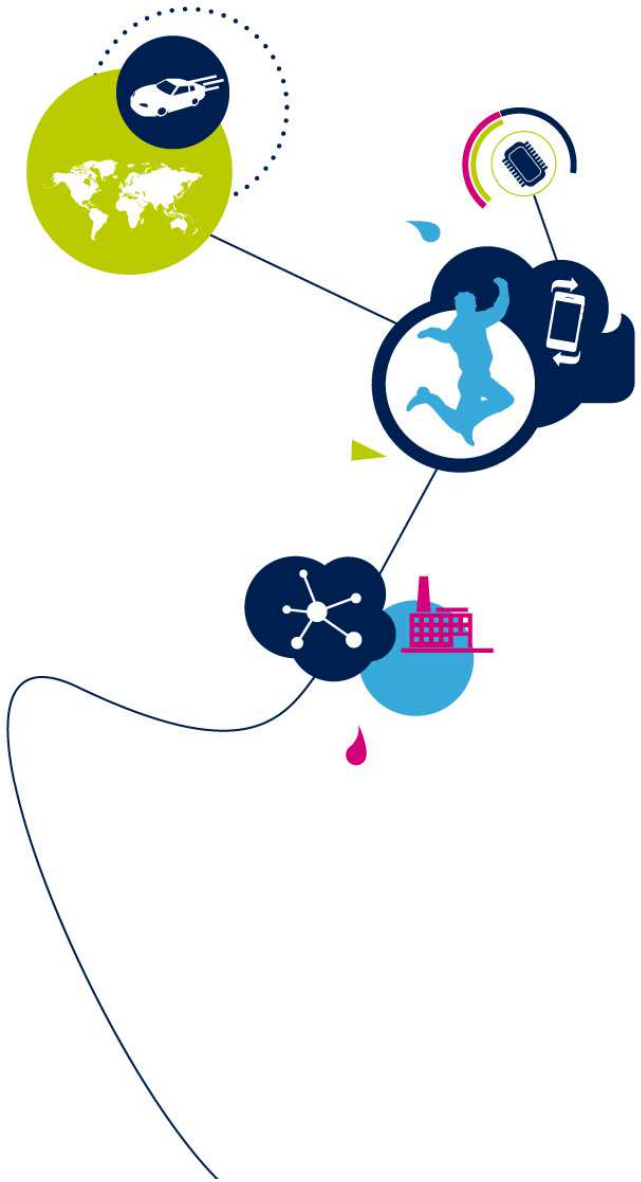
# STM32软硬件安全技术

STM32 Security

MCU China 2019



life.augmented

# STM32 Security典型应用



- 保护固件代码
  - STM32技术： STM32 RDP, PCROP
- 保护标志身份的密钥
  - STM32技术：STM32 SBSFU，STM32 Cryptolib
- 物联网安全---保护标志身份的密钥与通讯安全
  - STM32技术： STM32 SBSFU，MbedTLS
- 保护其他设备
  - STM32技术：STM32 Cryptolib

STM32用户可以免费、灵活地选择

- 直接使用高层软件模块
  - STM32 SBSFU
  - MbedTLS
- 单独以及组合选用底层组件
  - RDP
  - PCROP
  - STM32 Cryptolib
  - 其他

# STM32 Security



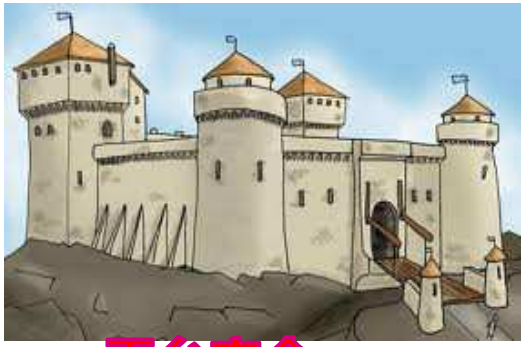
## STM32 功能宏模块

- 安全启动Secure Boot (SBSFU)
  - 信任根
- 安全固件更新 (SBSFU)
  - 阻止未授权固件更新(完整 & 认证)
  - 保护明文固件 (保密)
- 安全固件安装(SFI)
  - 固件保密
  - 量产计数
- 通讯安全TLS
  - 身份认证
  - 数据加密

# STM32 安全硬件功能以及加密库

- RDP
- PCROP
- WRP
- Secure User Memory
- **TrustZone**
- MPU
- Firewall
- Tamper Features…

- Crypto Hardware Accelerators
  - AES
  - Hash
- True Random Number Generator
- 96-Bit Unique ID
- **Crypto Library / Primitives**



**平台安全**

通讯安全

life.augmented

# 文档、软件、工具



文档

- **STM32 Security Introduction**
  - AN5156 ★
- **RDP & PCROP**
  - AN4701, STM32F4
  - AN4758, STM32L4
  - AN4968, STM32F7
  - AN4246, STM32L1
- **Firewall**
  - AN4729, STM32L0 & STM32L4:
- **MPU**
  - AN4838, STM32
- **Tamper**
  - AN3371, STM32
- **TRNG**
  - AN4230, STM32
- **SBSFU**
  - AN5056, Integration guide, STM32 ★
  - UM2262, STM32 ★

软件

- **X-CUBE-SBSFU**
- **TLS in CubeMX or X-CUBE-<<IOT>>**
- **X-CUBE-CRYPTOLIB**
- X-CUBE-PCROP

工具

- STM32 ST-LINK Utility
- STM32CubeMx
- STM32TrustedPackageCreator
- STM32CubeProgrammer

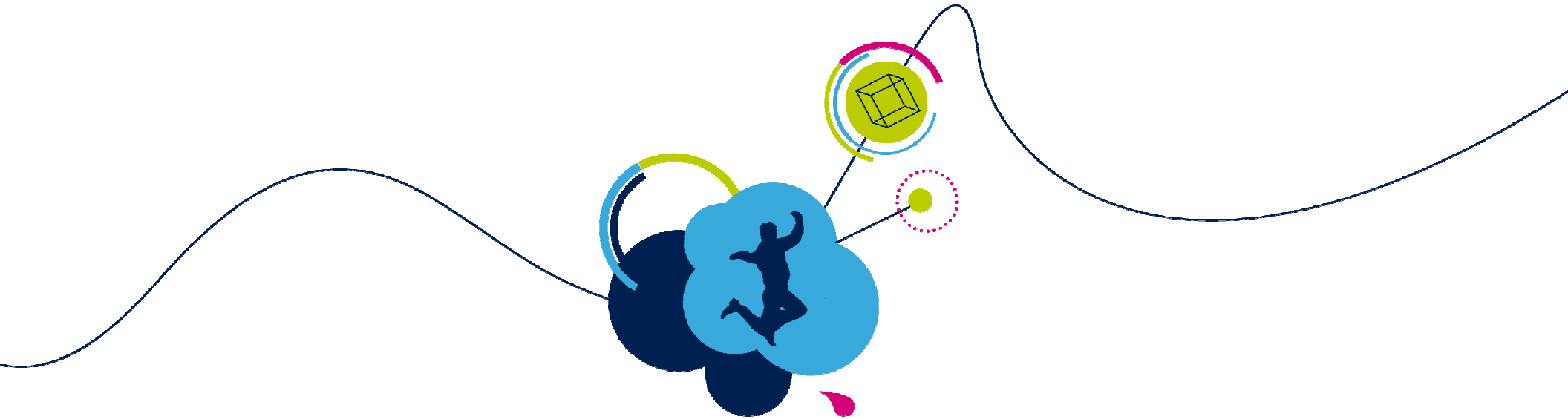
- **提示**
  - **访问 www.st.com**
  - **搜索关键字: AN5156 or X-CUBE-CRYPTOLIB**

life.augmented

# 加解密工具



- OpenSSL
- CrypTool
  - https://www.cryptool.org/en/cryptool2



# 安全启动
# (STM32 SBSFU)

life.augmented

# 安全启动的作用

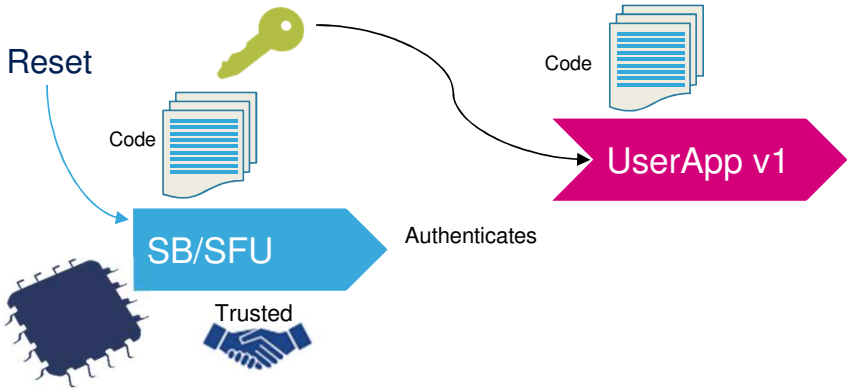


- 防非法固件更新
  - 刷机
  - 破坏
- 防恶意软件
- 防盗取机密数据
- 为其它安全应用提供基础支撑

# 安全启动的概念



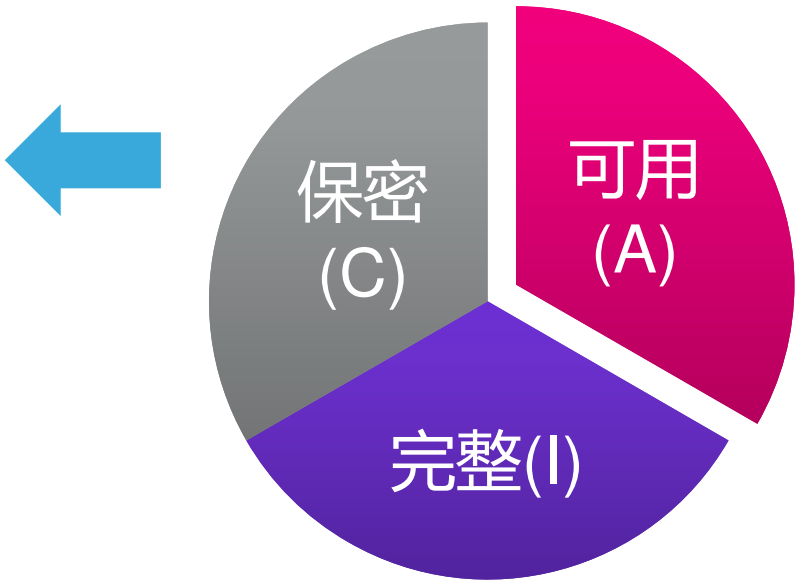
- 安全启动是系统启动的信任起点
  - 硬件保证
- 多阶段启动中从起点起逐级认证
  - 使用加解密技术



# 安全启动的原则

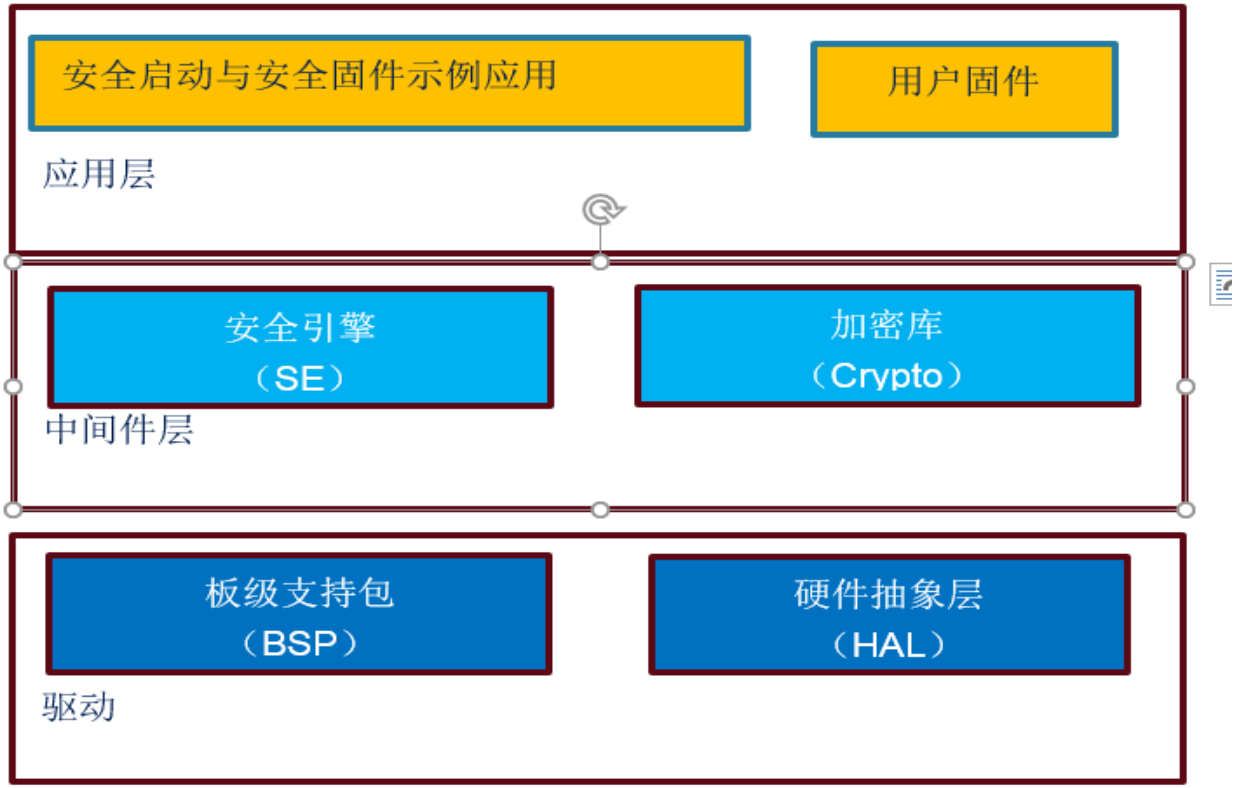


- 信任起点
  - 单一启动入口
  - 启动顺序不可改变
  - 安全启动与后续隔离(可选)
- 使用加解密技术创建信任链



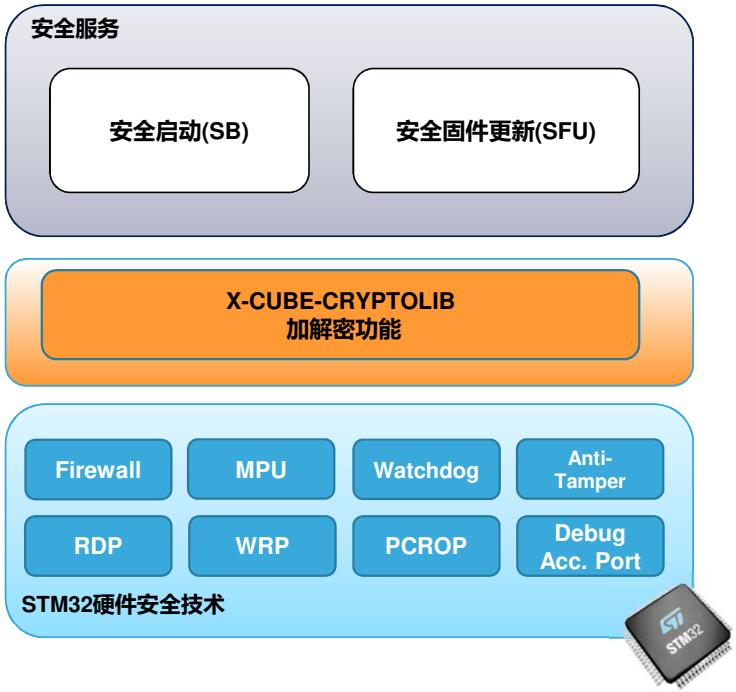
# STM32 SBSFU 软件架构





# STM32 SBSFU 安全架构



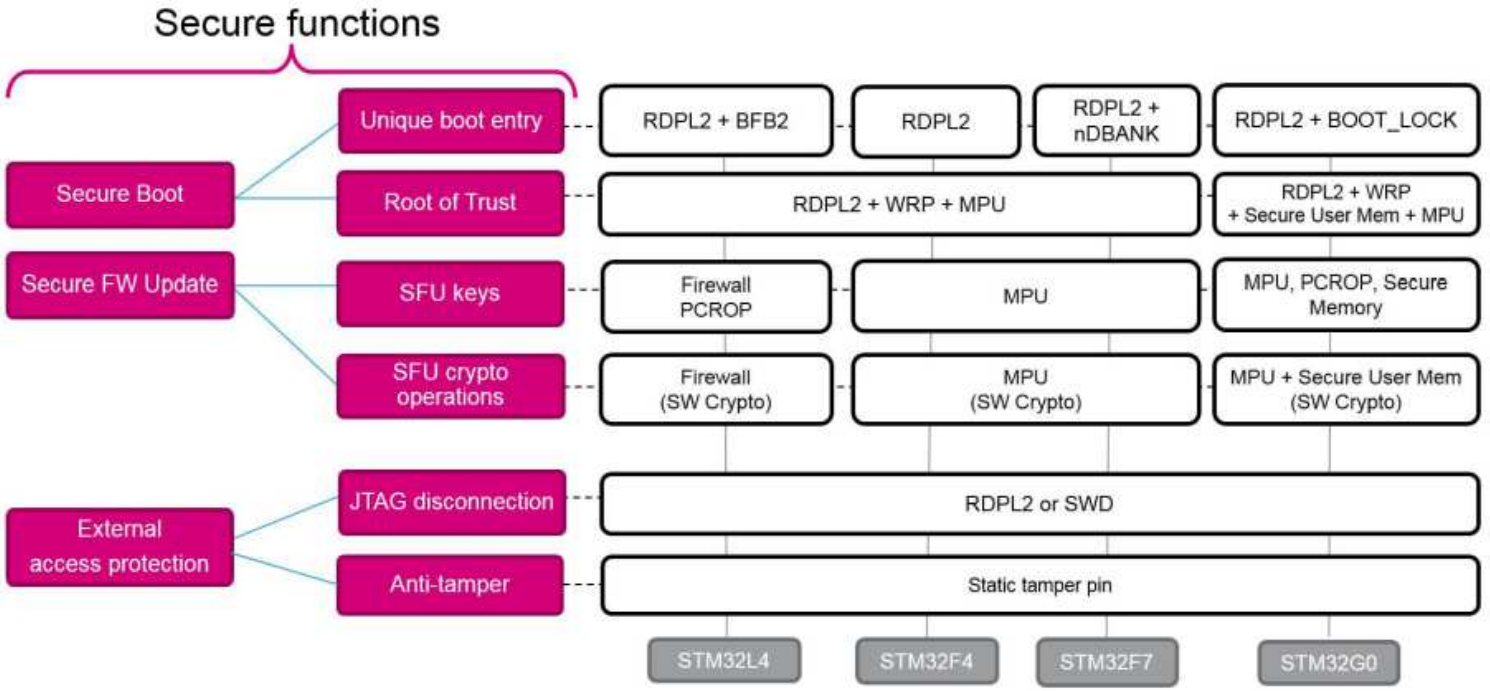



- STM32 硬件安全技术
  - 实现可信环境以运行可信计算，例如加解密
- 加解密功能
  - 实现保密，完整，认证
- 安全启动与安全固件更新
  - 综合STM32硬件安全技术与加解密功能实现多层次的保护

