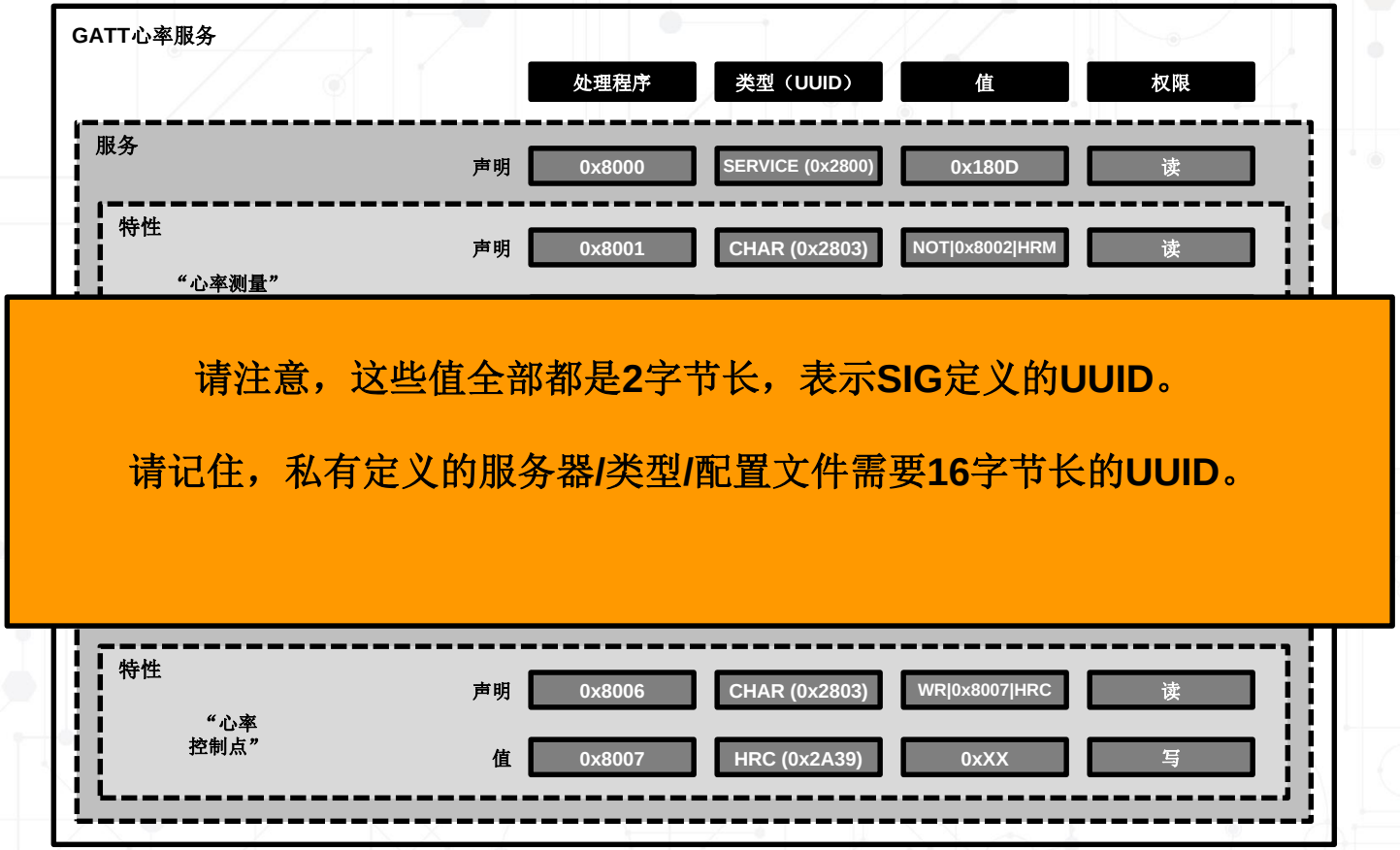
- GATT将ATT属性组织成组——称为服务
- GATT服务器配置文件中的属性的分组如下：
  - 服务
    - 特性
      - 声明属性
      - 值属性
      - 描述符属性
- 上述全部是ATT属性！



服务和特性



GATT配置文件示例
心率服务



请注意，这些值全部都是2字节长，表示SIG定义的UUID。

请记住，私有定义的服务器/类型/配置文件需要16字节长的UUID。

特性

- BLE的主要功能是围绕特性构建的
- 服务器维持一个服务中的特性值
- 客户端读/写服务器保持的特性值
- 服务器可以向客户端通知发生的变化（如果客户端使能）

基于BLE的UART安全传输

SecureUARTOverBLE库

- SecureUARTOverBLE库为开发人员提供了一个单独的类，其中包含处理蓝牙低功耗连接以及通过透明UART服务进行发送和接收的所有内容。

基于BLE的UART安全传输

SecureUARTOverBLE库

它可自主管理：

- **BLE**适配器和服务初始化
- 许可请求
- 外设扫描和通告列表
- 连接和断开
- 服务发现
- 特性发现和通知使能
- 异步发送和接收
- 基于**MTU**大小的队列管理，用于长时间传输
- 吞吐量测量
- 安全生成和私钥存储
- 证书生成和管理（签名、验证和压缩）
- 对称和非对称加密（**ECDSA**、**ECDH**、**SHA256**和**AES128**）
- 多线程通信
- 部分**UI**

基于BLE的UART安全传输

SecureUARTOverBLE库

API：

少于20个API…

Send(byte[] bytes, **SecureUartOverBLE_txDoneCallback** cb)
This is one of function to send bytes over the Secure Uart Over BLE.

SetProtocolDisabled()
Use this function to disable SecureUartOverBLE protocol and make bytes transfer without any header or protocol

SetReceiveBytesCallback(**SecureUartOverBLE_receiveCallback** rcvcallback)
This function sets a new callback to receive data that are sent on the Secure Uart Over BLE.

StartScanActivity(**SecureUartOverBLE_readyCallback** readyCallback)
Once a bleobject has been initialized you can call this API to launch a Scan Activity that will
show you the peripherals in range base on the scan filters used (if any)
When you will tap on a peripheral, the library will attempt to connect to it and will report the
result of this attempt in readyCallback

StartUSBProvisioning(boolean uiactive, **SecureUartOverBLE_provisionDoneCallback** callback)
Used only for tutorial purposes, it opens an Intent receiver to provision the device through USB

StopUSBProvisioning()
Used only for tutorial purposes, it stops USB provision previously initiaded by
public void StartUSBProvisioning(boolean uiactive, final SecureUartOverBLE_provisionDoneCallback callback)

AddScanFilter(int filtertype, java.lang.String filterString)
Prepare a scan filter to include only specific peripherals in the scan results set
it can be called multiple times: filters will work in AND
You can then remove filters with RemoveAllScanFilters()

ClearProvision()
Cancels all certificates of previous provisioning

Connect(java.lang.String address, java.lang.Boolean security, **SecureUartOverBLE_readyCallback** readyCallback)
This function can only be used after a successfull initialization of the ble object
It will start a connection attempt to the peripheral with MAC address equel to parameter "address"

DeInitialize()
This is the De-Initialization function.

Disconnect()
This instructs the BLE object to disconnect from the peripheral is connected to

Initialize(**SecureUartOverBLE_initCallback** callback)
Main initialization function: this MUST be called after the object has been instantiated and represent the beginning of all BLE initialization

IsPhoneProvisioned()
Checks if phone contains a valid Certificate Chain

Provisioning_GetPhonePublicCertificateZip()
EXTERNAL PROVISIONING FUNCTION: This function is the open door to the AndroidKeyStore to retreive the public certificate of the smart device.

Provisioning_SetCertificateChain(byte[] bytes)
EXTERNAL PROVISIONING FUNCTION: This function is the open door to the AndroidKeyStore to inject a new Certificate Chain obtained with an external function, coming from an arbitrary provisioning system

RemoveAllScanFilters()
Remove all scan filters added with public void AddScanFilter(int filtertype, String filterString)

ResetSavedMacAddress()
Clears the stored MAC address used in previous connection

Send(byte[] bytes)
This is one of function to send bytes over the Secure Uart Over BLE.

Send(byte[] bytes, boolean showprogress, **SecureUartOverBLE_txDoneCallback** cb)
This is one of function to send bytes over the Secure Uart Over BLE.

