Security Design and Implementation Guidelines

GSMA Mobile Money API

This is a Non-binding Permanent Reference Document of the GSMA

Security Classification: Non-confidential

Copyright Notice

Disclaimer

Antitrust Notice

Table of Contents

1 Introduction

The Mobile Money environment is fragmented with each platform vendor offering their own API. For this reason, the GSMA defined a RESTful harmonized Mobile Money API to standardize the connection between API Clients (e.g. Merchant, Aggregators, Utility Companies) and the Mobile Money Platforms.

Section 2 of Security design document for the GSMA Mobile Money API details the security methods to be implemented for the connection between the API client and the API Gateway. This security design provides guidelines on the security methods to be used and best practices for the platform/gateway providers.

Section 3 of Security design document for the GSMA Mobile Money API details the security methods to be implemented for securely authenticating end user to the Mobile Money platform. It captures various scenarios for authenticating end users using industry standard authentication and authorization protocol – OAuth 2.0 [22] and OIDC [27] along with custom authentication models to support existing username/MSISDN and PIN based credentials.

Objectives which are achieved by implementing the security mechanisms and best practices as described in this security design are to ensure confidentiality, integrity and authentication on the interface between the API Client and the API Gateway.

With this harmonized Mobile Money API the GSMA aims to provide easy and secure building blocks and rapid partner on-boarding and interoperability between multiple Mobile Money deployments/implementation. This security design describes the security mechanisms to be applied to this API.

REST APIs like the GSMA Mobile Money API, expose resources that could be associated to sensitive information about the user and his actions can be subject to malicious activities by third parties. We need to consider that to secure the information, it is not enough to use only cryptography mechanisms at the network layer, but it is also important to address the pillars of secure computing which will be further defined in section 2.2.

Security Pillar	Description
Confidentiality	The ability to keep information private while in transit
Integrity	The ability to prevent information from being changed undetectably while in transit.
Authenticity	The ability to verify that a message is originating from a specific source and hasn't been altered.
Availability	Information exchange through the API should always be available when needed. This implies that systems, access channels, and authentication mechanisms must all be working properly.

1.1 Objective

This security design for the GSMA Mobile Money API is to ensure that:

1. Applicable security measures and best practices are applied to the connection between the API Client and the API Gateway.

2. Applicable security measures and best practices are applied to authenticating end user to the Mobile Money platform.

1.2 Actors

Actor	Description
API Client	The backend system of the clients of the API. These will be systems from e.g. Merchants, Aggregators, Utility Companies.
API Gateway	The API Gateway is the entry point for API Clients to connect to the Mobile Mobile Platform. This API Gateway is the layer of harmonization standardized by the GSMA at this moment to provide a generic interface for API Clients across different Mobile Money Platform vendors.
Relying Party	A Relying Party can either be API Client and API Gateway and integrates with 3rd party IDP or OAuth 2.0 authorisation server for authenticating and authorising end user. Some examples of 3rd party IDP are GSMA's Mobile Connect, Facebook Connect, Google IDP etc.,
End user	User performing mobile money transactions on the consumption device and who will be authenticated on the authentication device as per the proposed security models in this document.
OIDC compliant Identity Provider	The entity providing the authentication and identity services for authenticating end users, e.g. GSMA's Mobile Connect, Facebook Connect, Google IDP
Consumption Device	This is the device where the user is consuming the service from the SP. This can be any Internet connected device, e.g. a mobile device, a laptop, table, smart TV etc. The access network used by this device can be any as long as it can initiate an HTTP(S) interaction.
Authentication Device	This is the device where the end user is authenticating or providing authorisation. This device is always a mobile device, connected to the mobile network.
Consent Device	The consent device is the logical device through which the end user provides consent to the IDP system, e.g. providing consent to debit wallet account in case of P2P transfer.
Authenticator	Authenticators are the authentication mechanism used by 3rd party IDP to authenticate the user. Some examples of authenticators are USSD Authenticator, SIM Applet Authenticator, Smartphone App Authenticator

1.3 Common OAuth 2.0 terms

Actor	Description
Resource owner	An entity capable of granting access to a protected resource. When the resource owner is a person, it is referred to as an end-user. API Client plays this role in the case of GSMA Mobile Money API.
Resource server	The server hosting the protected resources, capable of accepting and responding to protected resource requests using OAuth access tokens. API Gateway is a resource server responsible for OAuth token validation to process API requests. API Gateway interacts with its authorisation server for OAuth token validation.
Authorisation server	The authorisation server is implemented in compliance with the OAuth 2.0 specification, and it is responsible for validating authorisation grants and

Actor	Description
	issuance of access tokens that give the client access to the protected resources on the resource server. It should be possible to configure "token endpoints" on API Gateway, in which case the API Gateway takes on the role of authorisation server. Alternatively, the API Gateway can use a third party OAuth 2.0 compliant authorisation server.
Client Credentials grant type	The client credentials grant type can be used as an authorisation grant when the authorisation scope is limited to the protected resources under the control of the client. Client credentials are used as an authorisation grant typically when the client is acting on its own behalf (the client is also the resource owner), or is requesting access to protected resources based on an authorisation previously arranged with the authorisation server. This is the recommended grant type for authenticating API Client to API Gateway.
Authorization code grant type	Considered the most secure grant type. Before the authorization server issues an access token, the RP must first receive an authorization code from the resource server. In this flow, 3rd party app opens a browser to the resource server's login page. On successful log in, the app will receive an authorization code that it can use to negotiate an access token with the authorization server. This grant type is considered highly secure because the client app never handles or sees the user's username or password for the resource server. This grant type flow is also called "three-legged" OAuth. This is one of the recommended grant type for authenticating end users to API Gateway.
Access token	Access tokens are credentials used to access protected resources. An access token is a string representing an authorisation issued to the client. The string is usually opaque to the client. Tokens represent specific scopes and durations of access, granted by the resource owner, and enforced by the resource server and authorisation server.
Protected resource	Data owned by the resource owner. In case of GSMA Mobile Money API, the protected resources are identified by API resources URL.
Access token scope	The access token endpoint allow the client to specify the scope of the access request using the "scope" request parameter. In turn, the authorisation server uses the "scope" response parameter to inform the client of the scope of the access token issued. The value of the scope parameter is expressed as a list of space-delimited, case-sensitive strings. The strings are defined by the authorization server. This parameter can be used by API Gateway to control the access to different resources. It should be possible to group the API set into individual product set each identified by a "scope" value. The API Gateway can decide to assign these scope values to specific API Clients based on policy and licensing rules thereby enforcing authorisation of endpoints.

1.4 Scope

Section 2 of the security design focusses on protecting the interface between the API Gateway and other API Client systems which will always be backend systems. Examples of API Client applications are the following:

- Merchants
- Merchant aggregators
- Utility companies (for bill payments)
- Other Mobile Money platforms

Section 3 of the security design focusses on authentication of the end user using different security models. The proposed models in this document are:

1. End user authentication using 3 legged OAuth flow
2. End user (Debit party) authorisation by a 3rd party IDP (Delegated authorisation)
3. End user authentication based on username/MSISDN and PIN

Section 4 of the security design focusses on best practices in API design.

1.4.1 Interfaces

Interface 1: The interface between the API Client and the API Gateway
This interface is the interface which is in scope in this security design and for which integrity and confidentiality are considered. The security design covers the transport layer security on this interface as well as application level authentication and authorisation. The security mechanisms that must be applied to this interface are described in chapter **Error! Reference source not found.**.

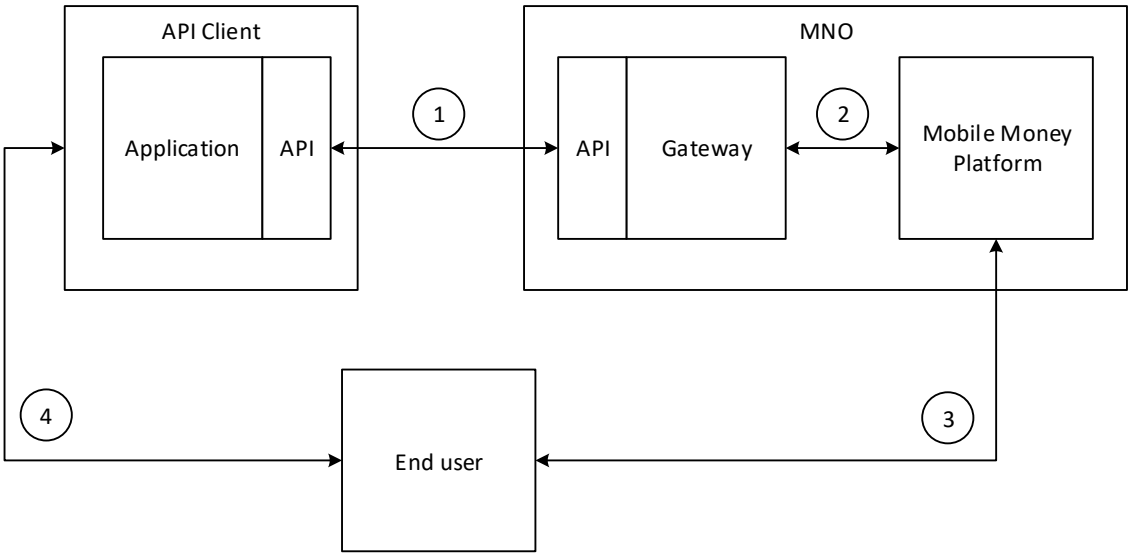

Figure 1: Solution overview including interfaces

Figure 1 also provides an overview of the other interfaces in a Mobile Money solution which are shortly described below but these are considered out of scope for this security design.

