MM App Security Best Practices
Version 1.0
29 June 2018

This is a White Paper of the GSMA

Security Classification: Non-confidential

Access to and distribution of this document is restricted to the persons permitted by the security classification. This document is confidential to the Association and is subject to copyright protection. This document is to be used only for the purposes for which it has been supplied and information contained in it must not be disclosed or in any other way made available, in whole or in part, to persons other than those permitted under the security classification without the prior written approval of the Association.

Copyright Notice

Disclaimer

The GSM Association ("Association") makes no representation, warranty or undertaking (express or implied) with respect to and does not accept any responsibility for, and hereby disclaims liability for the accuracy or completeness or timeliness of the information contained in this document. The information contained in this document may be subject to change without prior notice.

Antitrust Notice

The information contain herein is in full compliance with the GSM Association's antitrust compliance policy.

Table of Contents

1 Introduction

1.1 Overview

This document provides guidelines on best security practices for smartphone mobile money apps.

1.2 Scope

The guidelines described in this document are meant to raise industry awareness and understanding of security issues mobile money providers may face when deploying smartphone applications pre-loaded or available via download.

1.3 Definition of Terms

Term	Description
CA	Certification Authority – A trusted entity that issues digital certificates
DFS	Digital Financial Systems
IMSI	International Mobile Subscriber Identity - Unique number for identifying a GSM subscriber on a mobile phone network.
ITU	International Telecommunications Union –a specialized agency of the United Nations (UN) that is responsible for issues that concern information and communication technologies
NIST	National Institute of Standards and Technology - a non-regulatory agency of the United States Department of Commerce.
RC4	Rivest Cipher 4 - a stream cipher
SHA	Secure Hash Algorithms - A family of cryptographic hash functions published by the National Institute of Standards and Technology (NIST)
SSL	Secure Sockets Layer – A cryptographic protocol for secure Internet communication replaced by TLS
STK	SIM Application Toolkit - Set of commands programmed into the SIM to build up an interactive exchange between a network application and the end user
TCP/IP	Transmission Control Protocol (TCP) and the Internet Protocol (IP) – A set of communications protocols used on the Internet
TLS	Transport Layer Security – A cryptographic protocol for secure Internet communication
USSD	Unstructured Supplementary Service Data – A GSM signalling protocol

1.4 References

Ref	Title
[1]	Reaves et al., “Mo(bile) Money, Mo(bile) Problems: Analysis of Branchless Banking Applications in the Developing World”, 24th USENIX Security Symposium, August 12–14, 2015 https://www.usenix.org/node/190885
[2]	GSMA Intelligence Report on smartphones, https://www.gsmaintelligence.com/research/2017/02/smartphones-now-account-for-half-the-worlds-mobile-connections%20%20/600/
[3]	ITU X.805 : Security architecture for systems providing end-to-end communications https://www.itu.int/rec/T-REC-X.805-200310-I/en
[4]	Architecture of secure mobile financial transactions in next generation networks. ITU-T Study Group 13, January 2011 https://www.itu.int/itu-t/recommendations/rec.aspx?rec=Y.2741
[5]	J. H. Saltzer, D. P. Reed, and D. D. Clark. End-to-End Arguments In System Design. *ACM Trans. Comput. Syst.* 2, 277-288, 1984.
[6]	T. Dierks and E. Rescorla. The Transport Layer Security (TLS) Protocol Version 1.2. *Internet Engineering Task Force.* RFC 5246, 2008.
[7]	J. Salowey, A. Choudhury, and D. McGrew. AES Galois Counter mode (GCM) Cipher Suites for TLS. *Internet Engineering Task Force.* RFC 5288, 2008.
[8]	L. K. Gray. Date Change for Migrating from SSL and Early TLS., 2015. https://blog.pcisecuritystandards.org/migrating-from-ssl-and-early-tls
[9]	E. Rescorla. The Transport Layer Security (TLS) Protocol Version 1.3. *Internet Engineering Task Force.* Internet Draft, 2017
[10]	Android. Security with HTTPS and SSL., 2017. https://developer.android.com/training/articles/security-ssl.html
[11]	G. Chen. The Power of Smartphone Interfaces for Mobile Money. *CGAP Blog,* 2016. http://www.cgap.org/blog/power-smartphone-interfaces-mobile-money
[12]	T. Novais. Companion cards & mobile money: Industry landscape, insights and considerations. GSMA, 2016 https://www.gsma.com/mobilefordevelopment/programme/mobile-money/companion-cards-mobile-money-industry-landscape-insights-and-considerations
[13]	Mobile Connect: Mobile high-security authentication. GSMA, 2016 https://www.gsma.com/identity/wp-content/uploads/2016/10/MC_high-security-authentication_Sep-16.pdf
[14]	“Android phones rooted by “most serious” Linux escalation bug ever”, Dan Goodin, Ars Technica, October 24, 2016. https://arstechnica.com/information-technology/2016/10/android-phones-rooted-by-most-serious-linux-escalation-bug-ever/
[15]	Mobile App Collusion Can Bypass Native Android Security”, Michael Mimoso, Kaspersky ThreatPost, October 6, 2016. https://threatpost.com/mobile-app-collusion-can-bypass-native-android-security/121124/

1.5 Executive Summary

Mobile money services took off at a time when mobile devices became widely available for the low-income population. Providers took advantage of the ubiquitous text and USSD message capabilities of those devices to provide an interactive menu for mobile money transactions. In 2007, M-Pesa in Kenya reached scale by delivering a sound business

proposition to underbanked customers, paving the road for investment on digital financial services in many regions of the world.

The enabling mobile technology has evolved significantly since then and devices turned into “smartphones”, gaining more processing power, bigger screens and Internet access, still at affordable prices. Mobile pay-as-you-go Internet access also became part of the service offered by mobile operators. Mobile money providers, understanding opportunities accessible by the new ecosystem created by smartphones, began offering smartphone applications for their customers.

With regards to security, an essential asset for digital financial system providers, enhanced capabilities offered by smartphones pose opportunities and also potential threats. Internet access and open operational systems on smartphones can yield potential attackers compromising user data. However, the same operational systems also have a built-in set of tools to safeguard attacks, and Internet access allows applications to be upgraded promptly.

Mobile money providers have different levels of knowledge when it comes to leveraging adequate security on smartphone applications. A study published in 2015 by the University of Florida [1] showed lack of consistent security implementation principles for mobile money smartphone applications, raising the need to establish a common set of best practices amongst providers.

