Microsemi®
Power Matters.™

Mobile Payment Application (MPA) Solutions with WhiteboxCRYPTO

Whitepaper

September 2015

Mobile Payment Application (MPA) Solutions with WhiteboxCRYPTO

Introduction

Mobile payment applications must remain highly secure despite residing on uncontrolled platforms. Regulatory policy and industry best practices require a mix of information security and application security practices, including the use of anti-tamper/anti-reverse engineering technology, and sound crypto-key-refresh practices. WhiteboxCRYPTO™ helps meet these requirements by providing a method to deploy reverse engineering resistant cryptography in a manner that supports secure key-refresh, integration of tamper-resistance features, and synthetic diversity (deployment or redeployment with functionally identical ciphers that appear different to a reverse-engineer).

WhiteboxCRYPTO has been deployed domestically and internationally on Windows, Linux, MacOS, Android, iOS, and Solaris.

Technical Hurdles in the Mobile Payment Industry

Mobile payment applications (MPAs) are subject to multiple sources of requirements. The PCI standards are generally accepted; however, each card provider adds his own requirements and protocols. Mobile payment application providers are left to find their own methods of satisfying the security and protocol requirements for multiple vendors and standards bodies.

Commonly encountered requirements for MPAs are:

- Any public/private key pairs used by MPA must be generated on-device.
- All keys transferred to the MPA must be protected by cryptography at least as strong as the keys being transferred during transit.
- The MPA must not retain any portion of a user PIN longer than is required to generate authentication cryptograms.
- Anti-tamper and anti-reverse engineering technologies must be deployed to slow the process of compromising keys.
- MPAs must be redeployed bi-monthly with alternate anti-tamper and anti-reverse engineering techniques to limit the effectiveness of reverse-engineering efforts.
- Cryptographic keys must be discarded after a specified service lifetime (cryptoperiod).

These tasks are not technically challenging until one considers that the device running the MPA is completely under the control of a potentially malicious user. If the associated cryptographic keys are ever present in memory, they can simply be captured, and a malicious user can decrypt all other keys and sensitive messages. The preceding bullets become much more complicated if one adds an additional requirement:

- Cryptographic keys should never appear in the clear in the MPA.

With this addition, one requires a method of generating key pairs, receiving keys, and using keys without having them be exposed. One must perform PIN operations as a combination of dynamic user input and cryptographic keys, but without revealing the keys. Further, these approaches must be amenable to the application of anti-tamper techniques, and must ideally participate in the diversified redeployment requirement. Because keys must be discarded, one cannot simply deploy a set of heavily obfuscated cryptographic routines with the keys pre-embedded; one must deploy a key management solution that operates without exposing the keys.

WhiteboxCRYPTO is first and foremost a cryptographic key protection technology. In the following sections, we describe how it can be deployed to resolve some of these tricky design points.

Secure Key Management with WhiteboxCRYPTO

WhiteboxCRYPTO is designed around a set of features that make it possible to perform secure key management on an insecure platform. There are three critical features that enable this:

- WhiteboxCRYPTO ciphers operate using their keys in a protected form.
- WhiteboxCRYPTO ciphers can produce/consume data in an obfuscated form.
- WhiteboxCRYPTO ciphers can prepare protected form keys from obfuscated-form data.

For example, WhiteboxCRYPTO can generate a protected-form RSA private key, preparing a public key in the usual form. When it receives a standard RSA-wrapped Advanced Encryption Standard (AES) key, it can unwrap this key to an obfuscated form, and prepare a WhiteboxAES key from the obfuscated data. In this way, a server/HSM that knows nothing about WhiteboxCRYPTO can securely (blind-) transfer an AES key to an MPA, which can in turn establish a secure channel using the key—all without exposing any of the private/secret keys in memory on the MPA.

Anti-Tamper, Anti-Reverse Engineering, and Redeployment

WhiteboxCRYPTO ciphers are designed to be reverse engineering resistant. The protected form of the key and the cipher implementation constitute a matched pair. These are produced through a randomized generation process that yields ANSI-C compliant source code. The ability to re-generate randomized "instances" of the ciphers immediately lends support for the requirement that MPAs be redeployed periodically with different protection mechanisms.

Because cipher instances are ANSI-C source code, any anti-tamper tools/techniques that process source-code or object-code can be applied. All protected form key material takes the form of flat arrays of bytes[1]; this offers an ideal target for tamper reactions, as perturbing the key material will cause the ciphers to produce incorrect results, which can be a subtle mechanism for introducing a tamper response. Crypto failures can be difficult for a reverse engineer to diagnose, particularly when used in conjunction with other anti-debugging or anti-tampering mechanisms.

