White Paper

Automotive Security: From Standards to Implementation

Richard Soja
Automotive SoC Systems Engineer, Freescale

Abstract

The increasingly interconnected nature of a vehicle's control modules means there is no safety without security. Security features must include not just physical access and protection of confidential information, but also critical safety systems. Designers must anticipate every form of attack to prevent access to embedded systems and data. At the same time, the industry at large must develop standards, specifications and guidelines for vehicle security that enable interoperability.

Table of Contents

freescale™

Introduction

There has been a steady increase in demand for improving the security of microcontrollers in the automotive market. When car electronics were first introduced back in the 1970s, the functions implemented were quite simple—they were discrete and unconnected to other components in the vehicle. The component that controlled the spark plugs in the engine did not communicate with the speedometer or tachometer in the dashboard. Long before the days of Bluetooth® cell phone controls or integrated audio systems, a trip computer might be the most cutting-edge feature found in a new vehicle.

Over time, the combination of increased electronic complexity and integration with other fixed and portable components (e.g., keyless entry, audio systems, telematics, wireless communications, etc.) has provided portals into the control systems that are deeply embedded in the vehicle.

User-accessible systems potentially contain personal and private information, while the embedded systems are inextricably tied into the fundamental physical behavior of the vehicle. These two aspects mean that if any of the vehicle control and information systems are compromised, the opportunity for theft or damage from external entities appears.

Today's cars are also expected to hold even more private information as they become smart cards on wheels to simplify financial transactions at gas pumps, charging stations, parking lots, toll booths and drive-through establishments. The vehicle itself will be enabled to pay fees and fares, sometimes automatically.

Security has come a long way since the initial introduction of simple features like car alarms and keyless entry. In today's vehicles, security features must include not just physical access and protection of confidential information, but also critical safety systems such as drive-by-wire braking and steering. The increasingly interconnected nature of a vehicle's control modules means there is no safety without security.

The nature of the automobile industry itself also introduces some additional interesting security considerations. In order to support a very long and reliable operating life (which may be an order of magnitude longer than most consumer products), the installation of counterfeit parts and control units must be prevented. The vehicle's security systems can provide a means to do this using an authentication protocol.

Another factor that can affect system reliability is the practice of "chipping" or modifying the operating parameters stored in memory. Security features can prevent "chipping" altogether or can detect the presence of any unauthorized modification and take action to mitigate or eliminate the effects of the modification so that the vehicle is still safe to drive, while indicating the need for corrective attention.

Defining the architecture and implementation of secure microcontrollers requires a unique mindset. The designer must think like a hacker and come up with bulletproof ways to anticipate and prevent access to secure data.

The great range of available attack mechanisms (often referred to as the attack surface) normally means that designers must make a trade-off between the developer's cost of

protecting against an attack (or a customer's revenue lost as a result of an attack) versus the hacker's cost of mounting the attack. For example, if it is necessary to reverse engineer the silicon to uncover security codes, it might not make commercial sense to attempt this on something like a TV remote control.

There is also a great deal of activity in the industry at large to develop standards, specifications and guidelines for vehicle security. Standardization allows for greater interoperability between the various component suppliers to the auto industry. Having set specifications means that all manufacturers have an opportunity to develop security-aware products without compromise.

Being able to follow published guidelines allows the manufacturer implementation freedom while adhering to the overall architectural requirements and specifications that enable interoperability.

Standards, Specifications and Guidelines

Antiquated methods of "security by obscurity" offer highly precarious and ineffective approaches for protecting most modern environments. Today's most robust forms of security and encryption are those that survive scrutiny—in other words, security and cryptographic algorithm specifications that themselves do not have to be kept secret but are instead distributed in the public domain.

Within the automotive engineering community, a number of specification activities are either ongoing or have reached sufficient maturity to be accepted as a standard. For example, the Secure Hardware Extension (SHE) specification developed by Escrypt for Audi and BMW via the HIS Working Group, with early cooperation from Freescale in 2008, has now been accepted as an open and free standard.

The SHE specification defines a set of functions and a programmer's model (API) that allows a secure zone to coexist within any electronic control unit installed in the vehicle. The secure zone's most significant features are the storage and management of security keys, plus encapsulating authentication, encryption and decryption algorithms that application code can access through the API. These features help maximize flexibility and minimize costs. A later section of this white paper includes a description of the architecture of the SHE implementation on Freescale's MPC5646C single-chip microcontroller targeted at body control applications, where the security functions can be used for vehicle and ECU theft protection such as immobilizer activation.

The EVITA project, funded by the EU, has developed a set of guidelines that details the design, verification and prototyping of a range of security architectures for automotive ECUs. A number of companies have been active in the EVITA project, including BMW, Continental, Fujitsu, Infineon and Bosch. EVITA defines the overall functionality of three different hardware security module approaches: –full, medium and light. Moreover, it specifies an elaborate set of functions and their parameters for managing security keys as well as encryption and decryption operations.

A new European funded project called PRESERVE has emerged from the cooperating entities involved in EVITA. The aim of this new project is to develop, implement and test a scalable

security subsystem for Vehicle-to-Vehicle and Vehicle-to-Infrastructure (conflated to the acronym V2X) applications. The ongoing work is expected to be completed by the end of 2014.

The efforts of the PRESERVE project are targeted at demonstrating the secure transmission of data and control information for the future Intelligent Transportation System (ITS). The hardware security module implementation will include Elliptic Curve Cryptography (ECC), which is a form of public key cryptography.

Another good example of a security standard comes from the National Institute of Standards and Technology (NIST), which has issued the FIPS (Federal Information Processing Standards) 140 standard for both software and hardware components. FIPS 140-2 defines four levels of security ranging from Level 1 with a simple single security function and no physical security requirements up to Level 4 that mandates physical tamper detection mechanisms and protection against environmental attacks, such as voltage and temperature.

