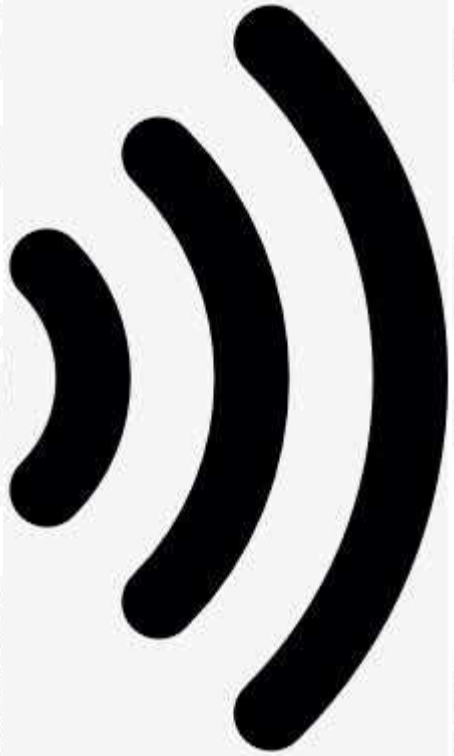
BLE通信给IoT设备带来的安全隐患

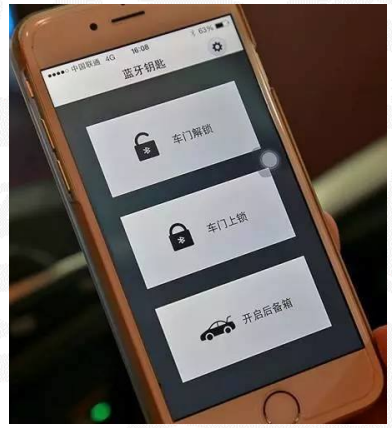
Lightτ

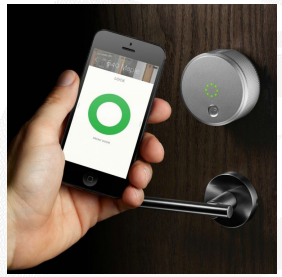
自我介绍

- 未来安全-胖猴实验室
- IoT 安全研究员
- 演讲者
- guotao@xfuturesec.com

ROARTALK 嘶吼







目录

CONTENTS

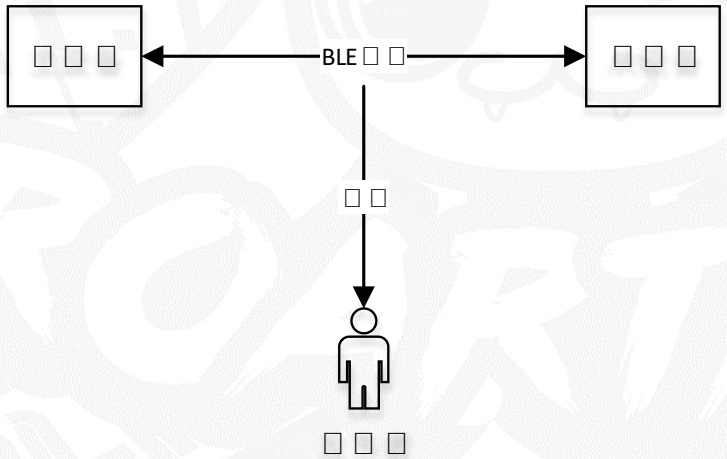
1

BLE协议的安全机制

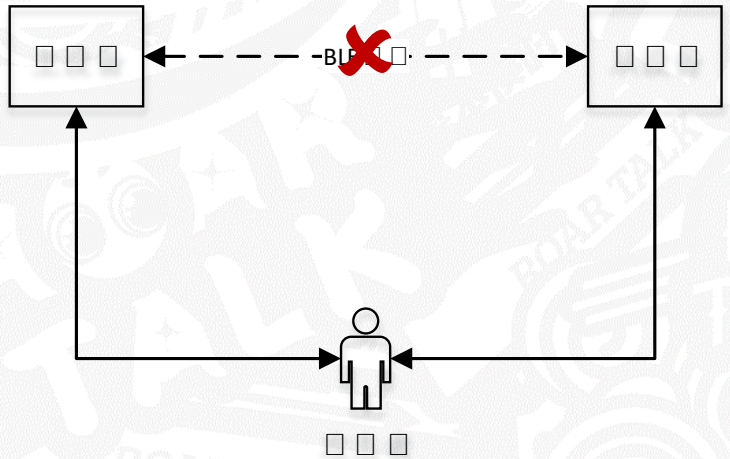
BLE协议的安全机制

BLE作为一种无线通信，面临的最主要的攻击手段有两种：

（1）通信数据嗅探



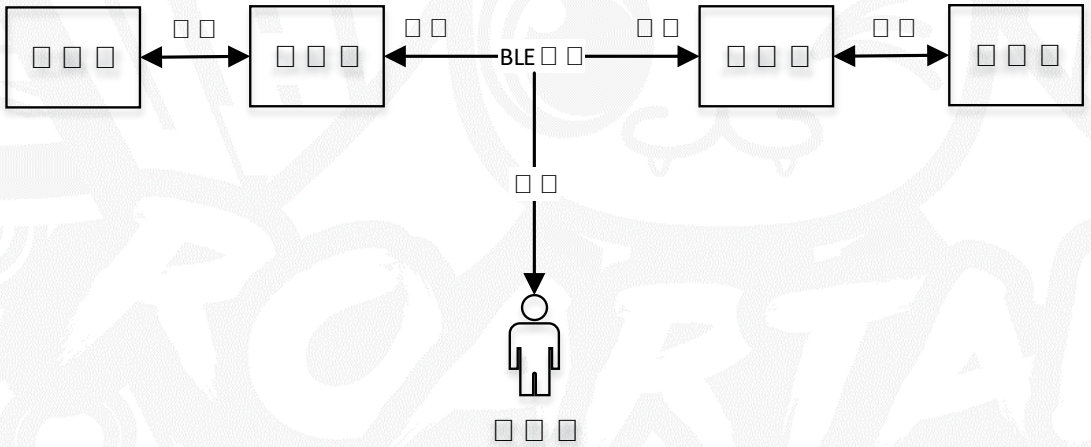
（2）中间人攻击



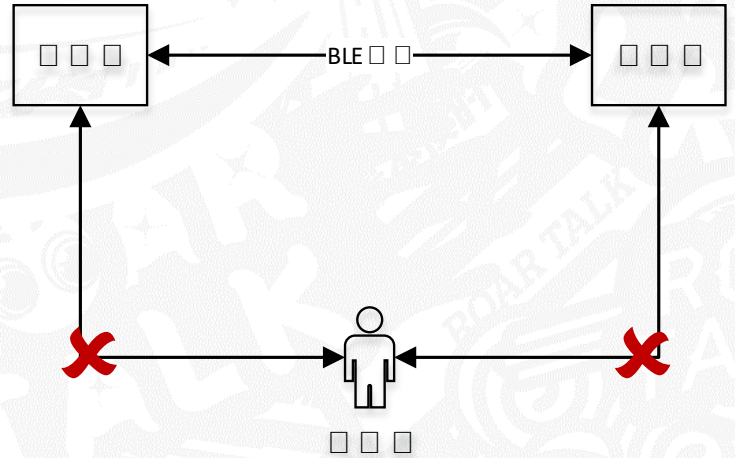