基于BLE的UART安全传输

SecureUARTOverBLE库

基本应用程序只有6个API：

- 构造函数： SecureUARTOverBLE bleObject = new SecureUARTOverBLE(activitycontext);
- 初始化程序： bleObject.Initialize(null);
- 连接： bleObject.Connect(null, false, null);
- 发送： bleObject.Send(bytes);
- 接收： bleObject.SetReceiveBytesCallback(new Receive_Callback());
- 取消初始化： bleObject.DeInitialize();

MICROCHIP
MASTERs 2019

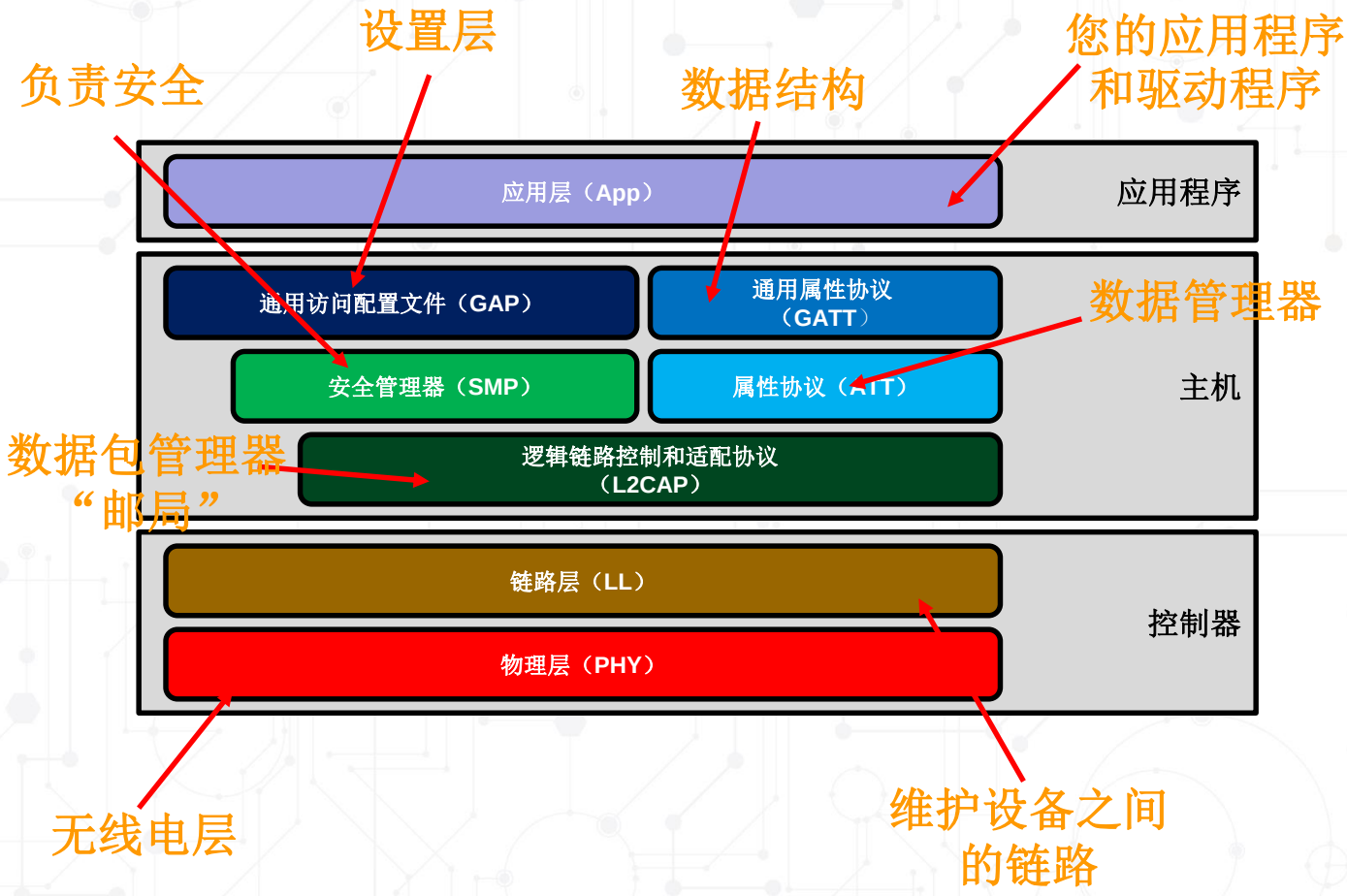
基于BLE的UART安全传输

课程安排

- 基本概念
- 基于BLE的UART安全传输
  - **蓝牙LE安全概述**
  - 为什么要使用不同的方法？
  - 实现基于BLE的UART安全传输
  - ECCX08概述和使用
- 演示
  - 配置ECC和Android™手机并建立安全连接

基于BLE的UART安全传输

蓝牙LE安全概述



基于BLE的UART安全传输

蓝牙LE安全概述

- 报文使用在配对阶段交换的对称密钥（**LTK**）进行加密，并最终存储（**BONDING**）起来以用于后续连接
- 主要安全实现：
  - LE传统连接（4.0、4.1和4.2设备）
  - LE安全连接（4.2和5）
- 通常，配对用于交换临时密钥并生成短期密钥。**STK**将用于加密后续报文以生成和交换**LTK**（长期密钥）、**CSRK**（连接签名解析密钥）和**IRK**（身份解析密钥）。
- **LE**安全连接引入了**ECDH**协议来交换提高整体安全性的常见机密信息

基于BLE的UART安全传输

蓝牙LE安全概述

配对方法：

- **直接连接（Just Works）**（TK为0，因此STK进行明文交换）
- **带外（OOB）配对**（在此方法中，TK使用NFC等不同的无线技术进行交换）
- **密钥（PassKey）**（在此方法中，TK是用户在设备之间传送的6位数字）

课程安排

- 基本概念
- 基于BLE的UART安全传输
  - 蓝牙LE安全概述
  - **为什么要使用不同的方法？**
  - 实现基于BLE的UART安全传输
  - ECCX08概述和使用
- 演示
  - 配置ECC和Android™手机并建立安全连接

基于BLE的UART安全传输
为什么要使用不同的方法？

- **关于BLE配对方法的实际注意事项：**
- 在安全应用中使用BLE的一个主要障碍是，即使最安全的配对方法也可能在其他方面存在显著的缺点。OOB配对要求设备具有额外的电路，这会增加设备的成本，而且设计人员还必须保证OOB通道是安全的，这本身就是一个重大的设计挑战。数字比较（Numeric comparison）要求每个设备具有显示功能，这会增加设备的成本，而且需要用户手动验证代码匹配，这对用户体验是不利的。因此，假设大多数设备将使用密钥方法或“直接连接”是合理的，这意味着大多数设备都会有一定程度的漏洞。安全性要求较高的产品（如医疗设备）的设计人员应在其设计中无法实现OOB配对或数字比较（Numeric comparison）时考虑其他无线协议。

https://www.digikey.com/eewiki/display/Wireless/A+Basic+Introduction+to+BLE+Security

基于BLE的UART安全传输

为什么要使用不同的方法？

凭借ECDH密钥交换，LE安全连接中的直接连接配对方法对被动窃听的应对能力要强于LE传统连接中的相同方法。但是，由于该方法无法为用户提供验证连接可靠性的方法，因此仍然容易受到MITM攻击

https://www.digikey.com/eewiki/display/Wireless/A+Basic+Introduction+to+BLE+Security

基于BLE的UART安全传输
为什么要使用不同的方法？

- 在WOOT 2013上，Mark Ryan发表了一篇论文，对BLE可能面临的攻击进行了推测。在他所推测的攻击中，攻击者向一个设备注入伪造的LL_REJECT_IND报文。显然，这将导致接收者忘记当前的长期密钥并强制新的密钥交换来获得新的长期密钥。攻击者可以窃听这一新的密钥交换过程，学习新的长期密钥，并解密所有后续流量

https://www.usenix.org/conference/woot13/workshop-program/presentation/Ryan:

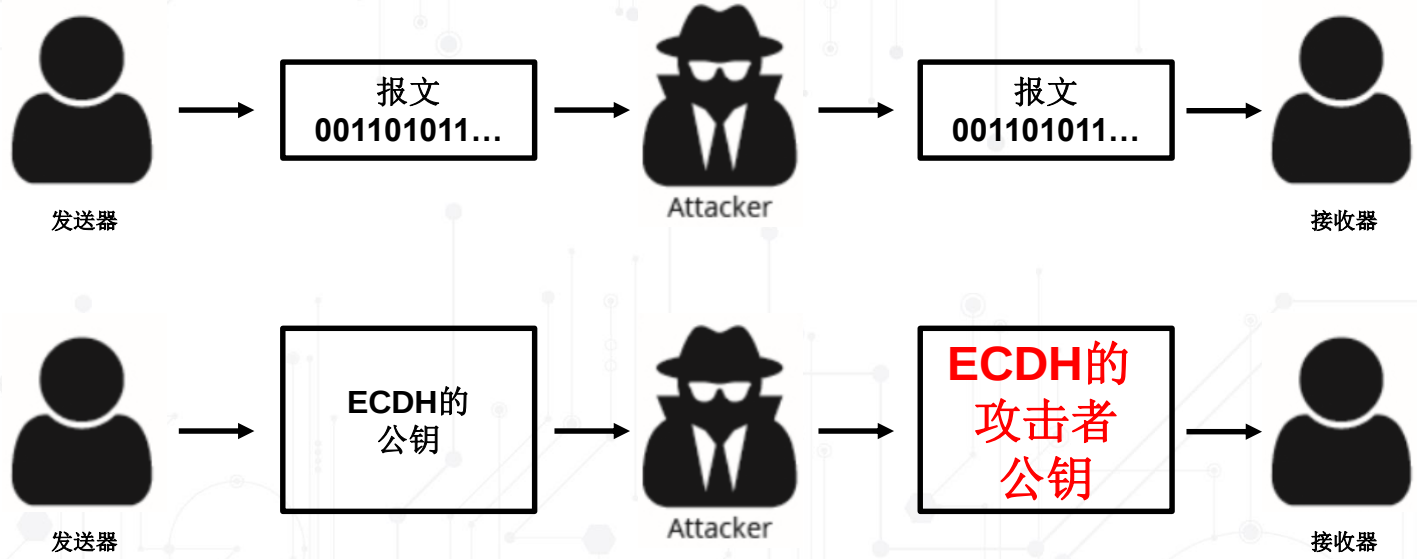
基于BLE的UART安全传输
为什么要使用不同的方法？

- 缺少正确的验证方法，因为它缺少一种在设备上安全地存储经过验证的身份的方法。

- ECDH性能十分出色，但如果公钥交换未经过验证，MITM可以提供自己的公钥并解密所有进一步的通信

基于BLE的UART安全传输
为什么要使用不同的方法？

MITM攻击



基于BLE的UART安全传输

为什么要使用不同的方法?

其他技术在做什么?

TLS 1.2

基于BLE的UART安全传输

为什么要使用不同的方法？

TLS密码套件示例

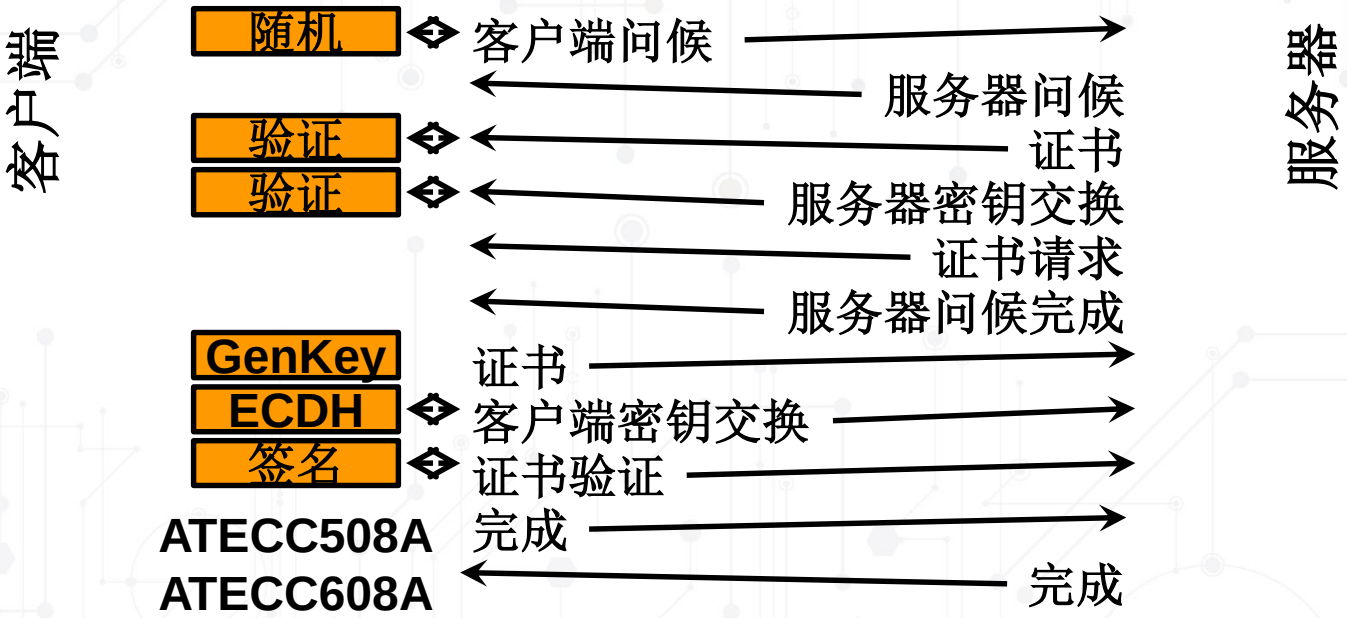
TLS_ECDHE_ECDSA_WITH_AES_128_GCM_SHA256

ECDHE	**密钥交换/协议算法**
ECDSA	**服务器验证算法**
AES	**密码算法**
128	**密码强度**
GCM	**密码模式**
SHA256	**PRF（伪随机函数）**

基于BLE的UART安全传输

为什么要使用不同的方法？

TLS握手——相互



基于BLE的UART安全传输
为什么要使用不同的方法？

我们可以借用一些最佳实践：

- 证书交换和验证（使用ECDSA进行验证）
- ECDH + 临时签名（完整性）
- AES（机密性）

课程安排

- 基本概念
- 基于BLE的UART安全传输
  - 蓝牙LE安全概述
  - 为什么要使用不同的方法？
  - **实现基于BLE的UART安全传输**
  - ECCX08概述和使用
- 演示
  - 配置ECC和Android™手机并建立安全连接

