CRYPTOGRAPHY: FUNDAMENTALS ON THE MODERN APPROACH

By: Zia A. Sardar

Abstract:

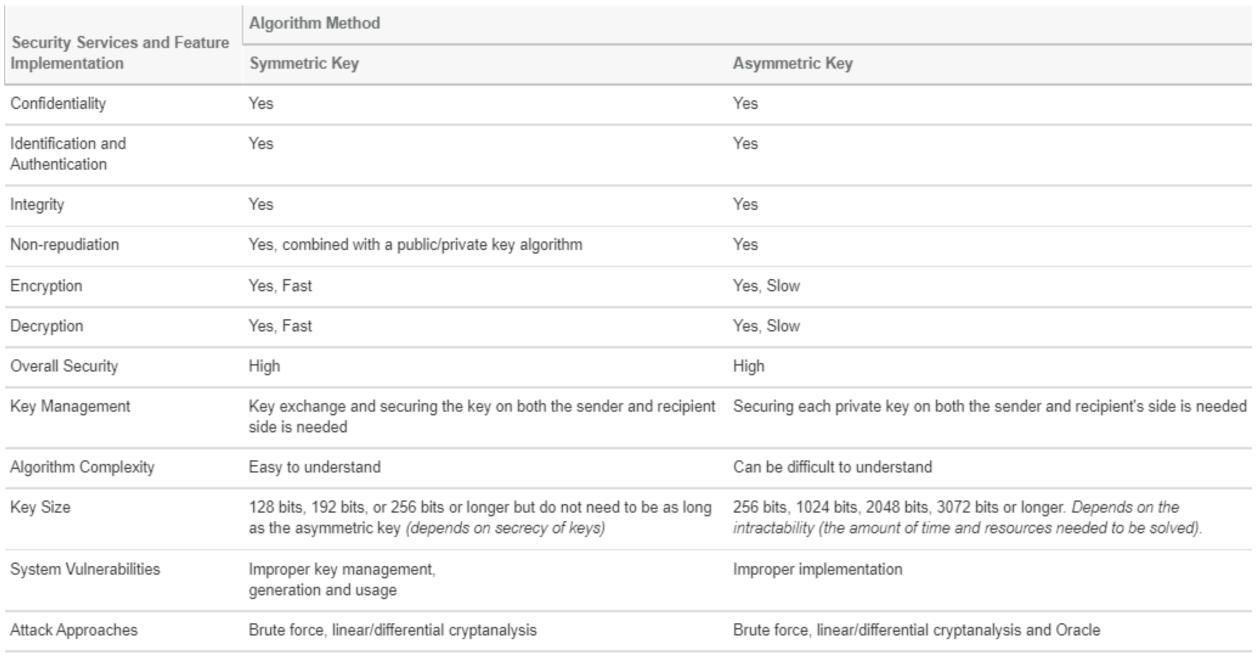
This installment is part of a series of application notes on cryptography. It is designed to be a quick study guide for a product development engineer and takes an engineering rather than theoretical approach. In this segment, we'll discuss the fundamental concepts behind modern cryptography. A similar version of this application note originally appeared on April 16, 2020, on Electronic Design.

Cryptographic Keys

Keeping cryptographic applications secure relies upon symmetric keys and private keys that are continually kept secret. The method used to keep them secret is also protected.

Asymmetric keys and symmetric keys are two basic types of algorithms that are used in modern cryptography. Asymmetric key algorithms use a combination of private and public keys while symmetric algorithms use only private ones, commonly referred to as secret keys. **Table 1** provides a snapshot of the main features of each algorithmic method.

Table 1. Cryptographic Algorithm Comparison



Security Services and Feature Implementation	Algorithm Method	
	Symmetric Key	Asymmetric Key
Confidentiality	Yes	Yes
Identification and Authentication	Yes	Yes
Integrity	Yes	Yes
Non-repudiation	Yes, combined with a public/private key algorithm	Yes
Encryption	Yes, Fast	Yes, Slow
Decryption	Yes, Fast	Yes, Slow
Overall Security	High	High
Key Management	Key exchange and securing the key on both the sender and recipient side is needed	Securing each private key on both the sender and recipient's side is needed
Algorithm Complexity	Easy to understand	Can be difficult to understand
Key Size	128 bits, 192 bits, or 256 bits or longer but do not need to be as long as the asymmetric key *(depends on secrecy of keys)*	256 bits, 1024 bits, 2048 bits, 3072 bits or longer. *Depends on the intractability (the amount of time and resources needed to be solved).*
System Vulnerabilities	Improper key management, generation and usage	Improper implementation
Attack Approaches	Brute force, linear/differential cryptanalysis	Brute force, linear/differential cryptanalysis and Oracle

Let's take a look at how we can achieve each of the cryptographic goals using these two types of algorithms.

Confidentiality Using Symmetric Key Algorithms

The main goal of confidentiality is to keep information away from all who are not privy to it. In a symmetric key cryptographic system, this is very straightforward and is achieved by encrypting the data that is exchanged between the sender and the recipient. Both the sender and the recipient have access to the same secret key that is used to encrypt and decrypt the exchanged message, as shown in **Figure 1**.

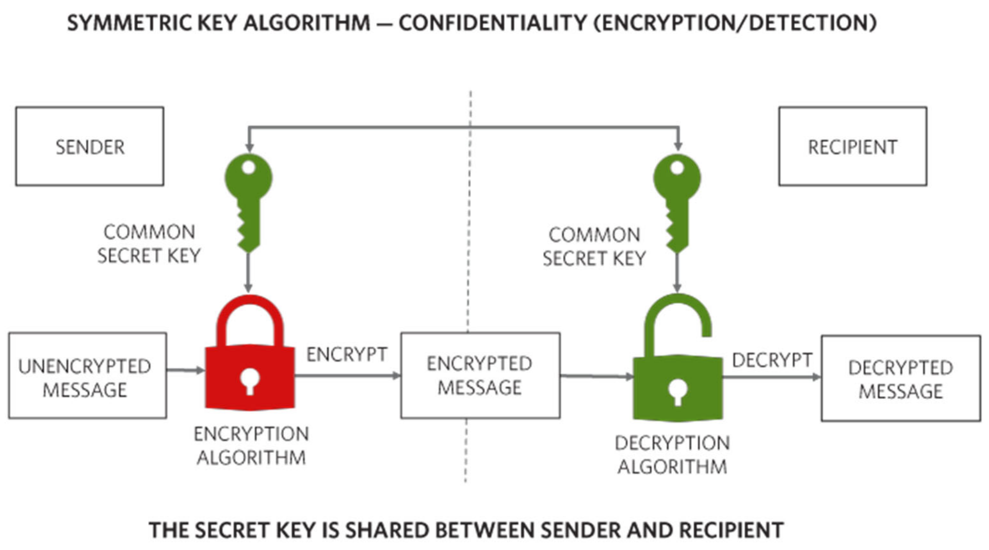


Figure 1. Symmetric key algorithms help achieve confidentiality using private or secret keys.

As long as the key is secured and only the sender and the recipient have access to the encryption/decryption key, no one else can receive the transmitted message even if it is intercepted mid-transmission. Thus, the message stays “confidential.”

Confidentiality Using Asymmetric Key Algorithm

In an asymmetric key system, the recipient freely distributes her/his public key. The sender acquires the public key and verifies the authenticity of it. There are a few steps, as shown in **Figure 2**, that are required to accomplish this. To keep things simple, let’s assume that the sender has access to the verified public key of the recipient. The sender then uses that public key to encrypt the message and sends it to the recipient.