BLE协议的安全机制

针对这两种攻击手段，至少需要有两方面的安全机制：

（1）通信加密



（2）身份认证



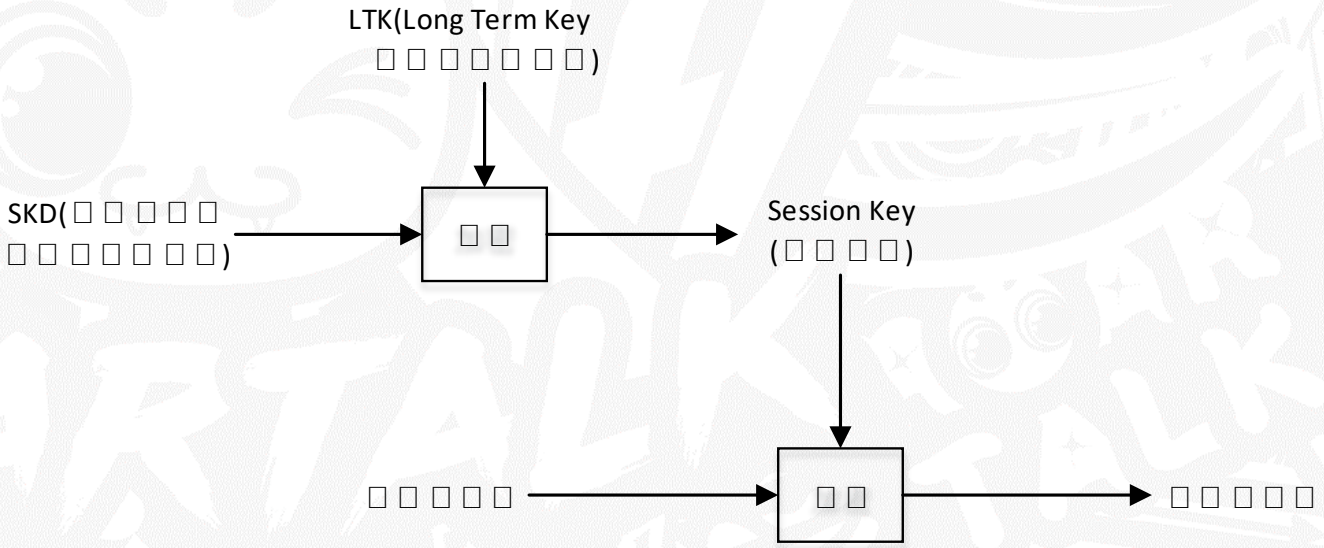
两种机制相辅相成，缺一不可

BLE协议的安全机制

对应的，BLE协议规定了应对的安全机制：

（1）LE Encryption

启用LE Encryption机制后，主从设备会通过LTK和SKD生成会话密钥Session Key，从链路层对数据包进行加密

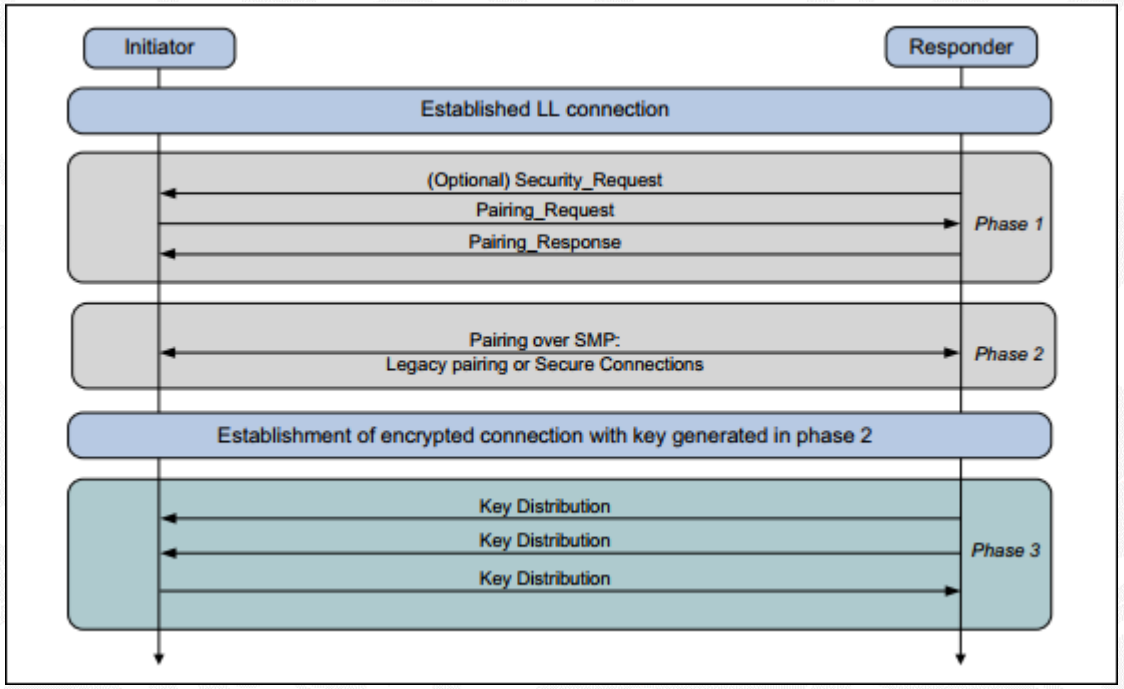


BLE协议的安全机制

对应的，BLE协议规定了应对的安全机制：

（2）Security Manager

Security Manager的作用主要是身份认证和密钥分发，启用该安全机制后，设备建立BLE连接前会先进行配对(pairing)



在第一阶段，主从设备之间将发起配对请求及配对响应通信，互相交换信息，决定第二阶段选择那种配对方式

