Keywords: IoT security, public key cryptography and countermeasures, private key management

APPLICATION NOTE 6004

SECURE THE IOT: PART 1, PUBLIC KEY CRYPTOGRAPHY SECURES CONNECTED DEVICES

By: Yann Loisel, Secuity Architect and Stephane di Vito, Security Expert, Maxim Integrated

Abstract: This application note is the first of two discussing the security of the IoT. It describes how to identify and then assess the security risks for a connected electronic device. It explains how the best, proven security is designed into electronic devices and loaded during manufacture. The focus is on countermeasures, specifically public key-based algorithms.

Introduction

Security of electronic devices is a "must have" in today's interconnected world of the Internet of Things (IoT). Electronic devices range from smart connected fridges to uranium centrifuge control systems. When the security of a device is compromised, "broken" as it were, we can no longer rely on the device for secure data exchange, processing, or storage. If electronic transactions, critical systems such as nuclear plants, or implantable medical devices are hacked, then the global trust would be impacted dramatically.

Electronic security is a far-reaching topic. This is the first application note in a two-part series on security for theIoT. In Part 1, we describe how to identify and then assess the security risks for a connected electronic device. We explain how the best, proven security is designed into electronic devices. Our focus is on countermeasures, specifically public key-based algorithms.

In our follow-up Part 2 application note we focus on the importance of a secure boot and the "root of trust," which are the cornerstones of the electronic device's trustworthiness. We will demonstrate how device security can be conveniently implemented and how devices can even be updated in the field. DeepCover® secure microcontrollers will serve as example trust-enabling components to secure the IoT.

The Connected World Reaches Out

Our lives are increasingly surrounded by interconnected electronic devices in what is now called the Internet of Everything. This IoT and every secure portable device, industrial, and medical equipment all have software running within the hardware. They ease our days, answer our needs, control electrical functions in our households, protect our lives in medical equipment, and provide us utility services (water, gas, electricity) through smart grids or by controlling power plants.

Secure personal devices and the IoT have altered personal behavior for many. The technology extends our arms, our wills, our minds beyond our bodies to help us communicate and consume. Manufacturers and many industries are embracing the IoT for business efficiencies and data tracking (i.e., Industry 4.0). Energy and water utilities only now are realizing the efficiencies and intelligence that they will gather with data management and data mining from remote access to smart meters[1] on an IoT network. Banks and payment processors now enable fast transactions with smart cards, at any time and any place, using free (or almost free), colorful, touch terminals. Home health with the IoT– ECG monitoring, glucose dispensers, or insulin pumps–is improving lives and saving time and money for both patients and medical facilities. Projections estimate that there will be 88M mobile POS connections in 2018.[2] Quite clearly, the connected electronic devices have definite value...and definite vulnerabilities too.

Recognize the Security Risks

It has become so easy, so comfortable surfing on the web from almost everywhere with our smartphone that we have forgotten about our old 56k modem. But today's connected devices and the instant accessibility to a bright world also give us a misguided sense of confidence. We should remember a sad, but simple truth: the investments, connections, and transactions over the IoT also whet the appetite of other, not-so-well-intentioned predators.

The security risks come from competitors, lone predators, and criminal organizations. The former are more inclined to duplicate/clone technology–the magical smartphones or the ink cartridges–saving them often years of R&D efforts. The second and latter will be more interested in stealing payment cards, PIN codes, keys in payment terminals, or in blackmailing individuals perhaps by sabotaging an account or remotely shutting down a portable medical device. We can also imagine terrorist threats by remote hacking of energy smart meters for energy distribution at industrial plants or hospitals.[3] There is no need for more examples here. Suffice it to say that the security risks are all around us.

The risks, generally speaking, to the *stakeholders* are numerous.

- **Loss of reputation.**"The battery that you (manufacturer 'x') claimed as genuine has exploded in my laptop."
- **Loss of IP.** "The terrific algorithm I've developed in my video decoder during the last five years has been copied and duplicated. And I did not patent it to avoid disclosure of my tricks!"
- **Loss of money.** "Tens of payment terminals are hacked in my retail chain store, so fake transactions are performed and/or cardholder sensitive data are stolen. Customers are going to blame me and I will need to identify the hackers."
- **Loss of goods.** "I just read about the hack of an energy meter published on the web and already thousands of dishonest subscribers are implementing it to pay a lower bill."
- **Loss of health.** "My insulin pump does not dispense any more, or it dispenses too much. Who ordered a change in delivery times?"
- **Loss of control of vital infrastructures.** "Who turned the lights off in the whole city?"