Freescale's P2041 devices support several encryption and authentication keys that are identified as being Critical Security Parameters (CSPs) in the FIPS 140-2 specification. An overview of P2041 security capabilities is described later in this white paper.

Other somewhat proprietary specifications and guidelines exist to aid the development of secure embedded systems. ARM® developed its TrustZone® security infrastructure, which has been integrated into microcontrollers and microprocessors from various manufacturers, including Freescale's i.MX series and Vybrid family of devices.

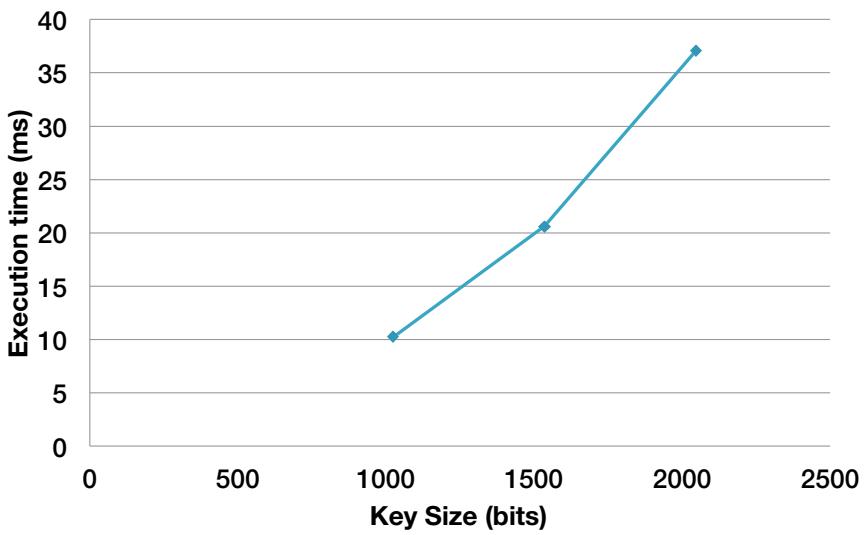
Another standards organization is the Trusted Computing Group (TCG), which claims to provide open, interoperable and international standards for trusted computing. One specification released by this organization is their Trusted Platform Module (TPM)—published as ISO/IEC 11889 Parts 1-4. Like the SHE specification, TPM supports secure keys for authentication and encryption functions. Developers generally implement TPM as an external peripheral with a communication bus to another microcontroller in the system. TPM specifies non-volatile memory, secret key storage, a random number generator, RSA, SHA-1, HMAC and Vernam one-time pad algorithms. The Advanced Encryption Standard (AES) is optional for TPM devices.

Security Mechanisms

The mechanisms needed to manage the security of an application may be implemented in software, hardware or a combination of both. In general, some form of software execution is always needed. Hardware is usually provided to accelerate the execution of the cryptographic algorithms to meet the performance requirements of the application. For example, an SHA-256 algorithm used to checksum the contents of memory could easily be two orders of magnitude faster with hardware acceleration, in comparison to a purely software-based equivalent.

The benefits of hardware acceleration become even more compelling for asymmetric cryptographic algorithms such as RSA and ECC, especially as the key size increases. Figure 1 shows the relative increase in computation time for software-based RSA authentication using a public key. With a hardware accelerator, the same public key authentication operation can be executed in under 100 microseconds.

FIGURE 1. Example of Relative Software Execution Times for Different Public Key Sizes



The partitioning between hardware and software implementations must also take in to consideration the types of malicious attacks that must be protected against. Typically, an attack will occur because malicious software has been allowed to execute during the boot process or during normal run time.

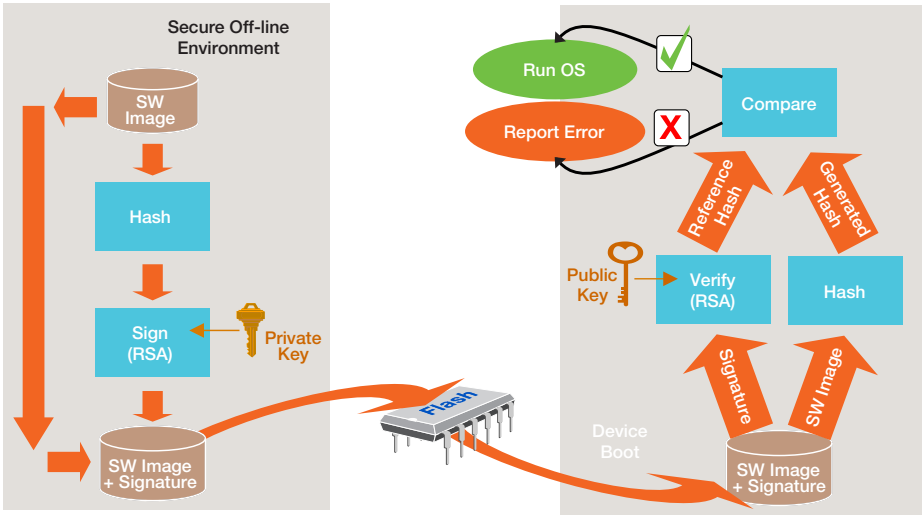
One approach to detect and mitigate boot code infection would be to set up a root of trust mechanism that authenticates the boot code before it executes. The authentication can be performed by a dedicated security module implemented entirely in hardware or a combination of hardware and software integrated in a security coprocessor. The challenge here is to guarantee that the root of trust is itself not infected with malware. One example of such a mechanism is the TPM specification mentioned in a previous section.

If the root of trust is based on software executing from embedded flash memory, then one way to ensure it is indeed uninfected is to use a secure, authenticated method of flash reprogramming. This practice is already implemented by some embedded control systems manufacturers.

