Developing supply chain confidence with TruST25 digital signature in RFID/NFC tags

Introduction

Supply chain inevitably includes the grey market and the risk of counterfeiting. The use of RFID/NFC tags is a widely used solution against these threats.

RFID/NFC tag customers take for granted that the tag will be working in all expected conditions of its expected lifetime. This is possible when it has been carefully manufactured according to standards matching the quality expectations.

However, before it reaches the end-customer, an RFID/NFC tag undergoes a rather long manufacturing process: from the silicon manufacturer to the inlay or raw material maker, to the system integrator and then to the end-product assembler. Each step involves different companies, with all the logistic associated.

Unfortunately, counterfeiting may come up in each of these phases. And because business requires trust, a commercial delivery requires guarantees. How to prove that a tag embeds the expected silicon with the expected quality? How can the customer prove that a tag is legitimate?

This is where cryptography comes in, with a solution to check (easily and reliably) the authenticity of the origin of an RFID/NFC tag: the digital signature. The digital signature has plenty of applications in business, among them the bitcoin and the cryptocurrencies.

This document describes applications of digital signature in RFID/NFC tags.

The use of the TruST25 digital signature, possible with the ST25 NFC / RFID Tags and ST25 Dynamic NFC Tags embedding this feature, enhances the security level of trust between all the players in the supply chain.

1 Applications of digital signature in RFID/NFC tags

RFID/NFC technology natively provides the key elements to identify itself, as each tag features a Unique Identifier (UID). The UID can easily be traced back to the silicon manufacturer and all along the supply chain, Clearly this requires that the UID be issued by the legitimate silicon manufacturer. As an example, if a silicon foundry manufactures RFID/NFC chips using UIDs assigned to another manufacturer, it opens the door to counterfeiters. Counterfeited tags and counterfeited end-products may enter the market and take a share of the business.

Thanks to the digital signature it is possible to build a cascade of trust either by adding a signature in the tag user area, or by re-using the digital signature of the UID.

1.1 Check silicon origin using UID digital signature

Each RFID/NFC tag contains a unique identifier (UID) written by the silicon manufacturer before the chip is shipped. In compliance with RFID/NFC specification, this UID is readable by any RFID reader or NFC-enabled phones. The UID embeds a manufacturer code assigned by ISO/IEC JTC1/SC17.

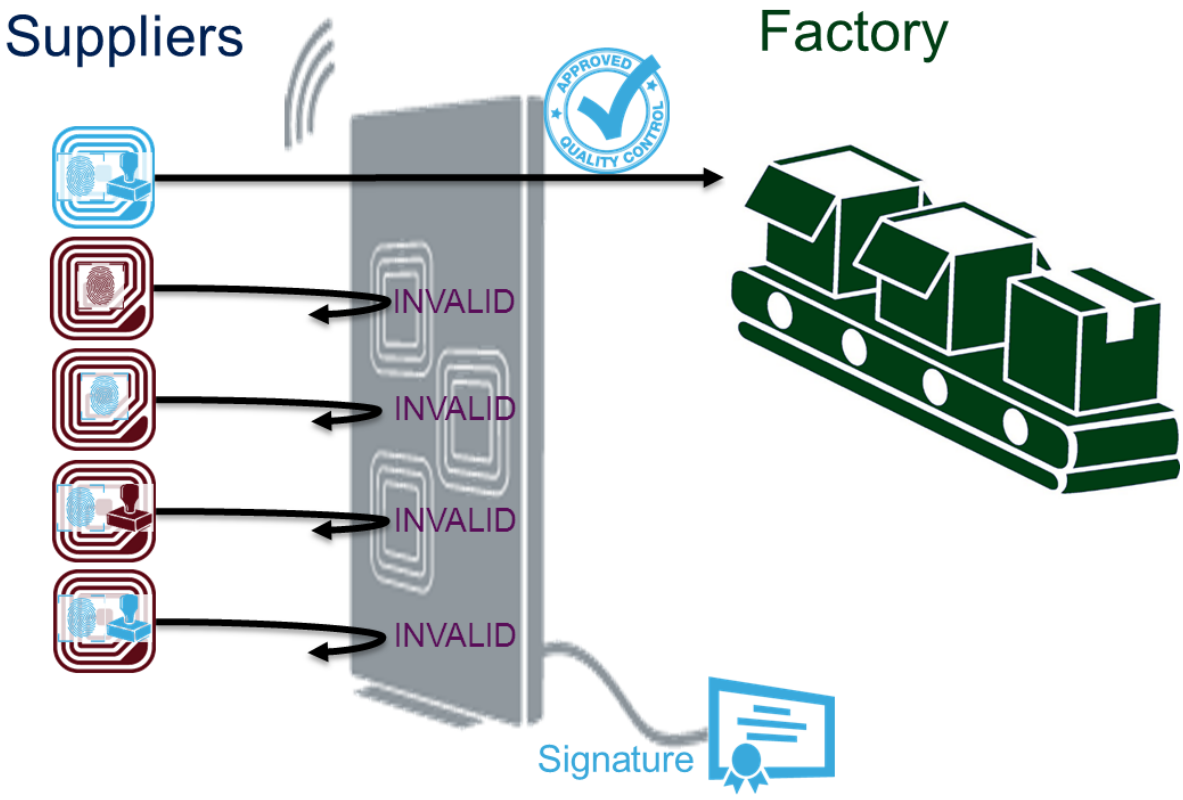
In addition to this information, some RFID/NFC chip models embed a digital signature and support commands to read it from a reader or phone. With this feature, the silicon manufacturer provides a mean to its customer to distinguish between its chips and counterfeited ones.