基于BLE的UART安全传输

实现基于BLE的UART安全传输

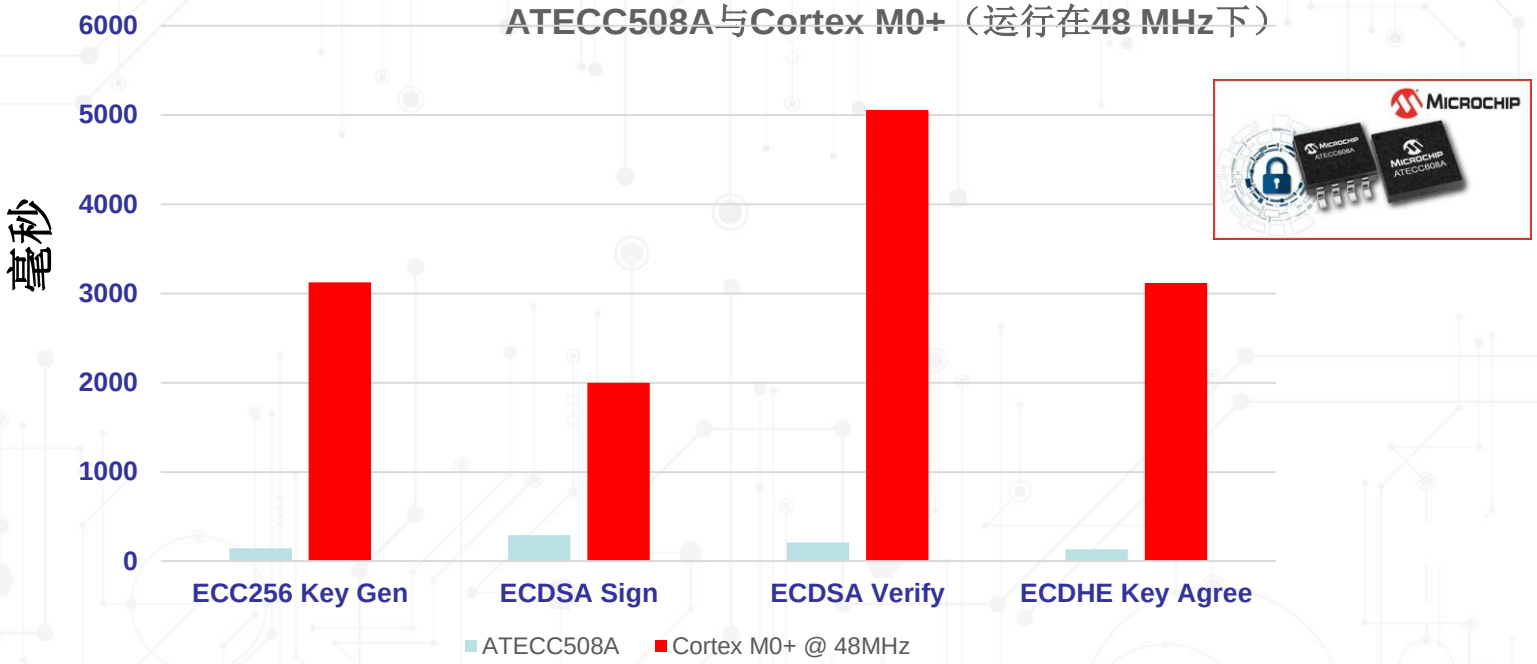
面临的挑战：

- 高计算量算法（嵌入式端）
- 私钥安全存储（两端）
- 低吞吐量链路上的证书长度
- 协议实现和配置（智能设备端）

基于BLE的UART安全传输

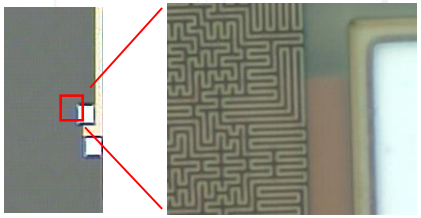
实现基于BLE的UART安全传输

硬件和软件基准



基于BLE的UART安全传输
实现基于BLE的UART安全传输

- **ECCX08**
  - 强大的多级安全性
    - 跨多层的主动屏蔽
    - 所有存储器均在内部加密
    - 执行数据独立的加密
    - 随机化数学运算
    - 内部状态一致性检查
    - 电压防篡改器，隔离电源轨
    - 生成内部时钟
    - 安全的测试方法，无需JTAG
    - 无调试探测点，无测试焊盘
    - 无封装或裸片标识



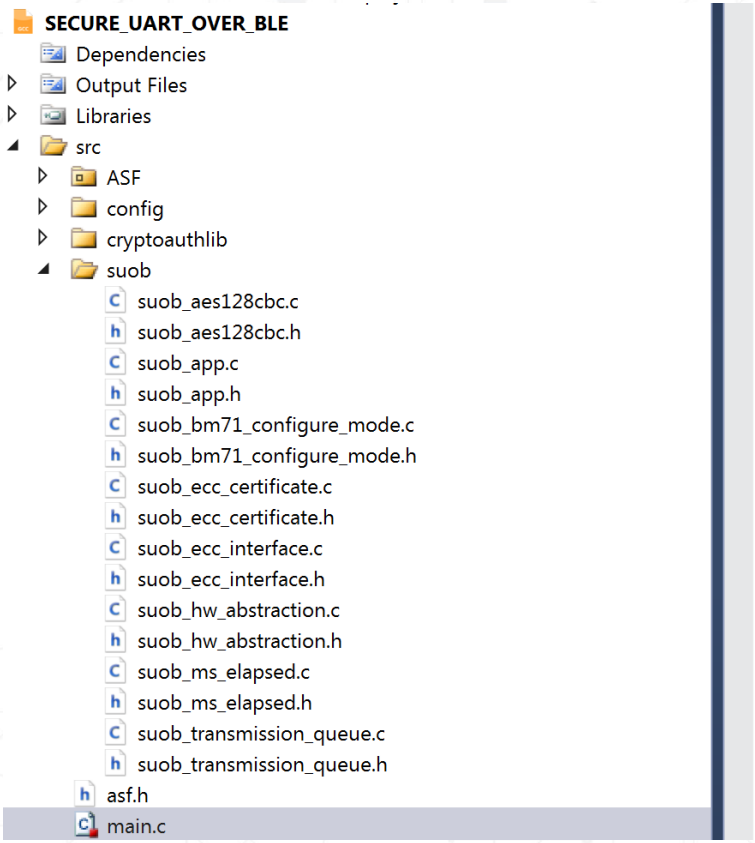
- **iOS安全区域**
  https://developer.apple.com/documentation/security/certificate_key_and_trust_services/keys/storing_keys_in_the_secure_enclave
- **Android™硬件支持的密钥库**
  https://source.android.com/security/keystore

基于BLE的UART安全传输

实现基于BLE的UART安全传输

MCU

源




```c
int main(void)
{
    system_init();
    delay_init();
    system_interrupt_enable_global();
    suob_aes_test();

    suob_app_init_hw();
    while(!SUOB_INIT_COMPLETE);

    while(1)
    {
        suob_APPLICATION();
    }
}
```

基于BLE的UART安全传输

实现基于BLE的UART安全传输

- 我们需要多少证书？
- X509证书采用何种形式？
- 它的典型大小为多少？
- 是否可通过某种方法将其压缩？

基于BLE的UART安全传输

实现基于BLE的UART安全传输

30 82 01 DC 30 82 01 81 A0 03 02 01 02 02 11 00
00 00 00 00 00 00 00 00 00 00 00 00 46 31 41 42
30 0A 06 08 2A 86 48 CE 3D 04 03 02 30 3A 31 1D

649

191

可以使用存储在NVM中的固定模板重建相同的证书，并且实际仅传输160字节

CE 3D 04 03 02 03 49 00 30 46 02 21 00 CE 1D 6B
99 98 F6 EE A8 A0 BD 3C 3A C4 48 00 E0 2F 5F 1E
C9 11 BB 2B C0 E0 FC E6 E2 77 AD ED 7E 02 21 00
FA D1 43 6D CE 84 D4 E9 3D 9A 53 71 04 28 11 0E
30 3F 6F 6F 83 B7 AB 2B 86 75 BE 65 45 00 C2 B2

基于BLE的UART安全传输

实现基于BLE的UART安全传输

基于到目前为止的所有注意事项，回顾一下保护管道和实现整个系统的要求

基于BLE的UART安全传输

实现基于BLE的UART安全传输

配置

- 生成证书颁发机构（私钥-公钥）
- 生成私钥并安全存储到ECC上以及智能设备的信任区内部。
- 将2个设备的相应公钥（证书形式）安全地传输到CA所在的PC
- 生成签名者证书并对证书链进行签名（CA对签名者进行签名，签名者对ECC和智能设备证书进行签名）
- 将新签名的证书、签名者证书和CA公共证书传输回ECC和智能设备