Figure 2 shows an example of a flash programming protocol that relies on opening a channel between an external (offline) programming tool and the microcontroller. The example shown uses an RSA private/public key pair to sign and verify that the code is authentic and has not been modified by an attack during the programming process. In the secure offline environment, a hash is made of the software image, perhaps using SHA-256. The hash value, which uniquely represents the software image, is then signed with a private key that uniquely identifies the owner of the software. The resulting signature plus software image is then transmitted to the embedded memory system, which performs its own hash on the software image.

The embedded system also authenticates the signature received from the offline environment using the public key associated with the private key that produced the signature. The authentication procedure results in a hash value that must match the value from hashing the software image. If they match, it means the software image has not been modified, and it was

FIGURE 2. Secure Flash Programming Example



received from the entity that used the corresponding private key. This confirms both that the software is good and where it came from.

This methodology does not encrypt the software that is being programmed, nor does it need to hide the signature or public key. Instead, the private key (which is stored “offline”) must be kept secret because it defines the identity of the provider of the software image. The advantage of the private/public key authentication mechanism is that only one private key is needed and it is not embedded in the application device. Therefore, its exposure to attacks can be quite low and extensive measures can be adopted to maintain its secrecy because it is essentially kept in “offline” storage.

At the same time, the single public key that matches the private key can be proliferated in all embedded devices and does not need to be hidden nor encrypted. The only provision is that a secure method must be enforced to restrict how the public key is changed, to allow for the case where the private key has been stolen. For example, an authentication or password-protected protocol could be used to allow the existing key to be replaced with a new one supplied by a certified entity. An alternative mechanism would be to revoke the existing public key and replace it with another key from a list of preprogrammed keys or keychain.

The signature could equally well be created and verified using symmetric cryptography, in which case the same key is used to create and verify the signature. Unlike asymmetric keys, the symmetric protocol relies on the embedded system and external entity both being able to secretly store the same key. A successful attack on either the microcontroller or external entity will compromise the security of the system.

The SHE specification defines a protocol for securely changing secret keys. The protocol uses a symmetric AES-128 CMAC for signing and authentication and an AES-128 CBC for encrypting and decrypting the message associated with key updates. This prevents attackers from obtaining the key by snooping the update process. AES-128 offers the advantage of encrypting and authentication algorithms that are typically much faster than the RSA algorithms used with public/private key pairs.

Flash programming authentication determines the trustworthiness of the code image prior to placing the microcontroller in an application. Once installed in the final system, however, further measures are needed to ensure the integrity of the code (which is expected to remain unchanged) is not modified by malware (such as a Trojan) while the application is running.

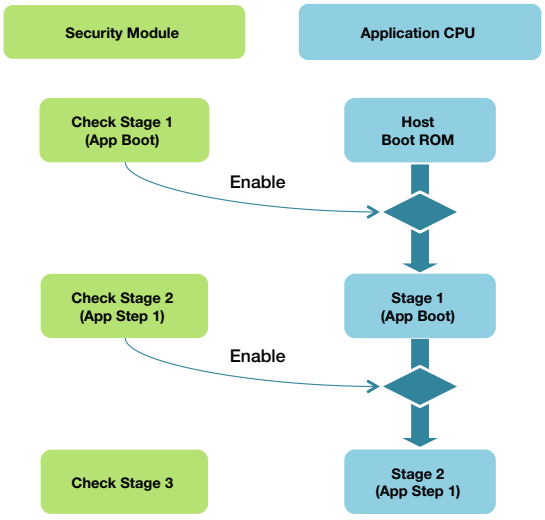
Checking the integrity can be done prior to running the application using a secure boot technique that executes the integrity checker from a root of trust. This eliminates the opportunity for code to execute if it or the data image in memory appears to be corrupted.

Alternatively, if no known software root of trust can be guaranteed, configuration and enablement of a run time integrity checker can be achieved solely in hardware and could execute prior to the application code starting. This technique also avoids the integrity process vying for memory bus bandwidth with the application process. If the resultant additional delay in start-up time is not acceptable, then another option might be to implement a run time integrity checker that executes in parallel with the application code, sharing memory bus bandwidth with the application.

The trade-offs between the two techniques are start-up time and memory bus bandwidth sharing, but there are mitigations for both these effects. Secure boot time could be shortened by using a chain of trust strategy which splits up the memory space into multiple partitions, each of which is checked in sequence as the application software progresses through its flow. Figure 3 shows an example of this strategy.

To minimize the impact that run time integrity checking has on the bandwidth loading for the application, some rudimentary Direct Memory Access (DMA) capability offering burst mode access to multi-ported system memory, plus local caching of data, can be implemented.

FIGURE 3. Chain of Trust



Another approach which may add to the security of software execution is the TrustZone architecture implemented on ARM-based products, such as the i.MX6 applications processor. TrustZone architecture provides partitioning of hardware that allows a single core to be operated in either a secure or unsecure “world.” The selection of worlds is orchestrated by

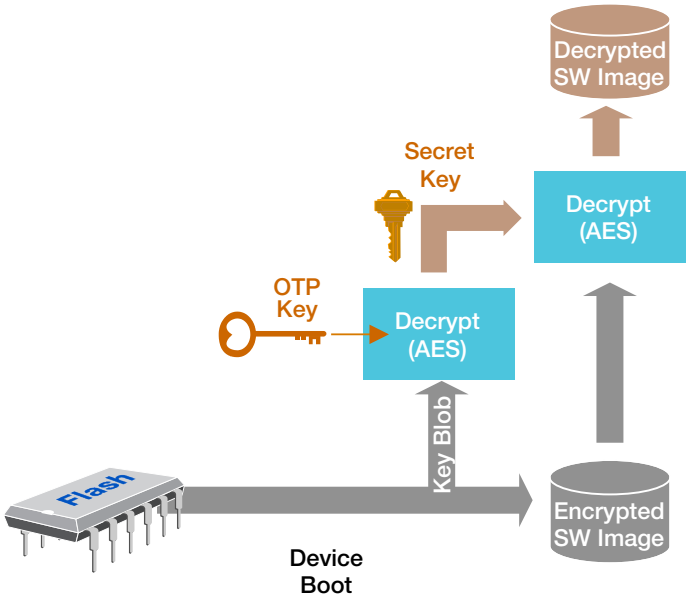
software called the Secure Monitor, which has the attributes of a root of trust. To switch worlds, application code must make a call to the Secure Monitor to request the switch. Memory is partitioned into secure and non-secure regions, and the non-secure world is unable to access memory that is tagged as secure. This architecture could be considered an extension of the user/supervisor model that is implemented on a number of existing microprocessor architectures and has similar attributes to a hypervisor.