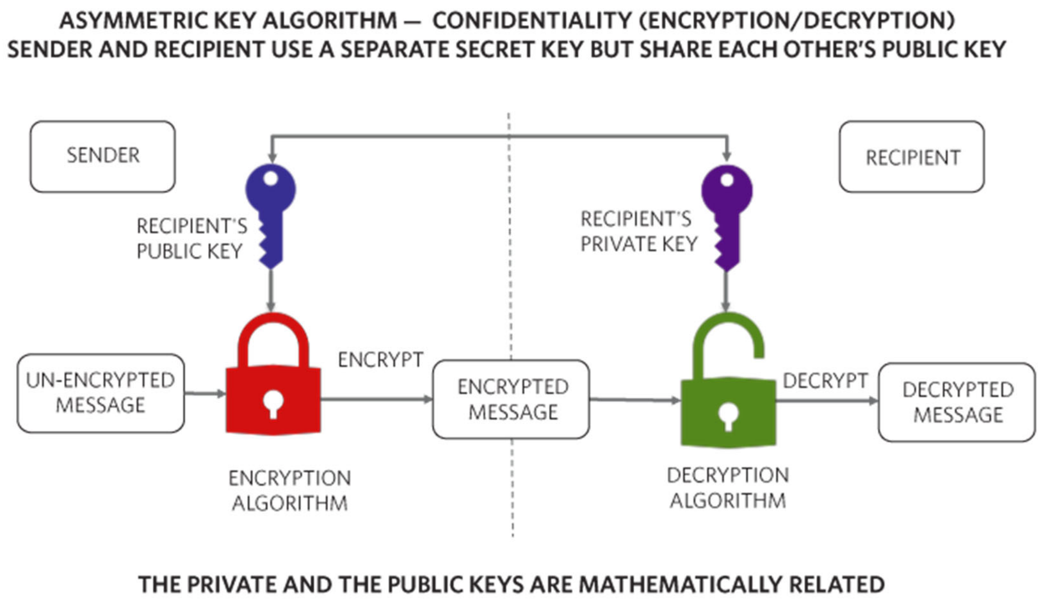

ASYMMETRIC KEY ALGORITHM — CONFIDENTIALITY (ENCRYPTION/DECRYPTION)
SENDER AND RECIPIENT USE A SEPARATE SECRET KEY BUT SHARE EACH OTHER'S PUBLIC KEY
SENDER
RECIPIENT
RECIPIENT'S PUBLIC KEY
RECIPIENT'S PRIVATE KEY
UN-ENCRYPTED MESSAGE
ENCRYPT
ENCRYPTED MESSAGE
DECRYPT
DECRYPTED MESSAGE
ENCRYPTION ALGORITHM
DECRYPTION ALGORITHM
THE PRIVATE AND THE PUBLIC KEYS ARE MATHEMATICALLY RELATED

Figure 2. Asymmetric key algorithm helps achieve confidentiality through the use of public and private keys.

The recipient's public key is mathematically related to the recipient's private key. The sender, and anyone else for that matter, doesn't have access to the recipient's private key. Once the recipient receives the message, the private key is used to decrypt the message. The recipient's private key is the only one that can be used to decrypt the message that was encrypted with the related public key. Since the private key only resides with the recipient, another person or organization can't decrypt the sent message. Thus, the message stays "confidential."

Identification and Authentication Using Symmetric Key Algorithm

The goal of identification and authentication is to first identify an object or a user and then authenticate them so we know that we are communicating with someone that we really meant to communicate with.

How is this achieved using a symmetric key scheme? **Figure 3** shows a simple example of the symmetric key identification and authentication process. Review Steps 1 to 6 for a better understanding. Step 4 uses a concept called the "digest." A digest or hash is a fixed-length value that is computed over a large data set.

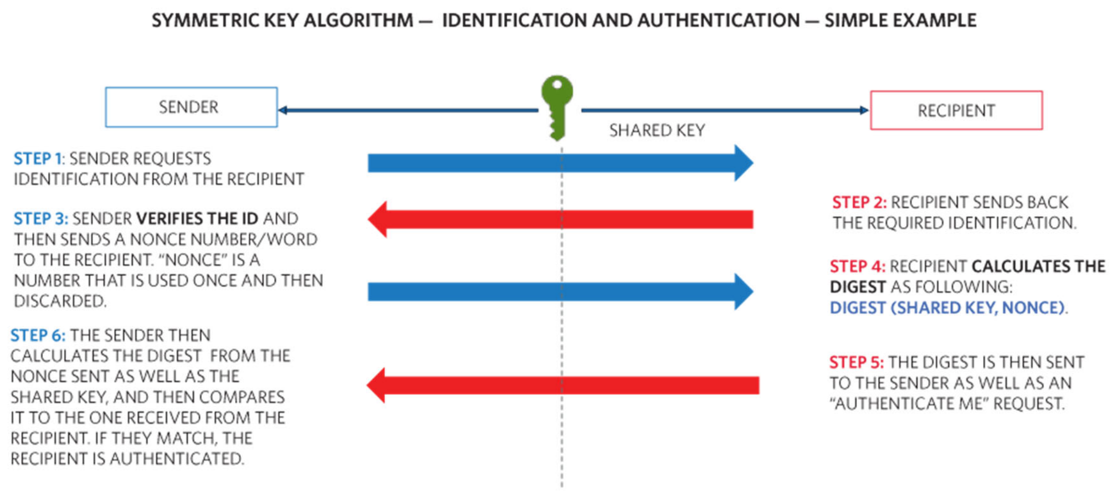

SYMMETRIC KEY ALGORITHM — IDENTIFICATION AND AUTHENTICATION — SIMPLE EXAMPLE
SENDER
RECIPIENT
SHARED KEY
STEP 1: SENDER REQUESTS IDENTIFICATION FROM THE RECIPIENT
STEP 2: RECIPIENT SENDS BACK THE REQUIRED IDENTIFICATION.
STEP 3: SENDER VERIFIES THE ID AND THEN SENDS A NONCE NUMBER/WORD TO THE RECIPIENT. "NONCE" IS A NUMBER THAT IS USED ONCE AND THEN DISCARDED.
STEP 4: RECIPIENT CALCULATES THE DIGEST AS FOLLOWING: DIGEST (SHARED KEY, NONCE).
STEP 5: THE DIGEST IS THEN SENT TO THE SENDER AS WELL AS AN "AUTHENTICATE ME" REQUEST.
STEP 6: THE SENDER THEN CALCULATES THE DIGEST FROM THE NONCE SENT AS WELL AS THE SHARED KEY, AND THEN COMPARES IT TO THE ONE RECEIVED FROM THE RECIPIENT. IF THEY MATCH, THE RECIPIENT IS AUTHENTICATED.

Figure 3. This diagram shows a simple example of the symmetric key identification and authentication process.