It seems obvious that any provider of electronic devices must have two objectives: first, deliver new, powerful, and cost-effective devices or services; and, second, be totally committed to the robustness, liability, and security of their product. This is, in fact, the only way for them to keep the confidence of users, stakeholders, and consumers.

Analyze the Risks

The above objectives, ambitious to be sure, are the sine qua non conditions for the longevity of a business. But recognizing security risks is only a first step in delivering secure products. Each provider must also employ, and enforce, a strict process of ongoing risk analysis.

Risk analysis, in its most simple form, is a three-step process. It starts by evaluating the assets, the goods to be protected, for their strengths and weaknesses. Then evaluate any potential attackers and profile their possible methods. Finally, examine any possible attack paths. Any consistency among the assets, attack method, and attack paths puts the device (the asset) at risk.

Consider a possible scenario. If hacking an energy smart meter with a simple Bluetooth® connection saves someone 20% on a monthly bill, there is a very high risk of massive, even wide-spread fraud. Similarly, if it costs very little to acquire the binary code of an application that controls the water and energy usage of household appliances, some dishonest competitors (or suppliers?!) would do it.

The options for action following a risk analysis require case-by-case decisions.

- **Take the legal/contractual approach.** This avenue is always very cost effective and worth setting up. A device manufacturer can easily ask subcontractors (e.g., manufacturing plants) to sign a nondisclosure agreement (NDA) and to promise to be honest and faithful.[4]
- **Implement technical countermeasures.** These steps will protect devices against dishonest partners, subcontractors, and outlaw/unreachable attackers. Technical countermeasures guarantee that a devices' expected behavior and functions are controlled, sealed, and sanctioned by the manufacturer. Nobody, nothing can then either modify defined operation or access protected functions.

When a manufacturer uses legal contracts with suppliers and technical countermeasures in a device, it is protecting its own assets and safeguarding the device against unauthorized tampering and theft of IP. A manufacturer is also ensuring the safe, reliable operation of the device for a user, you and me.

Makes sense so far, but how do you really implement countermeasures in a device? Part of the answer is cryptography in the software. We will now see how cryptography can be used as a toolbox to ensure device security and provide the trust and confidence for both the manufacturer and end user.

Countermeasures

What Is a Secure Boot?

We are not going to say a great deal about a secure boot in this application note because it is a major focus in our Part 2 application note. Nonetheless, we cannot discuss cryptography without some mention of a secure boot.

Electronic devices are composed of a set of electronic components mounted on a printed circuit board (PCB) with usually one (or more) microcontrollers that run embedded software. The software is seen as digital content and stored in some memory in a binary executable format. Enabling trust in the executed software is a fundamental expectation, and this trust is enabled thanks to the secure boot.

A secure boot is a process involving cryptography that allows an electronic device to start executing authenticated and, therefore, trusted software to operate. Now we will explore how to implement such a secure boot with the help of public key-based signature verification.

Public Key-Based Signature Verification

Existing public key cryptography schemes[5] verify, conveniently and securely, the integrity and authenticity of digital content. Integrity means that the digital content has not been modified since it was created. Authenticity means that the same digital content has been released by a well-identified entity. These two fundamental characteristics are provided by the digital signature scheme and required so the digital content (i.e., the binary executable code) can be trusted by an electronic device.

The integrity of digital content is guaranteed by a mechanism called the message digest, i.e., a secure hash algorithm like the famous SHA-1, SHA-256, and most recently the SHA-3. A message digest is like a "super cyclic redundancy check (CRC)"[6] but produces more bytes in the output. For instance, the SHA-256 algorithm produces a 32-byte output; a CRC-32 produces only 4 bytes. There is an important, fundamental property of a secure hash algorithm: it is impossible to forge digital content that produces a predefined hash value. The corollary is that two different random digital contents produce two different hash values; the probability of having two different digital contents producing the same hash value is virtually zero. Consequently, if some bytes of the digital content are changed, the hash value of the digital content changes. In addition, unlike a CRC, it is not possible to append some bytes to the modified digital content so that the resulting hash value will match the original, nonmodified digital content's hash value. Therefore, with a hash algorithm guarding the digital content, it is not possible to secretly modify that digital content. Lastly, computing a hash is like computing a CRC: no cryptographic keys are involved.