第二阶段阶段是配对(paring)的核心，SMP即Security Manager Protocol，身份认证和密钥协商主要在这一阶段完成

第三阶段会利用第二阶段生成的密钥进行其他密钥的分发

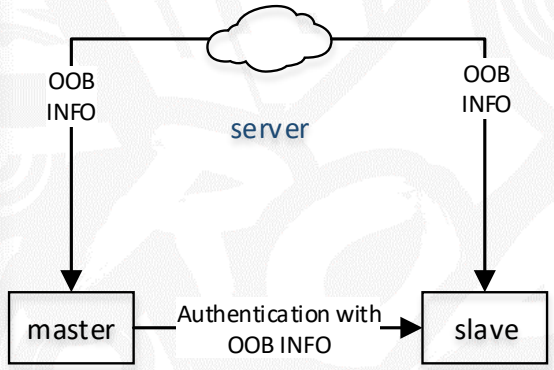
BLE协议的安全机制

对应的，BLE协议规定了应对的安全机制：

（2）Security Manager

Security Manager的身份认证主要有三种方式：

OOB(Out of band)



Passkey Entry或Numeric Comparison

Just Works

不进行认证

BLE协议的安全机制

对应的，BLE协议规定了应对的安全机制：

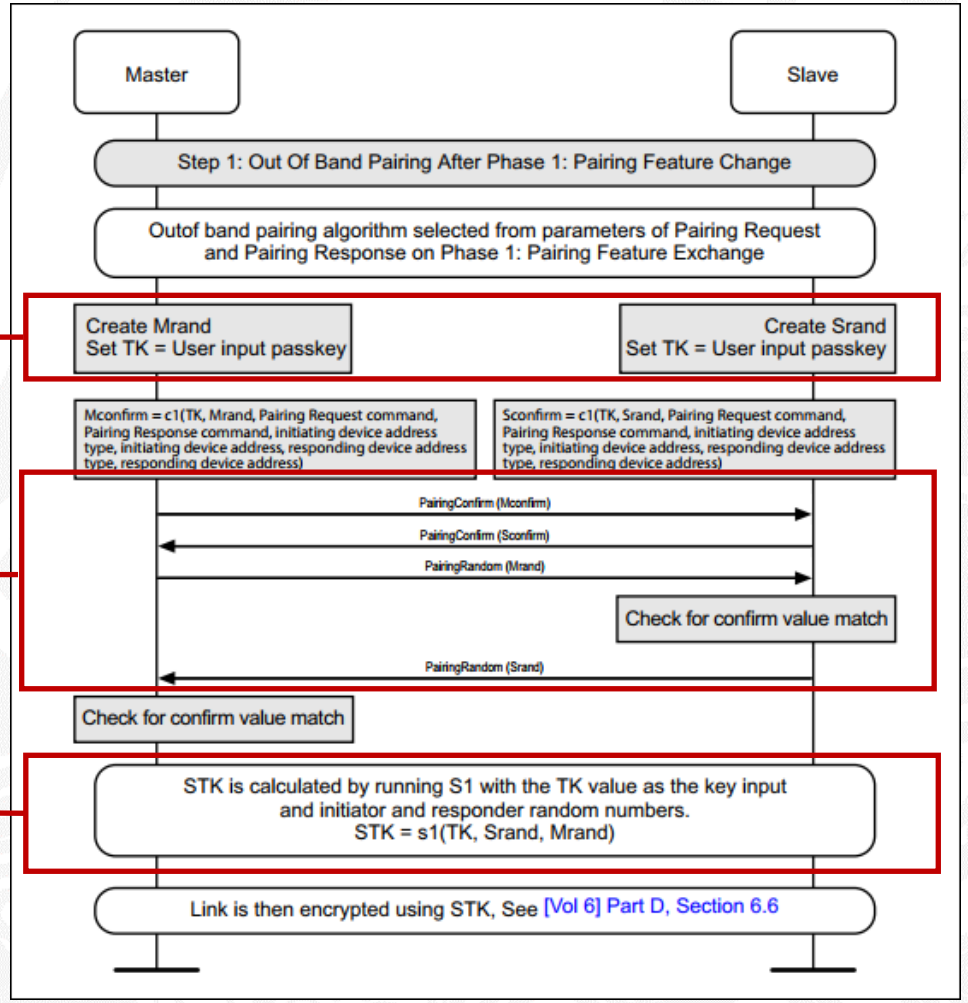
（2）Security Manager

密码分发会在身份认证的基础上进行，图例为OOB身份认证方式

TK是在身份认证的过程中得到的，根据在认证方式，TK可以是通过OOB得到OOB INFO，也可以是通过Passkey Entry得到，也可以是0（Just Work）

