AN12292

在 NXP LPC54S0xx 上使用 SRAM PUF 进行安全存储

版本 1.0 — 2018 年 11 月 6 日 应用笔记

目录

1 介绍

LPC54S0xx 是基于 ARM©Cortex-M4 的微控制器系列，用于嵌入式应用，具有丰富的外设，以及非常低的功耗和增强的安全功能。该系列微控制器包括一个基于 SRAM 的物理上不可克隆功能（PUF）控制器，该控制器能够安全地生成唯一的设备指纹和唯一的加密密钥。SRAM PUF 机制紧密集成到 LPC54S0xx 系列中，使来自 PUF 的密钥可以直接由设备的内部 AES-256 加密引擎使用。与其他密钥注入或存储方式相比，唯一且不可复制的密钥具有明显的安全优势。通过将安全性的基础建立在设备唯一性的不可复制密钥的基础上，PUF 密钥可降低 break once repeat everywhere（BORE）攻击时遭受破坏的威胁。

本应用笔记介绍了如何基于 NXP LPC54S0xx 系列 ARM Cortex-M4 的微控制器来开发安全的嵌入式应用，这些应用使用 SRAM PUF 为存储的信息提供非常强大的保护。加密可用于保护来自外部 SPIFI，EMC 和 SPI NOR 闪存设备或通过串行从机接口（UART，SPI，I2C）下载的数据和启动镜像。

1.1 基于 SRAM 的 PUF 密钥派生

由于深亚微米制造工艺的差异，集成电路（IC）中的每个晶体管的物理特性都会略有不同。这些导致了电子特性（如晶体管阈值电压和增益因数）方面的微小但可测量的差异。由于在制造过程中无法完全控制这些过程变化，因此无法复制或克隆这些物理设备属性。

事实证明，由于阈值电压的随机差异，每次给 SRAM 供电时，每个 SRAM 单元都有其自己的倾向状态。该倾向与相邻单元的倾向无关，并且与单元在芯片或晶片上的位置无关。

因此，SRAM 区域会产生一个唯一且随机的由一系列 0 和 1 组成的“图案”（pattern）。这种图案可以被称为 SRAM 指纹，因为它在每个 SRAM 以及每个芯片中都是唯一的。它可以用作 PUF。

SRAM PUF 派生出的密钥不存储在芯片上，仅在需要时才从芯片中提取。这样，它们仅在很短的时间窗口内出现在芯片中。当 SRAM 不通电时，芯片上就没有密钥，这使得该解决方案在应对反向工程和密钥提取时非常安全。

1.2 LPC54S0xx SRAM PUF 功能

SRAM PUF 硬件使用从未初始化的 SRAM 和称为激活码（Activation Code，AC）的错误校正数据得出的设备的数字指纹，构造出 256 位强度的设备唯一性密钥。激活码是在注册过程中生成的，注册过程是在制造商对设备进行配置/个性化过程中进行的。请参阅设备配置。

LPC54S0xx PUF 硬件支持：

1. 基于 PUF 的设备唯一性密钥的生成，存储和重建。
2. 将 PUF 生成的密钥路由到 AES 加密引擎，以在设备启动期间支持外部闪存的加密和身份验证。
3. 支持密钥大小从 64 位到 4096 位用户密钥的安全存储和重建。

每个生成的 PUF 密钥都分配有一个 4bit 索引值以标识其用法。可通过寄存器接口使用索引为非零的密钥。

索引值为 0 的密钥通过专用总线输出到 AES 引擎，这样它们就不能被软件访问。MCU 可以配置为直接从受 PUF 生成索引为 0 密钥保护的外部闪存启动。

PUF 硬件支持使用密钥代码（KC）保护其他应用程序的秘密。密钥代码（KC）是由 PUF 生成的密钥加密的数据包。通过加密，可以将机密信息安全地存储在处理器外部。虽然通常用于安全存储密钥，但是可以使用密钥代码存储任何敏感的应用程序数据。密钥代码数据的大小在 64 位和 4096 位之间。4 位索引值用于标识用于密钥代码的加密和解密的密钥。

存储在外部 NV 存储器中的加密信息无法转移到任何其他设备。加密过程对于特定处理器是唯一的，并且如果将加密的信息移至其他处理器，则无法正确解密。加密过程包括完整性检查，该检查还可以防止在受保护的 MCU 范围之外对信息进行任何修改。