For an RFID/NFC chip issued by manufacturer A counterfeiting legitimate manufacturer B there are four cases.

1. The chip features a UID not assigned to legitimate manufacturer B. In this case, simply reading the UID will discover the substitution.
2. The chip features a UID assigned to manufacturer B, but not implementing digital signature proprietary commands. In this case, when the digital signature is requested, the chip will not answer, thus revealing the counterfeiting.
3. The chip features a UID assigned to manufacturer B, and implements digital signature proprietary commands but with a digital signature not issued by manufacturer B. In this case, by checking the digital signature, the reader can determine if the UID has really been issued by the silicon manufacturer B.
4. The chip features a UID assigned to manufacturer B, and implements digital signature proprietary commands with a copy of a digital signature issued by manufacturer B. In this case, by checking the uniqueness of the UID, the reader can determine that the UID is a duplicate – hence the chip is a fake.

The counterfeit detection enabled by digital signature is shown in the next figure.

Figure 1. **Detection of counterfeiting**



Thanks to the digital signature of UID, if, for example, the silicon manufacturer customer is an inlay maker, it may sell its inlays with a guarantee that the embedded silicon chips are genuine. This leads to an increase of trust, and then of business.

Following this example, the inlay maker may also use digital signature to provide a mean for its customer to check the authenticity of the inlay. There are two different ways to do it, described in Section 1.1.1 and Section 1.1.2 .

1.1.1 Create a cascade of trust using signature in user area

RFID/NFC tags are essentially non-volatile memories where data can be stored. Until this memory is actually initialized for its end-user application, it may be used to store information useful to the manufacturing supply chain. If a supplier writes a digital signature using its own key in the tag, any customer may verify it so it defeats introduction of counterfeiting in the supply chain.

This method can be used with any tags providing at least 512 bytes of writable memory. If the tag memory is big enough to hold the signature and the end-user information, a signature may be used in the supply chain until the good reaches the end-user.

Unfortunately, it is likely that somewhere in the supply chain the definitive content of the tag (excluding a digital signature) has to be written. Thus writing the digital signature in user area is a compelling solution, as long as this is possible.

1.1.2 Create a cascade of trust using UID digital signature

When the silicon manufacturer already provides the UID signature, all actors of the supply chain can exploit this feature to defeat counterfeiting.