The authenticity of digital content is guaranteed by the public key-based digital signature scheme itself (i.e., a cryptographic recipe). Public-key cryptography is based on pairs of keys. Anyone can possess a pair of keys: one private key stored secretly (e.g., K_{PRIV}), and one public key (e.g., K_{PUB}) publicly available to anyone. The private key can be used to sign digital content. The issuer of the digital content uses its own secretly held private key to identify himself as the *issuer*. The public key can be used by anyone to verify a digital content's signature. Those two keys are tied together. Indeed, signing content with K_{PRIV} produces digital signatures that can be successfully verified by K_{PUB} only. No other public key can work. Conversely if a signature is successfully verified using K_{PUB}, then it was unquestionably signed by K_{PRIV} and no other private key.

Digital signature generation involves two steps. The first step consists of hashing the digital content and producing a hash value with the properties explained above. In the second step the former hash value is "signed" using the uniquely owned, undisclosed private key of the digital content author. This second step produces a value (the "signature") that is attached to the original digital content.

Now anyone who wants to verify the digital content signature has to perform the two following steps. In the first step, the digital content is hashed again, as in the signature generation process. Then in a second step, the resulting reconstructed hash value is used as an input to the signature verification algorithm, together with the signature attached to the digital content and the public key. If the algorithm determines that the signature is authentic, this proves that the digital content is identical to the original digital content (the integrity), and that the author of this digital content is really who he claimed to be (the authenticity) (**Figure 1**).