The document presented here, a joint effort between the GSMA and the University of Florida, aims to raise awareness on mobile money providers on security issues faced when developing smartphone applications, providing an initial set of safeguard recommendations. The principles revolve around a set of readily available security tools, which also includes good security protection levels.

Technology decision makers should use this document to guide technical teams or external developers to consider the recommendations. Technical experts can use and implement various sections or use the reference table provided at the end of the document for implementing a checklist on recommended best practices.

1.6 Objectives and methodology

Mobile money services have enabled financially underserved populations to safely store and transact money in digital form. However, despite evolution on internal IT systems from mobile money providers, the customer-facing interface remained until recently displayed on what is labelled today as feature phones devices that are able to handle basic cellular communication on voice calls, text messages (SMS) and Unstructured Supplementary Service Data (USSD) text strings. Notably from 2015, smartphones, or mobile devices with bigger screens, more processing power and Internet access, became an adoption trend in low- and middle-income countries, given decreasing device costs and greater access to mobile broadband [2] .Mobile money providers have quickly reacted to this trend by offering downloadable or preloaded applications for smartphones. As of November 2017, over 25% of the 276 mobile money providers registered by the GSMA tracker offer smartphone apps, and a clear upward trend is noted.

The use of smartphone-based mobile money apps has an additional potential benefit of vastly improving the security of mobile users. Conversely, if best practices for designing and implementing smartphone apps are not employed, serious security issues can result. The goal of this report, a joint work between the GSMA and the Computer and Information Science and Engineering Department at the University of Florida, is to identify best practices to address security issues within smartphone apps for mobile money and to provide mobile money operators, developers, and network providers with the tools that can be used to assure the security of their mobile money systems.

1.6.1 Security Principles

In order to evaluate the mobile money smartphone architecture with regards to criteria for security, both standards from regulators and other organizations were examined, as well as using work and expertise developed by the University of Florida on analysing the deployments of mobile money systems. ITU X.805 security dimensions [3] were used as a starting point. That work is aimed at regulators and considers the entirety of the Digital Financial Systems (DFS) infrastructure, with general recommendations for security, while in contrast, this report focuses specifically on smartphone apps for mobile money systems and is designed to be more technical and prescriptive in terms of recommendations. Thus, some of the security dimensions expressed in ITU X.805 and the security levels described in ITU-T Y.2741 [4] are a good starting point but are best tailored for the specifics of the smartphone environment and supporting server infrastructures found in mobile money deployments.

In particular, focus is given on the following security criteria:

1.6.2 Authentication

Authentication is methods of confirming the identities of communicating identities. In this work, authentication is considered within the mobile money server that clients connect with (generally done through use of the Transport Layer Security, or TLS, network protocol) and methods by which the client identifies itself (which often relies on a combination of user credentials and those provided by the device itself).

1.6.3 Access Control

The ITU-T X.805 [3] document is specific to network security, and thus defines access control as “protection against unauthorized use of network resources.” A more expansive view of access control is described in this document to consider unauthorized access not only over a network, but on servers and smartphones themselves, e.g., potentially unauthorized access to processes executing mobile money functionality or to the data that is used by these applications.

1.6.4 Integrity

Integrity is protection of the correctness and accuracy of data. Because integrity of information is vital to system security, it is particularly important and for ease of exposition, rather than creating a section of the report devoted strictly to issues of integrity, threats to data integrity are discussed in the context of authentication and access control, where failures to supply required protections can cause integrity threats to arise.

1.6.5 Confidentiality

Confidentiality is the protection of data from unauthorized disclosure. Confidentiality and integrity are both goals that are enforced through access control and authentication mechanisms (e.g., access control can prevent exfiltration of sensitive data, which is a breach of confidentiality), mechanisms by which confidentiality during communication of data between smartphone and mobile money server can be maintained are specifically described throughout the authentication section.

Other security criteria such as availability (i.e., resistance to denial of service) and privacy (i.e., protection of information from observation of activity) are worthy goals but are secondary in consideration to the security elements listed above.

1.6.6 Assumptions and restrictions

The principles above were moderated by considerations on the mobile money ecosystem, in particular trade-offs between usability and risks, limitations from entry-level smartphones, costs related to implementations and impact on IT staff resourcing.

For the smartphone, recommendations are based on the Android mobile operating system, as most entry and mid-level smartphones are based on this platform.

Finally, there is an assumption that Internet connection is available to the customer at speeds and latency levels compatible for mobile applications demanding low bandwidth, either through cellular network access or through Wi-Fi connections (public hotspots, home connections etc.).

2 Introduction to smartphone app components and architecture

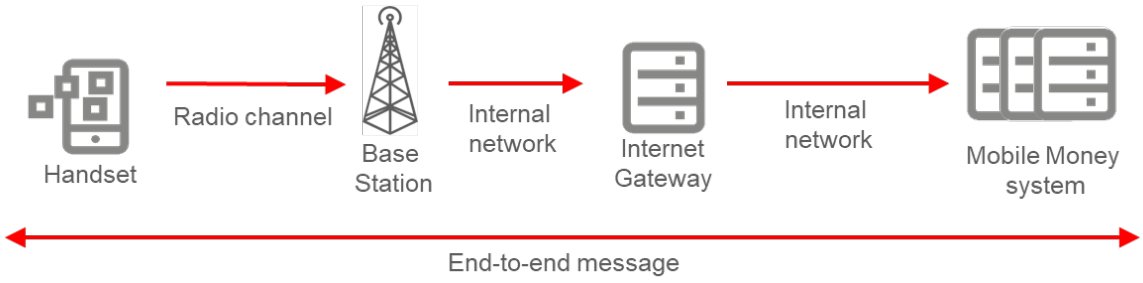

Figure 1: End-to-end security message flow

With regards to the smartphone app infrastructure, the major components are shown in the figure above. The handset (i.e., the smartphone) comprises of the physical platform for a user to interact with the mobile money application that executes on the smartphone. The handset can contain elements that aid in securing the platform, such as trusted execution environments and support for biometric authentication, both discussed in further detail below.

Smartphones support the installation of a SIM card**,** an integrated circuit chip designed to store the International Mobile Subscriber Identity (IMSI) number and its related key, in order to all mobile operators to identify and authenticate subscribers. It is possible that the SIM can contain a secure element that can also provide security rooted within the hardware, but primarily, security for the handset platform comes from components within the smartphone. Embedded SIM cards (eSIMs), which are integrated SIM chips that are not removed from the handset but are configurable by operators, may be incorporated into future devices.