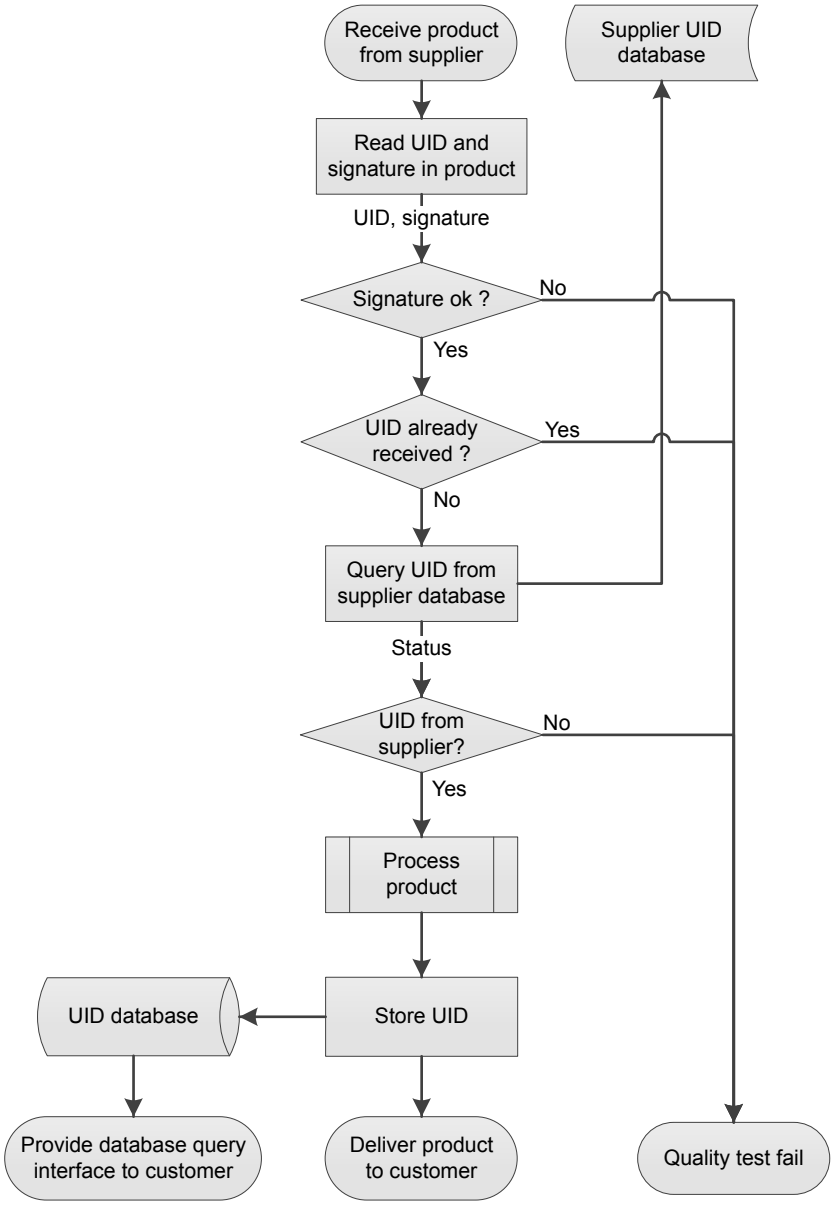
For this purpose, the entity receiving the product must

1. Read the UID and signature of incoming product
2. Verify the signature: if it fails the incoming silicon doesn't come from expected manufacturer, most likely is a counterfeited product.

3. Query the expected supplier database with the UID of the product received. If the query passes, the product has been delivered by the expected supplier. If the UID is not in supplier database whereas the digital signature is genuine it means that the product uses silicon from the same silicon manufacturer but hasn't been processed by the expected supplier. It is likely to be a counterfeited product.
4. Once the product has been processed and is ready to be delivered, store its UID in a database shared with customers. This will allow customers to verify the origin of the product.

The flow chart in Figure 2. Use of digital signature to create a cascade of trust describes the process:

Figure 2. Use of digital signature to create a cascade of trust



2 Understanding the digital signature

2.1 What is a digital signature? How is it verified?

A digital signature is a piece of data that can be processed to guarantee both data integrity of the signed content and authentication of the origin (signer).

For example, Renaldo reads a message and needs confidence that this exact message has been issued by Talia. For this purpose, Talia has provided to Renaldo a verification key and added a digital signature separated from the message. With the key and the signature, Renaldo can determine if Talia has issued the message he reads.

To be complete, the digital signatures has a third property named non-repudiation: if Renaldo has received a message passing the verification with Talia's verification key, Talia can't deny the message while claiming she's the only owner of the signatory key.

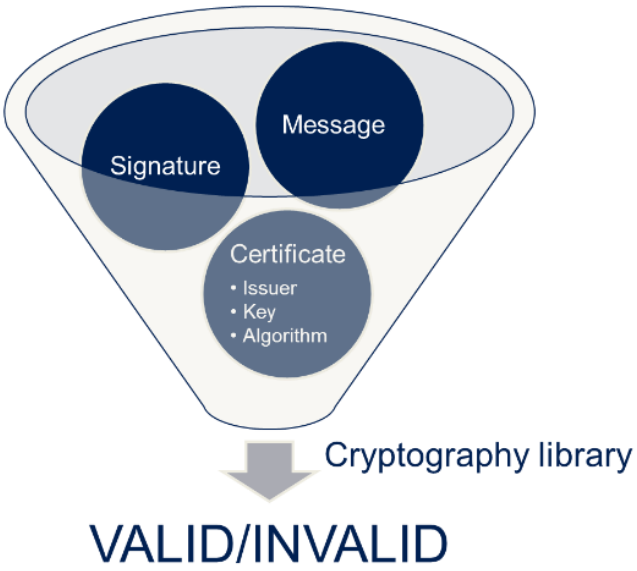
This is made possible by cryptography algorithms. Their specification is public, it describes the mathematical operations and processes to follow to generate a digital signature and to verify it.

