A1372_Maxim Integrated_Authentication

Implementing Secure Authentication Without Being a Cryptography Expert

By Christophe Tremlet, Executive Business Manager, Micros & Security, Maxim Integrated

Today, digital security is one of the most hyped topics in electronic design. For many engineers, encryption is probably the first word that comes to mind when they think about security. Probably only a few think initially about authentication.

However, authentication is a fundamental function of secure devices or transactions. Let's take the example of home banking. Clearly, you'd want confidential information such as balances and account numbers to be encrypted. This is what happens when your internet browser displays the green lock with https://. That said, the first thing the internet browser checks when establishing a secure connection is that the bank website is genuine; in other words, it *authenticates* the bank website. Indeed, sending login and password information to a mock-up site would be extremely harmful, as these credentials can be further re-used to run any kind of unauthorized transactions on behalf of an unsuspecting bank account holder. Secure internet browsing is generally achieved through the TLS/SSL protocol, which ensures authenticity and confidentiality.

Authentication is also important for Internet of things (IoT) applications: an untrusted endpoint could put a whole infrastructure at risk. Let's consider smart meters connected to the electrical power distribution system. An easy way for an attacker to disrupt the grid is to load a virus or malware in the smart meters. Infected meters could then send fake messages to the infrastructure reflecting a power consumption largely different from the actual one. The grid would then become unbalanced; worst case, the attack could trigger a full power outage. In order to avoid this situation, both the hardware of the meter and its firmware must be verified as genuine. The process of authenticating the firmware is called *secure boot*.

Implementing an Effective Authentication Method

Now that we understand the importance of authentication, let's discuss how to implement it. The most trivial way to authenticate is to use a password. In our smart meter example, the device could send a password to the grid control system. The server would verify the password and then authorize further transactions. While this method is easy to understand, it is by far not the best one. An attacker could quite easily spy on the communication, record the password, and re-use it to authenticate a non-genuine piece of equipment. For this reason, we consider password-based authentication as weak.

A much better way to perform authentication in the digital world is the *challenge-response* method. Let's take a look at two flavors of the challenge-response method: one based on symmetric cryptography and another one based on asymmetric cryptography.

Symmetric cryptography-based authentication relies on a shared secret. The host and the device to be authenticated hold the same secret number. The host sends a random number, the *challenge*, to the device. The device computes a *digital signature* as a function of the secret and the challenge and sends it back to the host. The host then runs the same computation and compares the result. If both computations match, then the device is authenticated (Figure 1). In order to make sure that the result cannot be mimicked, it's essential to use a function that has adequate mathematical properties; for example, making sure that it is impossible to retrieve the secret with the computation result is mandatory. Secure hash functions, such as SHA-256, support these requirements. For the challenge-response method, the device proves it knows a secret without disclosing it. Even if an attacker were to intercept the communication, the attacker still cannot access the shared secret.

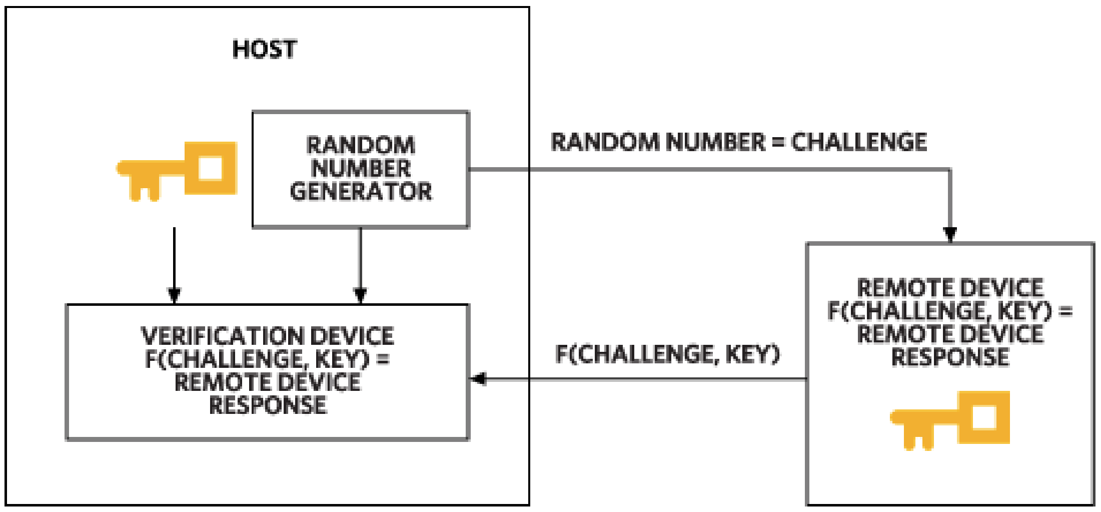
HOST
RANDOM NUMBER GENERATOR
RANDOM NUMBER = CHALLENGE
VERIFICATION DEVICE F(CHALLENGE, KEY) = REMOTE DEVICE RESPONSE
F(CHALLENGE, KEY)
REMOTE DEVICE F(CHALLENGE, KEY) = REMOTE DEVICE RESPONSE

Figure 1: Authentication based on symmetric cryptography relies on a secret number shared between the host and the device.