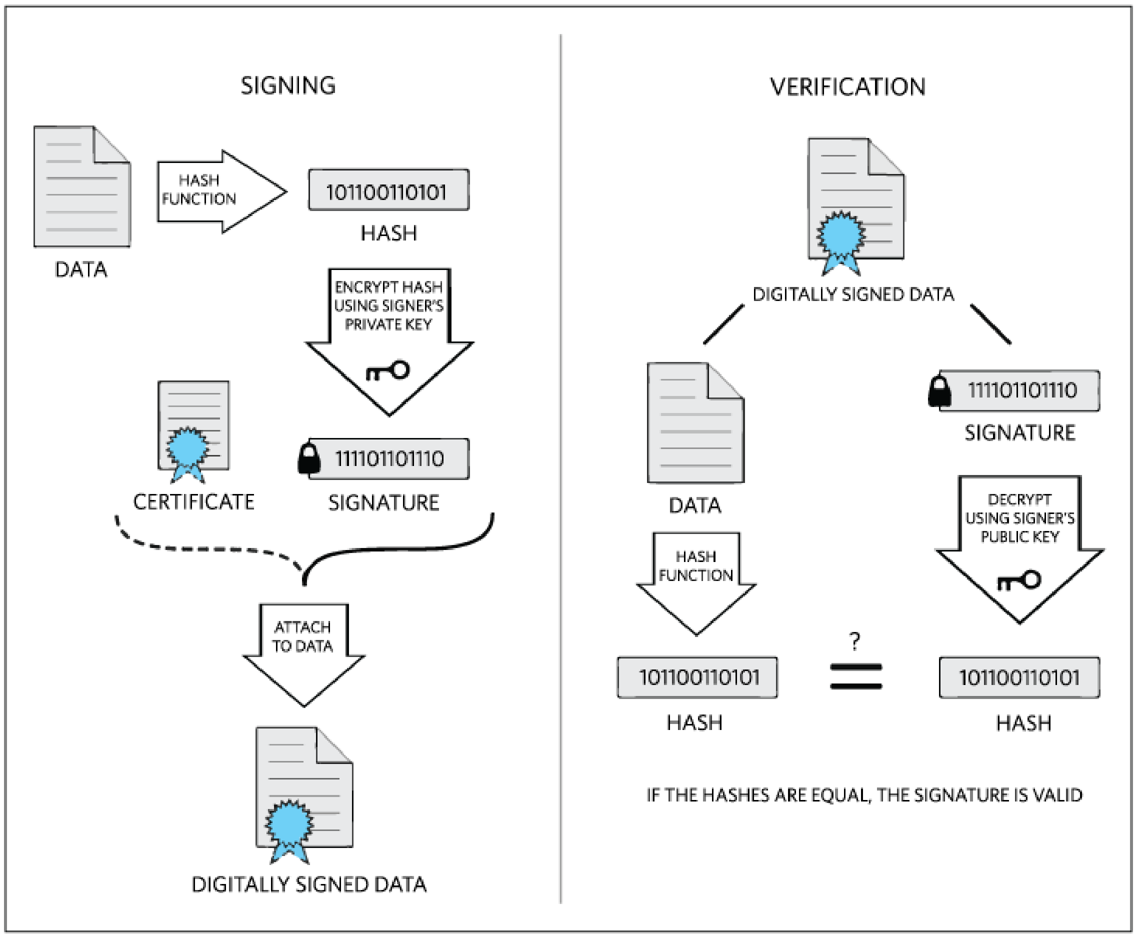


Figure 1. A diagram of a digital signature, how it is applied and verified. Illustration Licensed under Creative Commons Attribution-Share Alike 3.0 via Wikimedia Commons, http://commons.wikimedia.org/wiki/File:Digital_Signature_diagram.svg#mediaviewer/File:Digital_Signature_diagram.svg

Public key-based digital signature schemes work because the private key can be used for signing content only by the owner of this private key and no one else. Therefore, the private key has to be kept secret in good hands. Yet the public key need not be confidential because anyone can verify a digital content's signature. The only fundamental requirement for a public key is trustworthiness. Please note that public here does not mean insecure. The public key is freely accessible because it gives no indication about the private key; one cannot calculate the private key knowing the public key. The public key, moreover, does not allow anyone to perform personally identifiable actions like signing digital content.

Nevertheless, as anyone can generate a pair of keys, there must be a mechanism to verify the identity of the public key owner. Suppose that a public key has no strong binding with an identity. Then if you successfully verify the digital signature of a digital content with that public key, you still cannot trust this digital content because you do not know who actually signed this digital content.

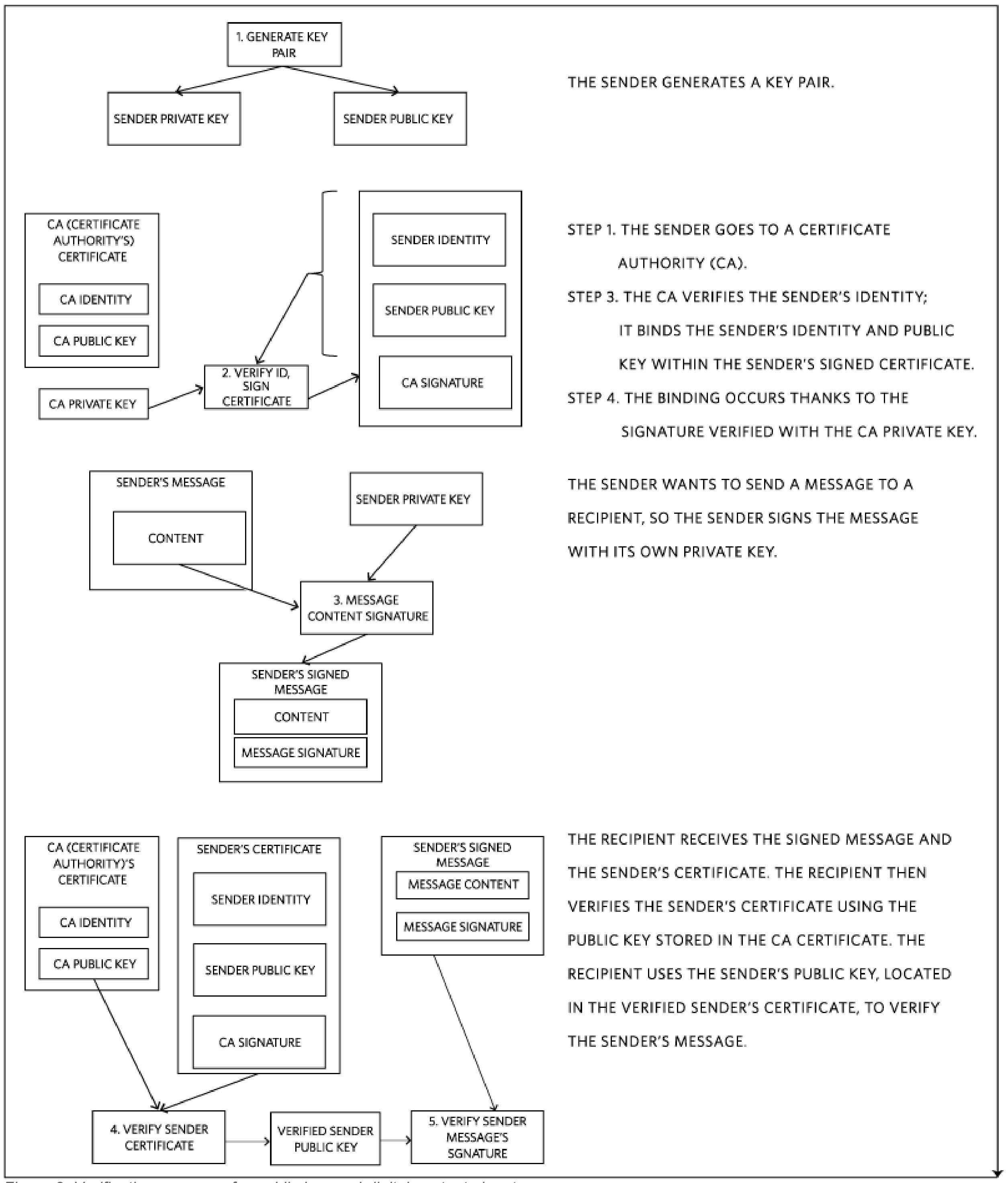
Therefore, public key *integrity*, *authenticity*, and *identity* must all be guaranteed. This can be done in different ways.