主从双方分别产生随机数Mrand和Srand，并通过蓝牙传输给对方

根据TK和Mrand、Srand，协商STK密钥



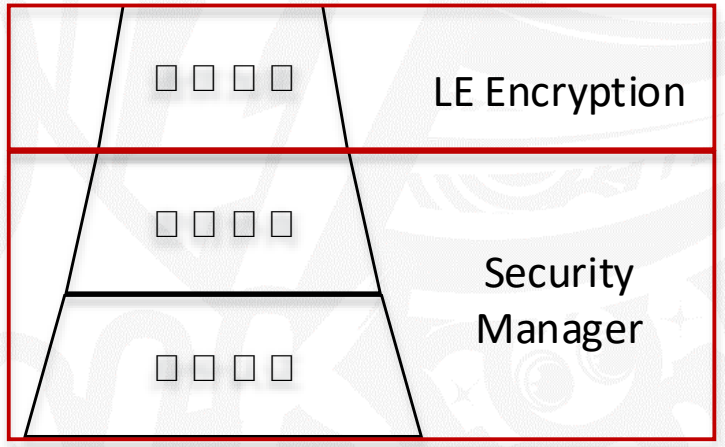
BLE协议的安全机制

总结下来，BLE协议中的安全机制由以下三部分组成

□□□□□□□□□□□□□□□□

□□TK□□STK/LTK □□□STK/LTK□□□□□□

□□□□□□TK



可以看到，身份认证是这两个安全机制正常工作的基础

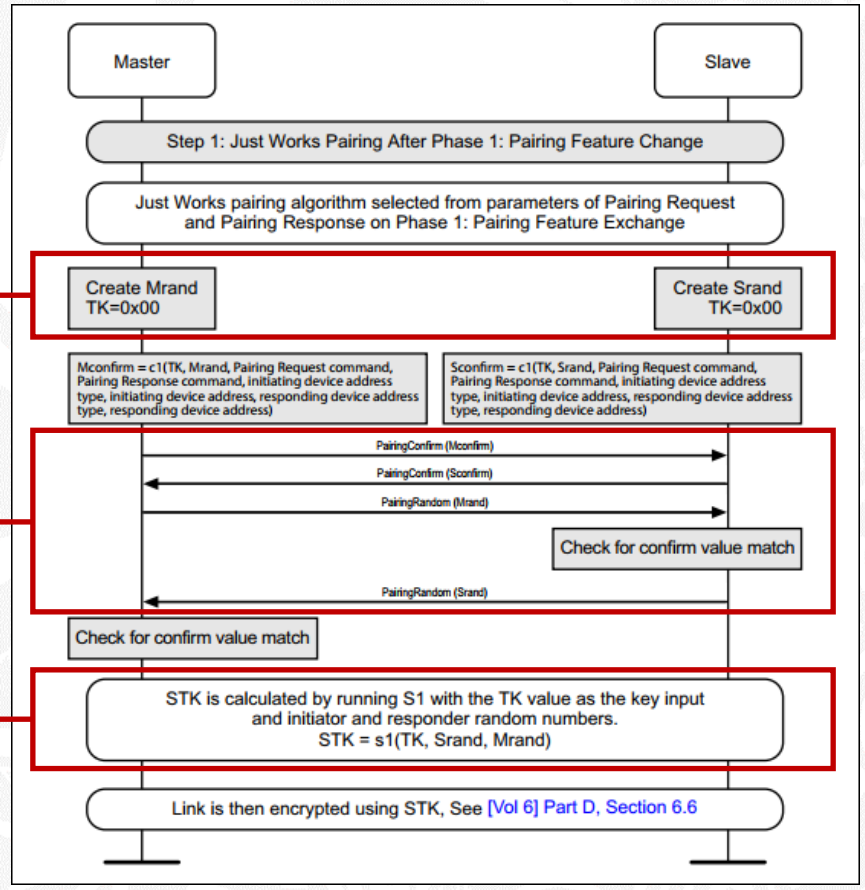
BLE协议的安全机制

尽管BLE协议规定了完备的安全机制，但是由于一些原因，这些安全机制并没有得到有效使用，例如受设备功能限制，无法使用完整的安全机制

某些IoT设备只能选择Just Works的身份认证方式，此时TK=0，且无法判定主从设备的身份

主从双方分别产生随机数Mrand和Srand，并通过蓝牙传输给对方，这两个随机数可以被嗅探到

此时攻击者可以根据嗅探到的Mrand和Srand计算STK，进而解密后续通信内容



BLE协议的安全机制

目前绝大多数厂商都没有采用完整的Security Manager机制，而是采用了其他方法：

1）不对通信进行任何保护措施

2）在应用层仅进行身份认证或数据加密

3）在应用层实现完整的身份认证和数据加密

接下来我们会对这些情况分别进行分析

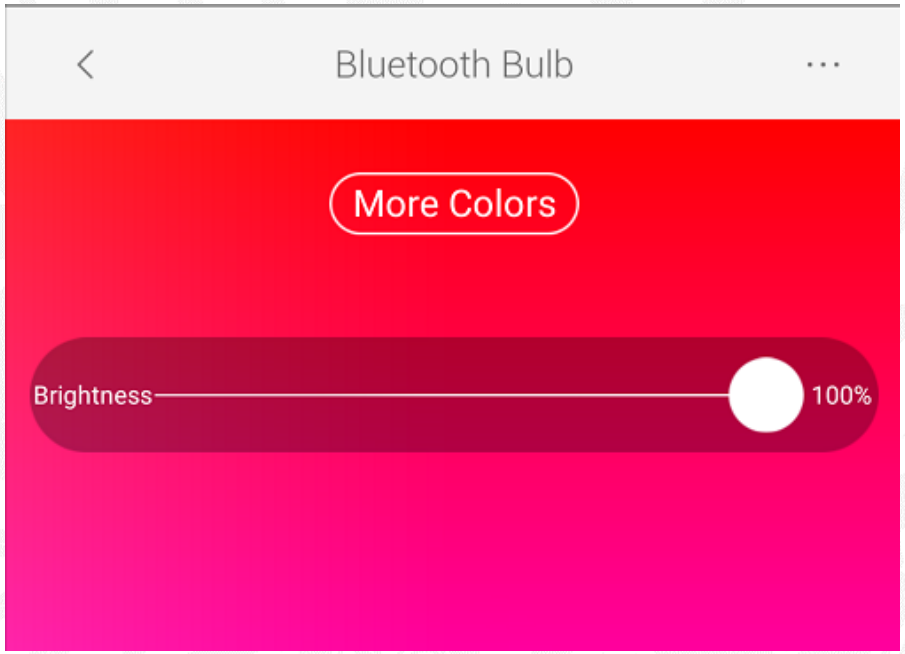
2

典型示例

典型示例

1、不对通信进行任何保护措施

以某款智能灯泡为例

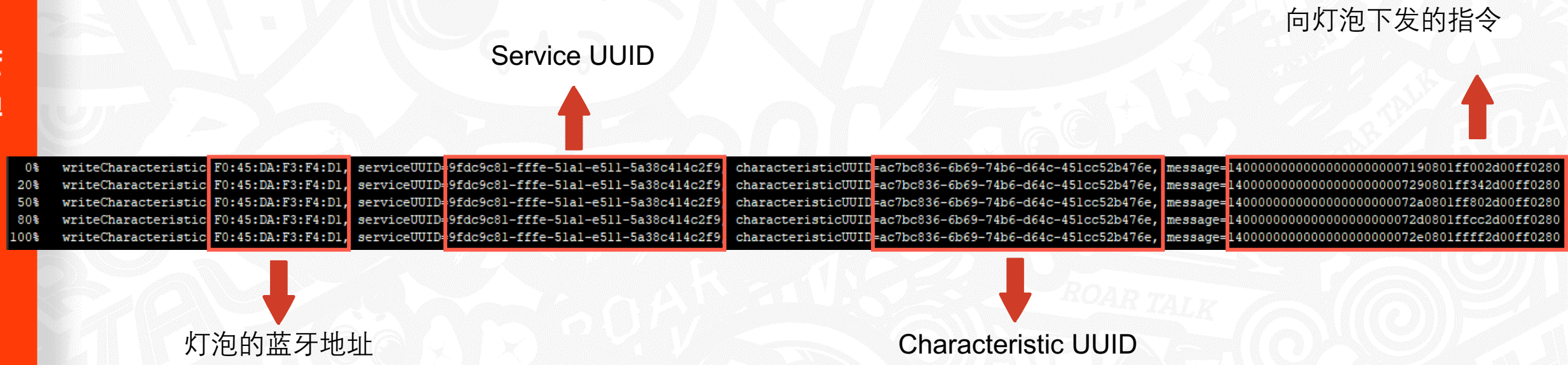


通过APP可以控制
灯泡的亮度和颜色

典型示例

1、不对通信进行任何保护措施

通过对数据包的嗅探，可以得知APP如何对灯泡进行控制



典型示例

1、不对通信进行任何保护措施

得到通信明文后，可以直接通过nRF Connect连接灯泡并下发指令

```
  0%   writeCharacteristic F0:45:DA:F3:F4:D1, serviceUUID=9fdc9c81-fffe-51a1-e511-5a38c414c2f9 characteristicUUID=ac7bc836-6b69-74b6-d64c-451cc52b476e, message=1400000000000000000000007190801ff002d00ff0280
 20%   writeCharacteristic F0:45:DA:F3:F4:D1, serviceUUID=9fdc9c81-fffe-51a1-e511-5a38c414c2f9 characteristicUUID=ac7bc836-6b69-74b6-d64c-451cc52b476e, message=1400000000000000000000007290801ff342d00ff0280
 50%   writeCharacteristic F0:45:DA:F3:F4:D1, serviceUUID=9fdc9c81-fffe-51a1-e511-5a38c414c2f9 characteristicUUID=ac7bc836-6b69-74b6-d64c-451cc52b476e, message=1400000000000000000000072a0801ff802d00ff0280