Because the implementation requires a lot of attention in details, dedicated libraries are available to perform the cryptographic processing. Most of these libraries are listed and compared in [8]. In addition, all operating systems implement native security frameworks (such as Android™ java.security, Apple® Security Framework, or Microsoft® CNG).

The use of a cryptography library is straightforward: inputs are the message, the signature, the key and the cryptographic algorithms used, while the output is a pass/fail status.

Note that the verification is not a decryption process: the processing of the digital signature doesn't output meaningful data.

Figure 3. Digital signature verification procedure



The message is a sequence of bits of any size. For RFID/NFC tags digital signature, it contains the UID of the tag (usually 7 bytes).

The cryptographic algorithm chosen by the signatory fixes the size of the signature. Increasing the number of bits increases its theoretical security level (assuming there is no flaw somewhere else). Cryptographic algorithm are described in Section 2.8 The digital signature algorithm.

The certificate is a piece of data provided once by the signatory (Talia) to the verifier (Renaldo). It embeds several information such as the key, the algorithm used and the issuer (among others). It is not strictly required to check a signature: what is required is the key and the algorithm. However, it is almost always used because it is a container allowing easy transfer, storage and verification of the key so it increases the overall security of the signature asset. It is described in the following section.

The trustability of the key is a critical issue to understand: If a malicious entity delivers a message with a signature not signed by the legitimate source (Talia) but by itself, this will be detected by the reader (Renaldo) given that the reader uses Talia's key in the verification process. However, if the malicious entity manages to introduce its key instead of Talia's key in the reader verification process, the digital signature verification will pass because the forged messaged is signed by the forged key. Consequently, the management of the key is very important.

The following section provides more details on the certificate and its content.

2.2 What is a certificate?

The certificate is a software container used to manipulate the key. In addition to the key bitstream, it usually embeds:

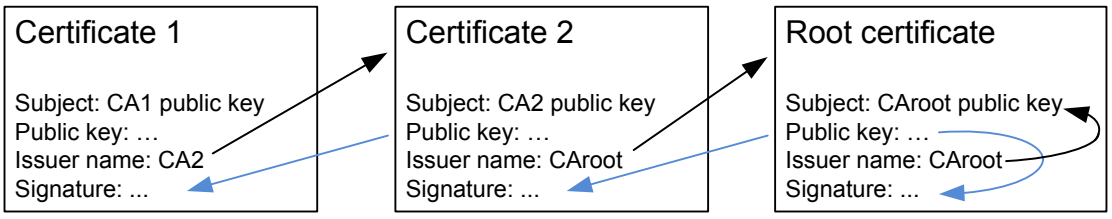
- the name of the key owner (issuer) so that the correct key can be selected where needed
- an identifier of the cryptographic algorithm to use with the key
- the name of the certificate authority that has issued the certificate
- the digital signature of the certificate content
- other fields such as validity timeframe, subject, serial number.

The certificate itself is signed so that its content can be verified using the key of the certificate issuer. This is important because the digital signature verification makes sense only if it uses a genuine key and not a forged one.

If the key of the certificate issuer is received in another certificate, this certificate has to be verified using the key of its certificate issuer. At the root of this chain, there should be a certificate that is embedded in the software or hardware and trusted without verification.

The certificate issuer is named "Certificate Authority" (CA), and the infrastructure set-up to issue, revoke and validate the certificates is named "Public Key Infrastructure" (PKI).

Figure 4. Chain of trusted certificates



The certificate must be in a known, standard format so that it can be easily transferred between computers and processed by any program. The most widely used standard is ITU-T X.509, encompassing not only the certificate format but also its usage and the PKI.

Security standard defines requirements on the PKI: the keys must be created, conducted and saved in a secure manner to protect the signature asset.

2.3 What is a keychain / keystore?

As explained, an application or operating system doing cryptography will have to manage a key in a certificate. In today's connected world, it is likely that there will be several certificates corresponding to several peers. In addition, it is very important to validate certificates to use the genuine key corresponding to a given peer. That's why security frameworks have developed keystores (also named keychains), which simplify the management of keys and certificates from an application and user point of view.

Example of keystore frameworks are BouncyCastle keystores (BKS, UBER, BKCS, BCFKS), Java™ KeyStore (JKS), Android Keystore, iOS Keychain, Microsoft “Cryptography API: Next Generation” (CNG). Microsoft Windows® certification store can be browsed with the tool “certmgr.msc”.

