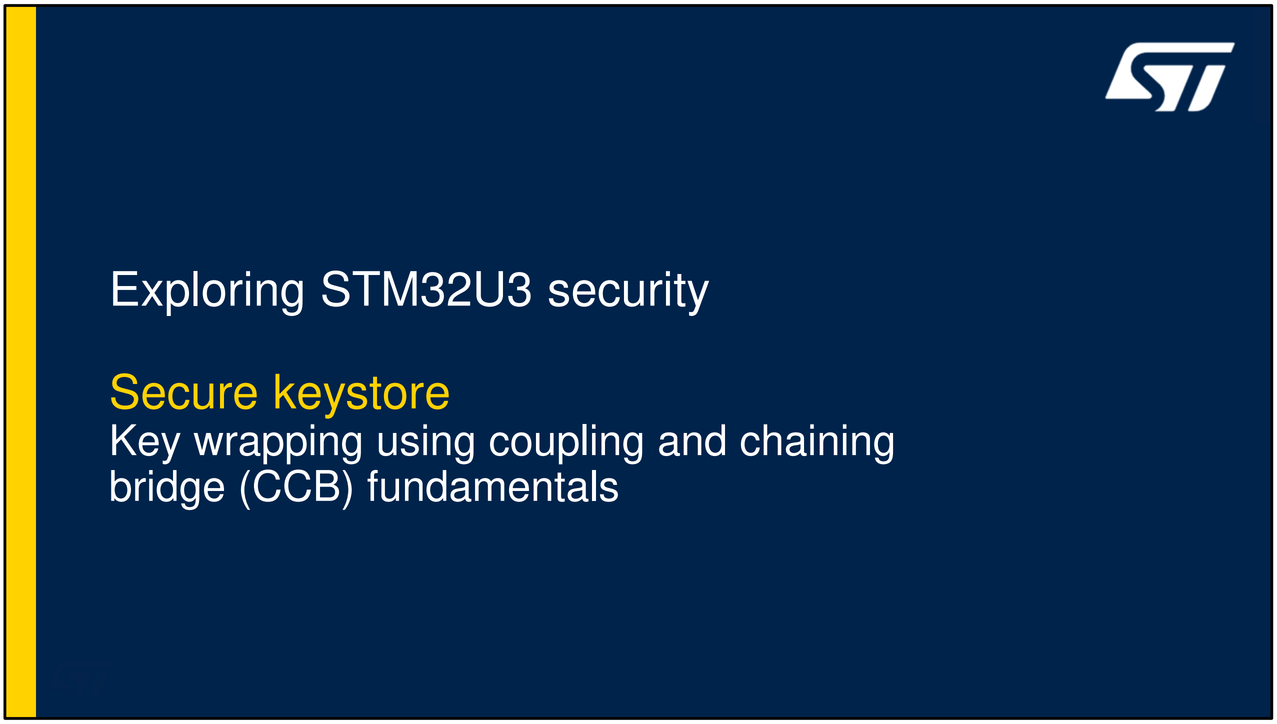


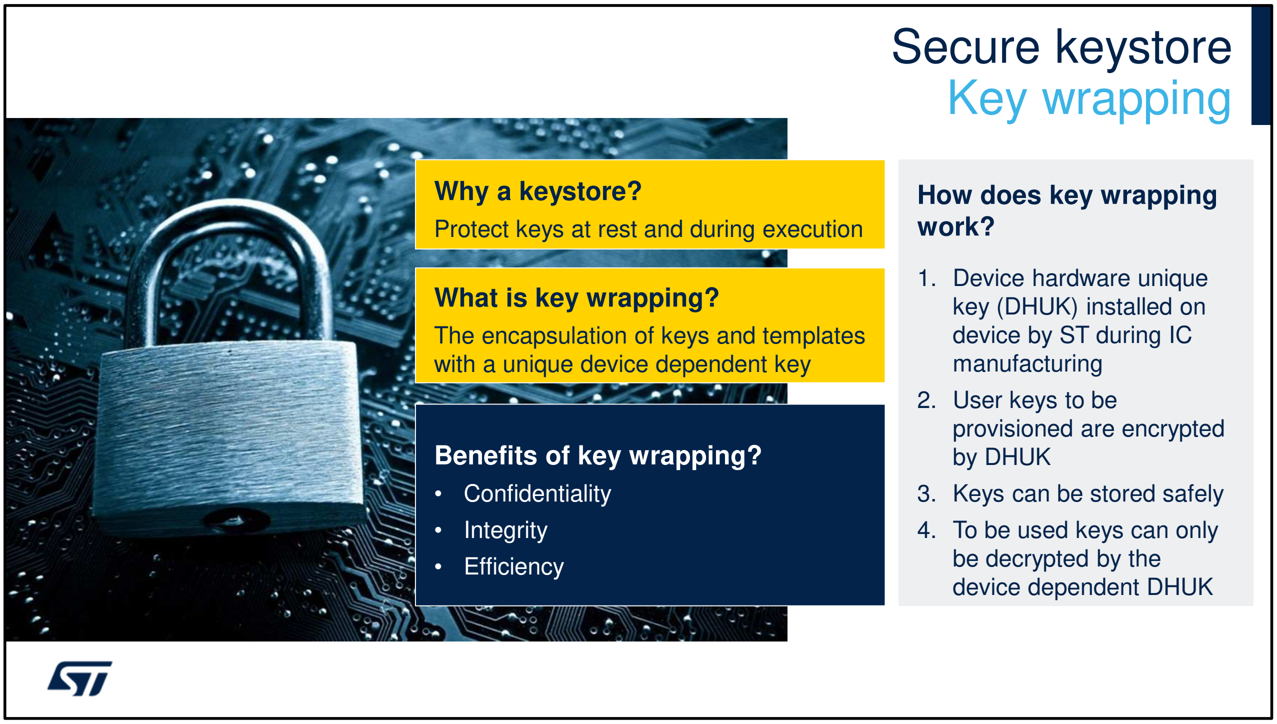
Exploring STM32U3 security

Secure keystore

Key wrapping using coupling and chaining bridge (CCB) fundamentals

Hello and welcome to this presentation on "Exploring STM32U3 Security: Secure Keystore and Key Wrapping Using Coupling and Chaining Bridge (CCB) Fundamentals."

Today, we will delve into the advanced security features of the STM32U3 microcontroller, focusing on its secure keystore capabilities and the innovative key wrapping techniques facilitated by the Coupling and Chaining Bridge (CCB).



Secure keystore
Key wrapping

Why a keystore?

Protect keys at rest and during execution

What is key wrapping?

The encapsulation of keys and templates with a unique device dependent key

Benefits of key wrapping?

- Confidentiality
- Integrity
- Efficiency

How does key wrapping work?

1. Device hardware unique key (DHUK) installed on device by ST during IC manufacturing
2. User keys to be provisioned are encrypted by DHUK
3. Keys can be stored safely
4. To be used keys can only be decrypted by the device dependent DHUK

Key wrapping is a cryptographic technique used to securely encapsulate encryption keys. It is a method of encrypting keys with another key, often referred to as a "key-encryption key" (KEK). On STM32 called DHUK, unique per device. to ensure their confidentiality and integrity during storage or transmission. Key wrapping is particularly useful in scenarios where multiple keys need to be managed securely, such as in key management systems or when distributing keys across different systems.

Key Features of Key Wrapping:

1.Confidentiality: The primary goal is to protect the wrapped key from unauthorized access. Only entities with the correct DHUK can unwrap and access the original key.

2.Integrity: Ensures that the wrapped key has not been altered or tampered with during storage or transmission.

3. **Efficiency**: Key wrapping algorithms are designed to be efficient, allowing for quick wrapping and unwrapping processes.

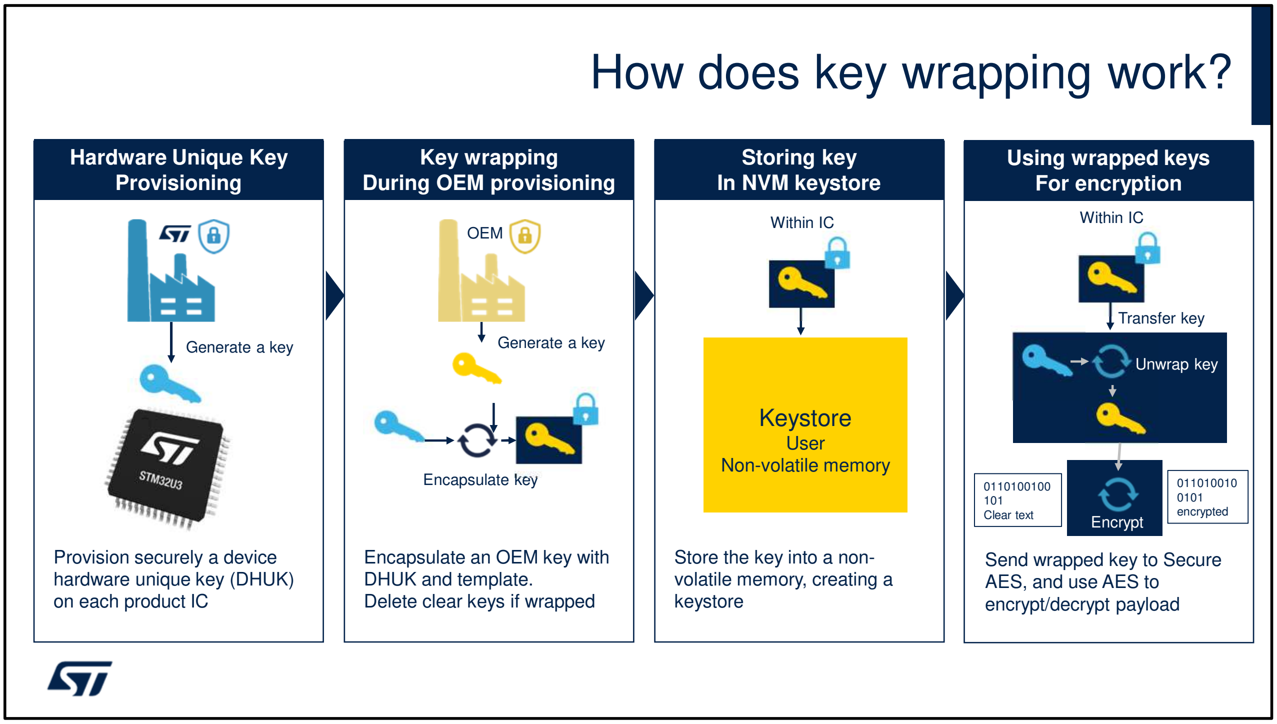
How Key Wrapping Works:

