Discretix Technologies Ltd.
43 Ha'Melacha Street
Beit Etgarim
Poleg Industrial Zone
Netanya, 42504
Israel
Tel: +972-9-8858810
Email: marketing@discretix.com

Security Implications of Hardware vs. Software Cryptographic Modules

Hagai Bar-El
hagai.bar-el@discretix.com

Abstract

Cryptographic modules can be implemented either by hardware or by software. Whereas software implementations are known for being easier to develop and to maintain, when it comes to cryptographic modules or security-related applications in general, software implementations are significantly less secure than their hardware equivalents. The reason for this is mostly the fact that software solutions make use of shared memory space, are running on top of an operating system and are more fluid in terms of ease of modification.

Introduction

This white paper discusses the weaknesses that are inherent to software-based cryptographic modules in relation to cryptographic modules that are hardware-based. Other advantages and disadvantages of hardware vs. software approaches are beyond the scope of this white paper. Each one of the following sections focuses on one security aspect in which hardware and software implementations of cryptographic modules differ.

Memory Access Prevention

Software solutions of any sort cannot facilitate their own physical memory. Therefore, software implementations are making use of externally available memory, usually through services of an underlying operating system. When the memory that is used by the application is provided externally, there is no guarantee as to what other processes can access the same memory space. Although most operating systems give some sort of random access memory space protection, it must be remembered that this protection is guaranteed only to the extent of the robustness of the operating system and its being free of flaws. Secondly, memory protection is even more difficult and more lacking where secondary memory is concerned.

Cryptographic modules are particularly sensitive to having their random access memory space well protected. Most cryptographic algorithms and probably all protocols require intermediate results to be stored, in some sort of a "scratch-pad", during the execution of the module. If the contents of this temporary storage are ever leaked the entire system can be easily compromised. This is because this memory is storing values that may be very closely related to the secret keys (or even the keys themselves). Therefore, the security level of a software-based cryptographic module is upper-bounded by the security level of the mechanism that protects the secrecy and integrity of the memory space it uses. This latter security level often cannot even be assessed properly. Generally, if the memory space that a cryptographic module uses is not in its own complete control, its overall security level cannot be guaranteed.

Secondary memory often requires the same level of secrecy protection as primary memory. It is usually used to store long-term keys and similar data (see chapter **Long-Term Key Storage**). However, preserving the secrecy of secondary memory contents on an application-shared platform is far from trivial. Real protection for secondary storage can often be obtained only through the use of encryption. Yet, when encryption is used it gives rise to the problem of storing the encryption key securely.

Hardware-based solutions can contain their own internally managed memory space, which solves the problem of memory-space protection. Furthermore, hardware solutions can be applied, for memory illegal access prevention, by hardware methods, which are inherently more secure than operating system services that are software-based in their nature.

Integrity Assurance

Software, as the name implies, is based on a set of instructions that are stored in memory and are executed upon demand or prior instruction. Since the protection of secondary memory is not guaranteed (see previous chapter **Memory Access Prevention**), the integrity of the code itself cannot be guaranteed either. An adversary can modify the application code either to cause it to malfunction or to cause it to leak critical information. Software code alteration can be done either manually, by changing specific instructions, or in an automated manner using hostile code such as a virus or a Trojan horse running on the same platform or on a platform, which has adequate access privileges.

Hardware-based solutions are safer in this respect for the code being burnt onto a chip. Physically burning the source code is probably the only proof way of causing it to be completely read-only, as source code of any cryptographic module should be.

There are various ways in which software code can be tampered with either by hostile content or by manual intervention. One interesting specific case, which is relevant to ARM processors, is by the use of the GP-IO[1] that is unique to ARM processors. At a particular point of time, during power-up, the bus can be used to reflect memory contents and assist in modification of code that is stored on the host.

[1] General-Purpose IO

Hardware-based solutions have the privilege of not being modifiable at any point, including during the power-up stages.

Reverse Engineering

Software implementations are more easily readable by adversaries and are therefore more susceptible to reverse engineering. Source code that cannot be viewed cannot be reverse-engineered either. Since software implementations are merely instructions that are stored in memory (see previous chapter **Integrity Assurance**), and since the protection of this memory cannot usually be guaranteed. An adversary who may attempt to reverse engineer the code can read these instructions.

Reverse engineering of cryptographic modules that implement publicly known algorithms and protocols is less risky than reverse engineering of other software modules that may implement classified proprietary algorithms[2]. However, reverse engineering of algorithms implementation can still cause significant damage to the security level due to its enabling the discovery of implementation flaws. Flaws that are discovered by an adversary who reverse-engineered the implementation can be taken advantage of by exploitation (manual or programmed).

