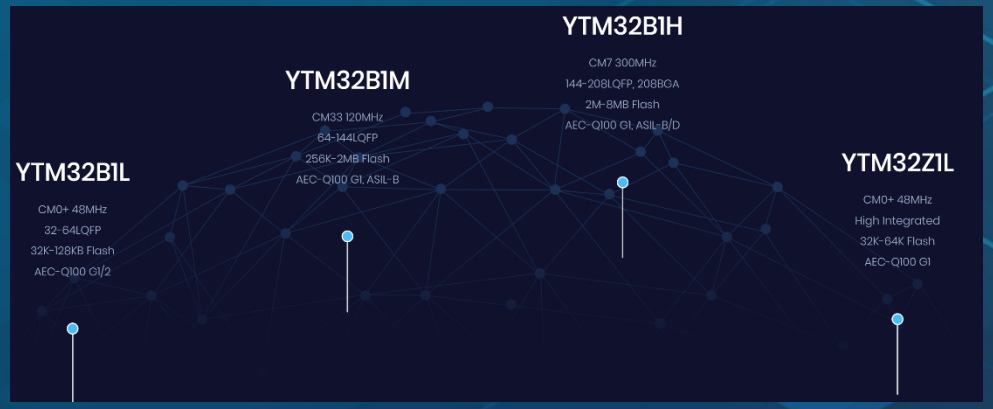
YUNTU AUTOMOTIVE SEMICONDUCTOR

真车规 高可靠 高性能

云途车规MCU信息安全(CyberSecurity)解决方案







Author: 云途AE team， 重庆RDC

Version： Rev 2.4 **Date**： 2025年4月18日

目录

CONTENTS

1. 云途车规MCU产品roadmap

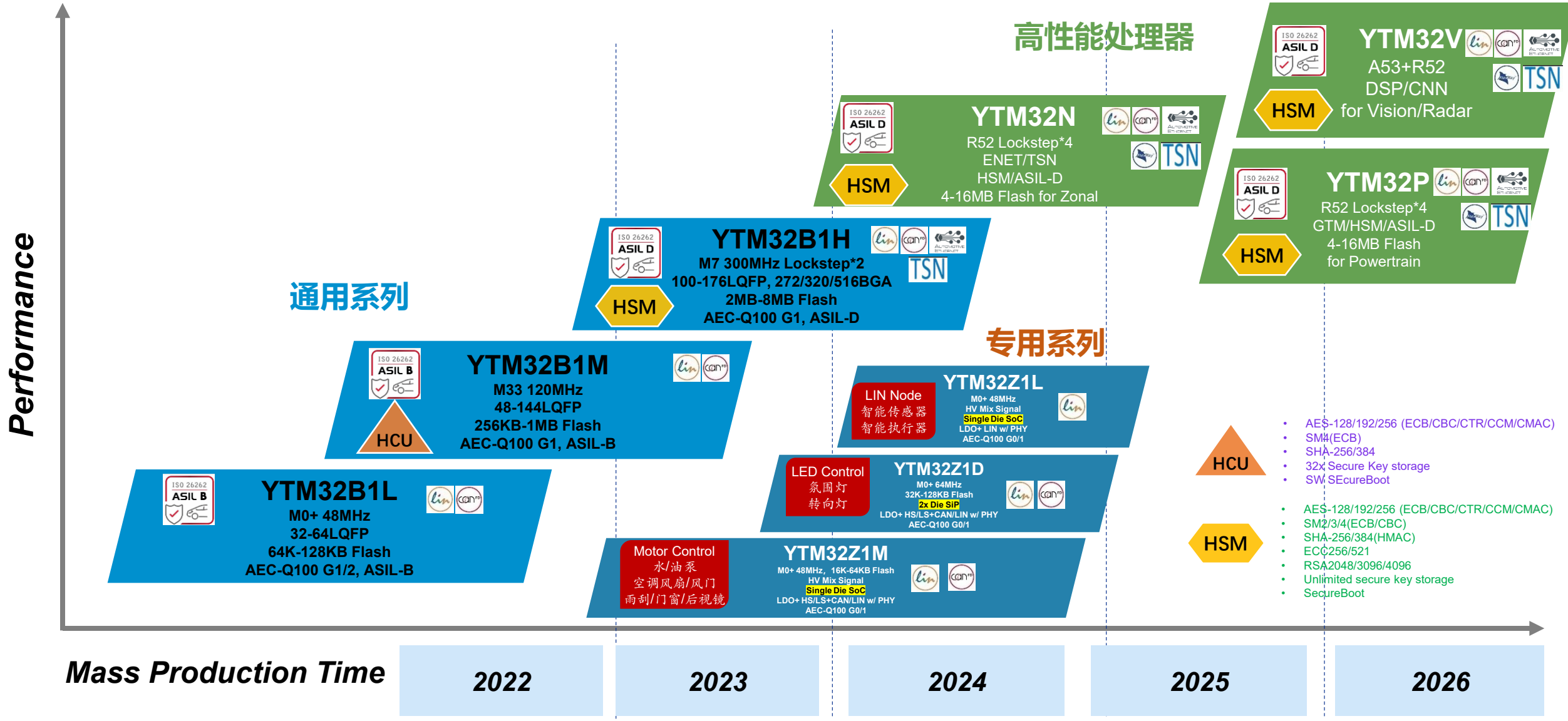
2. 汽车信息安全标准对比

3.云途车规MCU信息安全(Cybersecurity)支持

- HCU
- YUNTU HSM
- YUNTU eHSM
- 安全启动(SecureBoot)

