Application Scenarios for WhiteboxCRYPTO

Executive Summary

There are three families of problems that can be addressed by whitebox cryptography:

1. Secure storage of cryptographic keys
2. Protection of keys while in use
3. Ensuring the integrity of cryptographic keys

Items 1 and 2 regard sight-sensitivity at rest and in use, respectively; the third regards tamper resistance. Below, a scenario from each family of problem is expanded. Following this, an example system is presented that exhibits integration challenges for security. A scheme based on WhiteboxCRYPTO™ is devised to resolve the security issues.

Sight-Sensitivity at Rest: Implementing Key Management Functionality

System designs leveraging cryptography for Information Assurance (IA) purposes must specify cryptoperiods for the keys in use. After the expiration of the cryptoperiod, the subject key must be replaced/refreshed. Because the keys must change, they cannot be integrally incorporated into the system software/firmware, but instead must be held as mutable data. This yields a problem of securing the keys as data-at-rest.

Whitebox cryptography offers a solution to the data-at-rest problem for cryptographic keys by representing the keys in a manner that cannot be interpreted without the corresponding whitebox algorithm instance. Thus the whitebox version of the key can be stored in public view without further protection (assuming the viewing public has no means to access the corresponding whitebox algorithm instance so as to perform reverse-engineering).

This scenario is particularly relevant when the whitebox algorithm is implemented within field programmable gate array (FPGA) or other semi-secure processing node, yet the only non-volatile storage resides outside the security boundary (that is, accessible via JTAG or insecure channels).

Sight-Sensitivity in Use: Defense against Memory Capture

Consider a scenario in which data-at-rest is secured using traditional cryptography. That is, assume that an application manually encrypts all data written to non-volatile storage. In order to operate, this software must keep its cryptographic keys resident in memory. If the system is vulnerable to some manner of memory-capture, then the entire encrypted content of the disk becomes vulnerable.

In the pay TV market, there have been numerous attacks on the conditional access modules (CAMs – essentially smart-cards with custom ASIC security modules) where clock-glitching and fault injection attacks have been used to trigger diagnostic/failure-analysis modes that essentially dump the memory of the CAM. The well-publicized RAM remanence attacks provide another means of performing memory-capture, not to mention the recent OpenSSL “heartbleed” vulnerability which is again a memory-capture vulnerability. Finally, many commercial operating systems provide APIs for allowing one process to read the memory of another process (the debugger attachment API being only one such interface); if misused, these APIs form a potential attack vector for memory resident keys.

Using whitebox cryptography, it is not sufficient to capture the key from memory; an attacker must also capture the matched whitebox cipher instance. As most operating systems separate code and runtime data, it is unlikely for network-based vulnerabilities to leak portions of system code. In any case, an attacker must either reverse engineer the whitebox instance to recover the equivalent classical key, or they must successfully extract the entire algorithm instance to perform a “code lifting” attack to recover the content of the disk.

Application Scenarios for WhiteboxCRYPTO

Tamper Resistance: Securing Digital Signature Authentication against a Tamper-Capable Adversary

Many systems support dynamic upgrades for software and firmware. Consider such a system that requires system patches/system images to be digitally signed by a trusted authority. While the intent of such authentication is to ensure that untrusted software/firmware is not allowed into the system, the strength of the authentication is directly limited by one's ability to ensure the integrity of the public credentials of the trusted authority.

For example, an attacker may generate his own key pair, producing a certificate from his chosen public key. If he can replace the trusted authority's certificate with his own (that is, as an effect of tampering), then he can fool the system into accepting an untrusted image as follows: Since the attacker knows the private key for the replacement certificate, he may sign his own software/firmware image. When the system update mechanism checks the signature against the (replaced) trusted certificate, it will match and the system will accept the update as authentic.

Whitebox cryptography can mitigate this risk by representing the public credentials of the trusted authority in a "whitebox form". Unless an attacker is in possession of the key preparation tools for the specific whitebox cipher instance, he will not be able to represent his chosen key in the form required by the system-update mechanism (short of completely reverse engineering the whitebox cipher instances).

Systems Example: Configuration Enforcement

Defense systems typically feature multiple modes of operation – Operational Mode, Training Mode, Software/Firmware Upgrade Mode, etc. – selectable at boot time. The development below will build configuration management schemes to ensure that these three modes operate with approved software loads.

Software/Firmware Upgrade Mode

To produce a boot time selection of mode, we presume ownership of the BIOS code. After the normal BIOS routine to bring the system up into upgrade-mode, we initialize a WhiteboxAES™ decrypt only instance. This AES instance will be used to decrypt the upgrade image, thereby providing a barrier to prevent unauthorized software loads. Here the utility of whitebox cryptography is to preserve the secrecy of the decrypt key given that the BIOS code remains in plaintext. Since AES is a symmetric cipher, anyone recovering the decrypt key can "forge" a system upgrade image. (Due to performance reasons, public-key ciphers are not used for bulk decryption. Typically a public-key cipher will be used to decrypt a symmetric cipher key for bulk decryption. Note that in such an application of public key over symmetric cipher, observing the symmetric key is sufficient to forge system images, as the symmetric key wrapped under the public key cipher is reused.)

