

An Overview of Secret Key and Identity Management for System-on-Chip Architects

Introduction

As data communications and applications security is increasingly a requirement for modern system designs, embedded systems architects are coming to realize that identification and authentication are a major component of the systems they are designing. A key technological feature that must be incorporated in these components is the ability to create and protect unique identifiers that serve to prove the identity of connected entities to each other. From a security point of view, it is not sufficient to simply believe a device that asserts a particular identity. PCI bus identifier strings and Ethernet MAC addresses fall into the category of *insecure* identifiers. These identifiers may be fine for configuring a device and the system it operates in, but when used to protect systems from unauthorized access, they simply don't provide sufficient security. For example, applications that authenticate network access for a single user using MAC addresses are easily fooled by re-programming the MAC address of gateway routers that are capable of hiding a whole network behind them.

For security applications between automated systems, the gold standard for authenticating access is the use of secure cryptographic protocols that require the systems that are party to a transaction or session to prove their identities using cryptographic techniques. In device identification and authentication schemes, there are typically two similar ways in which to do this: prove knowledge of a secret; or prove possession of a “token” that identifies the device. In most schemes, a token is itself an active device that performs the service of identifying and

authenticating itself on behalf of another device in which it is installed. So in most cryptographic protocols, identification and authentication of entities to each other reduces to proof of knowledge of a secret.

The requirements of the particular application dictate how well protected the identifiers must be. A system protecting million dollar financial transactions should probably be better protected than one that protects MP3 files from unauthorized copying. The challenge is to find a solution that trades off security for cost, manufacturing complexity, ease of use and other factors.

This paper discusses those issues and others. The paper begins with a brief overview of techniques for identification and authentication then follows that with a discussion of technical means to implement them in a System-on-Chip (SoC).

Identification and Authentication Techniques

Identification and authentication (I&A) are almost inextricably tied together, and much of the literature about them deals with them in combination. In simple terms, identity is an assertion made by an entity about its name. Depending on the system, an entity may choose its identity, or it may be assigned one by its owner or someone in the manufacturing and distribution chain. Simple examples of cases where a name is assigned include system manufacturers or network owners, which may wish to identify the systems they will interact with.

Authentication is the process of proving that an entity possessing a particular identity is the originator of a particular message. This proof is accomplished by using a secret known to the originator (and possibly also the recipient) to demonstrate to the recipient that the originator must know the secret.

If both sides share a secret (and don't tell anyone else) this may be used to build an authentication protocol from a symmetric cryptographic primitive like a cipher. Suppose Adam sends a message to Brenda saying “Prove your identity to me”. Brenda can use the secret she shares with Adam as an encryption key to encrypt the response “I am Brenda”, and sends that back to Adam. Adam uses the shared secret key to decrypt Brenda’s message. If the result decrypts properly, he knows that Brenda must have sent it. There are some issues with this kind of protocol: Adam and Brenda have to agree what secret they will use and the encryption method. If Adam and Brenda are people, they can phone each other up or exchange paper mail to agree on the secret, but that is more difficult if they are machines. And Adam can pretend to be Brenda at will. (For example, he can send himself an encrypted message and claim that Brenda was the originator.) For those reasons and more, it is usually not desirable for Adam and Brenda to authenticate their identity to each other in this way, although, this kind of protocol is surprisingly common. Many virtual private network (VPN) systems use pre-shared keys for both encryption and authentication. Similarly, certain digital rights management (DRM) schemes also use pre-shared keys that must remain secret for the system to remain secure.

One last thing to notice about this protocol: an attacker, the ever wily Eve, can pretend to be Brenda (or Adam) by simply sending a captured copy of Brenda's response message whenever she sees the query for Brenda to “Prove your identity to me”. The way to defeat this is for Adam to include a random piece of data (a challenge) in his query. Brenda must include the challenge in her encrypted response. I Brenda’s response includes Adam’s challenge he knows that Brenda (or someone with her key) must have sent it. Without it, her authentication is deemed to be a failure.