Deployed on the handset is the mobile money app, which provides the interface between a user and the mobile money system they are using. Users can directly use the mobile money app, which resides directly on the smartphone in the majority of deployments rather than on the SIM card as in previous generations of mobile money phone deployments. The mobile money app provides a user interface to facilitate mobile transactions, payments, and other activities. Because of the importance of the activities that can be provided by the mobile money app and the existence of software design and implementation vulnerabilities that are often found in application code, this is a particularly target for attackers.

In smartphone apps, communication occurs over the TCP/IP network, connecting to a mobile base station over the Radio channel for further transit through the mobile provider network.[1] While this channel between the handset and the base station may be encrypted, smartphones support encryption directly between the device and the mobile money provider as described below in the discussion regarding the End-to-End Security Principle.

Data received by the base station may further pass through an Internet gateway for transmission through the mobile provider's internal network. The provider network provides transit connectivity for information originating at the handset as well as the gateway to external providers and the mobile money system. This end point is where the mobile money provider's servers are located, which process the information that originated at the handset. The security of these operations is reliant on the authenticity of the client as well as configuration of the servers.

2.1 The End-to-End Security Principle

The principle of end-to-end security goes back decades as a principle of designing computer systems and networks [5]. This means that it is possible to enforce properties of authentication, integrity, and confidentiality for all information that passes between the end device being used for a mobile money transaction and the back-end server that processes the transaction, without information being re-processed, decrypted and re-encrypted, or otherwise modified by intermediaries. Feature phones were able to provide end-to-end security provided cryptographic SIM Toolkit (STK) applications could be made available on the SIM Card.

[1] In TCP/IP deployments, it is also possible that mobile transactions can occur over Wi-Fi rather than through the cellular network. While some smartphone apps may still potentially make use of encrypted SMS messages to communicate to the mobile money provider, such deployments are not the focus of this analysis.

Low-end smartphones are able to provide end-to-end security guarantees at the operational system level, because of their ability to support symmetric and asymmetric cryptography using a wider combination of security algorithms. This means mobile money providers can ensure end-to-end security regardless of the underlying network transport and possible limitations on access to SIM Card memory.

3 Authentication and Data Confidentiality

Authentication and data confidentiality are at the centre of secure communication. Confidentiality is considered by many to be the original security problem, and much of the history of cryptography revolves around the challenge of ensuring that only authorized parties can gain access to the contents of a message.

Thankfully, there are many strong and freely available encryption techniques that allow any two communicating devices to guarantee[2] the security of their communications.

3.1 Considerations on TLS

Correctly deploying cryptography is a non-trivial problem. However, the computer security research and standards communities have made significant efforts to make the use of encryption easier. The most important contribution in this space is the widespread availability of the Transport Layer Security (TLS) protocol. TLS emerged from the Secure Sockets Layer (SSL) standard, which originated in the 1990s. Many refer to SSL/TLS interchangeably; however, the protocols themselves exist in different versions and are not compatible with each other. Because TLS represents the more modern of the two standards (SSL development ceased in the early 2000s), references will be exclusively related to it with regards to best practices.

TLS provides a scaffolding through which the communications of applications can be made secure. It provides for the validation of server identity (via the use of X.509 certificates), agreement on the use of specific cryptographic algorithms, the establishment of cryptographic keys and the application of these keys to preserve the confidentiality and integrity of all messages. Each one of these steps must be performed in a secure fashion, or the confidentiality of communications may be put at risk.

Significant effort has also been put into making TLS highly performant. While earlier versions SSL/TLS certainly increased the processing requirements of both servers and client devices, the most modern version of TLS (v1.2)[3] includes many optimizations that minimize its costs. As such, while many argued against the use of SSL/TLS in the 1990s because of

[2] Guarantees in cryptography are generally promised through the lens of “computational infeasibility”. That is, an adversary with access to massive computing power (e.g., multiple world-class supercomputers) should not be able to decrypt a message for many thousands of years despite having significant resources.

[3] TLS 1.3 has been released as a standard; however, there do not yet exist any widespread adoptions. It is expected that in the coming years, best practice will shift from v1.2 to v1.3 for reasons of improved performance and security.

performance overhead, such a claim holds essentially no weight in the modern smartphone ecosystem.

Finally, unlike the feature phone, all modern smartphones come equipped with TLS 1.2 and a selection of strong encryption algorithms. While more traditional web services may need to run older versions of TLS because of legacy applications (e.g., old email clients that have not been updated), the modern smartphone ecosystem lacks such legacy systems and should use the most up-to-date version of TLS.

Given that TLS is not patent encumbered and that free libraries are available in virtually every operating system, there is little practical reason not to run TLS v1.2.

It is possible to perform secure communications without TLS; however, by using a reference protocol available on virtually every operational system, many of the most difficult tasks are handled invisibly to the developer, providing fewer places where errors can be made (and data confidentiality compromised). Most critically, if an application developer decides not to use TLS, they should have to justify which desired security properties they were unable to achieve without the use of their own protocol.

3.2 Server Authentication

Because of the ability to use cryptographic libraries on low-end smartphone devices, server authentication becomes considerably more feasible. As stated, TLS is capable of providing server authentication, meaning that client devices (e.g., smartphones) can authenticate the server that they are connecting to (the usual method by which authentication on the Internet occurs, and what is responsible for the “green lock” icon in web browsers).[4]

The most recent form of the TLS protocol currently in common use is TLS 1.2, which was developed in 2008 [6] but has seen some changes in recommended cipher suite deployment. Within TLS, authentication occurs through presentation of a digital certificate by the server, which contains the server name, the certificate authority (CA) that signs the certificate (thus attesting to its authenticity) and the public key of the server. If the public key of the certificate has been pre-installed within the mobile device then it can verify the certificate chain offline, otherwise the mobile device can look up CA’s key online. It is strongly recommended that certificates offered by the server not be expired and that they are signed by a CA, not self-signed, since those cannot be effectively validated by the smartphone.

Recommendation R1:Ensure that TLS certificates presented by mobile money servers are not expired.

