Securing Mobile Payments

White Paper

October 2016

Driving Trust
www.insidesecure.com

Table of contents

Executive Summary

Tokenization was portrayed as the solution to all the security challenges of Mobile Payment. It has now been realized that Tokenization on its own is not enough and the mobile application needs to be protected. This has resulted in "WhiteBox" being the latest buzzword when it comes to Mobile Payment security.

Anyone bringing a Mobile Payment product to market needs to take security seriously. Organizations offering payment services put themselves at risk of fraud and serious reputational damage. Equally, consumers will not use a financial service that they perceive to be insecure.

Providers of Mobile Payment services cannot rely on the security of mobile phones. Users may not mind (or even notice) their phone being used as a spambot or the occasional advertisement being inserted into an application, but they will object to their accounts being used for fraudulent transactions. Also, someone attacking a payment application can easily breakdown the defenses provided by phone operating systems. This means that the payment application needs to be more secure than the phone it is running on.

Securing Mobile Payments requires a different mindset from traditional card payments. The hardware based security models are no longer available to issuing banks; but they no longer need to be. Security models can be put in place that embraces the updateable nature of software.

Tokenization provides over the air update of payment credentials. WhiteBox provides a method for protecting cryptographic operations and data. These techniques are only part of the security model as on their own they are vulnerable to attack. A full solution will also require powerful software protection technologies, including anti-tamper, to defend the application, its WhiteBoxes and the Tokenization process. When combined together, a strong model is achieved for protecting Mobile Payments.

This new model is starting to become the industry standard so each bank does not need to reinvent it. If a bank implements the model correctly, then they can have confidence that their Mobile Payment product will achieve acomparable level of security to their card products with the improved user experience of mobile applications.

INSIDE Secure works closely with the major payment schemes to accelerate Issuer Bank's integrations to the schemes' tokenization services. This has been demonstrated with both MasterCard through MDES engagements and by the company's accreditation as a Visa VTS integration partner. This provides banks with a smooth route to utilize the aggregation ability of these token services to develop their own wallet product.

Couple this expertise with INSIDE Secure's MatrixHCE technology, which provides HCE functionality based on the leading payment brand standards. It is targeted to allow Mobile Banking and Payment Application developers to speed up their development and time to market by combining HCE, Payment and Security as a packaged solution.

Securing Mobile Payment applications requires more than just data encryption. In addition, developers must secure the overall application code with all its vital logic & processes, data, and the cryptographic keys. MatrixHCE utilizes INSIDE Secure's software protection tools to make it extremely difficult and time-consuming for attackers to understand how a payment application works in order to compromise it.

Introduction

A fundamental requirement of any payment product is that it must be secure. Without this the organizations offering the service put themselves at risk of fraud and serious reputational damage. Equally, consumers will not use a financial service that they perceive to be insecure.

When it comes to Mobile Payments, the traditional security models are not available to the banks. This means that the banks, their product teams and their security teams need to understand the paradigm shift of the new security models required to support mobile payments. These models are tailored to a software only solution and take advantage of the flexible nature of software and connected devices.

This paper uses Host Card Emulation (HCE) as an example of mobile payments but the principles discussed apply to any mobile payment product.

Security vs Usability

Mobile Payments have to work in the real world. Any security model put in place needs to work with the usability requirements. If the usability is constrained, the product will not be commercial successfully.

There are two main constraints from a security perspective. The first is that the service is "always on" – i.e. there are no environmental inputs on the consumer's ability to pay. The second constraint is speed – payments are commonly made in environments where throughput is crucial (such as mass transit).

These constraints mean any security model should not block payments being made when there is no network coverage. Requiring a data connection breaks the "always on" rule and would create too much latency to meet the timing targets.

Further Reading

To learn more about mobile application security, INSIDE Secure's white paper discusses The Power and the Risk of Mobile :

https://www.insidesecure.com/Products-Technologies/Mobile-Payment-and-Banking/MatrixSSE/Documents/The-Power-and-Risk-of-Mobile.

More about HCE Payments can be found in INSIDE Secure's HCE Payment Journey whitepaper :

https://www.insidesecure.com/Products-Technologies/Mobile-Payment-and-Banking/Matrix-HCE

Understanding the Threat

The Source of the Threat

Anywhere that money is involved will attract the attention of criminals. Mobile payments are no different. As usage of mobile increases for financials transactions, so will the interest from criminals who wish to profit from weakness in the software.

To understand the scale of threats posed, it is important to understand the motivation of the attackers and their capabilities to mount an attack. Attackers can be loosely grouped as shown in Table 1 below.

Attacker	Motivation	Example of attack	Impact Rating	Damage to Bank
Kid on the street	Opportunistic	Lost or stolen device	Low	Negligible
Academic	Kudos	Single instance of application analyzed and sensitive data extracted	Medium	Adverse publicity, reputation & brand damage Exposure of new attacks Provide motivation to criminals
Criminal	Money	Malware used to extract sensitive data from every instance of a payment application.	High	Potentially hundreds of millions of dollars Serious reputational & brand damage

Table 1 - Attack Impact Levels

The biggest risk to any bank's mobile payment application is from criminal organizations. These organizations are highly resourced, highly skilled and are looking for a large return on their investment. This means that their intent is to perform a mass attack. As such, this paper will focus on how to defeat these criminal organizations; though the techniques discussed also thwart the lower impact attackers.