1.3 范围

本应用笔记涵盖了 LPC54S0xx 在 SRAM PUF 中的使用：

- 当存储在外部闪存设备上时，保护设备固件。
- 创建和使用设备唯一性密钥和标识符。
- 保护可能由应用程序使用或提供的密钥信息。
- 使用 SRAM PUF 时确保整体系统安全。

2 使用 SRAM PUF 硬件

在 LPC54S0xx 上使用 SRAM PUF 硬件的处理流程可在产品的整个生命周期中带来好处。SRAM PUF 硬件的使用说明如下：

- OEM 制造期间的设备配置。
- 从加密的闪存中安全启动。
- 派生供应用程序使用的设备唯一性密钥。
- 在设备运行期间保护应用程序秘密。

设备配置过程初始化设备的安全性，并为每个设备建立唯一的标识符。设置过程将固件的加密版本存储在外部 NV 存储器中。

每次引导时，固件解密都会使用 PUF 设备密钥（索引为零的密钥）将固件解密到片上 RAM 中。在操作中，固件永远不会在设备外部未经加密的情况下使用。

PUF 密钥生成可用于创建额外的设备唯一性密钥，以供应用程序使用。密钥可以是对称的密钥或不对称的公/私密钥对。 例如，应用软件可以使用密钥生成接口来创建供 TLS 协议使用的椭圆曲线公钥对。它启用了设备唯一性密钥的强大存储，可用于远程认证设备。

系统可能具有需要配置到设备中的其他机密。这样的秘密可以使用 PUF 生成的密钥来本地加密和存储此信息。由于该打包信息经过加密，因此可以安全地存储在外部 NV 内存中。

2.1 设备配置

产品在 OEM 工厂使用用于测试和编程的制造工具进行个性化配置。制造工具通过将注册镜像注入并运行到设备中来启动个性化过程。此代码是暂时的，且只能运行一次。

注册镜像的功能包括：

- 设备测试功能。
- PUF 初始化和设备唯一性密钥提取。
- 可选的提取唯一设备标识符（UDI）和其他设备唯一性密钥，以保护其他数据或机密。
- 将 PUF 激活码写入 NV 存储器。
- 在设备供应过程结束时，必须禁用所有调试和边界接口。

注册镜像特定于 MCU 类型，并且可能包含特定于应用程序的测试和初始化。PUF 注册应与现有注册镜像软件集成。

PUF 初始化会生成永远不会脱离设备的秘密密钥。第一个索引为零的索引密钥专用于外部 NV 存储器的加密和解密，并在启动引导过程中使用。

2.2 SRAM PUF 硬件和 AES 引擎

2.2.1 安全启动的用法

LPC54S0xx 没有用于代码和数据存储的内部闪存。应用程序镜像必须存储在外部设备（如 QSPI 和并行闪存）中。AES 引擎可以配置为从外部闪存（SPI，QSPI 或并行闪存）或串行端口（SPI，I2C，UART）解密引导镜像。设备唯一性的 PUF 密钥使任何外部存储的加密信息仅可由已注册 PUF 的设备使用。

LPC54S0xx 的 AES 引擎可以直接使用 SRAM PUF 创建的索引为 0 的密钥。与 OTP 相比，PUF 的使用为系统安全性提供了可观的好处。不可复制的 PUF 密钥对于一台设备是唯一的，并保证加密的信息只能与该设备一起使用。