An innovation within TLS v1.2 compared to earlier versions was the support for ciphers that support authenticated encryption, notably Galois/Counter Mode (GCM) and Counter with CBC-MAC (CCM) mode supported as modes of encryption for the Advanced Encryption

[4] TLS also supports *mutual* authentication, whereby not only are servers authenticated to clients, but clients are also authenticated to servers. Typically this requires clients to provide certificates to servers and is a far less commonly used mode of operation for TLS, so we do not discuss it further in this report.

Standard (AES) symmetric cipher [7]. Compared to other encryption modes, these provide not just encryption (and hence, confidentiality) of data as it is transmitted between the smartphone and server, but also an authentication tag for each transmitted piece of encrypted data; the use of a message authentication code demonstrates that only the smartphone or the server, who share a symmetric key, could have generated the encrypted data. The means by which the symmetric key negotiation occurs is through a Diffie-Hellman key negotiation at the beginning of the TLS handshake between smartphone and server; a session key is generated through this negotiation that is only used for the duration of the TLS communication, where the duration of the session lifetime can be negotiated by the server.

The important things to note here with regards to authentication is that the selection of authenticated encryption ciphers provides better overall security for data than modes that only enforce encryption. Thus it is recommended with regards to TLS v1.2 that when possible, the server should accept TLS handshakes from clients that advertise support of the `TLS_DHE_RSA_WITH_AES_128_GCM_SHA256` cipher suite. Using an encryption cipher with a longer key lengths such as AES-256 is possible but may cause performance overhead that may nullify the small benefits of increased collision resistance, as there are no currently-known major attacks against AES-128. The use of SHA-384 is acceptable but may cause increased overhead compared to SHA-256.[5] Therefore at this time it is recommended that AES-128 and SHA-256 will suffice but mobile operators should be aware that in the future such assumptions may change. SHA-384 is based on the SHA-3 algorithm, which is also recommended for use.

While authenticated encryption provides additional guarantees, certain versions of smartphones may not support GCM mode. Therefore, additionally accepting ciphers that do not provide authenticated encryptions is recommended, such as `TLS_DHE_RSA_WITH_AES_128_CBC_SHA`, which provides support for cipher block chaining (CBC) mode, a well-known mode of operation for assuring confidentiality.

Recommendation R2:Include support for TLS cipher suites that provide authenticated encryption to maximize security. Also include support for ciphers that do not provide authenticated encryption for maximum compatibility with clients.

Recommendation R3:Hashing should be performed with the SHA-256 or SHA-3 algorithms, notably SHA-384.

Note: In all examples above, RSA, representing the use of RSA-2048 public key encryption, is used. It is recommended to use RSA-2048 preferably. While Elliptic Curve-based options exist and provide shorter key lengths, they may

[5] Theoretical attacks against SHA-256 show that it is potentially vulnerable to a length extension attack, where an attacker uses the hash of a message and the length of that message to add information into the message and successfully calculate a new hash. By contrast, SHA-384 is not susceptible to this attack. However, the length extension attack in SHA-256 not known to be practically deployable at this time, and the increased overhead of using larger key lengths may not justify the small potential security benefit.

be patent encumbered in some jurisdictions, and may therefore need to be selectively applied.

With regards to the negotiation of the session key, there are varying modes of Diffie-Hellman key exchange that are supported by TLS 1.2. It is strongly recommended to use ephemeral Diffie-Hellman, characterized by cipher suites that begin with the prefix `TLS_DHE_`. In this mode, security parameters are not fixed but are newly generated for different runs of the TLS protocol. This ensures that public keys are different per session and provides the security property of perfect forward security, which guarantees that even if the server's key is compromised, past sessions and the transmitted data are not similarly compromised. This would prevent an adversary who has captured past traffic from being able to use the compromised key to read all of this information.

The other modes of Diffie-Hellman are fixed and anonymous. Fixed Diffie-Hellman negotiation, characterized by cipher suites that begin with the prefix `TLS_DH_`, are not optimal. They require that the server's certificate contain the public parameter used for Diffie-Hellman negotiation, which never change, and prevents the ability to guarantee perfect forward security. The mode of anonymous Diffie-Hellman, characterized by cipher suites that begin with the prefix `TLS_DH_anon`, is not recommended. It leaves the key negotiation subject to a *man-in-the-middle* attack, where a malicious adversary that can capture communications from the smartphone (e.g., a malicious Wi-Fi access point or a rogue base station) can pose as the server, while passing credentials from the phone to the real server, and hence gaining knowledge of the session key and hence the ability to read and modify all communications.

Recommendation R4:For TLS connections, make use of ephemeral Diffie-Hellman modes for performing key exchange.

It is strongly recommended that all mobile money systems adopt TLS v1.2 as a standard for ensuring authentication. Previous versions of the TLS standard contain cipher suites and other practices that can be weak or are subject to exploit. It is also essential that TLS implementations be kept up to date, as implementation errors can lead to widespread attacks such as key exposure. These recommendations are consistent with those from the Payment Card Industry Security Standards Council (PCI SSC), who have mandated the retirement of all versions of TLS prior to v1.2 by the end of June 2018 [8].

Recommendation R5:Use TLS v1.2 to secure the communication between mobile money clients and back-end servers.

To ensure compliance with current best practices, it is recommended that mobile money providers and developers use services such as the Qualys SSL Server Test,[6] a free online service that tests properties of servers to determine how they are configured and what versions and types of TLS cipher suites are supported, assigning a letter grade to the service based on vulnerabilities exposed.

[6] https://www.ssllabs.com/ssltest/

Recommendation R6:Use independent TLS testing services to assure the correct server configuration.

Mobile money providers should also be aware of the upcoming TLS v1.3 protocol, which exists as a draft standard as of November 2017 [9]. This version of TLS will remove cryptographic ciphers that are deprecated or otherwise susceptible to attack and will speed up communication between clients and servers through the removal of round trips during protocol handshakes. Providers should watch developments closely and consider deploying support for TLS v1.3 once it becomes commonly available as it has positive implications for both security and performance.

3.2.1 Deploying TLS on Servers

For mobile money services that are not currently making use of TLS, there are a number of steps that should be followed to successfully deploy the protocol. The first step is to configure the user-facing web server to make use of TLS, which can be done through the web server configuration. A self-signed certificate can be used to test the configuration, but the important next step would be to purchase a certificate from a certificate authority, unless the mobile operator is a CA authority in itself. For most deployments, sharing a single certificate and associated key across multiple servers that are load balancing and serving content for the same domain is sufficient.