life.augmented

# STM32 SBSFU中的硬件安全技术



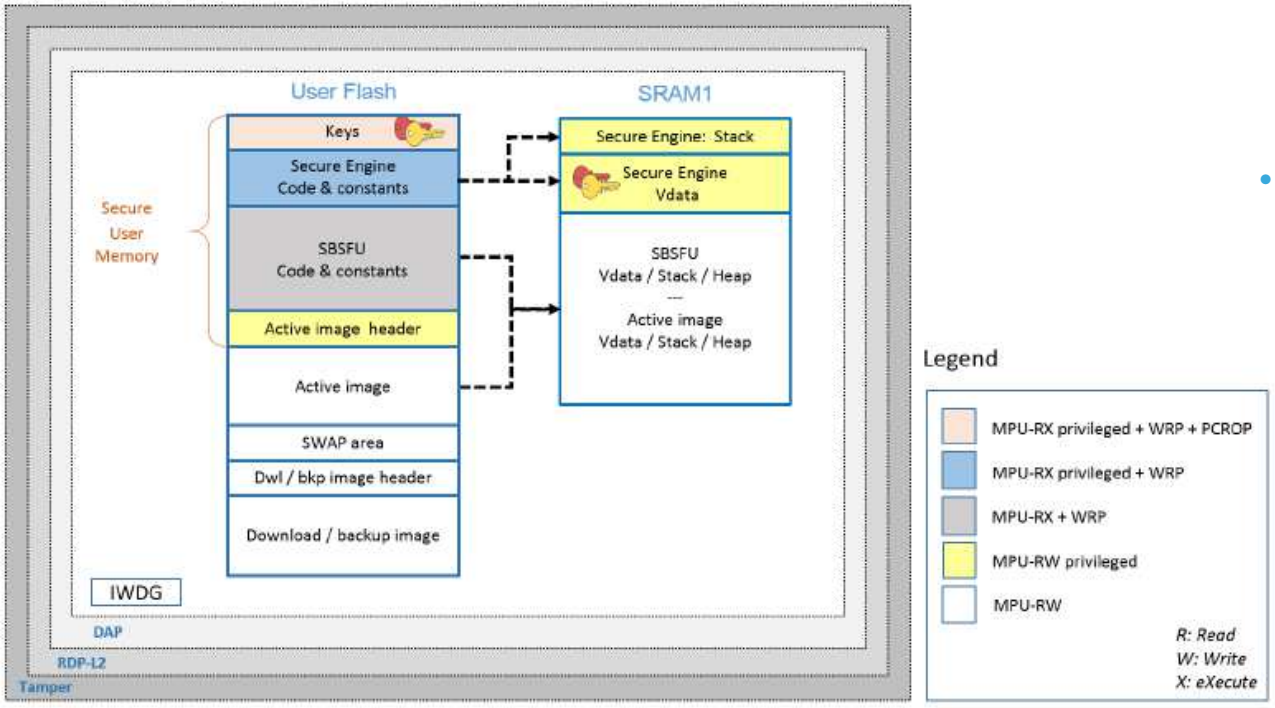



STM32不同系列的安全实现会有差异

life.augmented

# STM32G0 SBSFU硬件安全的内存布局



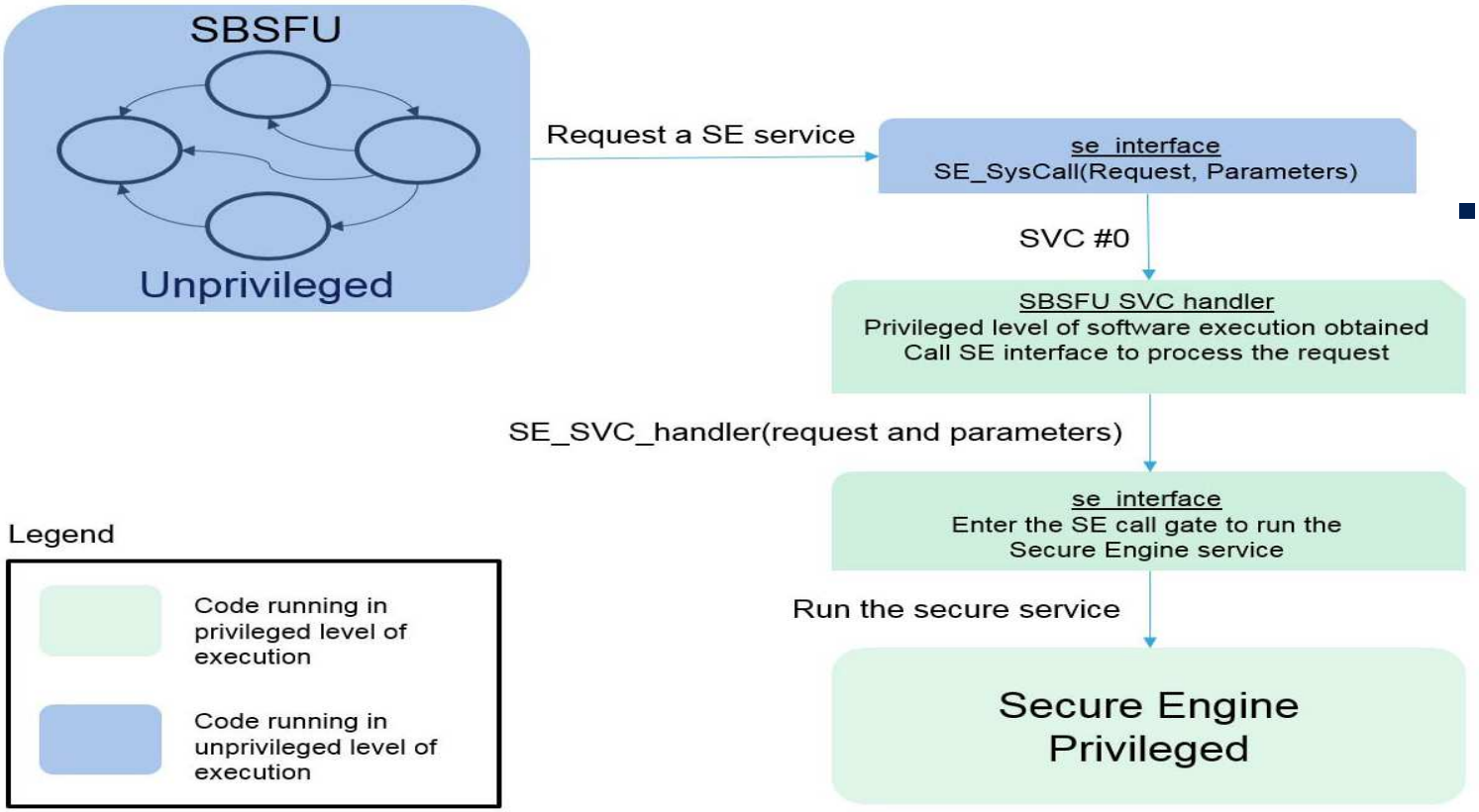



- 安全引擎保护
  - 密钥
  - 存放错误状态的引导区数据
- 安全引擎可供
  - 安全启动使用
  - 安全固件更新
  - 应用程序
    - *若使用了Secure User memory或者Trustzone隔离，则不能直接使用