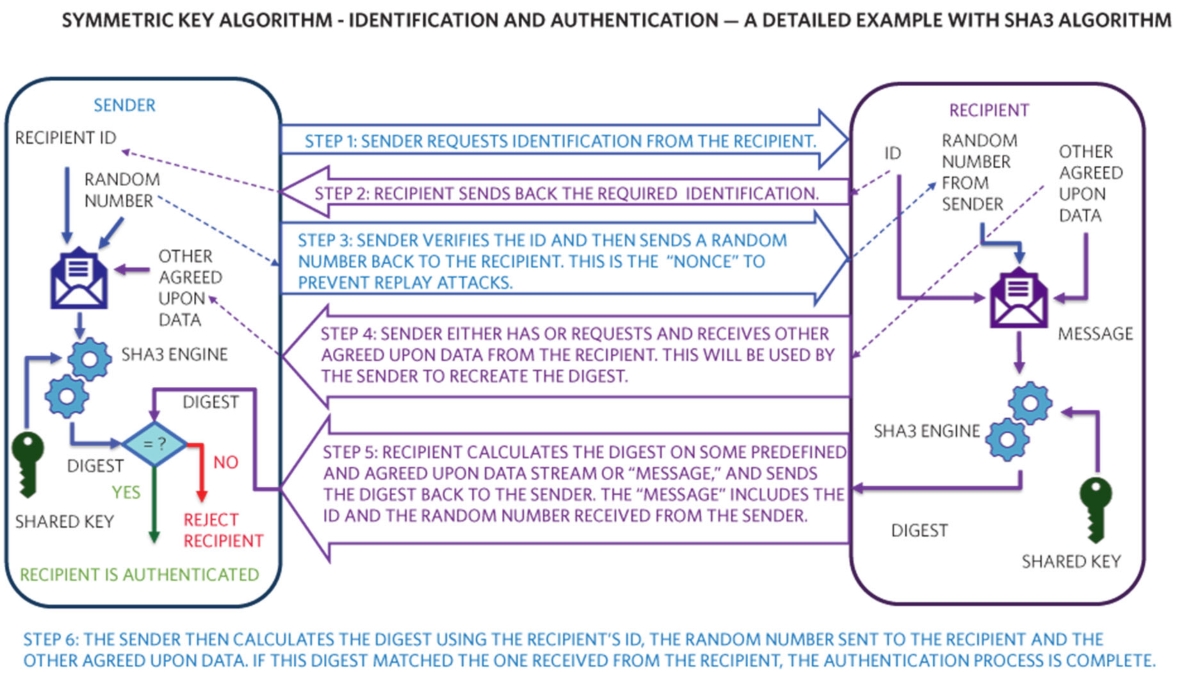
Why Do We Need a "Nonce"?

An imposter could gain possession of the last digest transmitted by the recipient and then issue an "authenticate me" with that digest. These types of attacks are called "replay attacks," i.e., a resend of a previously used digest. The use of a "nonce" or a single-use random number for authentication prevents such attacks. In this case, the authentication will fail, since for each authentication, the sender requires a new digest with a brand-new nonce number. These numbers are usually generated using an approved random number generator.

Now let's investigate a real-life example of identification and authentication using the SHA3-256 algorithm.

Identification and Authentication Using the SHA-3 Algorithm

Figure 4 shows a more complete example of the symmetric key identification and authentication process. This uses the SHA-3 symmetric key algorithm, which is the latest in the Secure Hash Algorithm (SHA) family. Maxim Integrated is the first to have a SHA3-256 secure authentication device in production. Review Steps 1 to 6 in the diagram to better understand the process. The "random number" in Figure 4 is basically the "nonce" needed to prevent replay attacks as discussed in the simple example in a section earlier.



STEP 1: SENDER REQUESTS IDENTIFICATION FROM THE RECIPIENT.

STEP 2: RECIPIENT SENDS BACK THE REQUIRED IDENTIFICATION.

STEP 3: SENDER VERIFIES THE ID AND THEN SENDS A RANDOM NUMBER BACK TO THE RECIPIENT. THIS IS THE "NONCE" TO PREVENT REPLAY ATTACKS.

STEP 4: SENDER EITHER HAS OR REQUESTS AND RECEIVES OTHER AGREED UPON DATA FROM THE RECIPIENT. THIS WILL BE USED BY THE SENDER TO RECREATE THE DIGEST.

STEP 5: RECIPIENT CALCULATES THE DIGEST ON SOME PREDEFINED AND AGREED UPON DATA STREAM OR "MESSAGE," AND SENDS THE DIGEST BACK TO THE SENDER. THE "MESSAGE" INCLUDES THE ID AND THE RANDOM NUMBER RECEIVED FROM THE SENDER.

STEP 6: THE SENDER THEN CALCULATES THE DIGEST USING THE RECIPIENT'S ID, THE RANDOM NUMBER SENT TO THE RECIPIENT AND THE OTHER AGREED UPON DATA. IF THIS DIGEST MATCHED THE ONE RECEIVED FROM THE RECIPIENT, THE AUTHENTICATION PROCESS IS COMPLETE.

Figure 4. This diagram shows a detailed example of symmetric key algorithm with SHA-3.

Identification and Authentication Using Asymmetric Key Algorithm

As mentioned, the goal for identification and authentication is to first identify an object or a user and then authenticate them so that we know we are communicating with someone that we really meant to communicate with.

How is this achieved using an asymmetric key scheme? **Figure 5** shows a simple example of the symmetric key identification and authentication process. Review Steps 1 to 6 in the diagram to understand the process.