The CA/Browser Forum, representing the major certificate authorities and web browser providers, voted in early 2017 to limit the lifetime of a certificate to 825 days,[7] and it is recommended that any newly issued certificates do not exceed this lifetime. Shorter certificate validity periods ensure that valid certificates can remain in compliance with guidelines and recommendations that may change in the future, as well as reducing the number of outdated certificates that may contain vulnerabilities. It is also important to ensure all certificates are replaced prior to their expiration date.

Extended Validation purports to offer stronger guarantees of server identity but the degree to which these additional guarantees are necessary in practice is still an open question. The recommendation is that standard certificate purchases are sufficient.

Recommendation R7:Ensure that new issued certificates are limited in lifetime to 825 days.

3.3 Common Pitfalls

There are many ways in which cryptographic algorithms (and the protocols that use them) can be poorly applied or even misused. Such mistakes can quickly remove all guarantees of confidentiality and render user data vulnerable to recovery by an attack. Below is a list of the most common dangerous misconfigurations usually observed in the mobile money space for SSL/TLS:

[7] https://cabforum.org/2017/03/17/ballot-193-825-day-certificate-lifetimes/

- Failure to Authenticate: The use of X.509 certificates provides a basis for identifying the party with whom a smartphone application is speaking. Smartphone operating systems properly check to see that new TLS connections include a valid X.509 certificate. However, many applications override this check. It is believed that this behaviour is a result of attempts to silence errors during the testing process, and there exist many copies of such insecure code on public discussion boards.

  Failure to determine if a certificate is valid means that an attacker can inject their own certificate into a communication stream, and the smartphone will accept this certificate as valid for the mobile money provider with whom they were attempting to speak. As such, the smartphone will create a secure connection to an attacker, and then wilfully provide them with all their sensitive information.
  Bypassing certificate validation and server authentication should never be done in production code.

- Poor Algorithm/Mode Selection: The protections offered by TLS are only as good as the encryption and hashing algorithms selected by the developers. While TLS contains a wide selection of algorithms, not all options are necessarily considered strong. For instance, the Data Encryption Standard (DES), was standardized in 1975 and deprecated (i.e., recommended against use) in 2005. Simply applying this algorithm is not sufficient, and an adversary can now practically crack ("brute-force") such communications. A significant number of mobile money applications still allow the use of DES in 2017, giving both these enterprises and the customers they serve a false sense of security.

  Other algorithms that should not be used include Triple DES ("3DES"), which was deprecated in Fall of 2017 by the US National Institute on Standards and Technology (NIST), and RC4. Moreover, even when strong ciphers are chosen, many applications pick weak modes of encryption (e.g., Electronic Code Book – ECB) that weaken the confidentiality guarantees of the encryption algorithm.

- Outdated SSL/TLS Deployments: TLS v1.2 is the industry standard for protecting communications. However, older versions of the standard (SSL 2.0, SSL 3.0, TLS 1.0, TLS 1.1) are still run by many servers. These older versions not only offer generally lower performance, but many are also vulnerable to attack. For instance, SSL 3.0 is subject to downgrade attacks that allow an attacker to force the use of either weak ciphers (e.g., DES40, DES, RC4) or the NULL cipher (i.e., no encryption).

3.4 Client (end user equipment) Authentication

On the device side, all of the above recommendations should be similarly applied to mobile money apps.

Note: All versions of Android beginning with API level 16+ (Android 4.1 Jelly Bean, released July 2012) have support for TLS 1.2, and it is enabled by default for API level 20+ (Android 5.0 Lollipop, released November 2014). There are, however, differences in the ciphers supported by different API levels. For

example, authenticated encryption support through GCM mode is only supported at API level 20+. Similarly, while the use of SHA-256 hashing is recommended over SHA-1, SHA-256 is only supported as a cipher option for API level 20+.

The Android Developer guides have extensive information regarding best practices relating to building secure TLS sessions [10]. In particular, to ensure correct server authentication, the app should be able to handle the verification of server certificates. Generally, this validation will occur automatically in Android if a URL with an https:// prefix is specified as long as the certificate authority (CA) is well known (i.e., its root certificate is pre-installed into the smartphone). For instances when this is not the case, the `HttpsURLConnection` API call can instruct the app to trust a set of CAs such that intermediate CAs that comprise a certificate chain can be used.

Note: This operation will require the building of a `KeyStore` object that contains the set of trusted CAs and a `TrustManagerFactory` object to store them in.

This is essential to do if an `SSLHandshakeException` message is thrown (e.g., on account of intermediate CAs requiring verification) and may differ from many sample solutions found on the Internet that deal with this issue by recommending installing an empty `TrustManager` object without a corresponding `KeyStore`; by following this incorrect advice, the developer leaves the application open to an attacker generating a false certificate that is not validated to originate from a trusted source by the app, allowing attackers to capture and record all sensitive information sent across the TLS connection. To minimize issues arising on the device, the server should ensure that if intermediate CA certificates are required that the entire certificate chain ends when the TLS negotiation is requested. Additionally, it is recommended that if the URL for the mobile money service is offered on “toll-free” Internet navigation within a mobile operator, that the URL to allow the CA validating the certificate for the site also be added to the toll-free list, so that the integrity of the session can be assured.

Recommendation R8:Follow best practices for Android developers regarding server certificate verification when setting up TLS connections from the mobile money app on smartphones.

It is similarly important for the app to verify the proper host name of the server being contacted, to ensure that the correct certificate is being presented to the app. This can happen inadvertently as a result of misconfiguration on the server or as a side-effect of virtual hosting being employed. In either event, the use of the Server Name Indication (SNI) as a TLS extension (supported by default in TLS v1.2) allows the presentation of multiple certificates on the same IP address, and contacting a server from the app using the `HttpsURLConnection` call supports SNI by default for all versions of Android newer than v2.3. There is thus no reason to override host name verification as virtually any exception thrown can be dealt with using one of the methods above. Overriding verification can allow a man-in-the-middle attack to occur and sensitive information to be compromised.

Note: That when the certificate authority is not expected to change, the use of certificate pinning is recommended, since it allows only the specified CA certificate to be used for validating a connection rather than any root CA certificate installed in the smartphone.

Otherwise, the trust model is such that a compromise in any root CA could result in forged certificates that the smartphone will validate by default. Certificate pinning can be implemented through APIs such as `CertificatePinner`, which provides an advantage over using `KeyStore` objects discussed above as they allow not just the pinning of certificates but of public keys (the `SubjectPublicKeyInfo` [SPKI] field in a certificate), which can allow for certificate rotation that would not be otherwise possible if the certificate itself was pinned.

