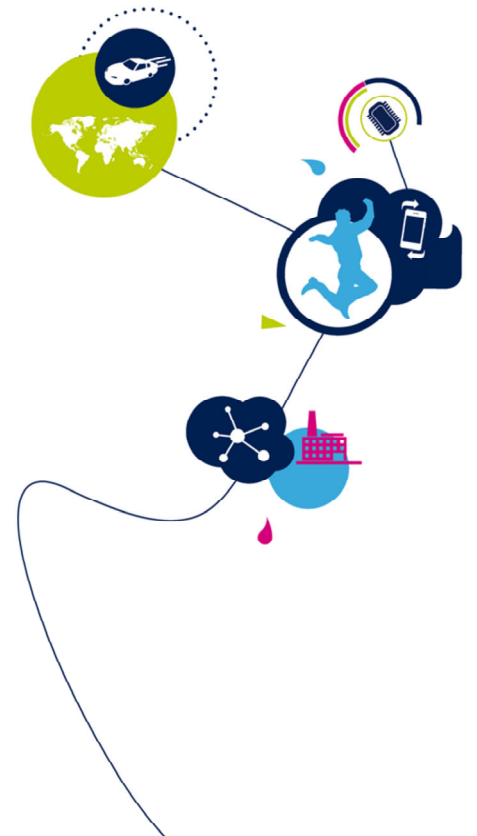


STM32WB – PKA

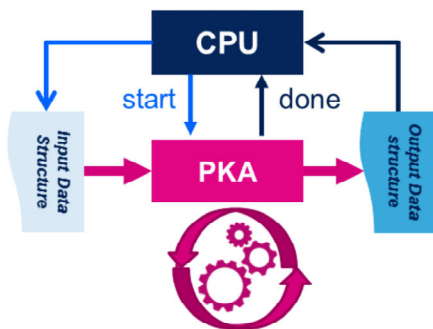
Public Key Accelerator

Revision 1.0



Hello and welcome to this presentation of the STM32 **Public Key Accelerator**, which is embedded in STM32WB microcontrollers.

It covers the features used to perform Asymmetric Key Cryptography, which is widely required for cryptographic applications.



- The use of the PKA greatly accelerates public key cryptography performed by the CPU
- The PKA supports many security standards widely used in the world (NIST, IEEE, ANSI...)

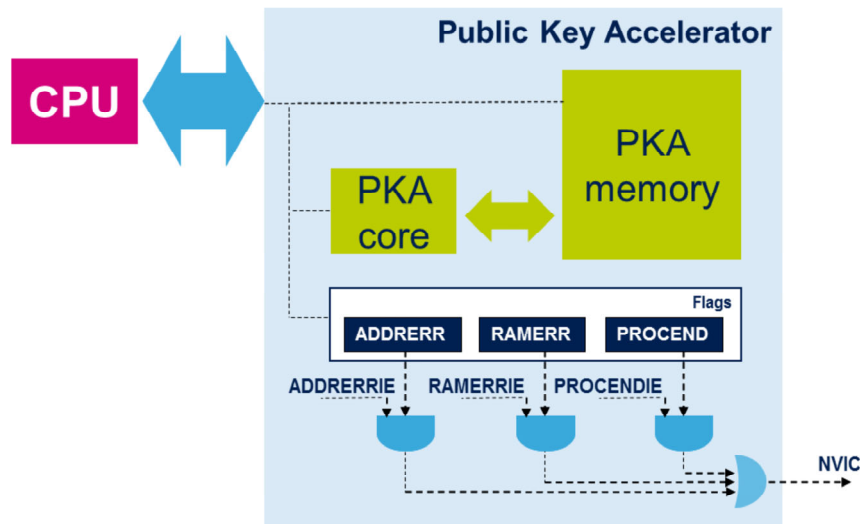
Application benefits

- Establish secure communication channels across open networks like the Internet, or provide integrity and authentication via electronic signatures
- Greatly reduces CPU processing time, which is an order of magnitude longer than for symmetric crypto (AES).