2.4 Why is the key “public” or “private”?

A valid digital signature must provide to the recipient guarantee that a message is received from an issuer (authenticity) without modification (integrity).

For this purpose, it is usually based on asymmetric cryptography: the signature is generated with a private key that only the issuer owns, while it is verified using a public key that can be distributed to anyone.

The private key and public key are generated together and can’t be dissociated: only the public key generated with the private key may validate a signature generated by a given private key, and only the private key generated with the public key may have generated a signature verified by a given public key.

Both keys are generated only once (and together) by the signatory (or its Certificate Authority, see Section 2.2 What is a certificate?). The signatory then uses the same private key to generate signatures of all devices. This private key must remain secret because if it is leaked, any signature validated by the corresponding public key doesn’t prove anything: it may have been signed by anyone owning a copy of the private key.

The signatory distributes the public key to people entitled to verify its messages. Anyone may know the public key without concern to the reliability of the digital signature.

2.5 Is it possible to keep the private key secret?

We have described the importance of the private key, which must be used to sign every silicon chip, and must be kept secret. This section will describe the equipment solving this dilemma in manufacturing process without security trade-offs.

A hardware security module (HSM) is a device that safeguards and manages digital keys and offer cryptoprocessing services. It provides protections against both hardware and software tampering. It is certified by security assessment standards such as Common Criteria or FIPS-140.

The equipment may generate cryptographic keys, store them, and perform digital signature using the private key. The benefit is that the private key never leaves the secure hardware and nobody can copy it. HSM is the root of digital signature writing in a manufacturing process without compromising security.

2.6 How does it work?

The process of generating and verifying a digital signature is described in detail in open standards, so that anyone can implement best practices and/or check the robustness of the algorithm. This section will present the process from a global point of view.

Weaknesses are hidden in details overviewed by this presentation. This is the reason why it is recommended to use certified frameworks or HSMs to generate and verify digital signatures.

2.7 The hash / message digest

First, the message to be signed, which can be of any size, must be “summarized” in a message digest of known fixed size. This process is referred to as “hash function” and the output is the “hash”.

This cryptographic hash function must have the following properties:

- It is easy to compute the hash value from a message, but it is extremely difficult to find a message corresponding to a given hash value. This one-way property is named “pre-image” resistance.
- The probability to have two different messages resulting in the same hash message is extremely low. This property is named collision resistance.
- A small change in the message results in a dramatic change of the hash value (making it almost random).

The well-known hash functions are MD5, SHA-0, SHA-1, SHA-2 and SHA-3 corresponding to generations of algorithm published. MD5, SHA-0 and SHA-1 are obsolete functions, not to be used for security purpose such as digital signature process. SHA-2 published in 2001 has several variants, some of them being vulnerable to attacks while others are still immune. SHA-3 published in 2015 is currently the safer cryptographic hash function.

The algorithms usually have variants generating a hash value with different number of bits. For example, SHA3-224 generates a hash value of 224 bits and SHA3-512 generates a hash value of 512 bits. A longer hash results in a higher security level, at the cost of processing resources.

2.8 The digital signature algorithm

The digital signature algorithm transforms the message digest into a digital signature that can be verified with a public key. A digital signature algorithm must ensure that:

- it is possible to verify a signature/message pair with a public key
- it is computationally infeasible to construct another message with the same signature
- it is computationally infeasible to construct a signature validated by a public key without the corresponding private key.

2.8.1 Strength of an algorithm

Cryptographic algorithms for digital signature are making use of known limitations of digital computations to guarantee a given level of security named “strength of the algorithm” and measured in bits. A strength of 128 bits means there are 2^{128} values to try to find the solution (forge a signature or a message or retrieve the private key). Of course, if there’s a flaw in the implementation of the generation of the digital signature, hackers will take advantage of the flaw and the figure of the theoretical security strength will not reflect the actual security level anymore.

Because the available computation power always increases over time, the recommended minimum number of bits of security strength level also increases over time.