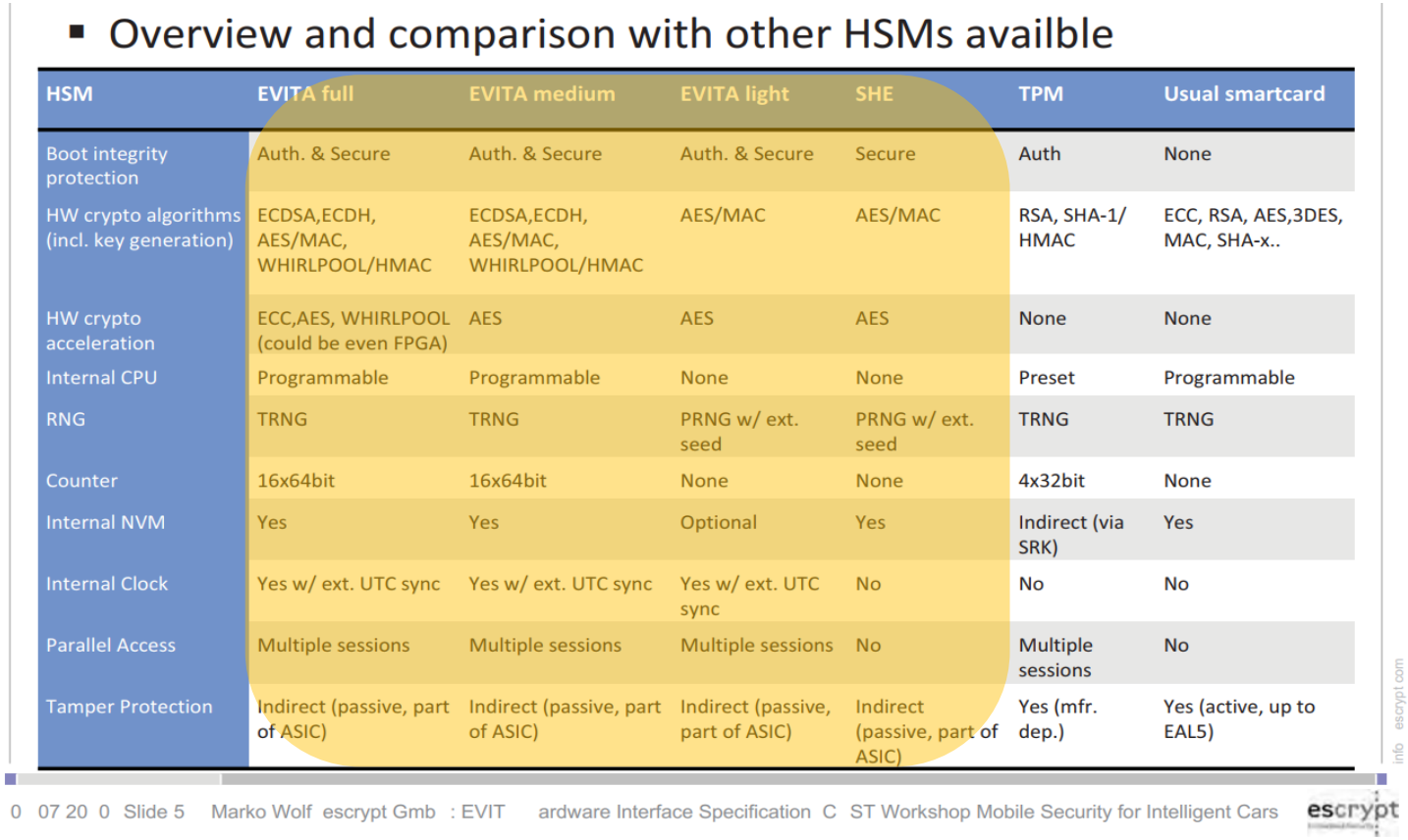
4. 总结

云途主控芯片Roadmap：新E/E架构下产品全系列覆盖



车规MCU芯片信息安全硬件架构及标准

- **硬件安全模块（HSM）**：HSM是车规MCU中用于提供安全保障的关键组件。它通常集成在MCU内部，负责加密/解密（对称加密算法）、签名/验证（非对称加密算法）、密钥管理、安全启动等功能。例如，EVITA项目中的HSM分为Light、Medium和Full三种变体，分别针对不同的应用场景提供不同程度的安全保护。此外，HSM还支持安全通信和安全存储功能，确保车辆内部及与外部系统之间的数据传输安全。



■ Overview and comparison with other HSMs availble

HSM	EVITA full	EVITA medium	EVITA light	SHE	TPM	Usual smartcard
Boot integrity protection	Auth. & Secure	Auth. & Secure	Auth. & Secure	Secure	Auth	None
HW crypto algorithms (incl. key generation)	ECDSA,ECDH, AES/MAC, WHIRLPOOL/HMAC	ECDSA,ECDH, AES/MAC, WHIRLPOOL/HMAC	AES/MAC	AES/MAC	RSA, SHA-1/ HMAC	ECC, RSA, AES,3DES, MAC, SHA-x..
HW crypto acceleration	ECC,AES, WHIRLPOOL (could be even FPGA)	AES	AES	AES	None	None
Internal CPU	Programmable	Programmable	None	None	Preset	Programmable
RNG	TRNG	TRNG	PRNG w/ ext. seed	PRNG w/ ext. seed	TRNG	TRNG
Counter	16x64bit	16x64bit	None	None	4x32bit	None
Internal NVM	Yes	Yes	Optional	Yes	Indirect (via SRK)	Yes
Internal Clock	Yes w/ ext. UTC sync	Yes w/ ext. UTC sync	Yes w/ ext. UTC sync	No	No	No
Parallel Access	Multiple sessions	Multiple sessions	Multiple sessions	No	Multiple sessions	No
Tamper Protection	Indirect (passive, part of ASIC)	Indirect (passive, part of ASIC)	Indirect (passive, part of ASIC)	Indirect (passive, part of ASIC)	Yes (mfr. dep.)	Yes (active, up to EAL5)

0 07 20 0 Slide 5 Marko Wolf escrypt Gmb : EVIT ardware Interface Specification C ST Workshop Mobile Security for Intelligent Cars escrypt

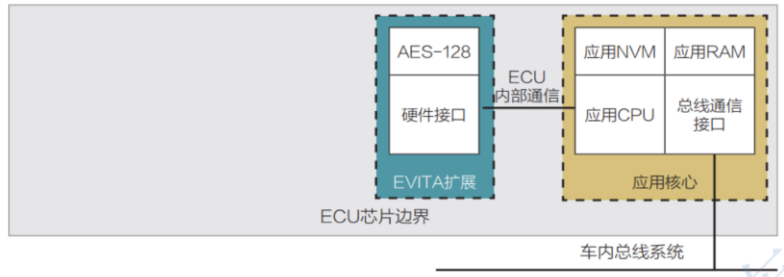


图 12 EVITA Light HSM 技术框架

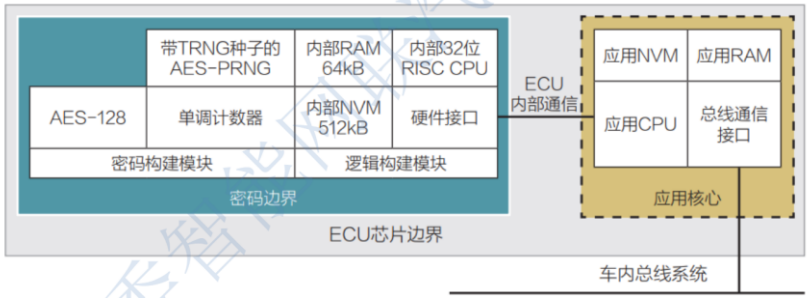


图 13 EVITA Medium HSM 技术框架

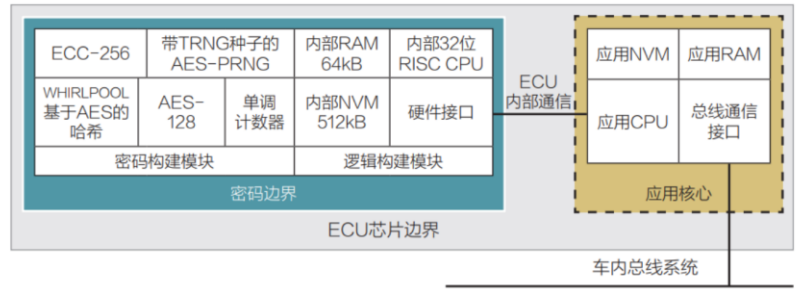
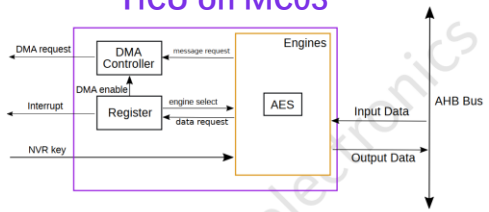


图 14 EVITA Full HSM 技术框架

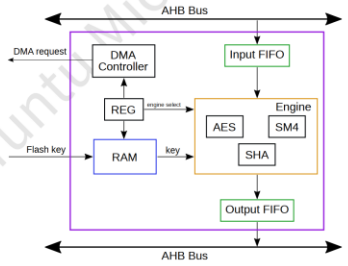
云途车规MCU信息安全(Cybersecurity)支持 – HCU

- **HCU Features**
  - Support 128-, 192- and 256-bit key length
  - Support both encryption and decryption
  - Support secure hardware key and flexible software key
  - Support several algorithm engines
    - AES (ECB, CBC, CTR, CCM, CMAC)
    - SM4 (ECB)
    - SHA (SHA-256, SHA-384)
  - All data input/output are in little-endian format
  - Support bit, byte and half-word data swapping
  - Size of input/output FIFO is up to 32*32 bits
  - Support DMA transport between chip memory and input/output FIFO
  - Support several interrupts
    - Operation done interrupt
    - Input FIFO empty interrupt
    - Output FIFO full interrupt
    - Input FIFO overflow interrupt
    - Output FIFO underflow interrupt
    - Input/output FIFO watermark interrupt
  - Clock gating strategy is applied for engine core when input/output FIFO is not ready
  - Add PKE with ECC-192/256/384 and RSA-2048/3072/4096 support(only for MD24)
- **满足SHE、Evita-Light和部分Evita-Medium功能要求**
- **低成本方案，在MC03、MD14、ME05、ME15和HA01上使用**