life.augmented

# STM32G0 SBSFU请求安全引擎服务



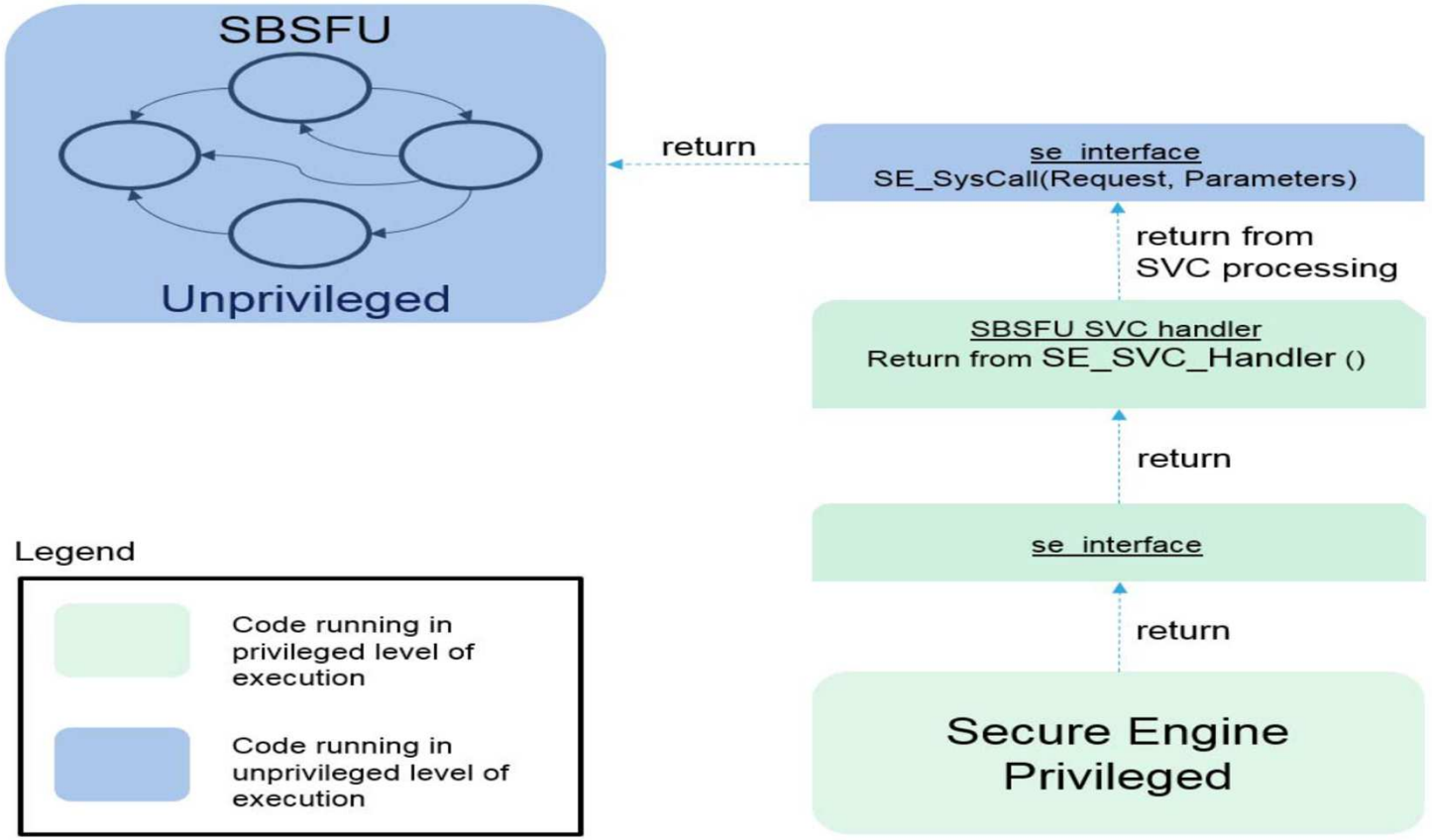



- 唯一的调用门入口
  - 这里基于MPU
  - STM32L0/L4也可以基于Firewall

life.augmented

# STM32G0 SBSFU离开安全引擎

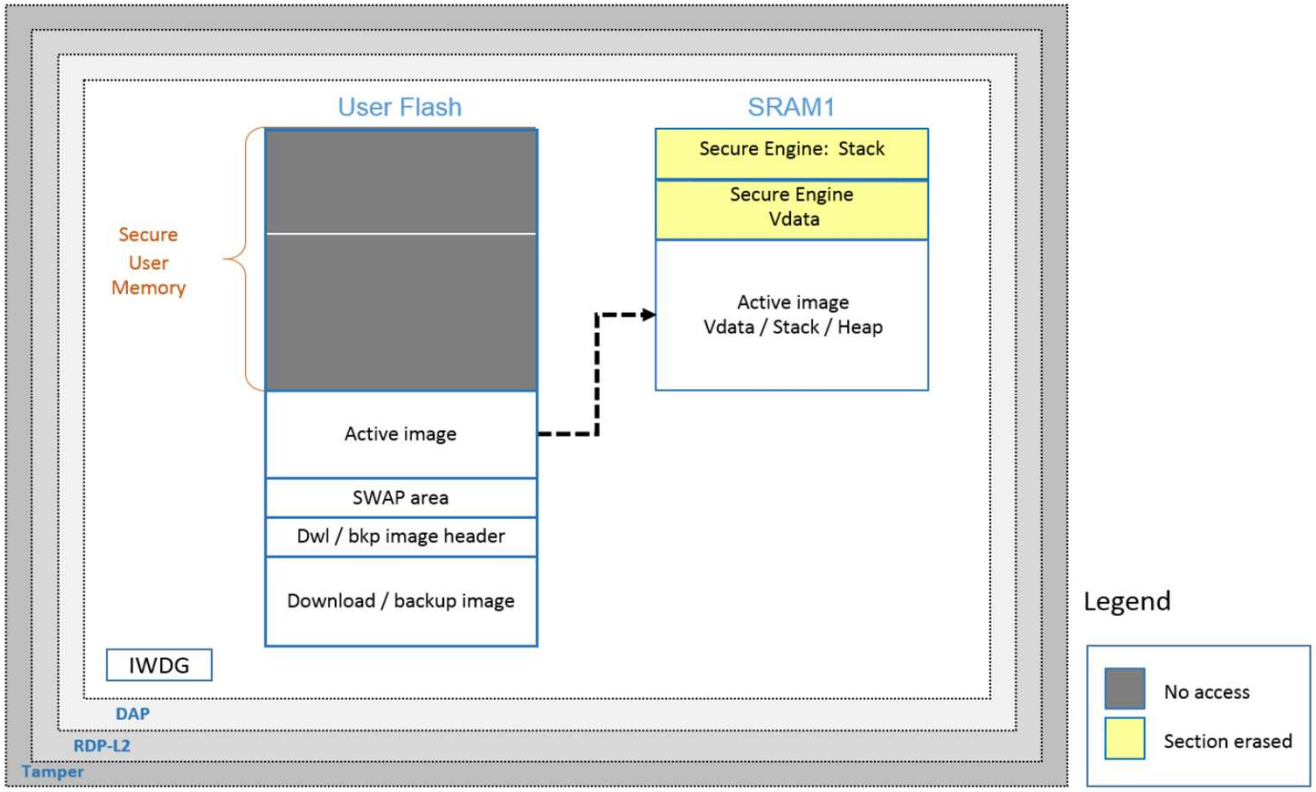





- 非STM32G0请参考用户手册

# STM32G0 SBSFU用户固件运行时内存



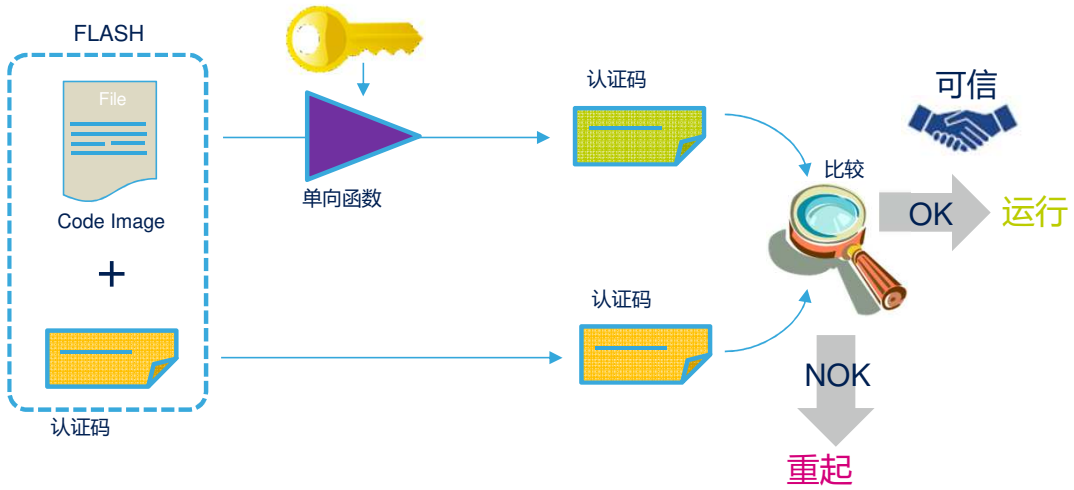



- Secure User Memory保护用户代码不能访问安全启动

# 信任链的构建



- 安全启动在构造安全环境后，使用对称密钥构建信任链

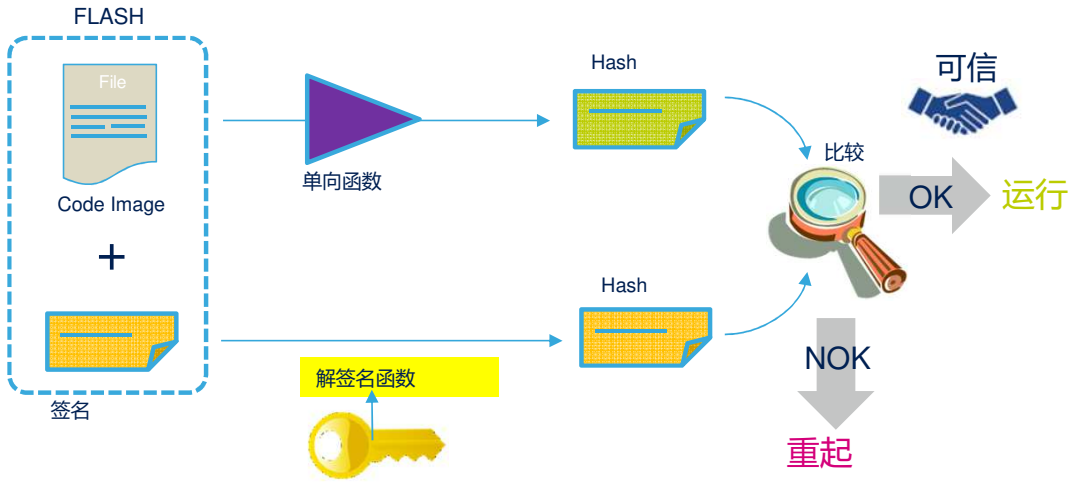



基于对称密钥

# 信任链的构建



- 安全启动在构造安全环境后， 使用非对称密钥构建信任链




基于非对称密钥

# STM32 SBSFU中的加解密技术

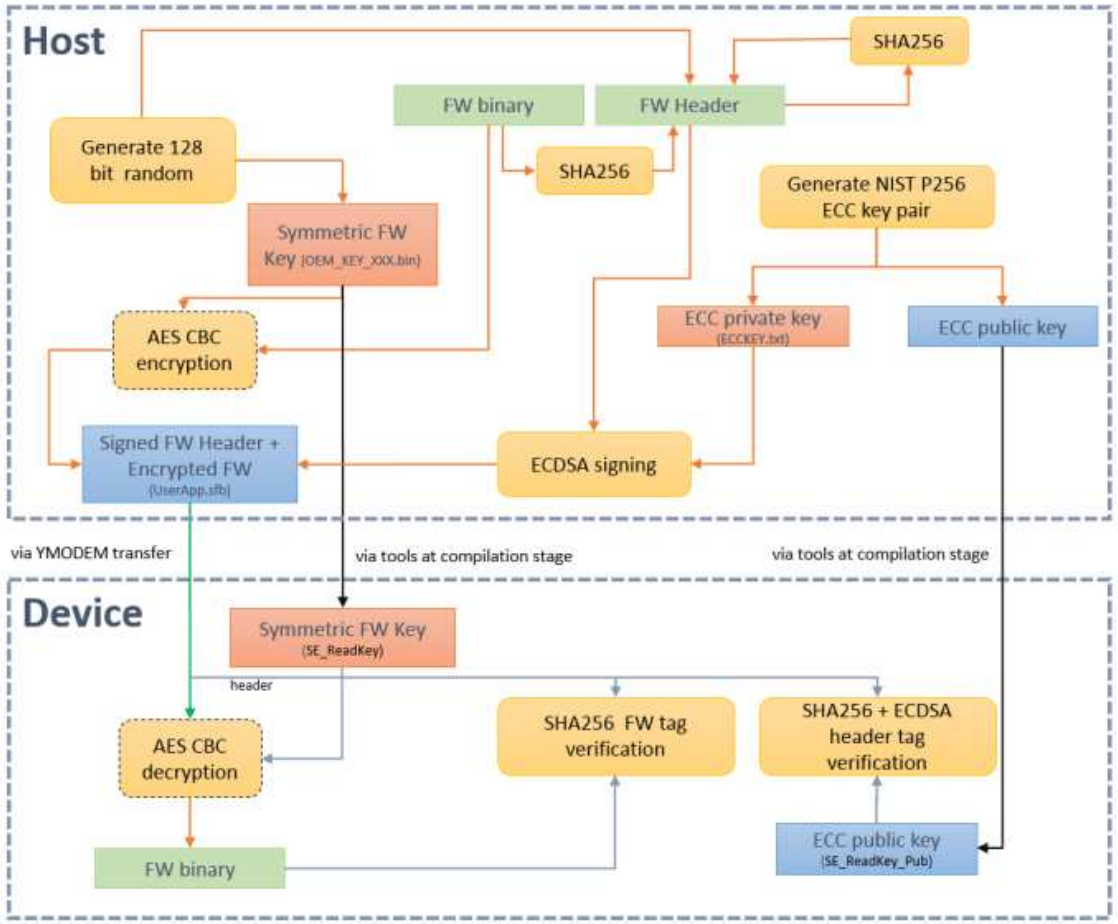


| 功能 | 非对称密钥认证+对称密钥加密 | 非对称密钥认证(无加密) | 对称密钥认证与加密 |
|---|---|---|---|
| 保密 | AES-CBC | 无 | AES-GCM |
| 完整 | SHA256 | | AES-GCM |
| 认证 | ECDSA | | |
| MCU中的密钥存储 | AES密钥<br>ECC公钥 | ECC公钥 | AES密钥 |