Recommendation R9:Ensure that mobile money apps verify the name of the server being contacted during TLS setup. Use certificate pinning if the CA is not expected to change.

3.5 User Authentication

Authentication of the user is critical. Both the smartphone app and the underlying device platform can play a role in establishing integrity of the user and the device that they are using to perform mobile money transactions with.

As a starting point, smartphones offer richer user interface and user experience (UI/UX). CGAP recently published a set of principles for UI/UX design [11] that mobile money app developers should consider adhering to. In particular, ensuring visual cues and locally relevant iconography are properly deployed can both have the benefit of improving aesthetics and authentication. Additionally, a more familiar interface can help users customers suspect and prevent other security vulnerabilities from happening (e.g., re-direction to an incorrect server address).

The most popular and straightforward to deploy mechanism for facilitating user authentication is through the use of a PIN or password. The smartphone environment provides substantially richer interfaces that can allow for moving beyond digit-based PINs that would be necessary on feature phones. This is beneficial for security since passwords can use a substantially larger character space, vastly increasing the universe of possible options for a password. However, it is anticipated that numeric passwords will still exist for some time to come, particularly in the case where companion cards [12] (which generally contain four-digit PINs) are linked with the same password as the mobile app. We thus recommend that mobile apps provide the means to disallow easily guessable numeric passwords, such as number sequences (e.g., “1234”), repeated digits (e.g., “1111”), user’s date of birth, and other easily-retrievable data. Similarly, when alphanumeric passwords are used, care should be taken within the app to disallow passwords that are easily guessable (those that are subject to *dictionary attacks*) based on ease of compromise (i.e., creating a blacklist of bad passwords and ensuring that users are not using one of these).

However, in line with recent recommendations NIST about password use,[8] it is recommended that apps not force users to change passwords on a regular basis and that apart from length requirements of 6 characters (or 8 if the password is numeric), no further restrictions be placed on the password to be selected by the user (i.e., no minimum number of digits, upper case characters, or special characters). While this recommendation may be counter to past practice, it is consistent with studies that show people have limited abilities to remember long and complex passwords and by forcing the constant changing of passwords, people are more likely to have the password written down and easily accessible to themselves (and potentially others) or to simply choose passwords that are not as strong as they are capable of recalling themselves. Longer passwords are preferred and providing users the options to input passphrases rather than passwords is recommended.

Recommendation R10:Disallow easily guessable PINs and passwords on mobile money apps, but do not force users to change passwords on a regular basis.

Authentication of users can often be characterized in terms of three factors:

1. what you know;
2. what you have;
3. who you are.

Passwords and PINs are examples of factor (1), but deploying the other authentication factors is becoming increasingly feasible for mobile devices. Factor (3) generally refers to the use of biometrics for authentication and such mechanisms have become common on mid-range and high-end smartphones. We anticipate that the cost of deploying these functionalities on mobile devices will continue to decrease and user acceptance of these features will continue to expand such that in a few years they will be predominant on all mobile devices. The precise mechanism for biometric authentication may differ between smartphone platforms (e.g., fingerprint scanners, face recognition systems) but all can provide a second factor of authentication to users. In particular, Android 6.0 contains APIs that support fingerprint authentication through the `authenticate()` method of the `FingerprintManager` class. Such data can be stored in KeyStore objects as with key material described above in our discussion of server authentication. The Android Keystore system ensures that key material and other sensitive information (e.g., the fingerprint credential) is not executed by the application itself but occurs as a system level process such that the application never has to directly process this information, mitigating exposure of this sensitive data.

Increasingly, smartphones are being deployed with hardware-backed secure components, such as trusted execution environments (TEEs, e.g., ARM TrustZone) or secure elements (SEs, e.g., deployed on SIM cards). Ensuring that sensitive information is stored in these hardware-backed secure features is straightforward for the application developer, who can use the `KeyInfo.isInsideSecureHardware()` API call in Android 6.0 and above to easily make this determination.

[8] https://pages.nist.gov/800-63-3/sp800-63b.html

In many cases, current programs that leverage biometrics require the initial entering of a password in addition to the biometric (e.g. a fingerprint) and subsequent authentication requests will only ask for the biometric. It is important that both authentication factors be presented initially. Because of the ease of use of biometrics, they also allow for more continuous authentication, where any security-sensitive operations (e.g., performing a money transfer) can be quickly authenticated without requiring the user to input their password; such authentication around security-sensitive events is strongly recommended regardless of the type of authentication that is being deployed. It is also strongly recommended to require input of the second factor of authentication at periodic intervals to ensure that the user is in possession of both credentials.

The third factor of authentication, factor (2) as described above, is determining the presence of a token used as an authentication credential. This is made particularly easy with a smartphone since it acts as the token itself.[9] When combined with hardware-backed security mechanisms such as trusted boot can enhance this trustworthiness. In Android 8.0, interfaces exist to retrieve an ID attestation of the mobile device directly from secure hardware, which can provide strong authentication guarantees regarding the device properties (e.g., the `ATTESTATION_ID_SERIAL` and `ATTESTATION_ID_IMEI` variables will securely retrieve the device's serial number and the IMEI of all radios, respectively, from a phone's TEE. These mechanisms should be leveraged by application developers who can build such support into applications for devices that support this secure hardware. We anticipate that secure hardware will be increasingly deployed in all smartphones in the future, and planning strategically to leverage this support for maximum protection of data will only help current mobile money deployments.

Recommendation R11:Ensure that user authentication is required on mobile money apps prior to performing security-sensitive operations.

Recommendation R12:When making use of biometrics as an authentication factor in mobile money apps, ensure that a password or PIN is also initially presented.

Recommendation R13:Mobile money apps should make use of trusted hardware on smartphones where such hardware is available to better secure sensitive information.

Another mechanism to support multi-factor authentication (1) and (2) where the smartphone acts as a token-based credential is one where information is communicated directly to the device from a server. In some cases this occurs through an SMS message to the phone; however, we would recommend for smartphones that the use of SMS be avoided because of the lack of confidentiality that SMS provides end-to-end;[10] also note that any other app on the smartphone that contains the `RECEIVE_SMS` Android permission would have the ability to read all incoming SMS messages, including those intended for the mobile money

[9] Other mechanisms such as hardware tokens plugged into the smartphone are also possible as an authenticator and may provide additional protection, but are subject to loss and may impede the user experience, so are not recommended for general use.