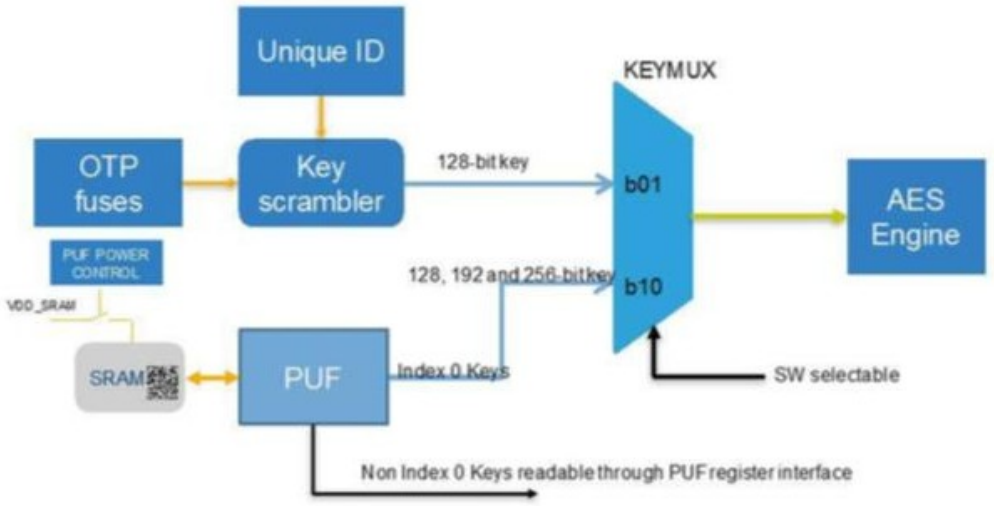

图 1. SRAM PUF 支持使用设备特定的不可复制密钥加密信息

PUF 硬件可用于生成应用程序的密钥和密钥标识符。这些密钥从未被存储，并且是唯一且不可复制的。密钥的大小可以从 64 位到 4096 位，并且可以由应用程序用作对称或非对称密钥。

2.2.2 派生应用程序的设备唯一性密钥

PUF 的机密可由应用程序使用。

2.2.3 保护用户机密

PUF 硬件支持保护用户机密。机密可以是机密配置信息或应用程序所需的密钥。通常在初始设备配置的工厂部分提供此秘密信息。此保护也可以在操作设备中使用，以保护选定的机密（例如，用户配置的密码）。这些机密经过加密，并以密钥代码（ Key Code, KC）的格式保护其完整性。密钥代码（KC）格式支持保护任意用户提供的信息。此信息的大小可以从 64 位到 4096 位。用于包装这些机密的设备唯一性密钥无法由软件读取，因此密钥代码可以安全地存储在外部非易失性存储器中。受保护的信息必须字（word）对齐。

3 示例软件片段

本节包含支持在 LPC54S0xx 上集成基于 PUF 的安全功能以创建安全应用程序的代码示例。

3.1 设置示例代码

以下示例代码应融合到制造配置过程中。提供的代码示例可用于：

- PUF 的制造期间设置。
- LPC54S0xx AES 引擎的制造期间配置，以使用 PUF 生成的密钥并加密安装的镜像。
- 制造期间配置以锁定调试和边界扫描接口。

本节提供了可以用作驱动程序代码开发起点的上层代码。该示例代码使用状态轮询来控制流程。

注意

1. 为了描述清楚起见，使用状态轮询方法。为了提高操作效率，建议使用中断驱动的架构。
2. 假定密钥直接进入设计的安全部分，见 图 1，所以不定义密钥输出寄存器。

复位 PUF 之后（用或不用进行 SRAM 的重新上、下电），它需要一些时间来初始化（由 busy 标识）。假定系统在开始发出命令之前等待初始化完成。函数 puf_waitForInit 可以用于此目的。

注意

该代码不包括用于打开和关闭 SRAM 的操作。假设它是在系统的其他部分完成的。

表 1. 词汇表

CTRL	PUF 控制寄存器
STAT	PUF 状态寄存器
ALLOW	PUF 允许寄存器
CODEINPUT	PUF 代码输入寄存器
CODEOUTPUT	PUF 代码输出寄存器
ENROLL	PUF 控制寄存器 bit1 –开始“注册”操作
START	PUF 控制寄存器 bit2 –开始“开始”操作
BUSY	PUF 状态寄存器 bit0 – 表示操作正在进行中
SUCCESS	PUF 状态寄存器 bit1 – 上一次操作成功
ERROR	PUF 状态寄存器 bit2 –PUF 处于错误状态； 不允许任何操作
CODEOUTAVAIL	PUF 状态寄存器 bit7 –AC/KC 的下一部分可用
ALLOWENROLL	PUF 允许寄存器 bit0 –允许“注册”操作
ALLOWSTART	PUF 允许寄存器 bit1 –允许“开始”操作
Initialize (target)	清空目标数据结构。
get_data (data, source)	从源结构中检索下一个数据字（word），将其放入数据中，并将其从头部移除。
append_data (data, target)	将数据中的数据追加到目标的末尾。