life.augmented

# STM32 SBSFU中的加解密技术流程






非对称密钥认证
+对称密钥加密

编译时工具处理

运行时设备处理

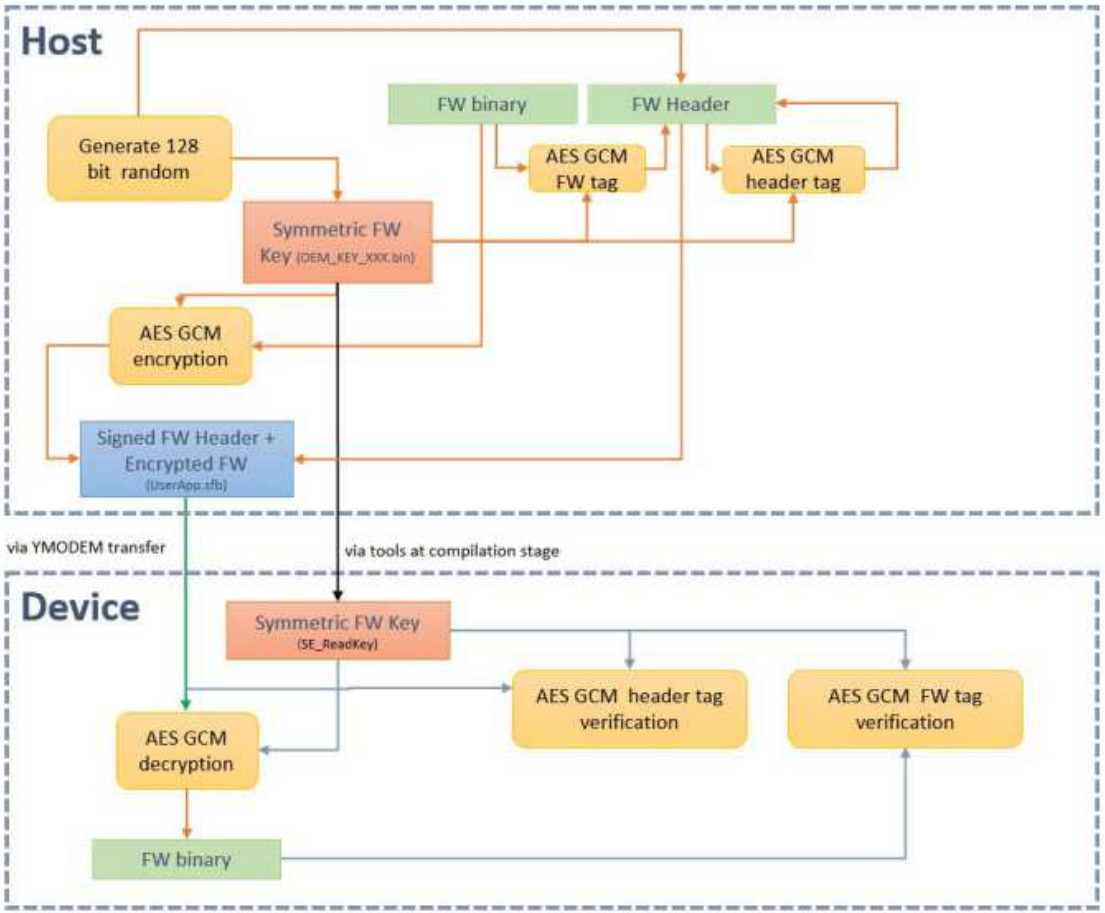
通讯

随代码烧入设备

life.augmented

# STM32 SBSFU中的加解密技术流程






对称密钥认证
+加密

编译时工具处理

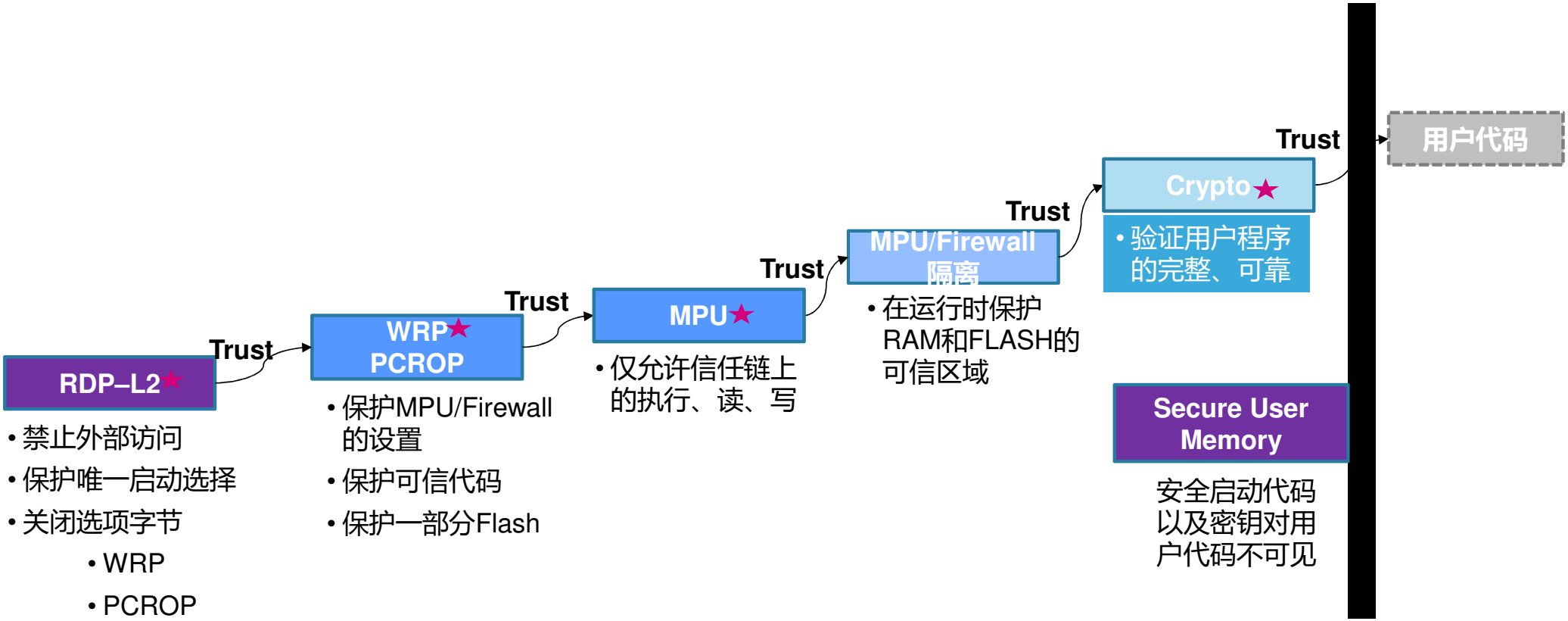
运行时设备处理

通讯

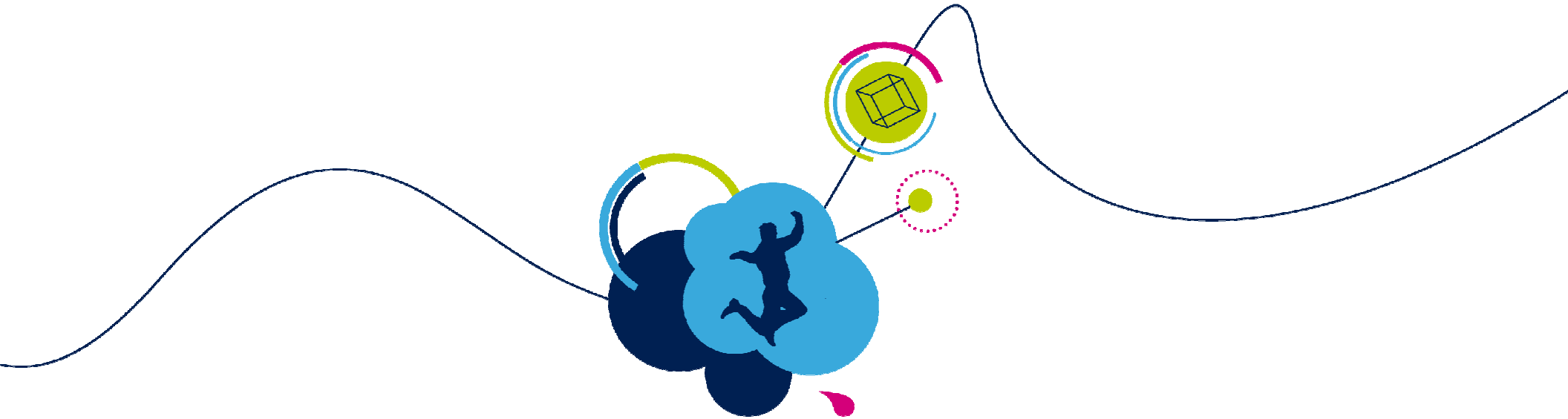
随代码烧入设备

# STM32 安全启动技术总结






life.augmented

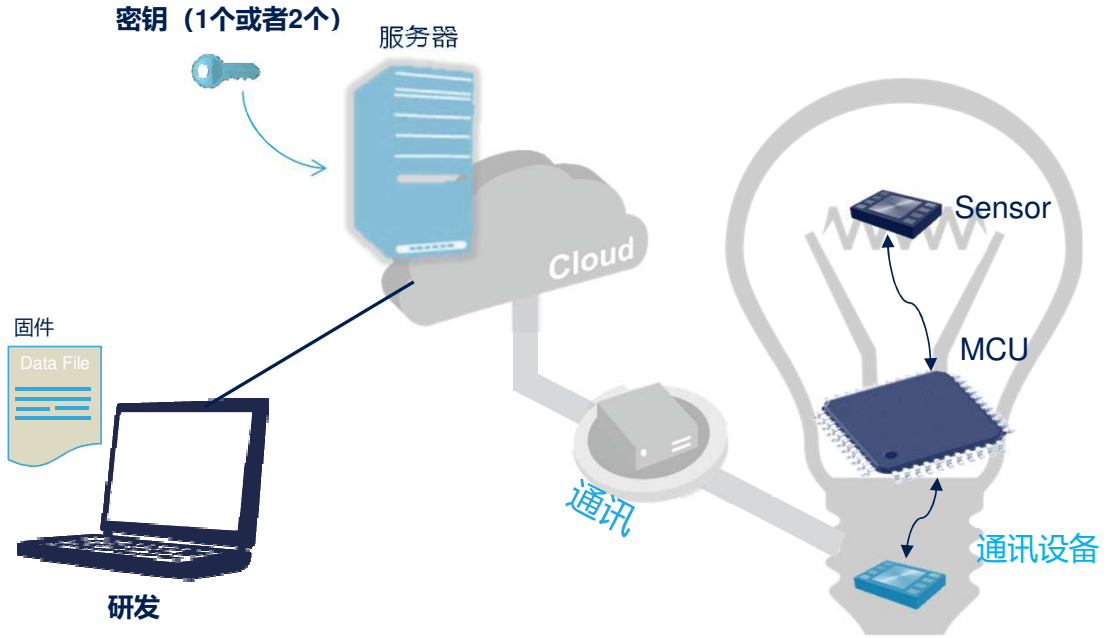


# 安全固件更新
# (STM32 SBSFU)

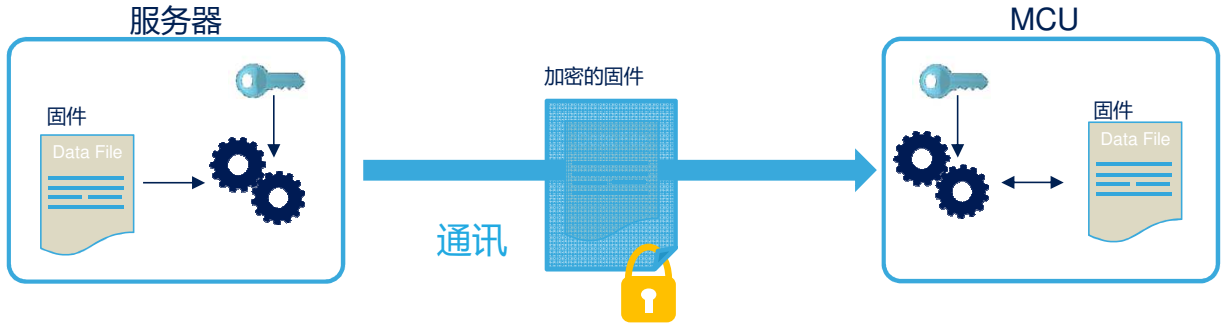
life.augmented

# 安全固件更新的概念





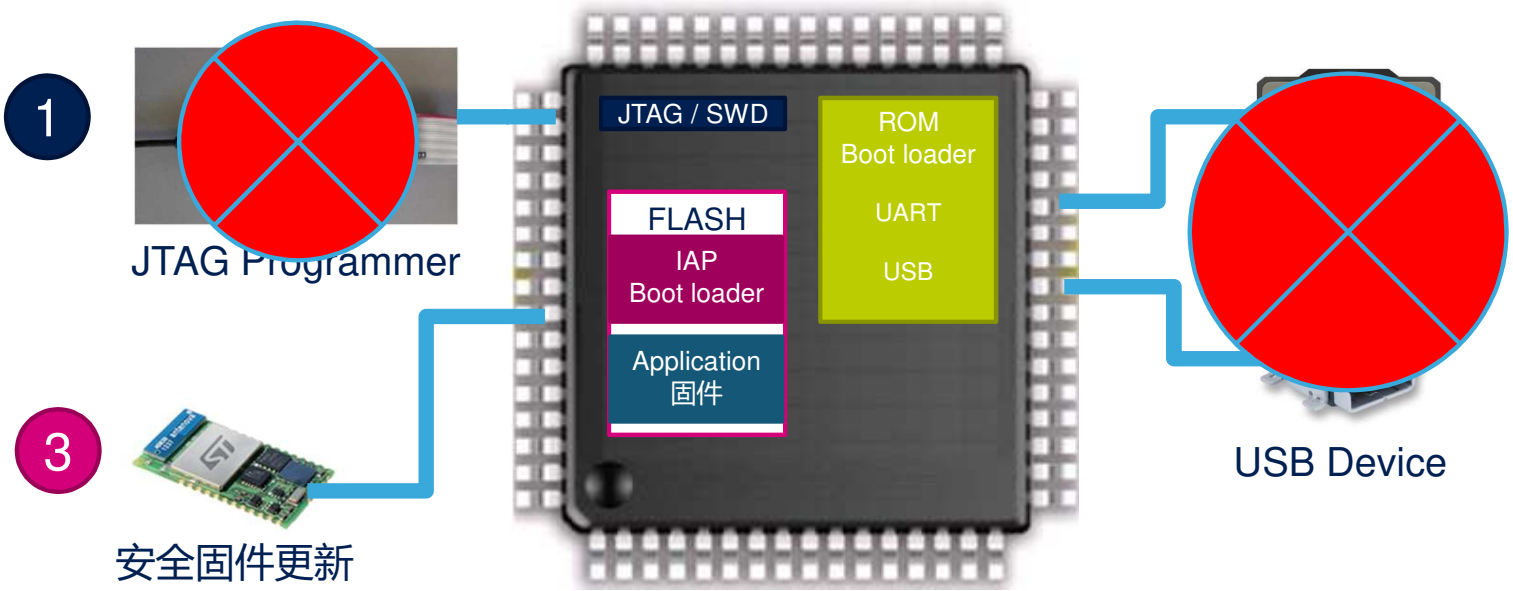

- 流程：
  - 服务器：发送 固件
  - 设备:接收，存储，执行固件
    - 安全要求：
      - ✓ 固件保密(可选)
      - ✓ 固件完整
      - ✓ 固件来源可靠




life.augmented

# 安全固件更新的入口






RDP 2

# 两种镜像模式



- 单镜像
  - 更多用户空间
  - 适合更小的Flash要求
  - 不支持用户更新
- 双镜像
  - 可以回退。更安全
  - 可以在用户应用程序里更新

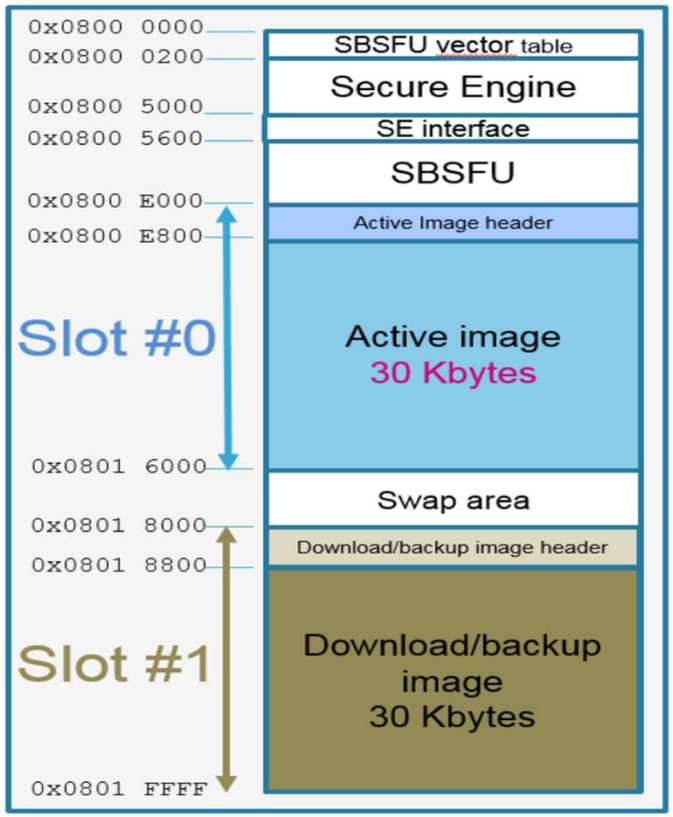
# 升级包的组成



- 固件头
  - 描述固件大小，版本，以及加解密和验证信息等
- 固件镜像
  - 实际运行代码

# STM32 SBSFU安全固件更新服务



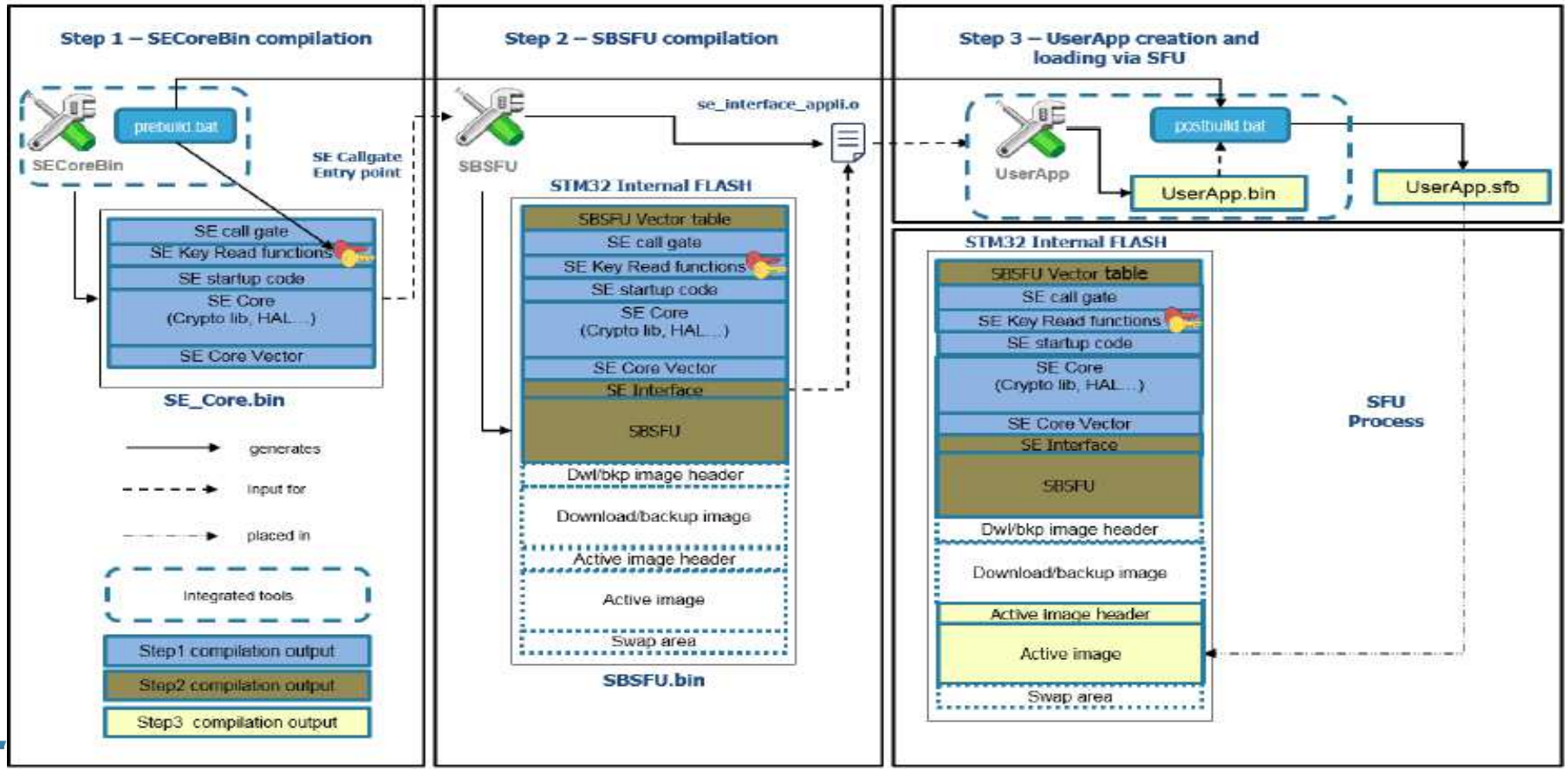


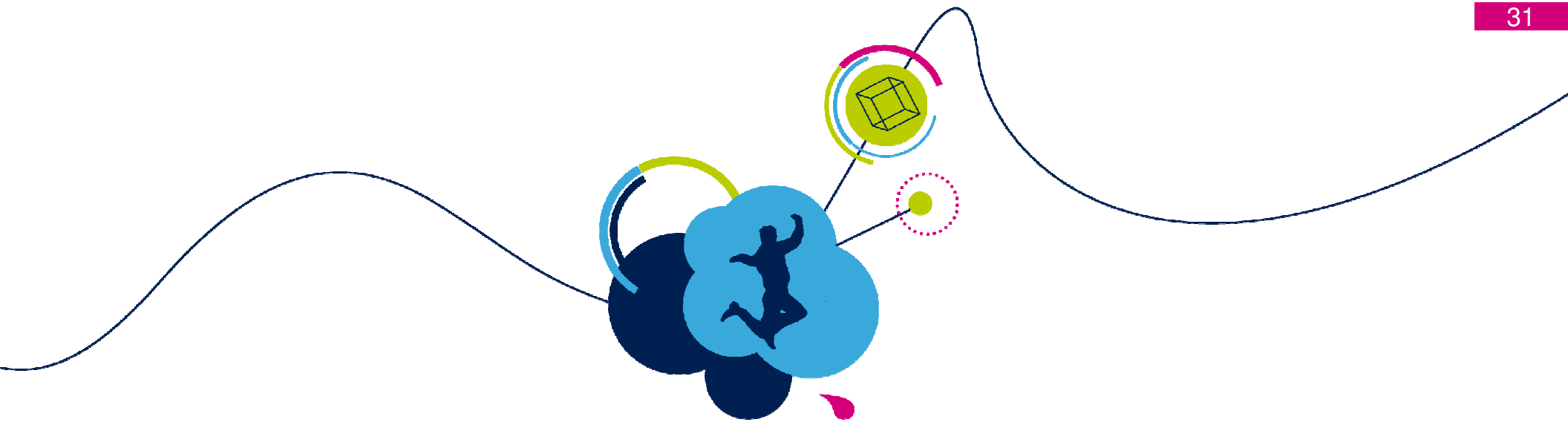

- 服务类别：
  - 固件下载(Ymodem)
    - 可在安全启动部分
    - 可在用户程序部分
  - 解密，安装，验证，执行
    - 安全启动部分
- 双镜像允许回退
  - Slot-0有一块执行区域
  - Slot-1有两块下载区域
    - 头部
    - 用户固件

life.augmented

# STM32 SBSFU映像生成







# MbedTLS

# TLS 的场景



- 身份认证
  - 单向认证
  - 双向认证
- 加密固件传输
- 加密消息传输

# TLS (HTTPS/SSL)

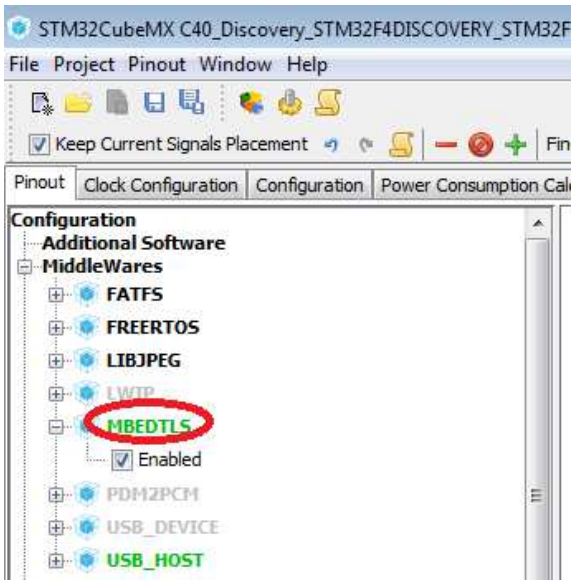


- Transport Layer Security (TLS) 基于TCP保证通讯安全的标准
  - 加解密功能
  - 协议实现
- DTLS 同样功能但基于UDP

# STM32上的MbedTLS



- MbedTLS是TLS的针对有限资源例如MCU的一种实现
  - 在PC上的一种实现是OpenSSL
- MbedTLS是CubeMX支持的中间件
  - X-CUBE-CRYPTOLIB 不提供TLS但提供加解密服务
  - MbedTLS包含加解密服务
- MbedTLS支持基于TCP的TLS和 UDP的DTLS



# MbedTLS的结构



# mbedTLS 的配置文件



- 高度可配置
  - 使用TLS
  - 使用证书解析
  - 使用哪些算法
- 可替换mbedTLS的加解密服务
- 配置文件名
  - config.h
  - 或者用户自定义

```
#if !defined(MBEDTLS_CONFIG_FILE)
#include "config.h"
#else
#include MBEDTLS_CONFIG_FILE
#endif
```

# medTLS的典型资源需求与技巧



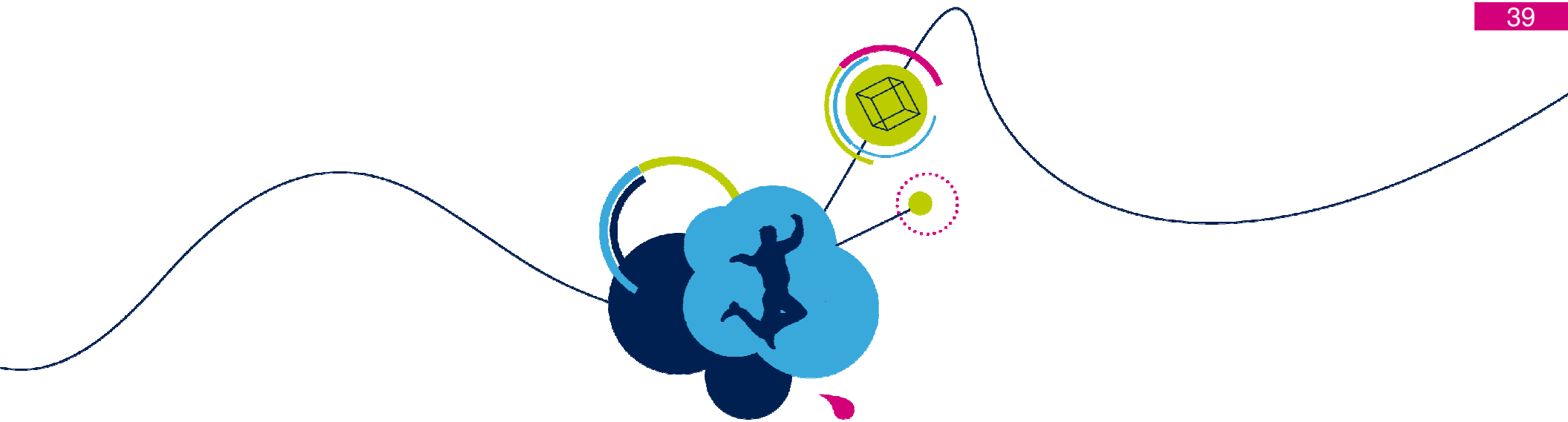
- 典型资源需要
  - Flash ~64K
  - RAM ~40K
- 技巧
  - ROM和RAM的平衡
    - #define MBEDTLS_AES_ROM_TABLES
  - 减小RAM
    - #define MBEDTLS_SSL_MAX_CONTENT_LEN 4*1024

# X.509证书解析示例



```
/*Init certificate structure*/
  mbedtls_x509_crt_init( &clicert );

  /*setup pem certificate as input*/
  ret = mbedtls_x509_crt_parse( &clicert, (const unsigned char *) xxx_crt,
                                sizeof(xxx_crt) );
  if( ret != 0 )
  {
    printf("failed\n");
    return( ret );
  }

  /*print out certificate information*/
  mbedtls_x509_crt_info( buf, sizeof( buf ) - 1, "", &clicert );
  printf( "%s", buf );
```