The previous security examples assume the use of internal flash memory and that there is no external visibility of the flash memory bus activity. There are, however, many security applications that rely on external memory—so some method of robust security must be provided for applications that expose the transfer of data and instructions on an external memory bus.

One way to achieve this is to encrypt all content that is stored in external memory, and implement a cryptographic engine on the microprocessor, between the external bus interface and the CPU. Code and data fetched from external memory must now be decrypted before being processed by the CPU. Likewise, data stored back to external memory must first be encrypted to prevent successful snooping attacks on the external bus.

The main challenges in this architecture are how to prevent reduction in CPU performance and how to support random access fetches from external memory. The attributes of a block cipher operating in counter mode make it suitable for on-the-fly decryption without impacting performance. Figure 4 shows the general architecture of such a secure system. The attributes of a block cipher operating in counter mode are described in a later section of this white paper.

FIGURE 4. Block Cipher Engine for External Memory Security



Other forms of software-based attacks include snooping and interfering with external I/O communications via Controller Area Network (CAN) or other industry-standard serial ports. These can also include physical attacks on the hardware, such as monitoring power fluctuations and EMI to determine the values of cryptographic keys.

If the financial benefit to the attacker is sufficiently high, then laboratory techniques such as decapping, electron microscopes, microprobing or noise injection using laser light may be employed to attack a device. Protecting against these types of attacks can take the form of physical meshes applied to the dies themselves or adding random noise to the power profile using on-chip logic.

The level of security applied to the device must take account of the value of the asset versus the cost to the attacker. If the attack takes too long or costs too much, then the countermeasure is sufficient.

Crypto Algorithms

Crypto(graphic) algorithms are essentially mathematical computations designed to perform encryption and decryption of a message. The two main categories of algorithms are symmetric and asymmetric.

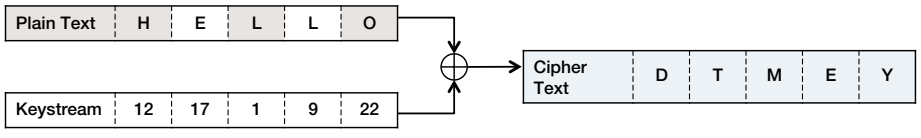
Symmetric cryptography algorithms use the same key for encryption and decryption. The key must therefore be kept secret by all entities that use the key for secure communications. This may pose a problem with distributing the key to new entities, unless a method such as Diffie-Hellman is used, as described in the Secret Key Exchange section below.

The main advantage of symmetric algorithms is their efficiency and speed of execution. For example, Freescale's AES-128 hardware accelerator on the MPC5646C device can execute at the rate of 70 MB/s. In addition to enabling encryption and decryption, symmetric keys are also used for digital signing and authentication.

Encryption is often categorized as being implemented as a stream cipher or a block cipher. The main operational differences between ciphers called "stream" and those called "block" is how the plaintext is encrypted.

A **stream cipher** operates by combining the digits of the plain text message with a stream of random and indeterministic cipher values, known as the keystream. Figure 5 shows the basic operation, using an exclusive-OR (XOR) operation to perform encryption of the message "HELLO" into the encrypted "DTMEY" using the randomly generated sequence of numbers shown as the keystream. The keystream's initial value is determined by a starting seed value.

FIGURE 5. Stream Cipher Encoding



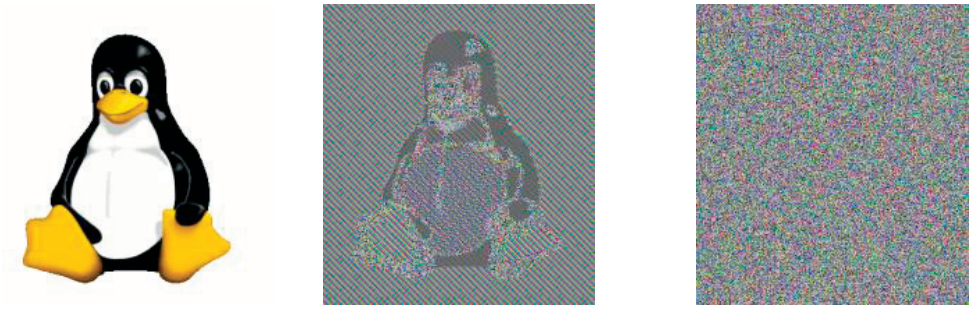
Stream ciphers are often faster than block ciphers and operate on small pieces of data typically using bit or byte-wise XOR operations. Examples of stream ciphers include RC4, SEAL and the Vernam, or one-time pad (OTP), cipher. While stream ciphers can execute at very high speed with simple hardware, they can be prone to security problems. The starting seed is also used to decrypt the message. To avoid security problems, the starting seed must never be used more than once.

One advantage of stream ciphers is that there is no dependency in encryption between any of the characters in the stream. This means they are relatively immune to noise in the transmission channel because any corruption of part of the transmitted cipher text does not prevent decryption of the part that was unaffected by noise. For this reason, stream ciphers such as RC4 are used in WEP and WPA Wi-Fi® security algorithms, although WEP is now deprecated due to its security flaws.