Interface 2: Connection from API Gateway to Mobile Money platform
This connection is the proprietary interface for the connection to the existing Mobile Money platform. This connection depends on the vendor chosen for the Mobile Money platform.

Interface 3: Interface between the Mobile Money platform and end user

For authentication within some use cases the Mobile Money platform might request the end user to provide an authentication. This interface could be IP based or be using protocols like USSD.

Interface 4: Interface between the end user and the API Client
In some use-cases the end user will communicate to the API Client. This interface can be digital but it can also be different in the case of smaller shops.

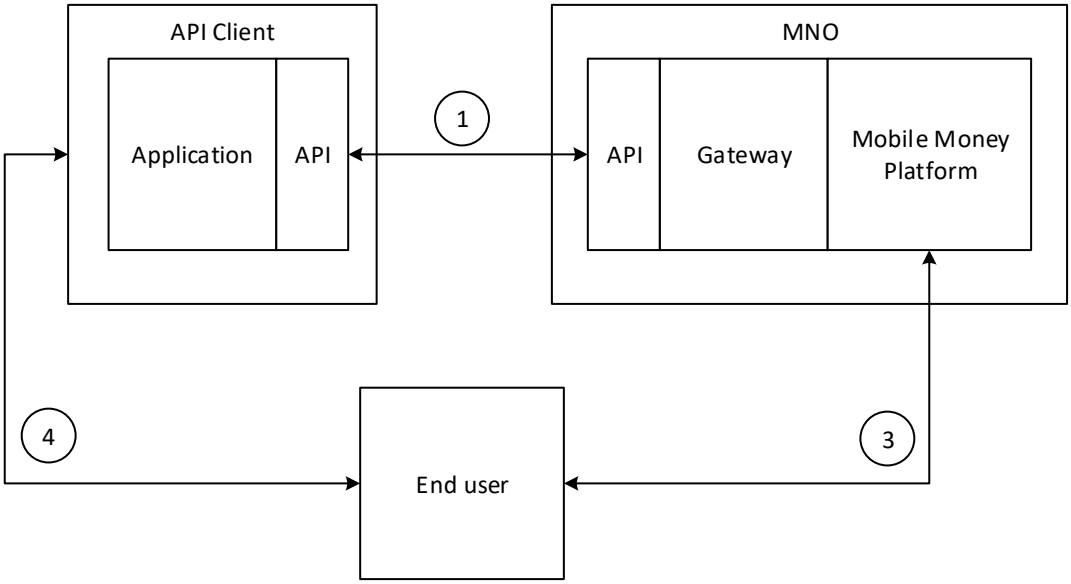

Figure 2: Alternative where the API Gateway is directly connected to the Mobile Money Platform

Figure 2 provides an alternative overview for when the API Gateway is integrated within the Mobile Money Platform. In this case interface 2 does not exist.

1.5 Intended audience

This security design is targeted for Mobile Money platform vendors and operators to guide them in implementing, setting up, and/or deploying a Mobile Money Platform compliant to the GSMA Harmonized Mobile Money API.

Mobile Money Platform vendors will need to adapt their interface to API Clients by implementing the harmonized API as defined by the GSMA and perform remapping of the data elements between the API Gateway and their Mobile Money Platform.

1.6 Document structure

The document acts as a single reference for the security design and implementation guidelines of GSMA Mobile Money API. The document is structured as follows:

Chapter **Error! Reference source not found.**	Describes the security principles, tiered security model and common security mechanisms should be implemented and used on the GSMA Mobile Money API to ensure these security principles can be achieved. A summary section provides summary for recommended security option.

Chapter 3	Describes various security models for authenticating and identifying end users
Chapter 4	Describes common best practices that must applied during the development phase

1.7 Conventions

The key words “must”, “must not”, “required”, “shall”, “shall not”, “should”, “should not”, “recommended”, “may”, and “optional” in this document are to be interpreted as described in RFC2119 [24].

1.8 Glossary & Abbreviations

Abbreviation	Description
AES	Advanced Encryption Standard
API	Application Programming Interface
BAM	Business Activity Monitoring
CA	Certificate Authority
CEK	Content Encryption Key
Consent	Agreement that SP can use the attributes they're requesting
Consent Device	The device through which the user provides consent for the sharing or validation of attributes
Consumption device	The device through which the user is accessing and consuming mobile money service
CRL	Certificate Revocation List
DDoS	Distributed Denial of Service
GSMA	GSM Association
HTTP(S)	HyperText Transfer Protocol (Secure)
Identity Token	Provides a set of metadata regarding the Authentication to the SP. This includes the PCR, authenticator used, Level of Assurance etc.
IETF	Internet Engineering Task Force
IP	Internet Protocol
JOSE	Javascript Object Signing and Encryption
JWA	JSON Web Algorithm
JWE	JSON Web Encryption
JWK	JSON Web Key
JWS	JSON Web Signing
JWT	JSON Web Token
KPI	Key Performance Indicator
MAC	Message Authentication Codes
MITM	Man-in-the-middle attack - is an attack where the attacker secretly relays and possibly alters the communication between two parties who believe they are directly communicating with each other
MLS	Message-Level Security - focuses on ensuring the integrity and privacy of individual messages, without regard for the network
MMP	Mobile Money Platform
MNO	Mobile Network Operator
OIDC	OpenID Connect
OWASP	Open Web Application Security Project
PII	Personally Identifiable Information
PKI	Public Key Infrastructure
REST	Representational State Transfer
RFC	Request For Comments
RSA	Asymmetric Encryption algorithm named after inventors: Rivest, Shamir, Adleman
RP	Relying Party (The application/service that needs the authentication and identity services). It can either be API Client or API Gateway.
Scope	Pre-defined collection of attributes that are logical to group together either for sharing or for simplifying policy management

Abbreviation	Description
SHA	Secure Hash Algorithm
SIEM	Security Information and Event Management
SLA	Service Level Agreement
SSL	Transport level security is based on Secure Sockets Layer (SSL) - The SSL is the industry accepted standard protocol for secured encrypted communications over TCP/IP
TLS	Transport-Level Security - such as HTTP Basic/Digest and SSL, is the usual "first line of defence", as securing the transport mechanism itself
XML	Extensible Markup Language
UMA	User Managed Access
USSD	Unstructured Supplementary Service Data
VPN	Virtual Private Network

1.9 References

Ref.	Title	Author	Date
[1]	RFC7515 - JSON Web Signature (JWS)	IETF	05-2015
[2]	RFC7516 - JSON Web Encryption (JWE)	IETF	05-2015
[3]	RFC7517 - JSON Web Key (JWK)	IETF	05-2015
[4]	RFC7518 - JSON Web Algorithms (JWA)	IETF	05-2015
[5]	RFC7519 - JSON Web Token (JWT)	IETF	05-2015
[6]	RFC7520 - Examples of Protecting Content Using JSON Object Signing and Encryption (JOSE)	IETF	05-2015
[7]	RFC4648 - The Base16, Base32, and Base64 Data Encodings	IETF	10-2006
[8]	RFC5246 - The Transport Layer Security (TLS) Protocol Version 1.2	IETF	08-2008
[9]	REST Security Cheat Sheet https://www.owasp.org/index.php/REST_Security_Cheat_Sheet	OWASP	04-2015
[10]	RESTful Service Best Practices, Recommendations for Creating Web Services http://www.restapitutorial.com/media/RESTful_Best_Practices-v1_0.pdf	Todd Fredrich	04-2012
[11]	NIST Special Publication 800-122 : Guide to Protecting the Confidentiality of Personally Identifiable Information (PII)	NIST	04-2010
[12]	ISO/IEC 27000:2014: Information technology -- Security techniques -- Information security management systems -- Overview and vocabulary	ISO	2014
[13]	ICT guidelines for TLS: https://www.ncsc.nl/actueel/whitepapers/ict-beveiligingsrichtlijnen-voor-transport-layer-security-tls.html	Dutch Ministry of safety and Justice	11-2014
[14]	PayPal security guidelines and best practices https://developer.paypal.com/docs/classic/lifecycle/info-security-guidelines/	Paypal	
[15]	JSON and XML Threat Protection Policies	MuleSoft	