Authentication based on asymmetric cryptography relies on two keys: a private and a public key. The private key is known only by the device to be authenticated, while the public key can be disclosed to any entity willing to authenticate the device. As in the previously discussed method, the host sends a challenge to the device. The device computes a signature based on the challenge and the private key and sends it back to the host (Figure 2). But here, the host will use the public key to verify the signature. It's also critical for the function used to compute the signature to have certain mathematic properties. The most commonly used functions for the asymmetric schemes are RSA and ECDSA. Here also, the device proves it has knowledge of a secret, the private key, without disclosing it.

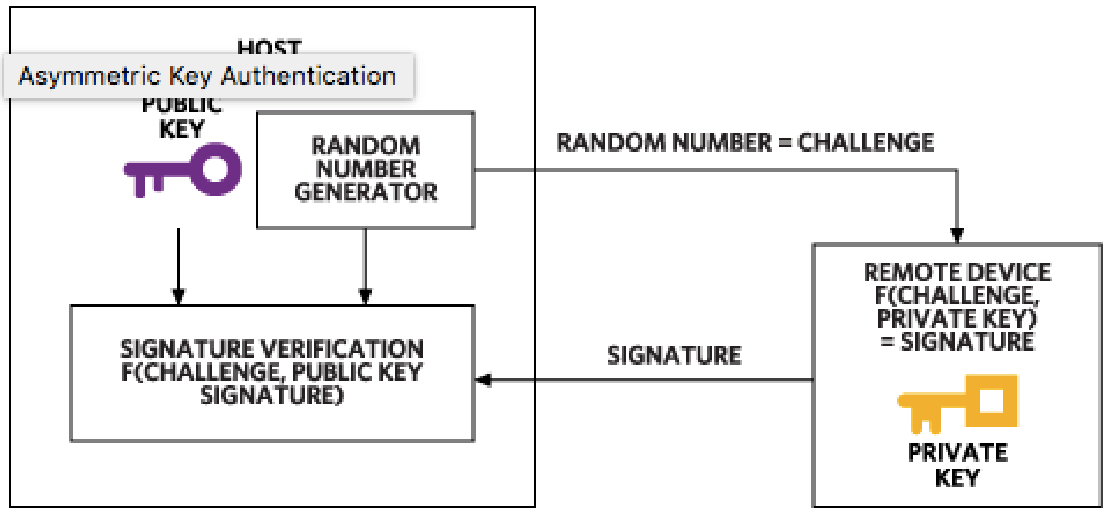
HOST
Asymmetric Key Authentication
PUBLIC KEY
RANDOM NUMBER GENERATOR
RANDOM NUMBER = CHALLENGE
REMOTE DEVICE F(CHALLENGE, PRIVATE KEY) = SIGNATURE
SIGNATURE VERIFICATION F(CHALLENGE, PUBLIC KEY SIGNATURE)
SIGNATURE
PRIVATE KEY

Figure 2: Asymmetric key authentication relies on public and private keys.

Why Security ICs Are Beneficial for Authentication

Challenge-response authentication always needs the object to be authenticated to hold a secret. In symmetric cryptography, this is the shared secret between the host and the device. For asymmetric cryptography, this is the private key. In any case, the security brought by challenge-response authentication breaks when the secret is revealed. Here's where security ICs can help. One fundamental feature of security ICs is to provide strong protection of keys and secrets.

Maxim offers three families of solutions to support authentication:

- Authentication ICs: These are configurable but fixed-function devices that provide the most affordable way to implement challenge-response authentication, along with a compact set cryptographic operations
- Secure microcontrollers: On top of supporting challenge-response authentication, these devices offer a full set of cryptographic functions, including encryption
- Low-power microcontrollers: While these products do not exclusively target security, they have all of the building blocks required to enable strong authentication

Within authentication ICs, the SHA-256-based products support authentication based on shared secrets (Figure 3), while ECDSA-based ICs use a private/public key pair (Figure 4). In addition to the cryptographic engines, these products feature on-board EEPROM memory. This memory is configurable and can be used to store authenticated user data such as calibration information for sensors.

SHA-256-based products are the most affordable solutions. While they enable mutual authentication, the distribution of the shared secret requires some precautions so that the secret is not exposed during device manufacturing and set-up. The secret can be programmed in a Maxim factory to circumvent this drawback.