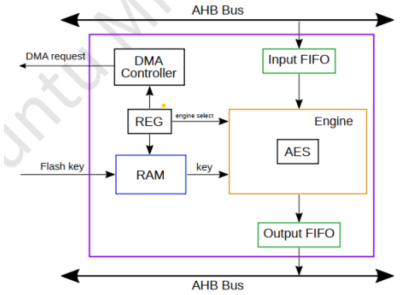
HCU on MC03



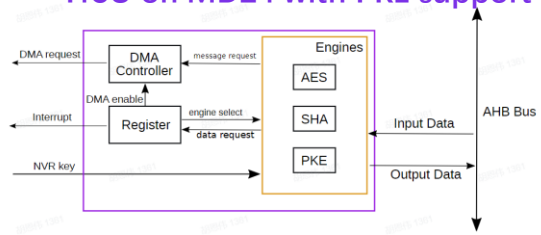
HCU on ME05, ME15 and HA01



HCU on MD14



HCU on MD24 with PKE support



HCU在不同系列MCU上实现的feature差异对比

	YTM32B1MC03	YTM32B1MD14	YTM32B1MD24	YTM32B1ME05	YTM32B1ME15	YTM32B1HA01
AES	AES-128 (ECB/CBC)	AES-128 (ECB/CBC)	AES-128 (ECB/CBC/CTR/CCM)	AES-128/192/256 (ECB/CBC/CTR/CCM)	AES-128/192/256 (ECB/CBC/CTR/CCM)	AES-128/192/256 (ECB/CBC/CTR/CCM)
CMAC	√	√	√	√	√	√
SM4(ECB only)	×	×	×	√	√	√
SHA	×	×	SHA256	SHA256/384	SHA256/384	SHA256/384
RSA	×	×	Up to RSA-4096	×	×	×
ECC	×	×	ECC-192/256/384	×	×	×
Secure Key Lot Number	32x 128-bit	32x 128-bit	32x 128-bit	32x 256-bit	32x 256-bit	32x 256-bit
IN/OUT FIFO	×	8*32 bits	32*32 bits	32*32 bits	32*32 bits	32*32 bits
Secure Boot	ROM_Boot**	User SW*	ROM_Boot**	User SW*	ROM_Boot**	ROM_Boot**

- *YUNTU provides the secure boot reference SW and AN. The secure boot firmware occupies the user P-Flash
- **The SecureBoot is implemented in ROMBoot firmware which is pre-programmed into secured ROM at chip FT, user can enable and configure it via the BVT located in P-Flash.

120MHz ME05 HCU性能测试结果

	1KB Message (us)	4KB Message (us)
AES-ECB	87.92	342.2
AES-CBC	87.65	340.34
AES-CTR	88.18	342.49
AES-CCM	104.49	357.28
AES-CMAC	75.68	272.37
SM4-ECB	86.75	339.52
SHA-256	65.54	247.92
SHA-384	65.86	248.25

200MHz HA01 HCU性能测试结果

	未开 Cache		开 Cache	
	1KB Message (us)	4KB Message (us)	1KB Message (us)	4KB Message (us)
AES-ECB	160.55	620.24	45.74	164.55
AES-CBC	174.34	673.28	42.76	162.06
AES-CTR	175.74	678.47	43.2	161.68
AES-CCM	197.57	695.32	72.82	240.72
AES-CMAC	129.11	452.15	36.03	121.29
SM4-ECB	155.17	598.69	42.24	161.42
SHA-256	124.88	467.23	32.92	118.88
SHA-384	124.51	467.7	26.5	93.43

云途车规MCU信息安全(Cybersecurity)支持 – YUNTU HSM

- **YUNTU HSM IP feature(available on YTM32B1HB02 and YTM32B1HB02 only)**

• Dedicated Cortex-M0+ CPU core for the HSM service management
• All crypto-related features are accelerated by the hardware crypto engine
• Powerful cross-core IPC hardware peripheral (CCU)
• Flexible key management, key partition, support three types of keys:
  – ROOT KEY
  – NVR KEY
  – RAM KEY

The size of Max keys depends on the format of key partitions.

• Security related features:
  – AES, includes CCM, max key size 256 bit
  – HMAC/CMAC, max key size 256 bit
  – RSA, max key size 4096 bit
  – ECC
  – SM4, max key size 128 bit
  – SM2
  – SHA2-256, SHA2-384
  – TRNG

• FEE integrated features. HSM FW manages all data flash regions, so HSM contains FEE init-, write-, read- and invalid-related features

- **云途提供免费HSM Firmware**
- **满足Evita-Full功能要求**

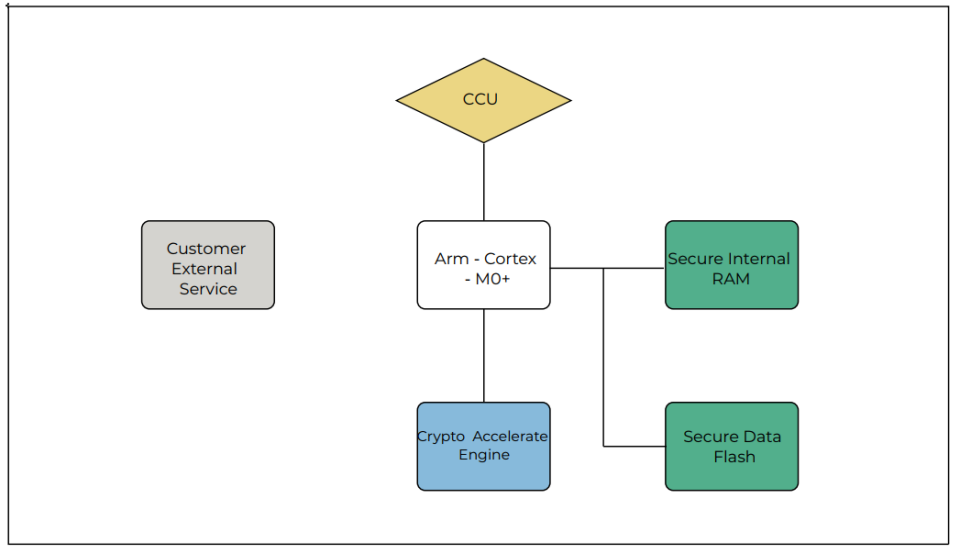
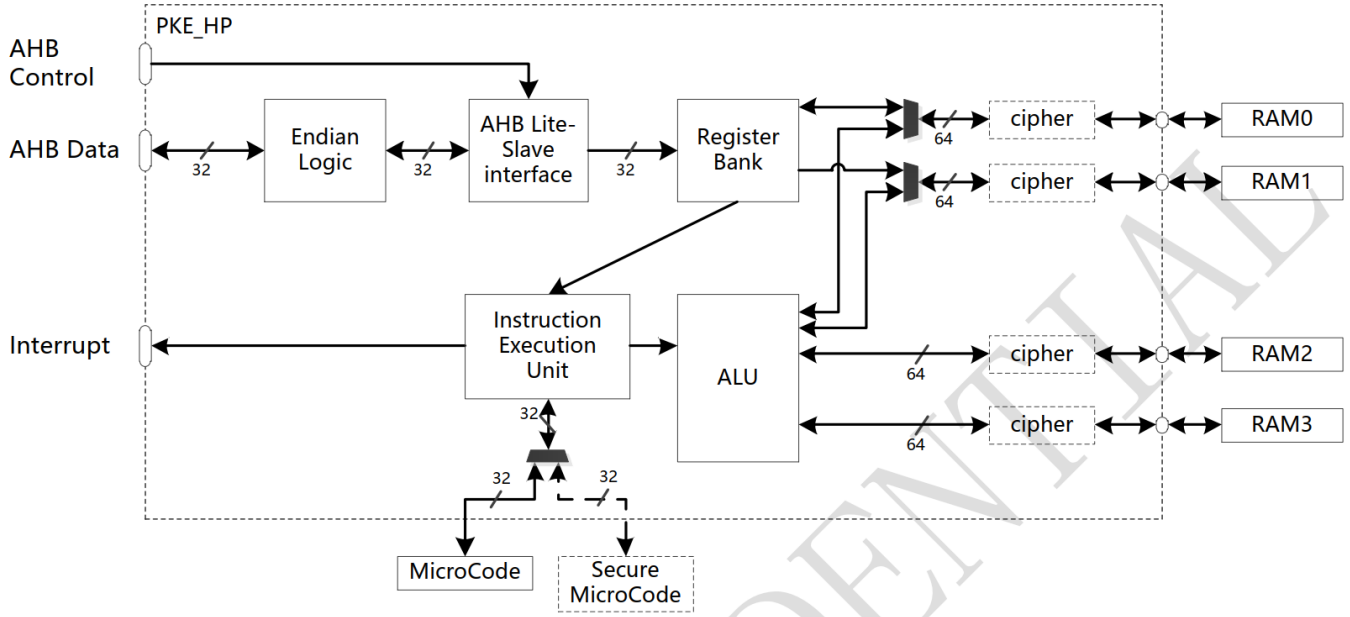


