09AE-0013

Secure Software Flashing

André Weimerskirch
escrypt Inc.
376 Harbor Way
Ann Arbor, MI 48103, USA

ABSTRACT

An increasing number of vehicular electronic control units (ECU) are equipped with reprogrammable flash memory. The software program in the flash memory determines the behavior of the ECU. The program code usually can be updated via a bootloader, e.g., for a firmware update, a bug fix, or an update enabling additional functionality. The download might be performed over a diagnostic channel or in the future increasingly through wireless channels, e.g. Bluetooth or GSM connection. Once such communication channels are opened to the outside world for downloading software, the authenticity of the software must be ensured. An example of a malicious software download is the replacement of firmware by an unauthorized party, e.g., as is done on a large scale through chip tuning in vehicles.

In order to control software updates, digital signatures play a central role. Here, a digital signature is generated in a secure back end, e.g. by the automobile manufacturer, and then attached to the new firmware as a cryptographic checksum. The ECU is now able to verify the authenticity of the new firmware by checking the authenticity of the signature. Only if the verification is successful is the new firmware actually run by the device. A proper signature verification algorithm is RSA with a short exponent that can be executed in a few milliseconds on an ARM-class CPU if implemented carefully. Provision of a digital signature algorithm itself is often not the main problem, but its integration into the ECU and adapting the organizational processes are. Certainly a secure software download is only useful if there are neither hidden access points to the firmware (e.g., an enabled debug interface) nor a flawed implementation that allows illegal access. Hence a very careful design and implementation phase has to be performed in addition to the secure software download to make sure that only the defined access path is given.

INTRODUCTION

Today's vehicles have several dozen electronic control units (ECUs) that control almost everything, such as air conditioning, electric windows, the engine, and the brake system. Several of these ECUs allow downloading of updated program and data code via a boot loader. Such software might be a control unit firmware update for fixing bugs, for improving features, or for downloading data such as additional multimedia files. The first case is also called a software download or simply flashing (since flash memory is updated). The download might be performed over a diagnostic channel or another available communication channel such as Bluetooth and GSM. Once such vehicle communication channels are opened to the outside world for downloading software, their integrity must be ensured. An example of a malicious software download is the replacement of firmware by an unauthorized party, e.g., as done for chip tuning. The main security objective is as follows:

1. Only original software must be accepted by the vehicle: No manipulated or malicious software may be downloaded to the ECU. In particular, software must not be successfully downloaded to the ECU that alters the defined behavior of the vehicle.
2. Only authenticated parties may alter data, e.g., parameters, stored in the ECU.

Furthermore, the following is desired for an actual security design:

- The compromise of a single control unit does not affect the entire system, i.e., a successful attack does not scale.
- The required computational performance on the control unit side shall be minimal.

DIGITAL SIGNATURES

The secure software flashing scheme we present is based on digital signatures. A digital signature provides the security objective of integrity and authenticity; data

being digitally signed cannot be altered by a malicious third party without being detected by the receiver. Furthermore, the receiver can verify that the data was indeed signed by the claimed signer.

A digital signature is computed as shown in Figure 1. There is a key pair consisting of a secret key *SK* and a public key *PK*. Only the signer has access to *SK* whereas *PK* can be publicly distributed. In our setting, *SK* is only known to the automotive manufacturer whereas *PK* is built into every ECU. Message *x* is first hashed to a short fixed length value *y*. Typically *y* is computed by applying a hash function of the SHA family, resulting in an output of 20 to 32 bytes. Finally, a digital signature is computed over *y* using the secret key *SK*. The signature can then be verified by using the public key *PK*.

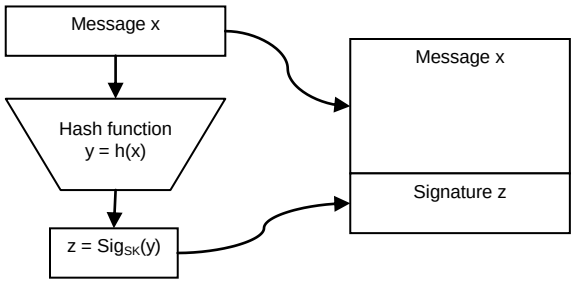

Figure 1: Digital Signatures

CERTIFICATES

As described above, a public key is used in each ECU and firmware programs are signed using the corresponding secret key. It is possible to use a separate public/private key pair for each model year, each ECU type, etc. Furthermore, it is possible to use certificates. A certificate compares to a passport; it certifies a public key and, if required, an identity. In our case, the certificate is simply a public key signed by the OEM's secret key together with the public key and some additional information: $Sig_{SK}(PK_A) \mid PK_A \mid data$. The flash program is then signed by PK_A. When the flashing process begins, the certificate first needs to be downloaded and verified using *PK*, and then the flash program is downloaded and verified using PK_A.