A better protocol will not rely on pre-shared secrets so that Brenda and Adam do not need to exchange

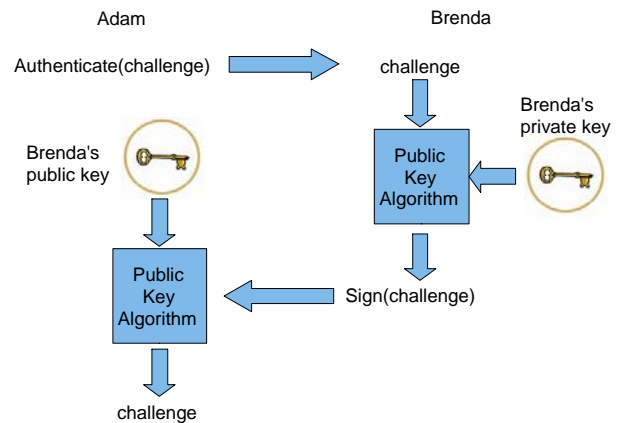


Figure 1. A public key authentication protocol.

secret information before they are able to authenticate the other's identity. Asymmetric cryptography has this characteristic. Brenda creates two keys: a public key that she can share with the world (for example by publishing it on her website) and a private key that she keeps secret. A message encrypted with Brenda’s private key can be decrypted with her public key. Now for Adam to authenticate Brenda’s identity (figure 1), he gets a copy of her public key, creates his random challenge message and sends it in cleartext to Brenda. Brenda encrypts the challenge using her private key and sends it back to Adam. Adam uses Brenda’s public key to decrypt the challenge, and if it is the same as the one he sent, it proves that she knows the secret key.

This is the essence of a one-way authentication protocol. The process of encrypting a message with Brenda’s private key is called “signing”, and the key Brenda uses for it is reserved for this purpose and referred to as a “signing key”. Brenda can likewise issue a challenge to Adam to have him authenticate himself, making the authentication mutual. Real signing and authentication algorithms are slightly more complicated, but not a lot. Many authentication protocols use the FIPS 186-2 standard for their signing and authentication operations. These algorithms are shown in Figures 2 and 3.

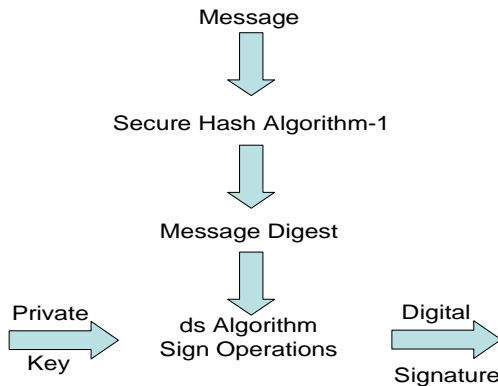


Figure 2 DSA Sign Operation

This protocol covers authentication, but does not explicitly deal with identification. There is an implicit understanding of the identity of the authenticated party, but no explicit identity associated with the protocol. For example, how does Adam know that the website with “Brenda’s” public key on it is not actually a decoy created by Eve? What is required is a means to tie an identity to the private key used for authentication. In most systems, this is accomplished by a third party or Certification Authority (CA) that is trusted by an authenticating party to vouch for the association between a public key and an identity. A CA is a “Root of Trust” and its identity can inherently be trusted. For example, the root keys for Verisign are signed by Verisign itself (along with other root CAs). These are distributed in web browsers and other software applications, as well as being listed on Verisign’s website. To publish her identity, Brenda sends an

electronic file that includes her name and other identifying information together with her public key to a CA like Verisign. The CA will confirm that Brenda submitted the request and perhaps that the information in it is correct (e.g. by checking with her bank), and if it is, signs the datafile that contains the public key and identity information using its signing key. The signed datafile is called a certificate, since it certifies Brenda’s identity. Because the signed file includes both identity and public key, the identity is said to be “bound” to the public key, and implicitly to the underlying private key.

Implementing Unique Identity in SoCs

What are the implications of these needs on SoC architecture and design? Throughout the previous section, we referred to the necessity to store secrets. To maintain the security of the system, secret key storage makes several demands of the system: secrets must remain secret; secrets must be unguessable; and they must be randomly generated. Failure to observe these requirements may result in compromise of the system and loss of its security. Particular cryptographic schemes may impose other technical requirements on the form or properties of secrets.

In addition, in some applications it is necessary to be able to create the secrets when devices are initially installed, or subsequently throughout their lifetime; and to be able to destroy stored secrets when certain kinds of attacks are detected.