3.1.1 PUF 设置代码示例

在设备配置期间，应通过注册镜像执行以下代码，以获取:

- 唯一的设备 ID。
- 设备激活码（AC）。

随后必须将激活码（AC）存储在非易失性存储器中，以备后用。

注意

激活码（AC）不需要存储在可寻址的内存 NVRAM 中。

3.1.1.1 用于初始化的 PUF 配置代码示例

```
status PUF_waitForInit() { 2
// wait until initialization has finished
while (*STAT & BUSY != 0) {} 5
// check that initialization has passed
if (*STAT & SUCCESS == 0) {
return ERROR
}
 return OK
 }
```

3.1.1.2 用于注册的 PUF 配置代码示例

```
/* output: ACdata – byte array for Activation Code storage */
status_t PUF_Enroll(
PUF_Type *base,
uint8_t *activationCode,
size_t activationCodeSize) 17    {
// clear the ACdata storage
initialize(ACdata) 20
// check if Enroll is allowed
if (*ALLOW & ALLOWENROLL == 0) {
return NOT_ALLOWED 24    }

// Make sure that the module that receives the key is initialized
// and can accept the key
// begin Enroll
*CTRL = ENROLL 30
// wait till command is accepted
while (*STAT & (BUSY | ERROR) == 0) {
}
// while busy read AC
while (*STAT & BUSY != 0) {
if (*STAT & CODEOUTAVAIL != 0) {
tempData = *CODEOUTPUT
append_data(tempData, ACdata) 39    }
// During this loop the key is transported to the
// receiving module using the interlocked interface
} // while
// check result
if (*STAT & SUCCESS == 0) {
return ERROR 46    }
return OK
}
```

3.1.1.3 用于 Start 的 PUF 配置代码示例

```
/* input: activationCode – byte array containing Activation Code */ 50
status_t PUF_Start(
PUF_Type *base,
const uint8_t *activationCode,
size_t activationCodeSize) 55    {
// check if Start is allowed
if (*ALLOW & ALLOWSTART == 0) {
return NOT_ALLOWED 59    }

```

```
61    // Make sure that the module that receives the key is initialized
62    // and can accept the key
63    // begin Start
64    *CTRL = START 65
66    // wait till command is accepted
67    while (*STAT & (BUSY | ERROR) == 0) { 68    }
69
70    // while busy send AC, read key
71    while (*STAT & BUSY != 0) {
72    if (*STAT & CODEINREQ != 0) {
73    get_data(tempData, activationCode)
74    *CODEINPUT = tempData 75    }
76
77    // During this loop the the key is transported to the
78    // receiving module using the interlocked interface
79    } // while
80
81    // check result
82    if (*STAT & SUCCESS == 0) {
83    return ERROR 84    }
85    return OK
86    }
```

3.2 PUF 的 AES 引擎用法

LPC54S0xx 上的 AES 引擎可以配置为直接使用来自 PUF 引擎的密钥。以下代码段演示了 PUF 密钥到 AES 引擎进行加密和解密的路由和用法。

3.2.1 使用 PUF 的 AES 引擎配置

必须将 AES 引擎配置为使用 PUF 密钥而不是 OTP 密钥。以下代码提供了此配置的详细信息。它显示了如何选择索引为 0 的 PUF 密钥。

```
87    #define PUF_INTRINSIC_KEY_SIZE 16
88    status_t result;
89    uint8_t keyCode[PUF_GET_KEY_CODE_SIZE_FOR_KEY_SIZE(PUF_INTRINSIC_KEY_SIZE)]; 90
91    result = PUF_SetIntrinsicKey(PUF, kPUF_KeyIndex_00, PUF_INTRINSIC_KEY_SIZE, keyCode,
sizeof(keyCode));
92    if(result != kStatus_PUF_Success) return result; 93
94    /* Get Intrinsic Key */
95    result = PUF_GetHwKey(PUF, keyCode, sizeof(keyCode), kPUF_KeySlot0, rand());;
96    if(result != kStatus_PUF_Success) return result;
```