Assets

When attacking a payment application the attacker is typically looking for one thing: the payment credentials (e.g. credit/debit card numbers). It is these credentials that the attacker can monetize by using them to make fraudulent transactions.

Other assets can include personal data, which may not be directly applicable to the payment use-case but which are harvested to be sold on for later fraudulent use. This risk is common to all mobile banking applications.

The cryptographic keys used in an application are also valuable assets. While they do not have any direct monetary value they can be used to unlock other assets stored within an application. This means that the cryptographic keys are frequently the primary targets of an attacker.

Preparing for an Attack

Before an attacker is able to stage a mass attack, they first need to understand how a mobile payment application operates. This gives the attacker the information they need to perform the attack. To do this an attacker will first analyze a single instance of the application. As the attacker will perform this analysis on a phone (or an emulator) to which they have full access and over which they have full control, they can use sophisticated tools and techniques to take the application apart in order to fully understand how it operates and where any sensitive data is stored.

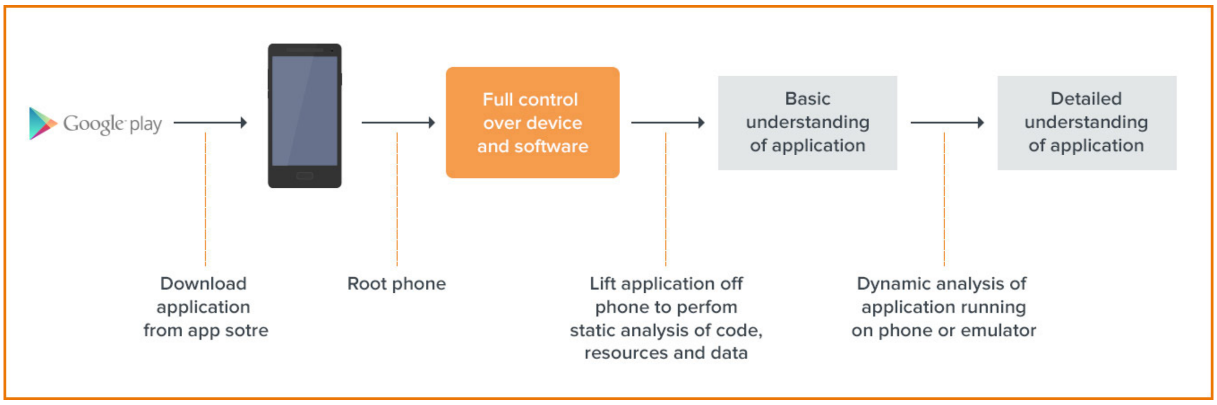

Google play
Full control over device and software
Basic understanding of application
Detailed understanding of application
Download application from app sotre
Root phone
Lift application off phone to perfom static analysis of code, resources and data
Dynamic analysis of application running on phone or emulator

Figure 1 - Stages in Analysis[1]

1 There are also tools available to download apps from the Google Play directly to a computer

Gain Root Access

Modern phone operating systems run applications within a "sandbox". These sandboxes are designed to provide isolation for the application's execution and data – stopping one application from seeing or interfering with another. The trouble with these sandboxes (and other OS provided protections) is that they are easily broken down. The most common way of achieving this is to gain access to an "administrator" mode. In this mode, the user or the application has full access to all the device's applications, resources and data.

This administrator mode is gained through "root access"; hence becoming an administrator is known as "rooting". This is colloquially referred to as "jailbreaking" when applied to iOS devices.

Given that it is relatively easily to achieve (there are many free tools online to do it), the first thing any attacker will do is to root their phone. This removes any blockages the operating system vendor may put in an attacker's way, freeing them and their tools to analyze the application.

It should be noted that there are many early-adopters of Mobile Technology who will deliberately root their own devices in order to give them greater access and freedom to innovate. Blocking rooted devices risks alienating users that legitimately rooted their phones. In many countries this is the majority of the early-adopter user base.

The challenge for any provider of a service is that there is no way to reliably determine if a device has been rooted or whether the rooting was deliberate. Techniques to detect rooting work on a blacklist/whitelist model – once the contents of these lists have been deduced it is easy to circumvent the checks.

Static Analysis of Application

The first stage of analyzing the application is "static analysis". This involves the attacker looking at the application code, resources and data when the application is not running. The aim being for the attacker to retrieve information about the application or steal data (e.g. cryptographic keys) from the application.

If an application has little or no protection applied, simple static analysis may be all an attacker needs to prepare for an attack. For example, the application developer may believe that encrypting the payment credentials before saving them to the phone's persistent storage is enough. Unfortunately, if the encryption key is stored unprotected in application code, it is easily recovered through static analysis. This would allow an attacker to then decrypt the payment credentials - not just for this instance of the application but all instances of the application.