1.Key in-ST provisioning: A Device Hardware Unique Key (DHUK) is provision by ST on each device. This key is used to wrap other keys.

2.Wrapping Process: The original key is encrypted using the DHUK. This process may involve additional steps to ensure integrity, such as adding authentication tags.

3.Storage/Transmission: The wrapped key is stored or transmitted securely. It is now protected by the DHUK.

4.Unwrapping Process: When access to the original key is needed, the wrapped key is decrypted using the DHUK, restoring the original key.



How does key wrapping work?

Hardware Unique Key Provisioning

Provision securely a device hardware unique key (DHUK) on each product IC

Key wrapping During OEM provisioning

Encapsulate an OEM key with DHUK and template.
Delete clear keys if wrapped

Storing key In NVM keystore

Store the key into a non-volatile memory, creating a keystore

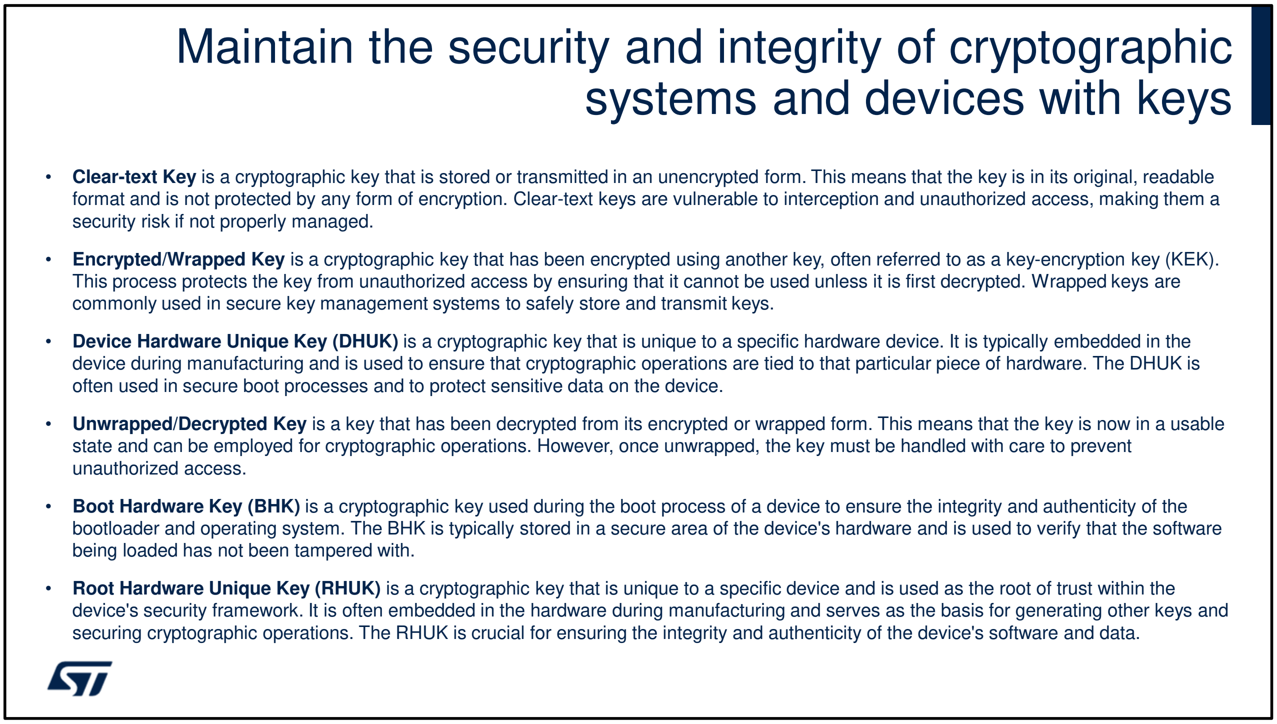
Using wrapped keys For encryption

Send wrapped key to Secure AES, and use AES to encrypt/decrypt payload

Let’s see how key wrapping is used:

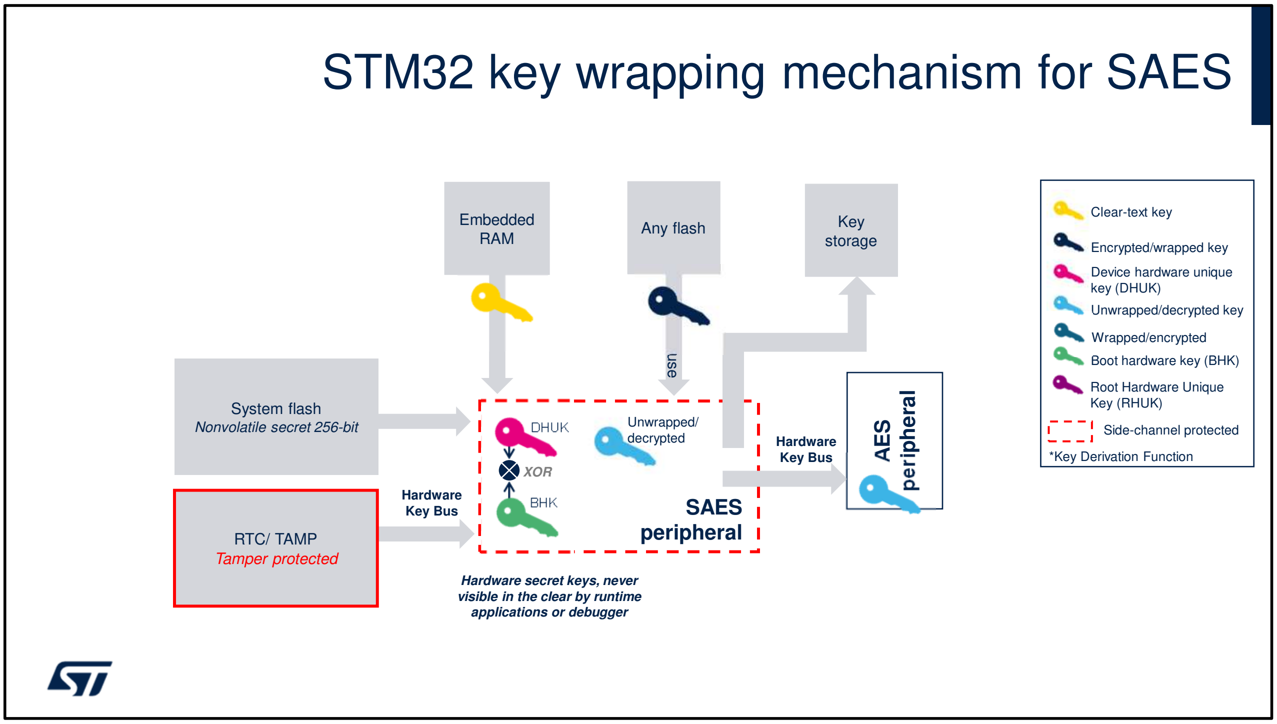
1) First, ST will generate within a secure environment a secret. This secret is different for all die/product/device. It will then be installed on the device and locked so that application CPU cannot get access to this secret. The secret is called in the device a Hardware Unique Key (HUK)
2) Second, OEM generates a secret (an ECC or AES key) within its secure environment. The secret is injected in the device and wrapped/encapsulated with the HUK. The original key in clear shall be destroyed.
3) Third, this wrapped secret can be stored into any application memory. As it is wrapped the secret cannot be recovered in clear.
4) Finally, to process an encryption/decryption of an asset, the user will provide the wrapped key to the Secure AES (SAES) and CCB and the blob to encrypt/decrypt. The SAES will

unwrap the key, always protecting it in confidentiality, and use it to encrypt/decrypt the asset. Once done the key in clear will be flushed to avoid leakage. Only SAES internal logic hac access to the key in clear.