Ref.	Title	Author	Date
	https://docs.mulesoft.com/anypoint-platform-for-apis/json-xml-threat-policy		
[16]	JSON Threat Protection policy http://docs.apigee.com/api-services/reference/json-threat-protection-policy	APIGEE	
[17]	JSON Threat Protection https://help.hana.ondemand.com/apim_od/frameset.htm?952cbd7d32c342788ba699227e734547.html	SAP HANA	
[18]	Quota policy https://github.com/apigee-127/a127-documentation/wiki/Quota-reference	Will Witman	08-2015
[19]	Spike Arrest Quick Start https://github.com/apigee-127/a127-documentation/wiki/Spike-Arrest-Quick-Start	Will Witman	03-2015
[20]	SANS Institute InfoSec Reading Room - Four Attacks on OAuth - How to Secure Your OAuth Implementation https://www.sans.org/reading-room/whitepapers/application/attacks-oauth-secure-oauth-implementation-33644	SANS	
[21]	RFC7617 – Basic HTTP Authentication Scheme	IETF	09-2015
[22]	RFC6749 - The OAuth 2.0 Authorization Framework	IETF	10-2012
[23]	RFC6750 - The OAuth 2.0 Authorization Framework: Bearer Token Usage	IETF	10-2012
[24]	RFC2119 - Key words for use in RFCs to Indicate Requirement Levels	S. Bradner	03-1997
[25]	OWASP Top Ten Project - https://www.owasp.org/index.php/Top10#OWASP_Top_10_for_2013	OWASP	06-2013
[26]	OWASP Cryptographic Storage Cheat Sheet - https://www.owasp.org/index.php/Cryptographic_Storage_Cheat_Sheet	OWASP	08-2016
[27]	OpenID Connect “An interoperable authentication protocol based on the OAuth 2.0 family of specifications” available at http://openid.net/specs/openid-connect-core-1_0.html https://openid.net/specs/openid-connect-basic-1_0.html	IETF	11-2014

2 API Client Authentication – Security Design

2.1 Solution overview

The API Gateway is responsible for confidentiality and integrity/authenticity on interface 1 (API Client to API Gateway), see Figure 1, and API level authentication of the API Client. The Mobile Money Platform is responsible for end user authorisation - whether this entity is allowed to access, create or modify the information, e.g. Transaction, Quotation.

Principles	Responsible component
Confidentiality and integrity/authenticity of the messages exchanges on interface 1: API Client – API Gateway	API Client - API Gateway
API Client authentication	API Gateway
Server Authentication	API Client

Table 1: Responsibilities of components

Authorisation of the end user is delegated to the Mobile Money Platform by the API Gateway by forwarding the end user identity.

Both confidentiality and integrity are not possible to the Mobile Money Platform as the gateway needs to be able to read the data to perform the mapping of the data elements to the format used by the Mobile Money Platform which depends on choices made by the vendor that delivered the platform. During this mapping it would be possible to change the data. This is why the API Gateway and mapping functionality should be performed in a trusted system.

2.2 Security principles

This section describes the definitions for the security concepts.

2.2.1 Confidentiality

In information security, confidentiality "*is the property, that information is not made available or disclosed to unauthorized individuals, entities, or processes*" [12]. This means protect data from unintended recipients, both at rest and in transit. **Error! Reference source not found.** below presents a comparison between Transport-Level security and Message-Level security.

Transport-Level Security	Message-Level Security[1]
Relies on the underlying transport	No dependency on the underlying transport
Point-to-point	End-to-end
Partial encryption not supported	Partial encryption supported
High performance	Relatively less performance

Table 2: Transport-Level Security vs. Message-Level Security

[1] With only application level security HTTP header information might be readable by eavesdroppers.

The use of TLS and MLS is complementary and the combination of both will enhance the overall end-end security between API Client and API Gateway. Any communication between API Client and API Gateway must always be protected by use of TLS. JOSE standards must be utilised for achieving MLS of individual messages. Please see section **Error! Reference source not found.** to understand JOSE concepts.

2.2.2 Integrity/authenticity

In information security, integrity is about "*to protect the accuracy and completeness of information*''. You can detect any unauthorized modifications of the message exchanged by some parties involved in the communication. The security techniques that you can apply are similar to those for the confidentiality property.
As a measure, TLS is the chosen approach for transport-level security, because it has a way of detecting data modification. In fact, it sends a message-authentication code in each message, which can be verified by the receiving party to be sure that data has not been modified while in transit. For message authenticity the payload of the message will be signed at application level.

2.2.3 Availability

Availability "*is a property or characteristic. Something is available if it is accessible and usable when an authorized entity demands access*". High availability systems aim to remain available at all times, preventing service disruptions due to power outages, hardware failures, and system upgrades but also malicious attacks. Especially on a public API, these attacks can vary from an attacker planting malware in the system to a highly organized distributed denial of service (DDoS) attack. DDoS attacks are hard to eliminate fully, but with a careful design their impact can be minimized.
In most cases, DDoS attacks must be detected at the network perimeter level—so, the application code doesn't need to worry too much. But vulnerabilities in the application code can be exploited to bring a system down.

2.2.4 Authentication/Authorisation

Authentication can be performed at two levels:

1. End user level authentication, also referred to as entity authentication, to be sure about the identity of the user
2. Application, to be sure that the connecting client application is trusted.

Each Mobile Money Platform will have its own mechanisms for authentication and the API is designed in a way that it is agnostic to the authentication method used by the Mobile Money Platform. By doing so the API can be used with different authentication methods described in this document in section 2.3 as well as existing methods already implemented within the existing Mobile Money Platforms.

2.3 Security models

The GSMA Mobile Money API will follow a tiered security model approach to become viable for different API Clients and Mobile Money platform providers. The tiered approach will allow the clients and platform providers to decide the best security model based on their expertise, convenience and regulatory requirements. The different tiered models are:

1. API Client Authentication using HTTP Basic Authentication method over SSL/TLS connection

2. API Client Authentication and Authorisation using OAuth 2.0 Client Credentials Grant type over SSL/TLS connection

The next subsections describe the details about these security models.

2.3.1 API Client Authentication using basic authentication

This is the minimum level of security mechanism to be supported by Mobile Money platform provider implementing GSMA Mobile Money API. It consists of authenticating API client using the standard HTTP Basic Authentication method over secure SSL/TLS connection - RFC7617 [21].
Client credentials are pre-shared as per Mobile Money platform provider's policy with API Clients. Each Mobile Money API call needs to have HTTP Basic Authentication header built from client username and password as Base64 encoded string. The API Client can follow the below process for generating the basic authorisation header:

1. Construct the user-pass by concatenating the user-id, a single colon (":") character, and the password. For example: if the user name is "Aladdin" and password is "open_sesame". The concatenated string will be "Aladdin:open_sesame"
2. Base 64 encode the concatenated string to get an encoded string. For example: encString = Base64Encode("Aladdin:open_sesame")
3. Pass the encoded string in Authorisation header as follows: Authorisation: Basic QWxhZGRpbjpvcGVuX3Nlc2FtZQ==

The API Gateway will be responsible for basic identity check of API Client as described in section 2.4.2.

2.3.2 API Client Authentication using OAuth 2.0 Client Credentials grant type

The GSMA Mobile Money API will utilise OAuth 2.0 authorisation framework (RFC 6749 [22]) for API Client authentication and authorisation. OAuth 2.0 is a standard way of allowing a third-party application (API Client) to obtain limited access to an HTTP service i.e., protected resources. The generic OAuth 2.0 flow is as follows:

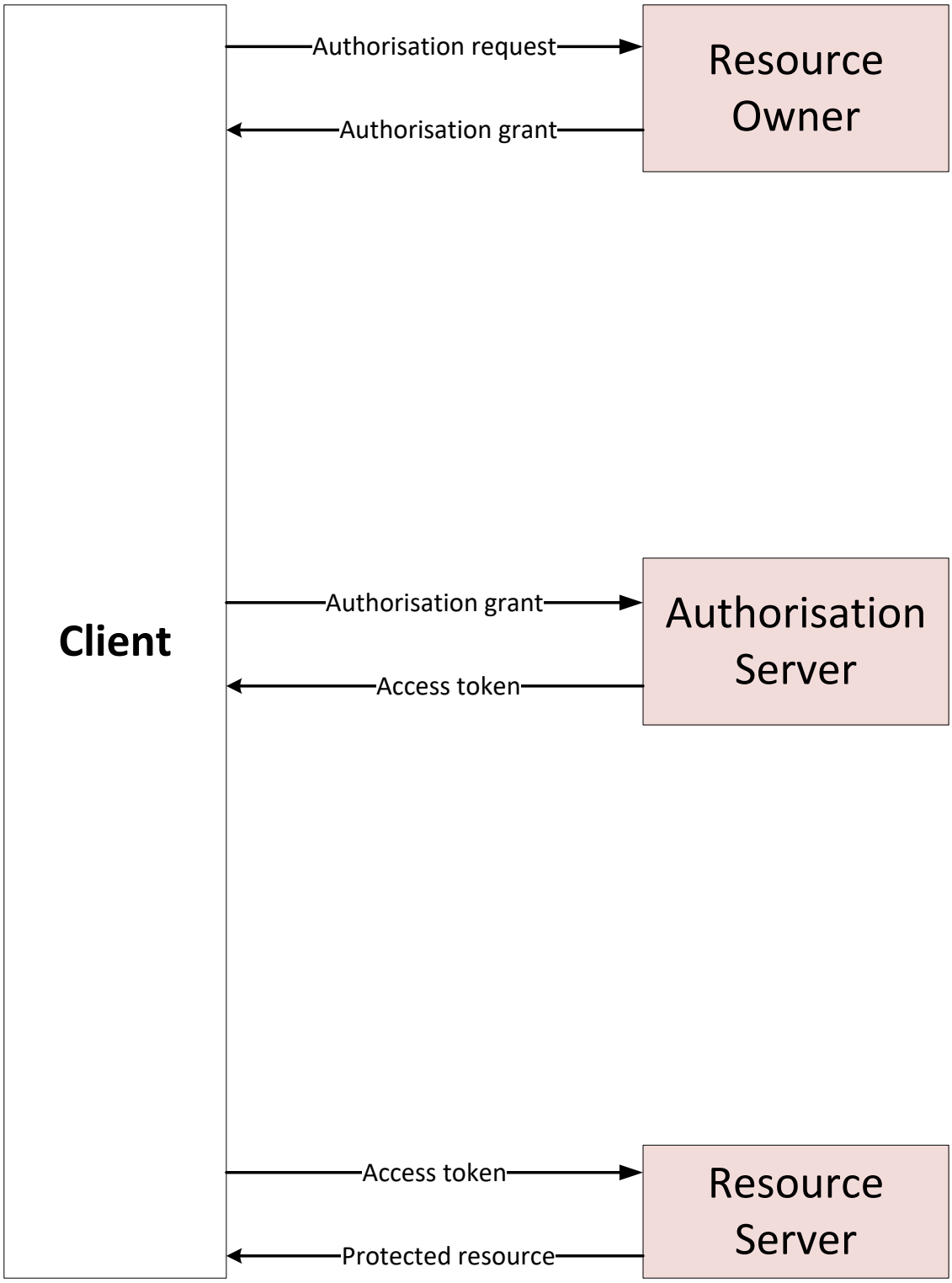

Figure 3: OAuth 2.0 standard flow

2.3.2.1 Issuance of Access Token using OAuth 2.0 Client Credentials grant type

The API Gateway will be responsible for exposing an additional token endpoint over SSL/TLS connection as defined in OAuth 2.0 specifications (RFC 6749 [22]). The API Client requests an

access token using only its client credentials as per client credentials grant type OAuth flow. These credentials are pre-shared as per Mobile Money platform provider's policy with API Clients. The client credentials grant type must only be used by protected and confidential clients. The client credentials flow is illustrated below:

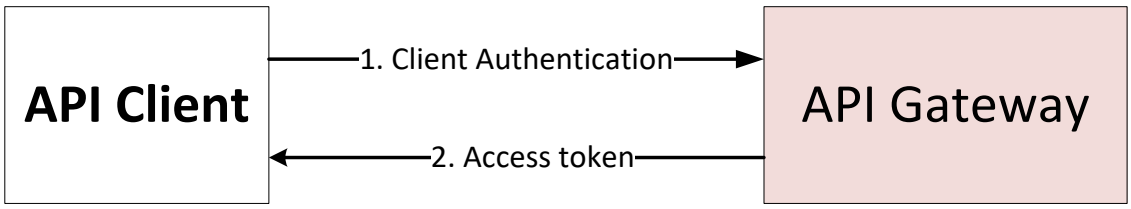

Figure 4: OAuth 2.0 client credentials flow

1. The API Client requests the access token from the token endpoint passing base64 encoded client credentials in basic authorisation header.
2. The API Gateway will be responsible for performing basic identity check of API Client as described in section 2.4.2. If valid, API Gateway interacts with its authorisation server to issue an access token response containing the access token, expiry time and optional scope values.

For example, the API Client makes the following HTTP request to API Gateway using secure SSL/TLS:

```
POST /token HTTP/1.1
Host: server.example.com
Authorization: Basic czZCaGRSa3F0MzpnWDFmQmF0M2JW
Content-Type: application/x-www-form-urlencoded
X-API-Key: czZCaGRSa3F0MzpnWDFmQmF0M2JW88jw66

grant_type=client_credentials
```

On successful authentication of API Client, the API Gateway responds with an access token response. For example:

```
HTTP/1.1 200 OK
Content-Type: application/json;charset=UTF-8
Cache-Control: no-store
Pragma: no-cache

{
       "access_token":"2YotnFZFEjr1zCsicMWpAA",
       "token_type":"example",
       "expires_in":3600
}
```

2.3.2.2 Usage of Access Token

The API Client will be responsible for passing the access token received in section 2.3.2.1 to every API call as a bearer token in the "Authorization" request header field as per RFC 6750 [23]. The API Gateway will be responsible for validating the token with its authorisation server, and if valid, check that the client is allowed to invoke the protected resource based on the access token scope value. It should return appropriate HTTP response codes in case of invalid access token (expired or revoked) or invalid scope.

For example:

```
GET /resource HTTP/1.1
Host: server.example.com
Authorization: Bearer mF_9.B5f-4.1JqM
X-API-Key: czZCaGRSa3F0MzpnWDFmQmF0M2JW88jw66
```

2.4 Common security mechanisms

The common security mechanisms that can be applied to the above two security models are:

1. Server-side TLS authentication
2. API Client basic identity check
3. Basic data integrity and authenticity check
4. JOSE standards for message encryption and validation
    - Message encryption/decryption using JWE
    - Message signature validation check using JWS
5. API Client certificate based authentication
6. Protection of sensitive request parameters – query parameters and path variables

The next subsections describe the details about these mechanisms. Some of these security mechanism allows the API requests and responses to be signed both by the API Client and API Gateway. The authentication and authenticity information is implemented using Javascript Object Signing and Encryption (JOSE) technologies as described in section **Error! Reference source not found.**. Each API request can optionally go through some of these checks when it arrives at API Gateway. If any of these checks fails, the request must be rejected with an error code in the response.

2.4.1 Server-side TLS authentication

Server-side TLS authentication will guarantee confidentiality and allow for detection of modification of the message (integrity) on the transport level. Server-side authentication takes place when API Gateway provides its public certificate for authentication to the API Client. The SSL layer authenticates both peers during the connection handshake. The API Client will need to pin the server certificate to be able to perform authentication of the server.

2.4.2 API Client basic identity check based on API key

The API Client will have pre-shared key with unique identifiers. The keys are shared as per Mobile Money platform provider's policy with API Clients. One of the ways of sharing the API key will be through the API Gateway developer portal. Initial identity of the API Client is confirmed by providing this identifier in a custom request header. If the combination of API key and client credentials/OAuth access token is not correct, the request must be rejected with an error code in the response. The custom request header used by API client to pass the key will be "X-API-Key".
It should be possible to revoke this key to stop a rogue API Client from accessing the API Gateway.

2.4.3 Basic data integrity and authenticity check

It is not mandatory for a Mobile Money platform provider to implement JOSE technology stack for achieving data integrity and authenticity. An alternate approach to achieve basic data integrity, detection of timing issues and authenticity checks is by using the following request

headers. The API Client must calculate these values and set it in the corresponding headers. These headers are optional and should only be used if JOSE is not used in a specific implementation. The different request headers are listed in **Error! Reference source not found.**.

HTTP Header Name	Description
X-Content-Hash	Custom request header - SHA-256 hex digest of the request content (encrypted or plain)
Content-Length	Length of request content - Requests having too long or non-matching length are rejected
Date	The date and time that the message was sent in HTTP-date format including the time zone. One of the policy can be to reject the requests having time deviation of more than ‘x’ minutes. It is the responsibility of API Gateway to normalize the time to server’s time zone for calculation purpose.

Table 3: Request headers for basic data integrity and authenticity check

2.4.4 JOSE standards for message validation and encryption

JOSE is a set of IETF standards to enable cryptographic protection of JSON objects, but also others type of objects, in fact, JOSE provides a general approach to signing and encryption of any content. However, it is deliberately built on JSON and base64 encoding, RFC 4648 [7] , to be easily usable in web applications.

The standards related to JOSE are listed in **Error! Reference source not found.**.

Standard	Description	How the standard is used on interface 1	RFC Reference
JSON Web Signature (JWS)	JSON objects with digital signatures or Message Authentication Codes (MAC)	JWS is used to sign the payload of the message being transmitted	RFC7515 [1]
JSON Web Encryption (JWE)	Encrypted JSON objects	JWE is used to encrypt the payload of the message	RFC7516 [2]
JSON Web Keys (JWK)	Public and private keys (or sets of keys) represented as JSON objects	Is used to exchange the public key used to sign the message payload with JWS	RFC7517 [3]
JSON Web Algorithms (JWA)	Authorizing a party to interact with a system in a prescribed manner	JWA is used to specify which algorithm is used for JWS and JWE	RFC7518 [4]
JSON Web Token (JWT)	Is a compact, URL-safe means of representing claims to be transferred between two parties Describes representation of claims encoded in JSON and protected by JWS or JWE	JWT is currently not used on interface 1 but can be used to transport OAuth tokens in future implementations. It is possible to encrypt and sign a JWT with JWE and JWS.	RFC7519 [5]

Table 4: Standards related to JOSE

JOSE has similar function to the XML Signature and XML Encryption standards, to provide message-level protection of message confidentiality, authenticity and integrity. Examples of protecting content using JSON Object Signing and Encryption can be found in RFC7520 [6].

2.4.4.1 Message encryption/decryption using JWE

Asymmetric encryption is used to exchange a symmetric CEK (Content Encryption Key). This CEK is encrypted with the public key of the receiving party to ensure that only the receiving party will be able to decrypt the CEK. From this moment on both parties are in the possession of the CEK. This Content Encryption Key can be used for the remainder of the session using a symmetric algorithm.

2.4.4.2 Message signature validation using JWS

The payload of the messages sent to the GSMA Mobile Money API will need to be signed by the private key belonging to the certificate of the API Client which must be enrolled within the API Gateway of the Mobile Money platform. This ensures integrity of the messages exchanged from the API Client to the API Gateway.