Dynamic Analysis of Application
To further their knowledge, an attacker could observe the application running on a mobile phone or emulator that they control. Simply observing the application running through standard development tools such as debuggers and memory analyzers could also allow the attacker to retrieve secret information. Furthermore, advanced tools for attacks (e.g. Differential Computational Analysis) are extensions of these tools and techniques.

A second stage to this would be to modify the execution of the application. This allows the attacker to better understand how an application behaves. It also enables the attacker to prepare malicious code that could be used within a mass attack.

Performing an Attack

Once an attacker has analyzed an application and found weaknesses, they can then scale-up the attack to multiple instances of an application. An attacker will often sit-and-wait until they are confident that their attack will reap the biggest possible reward.

An attack may be delivered directly against the application installed on users' phones (using a technique such as malware) or it may involve snooping on communication traffic (i.e. a man in the middle attack).

Malware

Malware is malicious software that is installed on a user's phone without the user's knowledge. Malware can be deployed through a variety of channels:

- **Phishing** : tricking the user into installing the software.[2]
- **Trojan applications** : hiding the malicious software within an apparently genuine application.[3]
- **Exploiting code weakness** : using a bug or flaw in some software running on the phone to execute some hidden instructions.[4]

2 *https://blog.lookout.com/blog/2016/08/25/trident-pegasus/*
3 *http://www.scmagazine.com/pokemon-goes-wrong-with-malware-and-armed-muggings/article/508755/*
4 *https://www.hotforsecurity.com/blog/ios-vulnerability-allows-remote-code-execution-triggered-by-image-files-16111.html*

Once on the phone, malware will have to breakdown the operating system's defenses - probably through establishing root access. As was discussed above, this is not difficult and remote rooting, without the legitimate user's knowledge, is not uncommon.

Once the malware has achieved this, it will be able to access the target application. The malware may target the application's code, the active data it is temporarily storing in the phone's memory (RAM) or the "rest" data it saves to persistent storage. The exact nature of this targeting will depend on what the attacker found during their analysis.

Ultimately, all these techniques are trying to achieve the same goal, allowing the attacker to utilize the payment credentials that are hidden within the application. This may involve sending the credentials to the attacker or secretly using them directly from within the payment application to make a transaction.

Man in the Middle

An alternative to targeting the application on the phone is to target the communication between the application and the bank. The most common way of doing this is known as a man-in-the-middle attack. This is where an attacker is in the middle of the communication between the phone and bank.

With these attacks, the phone and the bank mistakenly assume that they are interacting directly with the intended end point, but the attacker is in the middle eavesdropping or changing the information.

Changing the Mindset....

(or: We are not in Kansas anymore)

Kansas

The traditional approach for banks to add security at the customer interaction point is to introduce tamper resistant hardware. This has been seen with Internet banking where dongles were used to generate a login code, or in card payments where magstripe was replaced by chip-cards.

Figure 2 – Dorothy is no longer in Kansas

When bringing contactless payments to mobile, the obvious solution was to replicate the security model from plastic cards: chip-and-PIN. This uses a secure element (e.g. a phone's SIM card) to store the sensitive payment credentials. The secure element is fundamentally the same as the chip in an EMV (the formal name for chip-and-PIN) card, with some additional features due to its use in mobile phones.

EMV cards have been in use since the 1990s and are therefore a known entity. They have been proven in-market as a safe way of storing payment credentials. Therefore, by using a secure element for mobile payments, the confidence in the security of “chip” could be extended to mobile. This meant that mobile payments could be built on a proven and trusted base.

The difficulties with deploying mobile payment solutions that are based on secure elements are well documented. With a plastic card, the card issuer is the owner of the chip but on a mobile phone the owner of the secure element is typically a third party (e.g. a mobile network operator). This third party ownership has been one of the key reasons why mobile payment deployment has been slow as agreements had to be reached between the card issuer and the owner of the secure element. Even if commercial agreements could be reached, the number of parties required to deploy payment credentials to a secure element resulted in a technically complex ecosystem and difficult revenue sharing model.

All of this meant that no bank successfully managed to control the solution and achieve mass-market adoption with a secure element based deployment. Only the OEM players (e.g. Apple) have managed to successfully deploy secure element based solutions - because they own the chip.

If the banks were to play in the mobile payments market, they needed a different solution - one that did not rely on a third party. When, first Blackberry and then Google added HCE support to their operating systems (removing the link between the NFC communication chip and the Secure Element), the card issuers had a new architecture they could utilize. This allowed for a purely software solution so there was no longer a need to reach a commercial agreement with a secure element owner. The downside of this was that the card issuer no longer had access to a secure element; and therefore no longer had access to the proven security model.

Software isn’t the Land of Oz

Without access to a secure element, the banks no longer have access to the hardware security they are familiar and comfortable with. That means that new security models need to be developed for “software only” solutions.

When defining these new models, it is important to understand that software is not hardware. The rules of the game are different. Software is not a fixed entity. It is malleable and updateable; unlike plastic cards that are fixed and have to keep their contents secure for up to five years.