Figure 33.1: HSM Block Diagram

云途车规MCU信息安全(Cybersecurity)支持 – YUNTU eHSM

- YTM32B1Hx系列MCU 4MB及以上Flash的芯片将采用云途第二代HSM IP(eHSM)
  - ✓ 独立的RISC CPU内核和专业的安全存储Flash和SRAM
  - ✓ 全面支持国际通用加密算法硬件加速—AES-128/192/256，SHA256/384, ECC和RSA
  - ✓ 全面支持国密算法硬件加速— SM2/3/4/7/9
- 相应的HSM Firmware由第三方合作伙伴(伊世智能)提供
  - ✓ 满足ISO-21434汽车信息安全流程和产品认证
  - ✓ 支持lifecycle管理和安全调试授权
  - ✓ 支持HSM firmware安全升级
  - ✓ 支持SecureBoot
- 满足Evita-Full功能要求



分类	算法名称	选项
SM2 算法	密钥生成	支持
	密钥交换	支持
	签名	支持
	验签	支持
	加密	支持
	解密	支持
	安全签名	可选配
	安全验签	可选配
RSA 算法	密钥生成	支持
	CRT 密钥生成	支持
	签名/解密	支持
	验签/加密	支持
	CRT 签名/解密	支持
	安全签名/解密	可选配
	CRT 安全签名/解密	可选配
ECDSA 算法	密钥生成	支持
	签名	支持
	验签	支持
	安全签名	可选配
	安全验签	可选配
ECDH 算法	密钥生成	支持
	密钥交换	支持
X25519 算法	密钥生成	支持
	密钥协商	支持
Ed25519 算法	密钥生成	支持
	签名	支持
	验签	支持

CONFIDENTIAL

云途车规MCU信息安全(Cybersecurity)支持 – 安全启动(SecureBoot)

- Secure Boot 主要包含三个功能：
  - Secure Boot，安全验签功能，根据验签结果以及 BVT 配置决定是否 Boot
    - ✓ 支持最多8个安全启动区域配置(每个区域配置结构体可以独立加密)
  - Fast Wakeup support，可从 Power Down 模式唤醒，并快速跳转至 Application
  - Clock and WDG Config，可根据 BVT 配置系统时钟与看门狗
- 在云途配置工具(YCT)中提供SecureBoot的图形化配置和结果生成
- 详细使用请参考应用笔记《AN_0061_YTM32B1ME0_Secure_Boot_Guide》

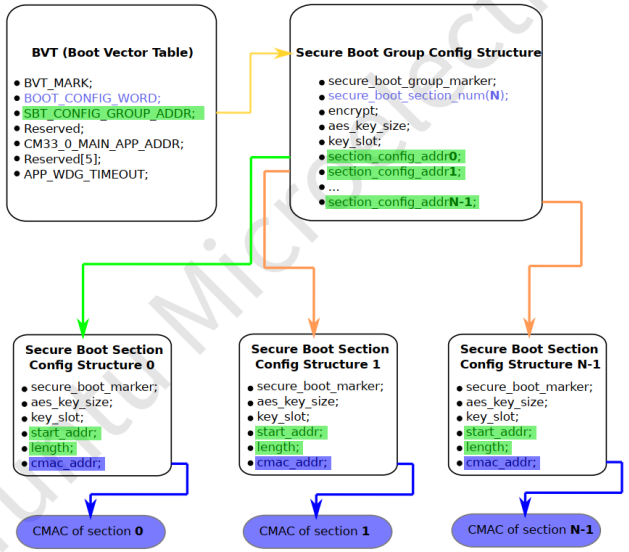


Figure 4.2: Secure Boot Group/Section Configuration Relationship with BVT

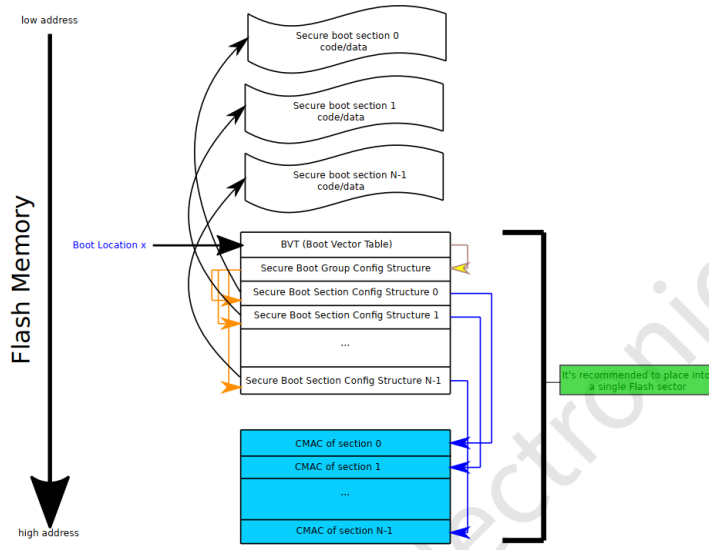


Figure 4.3: User Application Memory Map

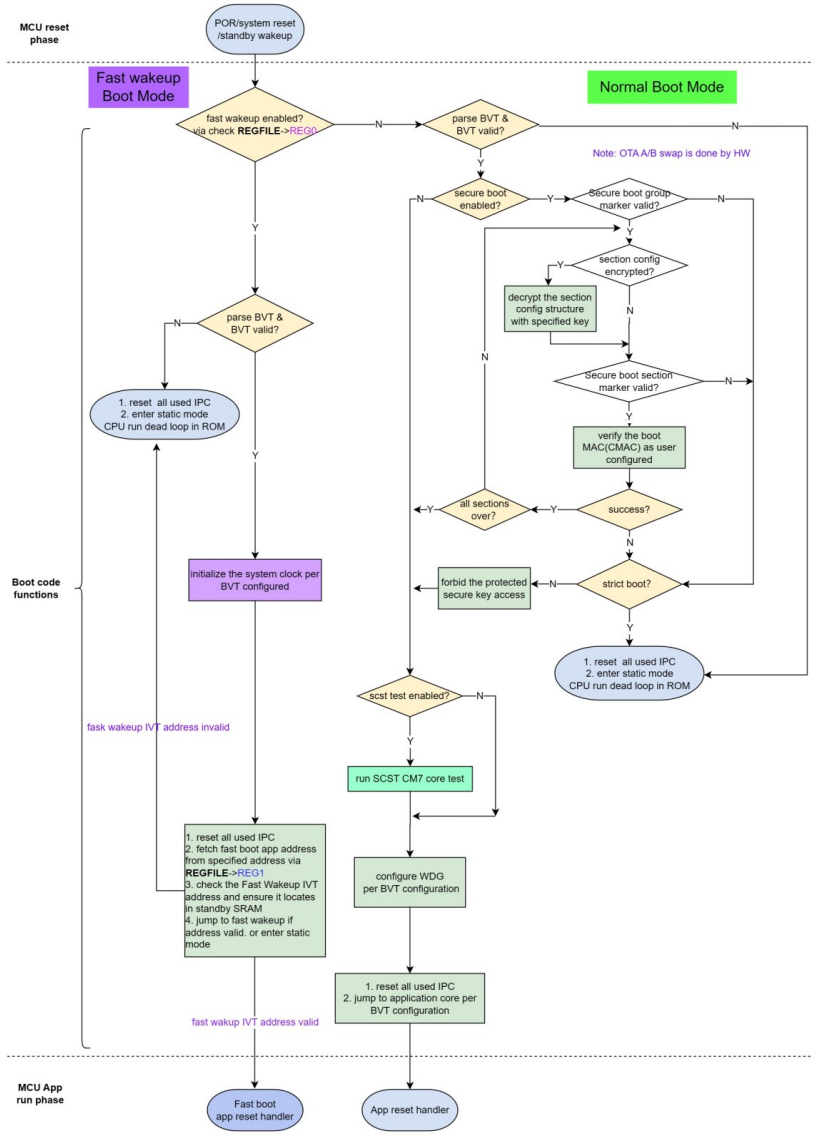


Figure 4.1: The ROM Boot Work Flowchart

总结

- 随着车联网技术(V2X和V2V)的日益普及和整车OTA技术的落地实施，现代汽车的网络信息安全(Cybersecurity)正变得越来越重要。
- 云途YTM32系列车规MCU产品全面配备硬件信息安全加速模块以保证汽车ECU和域控制器的信息安全
  - **HCU**：低成本解决方案，满足SHE、Evita-Light和部分Evita-Medium功能要求
  - **YUNTU HSM**：满足Evita-Full功能要求，免费提供云途自研HSM Firmware
  - **YUNTU eHSM**：高性能解决方案，满足Evita-Full功能要求，HSM Firmware有第三方合作伙伴提供(付费)，满足ISO-21434和FuSa ISO-26262规范
  - 支持安全启动(**SecureBoot**)并提供YCT图形化配置工具和应用笔记
  - 提供**MIRACL**和**MbedTLS**软件密码库(**Crypto Lib**)移植，通过对称和非对称加密算法以及国密SMx支持