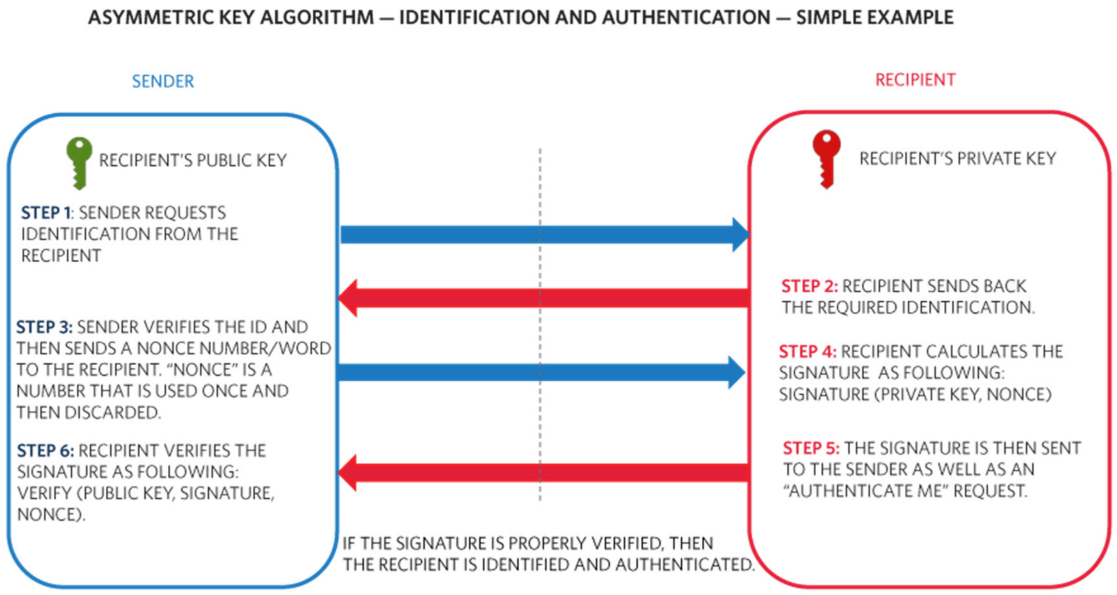

ASYMMETRIC KEY ALGORITHM — IDENTIFICATION AND AUTHENTICATION — SIMPLE EXAMPLE
SENDER
RECIPIENT
RECIPIENT'S PUBLIC KEY
RECIPIENT'S PRIVATE KEY
STEP 1: SENDER REQUESTS IDENTIFICATION FROM THE RECIPIENT
STEP 2: RECIPIENT SENDS BACK THE REQUIRED IDENTIFICATION.
STEP 3: SENDER VERIFIES THE ID AND THEN SENDS A NONCE NUMBER/WORD TO THE RECIPIENT. "NONCE" IS A NUMBER THAT IS USED ONCE AND THEN DISCARDED.
STEP 4: RECIPIENT CALCULATES THE SIGNATURE AS FOLLOWING: SIGNATURE (PRIVATE KEY, NONCE)
STEP 5: THE SIGNATURE IS THEN SENT TO THE SENDER AS WELL AS AN "AUTHENTICATE ME" REQUEST.
STEP 6: RECIPIENT VERIFIES THE SIGNATURE AS FOLLOWING: VERIFY (PUBLIC KEY, SIGNATURE, NONCE).
IF THE SIGNATURE IS PROPERLY VERIFIED, THEN THE RECIPIENT IS IDENTIFIED AND AUTHENTICATED.

Figure 5. This diagram shows a simple example of identification and authentication using the asymmetric key algorithm.

Why Do We Need a Nonce?

An imposter could obtain the last signature transmitted by the recipient and then issue an "authenticate me" with that signature. These types of attacks are called "replay attacks" i.e., a resend of a previously used signature. The use of a nonce or single-use random number for authentication prevents such attacks. In this case, the authentication will fail, since the sender requires a new signature with a brand-new nonce number for each authentication. These numbers are usually generated using an approved random number generator.

Now let's investigate a real-life example of identification and authentication using the Elliptic Curve Digital Signature Algorithm (ECDSA) algorithm.

Identification and Authentication Using the ECDSA Algorithm

Figure 6 shows a more complete example of the asymmetric key identification and authentication process using the ECDSA asymmetric key algorithm. Steps 1 to 6 in the diagram can help you better understand the process.

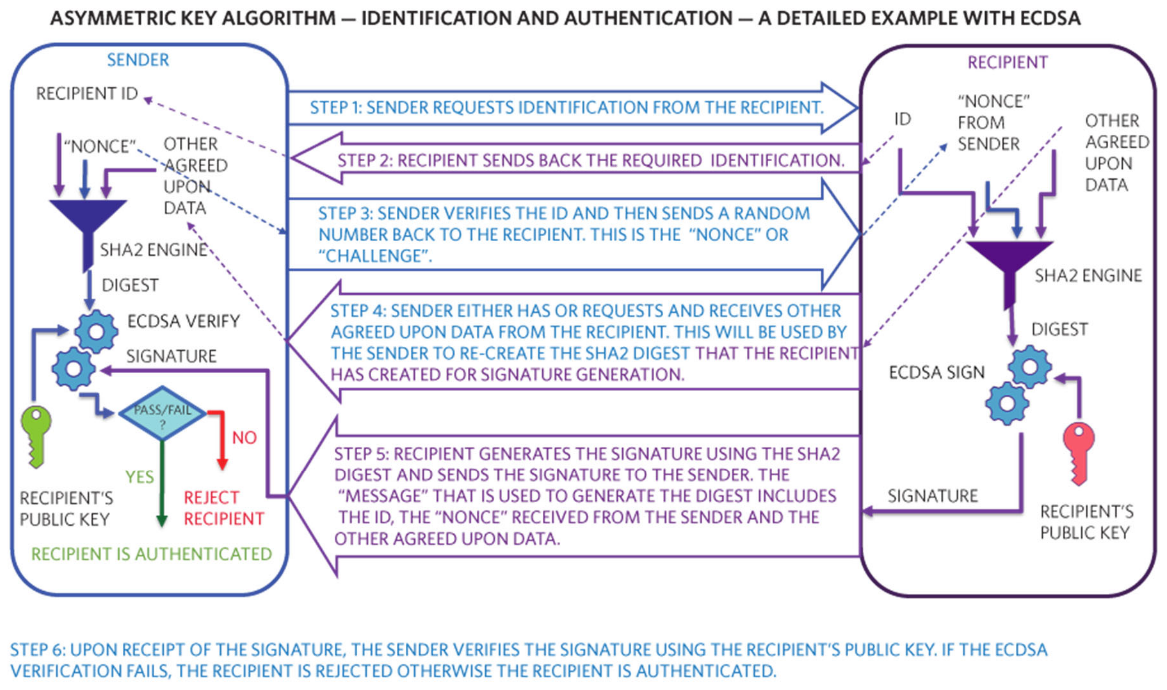

ASYMMETRIC KEY ALGORITHM — IDENTIFICATION AND AUTHENTICATION — A DETAILED EXAMPLE WITH ECDSA
SENDER
RECIPIENT ID
"NONCE"
OTHER AGREED UPON DATA
SHA2 ENGINE
DIGEST
ECDSA VERIFY
SIGNATURE
PASS/FAIL ?
NO
YES
RECIPIENT'S PUBLIC KEY
REJECT RECIPIENT
RECIPIENT IS AUTHENTICATED
STEP 1: SENDER REQUESTS IDENTIFICATION FROM THE RECIPIENT.
STEP 2: RECIPIENT SENDS BACK THE REQUIRED IDENTIFICATION.
STEP 3: SENDER VERIFIES THE ID AND THEN SENDS A RANDOM NUMBER BACK TO THE RECIPIENT. THIS IS THE "NONCE" OR "CHALLENGE".
STEP 4: SENDER EITHER HAS OR REQUESTS AND RECEIVES OTHER AGREED UPON DATA FROM THE RECIPIENT. THIS WILL BE USED BY THE SENDER TO RE-CREATE THE SHA2 DIGEST THAT THE RECIPIENT HAS CREATED FOR SIGNATURE GENERATION.
STEP 5: RECIPIENT GENERATES THE SIGNATURE USING THE SHA2 DIGEST AND SENDS THE SIGNATURE TO THE SENDER. THE "MESSAGE" THAT IS USED TO GENERATE THE DIGEST INCLUDES THE ID, THE "NONCE" RECEIVED FROM THE SENDER AND THE OTHER AGREED UPON DATA.
RECIPIENT
ID
"NONCE" FROM SENDER
OTHER AGREED UPON DATA
SHA2 ENGINE
DIGEST
ECDSA SIGN
SIGNATURE
RECIPIENT'S PUBLIC KEY
STEP 6: UPON RECEIPT OF THE SIGNATURE, THE SENDER VERIFIES THE SIGNATURE USING THE RECIPIENT'S PUBLIC KEY. IF THE ECDSA VERIFICATION FAILS, THE RECIPIENT IS REJECTED OTHERWISE THE RECIPIENT IS AUTHENTICATED.

Figure 6. A detailed example of identification and authentication using the ECDSA asymmetric key algorithm.

Although this method completes the device authentication, it doesn't cover the complete system authentication process. This includes verification that the recipient is part of the system and the required verification of the device digital certificates.