This gives a purely software solution two advantages :

- The exposure to any security breaches can be minimized by reducing the value of the sensitive credentials – they can be frequently updated through the mobile phone's data connection (this can be referred to as "cloud" based data update).
- The card issuers are in control - they can quickly respond to new threats or upgrade the security of the software. This means that the target of the attack (the mobile application) can be constantly evolving; resetting the attacker's knowledge.

Beware the Wicked Witch

It is easy to be led to believe that there are simple techniques to secure the phone a mobile application is running on. The argument goes :

"The phone provides sandboxing so malware cannot touch the payment application. Root detection will ensure the sandboxing is still in place and my application is safe."

Unfortunately, it is not that easy.

The reality is that mobile phones are wonderfully powerful and flexible devices on which to enhance a bank's services, however they are consequently incredibly insecure environments in which to store sensitive data. The security measures put in place by the mainstream mobile operating systems, such as sandboxing, are easily broken down; and even if they were not, the applications and their data can be "lifted" off the mobile device so that powerful computers can be used to attack them to reveal their secrets.

Solutions to identify security breaches (such as malware and rooting detection) simply do not work.

There are two reasons for this; firstly, such detections work on a blacklist model - they search for known compromises - this makes it an arms race and one the hackers are winning (they can simply use techniques the "blacklist" has never seen). Secondly, users react badly to applications that force restrictions on how they can use their phone - some users want to "root" their phone, others do not want to have to install anti-virus software.

The solution to this is to make the payment application protect itself and its data.

Reducing Exposure to Risk

The first part of the new “software only” security model is to reduce the value of any payment credentials that can be stolen due to a security breach. This is achieved by taking advantage of the fact that a mobile phone has a data connection that can be utilized to refresh the payment credentials.

Unlike in the chip/secure element model, the credentials no longer have to last for the lifetime of the card – typically three to five years – but can be regularly updated through the data connection of a mobile phone. This practice is commonly referred to as “tokenization”.

Tokenization

Tokenization is the act of replacing a sensitive credential with a pseudo credential that has restricted use cases. This pseudo credential can be mapped back to the original by a management service when the credential is used - provided the use case is allowed.

In mobile payments, tokenization is used to replace the account number (PAN) with a digital PAN that is in the same format as the real PAN. The acquiring network treats this digital PAN as if it was a real PAN. Either the Issuer or the payment scheme can perform the mapping between the real PAN and the digital PAN.

The allowed uses of the digital PAN are restricted; so a digital PAN issued for mobile contactless payments would only have mobile contactless transactions authorized, all other transactions would be declined. This reduces the value of the digital PAN, as it can only be used for a limited set of transactions.

If the digital PAN can only be used for physical in-store payments, this greatly reduces the scalability of a mass attack - the usage of the credential cannot be automated.

Limited / Single Use Keys

Tokenization restricts where the payment credential can be used; but it does not reduce the lifetime of the credential.

One idea was for the digital PAN to regularly change (either after a set number of transactions or time). It was realized that this would cause problems for use cases where the PAN has to be tracked through a system (e.g. in transit networks or using the card to identify a previous purchase) so the digital PAN is now typically static (or “one time”) for the lifetime of the mobile card.

The PAN is not the full payment credential though; there are associated cryptographic keys that are used to secure the transaction data as it passes through the acquiring network. There is no use case that requires different transactions to use the same keys, so these keys can have a very limited lifetime (either a given number of transactions, given cumulative value of transactions or period of time). Beyond this lifetime, any transaction that attempts to use these keys can be declined or at least identified as high risk.

Single use keys are a special case of limited use keys. These keys are restricted to one transaction only; after which they are discarded. Typically, when single use keys are utilized, a batch of them (i.e. 10) is stored on the handset. This allows the user to make more than one transaction without having to fetch a new key each time.

The phone's data connection can be used to replenish keys that are stored on the mobile device. How frequently the keys are updated is a balance between convenience and risk. Ideally a user should always be able to pay, even if they do not have a data connection; but equally providing too large a gap between required updates reduces the security gained through using limited or single use keys.

Software Secure Element

Tokenization is only part of the solution when it comes to securing mobile payments. There is still a fraud risk if tokens are exposed or, worse, an attacker is able to request new tokens. This means it is necessary to harden software to defend it against these attacks. Through this hardening, a mobile application can become a secure "element" on the mobile device for storing payment credentials.

Users may not mind (or even notice) their phone being used as a spambot or the occasional advertisement being inserted into an application1, but they will object to their cards being used for fraudulent transactions. This means that the payment application needs to deploy techniques to make itself and its data more secure than the phone.

Tamper Detection

As discussed earlier, the basis of an attack is to analyze the application. Anti-tampering protects applications against hackers who have unlimited access to deployed versions of the program.

Tamper detection involves injecting software security fragments ("checks") into an application. These checks monitor the application as it is running ensuring that no one is manipulating the application. This stops an attacker from changing the software to discover how it operates or from inserting malicious code into the application.

It's important that the anti-tamper solution deployed :

- Achieves a high check density to maximize the frequency and coverage;
- Provides intelligent analysis and optimization to maximize the number of checks while minimizing the performance impact;
- Automatically injects the checks to remove the possibility of coverage gaps caused by human error;
- Forms a "check network" which greatly increases the difficulty in removing the checks as they all need to be removed together;
- Makes it virtually impossibly to automate the removal of the check network.