Note: [1] All ciphers represent keys using a flat byte-array form for cross-platform interoperability. Some ciphers will prepare this array into a host-platform-specific form for faster operation at runtime.

Deploying WhiteboxCRYPTO

In this section, we describe the design process and logistics of deploying a WhiteboxCRYPTO-based solution.

Designing for Data Obfuscation

Data obfuscation is a key feature for WhiteboxCRYPTO, but using it effectively takes some consideration. First, in the context of WhiteboxCRYPTO, "data obfuscation" implies a parameterized family of data-representations. This means a design can employ more than one data-obfuscation format, which is useful for restricting data-flows within an application.

It can become tricky to discuss a system that uses multiple data-obfuscation forms; "colors" make a nice analogy for discussion purposes. For example, "you cannot pass the plaintext from RSA-decrypt to AES-encrypt, RSA produces blue data, and that AES instance expects green.” This color-based convention is used throughout the remainder of this whitepaper to indicate different non-interoperable data-obfuscation formats.

Each WhiteboxCRYPTO cipher instance can be configured to expect its inputs to be either classical (unobfuscated) or obfuscated under a specific color. A cipher instance's output can be independently chosen as classical or obfuscated. A common configuration is to leave the ciphertext side of a cipher classical, but produce obfuscated plaintext. This configuration is most useful for performing key-management operations, as they are commonly multi-step. Using obfuscated plaintext allows the key-management steps to proceed without revealing the intermediate values.

Input/output obfuscation is selected at instance generation time, and is therefore static at runtime. Passing data of the wrong color to an input will generate meaningless output. A helpful way to go about an obfuscated-data design is to first enumerate each cipher-use in the system and identify the producer/consumer relations between the cipher-uses. This can be drawn as a graph. The ciphers at the producer end of all edges directed to a common destination must use the same output color; likewise, the consumer end of all edges originating from an obfuscated output must expect that color on the incoming data.

For example, in Figure 1, Elliptic Curve Diffie Hellman (ECDH) is used to negotiate an ephemeral Key-encryption key (KEK). In this example, ECDH is configured to interact with a classical (that is, non-whitebox-aware) server, and produce the shared secret under blue obfuscation. Following this, an X9.63 Key Derivation Function (KDF) instance accepts the raw shared secret under blue obfuscation, and produces the KEK under red obfuscation. The main data path uses AES dynamic key preparation (from red data) to prepare a protected form version of the KEK for the AES-CBC-decrypt instance, which in turn takes a classical AES-wrapped payment key, and decrypts it to obtain a green obfuscated payment key. Data Encryption Standard (DES) dynamic key preparation (from green) is applied to prepare the payment key for use with the DES instance, which is subsequently used to process classical transaction data into a payment cryptogram.

Note: Obfuscated data can be treated as an array of ordinary bytes, and can be logged, persisted/recovered and so on, just like classical data. The same is true of protected form AES and DES keys. Therefore, legacy static KEKs (as illustrated) can be pre-converted to red, before storing.