SECURE SOFTWARE FLASHING

A solution for this problem in general is quite simple. Based on digital signatures, the issuer of the software signs the program code and the control unit in the vehicle verifies it. Hence, the issuer holds a secret key for signing the program code and the control units hold the corresponding public key for verifying it. This is shown in Figure 2 in more detail. First, the software is developed. Once it is finished (Step 1), the program object code is passed to a trust center (Step 2) that signs the object code using its secret key. The signature is then passed back and attached to the program object code. The package of code and signature is now stored in a database (Step 4) that might hold versions for different control units. Finally, the appropriate program code is downloaded to a control unit (Step 5) and verified by means of the public key stored in the ECU (Step 6).

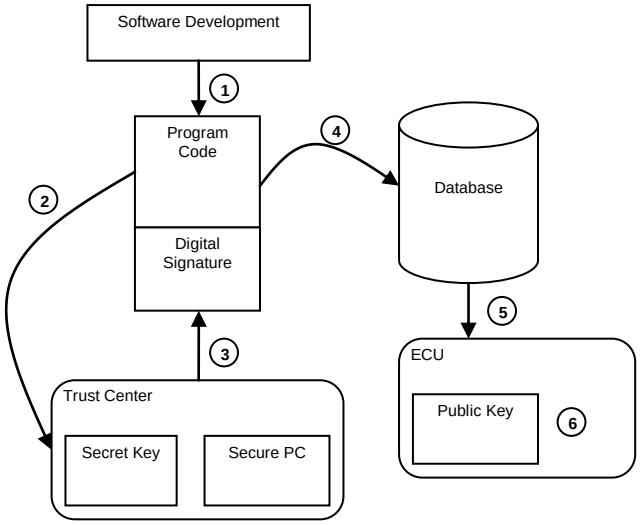

Figure 2: Secure Software Download

One can see that security objective (1) is clearly fulfilled: Only a legitimate authority can issue an appropriate signature for program code that will be accepted by the vehicle, i.e., only authentic software will be accepted by the vehicle. Most of today's cars already provide a mechanism for downloading software. Hence, only a mechanism for verifying the signature is additionally needed in the vehicle. RSA is an appropriate fit for signature verification, since it allows very fast signature verification and can be implemented in software.

Some performance values of our implementation are displayed in Table 1. Note that for the signature verification, first the program code needs to be hashed and then an RSA exponentiation is performed.

	Flash (ROM)	RAM	Run-time / Throughput
SHA-1	898 Bytes	352 Bytes	3,026 KB/s
RSA 1024 Decryption (short exponent)	1,410 Bytes	1,136 Bytes	2ms (exponent 3) / 13 ms (exponent 65,537)
AES-128 (CBC)	2,343 Bytes	236 Bytes	661 KB/s
ECC 160 (ECDSA Signature Generation)	4,244 Bytes	932 Bytes	35 ms
ECC 160 (ECDSA Signature Verification)	4,244 Bytes	1,108 Bytes	66 ms

Table 1: RSA signature verification on ARM MPCore @ 400 MHz

On the server side the key management and the organizational security must be thoroughly organized. Organizational security includes the organization of who has access to sign program code and how the process of signing is performed and recorded. However, no full-sized Public Key Infrastructure (PKI) is necessary. Basically, it is sufficient to issue a single private/public key pair such that the private key is stored in the trust center and the public key in the control unit. The trust center might simply be a PC that is disconnected from any computer network and a secure smart card that holds the secret key. If a finer-grained approach is desired, a key pair for each control unit type, for each production year, or each production location might be applied. No certificates that induce overhead are required, though. The ECU only needs to store a public key such that no secret information is stored here. However, this public key must be protected against manipulation. Otherwise, an adversary could replace this key in the ECU and then activate any manipulated software.

Security objective (2) can be fulfilled by a simple challenge-and-response mechanism as shown in Figure 3. The vehicle and an external party (e.g., a standard PC) share a secret key. The parties then run a challenge-response scheme in order to prove that the external party knows the secret key. After a successful run, the external party can access the vehicle's data to adjust parameters. However, it is crucial that a well-defined interface be specified. For instance, it is reasonable to record all changes in a log file and give access only to non-safety critical data. Using a symmetric-key management is reasonable - each ECU knows an individual symmetric secret key shared with the third party. This third party might be the car manufacturer storing all keys in a protected database. The security can be further increased by using security tokens that hold the secret key for accessing the ECU. Such an approach even allows several authorization levels to be implemented. For instance, there might be a basic level for workshops to adjust typical parameters, and another level for testing that allows all parameters to be changed.

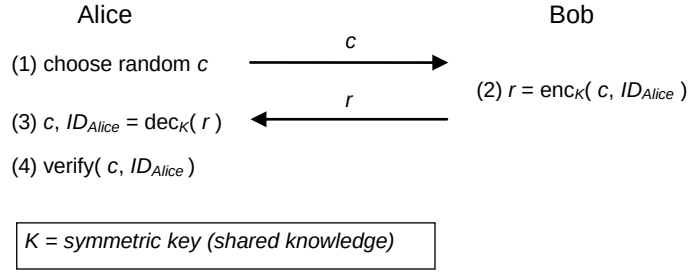

Figure 3: Challenge-and-Response Identification

Protecting the key built into the ECU is crucial. If an adversary is able to read out a symmetric key or replace a public key he might be able to manipulate program or data code. Thus, virtual software protection can be achieved only by applying hardware-assisted approaches employing a security anchor as described in [4].