2.4.5 Protection of sensitive request parameters – Query/URI Path variables

It is important to protect sensitive request parameters passed to a GET resource. These parameters can be passed either as query parameters or URI path variables.

For example:

1. MSISDN/{value}
2. /accounts/{accountIdentifier1}@{value1}${accountIdentifier2}@{value2}${accountIdentifier3}@{value3}

The following strategy can be used by API Client to protect these parameters:

1. API Client encrypts URI path variables using the pre-shared API key with pre-shared symmetric encryption algorithm. The API key and algorithm details are pre-shared during the provisioning of API Client.
2. Use a request object in a POST that can be either signed (JWS) or encrypted (JWE) using standard JOSE framework described in section **Error! Reference source not found.**.

2.5 Certificate/Key management

2.5.1 Enrolment

To enrol API Clients within the API Gateway of the GSMA Mobile Money API the certificate belonging to the API Client should be enrolled within the API Gateway. This can either be performed by operating a PKI and issuing the certificates from the API Gateway or by creating a trust store of certificates that are issued by a public CA.

In both cases it is important that the certificate is linked to the API Client and known within the API Gateway.

2.5.2 Certificate revocation management

The certificates of API clients enrolled in the API Gateway should be maintained and if the API Client is no longer trusted than the certificate should be blacklisted inside the API Gateway (in the case issued by a public CA) or placed on a Certificate Revocation List (CRL) in case the certificate was issued by a CA for the Mobile Money Platform.

When setting up a connection the certificate from the API Client should always be validated, including validation of the chain, and may not be present on the CRL.
The API Client should validate the certificate of the API Gateway, including validation of the chain, and may not be present on the CRL.

2.6 Algorithms selection

2.6.1 TLS

For the Server side TLS at transport level, as described in section **Error! Reference source not found.**, the minimum TLS version to use is TLS 1.2 and only a subset of the following cipher suites [13] shall be supported from server side:

- TLS_ECDHE_ECDSA_WITH_AES_256_GCM_SHA384
- TLS_ECDHE_ECDSA_WITH_AES_128_GCM_SHA256
- TLS_ECDHE_RSA_WITH_AES_256_GCM_SHA384
- TLS_ECDHE_RSA_WITH_AES_128_GCM_SHA256
- TLS_ECDH_ECDSA_WITH_AES_256_GCM_SHA384
- TLS_ECDH_ECDSA_WITH_AES_128_GCM_SHA256
- TLS_ECDH_RSA_WITH_AES_256_GCM_SHA384
- TLS_ECDH_RSA_WITH_AES_128_GCM_SHA256
- TLS_RSA_WITH_AES_256_GCM_SHA384
- TLS_RSA_WITH_AES_128_GCM_SHA256

A specific cipher suite shall not be hard coded in the configuration. Instead, the protocol must be allowed to negotiate the highest version automatically [14].

2.6.2 JOSE

For JOSE the algorithms are defined in JWA as specified in RFC 7518 [4].

2.6.2.1 JWE

For JWE one of the following algorithms must be applied to exchange the CEK:

- Key Agreement with Elliptic Curve Diffie-Hellman Ephemeral Static, as described in section 4.6 from [4]:
    - ECDH-ES+A256KW: ECDH-ES using Concat KDF and CEK wrapped with "A256KW"
    - ECDH-ES+A192KW: ECDH-ES using Concat KDF and CEK wrapped with "A192KW"
    - ECDH-ES+A128KW: ECDH-ES using Concat KDF and CEK wrapped with "A128KW"
- Key Encryption with AES GCM, as described in section 4.7 from [4]:

    - A256GCMKW: Key wrapping with AES GCM using 256-bit key
    - A192GCMKW: Key wrapping with AES GCM using 192-bit key
    - A128GCMKW: Key wrapping with AES GCM using 128-bit key
- Key Encryption with RSAES OAEP, as described in section 4.3 from [4]:
    - RSA-OAEP-256: RSAES OAEP using SHA-256 and MGF1 with SHA-256
    - RSA-OAEP: RSAES OAEP using default parameters
- Key Encryption with RSAES-PKCS1-v1_5, as described in section 4.2 from [4]:
    - RSA1_5: RSAES-PKCS1-v1_5

For the CEK one of the following algorithms must be applied:

- AES_CBC_HMAC_SHA2 Algorithms, as described in section 5.2 from [4]:
    - A256CBC-HS512: AES_256_CBC_HMAC_SHA_512
    - A192CBC-HS384: AES_192_CBC_HMAC_SHA_384
    - A128CBC-HS256: AES_128_CBC_HMAC_SHA_256
- Content Encryption with AES GCM, as described in section 5.3 from [4]:
    - A256GCM: AES GCM using 256-bit key
    - A192GCM: AES GCM using 192-bit key
    - A128GCM: AES GCM using 128-bit key

2.6.2.2 JWS

For JWE one of the following algorithms must be applied:

- Digital Signature with ECDSA, as described in section 3.4 from [4]:
    - ES512: ECDSA using P-512 and SHA-512
    - ES384: ECDSA using P-384 and SHA-384
    - ES256: ECDSA using P-256 and SHA-256
- Digital Signature with RSASSA-PSS , as described in section 3.5 from [4]:
    - PS512: RSASSA-PSS using SHA-512 and MGF1 with SHA-512
    - PS384: RSASSA-PSS using SHA-384 and MGF1 with SHA-384
    - PS256: RSASSA-PSS using SHA-256 and MGF1 with SHA-256
- Digital Signature with RSASSA-PKCS1-v1_5, as described in section 3.3 from [4]:
    - RS512: RSASSA-PKCS1-v1_5 using SHA-512
    - RS384: RSASSA-PKCS1-v1_5 using SHA-384
    - RS256: RSASSA-PKCS1-v1_5 using SHA-256

2.7 Summary of security guidelines

It is possible to combine security model described in section 2.3 and security mechanisms described in section 2.4 to come up with preferred security options. Some of the recommended options are:

Development level: Testing of API connections; Low-value connections (e.g. single customer smartphone)
Standard level: Low-value connections; and Medium value (e.g. single business / merchant / agent)
Enhanced level: Medium and High value (e.g. mobile money operator / IMT provider / bank). This option provides the same level of protection that can be achieved by using a dedicated VPN tunnel between API Client and API Gateway. It is recommended to use this option as an alternative to a more expensive VPN tunnel option.

Security option	Security method	Description	Server side TLS	API Client basic identity check based on API key	Basic Data Integrity and Authenticity Check	JOSE - JWE & JWS	API Client certificate based authentication
Development Level	Basic Auth + HTTPS	API Client Authentication	**X**	**X**	**X**		
Standard Level	OAuth2 + HTTPS	API Client Authentication & Authorisation with protection from message tamper and MITM attack	**X**	**X**	**X**		
Enhanced Level	OAuth 2 + HTTPS	API Client Authentication & Authorisation with advanced message level protection (message signing and encryption)	**X**	**X**		**X**	**X**

Table 5: Summary of security guidelines

Please note that Mobile Money platform provider should implement '**Development Level**' security option defined above as a bare minimum viable option. It should be possible for the providers to select any combination of security methods and mechanisms above the development level to meet the regional security requirements.

2.7.1 Comparison between Basic Auth and OAuth 2.0 client credentials flow

The table below provides a comparison between basic authentication and OAuth 2.0 client credentials flow:

Basic authentication	OAuth 2.0 client credentials flow
Requires passing of encoded API Client credentials (client_id, client_secret) in every API request in “Authorization” header.	Encoded API Client credentials (client_id, client_secret) are passed once in an authentication API to get back an access token.
In a man-in-the-middle SSL exploit, the credentials of API Client are left wide open. It makes it extremely difficult to diagnose the attack vector that is compromising logins.	It is possible to revoke the access token in the event of man-in-the-middle SSL exploit or if an API Client is generating excessive or illicit traffic.
There is no token management capability. This deficiency makes it nearly impossible to limit access to secured resources without potentially having to disable the client’s credentials completely.	Access token can be issued with well-defined scopes that can allow access to protected resources. It should be possible to package related APIs as products and assign a scope for each product.
	Possible to mitigate the risk of replay attacks by selecting appropriate values for time to live on access tokens.
Easier to implement and no inherent requirement for a separate authorisation server to manage lifecycle of access tokens.	Slightly complex to implement and ideally requires a separate OAuth 2.0 compliant authorisation server to manage lifecycle of access tokens.
	The investment in OAuth 2.0 compliant authorisation server can be reused in implementing end user authentication framework.

Table 6: Comparison between Basic authentication and OAuth 2.0 client credentials flow

In summary, while the OAuth 2.0 “client credentials” grant type is a more complex interaction than Basic authentication, the implementation of access tokens is worth it. Managing an API program without access tokens can provide you with less control, and there is zero chance of implementing an access token strategy with Basic authentication. As long as you stick to forcing SSL usage, either option is secure, but OAuth 2.0 “client credentials” grant type should give you a better level of control.

3 End User Authentication – Security Design

3.1 Solution overview

The previous section focussed on API Client authentication models and common security mechanisms applicable for protecting API Client and API Gateway. The focus of this section is to provide implementation guidelines on end user authentication using different security models. Some of the scenarios where the security models can be applied are:

1. Authentication and identification of end user (debit party/credit party) to API gateway/Mobile Money platform. For example: As part of initial login process, API Gateway can authenticate the user using Authorisation Code flow[2] and API Client in turn retrieving the access token from API Gateway. The API Client can subsequently pass the access token in API calls to API Gateway for validation purpose.