Maintain the security and integrity of cryptographic systems and devices with keys

- **Clear-text Key** is a cryptographic key that is stored or transmitted in an unencrypted form. This means that the key is in its original, readable format and is not protected by any form of encryption. Clear-text keys are vulnerable to interception and unauthorized access, making them a security risk if not properly managed.
- **Encrypted**/**Wrapped Key** is a cryptographic key that has been encrypted using another key, often referred to as a key-encryption key (KEK). This process protects the key from unauthorized access by ensuring that it cannot be used unless it is first decrypted. Wrapped keys are commonly used in secure key management systems to safely store and transmit keys.
- **Device Hardware Unique Key (DHUK)** is a cryptographic key that is unique to a specific hardware device. It is typically embedded in the device during manufacturing and is used to ensure that cryptographic operations are tied to that particular piece of hardware. The DHUK is often used in secure boot processes and to protect sensitive data on the device.
- **Unwrapped**/**Decrypted Key** is a key that has been decrypted from its encrypted or wrapped form. This means that the key is now in a usable state and can be employed for cryptographic operations. However, once unwrapped, the key must be handled with care to prevent unauthorized access.
- **Boot Hardware Key (BHK)** is a cryptographic key used during the boot process of a device to ensure the integrity and authenticity of the bootloader and operating system. The BHK is typically stored in a secure area of the device's hardware and is used to verify that the software being loaded has not been tampered with.
- **Root Hardware Unique Key (RHUK)** is a cryptographic key that is unique to a specific device and is used as the root of trust within the device's security framework. It is often embedded in the hardware during manufacturing and serves as the basis for generating other keys and securing cryptographic operations. The RHUK is crucial for ensuring the integrity and authenticity of the device's software and data.



STM32 key wrapping mechanism for SAES

Embedded RAM

Any flash

Key storage

use

System flash
Nonvolatile secret 256-bit

RTC/ TAMP
Tamper protected

Hardware Key Bus

DHUK

XOR

BHK

Unwrapped/ decrypted

SAES peripheral

Hardware Key Bus

AES peripheral

Hardware secret keys, never visible in the clear by runtime applications or debugger

- Clear-text key
- Encrypted/wrapped key
- Device hardware unique key (DHUK)
- Unwrapped/decrypted key
- Wrapped/encrypted
- Boot hardware key (BHK)
- Root Hardware Unique Key (RHUK)
- Side-channel protected

*Key Derivation Function

To better protect keys, the side-channel protected SAES peripheral can use special hardware secret keys (DHUK, BHK) to ensure that critical application keys are never visible in clear text to the runtime application or the debugger :

- DHUK: Nonvolatile, secret to application

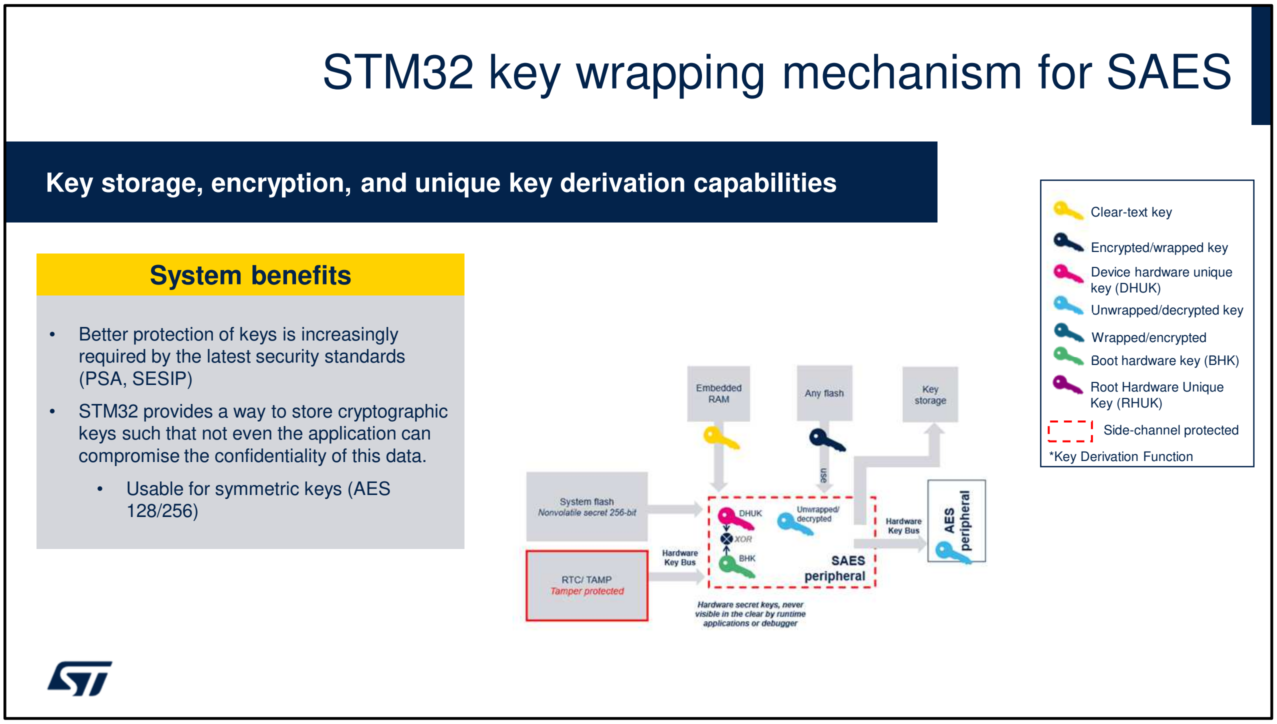
- Derived using a key derivation function that uses key usage (mode, security) as input

- Boot hardware key (BHK): Volatile, tamper protected (in TAMP)
  - Written then locked by boot application

SAES can be used to protect 128-bit and 256-bit keys used in SAES or AES peripherals.

This module will detail the following use cases:
- Encrypting keys with hardware secret keys (key wrapping)

- Decrypting keys with hardware secret keys (key unwrapping)
- Decrypting keys for AES peripheral (shared key unwrapping)



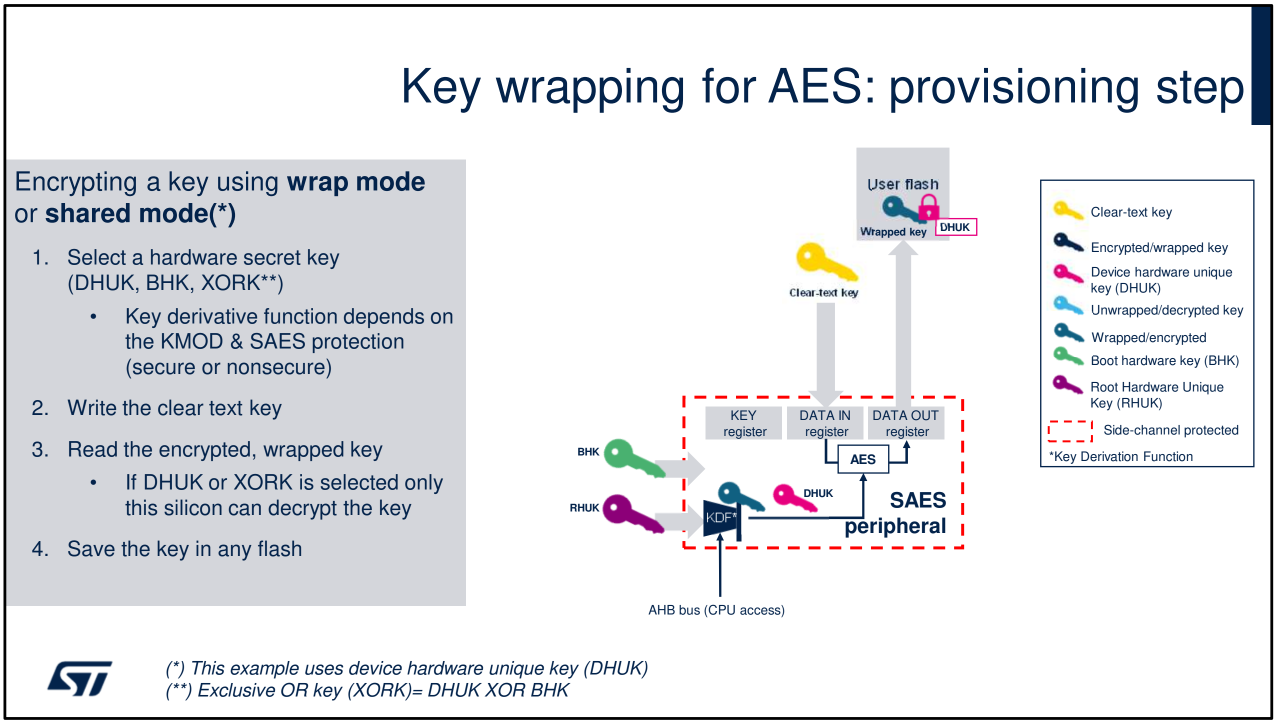
STM32 key wrapping mechanism for SAES

Key storage, encryption, and unique key derivation capabilities

System benefits

- Better protection of keys is increasingly required by the latest security standards (PSA, SESIP)
- STM32 provides a way to store cryptographic keys such that not even the application can compromise the confidentiality of this data.
  - Usable for symmetric keys (AES 128/256)