feature		HCU						YUNTU HSM		eHSM
		YTM32B1MC03	YTM32B1MD14	YTM32B1MD24#	YTM32B1ME05	YTM32B1ME15#	YTM32B1HA01	YTM32BHA11#	YTM32BHB02#	后续Flash＞4MB的Hx系列MCU
对称加密算法硬件加速支持	AES	AES-128 (ECB/CBC)	AES-128 (ECB/CBC)	AES-128 (ECB/CBC)	AES-128/192/256 (ECB/CBC/CTR/CCM)	AES-128/192/256 (ECB/CBC/CTR/CCM)	AES-128/192/256 (ECB/CBC/CTR/CCM)	AES-128/192/256 (ECB/CBC/CTR/CCM)	AES-128/192/256 (ECB/CBC/CTR/CCM)	AES-128/192/256 (ECB/CBC/CTR/CCM)
	CMAC	√	√	√	√	√	√	√	√	√
非对称加密算法硬件加速支持	ECC	×(SW Crypto Lib***)	×(SW Crypto Lib***)	ECC-192/256/384	×(SW Crypto Lib***)	×(SW Crypto Lib***)	×(SW Crypto Lib***)	ECC-256/384	ECC-256/384	ECC-256/384/521
	RSA	×(SW Crypto Lib***)	×(SW Crypto Lib***)	Up to RSA-4096	×(SW Crypto Lib***)	×(SW Crypto Lib***)	×(SW Crypto Lib***)	Up to RSA-4096	Up to RSA-4096	Up to RSA-4096
	ECDSA	×(SW Crypto Lib***)	×(SW Crypto Lib***)	×(SW wrapper)	×(SW Crypto Lib***)	×(SW Crypto Lib***)	×(SW Crypto Lib***)	√	√	√
国密算法支持	SMx	×(SW Crypto Lib***)	×(SW Crypto Lib***)	×(SW Crypto Lib***)	SM4(ECB only) /(SW Crypto Lib***)	SM4(ECB only) /(SW Crypto Lib***)	SM4(ECB only) /(SW Crypto Lib***)	SM2/4	SM2/4	SM2/3/4/9
信息安全固件 HSM Firmware	Dedicated programmable CPU, NVR and RAM	Not required	Not required	Not required	Not required	Not required	Not required	Provided by YUNTU	Provided by YUNTU	Provided by 3rd partner
Others	SHA	×	×	SHA256	SHA256/384	SHA256/384	SHA256/384	SHA256/384/512	SHA256/384/512	SHA256/384/512
	HMAC	×	×	×	×	×	×	√	√	√
	Secure Key Lot Number	32x 128-bit	32x 128-bit	32x 128-bit	32x 256-bit	32x 256-bit	32x 256-bit	64x 256-bit	64x 256-bit	No limit
	IN/OUT FIFO	×	8*32 bits	8*32 bits	32*32 bits	32*32 bits	32*32 bits	32*32 bits	32*32 bits	32*32 bits
	Secure Boot	ROM_Boot**	User SW*	ROM_Boot**	User SW*	ROM_Boot**	ROM_Boot**	ROM_Boot**	ROM_Boot**	ROM_Boot**
满足汽车信息安全标准	Evita-Light/SHE	√	√	√	√	√	√	√	√	√
	Evita-Medium	√ (partial)	√ (partial)	√ (partial)	√ (partial)	√ (partial)	√ (partial)	√	√	√
	Evita-Full	×	×	×	×	×	×	√	√	√

- *YUNTU provides the secure boot reference SW and AN. The secure boot firmware occupies the user P-Flash
- **The SecureBoot is implemented in ROMBoot firmware which is pre-programmed into secured ROM at chip FT, user can enable and configure it via the BVT located in P-Flash.
- YUNTU can provide the open-source MIRACL porting on YTM32B1Mx for customer reference, the performance is not good enough for some time-critical application.
- # YTM32B1MD24,YTM32B1ME05, YTM32B1HA11 and YTM32B1HB02 are not mass production by 2025.Q3



- 加密算法在汽车ECU中的应用
- 汽车ECU的信息安全需求

加密算法在汽车ECU中的应用

- 加密算法在汽车ECU中的应用包括以下几类：

✓ **数据保护**(Data protection)：用于对ECU中的关键数或者算法代码(知识产权)据进行加密，以防止数据被窃取或篡改。常用的加密算法包括RAS和椭圆曲线加密算法(ECC)和对称加密算法(AES)。

✓ **身份验证**(Authority Verification)：在安全启动(SecureBoot)和安全升级(Bootloader/OTA)时，通过身份验证(CMAC/HMAC/GMAC、RSA签名验签、ECDSA)来确保只有授权用户可以访问ECU中的数据和功能。这可以通过硬件安全模块(HSM)和国密安全算法来实现。

✓ **访问控制**(Access Control)：用于限制对ECU中敏感数据和功能的访问，确保只有经过验证的用户才能执行特定操作。

✓ **安全通信**(Secure Communication)：通过加密通信报文报文防止非法窃听和用户隐私泄露。由于实时性的要求，通常使用对称加密算法(AES-128)，比如以太网IPsec、MACsec和TLS。

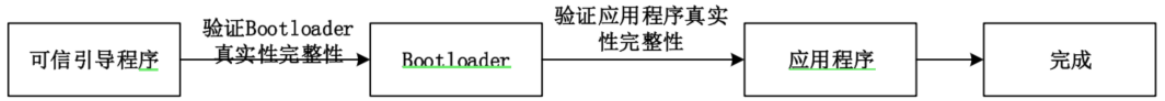


图1 ECU安全启动校验流程图

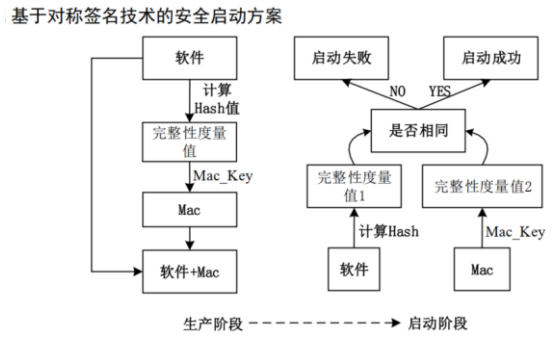


图3 ECU在生产阶段、上电启动阶段校验流程图

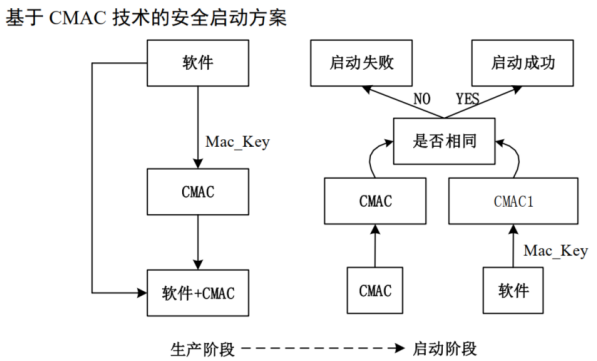
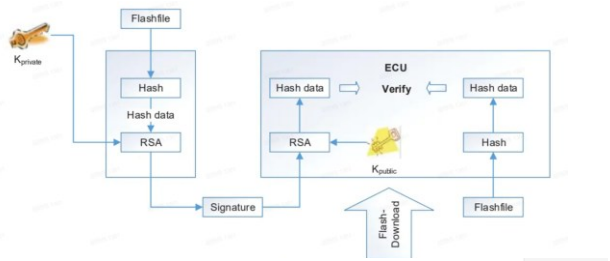