Comparing Cryptographic Algorithms

Figure 7 shows a side-by-side comparison of key usage for symmetric and asymmetric key algorithms. Before we go into the next topic, we need to understand the differences between the following two concepts:

- Secure hash
- HMAC (hashed message authentication code)

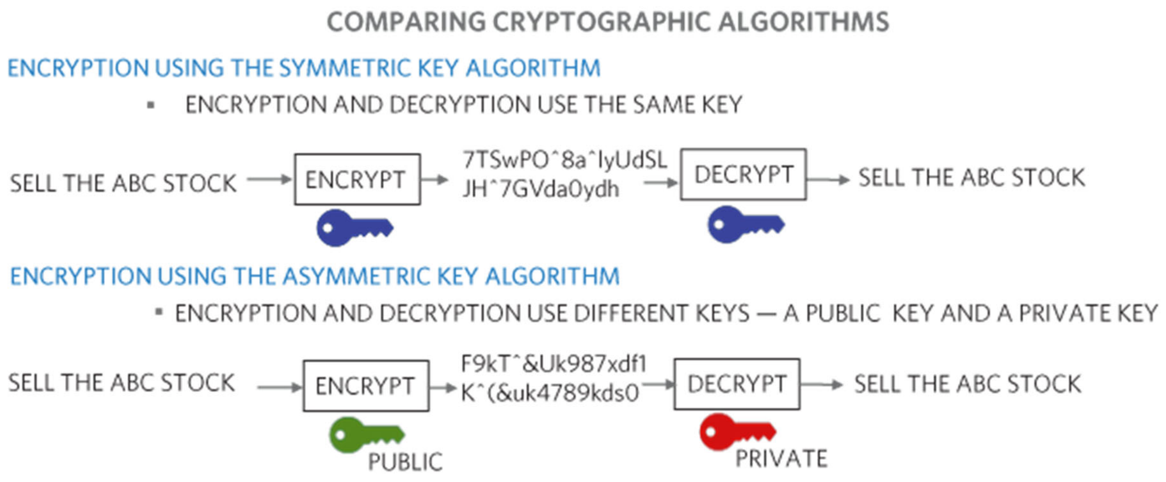

COMPARING CRYPTOGRAPHIC ALGORITHMS
ENCRYPTION USING THE SYMMETRIC KEY ALGORITHM
ENCRYPTION AND DECRYPTION USE THE SAME KEY
SELL THE ABC STOCK
ENCRYPT
7TSwPO^8a^lyUdSL JH^7GVda0ydh
DECRYPT
SELL THE ABC STOCK
ENCRYPTION USING THE ASYMMETRIC KEY ALGORITHM
ENCRYPTION AND DECRYPTION USE DIFFERENT KEYS — A PUBLIC KEY AND A PRIVATE KEY
SELL THE ABC STOCK
ENCRYPT
F9kT^&Uk987xdf1 K^(&uk4789kds0
DECRYPT
SELL THE ABC STOCK
PUBLIC
PRIVATE

Figure 7. This is a comparison of symmetric key and asymmetric key cryptographic algorithms.

Figure 8 illustrates the differences between HMAC and secure hash. Essentially, secure hash uses a hashing algorithm, such as SHA-3, to produce a fixed-length hash of the message regardless of the message length. HMAC is similar but uses a key as an additional input to the hashing engine. It also produces a fixed-length hash regardless of the input message length.

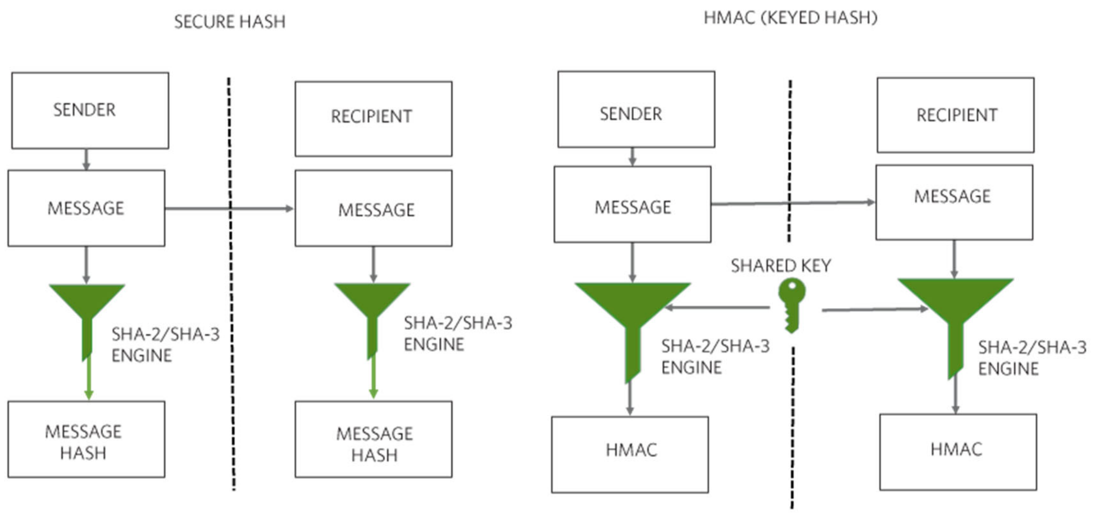


Figure 8. There are similarities but key differences between HMAC and Secure Hash.

Preserving Integrity Using Symmetric Key Algorithms

The goal of preserving the integrity of a message is to ensure that any message received, or any new device being connected, is not carrying unwanted code or information. Let's look at an example of how this can be achieved using a symmetric key algorithm such as SHA-3. Later, we'll review the specifics of how these algorithms work.

In **Figure 9**, the sender calculates the digest of a message by using a specific key. As this is a symmetric key scheme, this key is shared between the sender and the recipient. The digest or hash that is generated using a key is called an HMAC (hash-based message authentication code).