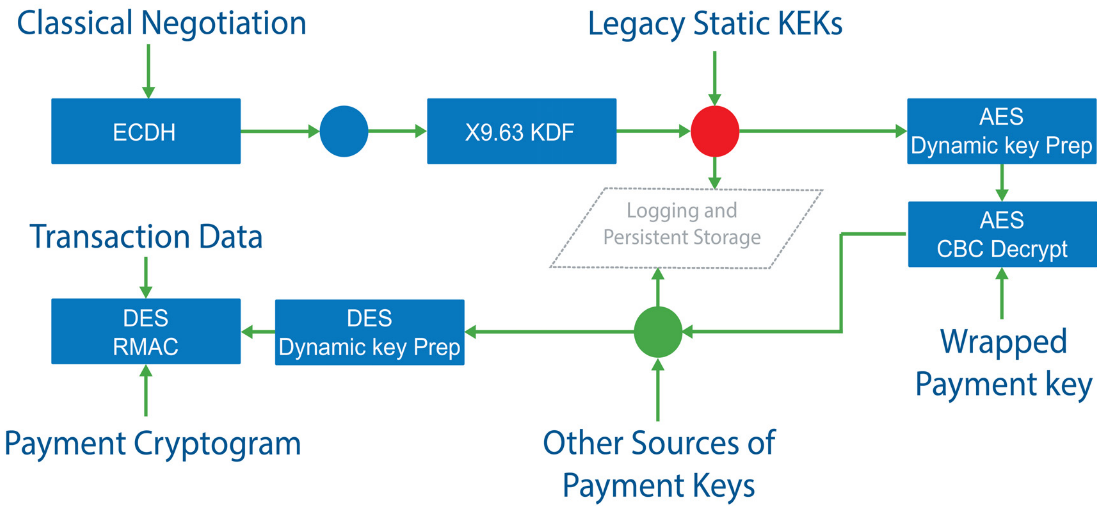

***Figure 1* • Designing for Obfuscation**

In Figure 1, three 'colors' of obfuscation are used, Anything in 'red' obfuscation can be used as a payment-key KEK. Anything in 'green' obfuscation can be used as a payment key. 'Blue' obfuscation is only used to hide the intermediate value between ECDH and the subsequent KDF. Therefore, colors directly constrain data-flow. For example, because KEKs are red, they cannot be prepared in any meaningful way for the DES RMAC instance.

As you can see, data obfuscation can be a powerful tool for controlling data flows within a cryptographic application, while at the same time preventing observation of the intermediate values in the key management operations.

Obtaining Protected Keys

The primary function of WhiteboxCRYPTO is to protect cryptographic keys during usage. This raises an immediate question: how does one get keys into a protected form? There are two methods:

- Static—offline key preparation via a command-line tool
- Dynamic—online key preparation from obfuscated data via an API call

Static offline key preparation is a simple command line process, wherein the target key (or key pair) is specified on the command line, and the tool produces a binary data file (or alternatively, a C source file and header) containing the protected form of the given key.

The dynamic key preparation process, on the other hand, is designed to remain secure even when operating on an untrusted device. It is a runtime API that permits the caller to translate obfuscated data into the protected key form for a given cipher instance.

Because each cipher instance is randomly generated, the key preparation process must be informed about the specific way the target cipher represents its keys. Thus, the static key preparation tool must be given a meta-data file that describes the target key format. This meta-data file is produced at the same time as the cipher instance. Dynamic key preparation uses code generated at the same time as the target cipher instance. Hence, it embeds knowledge of the target key format implicitly.

Note: The dynamic key preparation API expects its input in a particular color; therefore, it must be considered during the overall design process for data obfuscation.

Controlling Cipher Direction

In many cases, it is desirable to restrict the cipher direction for some keys. That is, a given key is only permitted to be used in encryption operations, but not in decryption operations. WhiteboxCRYPTO instances support single direction operations, even for symmetric ciphers. Bi-directional operation is supported by a pair of distinct instances; therefore, the protected key form for the encrypt instance is incompatible with the decrypt instance. (Note that it is possible to prepare the same key into both protected key forms, if desired.)

Single direction cipher control can be considered during the design of any WhiteboxCRYPTO deployment.

Supported Ciphers

For bootstrapping the key management infrastructure:

- RSA key pair generation
- Finite field Diffie-Hellman
- Elliptic curve Diffie-Hellman

Public key operations:

- Elliptic curve El Gamal
- Elliptic curve DSA sign/verify
- RSA encryption/decryption/sign/verify

Symmetric ciphers:

- AES (ECB, CBC, CTR, GCM, CCM, CMAC, and others)
- DES (ECB, CBC, RMAC)

Supporting operations:

- Digests: SHA1 / SHA2
- KDFs: X9.63 KDF, NIST SP800-108 KDF
- Various obfuscated data-manipulation tools.

Need something novel? Microsemi® has prepared white-box versions of custom ciphers for clients as well.

Power Matters.™

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

E-mail: sales.support@microsemi.com

© 2015 Microsemi Corporation. All rights reserved. Microsemi and the Microsemi logo are trademarks of Microsemi Corporation. All other trademarks and service marks are the property of their respective owners.

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 makes no warranty, representation, or guarantee regarding the information contained herein or the suitability of its products and services for any particular purpose, nor does Microsemi assume any liability whatsoever arising out of the application or use of any product or circuit. The products sold hereunder and any other products sold by Microsemi have been subject to limited testing and should not be used in conjunction with mission-critical equipment or applications. Any performance specifications are believed to be reliable but are not verified, and Buyer must conduct and complete all performance and other testing of the products, alone and together with, or installed in, any end-products. Buyer shall not rely on any data and performance specifications or parameters provided by Microsemi. It is the Buyer's responsibility to independently determine suitability of any products and to test and verify the same. The information provided by Microsemi hereunder is provided "as is, where is" and with all faults, and the entire risk associated with such information is entirely with the Buyer. Microsemi does not grant, explicitly or implicitly, to any party any patent rights, licenses, or any other IP rights, whether with regard to such information itself or anything described by such information. Information provided in this document is proprietary to Microsemi, and Microsemi reserves the right to make any changes to the information in this document or to any products and services at any time without notice.