 80%   writeCharacteristic F0:45:DA:F3:F4:D1, serviceUUID=9fdc9c81-fffe-51a1-e511-5a38c414c2f9 characteristicUUID=ac7bc836-6b69-74b6-d64c-451cc52b476e, message=1400000000000000000000072d0801ffcc2d00ff0280
100%   writeCharacteristic F0:45:DA:F3:F4:D1, serviceUUID=9fdc9c81-fffe-51a1-e511-5a38c414c2f9 characteristicUUID=ac7bc836-6b69-74b6-d64c-451cc52b476e, message=1400000000000000000000072e0801ffff2d00ff0280
```

典型示例

1、不对通信进行任何保护措施

视频展示

典型示例

1、不对通信进行任何保护措施

安全性分析：

1）通过嗅探BLE通信即可获取明文数据

2）任何设备使用嗅探到的明文数据可以直接控制灯泡

典型示例

2、在应用层仅进行身份认证或数据加密

以某款智能门锁为例，门锁需要对手机进行身份认证

典型示例

2、在应用层仅进行身份认证或数据加密

通过嗅探可以获取到身份认证数据包的明文

认证请求

P.nbr.	Time (us)	Channel	Access Address	Data Type	Data Header					L2CAP Header		ATT_Handle_Value_Notify			CRC	RSSI (dBm)	FCS
	+231				LLID	NESN	SN	MD	PDU-Length	L2CAP-Length	ChanId	Opcode	AttHandle	AttValue			
267	=1776457	0x08	0x6C552D61	L2CAP-S	2	1	0	0	24	0x0014	0x0004	0x1B	0x0017	72 B0 B0 0C 35 98 63 FB 40 D8 98 70 4B 23 D8 FB 70	0x0B3FF4	0	OK

认证响应

P.nbr.	Time (us)	Channel	Access Address	Data Type	Data Header					L2CAP Header		ATT_Write_Req			CRC	RSSI (dBm)	FCS
	+44269				LLID	NESN	SN	MD	PDU-Length	L2CAP-Length	ChanId	Opcode	AttHandle	AttValue			
268	=1820726	0x0D	0x6C552D61	L2CAP-S	2	1	1	0	24	0x0014	0x0004	0x12	0x001E	72 A1 01 0C C9 B8 B3 63 24 28 10 D8 9B 43 6C 73 7C	0x88C8F3	0	OK