2. Authorisation consent from a debit party/account holder for a financial transaction. For example: In the case of send money, cash out, buy goods; either API Gateway/API Client can authenticate the debit party using 3rd party OIDC compliant IDP and retrieve the consent proof (access token) and passing the access token to API Gateway in the API call.

3. End user consent to share their MSISDN and in-turn identifying their wallet account. For example: In case of an ecommerce checkout, the merchant server can authenticate the user using a 3rd party IDP and acquire consent to share their MSISDN. The consent proof (access token) can then be passed to API Gateway who can validate the token and retrieve the MSISDN to identify the wallet account.

4. Third party developer has developed an app to enable customers to send money. It should not be possible for customers to enter their credentials (MSISDN + PIN) into the app and pass it in Mobile Money API. Instead, the API Gateway should prompt the user to authenticate using Mobile Money platform credential mechanism (MSISDN + PIN) and if successfully authenticated, API Gateway issues an access token to the app. The app can supply the access token in Mobile Money API to API Gateway.

The security design proposes the following security models for authenticating end users:

1. End user authentication by API Gateway using OAuth 2.0/OIDC Authorisation Code Flow

2. Delegated end user (debit party) authorisation using 3rd party OIDC compliant IDP

3. End user authentication using username and PIN

3.1.1 Overview of OpenID Connect protocol

It is recommended that any 3rd party IDP used for authorising users should be OIDC compliant. Some of the popular OIDC compliant IDPs are GSMA's Mobile Connect, Facebook Connect and Google IDP etc.

[2] Also known as 3 legged OAuth flow

OpenID Connect (OIDC) [27] is an identity layer on top of OAuth 2.0 [22] that provides an authentication context for the end-user in the form of Who, When, How etc. in a JWT based claims set [ID Token].

The key functionality provided are:

- Pseudonymous Identity (claims assertion) /Authentication of end-user [ID Token]
- JSON/REST-like API for authentication and basic profile sharing [UserInfo]

OpenID Connect provides an additional token [ID Token] along with the OAuth 2.0 access_token. The ID Token is represented as a JWT and contains a claim set related to the authentication context of the subject. The JWT can be a plaintext JWT or cryptographically protected JWT – represented as signed JWT using JWS [JSON Web Signature] or as encrypted using JWE [JSON Web Encryption].
The security design recommends use of the OIDC Authorisation Code flow for the following reasons:

- Tokens not revealed to the User Agent
- RP must be authenticated
    - client_secret is used in Authorisation Code flow to retrieve access and ID tokens
- Usage of refresh token possible

3.2 Security models

3.2.1 End user authentication by API Gateway

The GSMA Mobile Money API will utilise industry standard OAuth 2.0/OIDC authorisation framework for end user authentication. Please note that the authorisation server can be embedded inside API Gateway depending on the implementation of API Gateway or it can be a separate authorisation server hosted by a 3rd party. The authorisation server must be either OIDC or OAuth 2.0 compliant.
It will utilise Authorisation Code flow/three legged OAuth flow for authenticating end users. Please see section 1.3 for definition of Authorisation Code flow. This flow is considered to be highly secure as Mobile Money credentials of end users are never requested directly by API Client.

Some of the advantages of using OAuth 2.0/OIDC authorisation framework are:

1. Use of industry standard protocols for authenticating users thereby avoiding build of bespoke solutions.
2. Use of OIDC compliant IDP providers means support for wide array of advanced authentication mechanisms including PIN and Biometrics[3].
3. Single integration model for API Client to authenticate end users.
4. More secured as end user credentials are never captured in API Client assets[4] directly.
5. The use of access token allows time bound/one time access, if required.

A high-level component view of various actors and flow of information is illustrated below:

[3] Finger scan or Facial recognition or Iris scan
[4] Website or apps

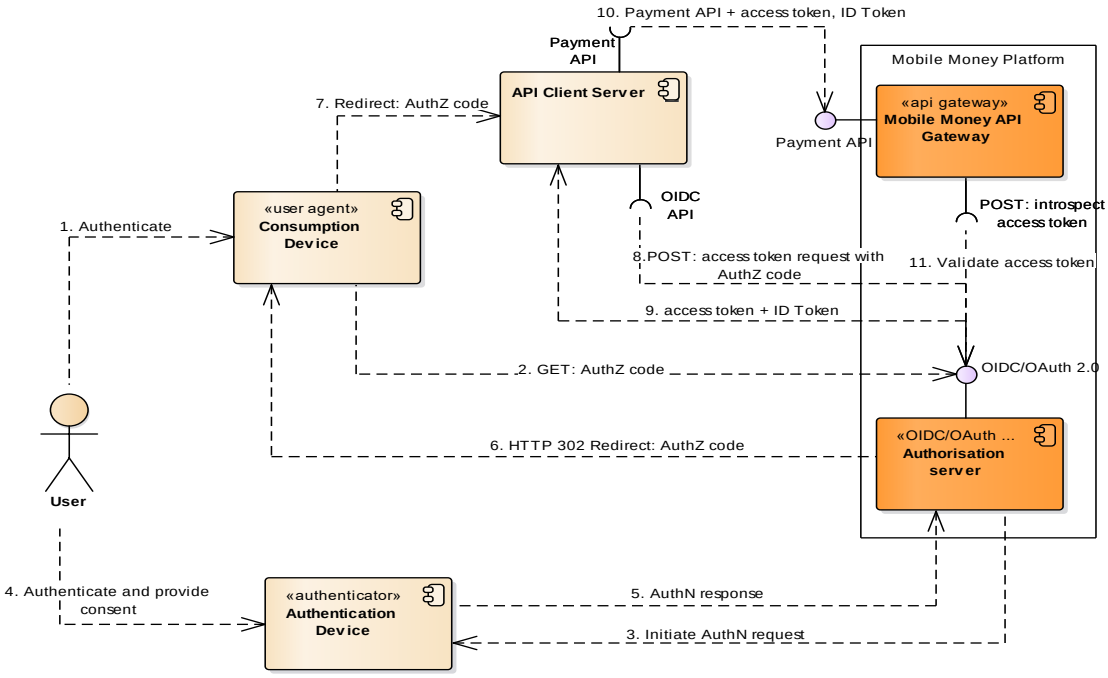

Figure 5: End user authentication by API Gateway

The process for authenticating end user and retrieving access token can be broken down into following steps:

1. End user initiates authentication request on the consumption device's user agent.
2. The user agent sends an authorisation request to authorisation server for authenticating end user passing client_id, redirect_uri, state and other parameters in the request. See section 3.2.1.1 for details.
3. Authorisation server initiates end user authentication process as per Mobile Money platform's authentication mechanism. The authorisation server can authenticate the end user by presenting an authentication page either in consumption device or separate authentication device. The actual implementation is left to the Mobile Money platform provider.
4. User is prompted to provide credentials either in consumption device or separate authentication device.
5. The authentication device generates an authentication response and returns to the authorisation server.
6. Authorisation server validates authentication response and returns a temporary authorisation code to RP server indirectly as a redirect through user agent. See section 3.2.1.2 for details.
7. RP server receives the authorisation code in the redirect URL. It extracts the authorisation code from the redirect URL's query parameter.
8. RP server exchanges the authorisation code to retrieve access token and optional ID Token with authorisation server. RP server will provide its client credentials in token API request to retrieve the tokens. See section 3.2.1.3 for details.
   Please note that if the authorisation server is not OIDC compliant, then it will only return access token. The advantage of using ID Token is to allow the RP to retrieve additional identity claims like MSISDN etc.
9. Authorisation server validates the authorisation code and client credentials, generates new access token and ID Token and returns to RP server.
10. RP server passes end user's access token to API Gateway in API requests as a custom header value. See section 3.2.1.4 for details.

11. API Gateway validates the access token with authorisation server before processing the API request.

3.2.1.1 Authorisation request for authenticating end user

API Client's user agent will send authorization request to the authorisation server's 'authorisation endpoint' as described in section 3.1.2 of OIDC [27], using HTTP GET or POST. Communication to the authorisation server endpoint MUST use SSL/TLS. The request parameters are added using query string serialization. The prompt parameter in the request must be "login".

Sample Request:

```
POST /authorize HTTP/1.1
Host: server.example.com
Content-Type: application/x-www-form-urlencoded

response_type=code&
client_id=s6BhdRkqt3
&redirect_uri=https%3A%2F%2Fclient.example.org%2Fcb
&scope=openid
&state=af0ifjsldkj
&nonce=n-0S6_WzA2Mj
&prompt=login
&login_hint=<MSISDN>
```

The authorisation server validates authorisation request and returns a HTML payload for authenticating the user. The actual authentication mechanism (MSISDN + PIN or Biometrics[5] or something else) is dependent on the downstream Mobile Money platform. It should also be possible for the authorisation server to perform out of band authentication using separate authenticators[6]. The authenticators can also act as consent device for displaying authentication prompt to the user. The actual implementation of authentication mechanism adopted by authorisation server is out of scope from this document.

3.2.1.2 Authorisation response

Authorisation server will generate authorization code after authenticating the end user. It will return the authorization code using redirect to the RP server[7] at the redirect_uri.