图4 ECU在生产阶段、上电启动阶段校验流程图



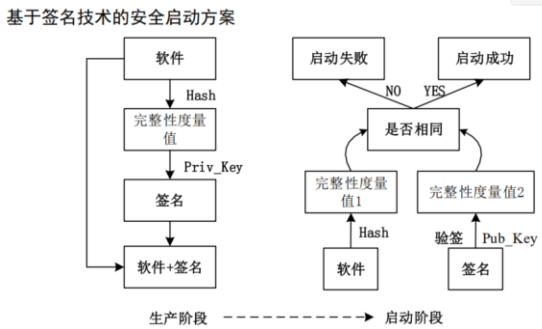
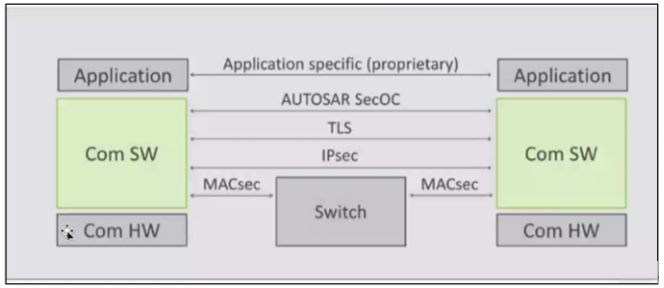


图2 ECU在生产阶段、上电启动阶段校验流程图



汽车ECU的信息安全需求

加密算法		国内整车厂1	国内整车厂2	国内整车厂3	国内整车厂4	备注
对称加密算法	AES	AES-128/192/256 AES-CBC/OFB/CFB/GCM CMAC HMAC	AES-128/192/256 AES-CBC/OFB/CFB/GCM CMAC HMAC GMAC	AES-128/192/256 AES-CBC/OFB/CFB/GCM CMAC HMAC GMAC	AES-128/192/256 AES-CBC/OFB/CFB/GCM CMAC HMAC GMAC	消息授权码(CMAC、HMAC和GMAC)用于关键用户代码和数据的一致性校验，防止恶意篡改
	Hash	SHA256/512	SHA256/512	SHA256/512	SHA256/512	
非对称加密算法	RSA	RSA-2048/3072/4096 RSA Encrypt/Verify	RSA-2048/3072/4096 RSA Encrypt/Verify	RSA-2048/3072/4096 RSA Encrypt/Verify	RSA-2048/3072/4096 RSA Encrypt/Verify	对于中央网关、T-BOX和座舱域控制器类应用，需要RSA秘钥≥3072bit，其他ECU应用RSA-2048即可。 RSA验签(verify)对应RSA加密，使用RSA公钥进行幂次方取模运算，其执行时间较签名/解密要短得多。
	ECC	ECC-256 ECDSA-verify ECDH	ECC-256 ECDSA-verify ECDH	ECC-256 ECDSA-verify ECDH	ECC-256 ECDSA-verify ECDH	ECC的性能和速度优于RSA算法
国密算法		N/A	SM2、SM3、SM4	N/A	N/A	SM2对应ECDSA(SHA-256 + ECC256)， SM3对应SHA-256， SM4对应AES-128。
随机数		TRNG	TRNG	TRNG	TRNG	用作秘钥交换或者安全通信时的种子使用，最好使用硬件TRNG
安全启动(secure boot)		√	√	√	√	要求上电后首先执行，其被保护不被改写
秘钥存储与管理	秘钥数量	TBD	TBD	TBD	TBD	
	存储位置	专用存储/加密存储	专用存储/加密存储	专用存储/加密存储	专用存储/加密存储	禁止明文存储秘钥，必须加密存储并使用secure_boot覆盖保证真实性
	秘钥更新	单独更新/产线烧录	单独更新/产线烧录	单独更新/产线烧录	单独更新/产线烧录	基于UDS进行秘钥分发，研发和量产使用不同的秘钥
其他要求(性能、法规要求)		TBD	TBD ISO-21434(prefer)	TBD	TBD	ISO-21434为具体的信息安全流程和认证规范，对使用的具体加密算法没有要求



- 软件密码算法库概览
  - ✓ MIRACL
  - ✓ MBEDTLS
- 软件密码算法库在YTM32上的移植与使用
  - ✓ 移植和使用
  - ✓ 性能测试

软件密码算法库概览--MIRACL

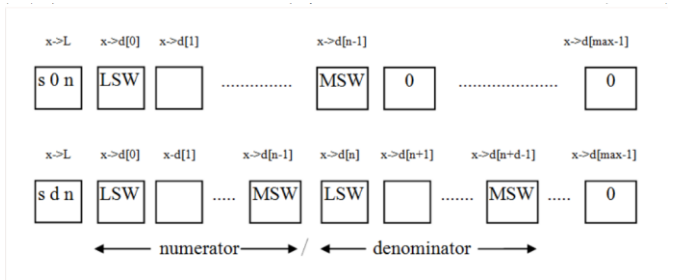
- 多精度整数和有理数算术C库(**MIRACL** - **M**ultiprecision **I**nteger and **R**ational **A**rithmetic **C** **L**ibrary 是一套由MIRACL开发的关于开源(open source on github: https://github.com/miracl/MIRACL)大数运算函数库，主要用于设计与大数运算相关的密码学应用。定义了两种新的数据类型- 用于大整数的big和用于大有理数的flash(缩写为浮动斜杠).它包含了多种加密算法和功能，如RSA公开密码学、Diffie-Hellman密钥交换、AES对称加密、DSA数字签名以及椭圆曲线密码学等。

MIRACL®

大数(big)和闪数(flash)数据类型的结构

```
struct bigtype
{
     mr_unsign32 L;
     mr_small *d;
};

typedef struct bigtype *big;
typedef struct bigtype *flash;
```



- **主要功能：**
  - ✓ **大数运算**：MIRACL能够处理非常大的整数和有理数运算，适用于需要高精度计算的场景。
  - ✓ **多种加密算法支持**：包括RSA、AES、DSA、Diffie-Hellman以及椭圆曲线加密等。
  - ✓ **模块化设计**：MIRACL的设计考虑到了可扩展性和模块化，使得添加新的加密算法或功能变得容易。
  - ✓ **多平台兼容性**：可以在Windows、Linux等多种操作系统上编译和运行。
  - ✓ **云服务支持**：MIRACL Trust平台提供基于云的多因素认证服务，无需将认证凭证发送到云端存储，从而提高了安全性。
- **优点：**
  - ✓ **速度快**：特别是在STM32平台上，MIRACL库具有较快的加密速度。
  - ✓ **体积小**：在嵌入式系统中，MIRACL库占用的内存较小，适合资源受限的环境。
  - ✓ **易于使用**：提供了丰富的示例和文档，方便用户学习和使用。
  - ✓ **跨平台兼容性**：支持多种操作系统和编程语言，包括C/C++。
  - ✓ **安全性高**：设计时考虑了侧信道攻击的防护，增强了系统的安全性。
- **缺点：**
  - ✓ **代码维护问题**：由于代码历史悠久，缺乏良好的维护，可能会导致一些潜在的问题。
  - ✓ **复杂性**：对于初学者来说，MIRACL库的学习曲线可能较陡峭，需要花费较多时间进行学习和调试。
- MIRACL库在密码学领域具有广泛的应用，提供了强大的大数运算和多种加密算法的支持，但在代码维护和学习难度方面存在一定的挑战。

MIRACL also supports a wide range of platforms, including, among others:

软件加密算法库在YTM32上的移植和使用 -- MIRACL

- 步骤① 、在PC上编译运行config.c获取推荐MIRAL配置(mirdef.h)和编译文件list(依赖的MIRACL库文件miracl.list);
- 步骤②、 在PC上编译运行mex.c 生成**KCM**加速所需的mrkcm.c，具体命令如下:

```
./mex.exe 8 ./source/ mrkcm.c
```

- 步骤③、将miracl.list中要求MIRACL库源文件、 mirdef.h和第②步生成的mrkcm.c添加开发到YTM32的工具链编译生成静态库文件(与具体选择编译工具链相关)
  - ✓ 配置编译选项(比如编译器优化等级、使用系统库等)
  - ✓ 添加MIRACL库文件到编译路径
  - ✓ 设置编译结果输出路径和名称
- 步骤④、将编译的静态库和MIRACL的头文件添加到应用工程，调用MIRACL底层大数运算API封装实现具体的加密算法，比如RSA和ECC加解密与签名验签
  - ✓ 配置MIRACL静态库搜索路径
  - ✓ 设置工程堆栈大小(heap和stack)
  - ✓ 调用所有MIRACL API之前必须初始化MIRACL库