- **Method 1: Self-certification.** The recipient of the digital content receives the public key from the sender in person, or the sender transmits the public key in a way that leaves no doubt about the legitimate origin and ownership of the public key. Then this public key (also called a root key) can be trusted, as long as it is stored where unauthorized persons cannot modify it.
- **Method 2: Hierarchical certification.** In this method a hierarchy of verifiers guarantees the origin of the public key. Public key infrastructures (PKIs) provide the definitions of such hierarchies. The physical association between a public key and the identity of the key owner is a certificate. Certificates are signed by intermediate entities (i.e., certification authorities) of the PKI hierarchy.

Assume that a person wants to have a certified public key. That person generates a pair of keys and keeps the private key in a safe, hidden place. Then in principle, a certification authority meets this person face-to-face and thoroughly verifies the identity of that person. If authenticated, the identity information (name, organization, address, etc.) is attached to the public key and the resulting document is signed by the certification authority's private key. This permanently binds the identity information to the public key. The resulting signature is attached to the certificate. If any one element among the identity information, the public key value, or the certificate signature is tampered with, then the certified signature becomes invalid. Therefore, the information contained in that certificate cannot be trusted. The certification authority's public key can, in turn, be certified by yet another certification authority.

Certificate validity is verified by using the same cryptographic signature verification scheme as for digital content. The signature verification of the certificate guarantees the integrity and authenticity of the certificate and, consequently, of the information contained in the certificate: the public key and the identity (Figure 1).

As a result, before using a public key, one must first verify the validity of that public key's certificate by using the certification authority's public key. Then make sure that this certification authority’s public key certificate is also valid by using the public key of the parent signing authority, and so on. Therefore, a chain of verifications can occur with successive certification authority's public keys until it is ultimately trusted as a root key. The root key, you will recall, is trusted because it was obtained using Method 1.



THE SENDER GENERATES A KEY PAIR.

STEP 1. THE SENDER GOES TO A CERTIFICATE AUTHORITY (CA).

STEP 3. THE CA VERIFIES THE SENDER'S IDENTITY; IT BINDS THE SENDER'S IDENTITY AND PUBLIC KEY WITHIN THE SENDER'S SIGNED CERTIFICATE.

STEP 4. THE BINDING OCCURS THANKS TO THE SIGNATURE VERIFIED WITH THE CA PRIVATE KEY.

THE SENDER WANTS TO SEND A MESSAGE TO A RECIPIENT, SO THE SENDER SIGNS THE MESSAGE WITH ITS OWN PRIVATE KEY.

THE RECIPIENT RECEIVES THE SIGNED MESSAGE AND THE SENDER'S CERTIFICATE. THE RECIPIENT THEN VERIFIES THE SENDER'S CERTIFICATE USING THE PUBLIC KEY STORED IN THE CA CERTIFICATE. THE RECIPIENT USES THE SENDER'S PUBLIC KEY, LOCATED IN THE VERIFIED SENDER'S CERTIFICATE, TO VERIFY THE SENDER'S MESSAGE.