If a tamper detection solution achieves all the above requirements, it provides a strong foundation on which to build the application's security.

5 *http://www.kaspersky.com/about/news/virus/2015/Quarter-of-Users-Do-Not-Understand-the-Risks-of-Mobile-Cyberthreats*

Obfuscation

Obfuscation on its own is not application protection; but it can be a useful technique to have as part of wider defenses. Application developers should hide sensitive data in software and obfuscate sensitive code.

Given the "always on" requirement of Mobile Payment applications, significant quantities of sensitive logic and key Intellectual Property (IP) will be installed on mobile devices. At the same time many new features deployed on such devices contain commercially sensitive algorithms, security-sensitive features that seek to fingerprint or authenticate devices and users, or those that decrypt streams of sensitive content are now included in applications.

These sorts of routines within applications can be routinely examined using a wide array of freely available debugging and hacking tools, creating significant risk of hacking and IP theft.

Powerful obfuscation, combined with anti-tamper, dramatically increases the complexity of reverse engineering an application's sensitive functions. This significantly hampers attempts to statically or dynamically analyze their operation, making analysis impractical for all but the most skilled attacker and ensuring that even elite hackers will move on to softer, less frustrating targets.
The obfuscation technology used should allow flexibility to target specific functions and provide the ability for complexity to be tuned to maximize efficacy while meeting performance and code size requirements. Obfuscation techniques that can be employed include control flow obfuscation, symbol name obfuscation, loop flattening, code mutation, decoy code, decoy functions, and literal substitution. Opaque predicates and expressions ensure that the code must be dynamically analyzed if any attempt at understanding is to be made.

WhiteBox Cryptography

For critical secret processing, WhiteBox technology should be employed to dissolve secrets into the code and to obscure algorithms. It does this by dissolving the WhiteBox's static key into the fabric of the WhiteBox algorithm such that they key effectively no longer exists in the software and the operation of the algorithm is virtually impossible to discern.

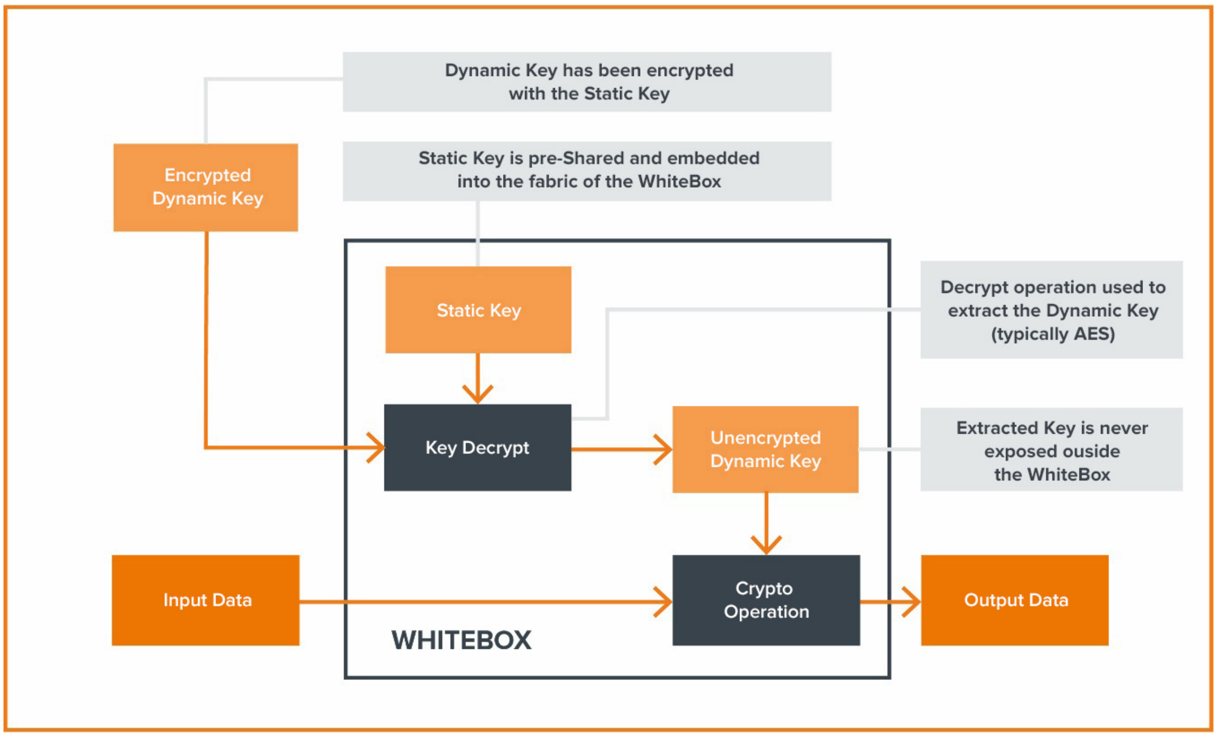

Dynamic Key has been encrypted with the Static Key
Encrypted Dynamic Key
Static Key is pre-Shared and embedded into the fabric of the WhiteBox
Static Key
Decrypt operation used to extract the Dynamic Key (typically AES)
Key Decrypt
Unencrypted Dynamic Key
Extracted Key is never exposed ouside the WhiteBox
Input Data
Crypto Operation
Output Data
WHITEBOX