假定你有一个证书xxx_crt，需要使用mbedTLS解析其中的内容



# 读保护RDP

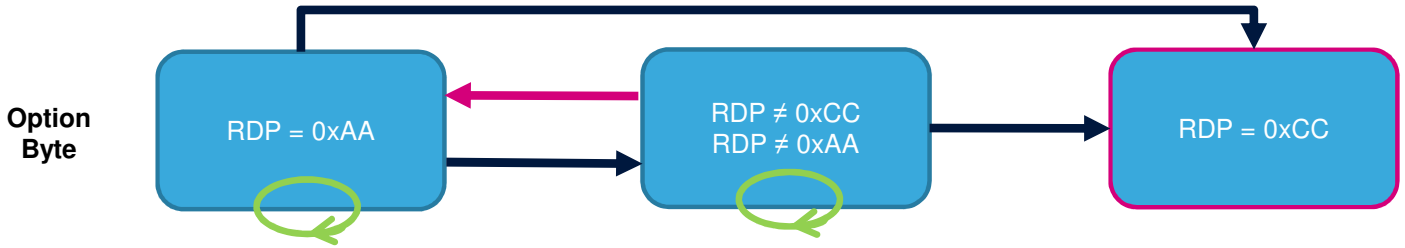
# 应用场景



- 固件读出保护
- 调试接口保护
- 启动位置保护
- 选项字节保护

# STM32读保护

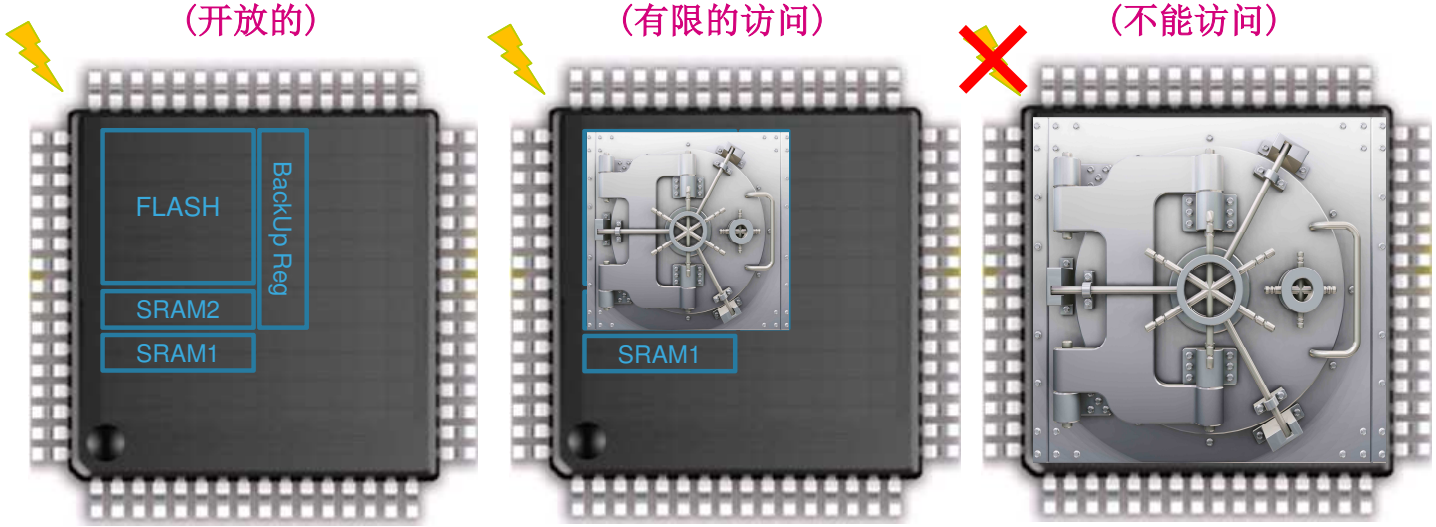





Level 0
(开放的)

Level 1
(有限的访问)

Level 2
(不能访问)




RDP2

- 仅能从用户 Flash启动
- Option bytes不可更改
- 调试端口不能连接

调试工具 / 从SRAM启动 / 从固化的系统存储区启动

life.augmented

# RDP级别的访问权限



| 区域 | RDP | 从用户Flash启动 | | | 调试模式或从SRAM启动<br>或从系统FLASH启动 | | |
|---|---|---|---|---|---|---|---|
| | | 读 | 写 | 擦除 | 读 | 写 | 擦除 |
| 用户FLASH | 1 | Yes | Yes | Yes | No | No | No |
| | 2 | Yes | Yes | Yes | No | No | No |
| 系统FLASH | 1 | Yes | No | No | Yes | No | No |
| | 2 | Yes | No | No | N/A | N/A | N/A |
| Option bytes | 1 | Yes | Yes | Yes | Yes | Yes | Yes |
| | 2 | Yes | No | No | N/A | N/A | N/A |
| 后备寄存器 | 1 | Yes | Yes | N/A | No | No | N/A |
| | 2 | Yes | Yes | N/A | No | No | N/A |
| OTP (F2/F4) | 1 | Yes | Yes | N/A | No | No | N/A |
| | 2 | Yes | Yes | N/A | No | No | N/A |
| SRAM2 | 1 | Yes | Yes | N/A | No | No | N/A |
| | 2 | Yes | Yes | N/A | No | No | N/A |

# RDP1状态下非Flash启动



✓寄存器

- 所访问到的寄存器内容只是系统复位值或复位后还能保留的值。

✓调试

- 任何调试的连接，将导致系统重启并挂起，

✓DMA

- 无法通过DMA转存Flash代码

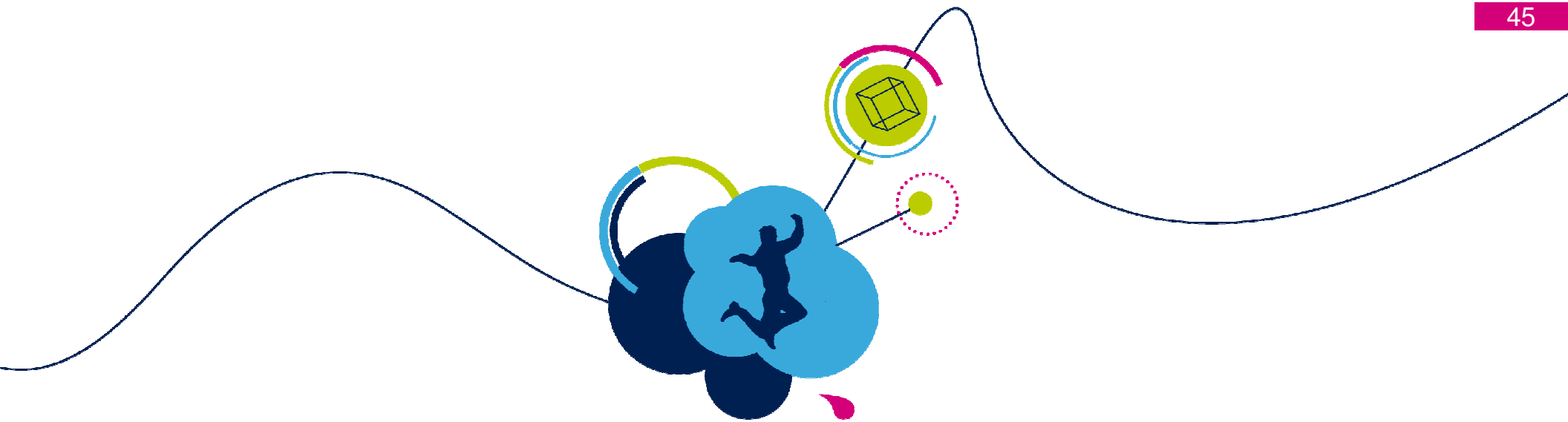
❑SRAM1

- ✓ 无法通过修改PC指针跳转到Flash
- ❑ 可以从SRAM1启动
- ❑ 可以得到SRAM1的不同时间点的快照
  - ❑ 在安全应用中，应尽量避免使用SRAM1作为堆栈或者其他敏感数据存放的区域
  - ❑ 应使用SRAM2或者采用RDP2

# Option bytes 注意事项



- 设置OBL_LAUNCH bit (bit 27) 可导致重启
- 安全选项应在出厂时完成
- 结合其他保护使Option byte不被误修改
- 推荐设置RDP到级别2
  - 但注意无法进行FA 分析



# 私有代码读保护PCROP

# 应用场景



- 客户n可以为STM32 MCU开发和销售专门的软件IP
  - 写入MCU
  - 设置PCROP保护

- 可防止恶意软件或者调试器对代码进行读出

- 客户n+1可以在它的应用程序中使用这些受PCROP保护的软件模块

- PCROP保护的Flash区域仅可执行。
  - 读/写/擦除都不被允许。

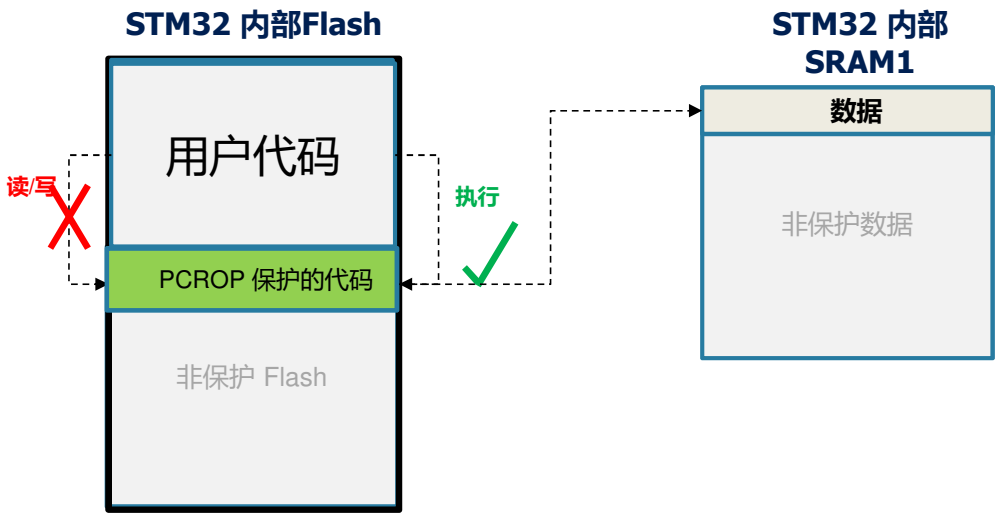
# PCROP



- 基于仅执行(Execute-only)机制的保护措施
  - 受保护代码仅能被执行
  - 受保护代码不能被任何读访问
  - 受保护代码不能被任何写访问
- 对芯片外部威胁，提供类似RDP的读保护
- 对芯片内部威胁，例如恶意软件或者不可信第三方，提供RDP**不具有的**读保护

# PCROP 扩展应用: 保护密钥

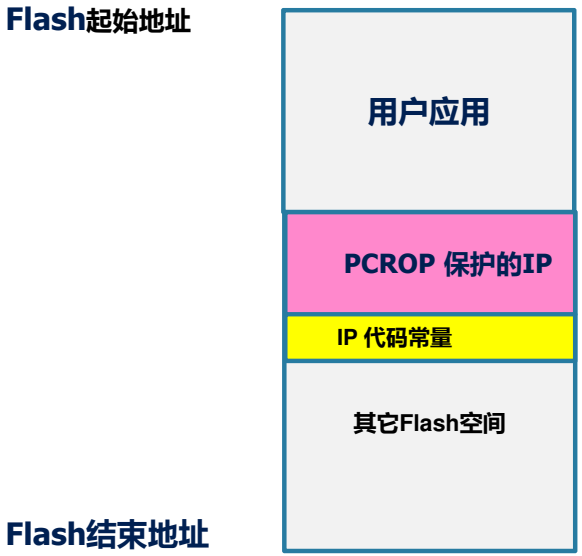





- PCROP不是保护数据的标准方法
  - 数据必须内嵌在指令里
- 用户应用必须调用PCROP所保护的代码将数据放到SRAM中去

# PCROP Flash布局

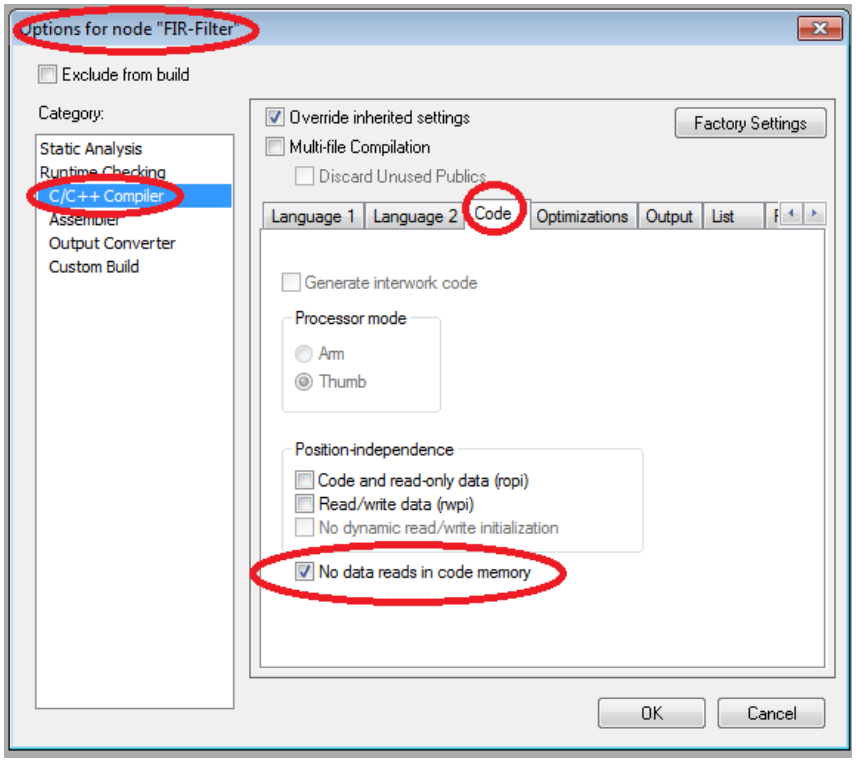


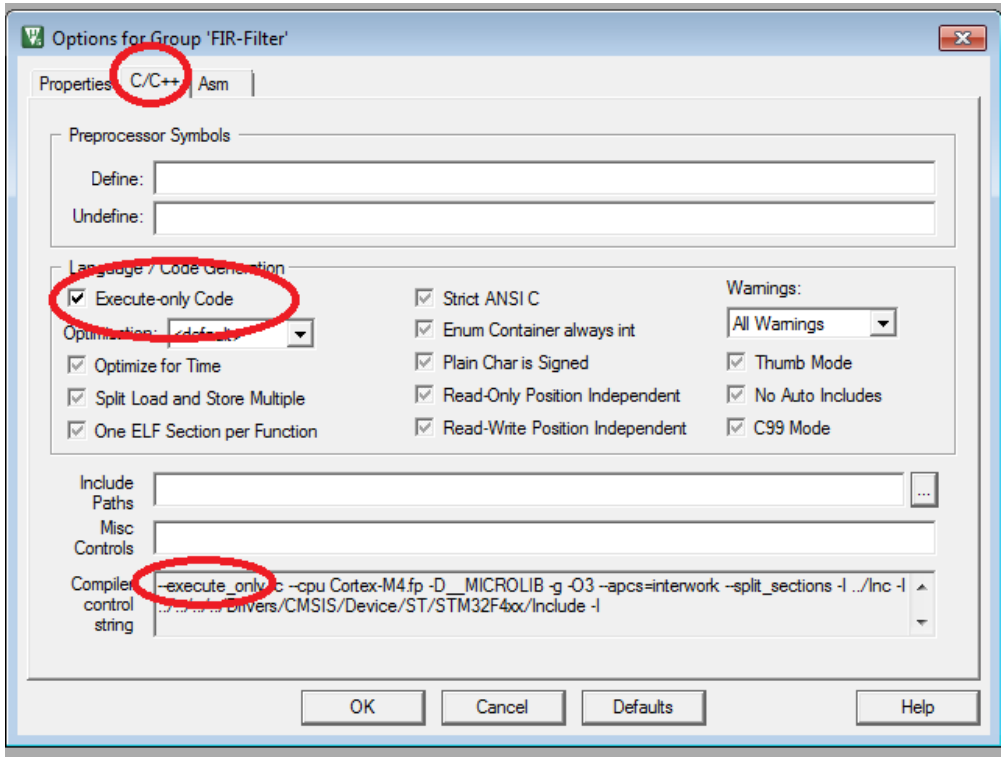



- PCROP所保护的区域仅可执行，任何数据必须放在其他区域.
  - 去掉文字库
    - IAR 选项
      - No data read from code
    - Keil 选项
      - Execute only
- IP代码的常量要放在PCROP保护的区域之外

# IAR和Keil的去文字库配置






life.augmented

# PCROP的设置



- 编译被保护的代码应去掉文字库
- 通过修改Option Bytes设置PCROP
  - 静态设置
  - 重启后依然有效
- 可基于Flash扇区Sector或者起始地址设置
  - 不同STM32系列方式略有不同
- PCROP同时提供写保护(Execute-only)。
  - PCROP的设置与写保护WRP不可同时(STM32F4)
- PCROP设置后仅可在RDP级别从1到0可清除，同时会引发整片擦除。