Name	Description
BOOL EXACT;	初始化为TRUE。如果在闪数运算过程中进行了舍入操作，则设置为FALSE。
int INPLEN;	输入字符串的长度。在输入二进制数据时必须使用。
int IOBASE;	用于输入和输出的"可打印"数制。可以在程序中随意更改。必须大于或等于2且小于或等于256。
int IOBSIZ;	I/O缓冲区的大小。
BOOL ERCON;	默认情况下，错误会生成错误消息并立即中止程序。或者通过设置mip->ERCON=TRUE将错误控制留给用户。
int ERNUM;	发生的最后一个错误的编号。
char IOBUFF[ ];	输入/输出缓冲区。
int NTRY;	isprime函数中用于概率素性测试的迭代次数。初始化为6。
int *PRIMES;	指向小素数表的指针。
BOOL RPOINT;	如果设置为TRUE，则输出数值带有小数点。否则它们以分数形式输出（默认模式）。
BOOL TRACER;	如果设置为ON，会导致打印出调试信息，跟踪所有后续对MIRACL例程的调用的进展情况。初始化为OFF。

在miracl.h中定义的**miracl**结构的成员。
通过**mip(MIRACL**实例指针)进行访问。

Tips：对于云途YTM32B1Mx和YTM32B1HA01系列车规MCU芯片来说，CM33和CM7内核都是32bit内核，其基础数据char为8bit；short为16bit；int，long和float都是32bit，double和long long为64bit

```
/*
 *   MIRACL compiler/hardware definitions - mirdef.h
 */
#define MR_LITTLE_ENDIAN
#define MIRACL 32
#define mr_utype int
#define MR_IBITS 32
#define MR_LBITS 32
#define mr_unsign32 unsigned int
#define mr_dltype long long
#define mr_unsign64 unsigned long long
#define MR_NOASM
#define MR_FLASH 52
#define MR_STRIPPED_DOWN
#define MR_NO_STANDARD_IO
#define MR_NO_FILE_IO
#define MAXBASE ((mr_small)1<<(MIRACL-1))
#define MR_KCM 8
#define MR_BITSINCHAR 8
#define MR_SMALL_EWINDOW
#define MR_NO_SS
```

步骤①生成的配置文件mirdef.h

```
mrcore.c
mrarth0.c
mrarth1.c
mrarth2.c
mrsmall.c
mrio1.c
mrio2.c
mrgcd.c
mrjack.c
mrxgcd.c
mrarth3.c
mrbits.c
mrrand.c
mrprime.c
mrcrt.c
mrscrt.c
mrmonty.c
mrpower.c
mrsroot.c
mrlucas.c
mrshs.c
mrshs256.c
mraes.c
mrgcm.c
mrfpe.c
mrstrong.c
mrcurve.c
mrbrick.c
mrebrick.c
mrzzn2.c
mrzzn2b.c
mrzzn3.c
mrecn2.c
mrfast.c
mralloc.c
mrshs512.c
mrsha3.c
mrflash.c
mrfrnd.c
mrdouble.c
mrround.c
mrbuild.c
mrflsh1.c
mrpi.c
mrflsh2.c
mrflsh3.c
mrflsh4.c
mrec2m.c
mrgf2m.c
mrkcm.c

Compile the above with -O2 optimization
```

步骤①生成的依赖文件miracl.list

```
#include "miracl.h" /* include the MIRACL header files */

void miracl_samples(void)
{
    epoint *g,*ea,*eb;  /* define the epoint variables */
    big a,b,p,q,n;      /* define the big variables */

#ifndef MR_STATIC
#ifndef MR_NOFULLWIDTH
    /* initialize the MIRACL *mip to support 40byte big/flash using heap memory*/
    mip = mirsys(40,256);
#else
    mip = mirsys(36,MAXBASE);
#endif
#else
   /* initialize the MIRACL *mip using the stack memory*/
    mip = mirsys(MR_STATIC,0);
    /* define the local array from stack for big/epoints*/
    char mem[MR_BIG_RESERVE(MR_STATIC)];
    /* clear the local stack memory */
    memset(mem, 0, MR_BIG_RESERVE(MR_STATIC));
#endif

#ifndef MR_STATIC
    /* allocate the bigs from heap and initialize them to zero */
    a = mirvar(0);
    b = mirvar(0);
    p = mirvar(0);
    q = mirvar(0);
    n = mirvar(0);
    /* allocate the epoints from heap and initialize them to zero */
    g = epoint_init();
    ea = epoint_init();
    eb = epoint_init();
#else
   /* initialize the big and epoint variables from the stack array */
    a   = mirvar_mem(mem, 0);
    b   = mirvar_mem(mem, 1);
    p   = mirvar_mem(mem, 2);
    q   = mirvar_mem(mem, 3);
    n   = mirvar_mem(mem, 4);
    g = epoint_init_mem(_MIPD_ mem, 5);
    ea = epoint_init_mem(_MIPD_ mem, 6);
    eb = epoint_init_mem(_MIPD_ mem, 7);
#endif /* MR_STATIC */

/* use the big variables with the MIRACL APIs */

/* free the memory used by the big/epoints at the end */
#ifndef MR_STATIC
    /* free the heap memory allocated for the bigs */
    mirkill(a);
    mirkill(b);
    mirkill(p);
    mirkill(q);
    mirkill(n);
    epoint_free(g);
    epoint_free(ea);
    epoint_free(eb);
#else
     /* clear stack memory used for the bigs */
    memset(mem, 0, MR_BIG_RESERVE(MR_STATIC));
#endif
}
```

步骤④ MIRACL使用样例

MIRACL在YTM32上的性能测试结果 – ME05

AES-128/192/256 加解密算法测试						
test cases	time(ms)					
	16B	64KB	128KB	256KB	512KB	1MB
AES-128 CBC	0.017	69.632	139.264	278.528	557.056	1114.112
AES-192 CBC	0.02	81.92	163.84	327.68	655.36	1310.72
AES-256 CBC	0.023	94.208	188.416	376.832	753.664	1507.328

SHA-256/384/512哈希算法测试						
test cases	time(ms)					
	56/112B	64KB	128KB	256KB	512KB	1MB
SHA-256	0.128	149.7965714	299.5931429	599.1862857	1198.372571	2396.745143
SHA-384	0.408	238.7382857	477.4765714	954.9531429	1909.906286	3819.812571
SHA-512	0.413	241.664	483.328	966.656	1933.312	3866.624

国密算法测试						
test cases	time(ms)					
	210B/3B/64B/32B*	64KB	128KB	256KB	512KB	1MB
SM2_Sign	87.629					
SM2_verify	176.33					
SM3	0.055(**3B**)/0.114(**64B**)/0.243(**210B**)					
SM4	0.022	45.056	90.112	180.224	360.448	720.896

Note: SM2 test with 210B message input which will also call SM3 to calculate the hash of the 210B message , SM3 test with 3B and 64B data input, SM4 test with 32B data input.