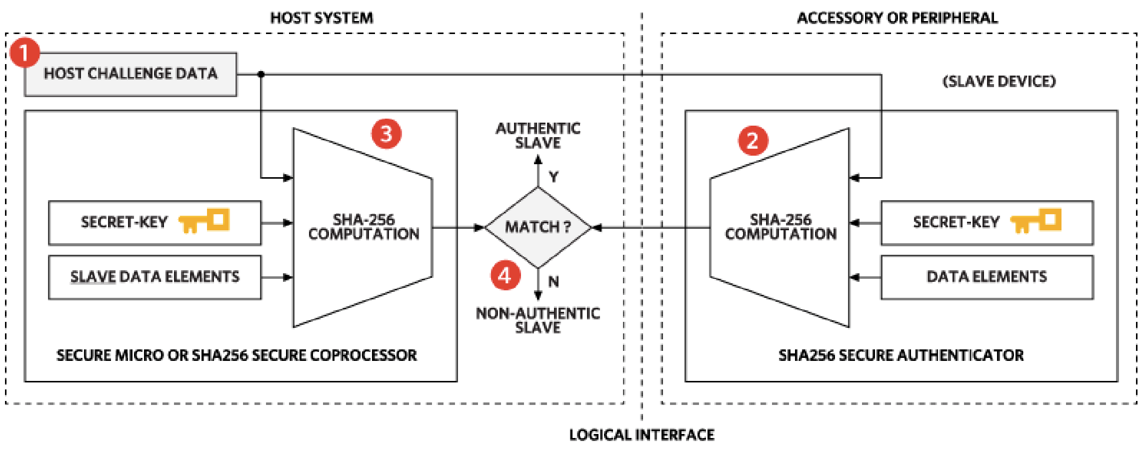

Figure 3: SHA-256 secure authentication is based on shared secrets.

Maxim's DS28E15/22/25 ICs are based on the SHA-256 technology and differ by their internal memory size. Since the same secret is stored on both the host and device sides, we recommend using a co-processor such as the DS2465 on the host side.

Asymmetric cryptography-based products such as DS28C36 and DS28E35 offer a more flexible scheme as the key does not need to be protected against disclosure on the host side. However, to offload public-key math and provide additional secure operations, host-side co-processors such as the DS2476 (companion IC to the DS28C36) are available to simplify development of the system solution.

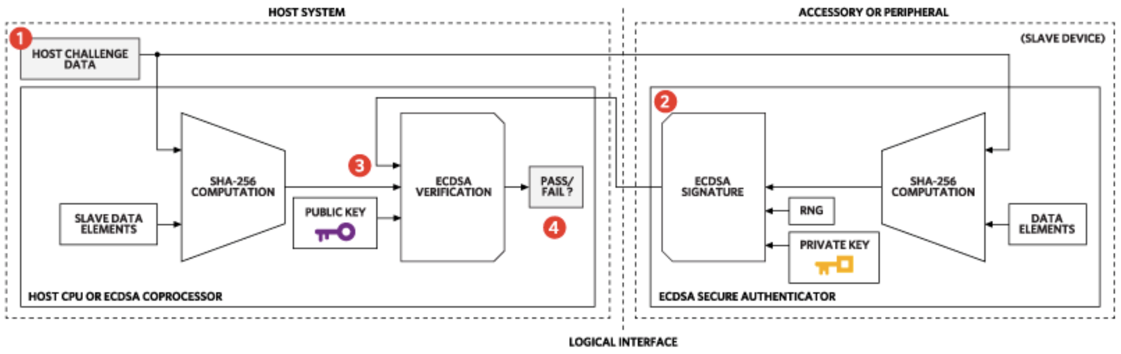

Figure 4: ECDSA-based authentication relies on a private/public key pair.

Secure Microcontrollers with Support for Symmetric and Asymmetric Cryptography
Maxim offers secure microcontrollers ranging from the MAX32590 (ARM9 running at 384MHz) application-class processor that can run advanced operating systems such as Linux down to small-footprint co-processors such as MAX32555 or MAXQ1061.

These microcontrollers support both symmetric and asymmetric cryptography for digital signature and authentication as well as encryption algorithms. They feature hardware accelerators for SHA, RSA, ECDSA, and AES as well as a full cryptography library providing a turnkey API aligned to standards. They have built-in secure boot, so that firmware authenticity is always guaranteed. Thanks to their comprehensive set of crypto functions, they can handle multiple authentication schemes.

The MAXQ1061 is a co-processor that not only enables authentication but also handles the most critical steps of the TLS/SSL standard secure communication protocol over IP. Handling TLS protocol within the chip improves the level of security and offloads the main processor from computing-intensive tasks. This is very valuable for resource-constrained embedded systems.

Low-Power Microcontrollers

Low-power microcontrollers such as MAX32626 target wearable devices, so are not “security-centric” ICs. With attacks becoming more and more frequent, however, this product has been designed with the security challenges of tomorrow in mind. Hence, MAX32626 has a hardware Trust Protection Unit supporting authentication as well as hardware AES for encryption and a built-in secure boot.

Summary

Through this article we have discovered what authentication is and why it is important. We’ve also seen that thanks to existing silicon-based solutions, one can enable authentication without being an expert in cryptography implementation.

embedded world 2017

The MAXQ1061 and other embedded security products will be on display at the embedded world trade show in Nuremberg, March 14 – 16, 2017, on Maxim’s stand in Hall 1, Booth #370.

Articles similar to this Application Note appeared in EE Times Europe on November 30, 2016, and ElektronikPraxis on January 24, 2017.