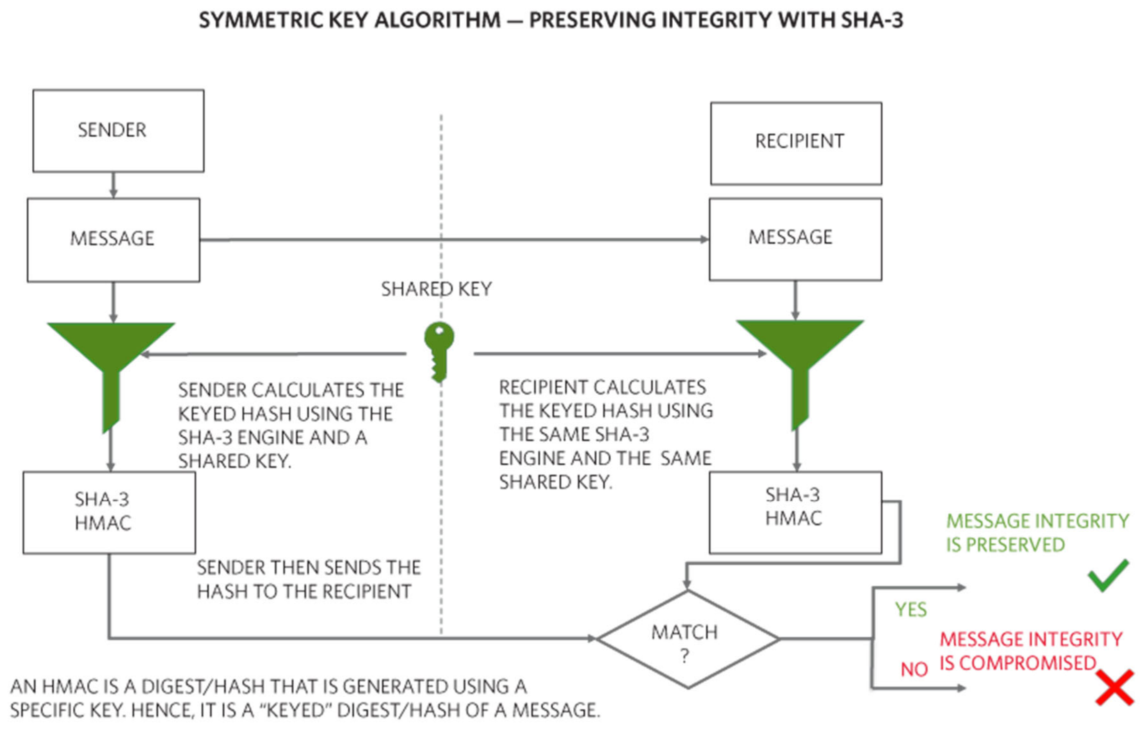

SYMMETRIC KEY ALGORITHM — PRESERVING INTEGRITY WITH SHA-3
SENDER
RECIPIENT
MESSAGE
MESSAGE
SHARED KEY
SENDER CALCULATES THE KEYED HASH USING THE SHA-3 ENGINE AND A SHARED KEY.
RECIPIENT CALCULATES THE KEYED HASH USING THE SAME SHA-3 ENGINE AND THE SAME SHARED KEY.
SHA-3 HMAC
SHA-3 HMAC
SENDER THEN SENDS THE HASH TO THE RECIPIENT
MATCH ?
YES
NO
MESSAGE INTEGRITY IS PRESERVED
MESSAGE INTEGRITY IS COMPROMISED
AN HMAC IS A DIGEST/HASH THAT IS GENERATED USING A SPECIFIC KEY. HENCE, IT IS A "KEYED" DIGEST/HASH OF A MESSAGE.

Figure 9. SHA-3 symmetric key algorithm preserves integrity.

This is generated by feeding the message and the key to the SHA-3 engine. The resultant HMAC and message is then sent to the recipient. The recipient then generates her own HMAC using the key she has. The two HMACs are then compared and, if they match, the message has not been tampered with. In this scenario, someone could intercept both the HMAC and the message and then alter the message and generate a new HMAC and send it to the recipient. This will not work, however, since the interceptor will not have the recipient's secret key and the HMACs will not match.

Preserving Integrity Using Asymmetric Key Algorithms

The goal of preserving the integrity of a message is to ensure that any message received, or any new device being connected is not carrying unwanted code or information. Let's look at an example of how this is achieved using an asymmetric key algorithm such as ECDSA (Elliptic Curve Digital Signature Algorithm).

The basic idea behind this is that the sender signs a message with a digital signature and the recipient verifies the signature, to be assured of the received message's integrity.

In **Figure 10**, the sender calculates the digest of a message by feeding the message to a SHA-2 hashing engine. As this is an asymmetric key scheme, this key is not shared between the sender and the recipient. The sender has a private key that is never shared, and the recipient has a public key that can be shared with many people and vice versa, unlike the symmetric key algorithm the digest/hash that is generated doesn't use a key.

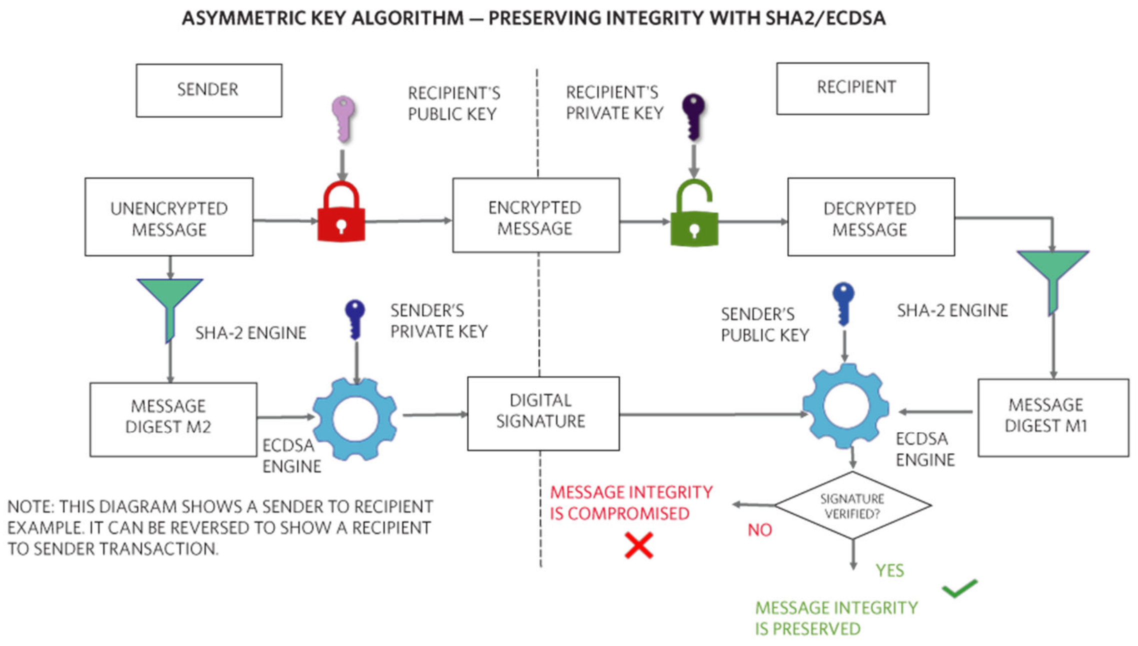


Figure 10. ECDSA asymmetric key algorithm helps preserve message integrity.

The generated digest is then fed to the ECDSA engine along with the sender's private key to generate a digital signature of the message. This signature, along with the message, is sent to the recipient. This completes the signing process for the sent message.

Now that the recipient has received the message and the digital signature from the sender, she can start the verification process. This process consists of two distinct steps:

Step 1: The recipient computes a message digest from the received message.

Step 2: This newly computed digest, the received digital signature from the sender, along with the sender's public key are then fed into the ECDSA engine for verification.

During the verification process, the ECDSA engine produces a "yes" or a "no" result. If the result is a "yes," then the message integrity has been preserved. If the result is a "no," the message integrity has been compromised.

Non-Repudiation Using Asymmetric Key Algorithms

A message signed by a digital signature from the sender can be used to prove that the message was sent by the sender and that the message was unaltered. However, a digital signature cannot prove the identity of the sender. Proof of identity is achieved by using a digital certificate. **Figures 11** through **14** show the complete steps needed to achieve a complete public key system where messages exchanged cannot be repudiated by either party.