Resistance to Power Analysis Attacks

Software based solutions are more vulnerable to attacks that are based on power consumption analysis. Every single software command is mapped by the compiler to a set of assembly language instructions that are well known and have a defined pattern in terms of power consumption. These known patterns are easy to identify using relatively simple power analysis techniques. By obtaining information about the internal state of the module the attacker can build a process to extract the secret key that is being used by the module.

Hardware-based solutions can apply special measures that mask the fluctuation in power consumption, to prevent the attacker from collecting power consumption information that can assist in the compromise of the secret key.

Long-Term Key Storage

Key storage problems can be considered as a part of memory access problems, which were discussed earlier. However, storage of long-term keys requires the use of secondary memory and opens the opportunity for additional attacks.

Long-term keys should be stored while protected from compromise to their secrecy or integrity. Moreover, since these keys are long-term (as opposed to session keys), they should be stored in non-volatile memory. As this type of memory can usually be read by external devices. Key encryption is used to protect the long-term keys secrecy and integrity. The difficulty arises when the key that is used to encrypt the long-term key needs itself to be stored securely.

Two options are common for the storage of the key-encryption-key. The first option is to derive it from a user-supplied passphrase and not store it anywhere. The key is reconstructed every time from the passphrase the user enters. If the user enters a correct passphrase then the key will be reconstructed properly and will be used for decryption of the long-term key. There are several drawbacks to this technique, two of which are hereby presented: The first drawback is that this technique cannot be applied for systems that need to work unattended, e.g. without the presence of a user to type in the passphrase. The second drawback arises from the low entropy of user-supplied passwords, if these cannot be forced to be long and unpredictable. In the cellular environment, a passphrase is usually a four or five digit PIN. The entropy of the generated key is therefore less than that of 14 or 17 bits, respectively, forming a long-term key protection scheme that is ridiculously weak.

A second approach is to encrypt the long-term key with an internal key, which is stored somewhere in the application. By doing so, the designer is basing the security of the system on the ability to hide the internal key properly. When using this approach, software solutions are usually weak for their disability to provide real "hiding places" where keys can be hidden. Since software is installed on accessible memory spaces[3], and since reverse engineering of code is often difficult but more often feasible, keys that are hidden inside software code can usually be retrieved after investing some level of effort.

When hardware implementation is concerned, the problem of key hiding has more effective solutions. Internal keys can be burnt as a part of the hardware implementation making them extremely difficult to extract. The internal key can also be stored in non-volatile memory, which is made inaccessible to other applications by hardware means.

Dependence on OS Security

When an application is running on top of another lower-layer application (such as an operating system) the higher-layer application's security is by many means dependent on the security level of the lower level application in terms of flaws. It follows that, if a flaw is discovered in an operating system implementation, this flaw is likely to lead to additional vulnerability of the application running on top of it. In general every security problem of the operating system, either known or yet unknown, may cause security problems with the cryptographic module implementation. Good examples for this phenomenon are operating systems that leak memory contents through swap files and flaws in memory management and protection schemes of operating systems. Open operating systems or operating systems that are providing high0-level services are even more problematic in this sense. The higher the level of services provided by the operating system, the higher the potential is for these kinds of flaws.

Hardware implementations are not dependent on high-level operating system services and are therefore not dependent on secure implementations of these services.

[2] This statement is brought based on security considerations only, completely disregarding loss of IP or other business damage caused by the revealing of implementation code.

[3] Generally, secrecy of source code cannot be assumed. Basing the security of a system on the secrecy of its implementation is called *"Security Through Obscurity"* and is widely considered as an unsafe practice.

Limitations Resulting From the Use of DSP

Software implementations often make use of DSP circuits that are available to allow for faster multiplications. DSP can provide faster multiplication of long integers than regular software code and is therefore commonly used by software implementations. The main limitation of DSP circuits is concerning the security they can offer. DSP is an open implementation receiving inputs and giving outputs through the publicly accessible bus. Using this mechanism for private key operations is highly risky. Since multiplications are only some of the required operations in public key cryptography, temporary values are left on the bus between operations. The values can easily be revealed exposing the private key value. Moreover, such values can be modified by an adversary prior to entering the DSP to allow for signature forging.

The only way to avoid the inherent problems with DSP is by avoiding the use of DSP entirely, as done by closed hardware solutions.