Figure 3 - WhiteBox Operation

WhiteBox solutions come in two models: those that provide a pre-created WhiteBox and those that provide the tools for developers to create their own WhiteBoxes. The former has advantages that the effort to create the WhiteBox is pushed to the solution provider. The latter gives a greater degree of flexibility and freedom; any algorithm (not just the crypto the solution provider supports) could be WhiteBoxed and algorithms can be chained together to increase the security of an operation.

The other point to remember is that whoever creates the WhiteBox controls the cryptographic keys that unlock the WhiteBox. If the application developer creates the WhiteBox then they are the sole entity to have those keys. If the solution provider creates the WhiteBox, then they have the keys.

As with obfuscation, WhiteBox technology is only one layer in the defenses and to be truly effective it needs to be used with a powerful anti-tamper solution. The highest protection comes when the anti-tamper solution can fully integrate the WhiteBox with the rest of the application. To fully anchor the WhiteBox into the application, it is important to have the WhiteBox's source code so it can be compiled along with the rest of the secure component.

It is possible to use a WhiteBox from Java (or another insecure component), but it is strongly discouraged. Even though the WhiteBox can be designed to ensure secrets are encrypted before they leave the WhiteBox and decrypted as they enter, allowing direct calls to the WhiteBox from an insecure component will place the WhiteBox at significant risk of ‘lifting’, and subsequent illicit control (performing work for Application and Hacker alike). A WhiteBox is only strong if it is ‘anchored’ to a secure component that bears tamper-resistance. This is a very important point and must be considered in design of any mixed-language application intending to process secrets in software.

Programming Language is Important (C versus Java)

Java has become the programming language of choice when developing for many mobile phone platforms (including the principal target of any HCE deployment: Android). The benefit of Java is that it compiles to hardware neutral bytecode – this means it can run on any target processor. Only at runtime (or immediately before) is it converted into commands that the target processor will understand at the point the user runs the application.

Tamper detection is difficult with "runtime compiled" languages, like Java, because the code shipped is in bytecode form. This bytecode is not what is executed directly when the application runs. When an application runs, the language runtime compiles the bytecode into machine code and executes that. Given that there is no way to distinguish this intended modification from malicious tampering, it is impossible to deliver strong runtime anti-tamper protection of Java code.

The best that can be achieved with languages like Java is to verify the integrity of the bytecode, but this will not detect any changes to the compiled code that the runtime is actually executing. A hacker can easily intercept the compiled code and modify that instead, thus evading the bytecode verification.

It is also possible to modify the language runtime to effectively neutralize any bytecode verification methods, although this is less common because modifying the runtime-compiled code is so simple.

The bottom line here is that Java applications are easy for hackers to modify, and if attackers can modify the code, developers cannot expect the application to be secure for very long.

Genuine anti-tamper can only be achieved with compiled languages like C and C++ because the code that executes is identical to the code that was originally compiled. Due to this, it is important to develop payment applications in C/C++.

Protecting Data at Rest

Sensitive information must be protected in use, at rest and in transit. “In use” is covered by the techniques discussed above. “In transit” requires secure communication protocols, the mobile end-point of which is fully secured.

When it comes to “at rest”, it is important to design the storage of an application in such a way that the critical information (e.g. passwords) do not reside directly on a device. If they do, they must be stored securely. They should reside within encrypted storage in the internal app data directory, and the app should be marked to disallow any backups to cloud services.

It is important to consider how the data is encrypted/decrypted. If the cryptographic key is stored insecurely or the algorithms are easily observed running, then it is easy for an attacker to circumvent the encryption. It is recommend that the data encryption/decryption is performed in a WhiteBox. That way it is extremely difficult to observe the cryptographic algorithm in a meaningful way or to find the cryptographic key.

If possible, the data should never leave the WhiteBox unencrypted. For example, the generation of an EMV payment cryptogram could be performed entirely within a single WhiteBox, including the decryption of the stored payment credentials.

Protecting Data in Transit

Communication Encryption

It is clearly important to encrypt data that is being transmitted over open and untrusted channels - such as the Internet. For Payment Applications, a layered approach is recommended, with sensitive data first protected with “application” encryption before being encrypted by transport layer encryption.

The transport layer security would typically use a standard protocol like SSL/TLS.

The application layer encryption would be under the control of the bank but normally utilizing well-known and trusted cryptographic algorithms. It is recommended that the mobile application end point for this encryption is a WhiteBox. This protects the cryptographic keys and so the data.

Device Binding

When deploying new or updated credentials to a mobile phone, it is important to be sure that they are being sent to the correct phone. If an attacker was able to request credentials while pretending to be a genuine user, it would be possible for the attacker to gain access to the sensitive data without having to breach the security of the mobile application.