Sample Response:

```
HTTP/1.1 302 Found
Location:https://server.sp.com/authorized?Code=AsdsdsMKDsd&stat
e=af0ifjsldkj
```

3.2.1.3 Issuance of tokens using Authorisation Code flow

RP server[8] makes a token request by presenting its authorisation code to the token endpoint exposed by authorisation server. The grant type value must be "authorization_code", as described in section 4.1.3 of OAuth 2.0 [22].

[5] Finger scan or Facial recognition or Iris scan
[6] USSD authenticator, SIM Applet authenticator, Smartphone App Authenticator
[7] API Client
[8] API Client

The RP server sends the parameters to the token endpoint using the HTTP POST method and the form serialization, as described in section 4.1.3 of OAuth 2.0 [22]. Communication to the authorisation server endpoint MUST use SSL/TLS.

The Authorisation Code flow is illustrated below:

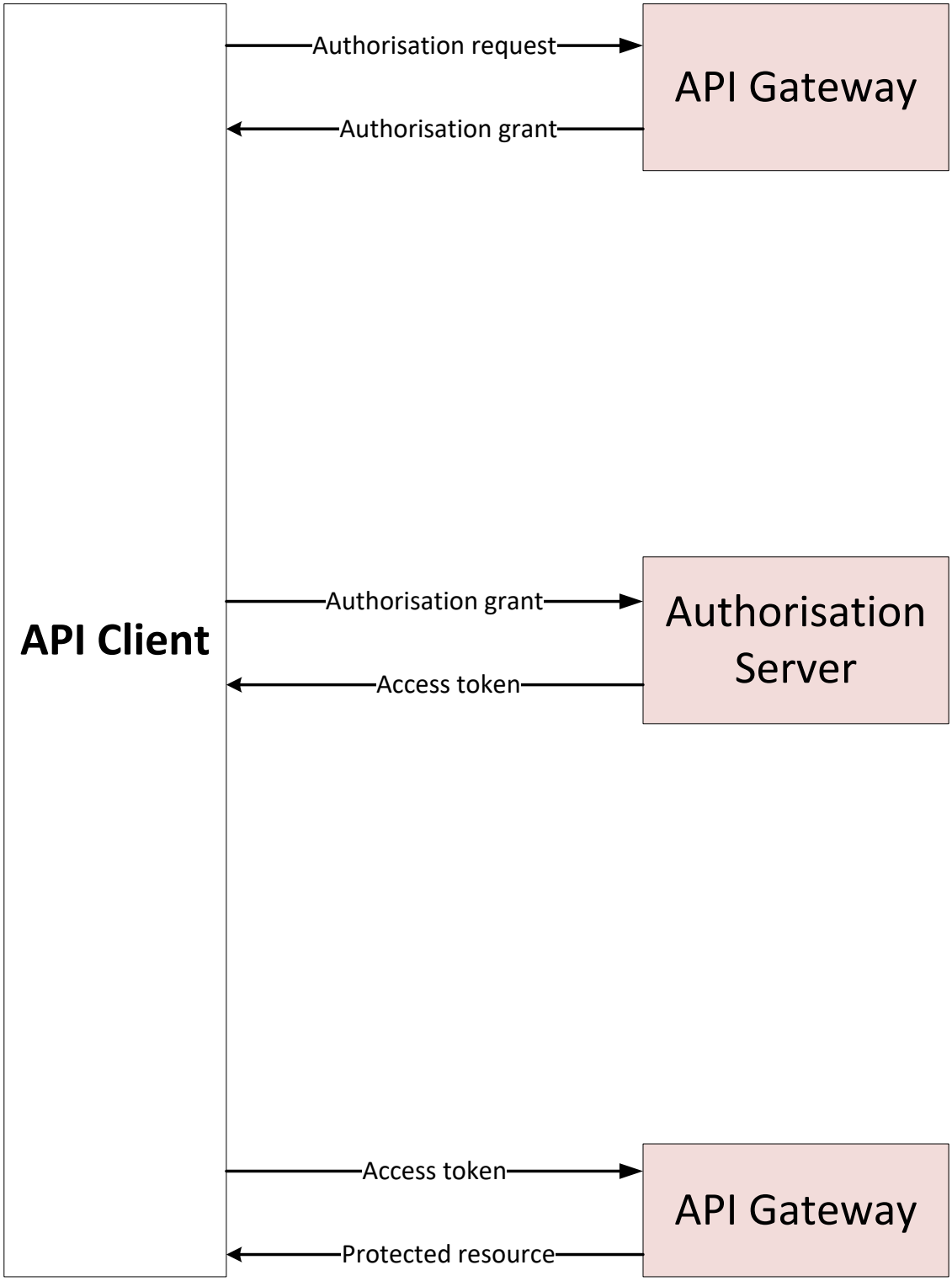

Figure 6: OAuth 2.0 Authorisation Code flow

1. The API Client requests the access token from the token endpoint of authorisation server passing base64 encoded client credentials in basic authorisation header. The request parameters includes grant type value, authorisation code received in section 3.2.1.2 and redirect URI value. These parameters are passed “x-www-form-urlencoded” values.’
2. The authorisation server validates client credentials of RP server and if valid, returns an access token response containing the access token, refresh token, expiry time and optional ID Token,.

Sample token request:

```
POST /token HTTP/1.1
Host: server.example.com
Content-Type: application/x-www-form-urlencoded
Authorization: Basic czZCaGRSa3F0MzpnWDFmQmF0M2JW

grant_type=authorization_code&code=SplxlOBeZQQYbYS6WxSbIA
    &redirect_uri=https%3A%2F%2Fclient.example.org%2Fcb
```

Sample successful access token response:

```
HTTP/1.1 200 OK
Content-Type: application/json
Cache-Control: no-store
Pragma: no-cache

{
      "access_token": "SlAV32hkKG",
      "token_type": "Bearer",
      "refresh_token": "8xLOxBtZp8",
      "expires_in": 3600,
      "id_token":
"eyJhbGciOiJSUzI1NiIsImtpZCI6IjFlOWdkazcifQ.ewogImlzc

yI6ICJodHRwOi8vc2VydmVyLmV4YW1wbGUuY29tIiwKICJzdWIiOiAiMjQ4Mjg5

NzYxMDAxIiwKICJhdWQiOiAiczZCaGRSa3F0MyIsCiAibm9uY2UiOiAibi0wUzZ

fV3pBMk1qIiwKICJleHAiOiAxMzExMjgxOTcwLAogImlhdCI6IDEzMTEyODA5Nz

AKfQ.ggW8hZ1EuVLuxNuuIJKX_V8a_OMXzR0EHR9R6jgdqrOOF4daGU96Sr_P6q
       Jp6IcmD3HP99Obi1PRs-cwh3LO-
p146waJ8IhehcwL7F09JdijmBqkvPeB2T9CJ
       NqeGpe-
gccMg4vfKjkM8FcGvnzZUN4_KSP0aAp1tOJ1zZwgjxqGByKHiOtX7Tpd

QyHE5lcMiKPXfEIQILVq0pc_E2DzL7emopWoaoZTF_m0_N0YzFC6g6EJbOEoRoS

K5hoDalrcvRYLSrQAZZKflyuVCyixEoV9GfNQC3_osjzw2PAithfubEEBLuVVk4
       XUVrWOLrLl0nx7RkKU8NXNHq-rvKMzqg"
 }
```

3.2.1.4 Usage of Access Token

The API Client will be responsible for passing user's access token received in section 3.2.1.3 to every API call as a custom header value. The API Gateway will be responsible for validating the token with its authorisation server, and if valid, allow the processing of the API request. It should return appropriate HTTP response codes in case of invalid access token (expired or revoked).

For example:

```
GET /resource HTTP/1.1
Host: server.example.com
Authorization: Bearer mF_9.B5f-4.1JqM
X-User-Bearer: czZCaGRSa3F0MzpnWDFmQmF0M2JW88jw66
```

3.2.2 Delegated end user authorisation

There are scenarios that requires debit party/account holder authentication to authorise a payment transaction. For example: In the case of Send Money, Cash Out, Buy Goods etc., it should be possible for the API Gateway to directly authenticate the debit party as described in section 3.2.1 or allow the API Client to use a 3rd party OIDC compliant IDP to authenticate the debit party and pass the access token as consent proof[9] to API Gateway in the API request. This allows the API Gateway to validate the access token using token introspection endpoint described in section 3.2.2.1. On successful validation, it should continue processing the payment request. The focus of this section is delegated authorisation of debit party/account holder using a 3rd party IDP.

Some of the advantages of delegated authorisation model are:

1. Use of industry standard protocols for authorising users thereby avoiding build of bespoke solutions.
2. Single integration model to support multiple 3rd party IDP providers.
3. User's credentials are never passed in API request thereby reducing risk and fraud.
4. The use of access token allows time bound/one time access, if required.
5. Allows API Gateway to verify the consent proof before proceeding with payment transaction. The consent proof can also provide audit trail as it contains exact timestamp of providing consent and mechanism used for authenticating the user.
6. The consent proof provides non-repudiation of payment transaction.
7. Use of OIDC compliant IDP providers means support for wide array of advanced authentication mechanisms including PIN and Biometrics[10].
8. Streamlined UX flow as the user is not required to authenticate separately with Mobile Money platform, resulting in fewer steps to complete a payment transaction.