ME05 MIRACL RSA加解密性能测试结果(GCC v10.3 -O3)
（使用heap动态内存，使用KCM（8）加速优化）
测试消息为512/1024/2048/3072/4096bit(64/128/256/384/512Byte)

	CPU cyclecount				
	RSA-512	RSA-1024	RSA-2048	RSA-3072	RSA-4096
加密开始	33799754	707454315	1969857201	2573590404	4188203529
加密结束/解密开始	34289114	708935381	1974520397	2584761005	4202626773
解密结束	39981440	747675263	2230821763	3654852391	5775758005
加密时间(inst cycle count)	489360	1481066	4663196	11170601	14423244
解密时间(inst cycle count)	5692326	38739882	256301366	1070091386	1573131232
加密时间(ms) @120MHz	4.078	12.34221667	38.85996667	93.08834167	120.1937
解密时间(ms) @120MHz	47.43605	322.83235	2135.844717	8917.428217	13109.42693

MIRACL在YTM32上的性能测试结果 – HA01

AES-128/192/256 加解密算法测试						
test cases	time(ms)					
	16B	64KB	128KB	256KB	512KB	1MB
AES-128 CBC	0.026	106.496	212.992	425.984	851.968	1703.936
AES-192 CBC	0.021	86.016	172.032	344.064	688.128	1376.256
AES-256 CBC	0.018	73.728	147.456	294.912	589.824	1179.648

SHA-256/384/512哈希算法测试						
test cases	time(ms)					
	56/112B	64KB	128KB	256KB	512KB	1MB
SHA-256	0.033	38.61942857	77.23885714	154.4777143	308.9554286	617.9108571
SHA-384	0.108	63.19542857	126.3908571	252.7817143	505.5634286	1011.126857
SHA-512	0.106	62.02514286	124.0502857	248.1005714	496.2011429	992.4022857

HA01 MIRACL RSA加解密性能测试结果(GCC v10.3 -O3) (i-Cache enabled，使用heap动态内存，使用KCM（8）加速优化，测试消息长度为512/1024/2048/3072/4096bit(64/128/256/384/512Byte)					
	CPU cyclecount				
	RSA-512	RSA-1024	RSA-2048	RSA-3072	RSA-4096
加密开始	118991441	559185573	3531144549	3692967396	2431168202
加密结束/解密开始	119282408	559951098	3533377631	3696229111	2438031484
解密结束	122279487	578371939	3649540004	4013440497	3182834126
加密时间(inst cycle count)	290967	765525	2233082	3261715	6863282
解密时间(inst cycle count)	2997079	18420841	116162373	317211386	744802642
加密时间(ms) @200MHz	1.454835	3.827625	11.16541	16.308575	34.31641
解密时间(ms) @200MHz	14.985395	92.104205	580.811865	1586.05693	3724.01321

国密算法测试						
test cases	time(ms)					
	210B/3B/64B/32B*	64KB	128KB	256KB	512KB	1MB
SM2_Sign	46.635					
SM2_verify	94.002					
SM3	0.023(**3B**)/0.035(**64B**)/0.072(**210B**)					
SM4	0.012	24.576	49.152	98.304	196.608	393.216

Note: SM2 test with 210B message input which will also call SM3 to calculate the hash of the 210B message , SM3 test with 3B and 64B data input, SM4 test with 32B data input.

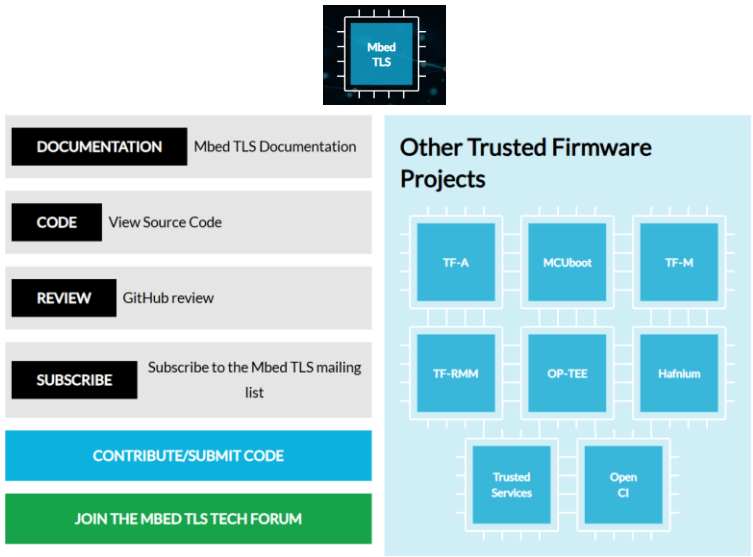
The precomputation should not be used for GF(2^m) **Koblitz** Elliptic Curve

ECC160/192/224/256/521测试									
test cases	time(ms)								
	160	192	224	256	NA	NA	NA	NA	
ECDH(no precom)	17.25	24.86	31.58	46.28					GF(p) Elliptic Curve
ECDH(w. precom)	9.35	12.71	16.41	25.37					
ECDSA signature(no precom)	17.25	24.86	31.58	46.28					
ECDSA signature(w. precom)	9.35	12.71	16.41	25.37					
ECDSA verify	17.25	31.8	39.36	59.53					
	163	163 Koblitz	233	233 Koblitz	283	283 Koblitz	571	571 Koblitz	
ECDH(no precomputation)	21.25	10.03	37.82	16.8	54.34	22.58	265.76	89.57	GF(2^m) Elliptic Curve
ECDH(w. precomputation)	12.39	11.94	23.18	23.35	32.81	34.83	180.35	166.86	
ECDSA(no precomputation)	21.25	10.03	37.82	16.8	54.34	22.58	265.76	89.57	
ECDSA(w. precomputation)	12.39	11.94	23.18	23.35	32.81	34.83	180.35	166.86	
ECDSA verify	29.05	21.48	51.83	36.13	77.2	54.59	397.8	243.34	

Note: **ECDH** - Diffie Hellman Key exchange; **ECDSA** - Digital Signature Algorithm, **precom** - precomputation to speed up EC Diffie-Hellman implementations.

软件密码算法库概览—Mbed TLS

- Mbed TLS(前身为PolarSSL)是由ARM公司开源和维护的一个加密库(https://github.com/Mbed-TLS/mbedtls)，使用标准C语言编写。它提供了一组可单独使用和编译的加密组件，并且可以通过单个配置头文件来加入或排除这些组件。
- 从功能角度来看，Mbed TLS主要分为三个部分：SSL/TLS协议实现、加密库和X.509证书处理库。具体功能包括初始化SSL/TLS上下文、进行SSL/TLS握手、发送/接收数据、通知对等方连接关闭等。此外，它还支持导入密钥、使用RSA签名消息、使用对称密码、哈希消息、生成随机值、从现有密钥派生新密钥、认证以及加密或解密消息等功能。
- Mbed TLS的优点在于其代码占用空间小，适合嵌入式系统。它提供了直观的API、可读性强的源代码和高度可配置的代码足迹。然而，缺点可能包括在某些情况下可能会有较高的耦合度，尽管其独立的模块设计有助于降低模块之间的耦合度。
- Mbed TLS是一个功能强大且灵活的加密库，适用于需要安全通信的嵌入式设备和物联网应用。
- **优势：**
  - ✓ **轻量级设计**：Mbed TLS的设计非常轻量级，特别适合用于嵌入式系统。它的代码结构紧凑，优化面向微控制器，因此在低功耗和小内存设备上表现出色。
  - ✓ **易于集成**：Mbed TLS的API简洁明了，易于集成到各种软件项目中。它旨在帮助开发者更好地创建应用程序，并提供丰富的文档、示例和支持。
  - ✓ X.509**证书管理功能**：Mbed TLS提供了X.509证书的生成、存储和验证功能，这对于保证通信双方的身份安全至关重要。
  - ✓ **模块化设计**：Mbed TLS的各个功能模块相对独立、耦合低，可以通过配置宏定义裁剪，非常适合用于嵌入式系统。
- **劣势：**
  - ✓ **文档和社区支持**：虽然Mbed TLS提供了丰富的文档和示例，但其社区支持和资源可能不如OpenSSL广泛。OpenSSL作为一个历史悠久的项目，拥有庞大的用户基础和社区支持。
  - ✓ **兼容性问题**：由于Mbed TLS是较新的项目，可能会存在一些兼容性问题或不被某些旧系统的完全支持。



Mbed TLS在嵌入式系统中的实际应用案例包括以下几个方面：

1. **网络通信安全**：Mbed TLS可以用于提高网络通信的安全性，适用于需要加密实现的应用[22]。
2. **物联网设备**：Mbed TLS非常适合用于物联网设备，因为其代码占用空间小，易于移植和集成。它提供了常用的加密/解密算法、X.509证书操作以及TLS/DTLS协议的实现[23][27]。
3. **STM32设备**：多个证据表明，Mbed TLS被广泛应用于STM32设备上。例如，有经验分享如何将Mbed TLS移植到STM32裸机，并进行测试与使用[28][29]。此外，还可以通过cubemx代码生成工具直接在STM32项目中添加Mbed TLS库，并调用其API[30]。
4. **Apollo3 MCU**：Mbed TLS也被应用于Apollo3 MCU上，实现了AES-256加密功能[21]。
5. **Sitara MCU**：Mbed TLS可以在Sitara MCU器件上集成，用于定义明确的方法来将加密操作从软件卸载到硬件[22]。

MIRACL和Mbed TLS软件密码库在YTM32上的移植工程