The main benefits of this system is :
Better protection of keys is increasingly required by the latest security standards (PSA, SESIP)
STM32 provides a way to store cryptographic keys such that not even the application can compromise the confidentiality of this data.
Usable for symmetric keys (AES 128/256)



Key wrapping for AES: provisioning step

Encrypting a key using **wrap mode** or **shared mode(*)**

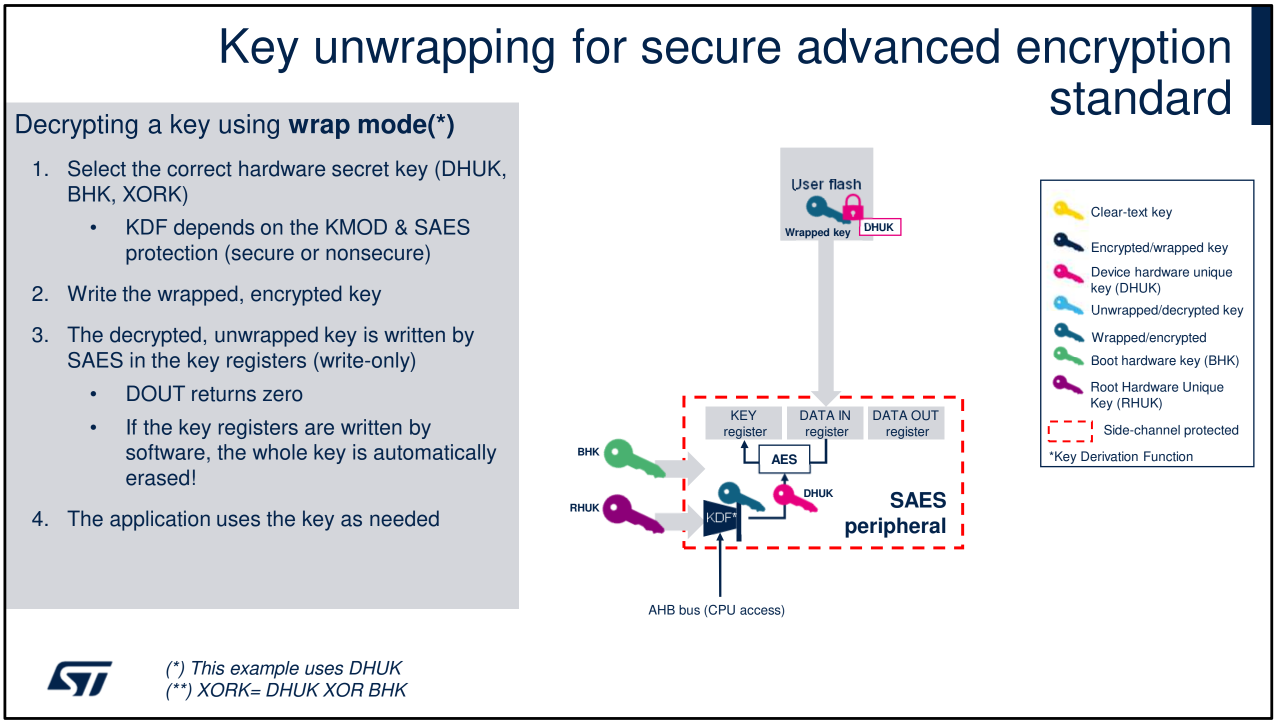
1. Select a hardware secret key (DHUK, BHK, XORK**)
   - Key derivative function depends on the KMOD & SAES protection (secure or nonsecure)
2. Write the clear text key
3. Read the encrypted, wrapped key
   - If DHUK or XORK is selected only this silicon can decrypt the key
4. Save the key in any flash

(*) This example uses device hardware unique key (DHUK)
(**) Exclusive OR key (XORK)= DHUK XOR BHK

This slide details how to encrypt a key using the wrap key mode. The resulting encrypted key can be stored in any flash memory.

The first step involves selecting a hardware secret key and then selecting the wrapped key mode with KMOD. The second step is to write the clear text key into the SAES DATA IN register, run the AES encryption, and then read the resulting encrypted key from the DATA OUT register.

If the AES peripheral needs to use this key instead of the SAES, the shared mode must be selected in KMOD instead of the wrap mode.



Key unwrapping for secure advanced encryption standard

Decrypting a key using **wrap mode(*)**

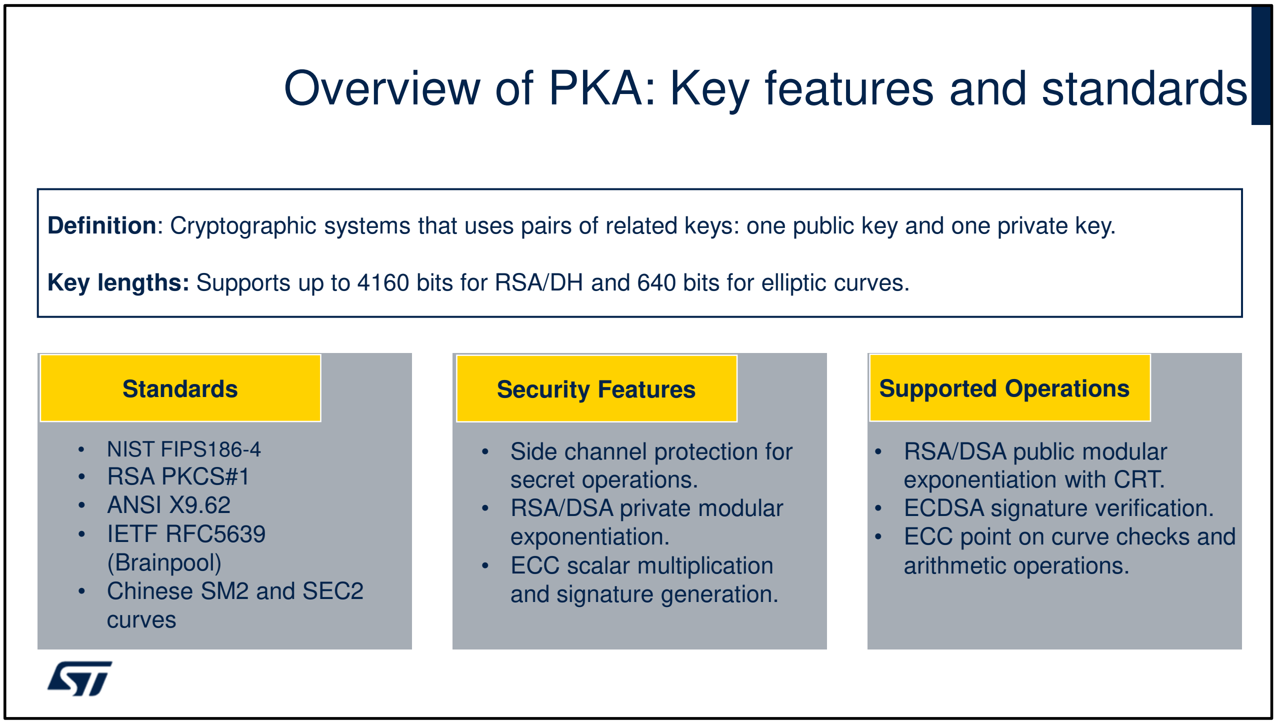
1. Select the correct hardware secret key (DHUK, BHK, XORK)
   - KDF depends on the KMOD & SAES protection (secure or nonsecure)
2. Write the wrapped, encrypted key
3. The decrypted, unwrapped key is written by SAES in the key registers (write-only)
   - DOUT returns zero
   - If the key registers are written by software, the whole key is automatically erased!
4. The application uses the key as needed

(*) This example uses DHUK
(**) XORK= DHUK XOR BHK

Before usage, any key encrypted with the wrap key mode must be decrypted using the same mode, with the correct key. The result is automatically stored in the write-only key registers.

The first step involves selecting the correct hardware secret key and then selecting the wrapped key mode with KMOD.
The second step is to write the wrapped text key into the SAES DATA IN register, run the AES decryption, and then wait for the SAES to store the original key in the write-only key registers. The application can now use the unwrapped key as needed.

Note that this unwrapped key is automatically erased when the software writes the key registers.



Overview of PKA: Key features and standards

Definition: Cryptographic systems that uses pairs of related keys: one public key and one private key.

Key lengths: Supports up to 4160 bits for RSA/DH and 640 bits for elliptic curves.

Standards

- NIST FIPS186-4
- RSA PKCS#1
- ANSI X9.62
- IETF RFC5639 (Brainpool)
- Chinese SM2 and SEC2 curves

Security Features

- Side channel protection for secret operations.
- RSA/DSA private modular exponentiation.
- ECC scalar multiplication and signature generation.

Supported Operations

- RSA/DSA public modular exponentiation with CRT.
- ECDSA signature verification.
- ECC point on curve checks and arithmetic operations.

This slide provides a concise overview of Public Key Asymmetric Cryptography, a fundamental concept, It highlights the key features, standards, and operations that make asymmetric cryptography a tool for ensuring secure communication and data protection.

•**Definition**: Explains the use of public and private keys for encryption and decryption.

•**Key Lengths**: Highlights supported sizes for RSA/DH and elliptic curves.

•**Standards**: Lists protocols like NIST FIPS186-4 and RSA PKCS#1.

•**Security Features**: Covers protections like side channel security.