A strong device binding solution locks an instance of the payment application to a single mobile phone. This stops an attacker from pretending to be the user or from cloning the user's application onto another device.

Certificate Pinning

It is not enough to just encrypt communication between the phone and bank. The phone must have confidence that it is talking to a genuine bank server. With standard use of SSL/TLS, the phone will accept any server certificate signed by a trusted Certificate Authority (CA). This gives an attacker the opportunity to spoof the server to perform attacks such as man in the middle.

Extra validation is needed to defend against these attacks. This validation is achieved through certificate pinning. This means that only a certificate the client knows and trusts can be used authenticate the server.

Conclusion

Providers of Mobile Payment services cannot rely on the security of mobile phones. Users may not mind (or even notice) traditional malware, but they will object to their accounts being used for fraudulent transactions. Also, someone attacking a payment application can easily breakdown the defenses provided by phone operating systems. This means that the payment application needs to provide its own protections.

Securing Mobile Payments requires a different mindset from traditional card payments. The hardware based security models are no longer available to issuing banks; but they no longer need to be. Security models can be put in place that embraces the updateable nature of software. This involves combining over the air update techniques, such as tokenization, with software hardening technologies.

These new models are starting to become the industry standard so each bank does not need to reinvent them. If a bank implements the models correctly, then they can have confidence that their Mobile Payment product will achieve a comparable level of security to their card products.

INSIDE Secure works closely with the major payment schemes to accelerate Issuer Bank's integrations to the schemes' tokenization services. This has been demonstrated with both MasterCard through MDES engagements and by the company's accreditation as a Visa VTS integration partner. This provides banks with a smooth route to utilize the aggregation ability of these token services to develop their own wallet product.

Couple this expertise with INSIDE Secure's MatrixHCE technology, which provides HCE functionality based on the leading payment brand standards. It is targeted to allow Mobile Banking and Payment Application developers to speed up their development and time to market by combining HCE, Payment and Security as a packaged solution.