3.2.2 AES 引擎使用 PUF 派生密钥进行加密

PUF 生成的设备唯一性密钥用于 AES 加密的使用可以与引导过程集成在一起，以保护固件镜像。

PUF 生成的密钥可与 AES 一起使用以创建数据的安全存储：如使用 PUF 的 AES 引擎配置所示设置 KEY。

```
97    /* Applicatin_Specific iv is 96-bit unique value */
98    uint8_t explicit_iv[12] =
         {0x58,0x9C,0x84,0xD1,0x7D,0x1B,0x43,0xBE,0x57,0xCB,0xD6,0xD9};
99
100    /* Application Specific/
101    uint8_t aad[] =
102       { 0xDF, 0x76, 0xB5, 0x5A, 0x48, 0x8E, 0x68, 0xF8,
103    0xF9, 0xCB, 0x82, 0x95, 0xC9, 0x1A, 0xCF, 0xEB };
104
105       uint8_t tag[16];
```

```
106
107       status = AES_EncryptTagGcm( APP_AES, clear_data,encrypt_data, *image_length,
108        explicit_iv, sizeof(explicit_iv),
109        aad, sizeof(aad),
110        tag, sizeof(tag) );
```

3.2.3 AES 引擎使用 PUF 进行解密

以下代码示例描述了使用 PUF 派生密钥来解密外部存储：如使用 PUF 的 AES 引擎配置所示设置 KEY。

```
111    /* iv is 96-bit unique value */
112    uint8_t explicit_iv[12] =
          {0x58,0x9C,0x84,0xD1,0x7D,0x1B,0x43,0xBE,0x57,0xCB,0xD6,0xD9};
113
114    uint8_t aad[] =
115       { 0xDF, 0x76, 0xB5, 0x5A, 0x48, 0x8E, 0x68, 0xF8,
116    0xF9, 0xCB, 0x82, 0x95, 0xC9, 0x1A, 0xCF, 0xEB };
117
118        uint8_t tag[16];
119
120    status = AES_DecryptTagGcm( APP_AES, encrypt_image, clear_image, *image_length,
121                                explicit_iv, sizeof(explicit_iv),
122                                aad, sizeof(aad),
123                                tag, sizeof(tag) );
```

3.3 用户密钥保护的示例

以下代码示例用于使用 PUF 生成的设备唯一性密码来加密或解密用户密钥。应用程序可以使用此密钥包装和解包来安全地存储机密信息。

3.3.1 包装（设置密钥，Set Key）

以下代码段演示了如何加密应用程序的机密信息。设备唯一性的加密允许将创建的密钥存储在设备之外的任何非易失性存储中。

KC：密钥代码，Key Code

```
124    status SetIntrinsicKey(KCdata, KeyIndex, KeySize) {
125    // clear the KCdata storage initialize(KCdata)
126    // check if Set Key is allowed
127    if (*ALLOW & ALLOWSETKEY == 0) {
128    return NOT_ALLOWED
129 }
130
131    // program the key size and index
132    *KEYSIZE = KeySize >> 6 // convert to 64-bit blocks
133    *KEYINDEX = KeyIndex 134
135    // begin Set Key
136    *CTRL = GENERATEKEY 137
138    // wait till command is accepted
139    while (*STAT & (BUSY | ERROR) == 0) {
140 }
141    // while busy read KC
142    while (*STAT & BUSY != 0) {
143    if (*STAT & CODEOUTAVAIL != 0) {
144    tempData = *CODEOUTPUT
145    append_data(tempData, KCdata)
146 }
147 } // while 148
149    // check result
```

```
150    if (*STAT & SUCCESS == 0) {
151    return ERROR
152 }
153 return OK
154 }
155    status SetUserKey(KCdata, KeyIndex, UKdata) {
156    // clear the KCdata storage
157    initialize(KCdata) 158
159    // check if Set Key is allowed
160    if (*ALLOW & ALLOWSETKEY == 0) {
161    return NOT_ALLOWED
162 } 163
164    // detect key size
165    KeySize = length_in_bits(UKdata) 166
167    // program the key size and index
168    *KEYSIZE = KeySize >> 6 // convert to 64-bit blocks
169    *KEYINDEX = KeyIndex 170
171    // begin Set Key
172    *CTRL = SETUSERKEY 173
174    // wait till command is accepted
175    while (*STAT & (BUSY | ERROR) == 0) {} 176
177    // while busy write UK and read KC
178    while (*STAT & BUSY != 0) {
179    if (*STAT & KEYINREQ != 0) {
180    get_data(tempData, UKdata)
181    *KEYINPUT = tempData
182 }
183    if (*STAT & CODEOUTAVAIL != 0) {
184    tempData = *CODEOUTPUT
185    append_data(tempData, KCdata)
186 }
187    } // while
188    // check result
189    if (*STAT & SUCCESS == 0) {
190    return ERROR
191 }
192 return OK
193 }
```