•**Supported Operations**: Details operations such as RSA/DSA exponentiation and ECDSA verification.

Exact list of operations the PKA can perform are below.

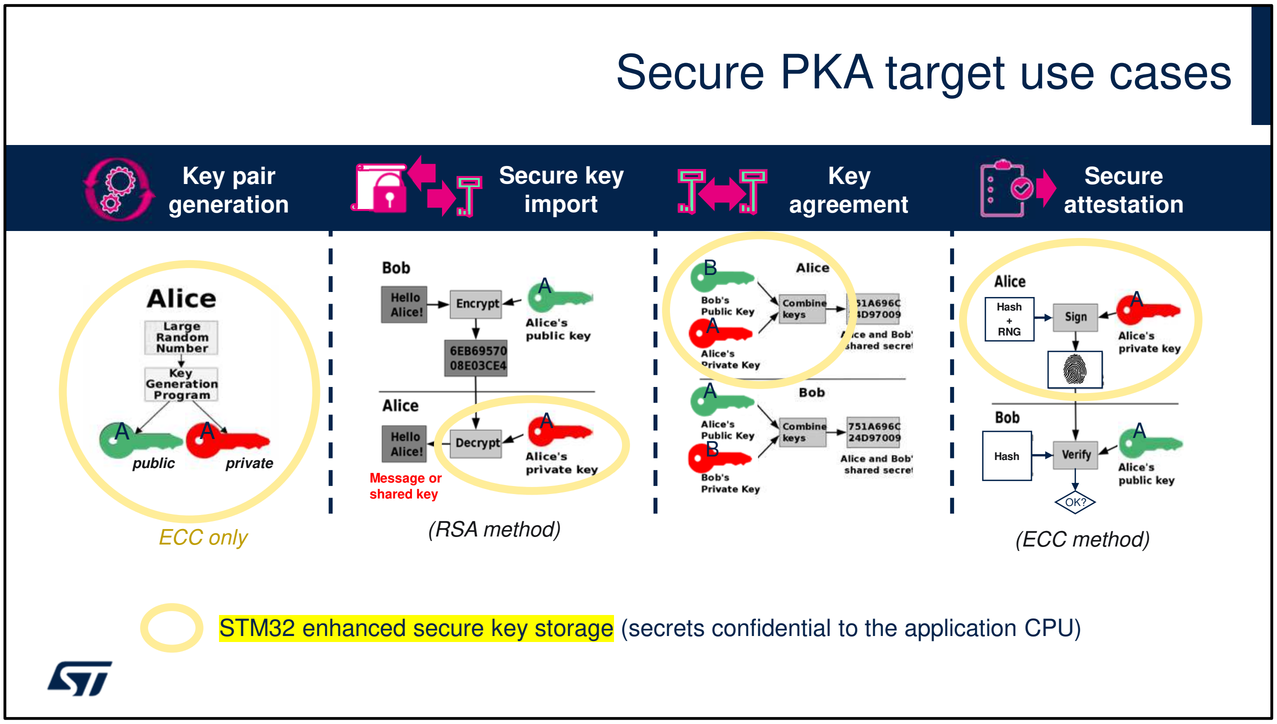
Acceleration of asymmetric cryptography:

- Modular exponentiation, RSA Chinese Remainder Theorem (CRT) exponentiation

- ECC scalar multiplication, point on curve check
- ECDSA signature generation and verification

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

Thanks to those operations PKA supports a lot of standard Public Key algorithms: Modular Exponentiation, CRT exponentiation, RSA cryptography, Elliptic Curve Cryptography (ECC), Digital Signature Algorithm (DSA), Elliptic Curve DSA (ECDSA)



Secure PKA target use cases

Key pair generation | Secure key import | Key agreement | Secure attestation

ECC only

(RSA method)

(ECC method)

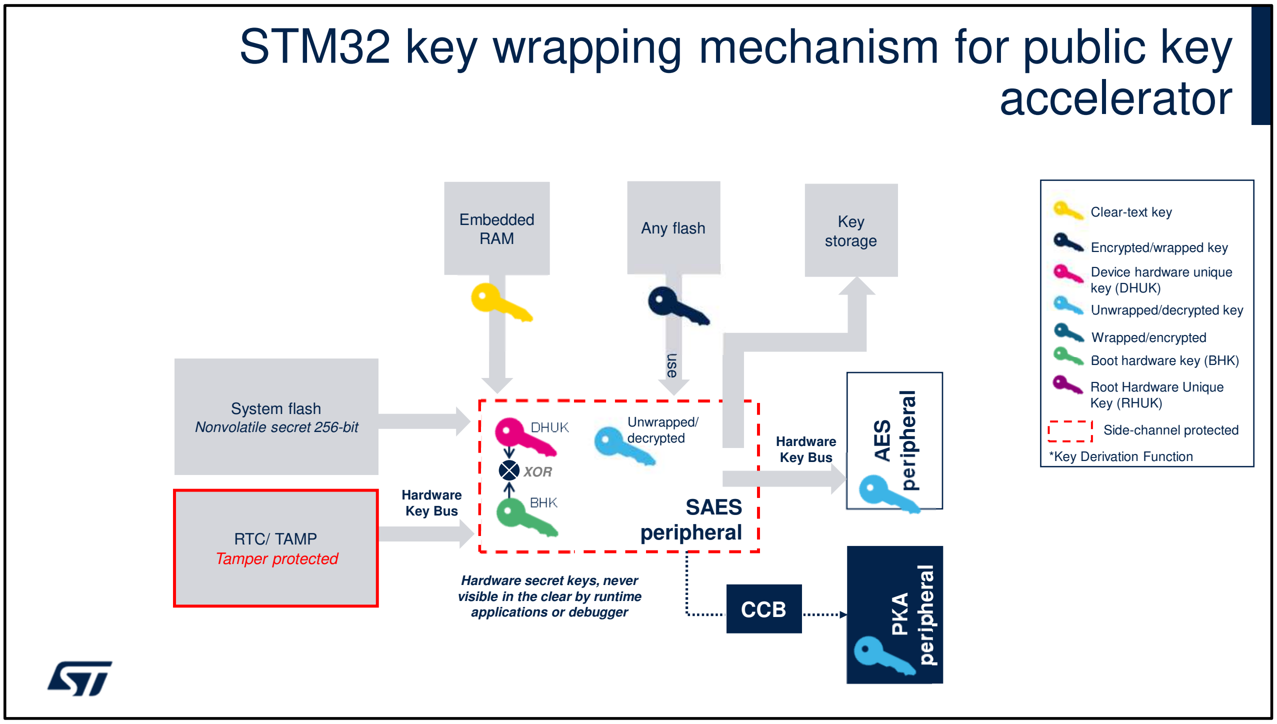
○ STM32 enhanced secure key storage (secrets confidential to the application CPU)

The security of asymmetric cryptography relies heavily on the protection of the private key. For example, as the private key is used to decrypt messages that were encrypted with the user's public key, if someone gains access to a private key, it is then possible to decrypt and read confidential messages.

Also, as the private key is used to create digital signatures that verify the user's identity, if someone other than the user has access to the private key, the user's digital signature can be forged, making it appear as though the user has approved or sent messages that were not written by the user.

Lastly, when the private key ensures that the messages sent by the user are not tampered with, if someone other than the users has the private key, the messages can be altered and still make them appear as if they are from the user.

The private key usage in the PKA can therefore be critical for device security. Hence it is important to make it as secures as possible. As described on previous slide, wrapped private keys can be stored anywhere, because they can only be used on this device. Also, the software that manipulates wrapped private keys can never access its value in the clear, making the overall application more robust against attacks.



STM32 key wrapping mechanism for public key accelerator

Embedded RAM

Any flash

Key storage

use

System flash
Nonvolatile secret 256-bit

RTC/ TAMP
Tamper protected

Hardware Key Bus

DHUK

XOR

BHK

Unwrapped/ decrypted

SAES peripheral

Hardware Key Bus

AES peripheral

CCB

PKA peripheral

Hardware secret keys, never visible in the clear by runtime applications or debugger