2.8.2 Elliptic curve digital signature algorithm (ECDSA)

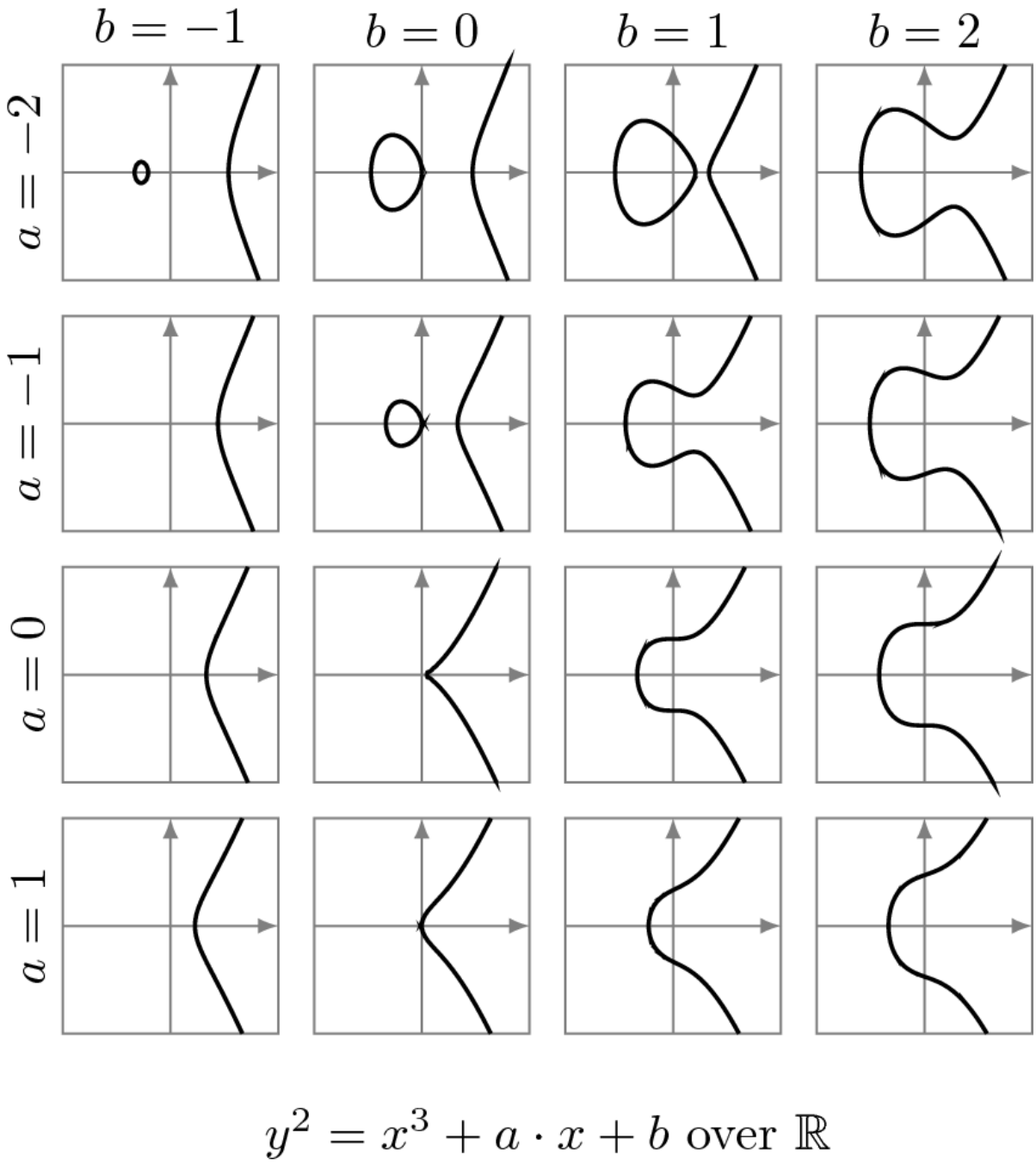
Current consensus is to use elliptic curve cryptography (ECC) for digital signature. ECC was invented by Neal Koblitz and Victor Miller in 1985, it is based on the resolution of a geometrical problem on elliptic curves.

Considering points solving the specific equation $y^2 = x^3 + ax + b$ in a field of finite elements with the required math defined, given a point P and an integer k it is easy to compute the coordinates of the point Q = kP. Given the points P and Q, it may be very (or extremely) difficult to find k. This is known as the discrete logarithm problem.

The elliptic curve digital signature algorithm (ECDSA) has been first specified in [6] by ANSI for the financial service industry. This document defines the process of generating and verifying a digital signature using ECC. It defines two family of curves (prime field and binary field), for each family some domain parameters are open for the implementation choice.

Figure 5 illustrates the influence of parameters a and b on elliptic curves.