Figure 2. Verification process of a public key and digital content signature.

Verifying Software with Public Key-Based Signature

When applied to software, this public key-signature technique allows you to trust executable binary code. Now you simply consider the software as digital content. The sender of this digital content is the software approver, the one charged with accepting the software validated for a device. The receiver is the electronic device. The software approver generates a pair of keys and loads the public verification key into the electronic device once during manufacturing. The private key is kept in a safe place, as explained below. The software approver signs the generated code before loading it into the electronic device by using its own private key. Then at power-on, the electronic device can use the preloaded public key to verify the integrity and authenticity of the binary code before running it.

As an interesting aside, you can go to the **Sidebar** below to read about a secret key, its limitations and how to protect it.

Public Key Cryptography and Countermeasures

Public Key Management

Public keys do not need to be protected against disclosure and, therefore, do not require any of the countermeasures designed to prohibit access to the key value. Unlike a secret key, a public key does not need to react to tamper events by deleting/erasing its verification key. No side-channel countermeasures are required. The only required protection mechanisms must target keys substitution/modification and modified software behavior. All this makes the device design simpler. The algorithms involved are not subject to export regulations because they do not include encryption, but merely a digital content digest (i.e., a hash algorithm and signature verification). Note, finally, that the digital signature verification algorithm (Figure 1) must still be robust enough to protect against intentional or accidental disturbances: power glitches, badly formatted digital content and digital signatures.

Private Key Management

As explained above in the presentation of the public key-based digital signature scheme, public-key cryptography is based on pairs of keys. (A key pair is made of a public key and a private key.) The private key is stored secretly because it allows a receiving device to authenticate content–only the key owner should be able to sign content. Conversely, the public key is available to anyone because anyone can verify a signature. This is not harmful or risky.

Obviously proper management of a private key is a critical requirement for key-based cryptography. Anyone who steals a private key can sign arbitrary executable code that will then be successfully verified by the electronic device. Therefore, software approvers must store the private key somewhere strongly protected from disclosure; they must make sure that no one can use the key (even without disclosure). Some certifications such as PCI PTS 4.0 require the use of hardware security modules (HSMs) to manage keys. HSMs are tamper-resistant devices capable of securely generating and using pairs of keys. While public keys can be exported, the associated private keys remain in the HSM. Using the private keys in the HSM (e.g., for signing digital content) requires a prior strong multifactor authentication. The person who needs to sign a binary executable code must use a smart card and a PIN code to unlock the private key in the HSM. Depending on the security policy, two persons or more may be required for that operation. In addition, HSMs have to be kept in safety deposit boxes to maximize the security when they are not used. This process ensures that only trusted persons sign binary executable code.

Benefits of ECC vs. RSA Cryptography

Public key-based cryptography has been an RSA algorithm for several decades. But in the last few years elliptic-curve cryptography (ECC) has emerged and spread through the security industry. Elliptic curve-based signature verification is the same order of magnitude compared to RSA, but uses far less computational resources. Its key sizes are much smaller, thus reducing the memory footprint. A secure application of RSA now requires at least 2048 bits of security; RSA keys need 256 bytes. Equivalent elliptic-curve keys are only 224-bits-long[7] and the keys are only 28 bytes. Elliptic-curve cryptography is, therefore, the preferred choice for securing newer devices.

Conclusion

Maxim Integrated has developed a public key-based secure boot mechanism that keeps production and key management simple. In addition, as for key management, Maxim Integrated fulfills the stringent PCI PTS 4 requirements.

Public-key cryptography removes some security constraints usually required for manufacturing subcontractors, including loading software in these devices. As public-key cryptography involves no secrets and yet the authenticity of the loaded software is guaranteed, we get the best of the two worlds.

Now we come full circle back to our opening discussion of the IoT. To ensure the integrity of the IoT, we cannot allow any security breach of electronic transactions, critical systems such as nuclear plants, or implantable medical devices. An IoT must secure financial, industrial, and medical devices if the broad-based communications are to thrive. Ultimately, IoT devices require the trust and the security mechanisms discussed here make that trust possible and simple to implement.

In our next application note, **Secure the IoT: Part 2, A Secure Boot**, we will explain that the best way to protect devices on the IoT is a secure boot, also called a root of trust. A secure boot is an unbreakable wall, a hard barrier against attackers trying to breach the casing of an electronic device. We will show how, ultimately, the best defense is to use a microcontroller that starts executing software from an internal, immutable memory.