A **block cipher** operates on larger chunks of data than stream ciphers, and each block usually incorporates values from previous blocks. This sometimes means more hardware memory is required than for stream ciphers.

Block ciphers have many operating modes, offering different trade-offs. The simplest form is Electronic Code Book (ECB) mode, which is the exception because each block of data is encrypted and decrypted independently of other blocks. This means that patterns in the plain text appear as similar patterns in the cipher text, which makes the encrypted message somewhat insecure. Figure 6 shows the effect of ECB encryption on an image.

FIGURE 6. Visual Representation of Differences between ECB and CBC

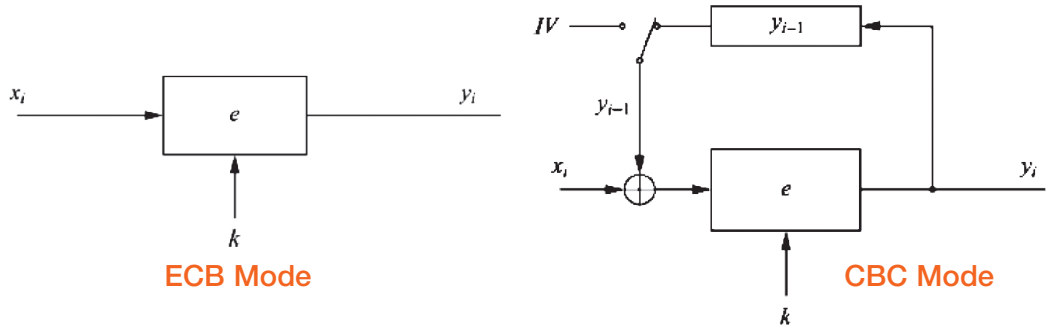


Original ECB CBC

To overcome this insecurity, a number of industry standards have introduced alternative block cipher modes of operation. The Cipher Block Chaining (CBC) mode uses the concept of applying an input variable to each block of plaintext to be encrypted. The input variable to the first block is a pseudo random initialization vector (IV). The input variable to subsequent blocks is the output of the preceding block, hence the chaining in the mode name. The result is that there is no correlation between input and output as shown graphically in Figure 6 and thus provides superior security to ECB.

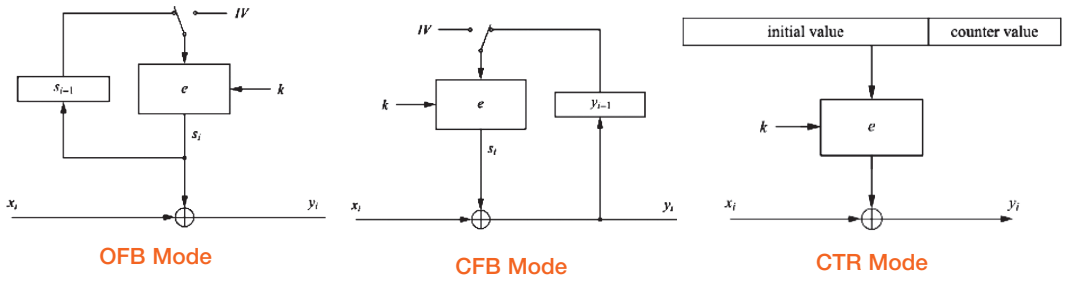
Figure 7 shows the general operation of a block cipher operating in ECB and CBC mode. Both encryption and decryption involved applying the crypto algorithm to the entire plaintext. One disadvantage of this method is latency of the cipher text output.

FIGURE 7. ECB and CBC Mode Encryption



There are alternative block cipher modes that eliminate this latency by operating in a streaming mode. That is, the plaintext is simply XOR'ed with a keystream, much in the same way as the stream cipher described previously. The streaming modes of operation are Output Feed Back (OFB), Cipher Feed Back (CFB) and Counter (CTR) modes. Their implementation differences lie in how their keystream is generated. Figure 8 shows the general data flow for encryption with these three modes.

FIGURE 8. OFB, CFB and CTR Mode Encryption



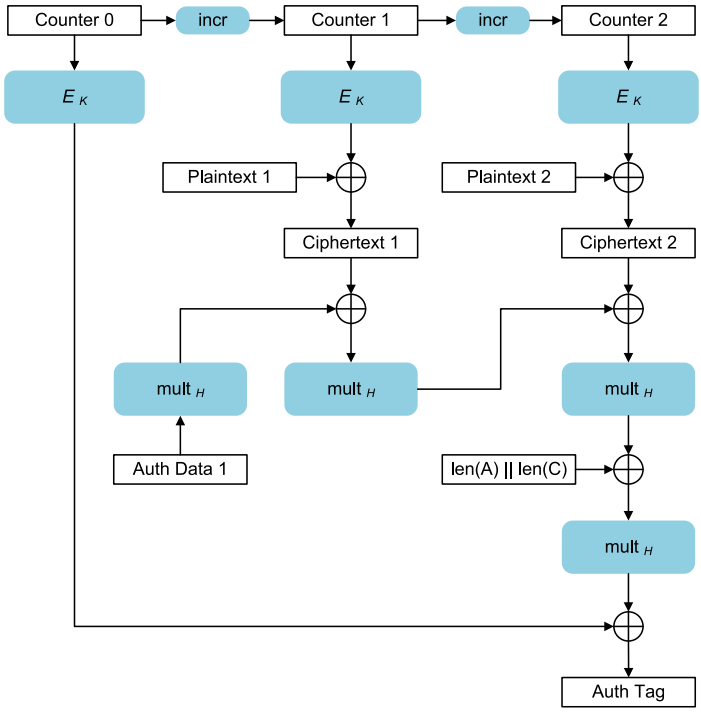
The Counter mode (CTR) is particularly interesting because its keystream is independent of the input plaintext (xi) and output ciphertext (yi), much like ECB mode. This means the keystream can be calculated while the input data (xi) is being fetched, and encryption can be parallelized. Unlike ECB, the CTR mode has an initialization vector (IV) and counter value that prevents patterns in the plaintext from being replicated in the ciphertext, which is the main vulnerability of ECB. Of course, to ensure the security, all data that is encrypted with the same key must allocate a different counter value for each block.