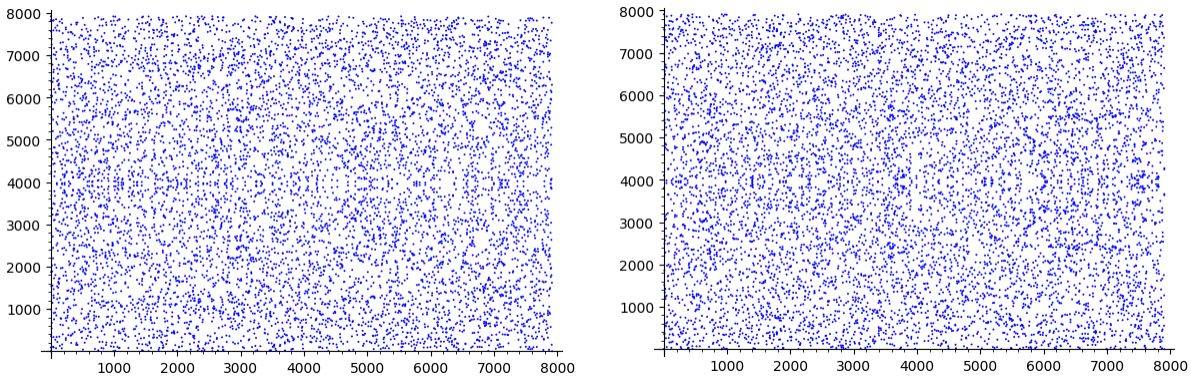
Figure 5. **Elliptic curves for different values of a and b parameters**



The previous illustration is over $\mathbb{R}$ (the field of real numbers), which is a “human-readable” version, but not actually used by computations. These are usually carried out over a finite field whose size is another parameter defined in [6].

The next figure illustrates the difference between points solving $y^2 = x^3 + x + 2$ over finite fields $\mathbb{Z}_{7919}$ (left side) and $\mathbb{Z}_{7907}$ (right side).

Figure 6. Solutions of $y^2 = x^3 + x + 2$ over different finite fields



The size of the mathematical finite field used has a special relevance for the user, because it defines the theoretical level of security, at the cost of processing and storage resources. As an example, a field size p of 256 bits leads to a security strength of 128 bits and a digital signature size of 512 bits (64 bytes). An advantage of ECC is that the number of bits is just a parameter of a general methodology, so it is possible to adapt the security / resource trade-off without changing the algorithm implementation.

A bad choice of parameters will lead to a weak security of the generated digital signature. That's why several standards for digital signature are based on ([6]), with additional restriction on domain parameters and additional requirements in the process to increase interoperability and robustness of digital signature.

A branch of [6] is the "Standards for Efficient Cryptography Group", or SECG. It defines itself as "an industry consortium, […] founded in 1998 to develop commercial standards that facilitate the adoption of efficient cryptography and interoperability across a wide range of computing platforms. SECG members include leading technology companies and key industry players in the information security industry".

In 2000 SECG has published [3] and [4]. The latter uses nicknames for each defined set of parameters (such as "secp256k1", "secp521r1" or "sect409k1") used in certificates to identify the algorithm.

For example, secp256k1 is defined to use elliptic curve $y^2 = x^3 + 7$ over a finite field F_p of dimension $p = 2^{256} - 2^{32} - 2^9 - 2^8 - 2^7 - 2^6 - 2^4 - 1$ (and other parameters not detailed in this presentation).

In parallel to SECG, USA National Institute of Standards and Technology (NIST) published [2], where it defines the ECDSA parameters to use to be compliant, and the process to generate a strong signature.

DSS and SECG specifications are fully compatible with ECDSA as defined by [6]. Both are providing detailed steps to implement to generate and verify a digital signature.

In summary, even if the algorithm used for digital signature is fixed, it takes as input some parameters that are normally chosen among predefined sets. It is up to the signatory to choose the parameters. First choice is the size of the digital signature, which is a compromise between strength and resources. Then, the set of parameters to be used within this size may be chosen according to the standard curves available in security framework.