信任区

手机
私钥

CERTIFICATE

手机
由签名者签名

ANDROID密钥存储

手机

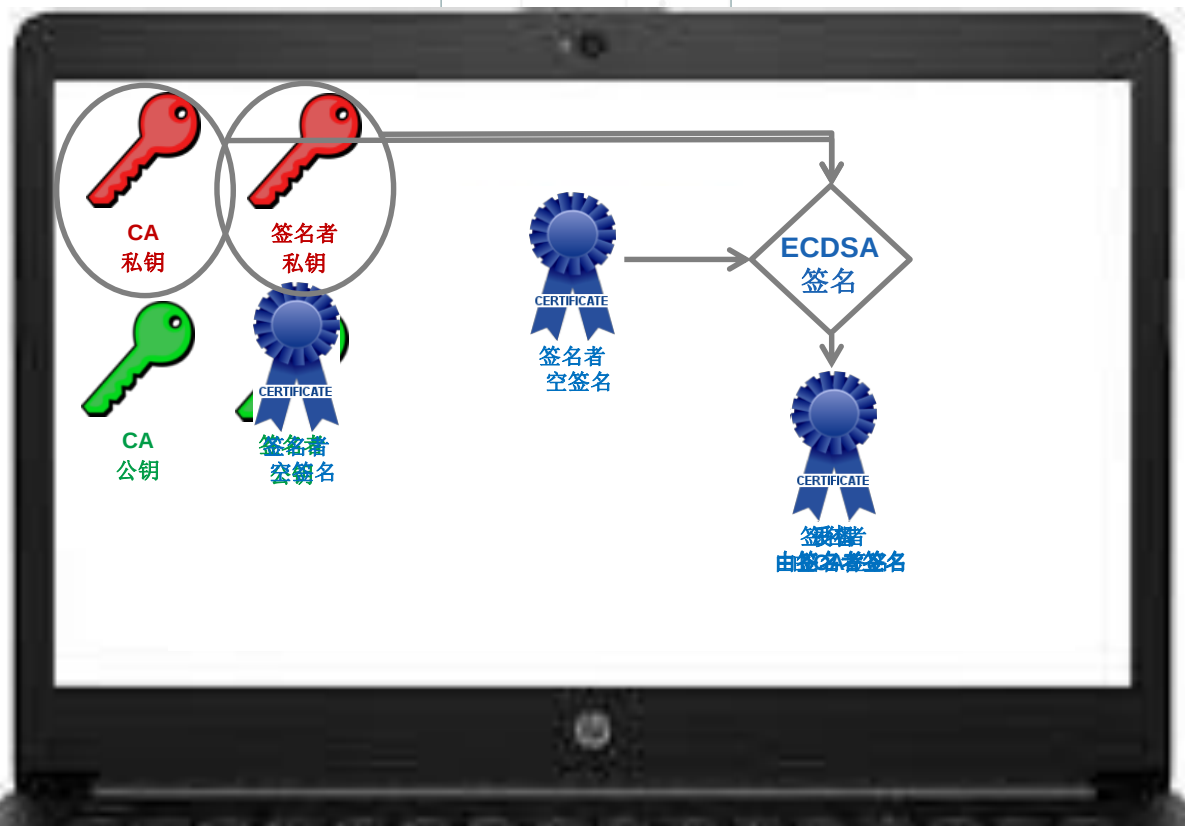
CERTIFICATE

设备
由签名者签名

设备
私钥

ECC608

评估板



信任区

手机
私钥

CERTIFICATE

手机
由签名者签名

CERTIFICATE

签名者
由CA签名

CA
公钥

ANDROID密钥存储

手机

CA
公钥

CERTIFICATE

签名者
由CA签名

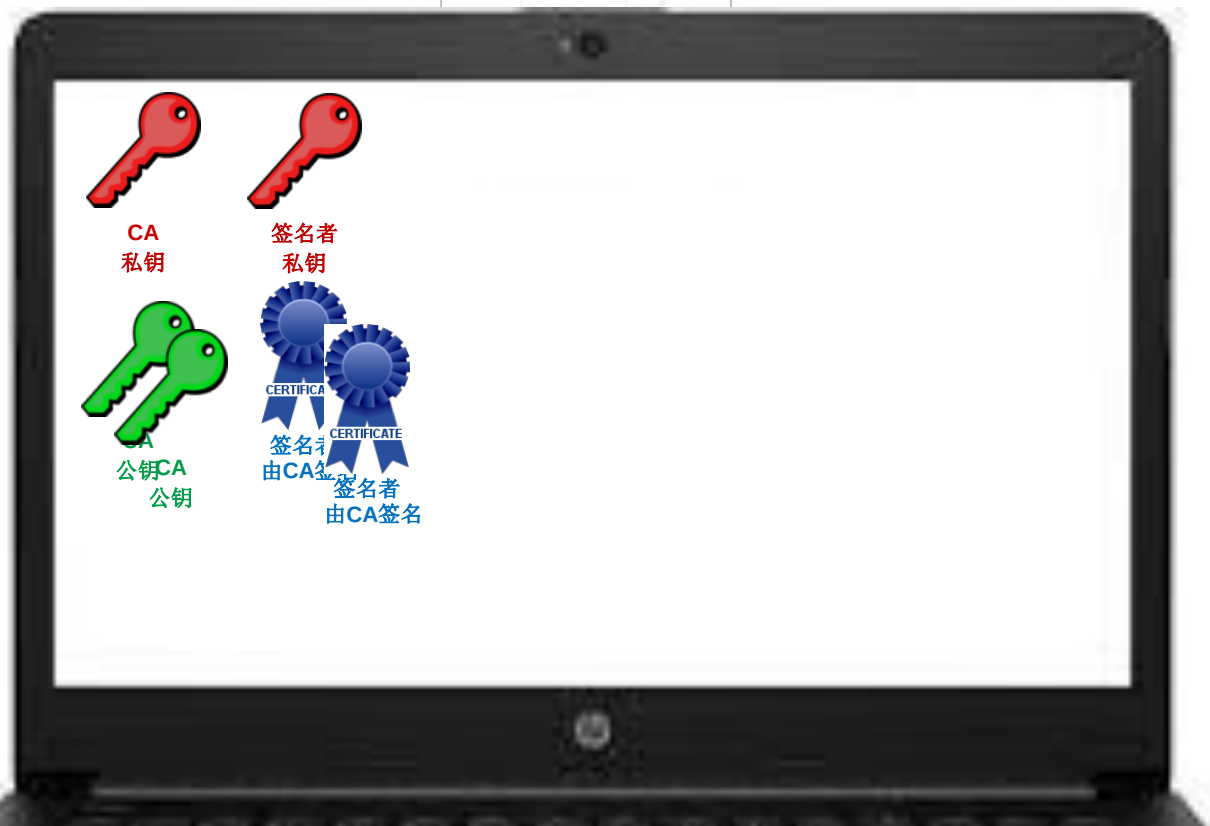
CERTIFICATE

设备
由签名者签名

设备
私钥

ECC608

评估板



手机

信任区

手机
私钥

手机
由签名者签名

签名者
由CA签名

CA
公钥

ANDROID密钥存储

评估板

CA
公钥

签名者
由CA签名

设备
由签名者签名

设备
私钥

ECC608

安全握手

成功连接BLE模块和智能设备后：

- 智能设备会将自己的证书和签名者证书与32字节的随机临时值（类似于客户端问候）一起发送
- MCU将接收2个证书，对它们进行验证并将临时值存储在RAM中；如果出现任何问题（验证失败），MCU将向BLE模块发出复位信号以断开连接。
- 验证成功后，MCU会将自己的证书和签名者证书与32字节的随机临时值（类似于服务器问候）一起发送
- 智能设备将接收2个证书，对它们进行验证并将临时值存储在RAM中；如果出现任何问题（验证失败），智能设备将断开连接。

安全握手（续）

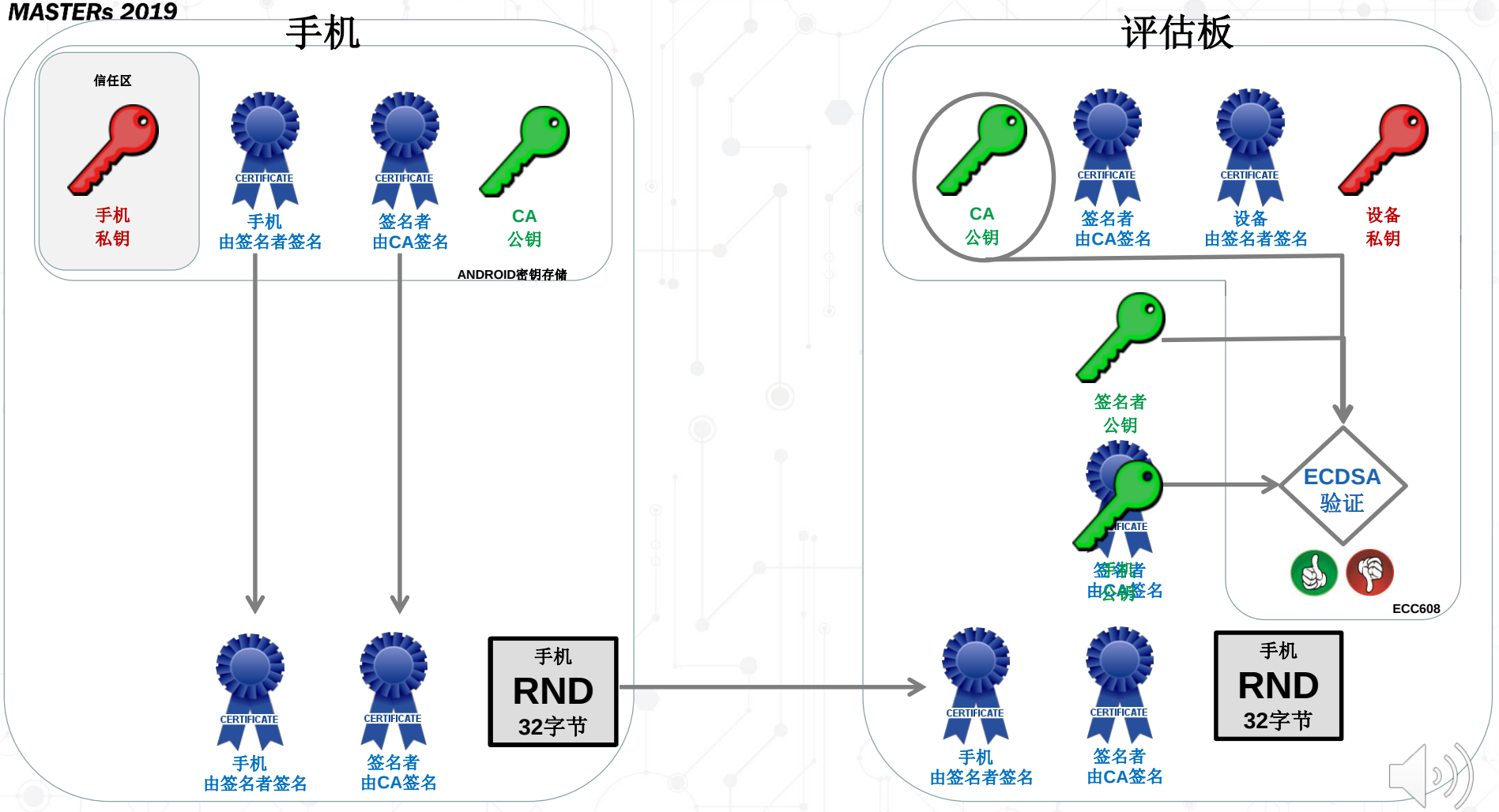
- 验证成功后，智能设备将在RAM中生成一个临时密钥对，并将派生的公钥与此公钥和先前获得的临时值的组合签名一起发送。此签名使用自己的设备私钥（用于验证的私钥）完成
- MCU将接收公钥（临时）和签名。它将在验证时考虑先前发送到经过身份验证的设备的临时值；如果出现任何问题（验证失败），MCU将向BLE模块发出复位信号以断开连接。
- 验证成功后，MCU将使ECC508在特定槽中生成一个临时密钥对，并将派生的公钥与此公钥和先前获得的临时值的组合签名一起发送。此签名使用自己的设备私钥（用于验证的私钥）完成