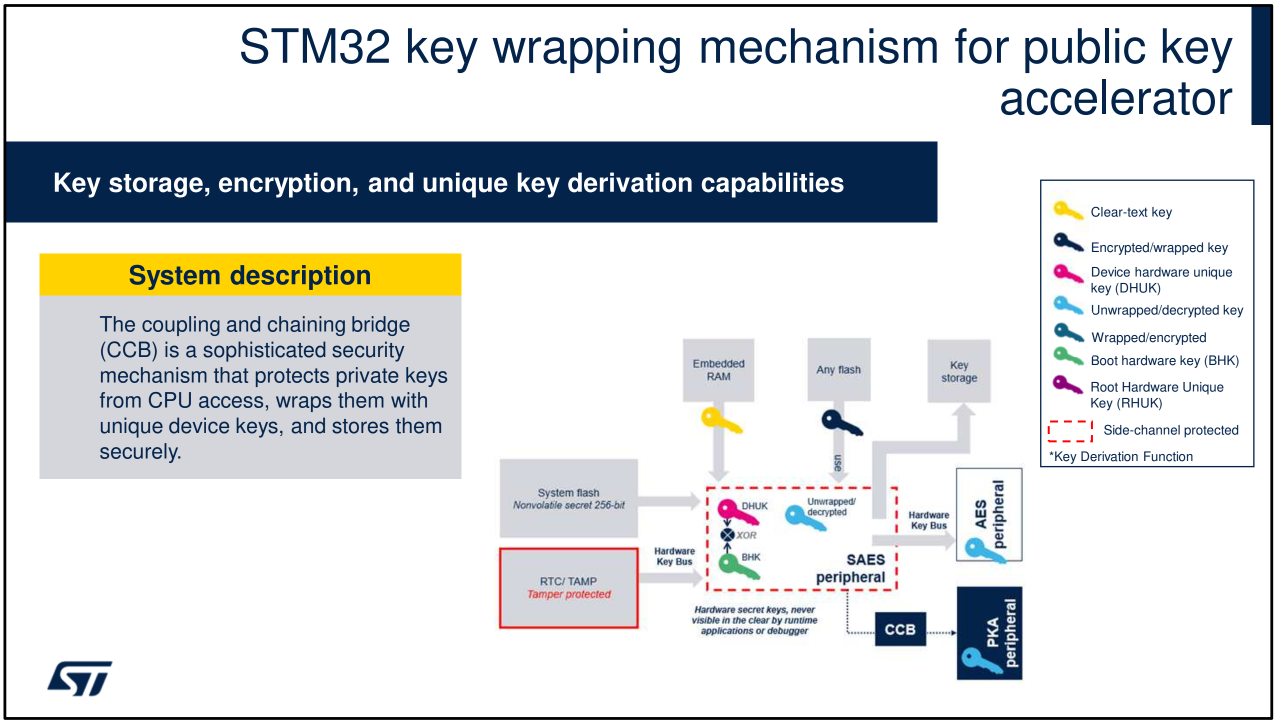
- Clear-text key
- Encrypted/wrapped key
- Device hardware unique key (DHUK)
- Unwrapped/decrypted key
- Wrapped/encrypted
- Boot hardware key (BHK)
- Root Hardware Unique Key (RHUK)
- Side-channel protected

*Key Derivation Function

To better protect keys, the side-channel protected SAES peripheral can use special hardware secret keys (DHUK, BHK) to ensure that critical application keys are never visible in clear text to the runtime application or the debugger.

Thanks to the Coupling and Chaining Bridge (CCB), this STM32 has a sophisticated security mechanism designed to ensure that private keys used in the PKA peripheral are protected using the same method as keys used in the SAES peripheral.

This module will detail the following use cases:
- Encrypting keys for PKA peripheral using the CCB
- Decrypting keys for PKA peripheral using the CCB



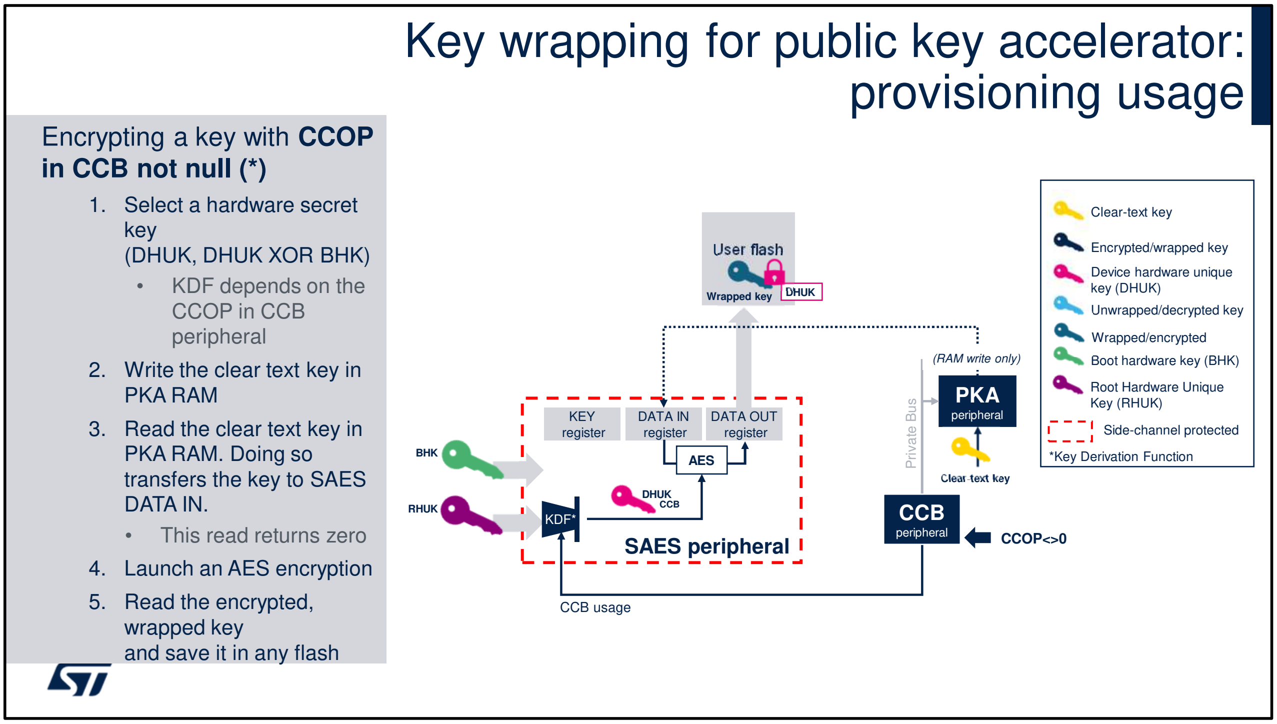
STM32 key wrapping mechanism for public key accelerator

Key storage, encryption, and unique key derivation capabilities

System description

The coupling and chaining bridge (CCB) is a sophisticated security mechanism that protects private keys from CPU access, wraps them with unique device keys, and stores them securely.

The coupling and chaining bridge (CCB) is a sophisticated security mechanism that protects private keys from CPU access, wraps them with unique device keys, and stores them securely.



Key wrapping for public key accelerator: provisioning usage

Encrypting a key with **CCOP in CCB not null (*)**

1. Select a hardware secret key (DHUK, DHUK XOR BHK)
   - KDF depends on the CCOP in CCB peripheral
2. Write the clear text key in PKA RAM
3. Read the clear text key in PKA RAM. Doing so transfers the key to SAES DATA IN.
   - This read returns zero
4. Launch an AES encryption
5. Read the encrypted, wrapped key and save it in any flash

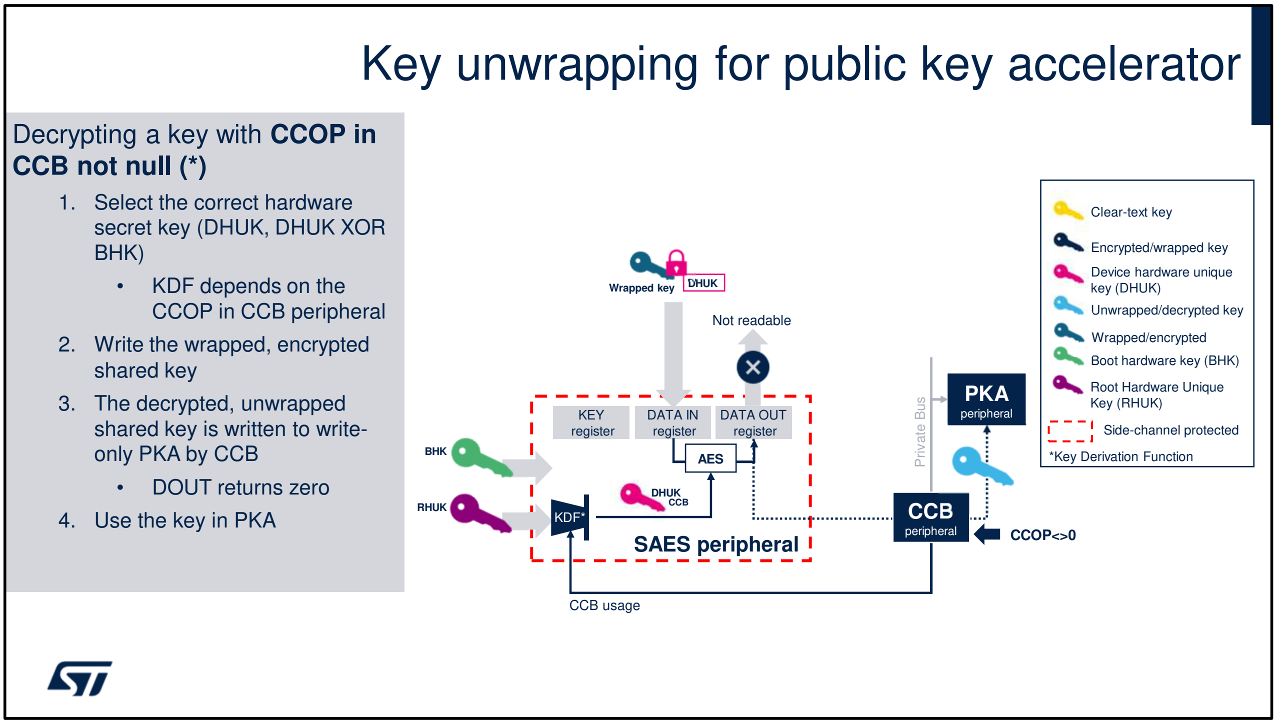
To encrypt a key using the CCOP in the CCB peripheral ,the process begins by selecting a hardware secret key, such as DHUK or DHUK XOR BHK. The Key Derivation Function (KDF) is dependent on the CCOP in the CCB peripheral.