3 TruST25 digital signature

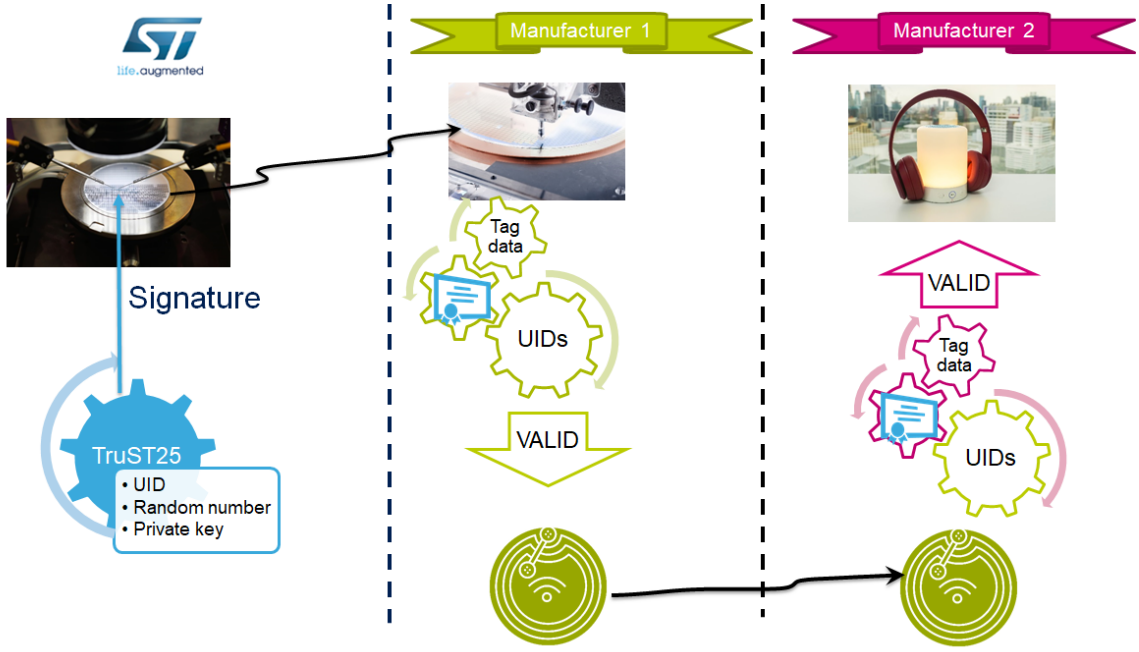
TruST25 digital signature is the solution proposed by STMicroelectronics, encompassing industrialization processes and tools deployed to provide a digital signature of the Unique Identifier (UID) embedded in the ST25 RFID/NFC tags.

STMicroelectronics has deployed TruST25 digital signature with the highest security standards, aligned with those of the secure MCU products, such as storing private keys and generating signature in a certified HSM in a secure room.

Moreover, STMicroelectronics guarantees uniqueness of UID across all ST25 tags, making it possible to catch clones.

The use of TruST25 digital signature raises confidence all along the supply chain. The figure below summarizes the cascade of trust enabled by TruST25 digital signature.

Figure 7. Cascade of trust based on TruST25



TruST25 digital signature is an enabler for confidence in the supply chain, provided a cascade of truth is set from silicon manufacturer to end customers. Created in house through cryptography tools and secure environment, it needs no secure environment to be verified.

In addition to the supply chain, it can be used in other cases, such as traceability for tracking products in the delivery journey, and as anti-counterfeiting where users/consumers can have a proof of origin.

The same HF tag used with RFID readers during supply chain can be used by consumers with NFC phones. This bridge engages consumers by offering new services and feedback relevant information to the brands. Here too, use of TruST25 digital signature brings trust up to the consumer level.

The introduction of this feature in ST25 NFC/RFID Tags and ST25 NFC Dynamic Tags allows the industry and consumers to benefit from RFID/NFC technology.

4 References

[1]	National Institute of Standards and Technology	FIPS 180-4	Secure Hash Standard (SHS)	August 2015
[2]	National Institute of Standards and Technology	FIPS 186-4	Digital Signature Standard (DSS)	July 2013
[3]	Standards for Efficient Cryptography Group	SEC 1-v2	Elliptic Curve Cryptography	May 21, 2009
[4]	Standards for Efficient Cryptography Group	SEC 2-v2	Recommended Elliptic Curve Domain Parameters	September 20, 2000
[5]	Internet Engineering Task Force	RFC 5280	Internet X.509 Public Key Infrastructure Certificate and Certificate Revocation List (CRL) Profile	May 2008
[6]	American National Standard Institute	X9.62	Public Key Cryptography For The Financial Services Industry: The Elliptic Curve Digital Signature Algorithm (ECDSA)	November 16, 2005
[7]	Andrea Corbellini		Elliptic Curve Cryptography: finite fields and discrete logarithms	May 23, 2015
[8]	www.wikipedia.org		Comparison of cryptography libraries	

Revision history

Table 1. **Document revision history**

Date	Revision	Changes
05-Mar-2019	1	Initial release.
02-Feb-2021	2	Updated Section Introduction. Removed former Table 1. ST25 devices embedding TruST25™digital signature. Minor text edits across the whole document.

Contents

List of tables

List of figures