# PCROP 限制的对策



- PCROP保护区域不可修改=〉无法升级
- 对策
  - 可预留大的空间
  - 逐步增加PCROP的保护空间

# 双bank的安全考虑

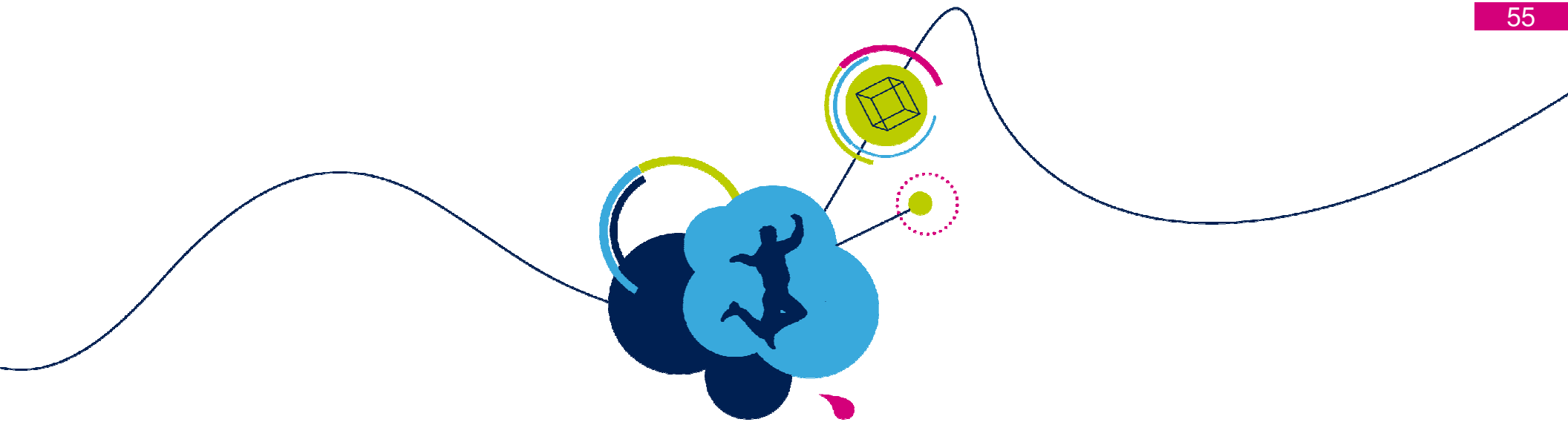


- PCROP可以对每一个Bank进行设置
- 如果Bank被交换，PCROP应设置在相应的Bank上

# PCROP的注意事项



- PCROP仅保护代码不能读写
- PCROP保护的代码可单步调试
- PCROP不能保护代码运行的结果



# 写保护WRP

# 典型应用



- 防错误写
- 生成仅可读的安全启动保护区

# 写保护WRP



- 保护Flash里的内容不被内外攻击写入
- 通过修改Option Bytes设置WRP
  - 静态设置
  - 重启后依然有效

# 写保护的设置与清除

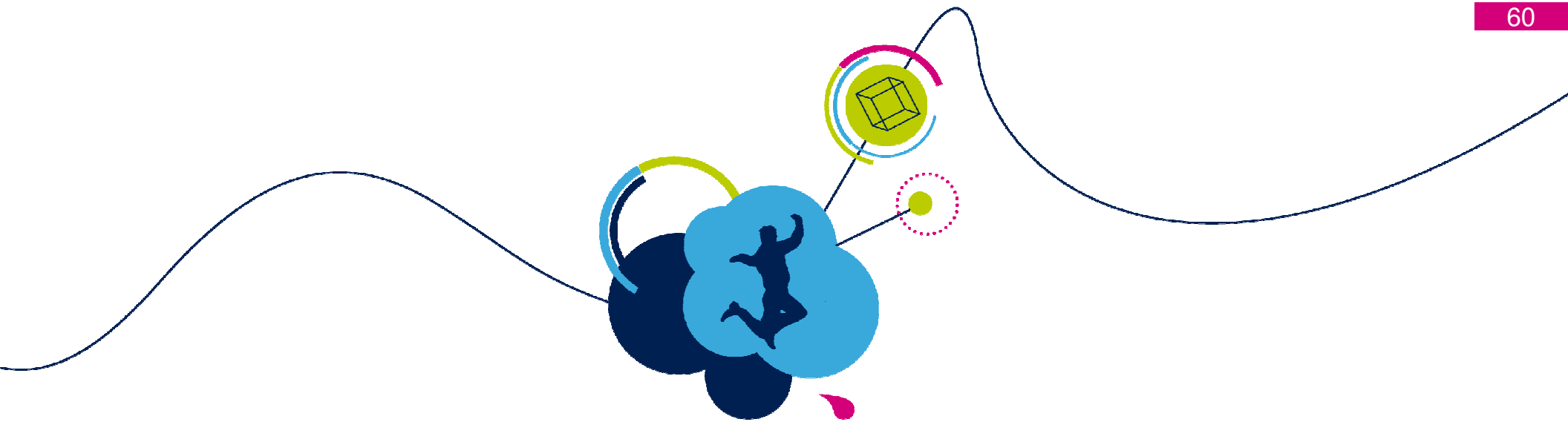


- 清除写保护不会引发块擦除
  - 写保护要结合其他保护使用
  - RDP的写保护选项字节还可以通过IAP修改
- 每Bank两块写保护区域，颗粒可至每页
  - 连续设置

# SRAM2的写保护



- SRAM2的写保护是通过寄存器设置不是WRP
- SRAM2的写保护一旦设置，重启才能清除



# 内存保护单元MPU

# 典型应用



- 辅助代码调试
  - 可将可疑区域权限修改成不可读写
- 防止远程缓冲区溢出攻击
  - 使用MPU将RAM设置成不可执行
- 防恶意软件
  - 代码隔离
    - 仅能执行，不能读写
  - 数据隔离
    - 仅能读/写，不能执行
  - 结合特权模式与用户模式

# MPU



- 对内存设置内存类型，属性和访问规则
  - 内存是指Cortex之外的所有内存地址
  - 无法保护MPU自身
- 运行时动态设置，不同于RDP, WRP和PCROP

# MPU的设置



| 位 | 名字 | 描述 |
|---|---|---|
| 28 | XN | 从不执行 |
| 26 : 24 | AP | 数据访问许可(RO, RW 或者无权限) |
| 21 : 19 | TEX | 类型扩展字段 |
| 18 | S | 共享 |
| 17 | C | 可缓存 |
| 16 | B | 可缓冲 |
| 15 :8 | SRD | 禁止子块 |
| 5 :1 | SIZE | 指定MPU保护区域的大小 |

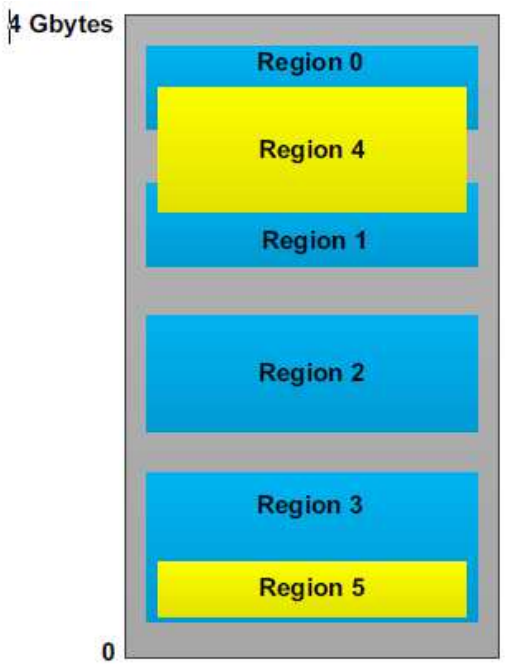
*Cache相关属性仅对有Cache系列有意义

设置内存属性的**MPU_RASR**寄存器字段

# 块与子块



- 每个Region的基地址必须与Region大小的倍数对齐
  - 64k的Region基地址低位只能是0x10000,0x20000等
- 每个Region可平分为8个子块
  - 可分Region的大小最小为256字节
- 优先级
  - 优先级别数字越大级别越高
    - 0最低，7最高
  - 高优先级属性覆盖低优优先级
  - 高优先级某子块若被禁止，则使用下一个低优先级的属性



# MPU的设置

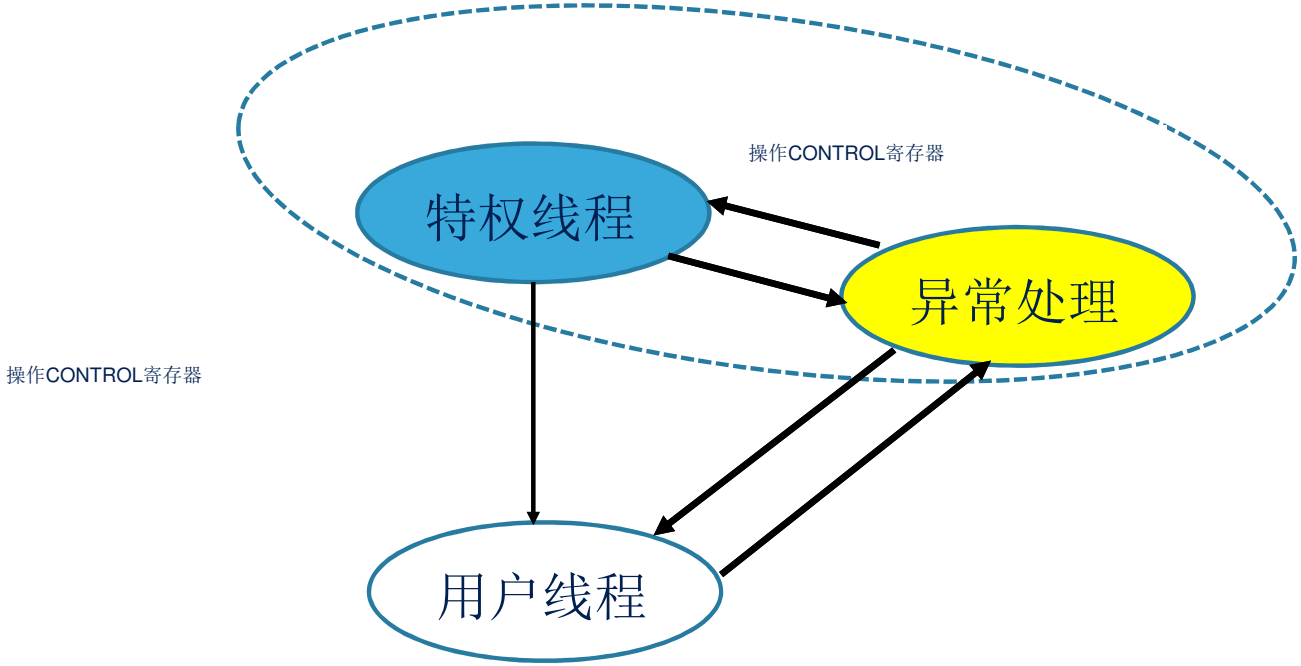


| AP[2 :0] | 特权模式 | 用户模式 | 描述 |
|---|---|---|---|
| 000 | 不可访问 | 不可访问 | 所有的访问产生一个内存管理异常 |
| 001 | RW | 不可访问 | 仅可从特权模式访问 |
| 010 | RW | RO | 在用户模式下的写访问会产生内存管理异常 |
| 011 | RW | RW | 完全访问 |
| 100 | 不可预知 | 不可预知 | 保留 |
| 101 | RO | 不可访问 | 仅可在特权模式下进行读访问 |
| 110 | RO | RO | 特权和用户模式只读 |
| 111 | RO | RO | 特权和用户模式只读 |

**MPU_RASR寄存器的AP字段指定读写访问规则**

# 特权模式与用户模式





# 典型MPU配置

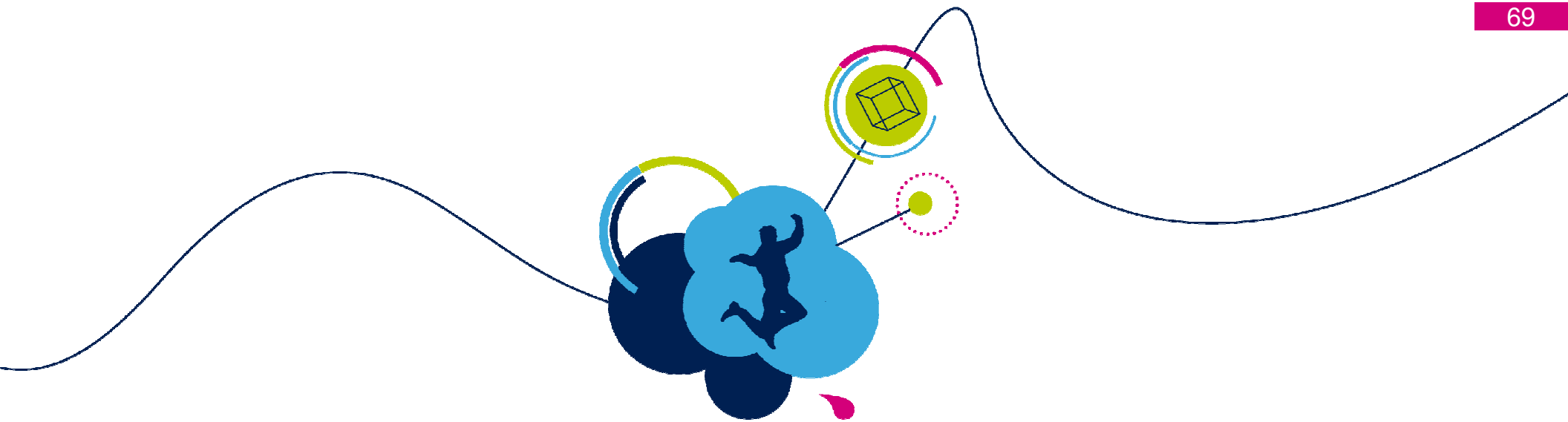


```
#define SFU_PROTECT_MPU_AREA_USER_EXEC
MPU_INSTRUCTION_ACCESS_DISABLE

#define SFU_PROTECT_MPU_AREA_PERIPH_EXEC
MPU_INSTRUCTION_ACCESS_DISABLE
```

# 典型MPU配置



```
static SFU_MPU_InitTypeDef MpuAreas[] =
{
 {MPU_REGION_NUMBER0,,,,    SFU_PROTECT_MPU_AREA_USER_PERM,
SFU_PROTECT_MPU_AREA_USER_EXEC, SFU_PROTECT_MPU_AREA_USER_SREG},
 {MPU_REGION_NUMBER1,,,   SFU_PROTECT_MPU_AREA_SFUEN_PERM,
SFU_PROTECT_MPU_AREA_SFUEN_EXEC, SFU_PROTECT_MPU_AREA_SFUEN_SREG_0},
 {MPU_REGION_NUMBER2,,,   SFU_PROTECT_MPU_AREA_SFUEN_PERM,
SFU_PROTECT_MPU_AREA_SFUEN_EXEC, SFU_PROTECT_MPU_AREA_SFUEN_SREG_1},
 {MPU_REGION_NUMBER3,,,    SFU_PROTECT_MPU_AREA_VECT_PERM,
SFU_PROTECT_MPU_AREA_VECT_EXEC, SFU_PROTECT_MPU_AREA_VECT_SREG},
 {MPU_REGION_NUMBER4,,, SFU_PROTECT_MPU_AREA_OB_BANK1_PERM,
SFU_PROTECT_MPU_AREA_OB_BANK1_EXEC, SFU_PROTECT_MPU_AREA_OB_BANK1_SREG},
 {MPU_REGION_NUMBER5, ,,    SFU_PROTECT_MPU_AREA_SRAM_PERM,
SFU_PROTECT_MPU_AREA_SRAM_EXEC, SFU_PROTECT_MPU_AREA_SRAM_SREG},
 {MPU_REGION_NUMBER6,,,   SFU_PROTECT_MPU_AREA_PERIPH_PERM,
SFU_PROTECT_MPU_AREA_PERIPH_EXEC, SFU_PROTECT_MPU_AREA_PERIPH_SREG}
};
```



# 防火墙Firewall

# 典型应用



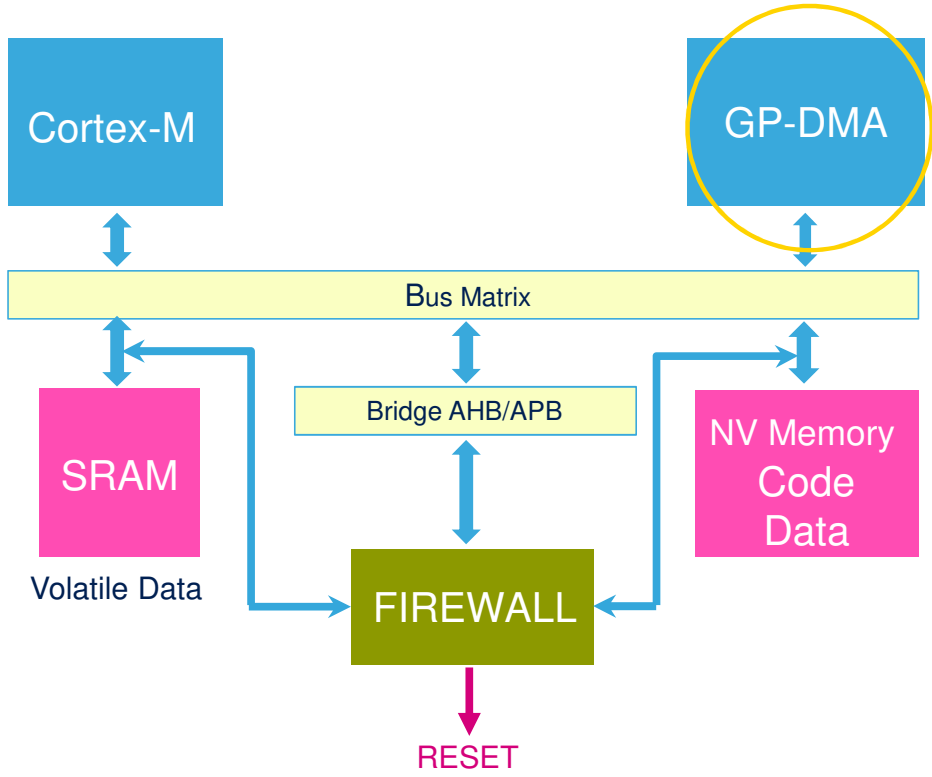
- 防内部恶意软件
  - 可信代码与可疑代码的空间隔离
  - 可信执行与可疑执行的时间隔离
  - 代码和数据的隔离
    - SRAM里的数据可被配置为不可执行