All the aforementioned block cipher modes can be used for encryption and decryption or can additionally be used to perform message authentication, creating a Message Authentication Code (or MAC, detailed in the next section). A security vulnerability exists, however, if the same block cipher and key are used to both encrypt a message and generate its MAC.

Several solutions exist for this problem: using different starting keys derived from one key, different block ciphers modes for encryption and authentication or some hybrid mode using a keyed hash function or asymmetric digital signature (described in the next sections).

Another alternative is to use a variant of the block cipher counter (CTR) mode known as Galois Counter Mode (GCM) shown in Figure 9. This algorithm can perform authentication in parallel with encryption and thus offers significant performance improvement for applications that need both. Encryption is performed by the CTR mode algorithm described previously, while authentication requires a Galois field multiplier that adds complexity to hardware and software implementations.

FIGURE 9. GCM Mode Encryption and Authentication



The crypto algorithms described thus far are AES symmetric functions. The other category of crypto algorithms is referred to as asymmetric, where the encryption key and decryption key are different. The pair of keys is mathematically related and unique. One key is called the “private” key while the other is called the “public” key.

The power of this type of cryptography is that the private key cannot be determined from the public key. Therefore, the public key generally can be distributed and stored in non-secure storage, while only the private key must be kept secret. This potentially reduces the opportunity for attacks on data protected by asymmetric cryptography. One common use of a public key is to encrypt data so that it can be sent securely to the owner of the corresponding private key. The public key cannot be used to decrypt the data, thus allowing anyone with the same public key to securely transmit data to the same entity.

Some examples of asymmetric crypto algorithms include RSA and ECC (Elliptic Curve Cryptography—not to be confused with Error Correcting Codes).

While asymmetric algorithms potentially provide more enhanced security than symmetric algorithms, they do so at a cost. To achieve the same level of security, asymmetric algorithms

(such as RSA) require larger keys than symmetric algorithms (such as AES). Moreover, the computational requirements for asymmetric keys are greater.

For this reason, many security systems use a hybrid approach, where a symmetric secret key is distributed using asymmetric cryptography. The symmetric key is then employed as a "session key" for further secure communications using the faster symmetric algorithms provided by AES.

Authentication

Cryptographic authentication is the process of verifying the identity of the sender of the data. In an embedded application, authentication can be used to verify the source of data transmitted on external interfaces between different control units. It can also be used to verify the trustworthiness of a software image prior to it being executed at run time, or during download from an external tool prior to the image being programmed into on-chip flash.

Authentication usually also provides data integrity if the entire data or software image is used to create the authentication code (more commonly known as a digital signature). Any modification of the data or software image will result in the authentication process failing.

Both symmetric and asymmetric cryptography support authentication algorithms. **Symmetric cryptographic** authentication uses a shared secret key. The sender creates a message authentication code (MAC) using the secret key as a seed. There are a number of forms of symmetric MAC functions, usually based on block cipher algorithms described in an earlier section. Some examples are CBC-MAC, CMAC, and PMAC. The Freescale Cryptographic Security Engine (CSE) and Hardware Security Module (HSM) modules provide a CMAC function which fixes vulnerabilities that exist in the CBC-MAC function that predates it.

Asymmetric cryptography uses a public key to authenticate data received from the owner of the private key. This digital signature scheme relies on the sender generating a unique signature that is formed from some combination of the message and the sender's private key, transmitting both the message and the signature to the recipient. The signature is often formed by generating a "checksum" of the entire message and then by encrypting only the checksum with the private key. The receiver of the message and signature can then perform the same checksum operation on the message and compare it with the checksum that was decrypted from the signature using the public key. If they do not match, then the data was either corrupted during transmission or the sender is not the expected entity.

An alternative method of authentication is based on hashing algorithms and has the generic name HMAC, which is normally prepended to the name of the underlying crypto algorithm. For example, the HMAC-SHA-256 is a hash-based message authentication code constructed from the SHA-256 algorithm. The code is referred to as a keyed-hash message authentication code and relies on the use of a shared secret key. The key is incorporated in the message in a manner that makes it resistant to attack. The resultant HMAC is transmitted with the message, and checked by the recipient using the same shared secret key. This means HMAC authentication has symmetric cryptography attributes.

An interesting difference between symmetric and asymmetric key authentication is that the former does not provide non-repudiation. This could be a problem in a network of nodes that all share the same secret key. It means that the receiver of data authenticated by a shared secret key cannot be sure which entity in the network sent the data. Obviously, there are other measures that can identify the sender of the message—such as adding a sender ID to

the message. This could easily be subverted, however, by a simple man-in-the-middle attack, which changes the ID. Encrypting the sender ID is a solution, but adds complexity which might make the asymmetric private/public key approach more desirable. Another approach is to restrict certain entities to MAC verification and disallow them to generate MACs.

Secrecy

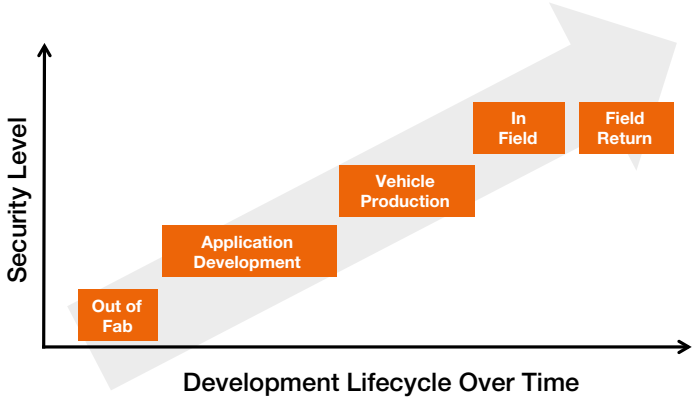
There are two main forms of secrecy that apply to embedded microcontrollers. One is the secret storage of information directly on the microcontroller. This could be passwords, keys, manufacturing information and product identity codes. The second is the secret storage of information remotely from the microcontroller, such as secret keys and passwords required for reprogramming the microcontroller with an external tool.