Public Key cryptography is part of many security standards and is widely used to establish secure communication channels across unsecure open networks like Internet or to provide authentication via electronic signatures. Software-only solutions can be too slow for real-time applications, impacting the system's overall performance. The PKA peripheral is an efficient hardware accelerator that speeds up the public key cryptography operations performed by the CPU.

PKA block diagram

3



Performing public key cryptography requires intensive computing which represents a huge workload when done entirely by software. The Public Key Accelerator lightens the STM32WB CPU's workload by performing key operations in the PKA core, using dedicated PKA memory.

The CPU loads initial data into the PKA internal RAM, which is located at address offset 0x400. Then in the PKA control register, the CPU specifies the operation which is to be executed and finally asserts the START bit. Once the PKA reports the end of operation (PROCENDF), the CPU reads the resulting data from the PKA RAM, then clears the PROCENDF flag.

Software can abort a PKA operation at any time by clearing the EN bit in PKA_CR register. In this case, the content of the PKA memory is not guaranteed.

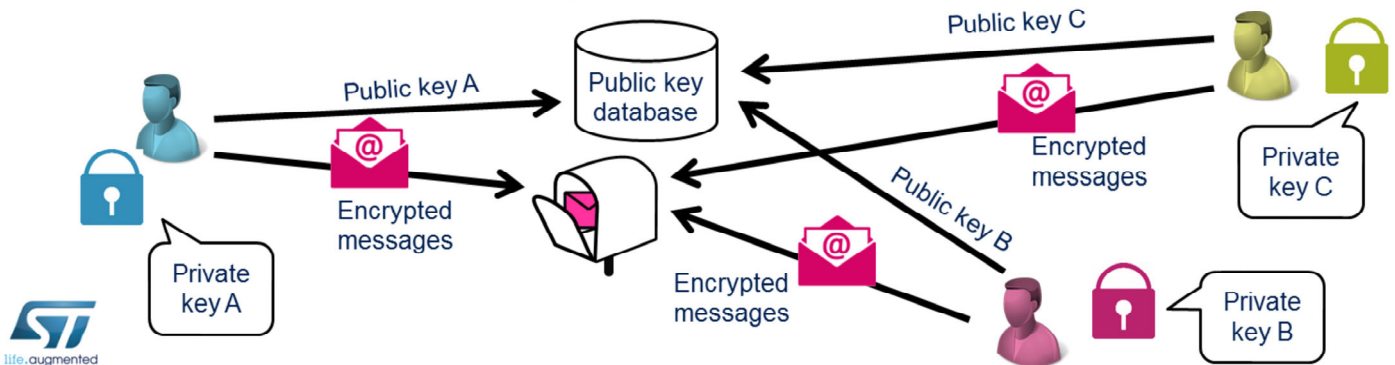
The PKA has two error flags: the Address Error flag (ADDRERRF) and the RAM Error Flag (RAMERRF). All flags can generate an interrupt if the corresponding Interrupt

Enable bit is set (PROCENDIE, ADDRERRIE or RAMERRIE).

Application example: key distribution

4

- When multiple persons wish to share information in a confidential way, it is much more efficient to use two different keys.
 - One key for the encryption of a plaintext message, one key for the decryption of a ciphertext message.
- An encryption key is public, so anyone can send an encrypted message to the sole owner of the decryption key (which is private)