Endnotes

1. See "Generating value from smart meter data," **Centre for Sustainable Energy**, www.cse.org.uk/projects/view/1210. managing big data for smart grids and smart meters: IBM white paper. For more background information, see Maxim Integrated application note 5832, "Water and Power in the Internet of Everything;" also Maxim Integrated application note 5536, "Energy Measurement and Security for the Smart Grid–Too Long Overlooked." Maxim Integrated application note 5144, "Environmental Benefits of Smart Meters." Maxim Integrated application note 5725, "Silicon, Security, and the Internet of Things." This application note also discusses the Internet of Everything.
2. Tang, Wincey, "Mobile Point-Of-Sale Shipments Surge by 50 Percent in 2013, but Many Devices Go Unused," **IHS Technology**, February 05, 2014, https://technology.ihs.com/487986/mobile-point-of-sale-shipments-surge-by-50-percent-in-2013-but-many-devices-go-unused.
3. Higgins, Kelly Jackson, "Smart Meter Hack Shuts Off the Lights," **DarkReading**, 10/1/2104, reports about Spanish smart meters that have been hacked, www.darkreading.com/perimeter/smart-meter-hack-shuts-off-the-lights/d/d-id/1316242. For more information, see Maxim Integrated application note 5537, "Smart Grid Security: Recent History Demonstrates the Dire Need;" also Maxim Integrated application note 5545, "Stuxnet and Other Things that Go Bump in the Night."
4. For more information, see Maxim Integrated application note 5486, "Securing the Life Cycle in the Smart Grid," and application note 5631, "Ensuring the Complete Life-Cycle Security of Smart Meters."
5. For a detailed introduction on public key cryptography, see http://en.wikipedia.org/wiki/Public-key_cryptography.
6. For a general introductory discussion of CRC, see http://en.wikipedia.org/wiki/Cyclic_redundancy_check .
7. For more information, see www.keylength.com.

Sidebar: Short Discussion of a Secret Key

Limitations of a Secret Key in the Field

Let us digress for a moment and talk about secret key cryptography. Secret key cryptography looks simpler than the public-key system discussed above because there is one unique key for signing and verifying signatures, and there is no need for certificates.

In fact secret key-based cryptographic algorithms such as AES, which is based on the cryptographic algorithm, FIPS 197: Advanced Encryption Standard. NIST, are not suitable for protecting software integrity in the field because the same secret key has to be used for signing and verifying signatures. The secret key must, therefore, also be stored inside the deployed electronic devices. However, protecting secret keys stored in electronic devices against disclosure is not an easy task. Disclosure of the secret key can happen at design, during manufacturing, and in the field. At design, insiders could leak the key to an outsider. At manufacturing, third parties could dump the secret keys from the device memory and leak them out. In the field, attackers can reverse engineer the key (e.g., memory dump, fault attacks, power analysis).[A]

Protecting a Secret Key

You can mitigate the disclosure of secret keys by secret key diversification, i.e., by using a different key for each deployed device. This makes sense, but the manufacturing becomes much more complex and requires trusted, secure manufacturing plants and huge databases of keys. Beyond manufacturing, you can also install anti-theft defenses in a device itself. A secure electronic device must detect tamper events and then destroy its secret keys. There must also be countermeasures to resist power disruptions or faults directed against the encryption algorithm.[B] Encryption algorithms are, moreover, subject to export regulations, so these devices can have regulatory issues involving international markets.

Secret key cryptography is conceptually simpler, but forces manufacturers to expose secrets in the field and implement expensive, and yet imperfect, countermeasures. The countermeasures described above protect a secret key, but they also make the design of a secure device more complex and costly. Secret key cryptography has been used against threats regarding confidentiality, but is too risky/expensive for only verifying software signatures. Ultimately, it resolves and protects against some security problems that can justify the burden of its heavy key management.

A. Cf. note 4 in the main article.

B. See the section in the main article, **Public Key Cryptography and Countermeasures**.

A similar version of this article appeared January 11, 2015 on *Embedded*.

Related Parts		
MAX32550	DeepCover Secure Cortex-M3 Flash Microcontroller	Free Samples
MAX32590	DeepCover Secure Microcontroller with ARM926EJ-S Processor Core	Free Samples