These secret assets are often used in conjunction with crypto algorithms and authentication procedures, but can also be used to directly unlock access to the microcontroller by an external tool.

Secret Storage: In the context of an embedded microcontroller, there is a dilemma between the engineer's desire to access all memory and functions on the microcontroller and the demands of the end customer to deny access to embedded confidential information or certain operating modes of the application. Fortunately, both can be accommodated if the notion of a security lifecycle is considered, where the level of security of the microcontroller can be adapted to satisfy the differing needs during the development, manufacture, and use of the application.

Examples of products that incorporate a security lifecycle are Freescale's Qorivva MPC5746M and MPC5777M MCUs. Figure 10 shows the increased levels of security that are applied during product development through the application of the security lifecycle state variable.

FIGURE 10. Security Level Increases as Product is Developed



The security lifecycle provides a method to irrevocably increase the security level of the MCU as it progresses from the chip manufacturer, through the Tier 1 manufacturing process, then on to the OEM production line, and finally the end customer who drives the vehicle. In addition, the security lifecycle is extended to support failure analysis of any suspect device, without compromising OEM and Tier 1 software investment or any customer's confidential material.

There are a number of objectives for the security lifecycle:

- It prevents compromising silicon device production test times when no security measures are required on unprogrammed devices.
- It allows ease of use during the earliest stages of product development when security measures can be tested and adapted.
- It provides the maximum level of security when the device is in the field to prevent theft or unauthorized tampering and modification of software installed in the device.
- It allows devices to be returned from the field for factory inspection without allowing access to confidential or proprietary code and data.

The security lifecycle affects the interaction between the debug interface, application code and the hardware security module—as well as flash program, erase and read interfaces. Debug access can be allowed or denied based on a combination of Security Lifecycle State and various passwords. Moreover, flash memory regions can be hardware configured to prevent reading during debug or during failure analysis. Flash erasure and programming can be controlled by user-defined hardware configurations that can be temporarily overridden by passwords.

Secret Key Exchange: Key exchange can be used to create a one-time shared secret between two entities such as the microcontroller and a programming device or between a powertrain controller and a braking controller. One implementation of such an exchange is the Diffie-Hellman key exchange method. It allows two entities that have no prior knowledge of each other to jointly establish a shared secret key over an insecure communications channel such as a CAN network.

Another advantage of this method is that the shared secret key is never transmitted across the communication channel. Even if it is discovered by a successful attack on either entity, the entities can renegotiate a new secret key using the key exchange protocol at any time.

While this secret key could be used to perform secure communications, usually the key encrypts a symmetric key that is then distributed to each entity. The symmetric key is then used for subsequent secure communications. Rather than just use the one-time shared secret for secure communications, the symmetric key approach is preferred as it performs encryption and decryption considerably faster. The one-time secret created by the Diffie-Hellman algorithm is essentially an asymmetric key, which makes it unsuitable for use in encrypting and decrypting large quantities of data in a timely manner.

The Diffie-Hellman protocol relies on each entity being able to generate its own secret integer and a public integer derived from the secret integer. The algorithm used to calculate each public integer is based on a shared prime number (p) and number base (g), as shown in Figure 11.

Each entity (referred to as Alice and Bob) then transmits its public integer (A, B) to the other entity. Each entity then applies the Diffie-Hellman algorithm using its own secret integer and the other's public integer, resulting in a secret key. The secret key is the same for both entities. The public integers transmitted between legitimate entities have no value to an attacker that snoops the communication channel. The asymmetric algorithms used to create the public integers make it unfeasible for the attacker to calculate the corresponding secret integers retained by each legitimate entity.

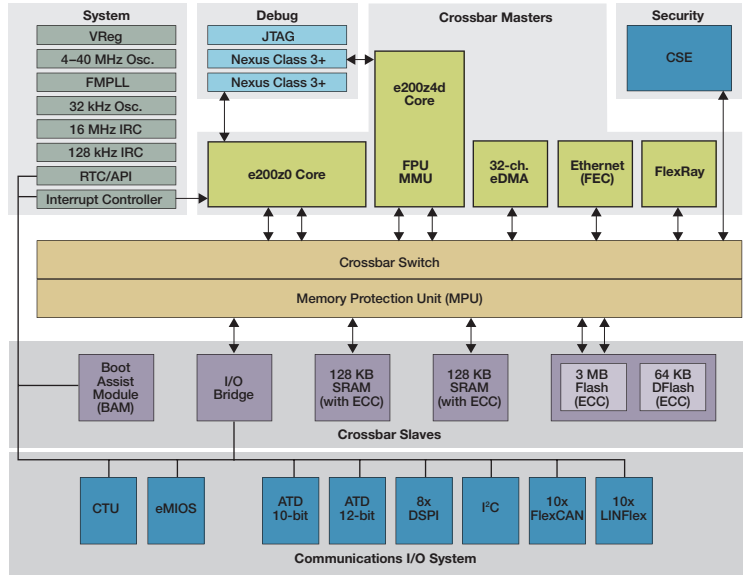
FIGURE 11. Diffie-Hellman Key Exchange

Alice				Bob		
Secret	Public	Calculates	Sends	Calculates	Public	Secret
a	*p*, *g*		$p,g \rightarrow$			*b*
a	*p*, *g*, *A*	$g^a \bmod p = A$	$A \rightarrow$		*p*, *g*	*b*
a	*p*, *g*, *A*		$\leftarrow B$	$g^b \bmod p = B$	*p*, *g*, *A*, *B*	*b*
a, *s*	*p*, *g*, *A*, *B*	$B^a \bmod p = s$		$A^b \bmod p = s$	*p*, *g*, *A*, *B*	*b*, *s*

Current Security Implementations

Figure 12 shows the Freescale MPC564xB/C family of microcontrollers that include a module designed to implement the SHE specification. SHE is implemented as a dedicated but securely firewalled microcontroller architecture called the CSE[1] that coexists on the same silicon as the main microcontroller CPU, memory and peripherals used for traditional application use. Secure cryptographic keys are stored in embedded flash memory that is restricted to the CSE module alone. An AES-128 hardware accelerator is included to provide support for fast ECB and CBC encryption and decryption and CMAC generation and verification.