[10] NIST no longer recommends the use of SMS as a second authentication factor; see https://pages.nist.gov/800-63-3/sp800-63b.html

application. Instead, we recommend apps that generate one-time passwords on the device itself, such as Google Authenticator or the open-source FreeOTP provide strong notions of end-to-end security and act out of band to the mobile money application.

Recommendation R14:Use smartphone-based authenticators for one-time passwords rather than relying on SMS.

The GSMA has introduced Mobile Connect, an industry-led common authentication system[11] that serves as a mechanism for ensuring strong authentication through a consistent API. In this case, the credentials should still be stored through `KeyStore` objects as described above, preferably those that leverage secure hardware within the smartphone. The benefit of a solution such as Mobile Connect is the enhancement of interoperability for the mobile money application, since multiple mobile operators can provide the authentication credentials through the Operator Discovery phase of the protocol. Mobile Connect is also designed to be used in its most secure fashion with smartphones containing TEEs [13].

Mobile Connect specifies four Levels of Assurance[12] that provide progressively stronger assurances of authentication. PIt is recommended for providers to implement API support Level of Assurance 3. In the current term, MSISDN+PIN support is sufficient; however, given the sensitivity of mobile money apps data, it is recommend mobile app developers closely watch the evolution of the Mobile Connect API for completion of support for Level of Assurance 4, when support for smartphone app authentication is fully built in.

Recommendation R15:Make use of standardized APIs such as the GSMA's Mobile Connect when developing mobile money apps.

4 Access Control

Mobile devices have access to a significant amount of sensitive user data. From account information to login data, such data is tempting target for an attacker. App developers should consider how to best protect this information and to consider not just the security of the app but also the operating system and the underlying mobile platform that the app is to be deployed upon, as different security postures may exist for different app environments.

The Android mobile operating system is the primarily focus in this document as it is the OS platform for the vast majority of users of mobile money apps. Android itself is based on the Linux operating system kernel, meaning that any vulnerabilities in Linux can also percolate into Android. A prime example of this is the so-called "Dirty Cow" exploit, which leverages a vulnerability that existed within Linux for nine years before being patched; soon after the exploit was disclosed for Linux, it was developed to be able to exploit Android devices [14]. It is thus recommended that app developers stay cognizant of threats to both the Linux kernel and Android operating system and ensure that they work in conjunction with mobile network operators to push operating system updates to phones capable of taking them; applications

[11] https://www.gsma.com/identity/mobile-connect

[12] https://developer.mobileconnect.io/level-of-assurance

should also be written using defensive programming practices to attempt to mitigate vulnerabilities should exploits occur.

Recommendation R16:Maintain awareness of vulnerabilities against the Android operating system and the underlying Linux kernel, and what effect those exploits may have on the security of mobile money apps.

App developers should exercise great caution with regards to where application data is stored. As described in previous sections, the use of `KeyStore` objects is recommended for sensitive information, as the operating system will handle information protection using software mechanisms and, where available, hardware mechanisms. However, other information not stored in the Android KeyStore should also be evaluated for its potential to be used inappropriately and protected accordingly.

One of the issues that should be considered is the partition and information storage object used for information relating to the mobile money app. For example, data in Android is often stored in content providers, which manage access to local data stores on the smartphone. It is recommended that if mobile money apps are to use `ContentProvider` objects, that these objects not provide other applications with access to the object, i.e., within the manifest for the app, marking `android:exported=false` to disable other applications from accessing the `ContentProvider`. Shared content provider objects can provide an unintentional means by which applications can share information and can provide mechanisms for malicious apps to collude to steal information from sensitive data stores used by the mobile money app [15].

If there is a need for multiple apps to share access to a `ContentProvider` used by the mobile money app (e.g., a companion mobile wallet app), it is recommended that app developers closely examine the flows of information to the `ContentProvider`. There must be no pathway by which potentially sensitive information can leave the provider object and used by another app with access to information from the ContentProvider without having communication privileges to the mobile money app. Such a scenario would be possible if, for example, a third-party app has permissions to access a companion mobile wallet app that can access the same `ContentProvider` as the mobile money app. If such as flow was possible then that third-party app would be able to potentially retrieve sensitive information used by the mobile money app that it was not authorized to use. This can allow the third-party application to also potentially pass the mobile wallet app information that is then placed into the ContentProvider object and subsequently used by the mobile money app. In this way, the third-party app would have the ability to influence the mobile money app's activities by indirectly providing it with data; this is known as a confused deputy attack. Ensuring that these types of attacks do not occur is critical to the integrity of the mobile money application.

Recommendation R17:Use the Android Keystore for storing sensitive information and avoid allowing access to databases where sensitive information is stored to applications other than the mobile money app.

Another shared channel for information exists through placing information in external storage. Regardless of whether the smartphone contains the ability to access external

storage media (e.g., an SD card) or not, Android filesystems have the ability to partition all accessible storage (including information that only resides on the phone's internal data stores) into internal and external storage. Once data is placed in external storage, it becomes readable and writable to any other application that resides on the smartphone. It is therefore recommended that data used by the mobile money app only be stored onto a data storage area designated as internal storage. However, if any data is must retrieved from external storage, it is strongly recommended that this data be subject to validation by the mobile money app before it is used. It should be considered low integrity and potentially malicious data and the app should ensure that the data conforms to the structure and content that is expected.

The user of permissions should also be minimized to the minimum necessary for the app to effectively function. This is an example of the principle of least privilege, which can protect the information within the app. Importantly, it is recommended that app developers ensure any information sent over inter-procedural calls does not include information that was retrieved through request of a system permission.

Recommendation R18:Avoid the use of external storage for information stored in mobile money apps and minimize the number of permissions required.

4.1 Protection of User Credentials and Sensitive Information

The certificate keys that are used to validate secure connections between the smartphone and the mobile money server must be properly protected on the device. There are a number of ways in which this can occur. It is recommended that the most secure way of achieving secure storage of this information is through use of the Android KeyStore mechanism described above.