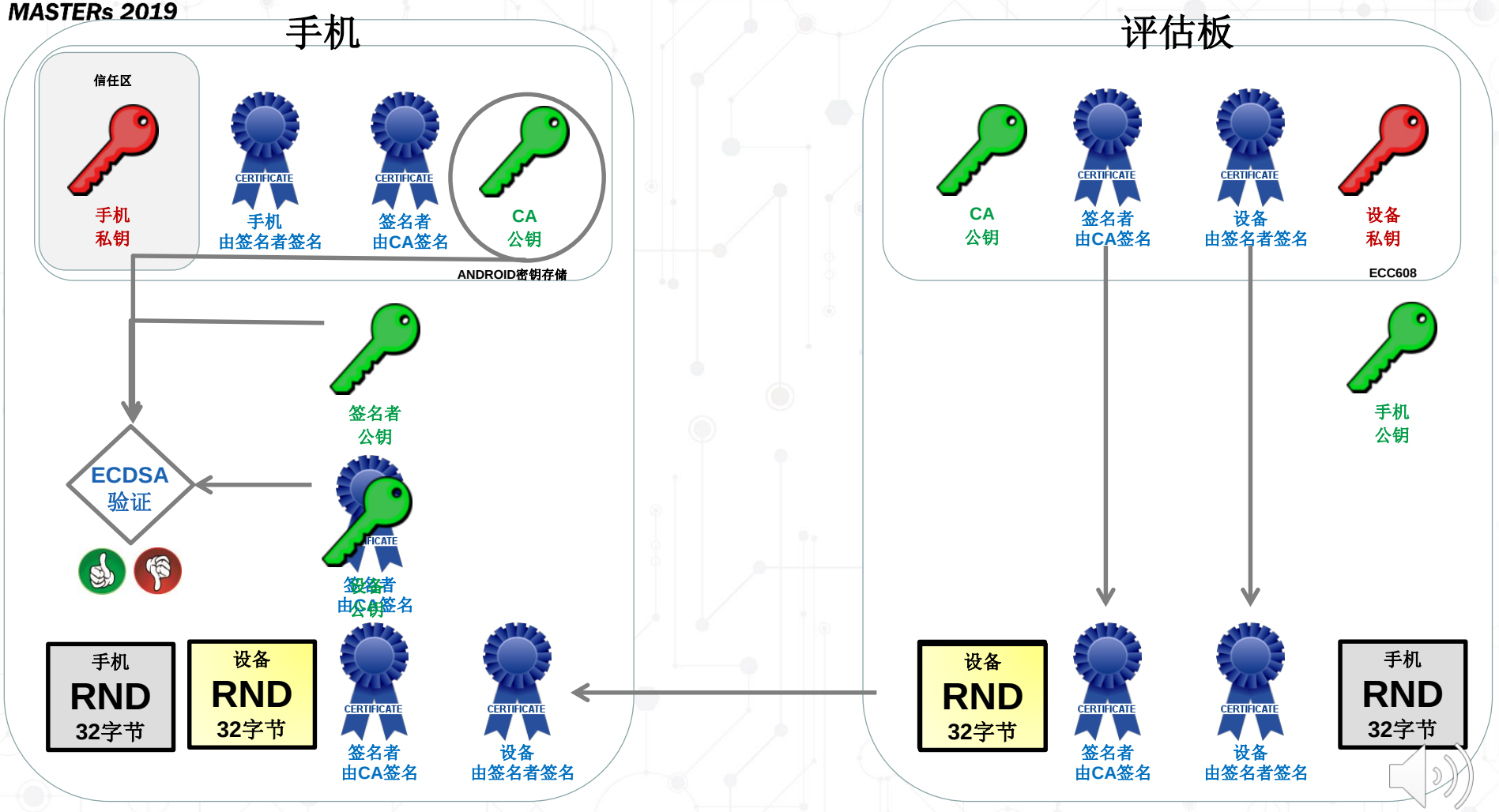
基于BLE的UART安全传输

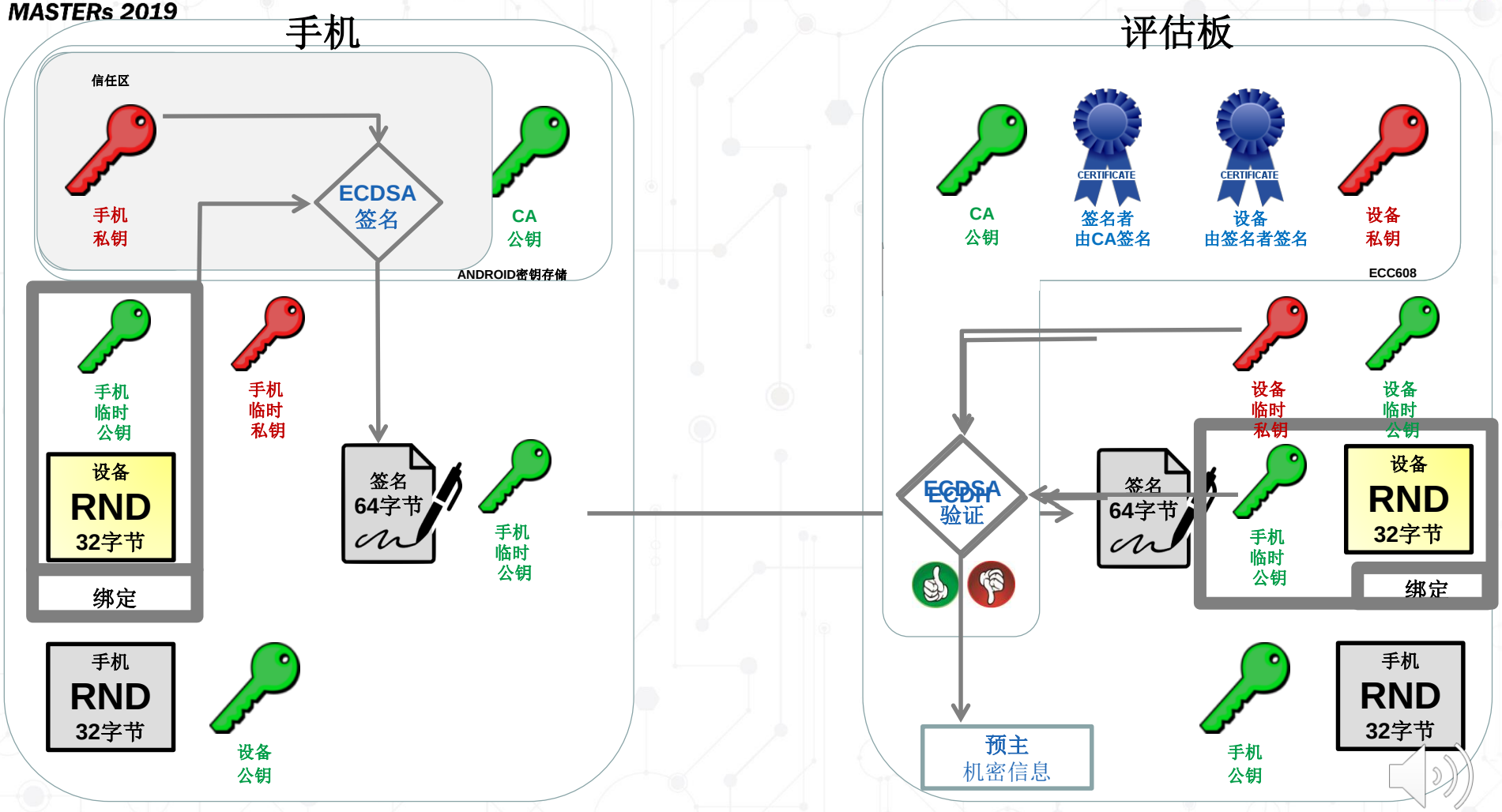
实现基于BLE的UART安全传输

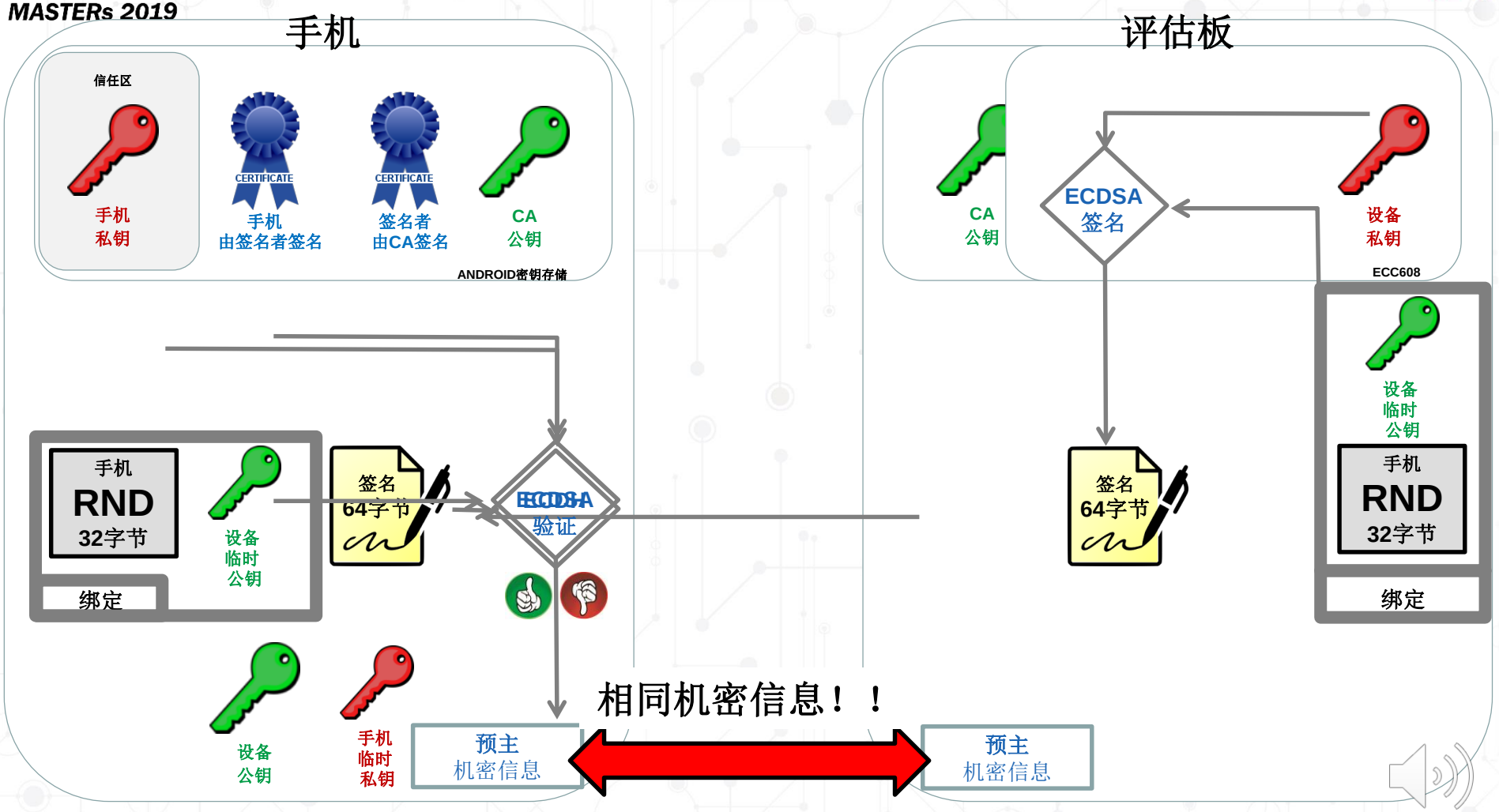
安全握手（续）

- 此时MCU认为握手已成功完成，并将派生出主机密信息，使ECC通过临时公钥及自己的临时私钥执行ECDH，获得32字节的主信息，此信息的具体使用方式如下：前16字节作为AES128的密钥，后16字节作为AES128的初始向量
- 智能设备将接收公钥（临时）和签名。它将在验证时考虑先前发送到经过身份验证的设备的临时值；如果出现任何问题（验证失败），智能设备将断开连接。
- 验证成功后，智能设备将认为握手已成功完成，并将派生出主机密信息，从而通过临时公钥及自己的临时私钥执行ECDH，获得32字节的主信息，此信息的具体使用方式如下：前16字节作为AES128的密钥，后16字节作为AES128的初始向量


手机
信任区
手机
私钥
CERTIFICATE
手机
由签名者签名
CERTIFICATE
签名者
由CA签名
CA
公钥
ANDROID密钥存储
CERTIFICATE
手机
由签名者签名
CERTIFICATE
签名者
由CA签名
手机
RND
32字节
评估板
CA
公钥
CERTIFICATE
签名者
由CA签名