FIGURE 12. MPC564xB/C Block Diagram



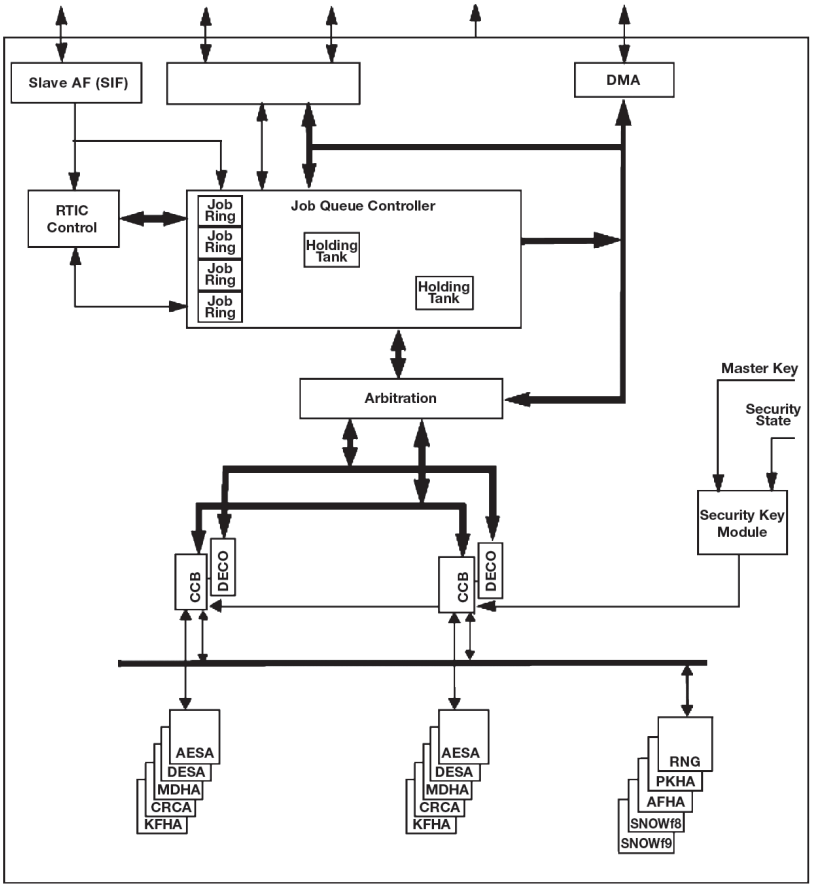
The family of QorIQ processing platform trust architecture microprocessors (such as the P2041) contains the SEC 4.2 security engine that implements block encryption algorithms, stream cipher algorithms, hashing algorithms, public key algorithms, hardware random number generator and run time integrity checking. A block diagram of the security engine is shown in Figure 13.

Vybrid VF3xx devices incorporate a variety of data integrity and security hardware features for safeguarding memory, communication and system data. An optional tamper detection

[1] Freescale's CSE should not be confused with the Communications Security Establishment (CSE) for the Canadian government.

system includes integrated sensors for voltage, frequency and temperature as well as external sensing for physical attack detection.

FIGURE 13. P2041 SEC 4.2 Security Engine



Future Security Implementations

The very physical properties that theoretically might be used to crack the most robust security keys might also provide the answers to creating unbreakable keys. The technique harnesses the laws of quantum mechanics to create keys that can be proven invulnerable to eavesdropping. These unbreakable keys can then be distributed with the knowledge that they are 100% secure.

This type of cryptography is known as Quantum Key Distribution (QKD) and relies on the quantum properties of light. This means the key must be sent through a medium that supports light—such as a laser, optical fiber or even free space.

The security of a distributed key relies on the Heisenberg uncertainty principle. Reading the quantum states of the key causes state values to change and errors produced in the key sequence, which are subsequently detected by the legitimate receiver of the key. If the legitimate receiver detects a high error rate when reading the key, it means the key has been intercepted during its transmission and can be discarded. QKD is not a complete solution and must be used in tandem with classical security methods, so there is an overhead associated with this technique that may be difficult to justify.

Conclusion

In the future, it is conceivable to imagine that robust vehicle security could potentially reduce insurance rates and even lower vehicle depreciation. Integrated security design also offers the added benefit of allowing higher dollar value feature options to be delivered as an option in all vehicles, enabled by a software switch as an after-market purchase.

The continued growth in automotive security will be driven by a combination of factors. The added connectivity to the external world through entertainment portals such as Bluetooth and USB introduces an attack surface for installing malicious software or unauthorized modifications to existing software.

Another factor is the increase in safety-aware applications such as HEV and advanced driver assistance systems in the form of collision avoidance and self-parking. Moreover, emerging markets introduce a new world order to product development and manufacture. The embedded security measures described in this paper provide a way to ensure that safety is not compromised while also providing the features necessary to ease product development—as well as protecting manufacturers' investment and drivers' privacy.

For more information, please visit freescale.com/security

Document Number: AUTOSECURITYWP REV 1

January 2014