典型示例

2、在应用层仅进行身份认证或数据加密

通过对APP的逆向，可以得到身份认证算法

```
protected void handleLockStatus(FrameModel model, String data) {
    switch (AnonymousClass8.$SwitchMap$com$irevo$nprotocol$Events[model.event.ordinal()]) {
        case 1:
            ILog.d("_72_AUTHENTICATION_EVENT " + data + " : " + getDateDiff());
            send72Response(encodeCounter(model.data));
            return;
        case 2:
```

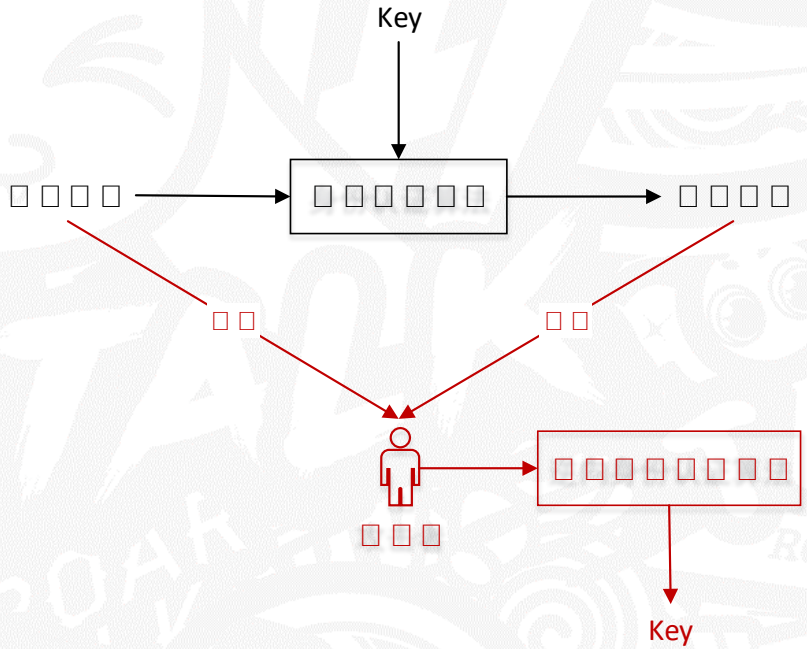
```
String keyString = this.keyModel.productInfo;
int i = 0 + 2;
HexInt hexInt = new HexInt(new HexInt(keyString.substring(0, i)).intVal);
if (hexInt.hexVal.length() > 2) {
    key1 = new HexInt(hexInt.hexVal.substring(1));
}
int start = i + 2;
hexInt = new HexInt(new HexInt(keyString.substring(i, start)).intVal);
if (hexInt.hexVal.length() > 2) {
    key2 = new HexInt(hexInt.hexVal.substring(1));
}
```

```
HexInt c1r1 = new HexInt(new HexInt(counterData.stringVal.substring(0, i)).intVal + key1.intVal);
if (c1r1.hexVal.length() > 2) {
    c1r1 = new HexInt(c1r1.hexVal.substring(1));
}
start = i + 2;
HexInt c1r2 = new HexInt(new HexInt(counterData.stringVal.substring(i, start)).intVal + key2.intVal);
if (c1r2.hexVal.length() > 2) {
    c1r2 = new HexInt(c1r2.hexVal.substring(1));
}
i = start + 2;
HexInt c1r3 = new HexInt(new HexInt(counterData.stringVal.substring(start, i)).intVal + key3.intVal);
if (c1r3.hexVal.length() > 2) {
    c1r3 = new HexInt(c1r3.hexVal.substring(1));
}
start = i + 2;
HexInt c1r4 = new HexInt(new HexInt(counterData.stringVal.substring(i, start)).intVal + key4.intVal);
if (c1r4.hexVal.length() > 2) {
    c1r4 = new HexInt(c1r4.hexVal.substring(1));
}
i = start + 2;
HexInt c2r1 = new HexInt(new HexInt(counterData.stringVal.substring(start, i)).intVal + key5.intVal);
if (c2r1.hexVal.length() > 2) {
    c2r1 = new HexInt(c2r1.hexVal.substring(1));
}
start = i + 2;
HexInt c2r2 = new HexInt(new HexInt(counterData.stringVal.substring(i, start)).intVal + key3.intVal);
if (c2r2.hexVal.length() > 2) {
    c2r2 = new HexInt(c2r2.hexVal.substring(1));
}
```