CERTIFICATE
设备
由签名者签名
设备
私钥
签名者
公钥
ECDSA
验证
ECC608
CERTIFICATE
手机
由签名者签名
CERTIFICATE
签名者
由CA签名
手机
RND
32字节


手机
评估板
信任区
手机
私钥
CERTIFICATE
手机
由签名者签名
CERTIFICATE
签名者
由CA签名
CA
公钥
ANDROID密钥存储
CA
公钥
CERTIFICATE
签名者
由CA签名
CERTIFICATE
设备
由签名者签名
设备
私钥
ECC608
签名者
公钥
手机
公钥
ECDSA
验证
设备
由CA签名
手机
RND
32字节
设备
RND
32字节
CERTIFICATE
签名者
由CA签名
CERTIFICATE
设备
由签名者签名
设备
RND
32字节
CERTIFICATE
签名者
由CA签名
CERTIFICATE
设备
由签名者签名
手机
RND
32字节




手机
评估板
信任区
手机
私钥
CERTIFICATE
手机
由签名者签名
CERTIFICATE
签名者
由CA签名
CA
公钥
ANDROID密钥存储
CA
公钥
ECDSA
签名
设备
私钥
ECC608
设备
临时
公钥
手机
RND
32字节
绑定
设备
临时
公钥
签名
64字节
ECDSA
验证
签名
64字节
手机
RND
32字节
绑定
设备
公钥
手机
临时
私钥
预主
机密信息
相同机密信息！！
预主
机密信息