Next, the clear text key is written into the PKA RAM. Once written, the clear text key is read from the PKA RAM, which transfers the key to the SAES DATA IN.

It is important to note that this read operation returns zero.

After this, an AES encryption is launched.

Finally, the encrypted (wrapped) key is read and saved into any flash memory for secure storage.

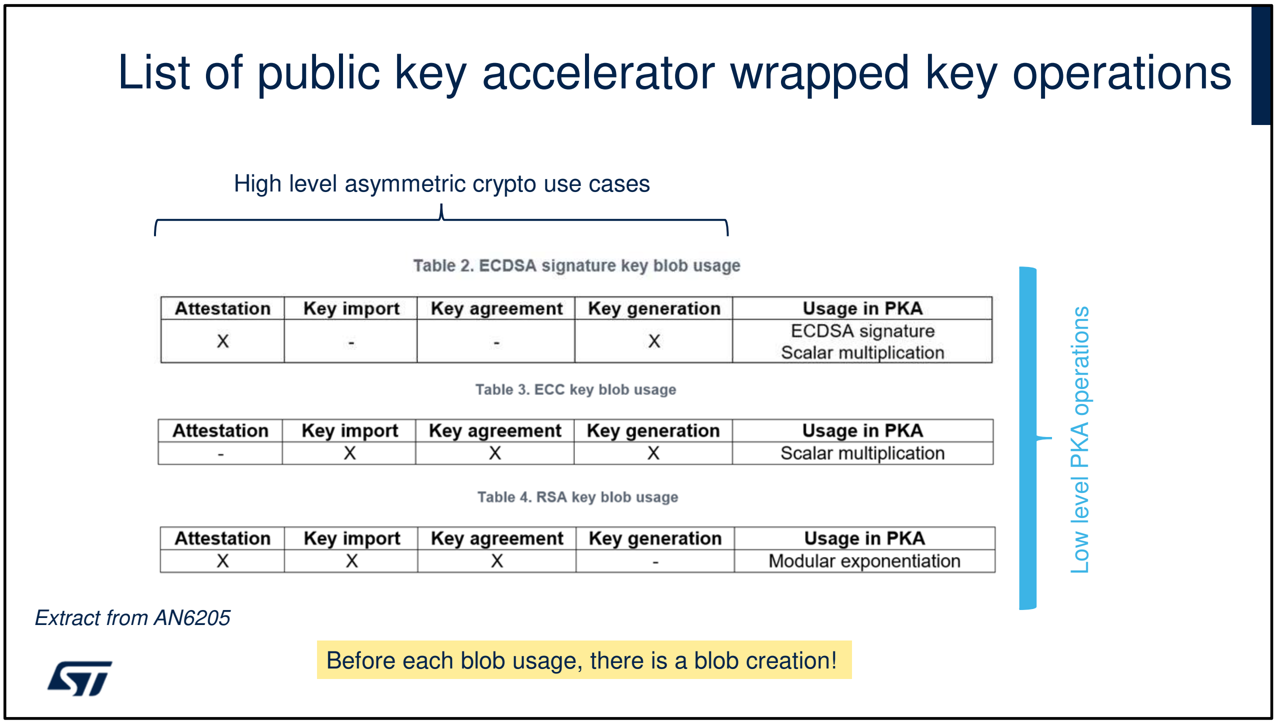


Key unwrapping for public key accelerator

Decrypting a key with **CCOP in CCB not null (*)**

1. Select the correct hardware secret key (DHUK, DHUK XOR BHK)
   - KDF depends on the CCOP in CCB peripheral
2. Write the wrapped, encrypted shared key
3. The decrypted, unwrapped shared key is written to write-only PKA by CCB
   - DOUT returns zero
4. Use the key in PKA

Before unwrapping a key for the PKA, you must select the CCOP code in the CCB that corresponds to the operation you want to perform in the PKA. The STM32 reference manual provides detailed instructions on the sequence of actions to take. But to summarize, to decrypt a key using the CCOP in the CCB peripheral (not null configuration), the process starts by selecting the correct hardware secret key, such as DHUK or DHUK XOR BHK. The Key Derivation Function (KDF) depends on the CCOP in the CCB peripheral. The next step involves writing the wrapped, encrypted shared key. The decrypted (unwrapped) shared key is then written to the write-only PKA by the CCB. During this process, the DOUT operation returns zero. Finally, the decrypted key is ready to be used directly in the PKA.



List of public key accelerator wrapped key operations

High level asymmetric crypto use cases

Table 2. ECDSA signature key blob usage

Attestation	Key import	Key agreement	Key generation	Usage in PKA
X	-	-	X	ECDSA signature Scalar multiplication

Table 3. ECC key blob usage

Attestation	Key import	Key agreement	Key generation	Usage in PKA
-	X	X	X	Scalar multiplication

Table 4. RSA key blob usage

Attestation	Key import	Key agreement	Key generation	Usage in PKA
X	X	X	-	Modular exponentiation

Low level PKA operations

Extract from AN6205

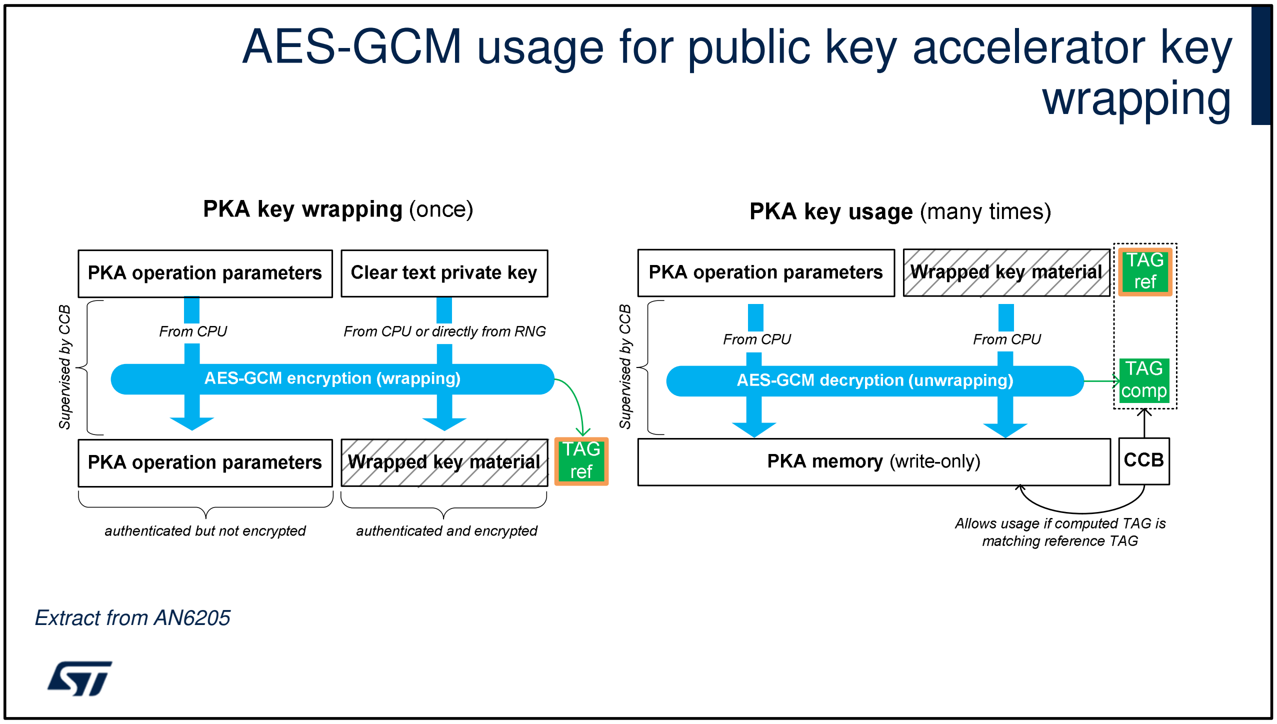
Before each blob usage, there is a blob creation!

The high-level asymmetric cryptography use cases described earlier corresponds to the low-level PKA operations detailed in this slide.

Wrapping a private key for ECDSA signatures corresponds to creating a ECDSA signature key blob. Its usage is summarized here.

Wrapping an ECC private key corresponds to creating an ECC key blob. Its usage is summarized here.

Wrapping an RSA private key corresponds to creating an RSA key blob. Its usage is summarized here.