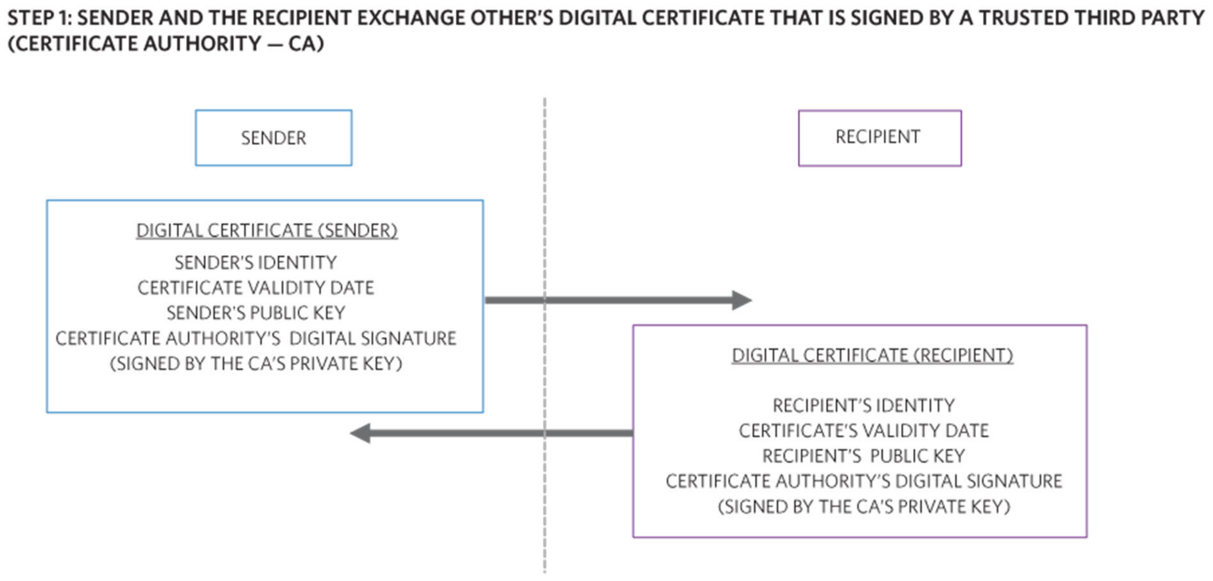

STEP 1: SENDER AND THE RECIPIENT EXCHANGE OTHER'S DIGITAL CERTIFICATE THAT IS SIGNED BY A TRUSTED THIRD PARTY (CERTIFICATE AUTHORITY — CA)
SENDER
RECIPIENT
DIGITAL CERTIFICATE (SENDER)
SENDER'S IDENTITY
CERTIFICATE VALIDITY DATE
SENDER'S PUBLIC KEY
CERTIFICATE AUTHORITY'S DIGITAL SIGNATURE
(SIGNED BY THE CA'S PRIVATE KEY)
DIGITAL CERTIFICATE (RECIPIENT)
RECIPIENT'S IDENTITY
CERTIFICATE'S VALIDITY DATE
RECIPIENT'S PUBLIC KEY
CERTIFICATE AUTHORITY'S DIGITAL SIGNATURE
(SIGNED BY THE CA'S PRIVATE KEY)

Figure 11. A sender and recipient exchange a trusted third-party-signed digital certificate.

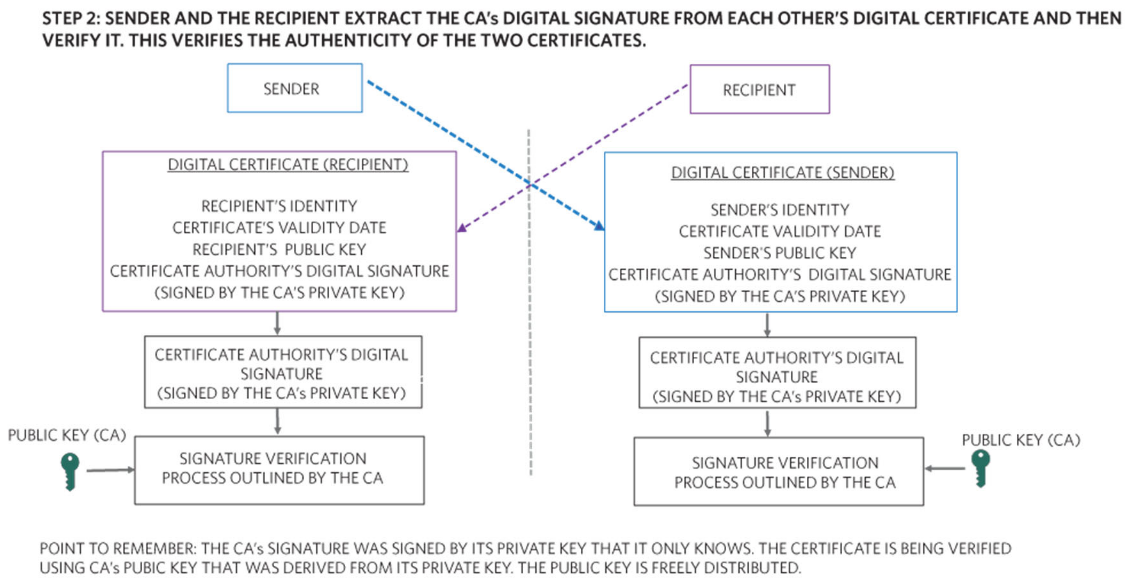

STEP 2: SENDER AND THE RECIPIENT EXTRACT THE CA's DIGITAL SIGNATURE FROM EACH OTHER'S DIGITAL CERTIFICATE AND THEN VERIFY IT. THIS VERIFIES THE AUTHENTICITY OF THE TWO CERTIFICATES.
SENDER
RECIPIENT
DIGITAL CERTIFICATE (RECIPIENT)
RECIPIENT'S IDENTITY
CERTIFICATE'S VALIDITY DATE
RECIPIENT'S PUBLIC KEY
CERTIFICATE AUTHORITY'S DIGITAL SIGNATURE
(SIGNED BY THE CA'S PRIVATE KEY)
DIGITAL CERTIFICATE (SENDER)
SENDER'S IDENTITY
CERTIFICATE VALIDITY DATE
SENDER'S PUBLIC KEY
CERTIFICATE AUTHORITY'S DIGITAL SIGNATURE
(SIGNED BY THE CA'S PRIVATE KEY)
CERTIFICATE AUTHORITY'S DIGITAL SIGNATURE
(SIGNED BY THE CA's PRIVATE KEY)
CERTIFICATE AUTHORITY'S DIGITAL SIGNATURE
(SIGNED BY THE CA's PRIVATE KEY)
PUBLIC KEY (CA)
SIGNATURE VERIFICATION PROCESS OUTLINED BY THE CA
SIGNATURE VERIFICATION PROCESS OUTLINED BY THE CA
PUBLIC KEY (CA)
POINT TO REMEMBER: THE CA's SIGNATURE WAS SIGNED BY ITS PRIVATE KEY THAT IT ONLY KNOWS. THE CERTIFICATE IS BEING VERIFIED USING CA's PUBIC KEY THAT WAS DERIVED FROM ITS PRIVATE KEY. THE PUBLIC KEY IS FREELY DISTRIBUTED.

Figure 12. A sender and recipient verify the authenticity of a trusted third-party-signed digital certificate.