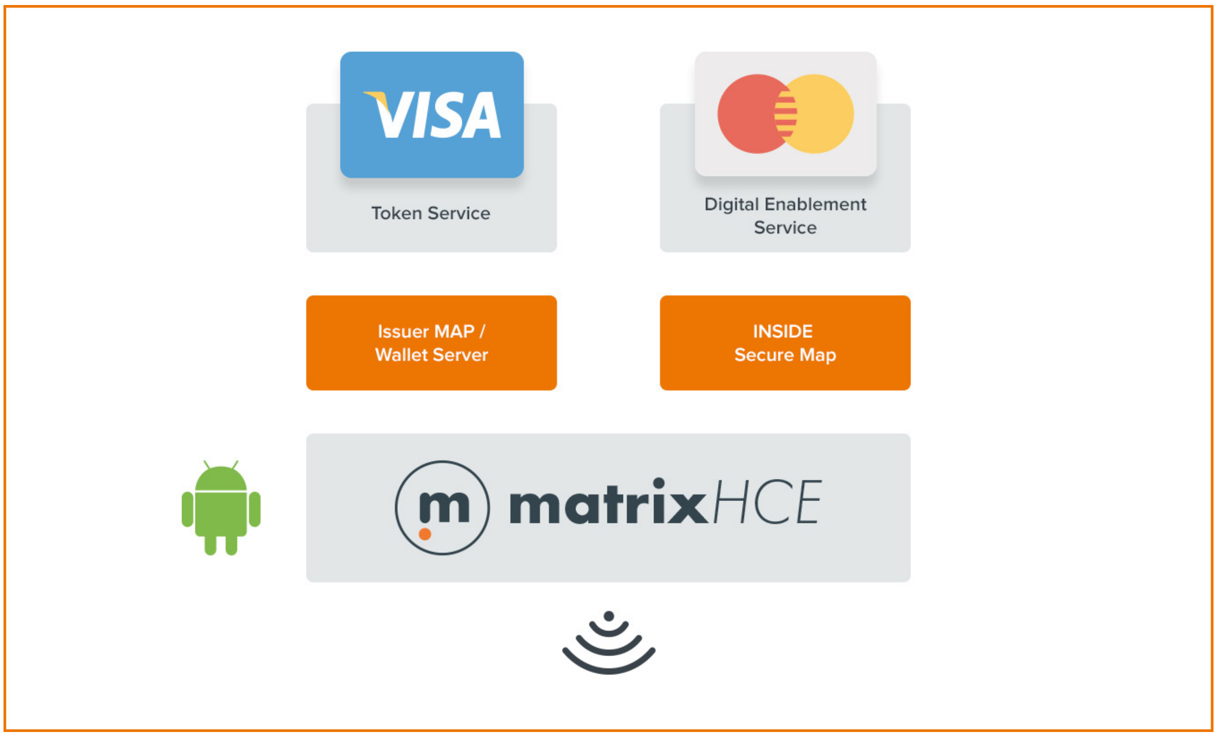
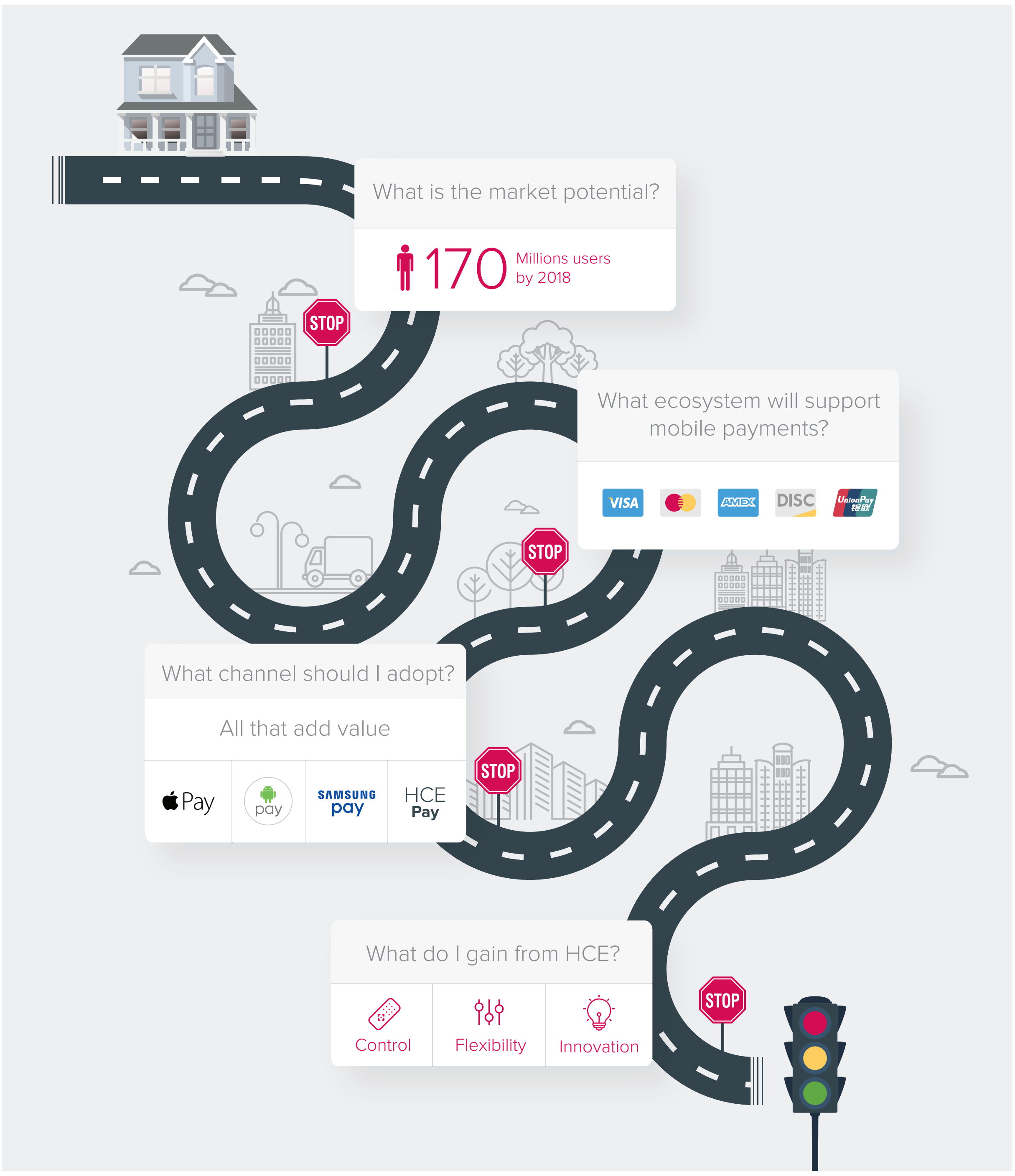


Figure 4 - MatrixHCE provides a rapid and secure development platform for Issuer HCE applications

Securing Mobile Payment applications requires more than just data encryption. In addition, developers must secure the overall application code with all its vital logic & processes, data, and the cryptographic keys. MatrixHCE utilizes INSIDE Secure's software protection tools to make it extremely difficult and time-consuming for attackers to understand how a payment application works in order to compromise it.



The HCE Payment Journey

About INSIDE Secure

INSIDE Secure (Euronext Paris – INSD) is at the heart of security solutions for mobile and connected devices, providing software, silicon IP, tools and know-how needed to protect customers' transactions, content, applications, and communications. With its deep security expertise and experience, the company delivers products having advanced and differentiated technical capabilities that span the entire range of security requirement levels to serve the demanding markets of network security, IoT security, content & application protection, mobile payment & banking. INSIDE Secure's technology protects solutions for a broad range of customers including service providers, content distributors, security system integrators, device vendors and semiconductor manufacturers. For more information, visit http://www.insidesecure.com.

INSIDE Secure S. A.

Arteparc Bachasson • Bât. A
Rue de la carrière de Bachasson
CS 70025 • 13590 Meyreuil • France

Tél : + 33 (0)4 42 90 59 05
Fax : + 33 (0)4 42 37 01 98

www.insidesecure.com

Driving Trust