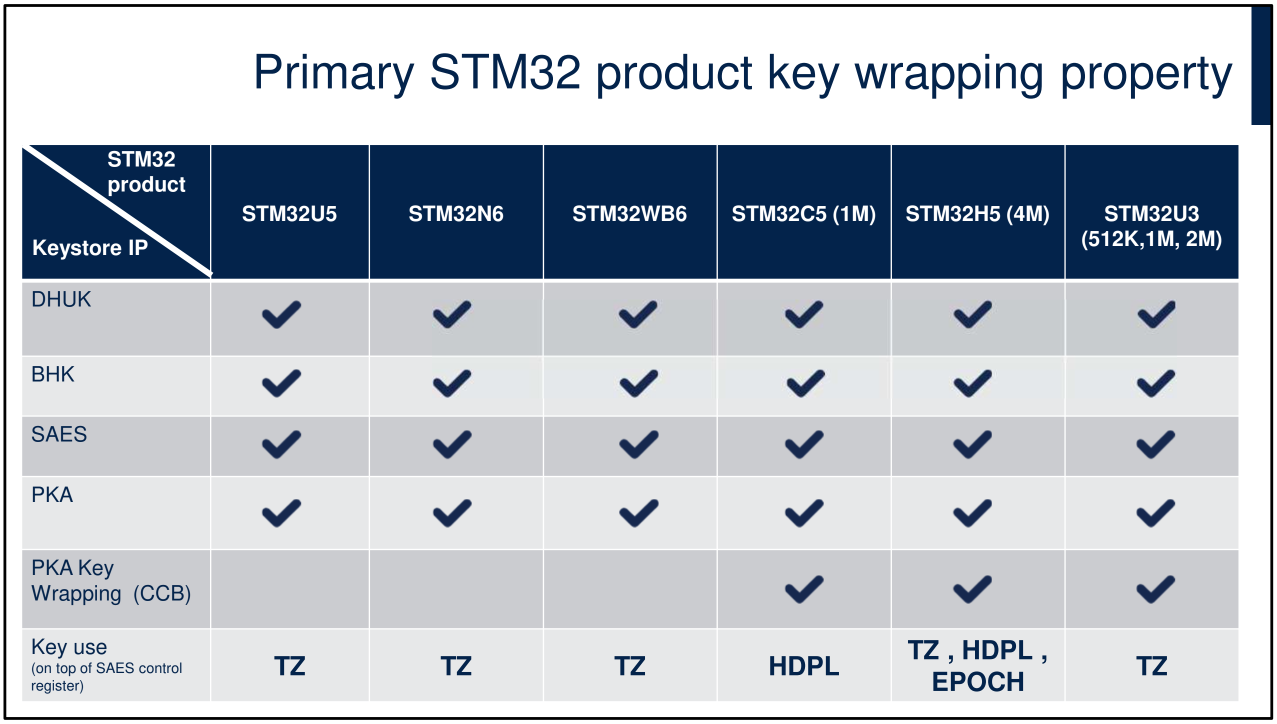
AES-GCM usage for public key accelerator key wrapping

PKA key wrapping (once)
PKA operation parameters
Clear text private key
Supervised by CCB
From CPU
From CPU or directly from RNG
AES-GCM encryption (wrapping)
PKA operation parameters
Wrapped key material
TAG ref
authenticated but not encrypted
authenticated and encrypted
PKA key usage (many times)
PKA operation parameters
Wrapped key material
TAG ref
Supervised by CCB
From CPU
From CPU
AES-GCM decryption (unwrapping)
TAG comp
PKA memory (write-only)
CCB
Allows usage if computed TAG is matching reference TAG

Extract from AN6205

AES-GCM is a widely used cryptographic method that provides both confidentiality and data integrity. It combines the AES encryption algorithm with the Galois/Counter Mode of operation to achieve authenticated encryption. The PKA key wrapping with the CCB uses the AES-GCM algorithm, with a key size of 256-bit.

Under the supervision of the CCB, the *PKA operation parameters* are only authenticated while the *key material* is authenticated and encrypted using AES-GCM. At the end of the wrapping (or encryption) process, the application must store the *reference tag* together with the operation parameters and the wrapped key material. Indeed, once the PKA wrapped key is decrypted in PKA memory, PKA can use it with the verified operation parameters only if the reference and the computed AES-GCM tags match.



Primary STM32 product key wrapping property

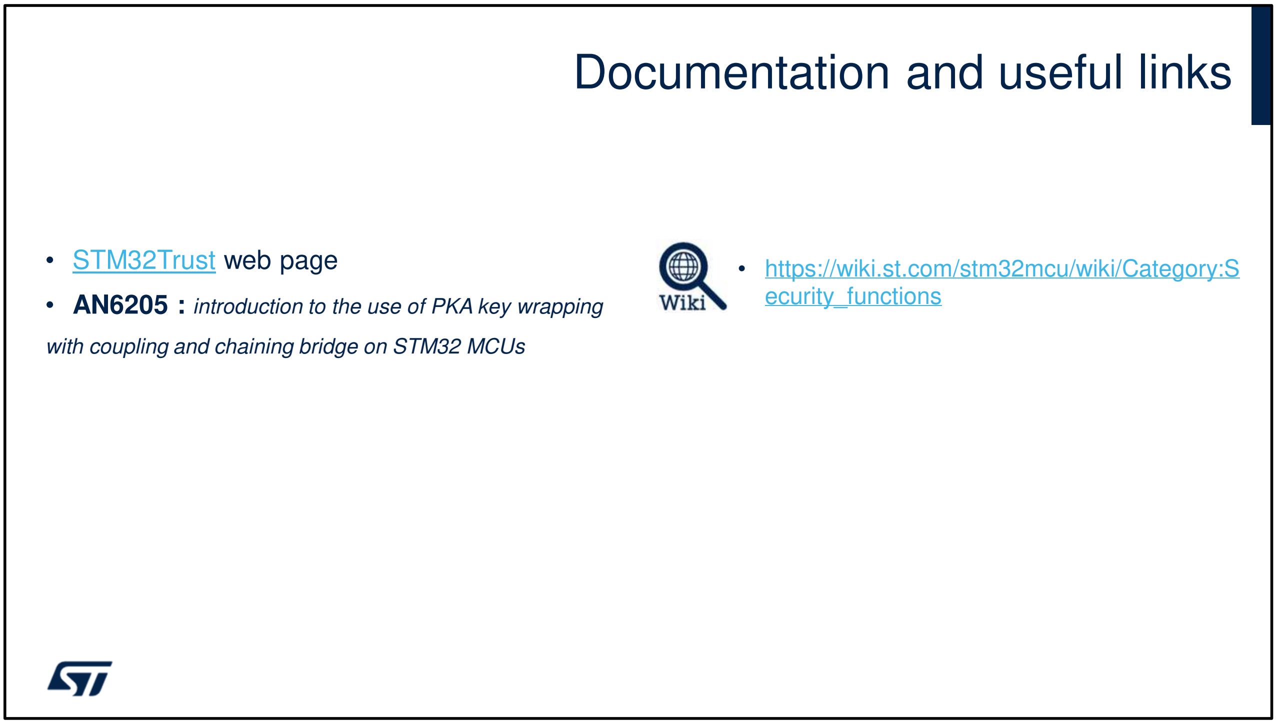
STM32 product / Keystore IP	STM32U5	STM32N6	STM32WB6	STM32C5 (1M)	STM32H5 (4M)	STM32U3 (512K,1M, 2M)
DHUK	✓	✓	✓	✓	✓	✓
BHK	✓	✓	✓	✓	✓	✓
SAES	✓	✓	✓	✓	✓	✓
PKA	✓	✓	✓	✓	✓	✓
PKA Key Wrapping (CCB)				✓	✓	✓
Key use (on top of SAES control register)	TZ	TZ	TZ	HDPL	TZ , HDPL , EPOCH	TZ

Here is a comprehensive overview of the STM32 product lineup and how key wrapping is managed across different devices using various Keystore IPs.

This slide presents a **panoply of the new STM32 products** and highlights the flexibility in managing key wrapping operations.

Each product leverages a different Keystore IP, such as DHUK, BHK, SAES, or PKA, depending on the device's capabilities and security requirements.

This differentiation ensures that each STM32 product is optimized for its intended application, offering tailored security features while maintaining robust key management practices.



Documentation and useful links

- STM32Trust web page
- **AN6205 :** *introduction to the use of PKA key wrapping with coupling and chaining bridge on STM32 MCUs*

- https://wiki.st.com/stm32mcu/wiki/Category:Security_functions

In this presentation, we delved into the advanced security features of the STM32U3 microcontroller, focusing on its **secure keystore capabilities** and the innovative **key wrapping techniques** enabled by the Coupling and Chaining Bridge (CCB). We explored how STM32 products provide **flexible and robust key management solutions**, leveraging different Keystore IPs (such as DHUK, BHK, SAES, and PKA) to meet the specific needs of various applications.

A key highlight of this presentation was the **provisioning, wrapping, and unwrapping features** offered by the STM32U3. The provisioning process ensures that keys are securely initialized and stored in the device. The **key wrapping mechanism**, facilitated by the CCB, allows sensitive keys to be securely encrypted and transferred, while the **unwrapping process** ensures that keys are securely decrypted and made available for cryptographic operations. These processes are designed to be highly robust, leveraging hardware-based security features to protect against physical and software-based attacks, ensuring the confidentiality and integrity of cryptographic keys throughout their lifecycle.

For further details and implementation guidance, refer to the **documentation and useful links** provided below:

to the application note **AN6205** *Introduction to the use of PKA key wrapping with coupling and chaining bridge on STM32 MCUs.*

This application note introduces the use of PKA key wrapping with the CCB peripheral and answers the following questions:

- Why do we need PKA key wrapping?
- What are the cryptographic functions involved in wrapping PKA keys?
- What set of functions does STM32CubeMX propose to run these updated cryptographic functions?

Our technology starts with You

Find out more at www.st.com