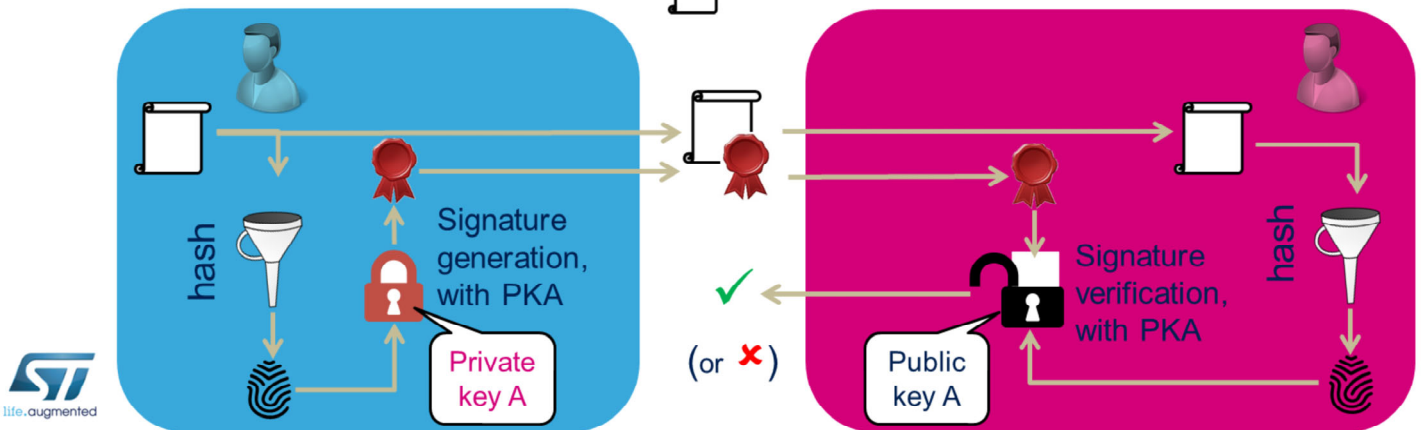
Public key cryptography introduces a very elegant solution for the problem of exchanging messages in a confidential way over an unsecure network like the Internet. Each person exchanging messages possesses a private key used to decrypt messages sent to him/her encrypted using his/her public key.

For this technology to work, a trusted central repository of the people's public key is recommended.

Application example: digital signatures

5

- The private key, accessible only to the owner, can be used to generate a digital signature that anybody can verify using the owner's public key information.
- If the digital signature  does not work with a given public key, its sole owner  cannot be the one that prepared the message 



A digital signature is a powerful technology to ensure the integrity, authentication and non-repudiation of digital assets such as financial transaction tokens.

Person "A" can prepare a signed message by first performing a secure hashing function on it, then encrypting the resulting digest using his private key. The resulting signature is sent alongside with the message to Person "B". Person "B" can verify A's signed message by performing the same hashing function on it, and then use the result when performing the signature verification function using A's public key. The result of the verification function will determine if the message is genuine or not.

- Acceleration of asymmetric cryptography (FIPS 186-4, RSA PKCS#1, ANSI X9.62, Brainpool, ...)
 - Modular exponentiation and its faster CRT version (Chinese Remainder Theorem)
 - ECC scalar multiplication
 - Point on curve check, critical to verify the received public key (from the other peer)
 - ECDSA signature generation and verification
- Arithmetic and modular operations like addition, subtraction, multiplication, comparison, reduction, ...



Here is a list of operations the PKA can perform.

Acceleration of asymmetric cryptography:

- Modular exponentiation and RSA Chinese Remainder Theorem (CRT) exponentiation
- ECC scalar multiplication and point on curve check
- ECDSA signature generation and verification

Arithmetic and modular operations:

- Arithmetic addition, subtraction, multiplication, and comparison
- Modular addition, subtraction, and reduction & inversion
- Montgomery multiplication

Thanks to these operations, the PKA supports many standard Public Key algorithms: Modular Exponentiation, CRT exponentiation, RSA cryptography, Elliptic Curve Cryptography (ECC), Digital Signature Algorithm (DSA), and Elliptic Curve DSA (ECDSA).

PKA Features (2/2)

7

- Capability to handle operands up to 3136 bits for RSA/DH and 640 bits for elliptic curves
 - According to the cryptographic key length recommendation website <https://www.keylength.com>, these key values are future-proof using current computer architectures
- Montgomery domain inward and outward transformations
- AMBA AHB slave peripheral



Public Key Accelerator (PKA) is used to accelerate Rivest, Shamir and Adleman (RSA); Diffie-Hellman (DH); as well as Elliptic Curve Cryptography (ECC) over prime field operations. Supported operand sizes are up to 3136 bits for RSA and DH, and up to 640 bits for ECC.

The PKA is an ARM Advanced Microcontroller Bus Architecture (AMBA) AHB slave peripheral, accessible through 32-bit word single accesses only (otherwise, for writes, an AHB bus error is generated, and write accesses are ignored).