基于BLE的UART安全传输

实现基于BLE的UART安全传输

它似乎很复杂，但实际上全部过程只需转换为4条报文，如下所示：

4条报文总共交换960个字节

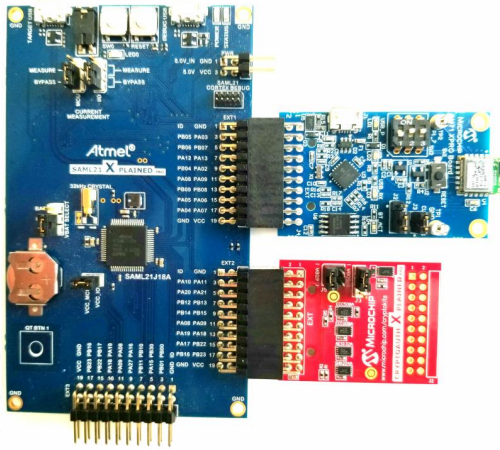
+

两端的所有计算

=

1.2-1.3秒

（比正常的配对过程快）



加密–解密传输

- MCU和智能设备现已准备就绪，可生成相同的会话主机密信息，以设置将加密和解密所有后续通信的AES128引擎

基于BLE的UART安全传输

实现基于BLE的UART安全传输

配置设备：

ECCX08可由Microchip根据您的说明进行配置。
请与我们的SPG市场营销代表之一共同审核这种可能性

对于生产中的智能设备的配置，请与您首选的IoT云提供商（AWS和Google等）共同完成审核

基于BLE的UART安全传输

实现基于BLE的UART安全传输

智能设备配置示例：

- 使用首选的验证方法访问云端上的网页。
- 提供一次性令牌（页面上的二维码或任何其他方法）（您最终也可以通过电子邮件发送）
- 使用应用程序端的令牌向云端发送您使用令牌签名的智能设备证书MAC（如果签名不匹配，将丢弃192字节的报文）
- 收回使用相同令牌签名的证书链MAC
- 关闭连接并使令牌无效

（其中一个示例包含在提供的JAVA应用程序中）

课程安排

- 基本概念
- 基于BLE的UART安全传输
  - 安全入门介绍
  - 蓝牙LE安全概述
  - 为什么要使用不同的方法？
  - 实现基于BLE的UART安全传输
  - **ECCX08概述和使用**
- 演示
  - 配置ECC和Android™手机并建立安全连接

基于BLE的UART安全传输
ECCX08概述

- 为密钥提供安全的存储和执行环境
  - 对称（SHA256）
  - 非对称（椭圆曲线）
- 支持NIST P-256曲线
  - 即secp256r1和prime256v1
- 16个槽共计10.5 Kb存储空间
- 高质量的内部RNG
- 支持SHA256、ECDSA、ECDH以及各种KDF和AES算法

基于BLE的UART安全传输
ECCX08概述

SLOT # Assignement	READ/WRITE	SIZE(bytes)
0 - Private Key of DEVICE	LOCK	36
1 - Not used	OPEN	36
2 - Not used	OPEN	36
3 - Not used	OPEN	36
4 - Not used	OPEN	36
5 - Not used	OPEN	36
6 - Not used	OPEN	36
7 - Not used	OPEN	36
8 - Not used	OPEN	416
9 - IssuerID + Signature of DEVICE	OPEN	72
A - Public Key of SIGNER	OPEN	72
B - SN + Signature of SIGNER	OPEN	72
C - Issuer of SIGNER + Public Key of CA	OPEN	72
D - Not used	OPEN	72
E - Not used	OPEN	72
F - Not used	OPEN	72

演示：
配置ECC和Android™手机并建立安全连接

演示总结

- 我们使用智能手机和**Java**应用程序通过蓝牙®以一种安全的方式与演示板交换字节
- 我们观察了如何创建证书并对证书签名以及如何配置我们的评估板和智能手机

总结

今天介绍了：

- 如何使用智能手机库创建简单的应用程序以基于**BLE**通过类似**UART**的通道进行通信
- 如何使用包含安全元件（**ECCX08**）的安全层来使用现代技术对**BLE**通信进行验证和加密

更多资源

- **Microchip安全：加密入门**
  https://www.youtube.com/watch?v=TdexOLD33bs
- **https://developer.android.com**
- **https://developer.apple.com**
- **https://microchipdeveloper.com/ble:bm70-app-example-transparent-uart-auto-pattern-test-too**
- **https://asn1.io/asn1playground/**
- **https://www.digikey.com/eewiki/display/Wireless/A+Basic+Introduction+to+BLE+Security**
- **https://www.design-reuse.com/articles/39779/security-considerations-for-bluetooth-smart-devices.html**

本课程使用的开发工具

- Microchip MUOB Java应用程序v 1.0
- ATSAML21-XPRO-B SAML21 Xplained PRO
- BM71 Xplained PRO
- ATCRYPTOAUTH-XPRO-B ECC608评估板

法律声明

软件：

Microchip软件仅允许用于Microchip产品。 此外，Microchip软件的使用受软件附带的版权声明、免责声明以及任何授权许可条款的限制，无论这些内容是在安装各个程序时阐明还是在头文件或文本文件中公告。

尽管有上述限制，但Microchip和第三方提供的软件的某些组件仍可能被“开源”软件许可覆盖，其中包括要求分发者提供软件源代码的许可。 在开源软件许可要求的范围内，许可条款将起主导作用。

注意事项和免责声明：

这些材料和随附信息（例如，包括任何软件以及对第三方公司和第三方网站的引用）仅供参考，并且按“现状”提供。 Microchip对第三方公司做出的声明或第三方可能提供的材料或信息不承担任何责任。

MICROCHIP不承担任何形式的保证，无论是明示的、暗示的或法定的，包括有关无侵权性、适销性和特定用途的暗示保证。 在任何情况下，对于与MICROCHIP或其他第三方提供的材料或随附信息有关的任何直接或间接的、特殊的、惩罚性的、偶然的或间接的损失、损害或任何类型的开销，MICROCHIP概不承担任何责任，即使MICROCHIP已被告知可能发生损害或损害可以预见。请注意，使用此处所述的知识产权时可能需要第三方许可。

商标：

Microchip的名称和徽标组合、Microchip徽标、Adaptec、AnyRate、AVR、AVR徽标、AVR Freaks、BesTime、BitCloud、chipKIT、chipKIT徽标、CryptoMemory、CryptoRF、dsPIC、FlashFlex、flexPWR、HELDO、IGLOO、JukeBlox、KeeLoq、Kleer、LANCheck、LinkMD、maXStylus、maXTouch、MediaLB、megaAVR、Microsemi、Microsemi徽标、MOST、MOST徽标、MPLAB、OptoLyzer、PackeTime、PIC、picoPower、PICSTART、PIC32徽标、PolarFire、Prochip Designer、QTouch、SAM-BA、SenGenuity、SpyNIC、SST、SST徽标、SuperFlash、Symmetricom、SyncServer、Tachyon、TempTrackr、TimeSource、tinyAVR、UNI/O、Vectron及XMEGA均为Microchip Technology Inc.在美国和其他国家或地区的注册商标。

APT、ClockWorks、The Embedded Control Solutions Company、EtherSynch、FlashTec、Hyper Speed Control、HyperLight Load、IntelliMOS、Libero、motorBench、mTouch、Powermite 3、PrecisionEdge、ProASIC、ProASIC Plus、ProASIC Plus徽标、Quiet-Wire、SmartFusion、SyncWorld、Temux、TimeCesium、TimeHub、TimePictra、TimeProvider、Vite、WinPath和ZL均为Microchip Technology Inc.在美国的注册商标。

Adjacent Key Suppression、AKS、Analog-for-the-Digital Age、Any Capacitor、AnyIn、AnyOut、BlueSky、BodyCom、CodeGuard、CryptoAuthentication、CryptoAutomotive、CryptoCompanion、CryptoController、dsPICDEM、dsPICDEM.net、Dynamic Average Matching、DAM、ECAN、EtherGREEN、In Circuit Serial Programming、ICSP、INICnet、Inter-Chip Connectivity、JitterBlocker、KleerNet、KleerNet徽标、memBrain、Mindi、MiWi、MPASM、MPF、MPLAB Certified徽标、MPLIB、MPLINK、MultiTRAK、NetDetach、Omniscient Code Generation、PICDEM、PICDEM.net、PICkit、PICtail、PowerSmart、PureSilicon、QMatrix、REAL ICE、Ripple Blocker、SAM-ICE、Serial Quad I/O、SMART-I.S.、SQI、SuperSwitcher、SuperSwitcher II、Total Endurance、TSHARC、USBCheck、VariSense、ViewSpan、WiperLock、Wireless DNA和ZENA均为Microchip Technology Inc.在美国和其他国家或地区的商标。

SQTP为Microchip Technology Inc.在美国的服务标记。

Adaptec徽标、Frequency on Demand、Silicon Storage Technology和Symmcom为Microchip Technology Inc.在除美国外的国家或地区的注册商标。

GestIC为Microchip Technology Inc.的子公司Microchip Technology Germany II GmbH & Co. & KG在除美国外的国家或地区的注册商标。

在此提及的所有其他商标均为各持有公司所有。