To know how to answer the questions posed by these needs it is important to understand the threat environment and costs of compromise should one occur. We can catalog the threats into one of the following categories

Passive attacks include traffic analysis, eavesdropping, decrypting weakly encrypted traffic, and capturing authentication information or other security

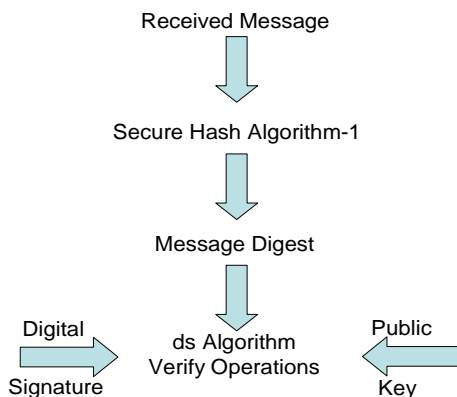


Figure 3 DSA Verify Operation

parameters (e.g. passwords, cryptographic keys moved in cleartext over an external bus). Consequences are usually limited to data disclosure, although sloppy design can leak keys that allow an attacker to produce forged devices that can pass all cryptographic authentication.

Active attacks include attempts to circumvent protection features, introduce malicious software or firmware code, or steal or modify data. Consequences include data disclosure or corruption and service disruption.

Close-in attacks are those in which an unauthorized entity gains close physical proximity to resources including critical security parameters with malicious intent. This can be due to either unsecured resource access (e.g. through poorly implemented scan and BIST design, debugging ports and registers), through side-channel attacks (e.g. timing, power analysis, RF emanations), and deliberate reverse engineering with the intent to recover keying data and/or design details.

Insider attacks include both malicious and non-malicious actions of authorized users. Non-malicious actions may be due to carelessness, incompetence, or irresponsible circumvention of security (i.e. short-cuts). Examples include leak of keys coded into ROM or gates, etc.

Distribution attacks are modifications introduced during the hardware/software manufacturing or distribution process. These include software backdoors, unauthorized access to key provisioning systems, etc.

The desired strength of a security solution can generally be defined in terms of the estimated cost of successfully attacking the solution under a defined threat model. The business case, or Return On Investment (ROI) of a solution can in turn be measured by comparing the costs to implement the system vs. the resulting cost if the system is compromised (i.e. risk analysis).

There is no such thing as perfect security. Or more accurately, a “perfect” security solution has infinite cost and zero performance. However, a good security policy utilizing strong cryptographic tools and methodologies should render a system secure enough to protect it from all but the most expensive or elaborate Passive, Active or Close-in attacks from unauthorized persons. Further, it should limit, as much as possible, exposure to Insider attacks by supporting thorough identification and authentication, and the related function of authorization, of users and equipment and minimizing the group of authorized users to those that genuinely require access at the appropriate level of authorization.

Malicious Attack Scenarios

SoC designers are relatively lucky in that they have access to the technological means to create very secure systems that include stored secret data such as cryptographic keys. Among the best of these techniques is the use of unique-per-chip embedded secret storage in non-volatile memory. The advantage of on-chip hardware embedded keys is that they can be extracted only by a determined and knowledgeable malicious party with technical expertise and financial resources sufficient to mount an attack.

Physical attacks may include a variety of techniques including decapsulation of the SoC package, layer-by-layer removal and analysis of the IC dice, surface probing, cross-sectional X-Ray microanalysis, optical microscopy, and e-beam analysis to enable reverse engineering of the on-chip secret keys. The SoC technology employed to store the secret keys will impact the cost and effort of a physical attack. The use of unique keys per integrated circuit instance is almost completely effective at defeating destructive attacks. While a single key can be recovered through such attacks, it will generally be very difficult to reuse that key to useful effect.

Attacks may be mounted through on-chip resources by analyzing logic outputs of the system. Attackers will routinely attempt to recover secret keys through any accessible external port, including those not normally intended for system users. Features such as debug ports and scan chain outputs are examples of this kind of attack. Such attacks can be curtailed by designing the BIST and scan logic to produce aggregated outputs that provide feedback about correct operation of the relevant registers without providing their values directly.