# STM32 Firewall



- 创建可信区域，与其他代码隔离
- 运行时保护安全敏感的IP与操作
- 单一调用门，其他非调用门访问会触发系统重起

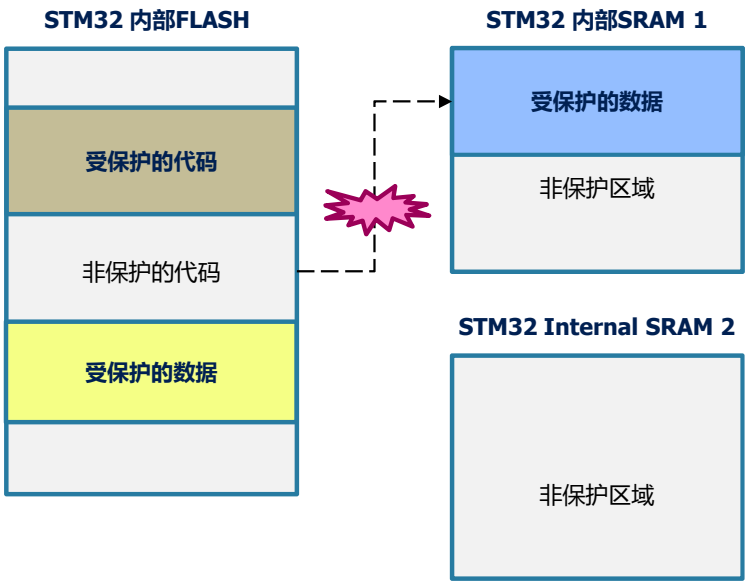
DMA和中断也不可跳过调用门




life.augmented

# STM32 Firewall







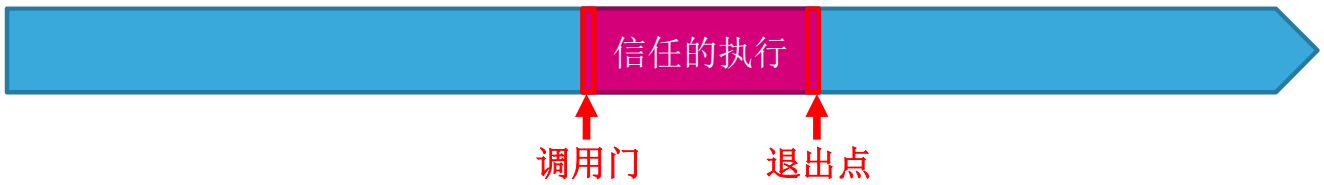
- Firewall 监测可信区域
  - **代码 (FLASH or SRAM 1)**
    - **指令提取和数据读取仅可在Firewall 开状态时允许**
  - **数据(FLASH)**
    - **安全敏感的常量(例如加解密的密钥)**
  - **数据(SRAM 1)**
    - **包括受保护代码需要的可变数据**
- 任何不符合保护设置和Firewall状态的访问将导致重启

life.augmented

# 调用门与代码执行

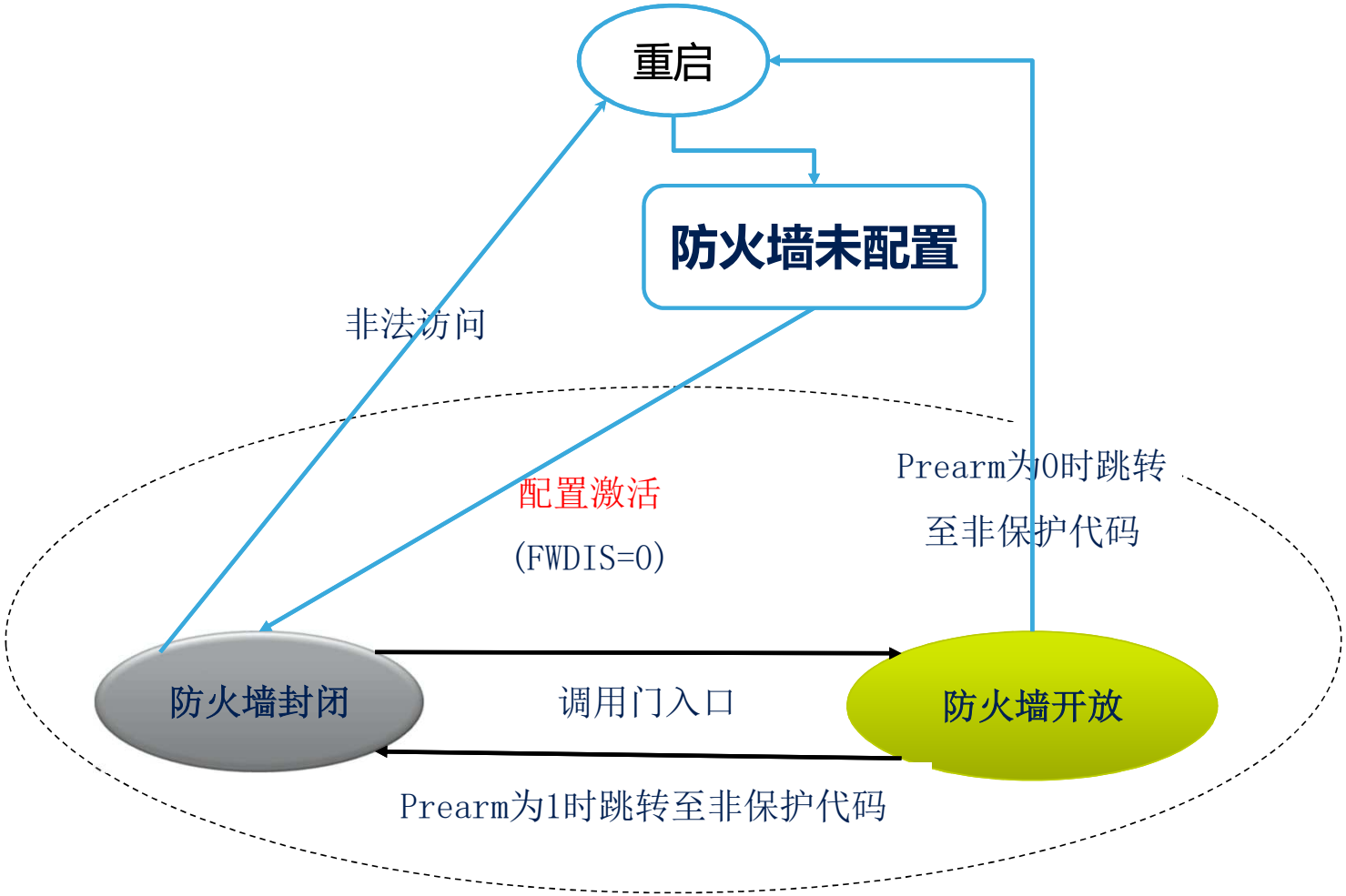


- 防火墙的调用门位于受保护区开始地址的前三个32位
- 代码执行通过调用门进入
- 代码通过特定标志退出保护执行
  - 设置Prearm bit，就可以退出，所以也可以有多个退出点



# Firewall状态转换





# Firewall特点



- 进入Firewall前建议关闭中断
  - 参考SBSFU实现伪VTOR
  - 避免中断导致非法访问
- DMA不能访问受保护区域
- 上电周期内一直有效

# Firewall代码调试



- 调用门打开时可调试
- 调试方法
  - 若直接设置在调用门指针step into会引发Reset
  - 将断点设置在调用门函数内则可单步调试

# Firewall可选配置

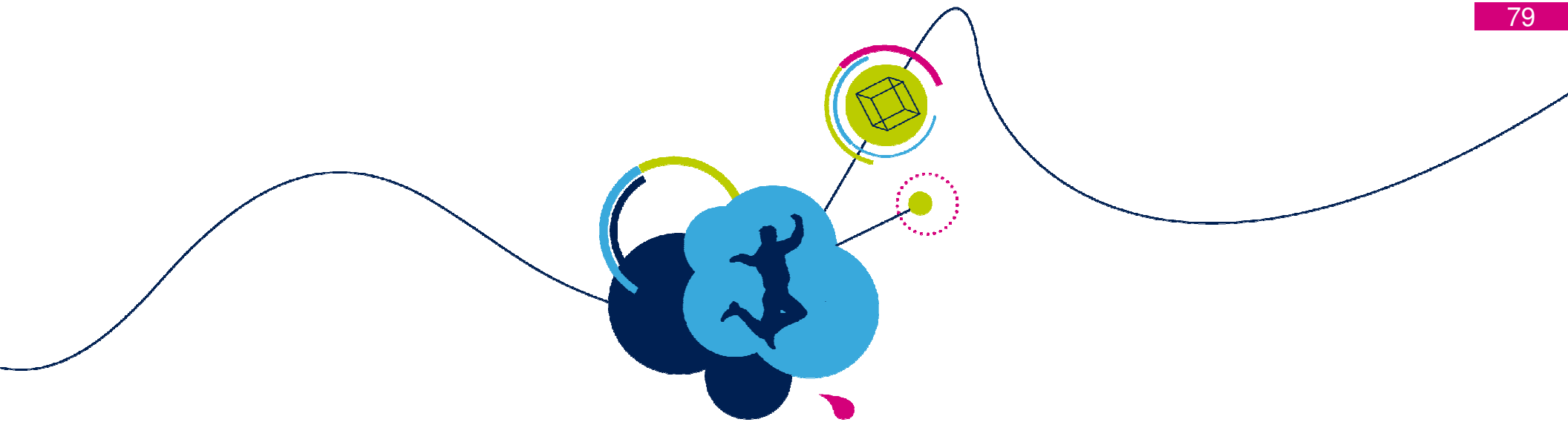


- 易失数据段共享或者非共享
- 易失数据段执行或者非执行
- 三个保护段可选择配置

# 典型防火墙配置



```
FWALL_InitStruct.CodeSegmentStartAddress      = SFU_PROTECT_FWALL_CODE_ADDR_START;

FWALL_InitStruct.CodeSegmentLength            = SFU_PROTECT_FWALL_CODE_SIZE;

FWALL_InitStruct.NonVDataSegmentStartAddress    = SFU_PROTECT_FWALL_NVDATA_ADDR_START;

FWALL_InitStruct.NonVDataSegmentLength          = SFU_PROTECT_FWALL_NVDATA_SIZE;

FWALL_InitStruct.VDataSegmentStartAddress     = SFU_PROTECT_FWALL_VDATA_ADDR_START;

FWALL_InitStruct.VDataSegmentLength           = SFU_PROTECT_FWALL_VDATA_SIZE;

FWALL_InitStruct.VolatileDataExecution        = FIREWALL_VOLATILEDATA_NOT_EXECUTABLE;

FWALL_InitStruct.VolatileDataShared           = FIREWALL_VOLATILEDATA_NOT_SHARED;
```



# Secure User Memory

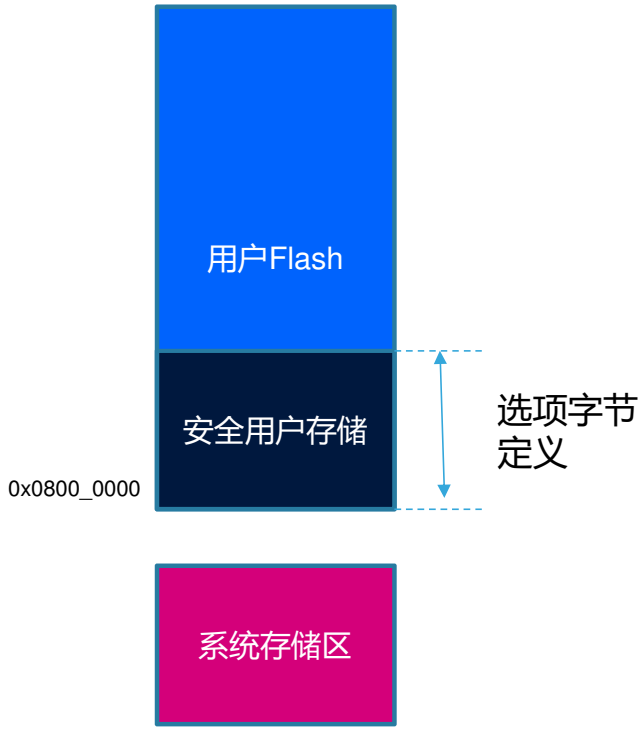
# 典型应用



- 防内部恶意软件
  - 可信代码与可疑代码的空间隔离
  - 可信代码与可疑代码在时间上不共存

# Secure User Memory



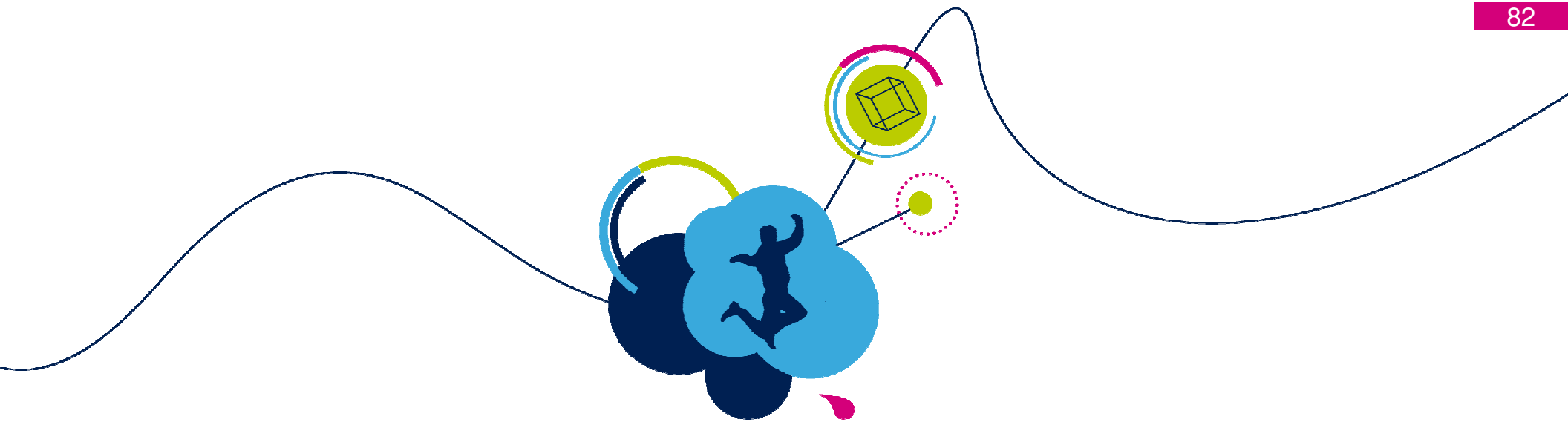



- UBE单一启动入口

通过设置选项字节，开机必须从安全用户存储区执行，不能跳过

- 固件隔离

安全用户存储区执行完毕，通过设置寄存器，后续的代码不能访问该安全用户存储区



# Unique ID

# STM32 96位Unique ID

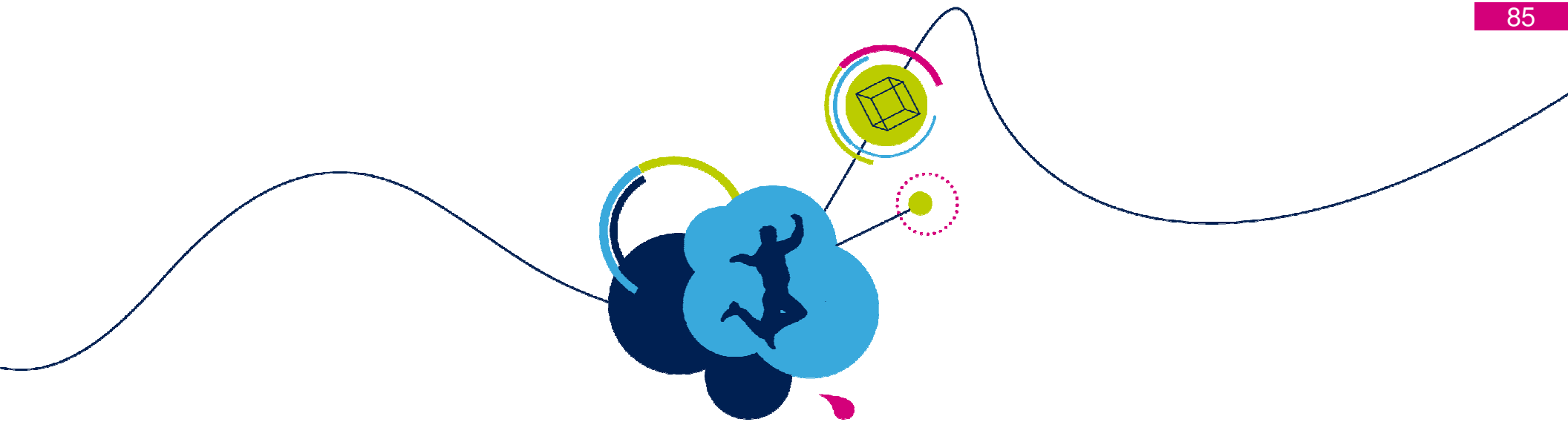


- ST工厂设置
  - 唯一
  - 在多年内不会重复
- Unique ID可用于多个方面
  - 使用算法产生唯一序列号
  - 在烧录以及密钥派生时结合加解密算法增强安全性
  - 安全启动时认证设备