典型示例

2、在应用层仅进行身份认证或数据加密

如果身份认证算法强度不够，可以通过逆向运算破解身份认证



典型示例

2、在应用层仅进行身份认证或数据加密

视频展示

典型示例

2、在应用层仅进行身份认证或数据加密

安全性分析：

1）身份认证的凭证或数据加密的密钥仍然需要通过BLE通信传输

2）如果身份认证算法或加密强度不够，攻击者仍有机会获取身份认证的凭证或数据加密的密钥

典型示例

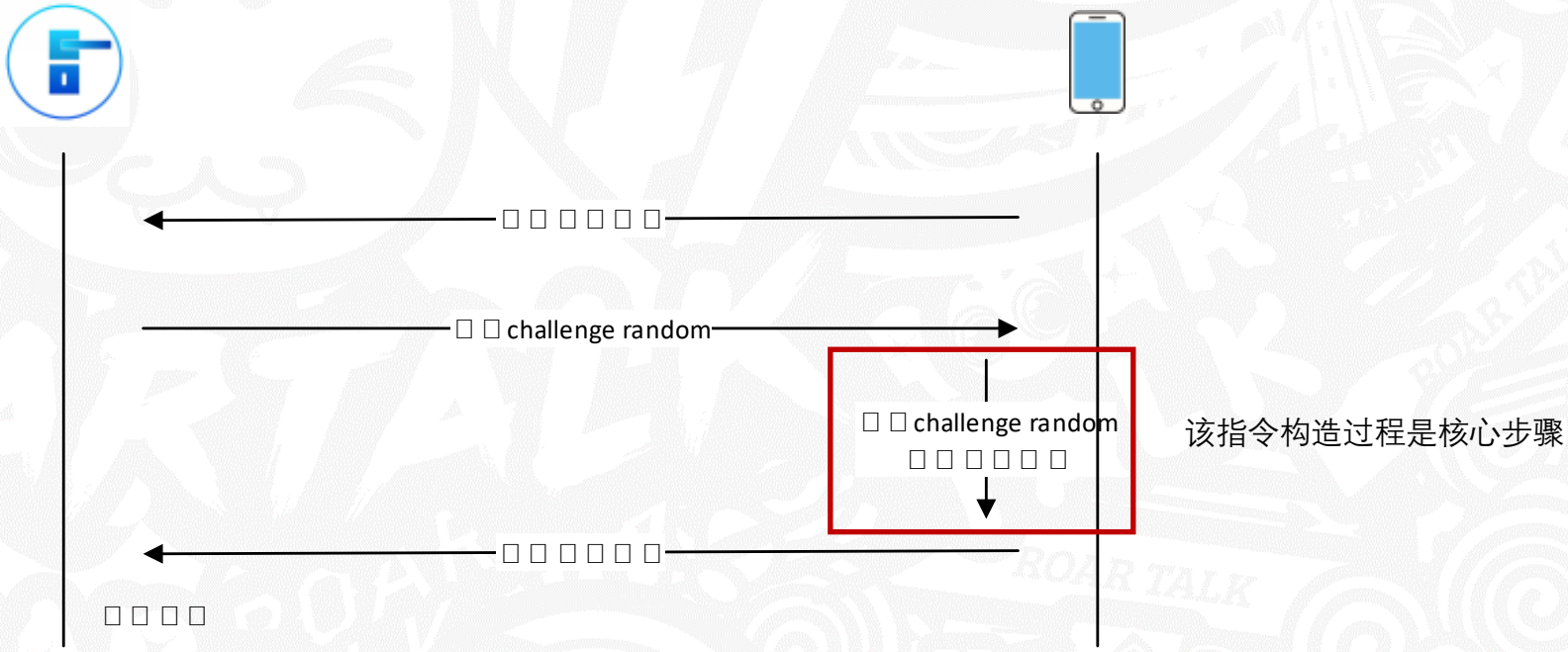
3、在应用层实现完整的身份认证和数据加密

依然以某款智能门锁为例

典型示例

3、在应用层实现完整的身份认证和数据加密

其开锁指令下发过程为

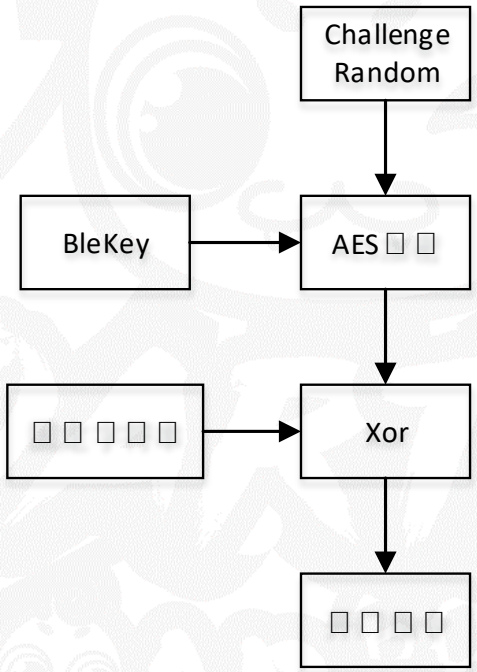


该指令构造过程是核心步骤

典型示例

3、在应用层实现完整的身份认证和数据加密

指令构造过程使用AES加密




```
vate void sendUnlockCommandNewProtocol(BluetoothGatt arg4, BleKeyInfo arg5,
 MyLogger.ddLog(this.TAG).e("sendUnlockCommandNewProtocol blekey:" + arg5.g
 this.mData = BleStack.constructUnlockNewProtocol(arg5, arg6, arg7, arg8);
```

```
constructUnlockNewProtocol(BleKeyInfo arg10, int arg11, int arg12, byte[] arg13) {

|| (TextUtils.isEmpty(arg10.getToken())) || (TextUtils.isEmpty(arg10.getUuid()))) {
```

```
g("111").e("open lock key.getToken():" + arg10.getToken());
g("111").e("open lock key.getId():" + arg10.getId());
endar.getInstance().getTimeInMillis() / 1000;
constructRequest(StackL2.constructL2UnlockNewProtocol(arg10.getId(), BleStack.encryptBlekeyNewProtocol
```

```
blic static byte[] encryptBlekeyNewProtocol(String arg9, String arg10, 
    byte[] v4;
    byte[] v5 = null;
    if(arg9 == null) {
        Log.e("BleStack", "blekey is null");
        v4 = v5;
        return v4;
    }

    if(arg9.length() != 16) {
        Log.e("BleStack", "blekey is invalide, length=" + arg9.length());
        return v5;
    }

    new byte[0];
    new StringBuilder().append(arg11).append(arg12).toString();
    try {
        v4 = AESUtil.encryptEbc(arg9, arg13);
    }
```