Side-channel attacks (e.g. simple and differential power and timing analysis) exploit information that leaks from a cryptographic device such as power consumption signatures to uncover the secret keys buried on-chip. Good design will employ various types of protection against these types of attacks, including the use of additional interference (chaff) or the reduction in noise created by the chip through special filtering circuits. Again, careful design can eliminate the root cause of this effect through a dedicated design methodology applied to critical parts of the crypto engines. Another option is to make the crypto algorithm perform in a different way each time it runs, introducing randomness.

No device can be made totally safe against all forms of attack; however, the damage can be limited by using unique, randomized keys for each device so that any information garnered is of no utility beyond the one device compromised. The various common methods to provide this capability are now discussed.

One-Time-Programmable (OTP) Memory

OTP memory uses permanently programmed memory cells to implement small memories with good security properties. These memories represent the most secure solution available for deployed systems. Access to them can be tightly controlled, and they are difficult to reverse engineer through attacks

as sophisticated as E-beam microscopy, IC decapsulation/layer stripping, RF emanation detection and the like using expensive professional tools.

OTP memories are programmed during IC or subsequent system manufacturing. They can be designed to disable access via scan chain and BIST following initial programming of the keys. In addition, in some technologies, it is possible to destroy the stored keys in response to tamper attempts, as well as write new keys.

OTP memories are relatively recent innovations, and are available in only a few process technologies. This situation will improve over time. In addition, OTP memory must be qualified in each target process and usually requires a license from the IP provider to use the technology.

Security of the keys may be maximized if the IC generates its own keys using hardware designed into the device for this purpose. However, except in systems where the security requirements justify the cost of this solution, it is usually not feasible to generate and program secrets internally. In most cases, therefore, it is necessary to set up and manage a secure key injection process to install randomly generated keys into the devices. This process can occur during wafer test, coincident with final package assembly and test, or in a separate process following delivery of packaged devices should the selected OTP memory technology allow.

Embedded Flash Memory

Embedded Flash memory may be used in a similar manner to OTP memory. Flash memories are typically larger than OTP and their physical structure makes them more susceptible to reverse engineering using professional tools. To maximize security, a separate small flash memory reserved for secret keys should be used, rather than using space in another memory that may be used for different, non-security related purposes. After OTP memory, embedded Flash is the next most secure technology.

If high security is an objective, care must be taken with the design of BIST and scan circuitry to ensure that keys do not leak through this path.

Embedded Flash memory requires a special fabrication process. Typically, the most advanced process nodes do not have qualified Flash memory technologies. In a sense, embedded Flash is complementary to OTP, since older processes support Flash but not OTP, whereas more advanced process nodes tend to have OTP but not Flash. Programming Flash memories requires a similar key injection process to that of OTP.

Fuse Technologies

A variety of fuse and antifuse technologies are available that are suitable for programmable key storage. Fuses are programmed using laser or electrical programming methods. Fuse structures are physically large and the programming techniques often leave visible clues to the state of individual fuses that may be observed under an optical microscope. This makes them vulnerable to reverse engineering, sometimes only partially destructive (e.g. careful decapsulation to recover a key in a still operational IC).

Laser fuses are programmed at probe. Antifuses or eFuses can be programmed using on-chip charge pumps which permit the fuses to be configured at probe, final test and end product production in a similar to OTP or embedded Flash.

RTL Keys

In some applications, security is secondary to cost. In such applications it is possible to consider permanently encoding the keys in RTL, i.e. gates. This method is always a compromise, and it should not be considered in any application where it is not permissible for the keys to become known. Like all keys, RTL keys must be chosen randomly. In this case, the IC designer must generate a random key

of the appropriate number of bits during the design process. This key is then encoded as a permanently stored key in gates. Naturally, it is important that as few people as possible know the key as all ICs of the same design share the same key.

RTL keys are a very weak form of security and should be avoided whenever possible. Because a single key is common to every IC of the same design, compromise of the key compromises the entire design. Besides compromise through insider leaks of the design value, RTL keys are susceptible to reverse engineering through destructive analysis techniques, and if incorporated in the scan chain, can be leaked through the JTAG port. A physical design that distributes the gates for the key around the IC can help obfuscate the value, but ultimately does not prevent recovery of the key through design analysis. A key management module, discussed below, can help to mitigate these risks but it must be understood that RTL keys are inherently insecure.