# 获取Unique ID



- uint32_t HAL_GetUIDw0(void);
- uint32_t HAL_GetUIDw1(void);
- uint32_t HAL_GetUIDw2(void);

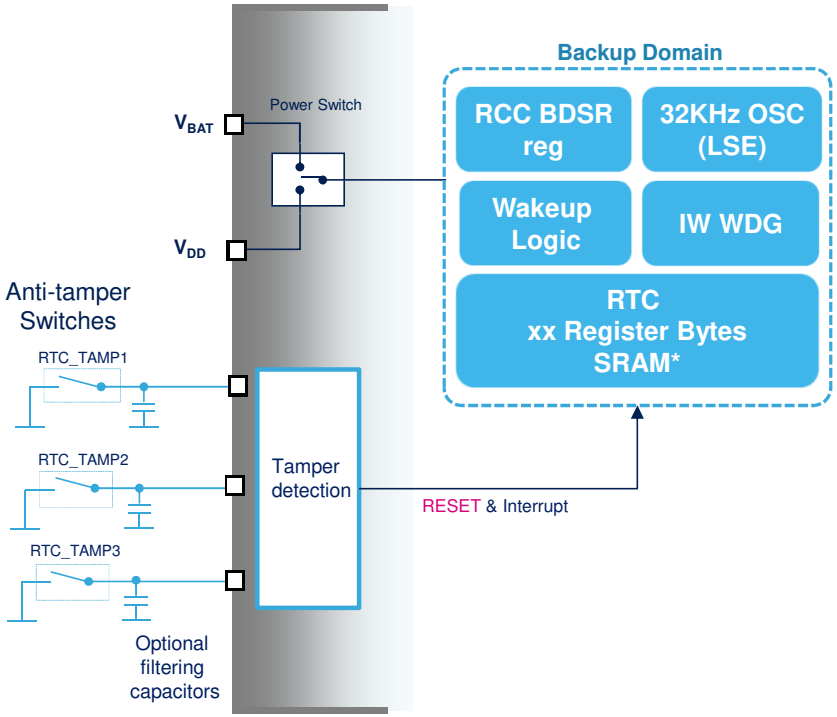


# Anti-tamper

# 后备域
# Backup Domain



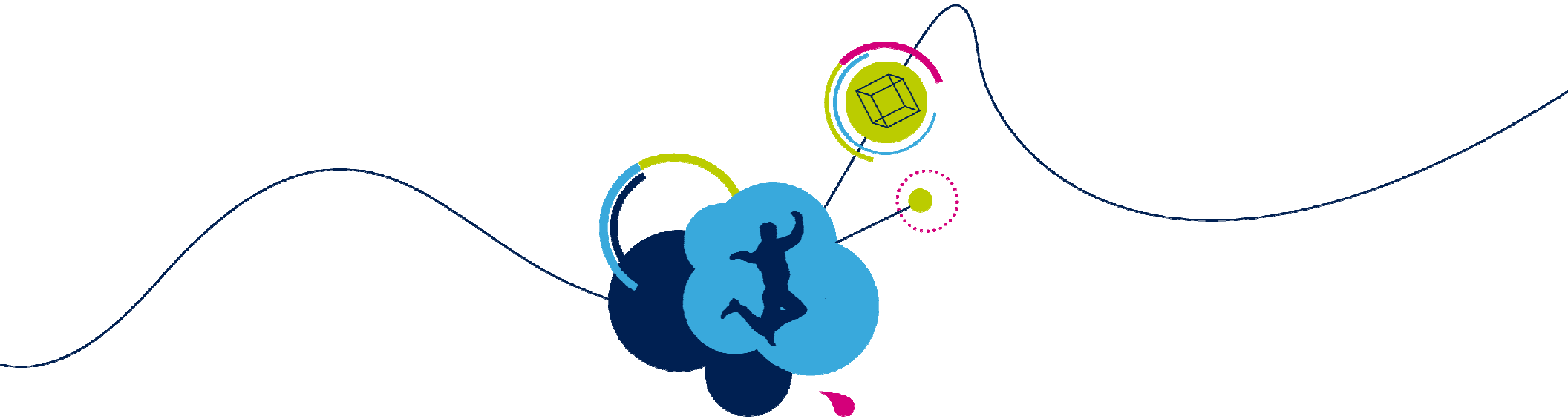
- 后备域包括:
  - RTC
  - xx寄存器字节，后备SRAM
  - 为RTC准备的独立 32kHz晶振
- 一旦在管脚上检测到入侵事件
  - 在两个系统时钟内重置所有RTC后备寄存器(SRAM)
  - 产生入侵时间戳事件



# 扩展防篡改



- 扩展防篡改功能
  - 模式控制Pattern Control (定时器)
    - 外部连接I/O口，组成模式输入输出对
  - 电压控制
    - DAC 输出 / ADC 输入 + ADC 看门狗
  - 温度防篡改功能
    - 使用内部温度传感器
  - 低压/过压篡改
    - STM32供电电压检测功能



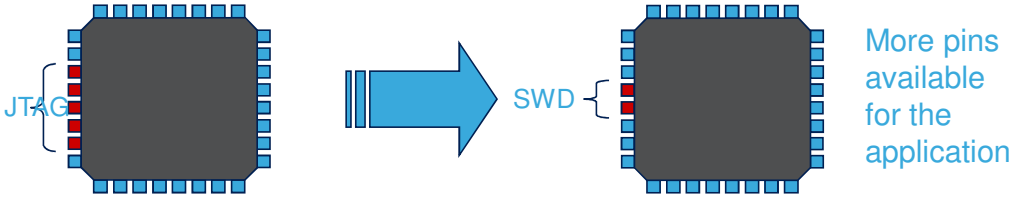
# 调试保护

life.augmented

# 调试端口

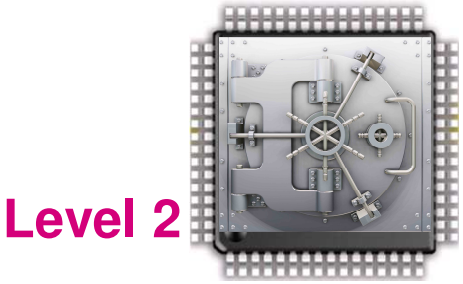


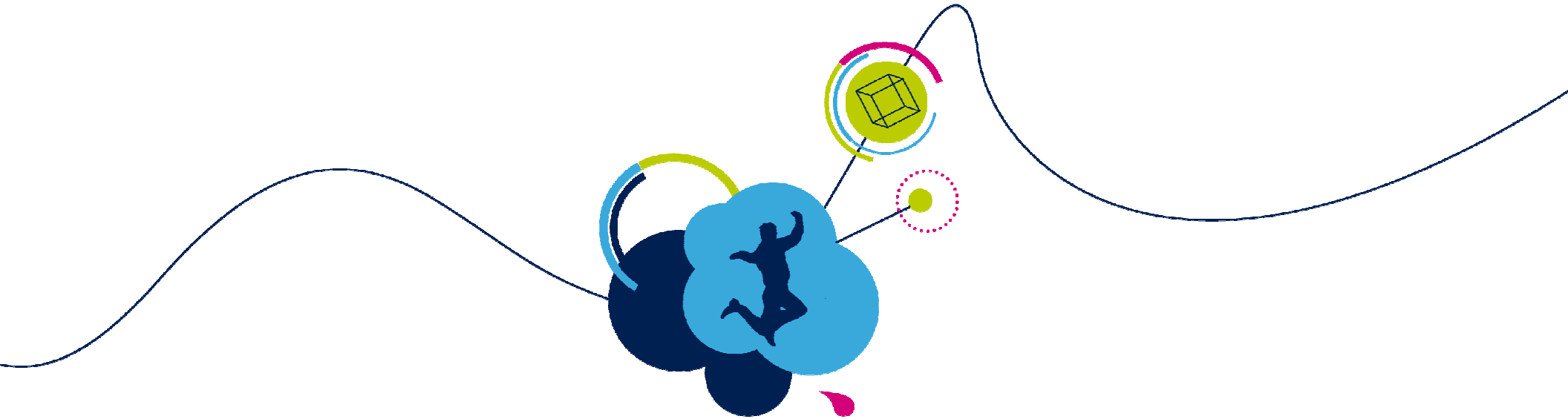
- JTAG之类的端口给与黑客最方便的通路
- 保护措施
  - 软件关闭JTAG口
  - 可移除调试接头，以及板上的调试点
  - 从代码移除任何调试程序
    - UART, SPI, I2C, USB, etc
  - 规范对产品密钥的访问
    - 开发时使用测试密钥或者占位密钥




NOTE

- 调试端口可以关闭，则不要忘记





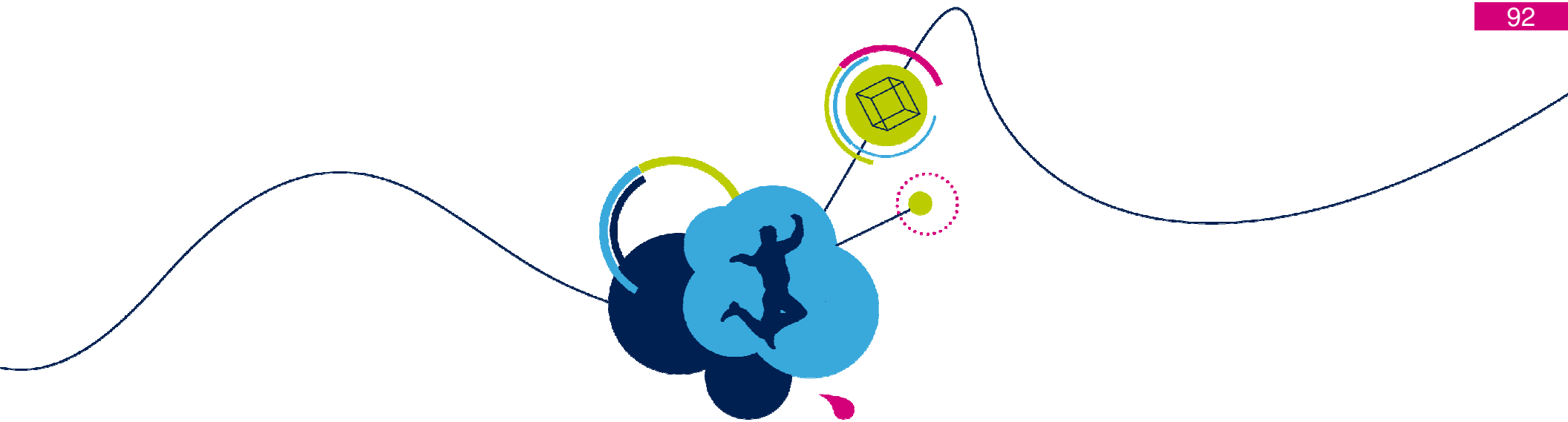
# 安全设置

life.augmented

# 安全设置



- Option bytes进行静态设置
  - 工具：ST-Link/STM32 Cube Programmer
    - 图形界面
    - 命令行
  - 代码
- Registers进行动态设置
  - 代码
    - 例子: SBSFU



# STM32 加密库CryptoLib以及硬件加速

life.augmented

# STM32 MCU 加密库



- 含有一系列密码算法实现，可运行在所有的STM32 MCU设备上
- 含有软件实现的算法库，也包含对一部分算法的硬件实现
- 默认以目标文件的方式提供

# STM32 MCU 加密库



- **STM32 加密库 V3.1.0分成两类**
  - STM32 固件加密库V3.1.0
    - 基于STM32 Cube架构
    - 支持所有STM32系列
    - 完全基于固件实现，无硬件加速
    - 以目标代码库方式提供，通过API调用
    - 两种优化方式：大小和速度
    - 开发工具：**EWARM, MDK-ARM and GCC (Atollic)***
  - STM32 硬件加速加密库V3.1.0
    - 基于STM32 Cube架构
    - 支持所有带硬件加速的STM32系列
    - 在固件实现算法上支持硬件加速。*完全由硬件支持的算法请参考Hal 驱动
    - 以目标代码库方式提供，通过API调用
    - 两种优化方式：大小和速度
    - 开发工具：**EWARM, MDK-ARM and GCC (Atollic)***

查阅软件包中 的文档可获得 实际的算法实测数据

- 性能大小
- 代码大小
- 内存需求

life.augmented

# STM32 X-CUBE-CryptoLib V3.1.0 所支持的算法



- STM32 X-CUBE-CryptoLib V3.1.0
  - DES, 3DES with ECB and CBC
  - AES with ECB, CBC, OFB, CCM, GCM, CMAC, KEY wrap, XTS
  - 哈希 : MD5, SHA-1, SHA-224, SHA-256, SHA-384, SHA-512
  - 其他： ARC4, ChaCha20, Poly1305, Chacha20-Poly1305
  - RSA PKCS#1v1.5标准的签名
  - ECC 密钥生成，点乘, 以及ECDSA + ED25519和Curve 25519
- **CAVP FIPS certified**
  - **TLS也支持加解密算法，但没有该认证**

# 密钥生成-Symmetric key



- 利用随机数直接生成
- 在安全的区域存放

# 密钥生成-RSA



- 选择两个素数P和Q
- 计算n= P*Q
- 计算Ø=(P-1)*(Q-1)
- 选择任意的e满足1<e< Ø且gcd(e, Ø)=1
- 计算d 满足 1<d< Ø , 且 e.d ≡ 1 (mod Ø)
- 返回(n, e,d)

# 密钥生成-ECC

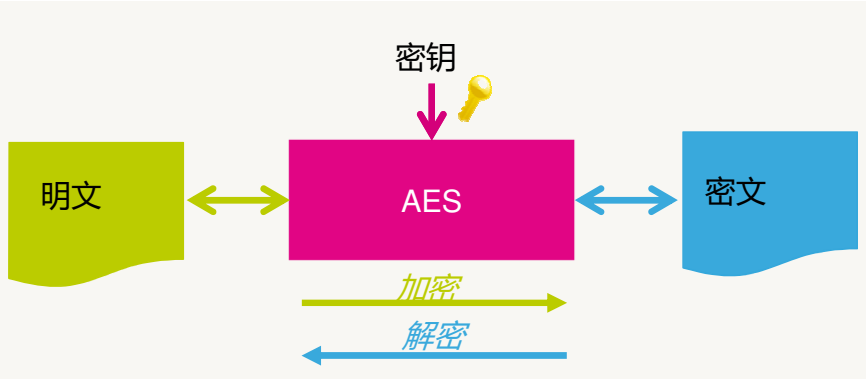


- 输入: 椭圆曲线的参数包括参考点G，曲线参数，模数
- 输出: 公钥 K 和私钥k
  - 选择随机数k Є [1, n-1]作为私钥
  - 计算 K=kG作为公钥
  - 返回 (K,k)

# STM32 AES 加速器



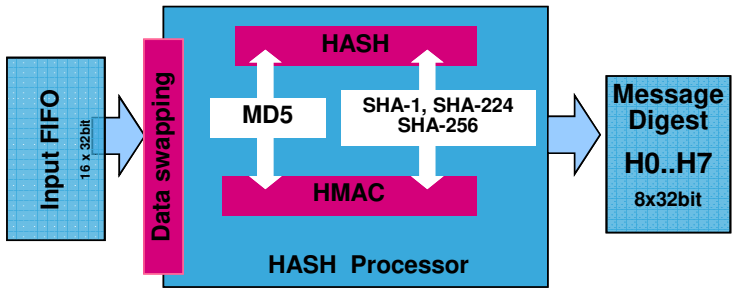
- 硬件实现的AES加速器，支持标准的加解密算法
  - 支持标准的操作模式和密钥大小
  - 支持字节交换
  - 支持DMA
  - 比软件实现更快
  - 减轻CPU负载
  - 更低功耗



# STM32 Hash加速器



- 对SHA1,SHA2 & MD5算法的硬件加速实现
  - 支持字节交换
  - 支持DMA
  - 比软件实现更快
  - 减轻CPU负载
  - 更低功耗




**符合:**

**FIPS Pub 180-2**

**安全**Hash标准(SHA-1*, **SHA-224, SHA-256)**

**IETF RFC 1321 (MD5*)**

life.augmented

# STM32 TRNG



- 基于物理噪声源的硬件随机数生成器

⇒ 广泛使用在安全体系和协议中

- 密钥生成
- 认证的质询(Challenges)
- 初始化向量(IV)
- 随机填充
- 随机数(Nonce), 一次性密码,…
- 数字签名中的随机值
- 侧信道以及错误攻击的保护措施
- 其他

# STM32软硬件安全技术



- STM32软硬件安全技术
  - STM32安全技术的大众市场目标
  - 理解安全启动的原则与实现的多样性
    - 使用STM32 SBSFU
    - 直接使用STM32硬件安全技术
  - 理解STM32 SBSFU的整体结构
  - 理解内外防护的STM32硬件安全技术
    - RDP (基础性设置)，WRP，PCROP
    - Secure User Memory，MPU，Firewall
    - 其它(DAP, UBE，TrustZone)
  - 了解MbedTLS的基本原理与两类用途

- STM32软硬件安全技术
  - 了解STM32加密库资源以及与MbedTLS的区别
    - 支持的算法
    - 性能与内存统计
  - 了解STM32 加解密硬件模块资源极其优势
    - AES
    - Hash
    - TRNG
    - PKA