Other mechanisms have been deployed in the past and can be used for legacy code with some attention paid to security. For example, app developers may manage this data through Android `ContentProvider` interfaces to secure data repositories placed in encrypted storage. In legacy deployments where refactoring application code to use ContentProviders is difficult, there are other alternatives. Data can be stored in hardcoded form on the device, bundled inside the application. In this case, it is recommended that app developers ensure data is properly managed since the certificate details, if they change, will need to be accompanied by a corresponding change in the application that would have to be pushed to consumer devices. Alternatively, the application can leverage trust on first use with regards to the certificate information retrieve and store this data securely on the device. In this manner, the application would not be responsible for distribution of the information, but it then becomes particularly important to ensure the trustworthiness of the initial connection to the server. Finally, a separate data store could be established on the smartphone and the mobile money app could dynamically update this store over the air. In this case, it is also critical to ensure the authenticity and the integrity of both the server and the data in flight from the server to the app. If data is stored within the app itself, recommended practice is to use techniques such as obfuscation to protect it. We note, however, that de-obfuscation can be performed and determined attackers could potentially gain access to any sensitive data embedded within the app.

5 Technical Recommendations Summary

A summary of the recommendations made in this document are provided in the table below. These recommendations are based on current best practices as supported by organizations such as NIST and the consensus of the security community.

Ref	Recommendation	Type	Remarks
R1	Ensure that TLS certificates presented by mobile money servers are not expired.	Highly recommended	
R2	Server administrators should Include support for TLS cipher suites that provide authenticated encryption to maximize security. Also include support for ciphers that do not provide authenticated encryption for maximum compatibility with clients.	Recommended	An example of a recommended authenticated encryption mode is `TLS_DHE_RSA_WITH_AES_128_GCM_SHA256`. An example of a recommended mode without authenticated encryption is `TLS_DHE_RSA_WITH_AES_128_CBC_SHA`.
R3	Hashing should be performed with the SHA-256 or SHA-3 algorithms, notably SHA-384.	Recommended	
R4	For TLS connections, make use of ephemeral Diffie-Hellman modes for performing key exchange.	Recommended	These modes start with the prefix `TLS_DHE`.
R5	Use TLS 1.2 to secure the communication between mobile money clients and back-end servers.	Highly Recommended	
R6	Use independent TLS testing services to assure the correct server configuration.	Recommended	
R7	Ensure that new issued certificates are limited in lifetime to 825 days.	Recommended	
R8	Follow best practices for Android developers regarding server certificate verification when setting up TLS connections from the mobile money app on smartphones.	Highly Recommended	
R9	Ensure that mobile money apps verify the name of the server being contacted during TLS setup. Use certificate pinning if the CA is not expected to change.	Recommended	

Ref	Recommendation	Type	Remarks
R10	Disallow easily guessable PINs and passwords on mobile money apps, but do not force users to change passwords on a regular basis.	Highly Recommended	Minimum PIN/password lengths are recommended as 6 characters for an alphanumeric password and 8 digits for a numeric PIN.
R11	Ensure that user authentication is required on mobile money apps prior to performing security-sensitive operations.	Highly Recommended	
R12	When making use of biometrics as an authentication factor in mobile money apps, ensure that a password or PIN is also initially presented.	Recommended	
R13	Mobile money apps should make use of trusted hardware on smartphones where such hardware is available to better secure sensitive information.	Recommended	
R14	Use smartphone-based authenticators for one-time passwords rather than relying on SMS.	Recommended	
R15	Make use of standardized APIs such as the GSMA's Mobile Connect when developing mobile money apps.	Recommended	If using Mobile Connect, applications should be built against Level of Assurance 3 APIs.
R16	Maintain awareness of vulnerabilities against the Android operating system and the underlying Linux kernel, and what effect those exploits may have on the security of mobile money apps.	Recommended	
R17	Use the Android Keystore for storing sensitive information and avoid allowing access to databases where sensitive information is stored to applications other than the mobile money app.	Highly Recommended	
R18	Avoid the use of external storage for information stored in mobile money apps and minimize the number of permissions required	Recommended	If external storage is required, ensure that apps fully validate input from this storage area prior to use within the app.

6 Conclusions

This document provides a set of recommendations designed with the needs of mobile providers and app developers in looking to deploy mobile money apps on the Android operating system in mind, assuring the security of the communication between these apps and the back-end servers that they connect to.

Good practices considered revolve around basic security principles readily available on the underlying platforms and operating system, which could be common knowledge for most IT developers. However, the recommendations set here should provide a checklist to ensure providers satisfactorily fulfil the security principles on authentication, access control, integrity and confidentiality,

It should be noted that these recommendations are far more focused on the technical details to provide a tactical approach on implementation rather than on procedural elements. In comparison, frameworks such as the US Sarbanes-Oxley Act and those from ISO can provide value for general IT policies and practices but are far more general in nature and are not tailored to the mobile money environment. The ITU issued recommendations for the DFS ecosystem as well, but those recommendations are also more general and less technical in nature and attempt to cover a broader spectrum of the digital financial services ecosystem.

While this document is designed with the needs of mobile providers and app developers in looking to deploy mobile money apps on the Android operating system in mind, information such as recommended cipher suites for TLS and recommendations on access to permissions and storage will hold for other mobile operating systems as well such as Apple’s iOS.

Finally, because these recommendations are technical and based on current best practices, while they are designed to be forward-looking in nature, they are tailored to current deployments. As such, it is possible that new exploits, vulnerabilities, and advances in computer security may render some of these recommendations obsolete in the future. To that end, the goal of the GSMA will be to update these recommendations in the future such that it remains a valuable and current resource for any mobile provider planning to launch smartphone-based mobile money systems.

Annex A Document Management

A.1 Document History

Version	Date	Brief Description of Change	Approval Authority	Editor / Company
1.0	07th February 2018	Initial version	TG and PLG	Tiago Novais, GSMA

A.2 Contributing authors and owners

Type	Description
Document Owner	GSMA Mobile Money Programme
External Contributor	University of Florida, Computer and Information Science and Engineering Department
Lead External Authors	Kevin Butler and Patrick Traynor, UF
External reviewers	Gareth Pateman, MFX Partners
Lead GSMA Reviewer	Tiago Novais
GSMA reviewers	Richard Murray, Sophia Hasnain

This document has additional contributions from Airtel, Millicom, MTN, Ooredoo. Oragne, Telenor, Vodafone, Jazz (Mobilink) through feedback comments directly on this document or at a workshop held at the GSMA offices on 01st of September, 2016.

It is our intention to provide a quality product for your use. If you find any errors or omissions, please contact us with your comments. You may notify us at prd@gsma.com

Your comments or suggestions & questions are always welcome.