Nevertheless, there are also mechanisms that try to complicate utilization or at least try to help identify the origin in case software could be successfully read out by an attacker. In order to make decompiling and re-engineering of program binaries more difficult, programs known as “obfuscators'' convert source code, object code, or both, into obfuscated code, making the result excessively complicated and thus far less readable and almost impossible to understand by a human being. However, obfuscation [3] only increases the difficulty for reverse engineering, limits portability and is regarded as “security through obscurity''. Digital watermarking [1] and fingerprinting are techniques which embed visible or invisible information into a digital content (software or data) that cannot be removed or modified or can only be removed or modified with difficulty. Original owners then can use tools to extract the embedded information to detect, e.g., the origin of an illegitimate copy or tampering. However, there are already technologies for both mechanisms to abolish respective restrictions. Thus, such mechanisms cannot replace proper hardware-based software protection.

SECURE FLASH PROCESS

The flash program file has several parts. Each block is optionally encrypted, and the signature of the file was computed beforehand. In the first step, the external device that triggers the flashing process optionally authenticates the bootloader (e.g., by a challenge-response mechanism as described above). In the next step, a certificate might be passed to the bootloader which verifies it using its stored public key. Then, the external device passes block by block to the ECU’s bootloader. The bootloader first decrypts the block and then stores it in the flash memory. At the same time, the bootloader incrementally computes the hash over each block. Once all blocks are passed, the bootloader finalizes the computation of the hash value over the new flash program file and performs a digital signature verification using the determined hash value. If the signature verification is successful, the downloaded file is accepted and activated. Otherwise, a safety procedure is activated and the bootloader awaits the download of a proper flash file. This is shown in Figure 4.

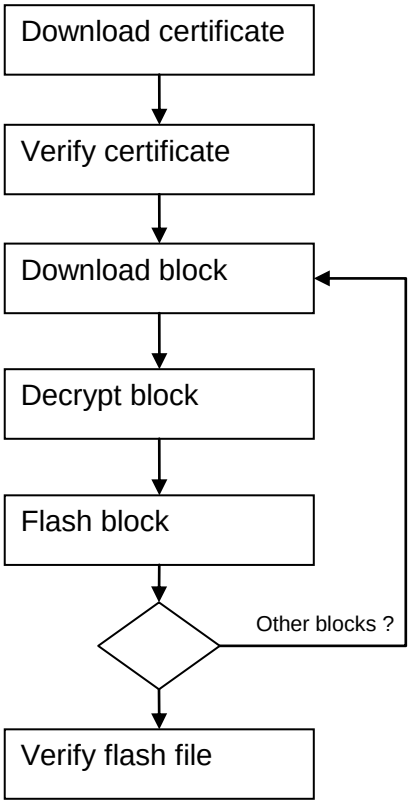

Figure 4: Flash Process

CONCLUSIONS

More and more vehicles' ECUs are equipped with flash memory. Usually, a bootloader is built into the firmware to update the program. However, in most cases there are no mechanisms implemented to avoid downloading a manipulated program that alters the device's behavior in a manner not authorized by the OEM. In this article we presented mechanisms to protect the software update process. Such a mechanism was already implemented in a variety of applications, ranging from the automotive domain to gambling and the mobile phone industry. The German OEMs even standardized the mechanism for a secure bootloader [2]. While this standard allows the use of symmetric cryptography only via a so-called message authentication code (MAC), we strongly suggest implementing the asymmetric cryptographic approach based on digital signatures; otherwise either our original security requirements cannot be fulfilled or the organizational effort is immense.

While the cryptographic mechanisms are generally straightforward to implement, security needs to be provided at all levels, ranging from organization security to appropriate mechanisms for signing the firmware up to inclusion of suppliers. In particular, the security mechanisms need to be carefully incorporated into the existing development processes.

REFERENCES

1. Christian S. Collberg and Clark D. Thomborson, "Watermarking, Tamper-Proofing, and Obfuscation Tools for Software Protection", IEEE Transactions on Software Engineering, vol. 28, nr. 8, 2002.
2. Hersteller Initiative Software (HIS), "HIS Security Module Specification, Version 1.1", available at http://www.automotive-his.de/download/HIS\%20Security\%20Module\%20Specification\%20V1.1.pdf, July 2006.
3. Cullen Linn and Saumya Debray, "Obfuscation of Executable Code to Improve Resistance to Static Disassembly", ACM Conference on Computer and Communications Security (CCS), 2003.
4. Marko Wolf, André Weimerskirch, and Thomas Wollinger, "State-of-the-Art: Embedding Security in Vehicles", EURASIP Journal on Embedded Systems, Special Issue on Embedded Systems for Intelligent Vehicles, 2007.

CONTACT

André Weimerskirch
escrypt Inc.
376 Harbor Way
Ann Arbor, MI 48103
USA
aweimerskirch@escrypt.com