Separate Key Memories

Depending on the larger system context, separate key memories such as serial E²PROM may be used. These are most effectively used in multichip modules which package the memory together with the SoC, but are also appropriate to security module designs that draw the cryptographic boundary around the subsystem composed of the memory and the SoC. Depending on the application, security considerations now extend to physical design of printed circuit boards, busses and backplanes. Like other memory technologies, key injection is the most usual to provision devices with secret keys.

Software Keys

The weakest (but certainly the most widely deployed) form of security is software based keys, loaded into the hardware when needed, typically at system initialization. While it is possible for every

instance of a design to have unique keys, software keys can be attacked through both software and hardware based means to discover the keys. At best it is possible to obfuscate software keys and the ways in which they are used, but it is virtually impossible to secure the keys against discovery by a determined attacker.

Enhancing Embedded Key Protection with Key Management

While embedded secret keys may be used directly with symmetric or asymmetric ciphers, a more secure solution is to use a Key-Wrap Module (KWM) to enhance the security of the stored embedded secret. The KWM makes use of the embedded secret key to cryptographically wrap the potentially large number of symmetric or asymmetric keys so that they may be safely exported for storage in low cost, insecure media or for transport to other hosts. A KWM provides good ROI to implement a unique on-chip secret using more expensive but highly secure SoC technologies around a single key rather than having to utilize expensive SoC technology for a large number of keys.

A KWM also enables the use of SoC key storage technologies which may only be available for programming at wafer probe, well in advance of the availability of the symmetric or asymmetric keys.

In the KWM, the stored embedded secret is referred to as a Key Generating Key (KGK). The KWM initialization process transforms the KGK, combined with an optional software-supplied initialization vector (IV), through a cryptographic process to produce a master Key Encrypting Key referred to as KEK0.

This use of the KGK isolates the stored secret on the IC from the keys used for other cryptographic operations. In order to attack the security of the system, it is necessary to recover the stored key, reverse engineer and understand the KWM, and recover the software IV corresponding to the par-

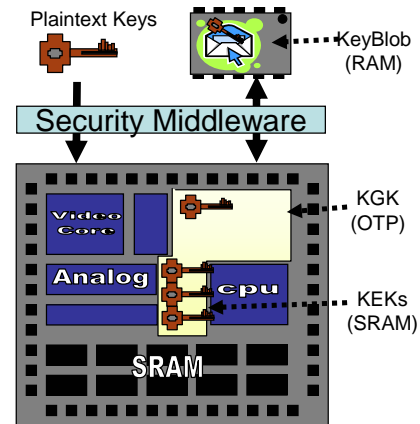


Figure 4. SoC with Key Wrap.

ticular KGK. While not impossible, the effort required to recover and make use of the key is significantly greater than using an unadorned key, and requires a sophisticated combination of hardware and software engineering skills and professional grade tools to attack the system.

KEK0 is used exclusively to encrypt other key encryption keys. These are then used to wrap data encryption keys (DEKs). Unwrapped DEKs are transferred directly to the cryptographic hardware that they are to be used in. As a result, DEKs and KEKs are never exported in plaintext from the cryptographic subsystem in which they are used.

Use of the KWM affords some extra protection for systems that wish to use RTL keys. Combining an RTL key with a software IV through the KWM initialization process produces a KEK0 that is completely uncorrelated with the RTL key and the IV. KEK0 is never exported from the hardware, so the key used to encrypt other KEKs (and hence DEKs) is effectively secret. If software IVs are unique and randomly chosen in each system instance, the security of the system approaches that of systems incorporating uniquely programmed secret keys. Two identical systems using different IVs have no way to reuse or attack keys created on the other.

Summary

In applications requiring secure storage of embedded secrets for cryptographic identity and key storage applications, the gold standard is unique per-chip keys programmed in OTP or embedded Flash memories. Fuse and external memory technologies have applications where physical integrity of the integrated circuit or security subsystem can be maintained to protect the stored secret. Where less security is required, RTL keys can be used, but great care must be exercised in their use. In general, the use of software keys should be discouraged. In all cases incorporating hardware keys, a KWM can be used to provide greatly enhanced security of stored keys.

For more information on Elliptic products and services please contact us at:

Elliptic Technologies Inc.
62 Steacie Dr., Suite 2021
Ottawa, ON, K2K 1Y6
Phone: 613 254-5456
info@elliptictech.com