PKA processing time 8

- Modular exponentiation operation (in milliseconds)

Exponent length (in bits)	Operand length (in bits)		
	1024	2048	3072
2¹⁶+1	3 or 1 (fast)	9 or 4 (fast)	20 or 9 (fast)
1024	91 or 88 (fast) or 27 (CRT)	-	-
2048	-	654 or 640 (fast) or 183 (CRT)	-
3072	-	-	2140 or 573 (CRT)

Note 1: Fast mode requires one Montgomery parameter computation
 Note 2: CRT is Chinese Remainder Theorem optimization

- Other operations (in milliseconds)

	Modulus length (in bits)			
	256	384	512	521
ECC scalar multiplication	38	106	225	259
ECDSA signature	41	114	239	277
ECDSA verification	82	227	479	559



Here are the modular exponentiation processing times using different exponent and operand sizes.

Figures with the (fast) indication requires the application to perform a Montgomery parameter computation, as this information is needed to run the fast operation. The Montgomery parameter can be reused for several computation in a row, making the overall operations more efficient if repeated many times.

Montgomery multiplication overhead: 1024-bit (+0ms), 2048-bit (+3ms), 3072-bit (+8ms).

Interrupt event	Description
PKA end of operation	Set when the computation is completed.
PKA RAM access error	Set when a PKA RAM access is detected while a PKA operation is in progress.
Access to unmapped address error	Set when a PKA RAM access is detected out of range (unmapped address)

- DMA usage is not supported with the PKA



Here is a summary of the PKA events able to trigger an interrupt in the nested vectored interrupt controller: PKA computation completed, PKA RAM access error, and access to unmapped address error.

The Direct Memory Access (DMA) controller cannot be used with the PKA.

Low-power modes

10

Mode	Description
Run	Active
Sleep	Active by default (but can be disabled)
Stop 0 / Stop 1 / Stop 2	Disabled
Standby	Powered-down
Shutdown	Powered-down



Here is an overview of the status of the PKA peripheral in each of the low-power modes.
PKA operations are not possible when the device is in Stop mode.

Related peripherals

11

- Refer to these peripheral trainings linked to this peripheral
 - RCC (PKA clock enable, PKA reset)
 - Interrupts (NVIC)



This is a list of peripherals related to the PKA. Please refer to these peripheral trainings for more information if needed.

- For more details and additional information, refer to the following
 - RSA
 - PKCS#1: RSA Cryptography Standard (v1.5, 2.1 and 2.2) [RSA Labs]
 - DH
 - ANSI X9.42: Implementation of Diffie-Hellman [ANSI]
 - PKCS#3: Diffie-Hellman Key Agreement Standard [RSA Labs]
 - Elliptic curves
 - ANSI X9.63: Key Agreement and Key Transport Using Elliptic Curve Cryptography [ANSI]
 - IEEE 1363: Standard Specifications For Public Key Cryptography [ANSI]
 - Elliptic curves (cont.)
 - ANSI X9.62: The Elliptic Curve Digital Signature Algorithm [ANSI]
 - FIPS 186-4: Digital Signature Standard (DSS) [NIST]
 - SP 800-56A and SP 800-56B [NIST]
 - Curve25519: Key establishment based on ECC [Daniel J. Bernstein]



If these links and the reference manual are not enough, please refer to the PKA driver in the STM32CubeMX repository described in the next slide.

- For more details and additional information, refer to the following STMicroelectronics software references
 - Two C software libraries are provided to help users developing an application with PKA on STM32WB
 - Low-layer (LL) Library is for experienced users, programming PKA by direct register accesses.
 - Hardware abstraction layer (HAL) library, providing easy-to-use functionalities.
 - These libraries come with STM32CubeMx (a graphical configuration tool for STM32) directly available on www.st.com:
 - [stm32cubemx \(www.st.com\)](http://stm32cubemx.st.com)
 - These libraries can be downloaded directly as a .zip on the STM32CubeWB page



For more details and additional information, refer to the following useful software references.