To provide additional defense against reverse-engineering, the optional hardware ID (HWID) feature of WhiteboxAES can be used to tie correct decryption to the presence of a particular byte string in memory. The intent is that such a byte string be characteristic of a device or system at the unit , model , or family level, so as to prevent correct decryption if the code is lifted and hosted on different hardware; in practice a HWID can be any string of bytes that can be assembled at runtime. HWIDs are algebraically incorporated into WhiteboxAES keys, meaning there are no ways to bypass the HWID checks – the HWID is integral to the key.

If the system non-volatile memory (NVM) controller can read the BIOS area of memory, then the NVM controller can validate whether NVM writes to the system upgrade area are made by the authentic upgrade BIOS code using digital signatures. That is, a digital signature of the BIOS can be resident alongside the BIOS itself. The NVM controller (perhaps an FPGA) could then verify the digest before accepting any update block.

Application Scenarios for WhiteboxCRYPTO

Operational Mode

Our security objectives for the operational mode revolve around ensuring that the operational image remains intact, and that the operational image is one authorized for the subject unit/model/family. For simplicity, we assume a bare-metal software application, directly entered from the BIOS.

The design centers around embedding another WhiteboxAES instance into the Operational mode portion of the BIOS. The portion of the BIOS that transfers control to the operational image is then augmented to expect a WhiteboxAES key prepended to the operational image. The BIOS then decrypts the operational image into RAM using the BIOS resident instance, and application specific key. If AES GCM is used, then a post-decryption authentication tag is produced, which stands as witness that the image remains unmodified.

Because the decryption instance is BIOS-resident, and distinct instances of WhiteboxCRYPTO are incompatible, the installed BIOS selects which system images may be loaded. Such can provide a mechanism to prevent an US only software load from being inadvertently loaded onto an FMS/DCS destined box.

Training Mode

Training mode mirrors operational mode, except that it is likely desirable to specify a distinct WhiteboxAES instance for controlling operational vs training mode, simply to facilitate different control policies between operational and training system images.

Additional Layers

Assume we wish to define a security-heartbeat, such that a reverse-engineering/tamper-vulnerable host must respond in a way that proves knowledge of a particular key, but does so without revealing the key. In this example, we assume the party demanding the security heartbeat is adequately secured, and has no knowledge of WhiteboxCRYPTO.

Once the system is up in an approved configuration, then Diffie Hellman, the Key Derivation Functions (KDFs), and the Dynamic Key Preparation features can be leveraged to establish secure communication channels with other parties, without revealing the session keys.

Given such a channel, a security heartbeat can be established, as follows: The secure party encrypts a random message, and sends it to the insecure device. The insecure device performs a decrypt from classical to obfuscated of the heartbeat message. Next, the insecure device uses WhiteboxSHA™, configured in obfuscated-in/obfuscated-out mode, to digest the message. Finally, the insecure device performs an encrypt-from-obfuscated-to-classical operation to produce the reply. In this way, a heartbeat that depends on knowing a crypto key can be implemented without revealing the particular key. Further, the secure party can operate without knowledge of WhiteboxCRYPTO.

Now consider an augmentation of this approach wherein the read-only code segments were digested to produce a HWID string for the whitebox cipher. Then the heartbeat message can convey information about the status of the insecure device's software in a manner that doesn't depend on explicit integrity checks.

Summary

The preceding sections have addressed archetypal security issues and offered a number of design approaches based in whitebox cryptographic implementations.

Cryptography is not a silver bullet—but it is a critical tool in the system-security-engineering toolbox. This whitepaper has identified the three major families of design problems where whitebox cryptography has a clear use-case: protection of data at rest, protection of keys during usage, and protection of keys against replacement. The scenarios given are representative of real-world security challenges, and the WhiteboxCRYPTO-based solutions are feasible for implementation in practice.

Microsemi

Microsemi Corporation (Nasdaq: MSCC) offers a comprehensive portfolio of semiconductor and system solutions for communications, defense & security, aerospace and industrial markets. Products include high-performance and radiation-hardened analog mixed-signal integrated circuits, FPGAs, SoCs and ASICs; power management products; timing and synchronization devices and precise time solutions, setting the world's standard for time; voice processing devices; RF solutions; discrete components; security technologies and scalable anti-tamper products; Ethernet solutions; Power-over-Ethernet ICs and midspans; as well as custom design capabilities and services. Microsemi is headquartered in Aliso Viejo, Calif., and has approximately 3,600 employees globally. Learn more at www.microsemi.com.

Microsemi Corporate Headquarters
One Enterprise, Aliso Viejo, CA 92656 USA
Within the USA: +1 (800) 713-4113
Outside the USA: +1 (949) 380-6100
Sales: +1 (949) 380-6136
Fax: +1 (949) 215-4996
email: sales.support@microsemi.com
www.microsemi.com