3.3.2 打开包装（获取密钥，Get Key）

以下代码段演示了如何解密应用程序机密信息。

获取密钥示例（打开包装）：

```
194 status GetKey( KCdata, KeyIndex, KeyData) {
195 // clear the KeyData storage
196 initialize(KeyData)
197 // put unused value in KeyIndex
198 KeyIndex = 255
199
200 // check if Get Key is allowed
201 if (*ALLOW & ALLOWGETKEY == 0) {
202 return NOT_ALLOWED
203 }
204
205 // begin Get Key
206 *CTRL = GETKEY
207
```

```
208 // wait till command is accepted
209 while (*STAT & (BUSY | ERROR) == 0) {
210 }
211
212 // while busy send KC, read key
213 while (*STAT & BUSY != 0) {
214 if (*STAT & CODEINREQ != 0) {
215 get_data(tempData, KCdata)
216 *CODEINPUT = tempData
217 }
218 if (*STAT & KEYOUTAVAIL != 0) {
219 KeyIndex = *KEYOUTINDEX
220 tempData = *KEYOUTPUT
221 append_data(tempData, KeyData)
222 }
223 } // while
224
225 // check result
226 if (*STAT & SUCCESS == 0) {
227 return ERROR
228 }
229 return OK
230 }
```

4 总结

LPC54S0xx 器件提供 PUF 功能以生成和使用设备唯一性的加密密钥。这些密钥可用于支持将固件镜像或其他用户数据安全地存储在片外非易失性存储器中。密钥也可用于应用程序身份验证服务。本应用笔记提供了使用说明和详细的代码示例，以将 PUF 集成到 LPC54S0xx 中。

How To Reach Us

Home Page:

nxp.com

Web Support:

nxp.com/support

Limited warranty and liability — Information in this document is provided solely to enable system and software implementers to use NXP products. There are no express or implied copyright licenses granted hereunder to design or fabricate any integrated circuits based on the information in this document. NXP reserves the right to make changes without further notice to any products herein.

NXP makes no warranty, representation, or guarantee regarding the suitability of its products for any particular purpose, nor does NXP assume any liability arising out of the application or use of any product or circuit, and specifically disclaims any and all liability, including without limitation consequential or incidental damages. “Typical” parameters that may be provided in NXP data sheets and/or specifications can and do vary in different applications, and actual performance may vary over time. All operating parameters, including “typicals,” must be validated for each customer application by customer's technical experts. NXP does not convey any license under its patent rights nor the rights of others. NXP sells products pursuant to standard terms and conditions of sale, which can be found at the following address: nxp.com/SalesTermsandConditions.

Right to make changes - NXP Semiconductors reserves the right to make changes to information published in this document, including without limitation specifications and product descriptions, at any time and without notice. This document supersedes and replaces all information supplied prior to the publication hereof.

Security — Customer understands that all NXP products may be subject to unidentified or documented vulnerabilities. Customer is responsible for the design and operation of its applications and products throughout their lifecycles to reduce the effect of these vulnerabilities on customer’s applications and products. Customer’s responsibility also extends to other open and/or proprietary technologies supported by NXP products for use in customer’s applications. NXP accepts no liability for any vulnerability. Customer should regularly check security updates from NXP and follow up appropriately. Customer shall select products with security features that best meet rules, regulations, and standards of the intended application and make the ultimate design decisions regarding its products and is solely responsible for compliance with all legal, regulatory, and security related requirements concerning its products, regardless of any information or support that may be provided by NXP. NXP has a Product Security Incident Response Team (PSIRT) (reachable at PSIRT@nxp.com) that manages the investigation, reporting, and solution release to security vulnerabilities of NXP products.

For more information, please visit: http://www.nxp.com

For sales office addresses, please send an email to: salesaddresses@nxp.com

Date of release: 2018 年 11 月 6 日

Document identifier: AN12292

arm