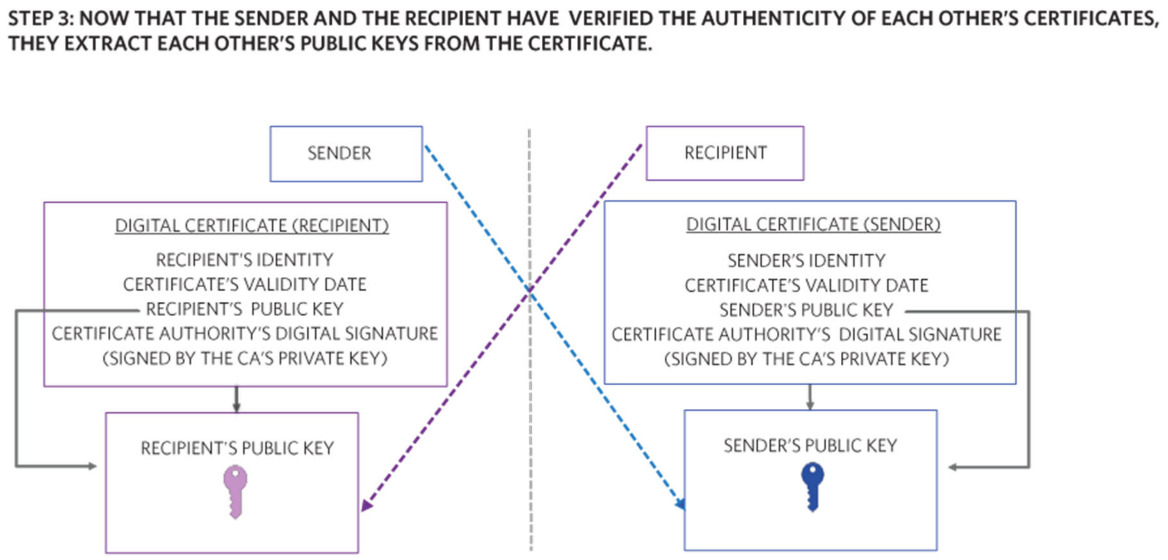

STEP 3: NOW THAT THE SENDER AND THE RECIPIENT HAVE VERIFIED THE AUTHENTICITY OF EACH OTHER'S CERTIFICATES, THEY EXTRACT EACH OTHER'S PUBLIC KEYS FROM THE CERTIFICATE.
SENDER
RECIPIENT
DIGITAL CERTIFICATE (RECIPIENT)
RECIPIENT'S IDENTITY
CERTIFICATE'S VALIDITY DATE
RECIPIENT'S PUBLIC KEY
CERTIFICATE AUTHORITY'S DIGITAL SIGNATURE
(SIGNED BY THE CA'S PRIVATE KEY)
DIGITAL CERTIFICATE (SENDER)
SENDER'S IDENTITY
CERTIFICATE'S VALIDITY DATE
SENDER'S PUBLIC KEY
CERTIFICATE AUTHORITY'S DIGITAL SIGNATURE
(SIGNED BY THE CA'S PRIVATE KEY)
RECIPIENT'S PUBLIC KEY
SENDER'S PUBLIC KEY

Figure 13. A sender and recipient extract each other's public keys from a digital certificate.

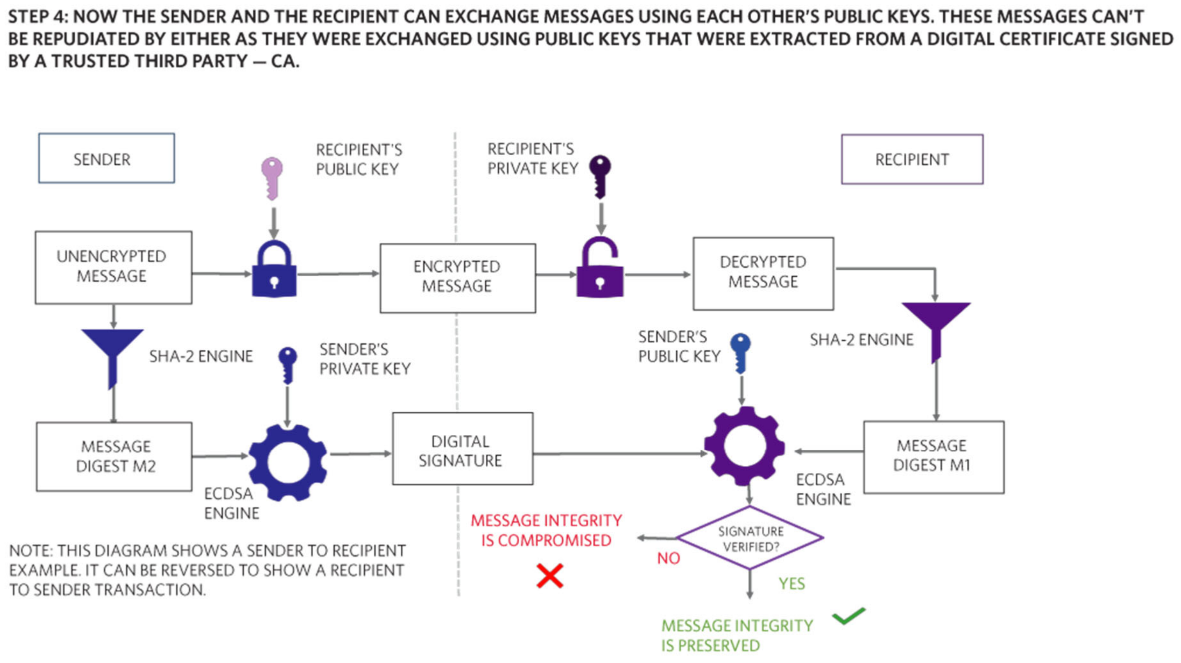

STEP 4: NOW THE SENDER AND THE RECIPIENT CAN EXCHANGE MESSAGES USING EACH OTHER'S PUBLIC KEYS. THESE MESSAGES CAN'T BE REPUDIATED BY EITHER AS THEY WERE EXCHANGED USING PUBLIC KEYS THAT WERE EXTRACTED FROM A DIGITAL CERTIFICATE SIGNED BY A TRUSTED THIRD PARTY — CA.
SENDER
RECIPIENT'S PUBLIC KEY
RECIPIENT'S PRIVATE KEY
RECIPIENT
UNENCRYPTED MESSAGE
ENCRYPTED MESSAGE
DECRYPTED MESSAGE
SHA-2 ENGINE
SENDER'S PRIVATE KEY
SENDER'S PUBLIC KEY
SHA-2 ENGINE
MESSAGE DIGEST M2
DIGITAL SIGNATURE
MESSAGE DIGEST M1
ECDSA ENGINE
ECDSA ENGINE
MESSAGE INTEGRITY IS COMPROMISED
SIGNATURE VERIFIED?
NO
YES
MESSAGE INTEGRITY IS PRESERVED
NOTE: THIS DIAGRAM SHOWS A SENDER TO RECIPIENT EXAMPLE. IT CAN BE REVERSED TO SHOW A RECIPIENT TO SENDER TRANSACTION.

Figure 14. The sender and recipient exchange messages that cannot be repudiated.

The main idea is that both the sender and recipient need to prove their identity to one another, and their respective public keys need to be proven authentic by a trusted third party.

Why is it so important to use a digital certificate? Without it, someone pretending to be the sender (i.e., an imposter) could send a message encrypted with the recipient's public key along with a digital signature signed with the imposter's private key. The imposter would then send the recipient his/her made-up public key. The recipient would then use that public key to verify the digital signature and everything would be validated. But the message from the imposter may have malicious information that the recipient will never suspect. This is the issue that can be avoided by using a digital certificate that verifies that the public key received did indeed belong to the sender and not some imposter.

Maxim Integrated has a wide variety of symmetric and asymmetric key based hardware authenticators that can be used to accomplish all the concepts discussed in this chapter. Watch for other segments in our series of cryptography application notes to continue deepening your understanding of this important security technique.