只有保证了作为密钥的BleKey的安全性，该加密体系才是安全的

典型示例

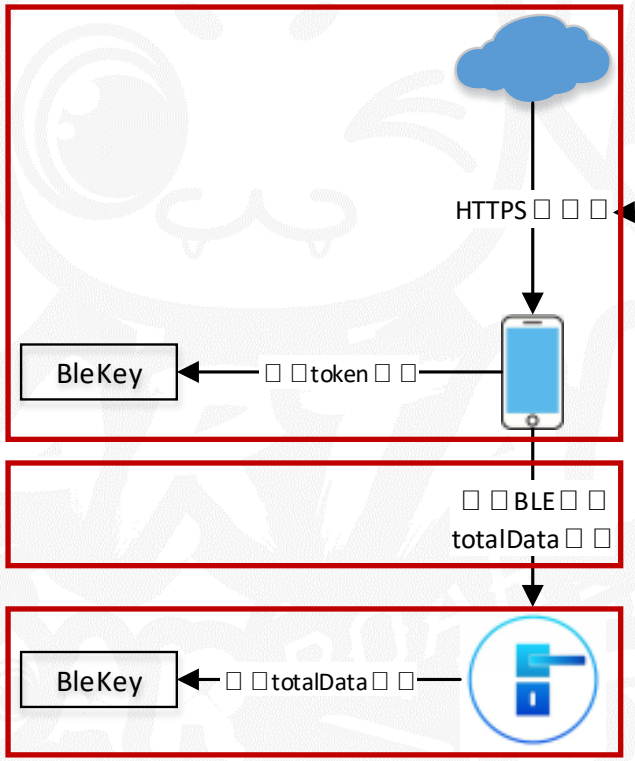
3、在应用层实现完整的身份认证和数据加密

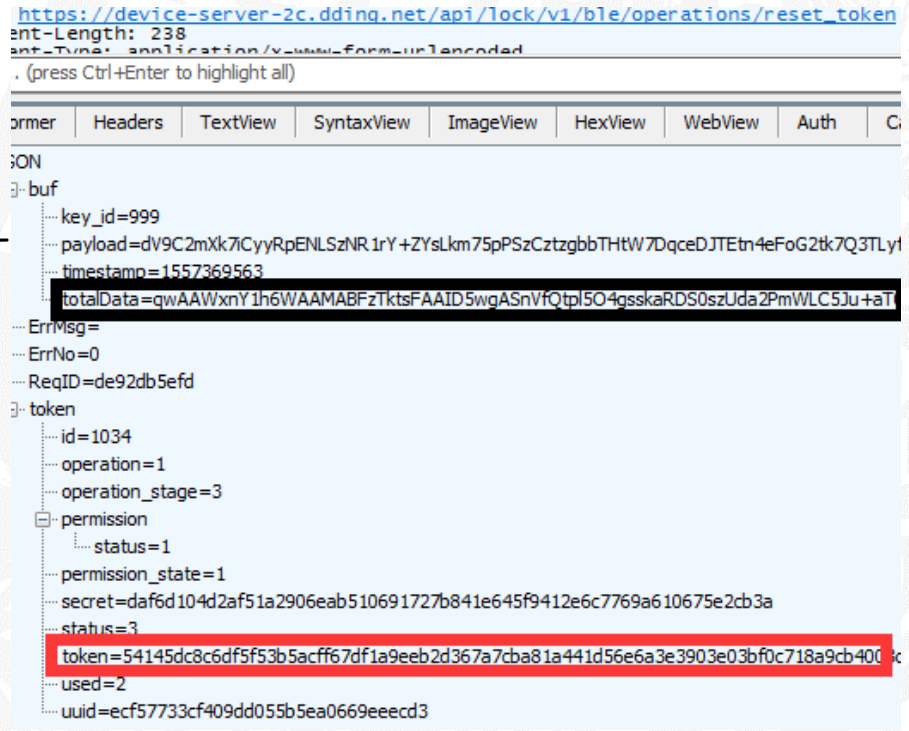
BleKey直接由服务器下发

APP有加壳，无法监听HTTPS通信

BLE通信可以被嗅探到

攻击者能否解密totalData，是BLE通信安全性的关键

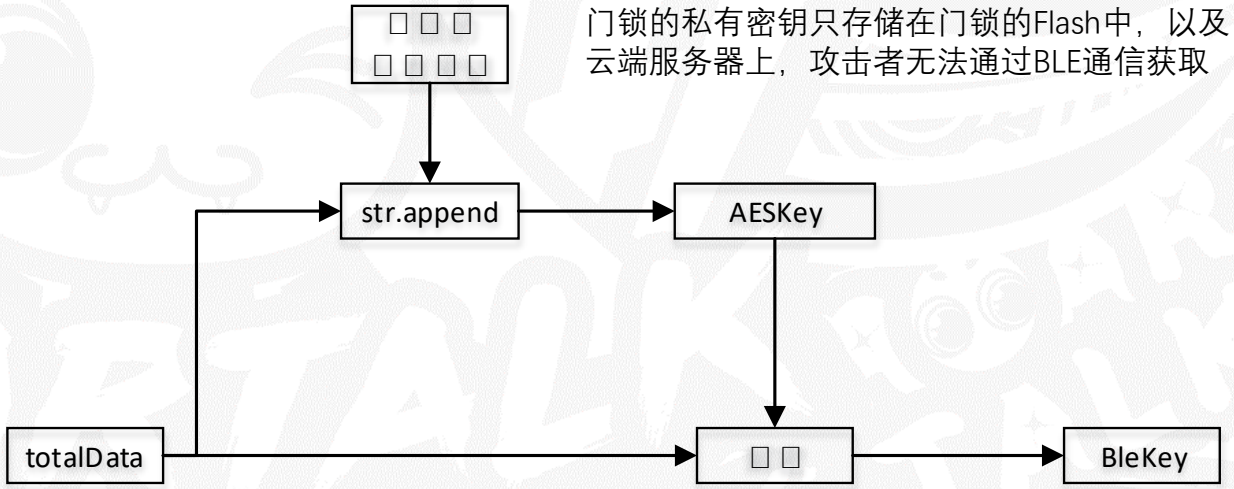




典型示例

3、在应用层实现完整的身份认证和数据加密

门锁端解密totalData字段



只有门锁的合法使用者能够获取正确的totalData
相当于服务器对手机端进行了身份认证

典型示例

3、在应用层实现完整的身份认证和数据加密

安全性分析：

（1）通信内容有AES加密保护

（2）攻击者无法通过嗅探BLE通信获取AES的加密密钥

（3）由服务器完成对手机的身份认证

3

安全建议

安全建议

1、配对时尽可能采用OOB方式

2、在应用层建立完善的安全机制

- 身份认证
- 密钥协商/密钥分发
- 通信加密

THANKS!