A high-level component view of various actors and flow of information is illustrated below:

[9] access token
[10] Finger scan or Facial recognition or Iris scan

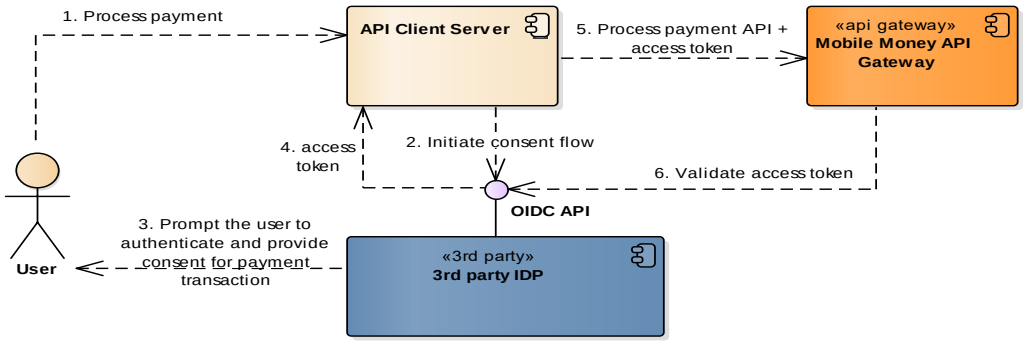

Figure 7: Delegated user authorisation

1. User initiates payment request (For ex: Send Money) with API Client.
2. API Client uses 3rd party IDP to authenticate the user and authorise the payment transaction.
3. 3rd party IDP prompts the user to authenticate and provide consent for the payment transaction.
4. On successful user authentication and consent, 3rd party IDP issues an access token and ID Token to API Client.
5. API Client invokes Mobile Money payment API passing the access token, ID Token and introspection endpoint URL of 3rd party IDP.
6. API Gateway validates the access token by invoking introspection endpoint URL. The introspection endpoint is a protected endpoint requiring API Gateway to pass its client credentials or bearer token when invoking this endpoint. 3rd party IDP returns meta-information of the access token if the token is still valid[11]. API Gateway can optionally introspect the ID Token to retrieve identity claims like MSISDN etc. It can also check the level of assurance[12] achieved by 3rd party IDP when authenticating the user before continuing with payment processing flow with downstream Mobile Money platform.

Please note that user has to be registered with 3rd party IDP in order to authenticate and provide consent. Also, API Gateway should have client relationship with 3rd party IDP in order to invoke the introspection endpoint.

3.2.2.1 OAuth 2.0 Token Introspection

The token introspection endpoint allows a resource server[13] to query OAuth 2.0 authorisation server to determine the active state of an access token and to retrieve meta-information about this token. This method can be used by RP[14] to convey information about the authorisation context of the token from the authorisation server to the protected resource. In the context of Mobile Money APIs as illustrated in section 3.2.2, API Client can pass user's access token and introspection endpoint URL of 3rd party IDP server to API Gateway allowing the gateway to validate the access token by invoking the introspection endpoint URL and passing the access token as "application/x-www-form-urlencoded" data. The successful response contains meta-information about the token.

[11] Not expired or revoked
[12] acr_value attribute in ID Token
[13] API Gateway
[14] API Client

The endpoint also requires some form of client authorization to access this endpoint. The calling client[15] can authenticate using the mechanisms described in section 2.3 of OAuth 2.0 [22] or by passing a separate OAuth2.0 access token as bearer token.

The following is a non-normative example request:

```
POST /introspect HTTP/1.1
Host: server.example.com
Accept: application/json
Content-Type: application/x-www-form-urlencoded
Authorization: Bearer 23410913-abewfq.123483

token=mF_9.B5f-4.1JqM&token_type_hint=access_token
```

3.2.2.2 Introspection request attributes

The protected resource[16] calls the introspection endpoint using an HTTP POST request with parameters sent as "application/x-www-form-urlencoded" data

Parameter	Required category in spec	Description
token	Mandatory	The string value of the token. • For access tokens, this is the "access_token" value returned from the token endpoint • For refresh tokens, this is the "refresh_token" value returned from the token endpoint
token_type_hint	Optional	A hint about the type of the token submitted for introspection. The possible values are: • “access_token” if the token is of type access token • “refresh_token” if the token is of type refresh token

Table 7: Introspection request attributes

3.2.2.3 Introspection response attributes

The authorisation server responds with a JSON object in “application/json” format with the following top-level members:

Attribute	Required category in spec	Description
scope	Optional	A JSON string containing a space-separated list of scopes associated with this token, in the format described in **Error! Reference source not found.** section 3.3
client_id	Optional	Client identifier for the RP that requested this token
username	Optional	User’s username
token_type	Optional	Type of token as defined in section 5.1 of OAuth 2.0 [22]

[15] API Gateway
[16] API Gateway

exp	Optional	The expiration time after which the access token MUST NOT be accepted for processing. The format is the number of seconds from 1970-01-01T0:0:0Z as measured in UTC until the date/time specified.
iat	Optional	The time of issue of access token. The format is the number of seconds from 1970-01-01T0:0:0Z as measured in UTC until the date/time specified.
nbf	Optional	Timestamp indicating when the access token is not to be used before. The format is the number of seconds from 1970-01-01T0:0:0Z as measured in UTC until the date/time specified.
sub	Optional	Subject identifier of the user (PCR)
aud	Optional	The intended audience for the access token. It is an array of case-sensitive strings. It MUST contain the client_id of the RP/Client, and MAY contains identifiers of other optional audiences. If there is one audience, the aud value MAY be a single case sensitive string OR an array of case sensitive strings with only one element. An implementation MUST support both scenarios.
iss	Optional	Issuer Identifier. It is a case-sensitive HTTPS based URL, with the host. It MAY contain the port and path element (Optional) but no query parameters.
jti	Optional	Access token string identifier

Table 8: Introspection response attributes

3.2.3 End user authentication using username and PIN

There is a legacy option in some of the existing Mobile Money platforms that allows API Client to capture user's credentials[17] directly in their own assets[18] and pass it to Mobile Money platform securely in the API payload. This is not a recommended option but is still provided in this document to support existing implementation requiring minimum changes.

Some of the drawbacks of this option are:

1. User's credentials are known to API Client resulting in increased fraud due to possibility of altering the credentials.
2. API Gateway and Mobile Money platform is unable to receive explicit consent from the user thereby potentially increased customer service complaints and financial liability.
3. API Client has to support multiple authentication models for different Mobile Money platform providers.

Some of the recommendations to support this option are:

[17] MSISDN and PIN
[18] Website or app

1. This option should only be used in scenarios where the user is directly controlled by Mobile Money Platform. For example: if the user is using Mobile Money platform's website or app directly and needs to authenticate.
2. User's MSISDN is passed in API and should be encrypted as defined in section 6.
3. The PIN is encrypted at source using pre-shared API key and symmetric encryption algorithm. The API key and algorithm details are shared during the provisioning of API Client.
4. Use of custom header in the API request to pass encrypted PIN.

4 API Best Practices

This chapter describes a collection of the common security practices that must be applied to RESTful API and API Gateway platform. For these common best practices, the following references have been used as a reference:

- OWASP REST Security Cheat Sheet [9].
- OWASP Top Ten [25]
- RESTful Service Best Practices [10]
- OWASP Cryptographic Storage Cheat Sheet [26]

Each section contains a table with a set of best practices encoded in the following way: {BP_Category_number]

4.1 Auditing/Monitoring

4.1.1 Logging

An important aspect of building RESTful services in a complex distributed application is to address logging functions, especially for the purpose of debugging production issues and investigating eventual points of failure. With good logging practices you can detect security issues. Keep in mind that PII (Personally Identifiable Information) [11] data should be handled with care avoiding the logging of these types of information.

Table 9 - Best Practices Logging [BP_LOG]

Code	Description
BP_LOG_1	A detailed consistent pattern should be applied to log messages across service logs. It is a good practice for a logging pattern to at least include the following: date and current time, logging level, the name of the thread, the simple logger name and the detailed message.
BP_LOG_2	It is important to obfuscate sensitive data. It is important to mask or obfuscate sensitive data in production logs to protect the risk of compromising confidential and critical PII information.
BP_LOG_3	Identifying the caller or the initiator as part of logs.
BP_LOG_4	Do not log payloads by default.

4.1.2 Monitoring/Reporting

Monitoring activities is useful to protect your application from some misuses or external attacks, but also to keep track, with the help of a BAM (Business Activity Monitoring), of KPIs (Key Performance Indicator) to verify the adherence to the SLA agreed with the stakeholders. API Gateway can be used to monitor, throttle, and control access to the API. The following can be done by a gateway or by the RESTful service:

- Monitor usage of the API and know what activity is good and what falls out of normal usage patterns and implement appropriate reporting functionality

- Throttle API usage so that a malicious user cannot take down an API endpoint (DOS attack) and have the ability to block a malicious IP address

Table 10 - Best Practices Monitoring [BP_MON]

Code	Description
BP_MON_1	Use a monitoring system which can collect data to evaluate and to control anomalous behavior, SLA and other statistics in the background. It is a good practice to collect logs in a SIEM (Security Information and Event Management), to discover some anomalous behavior and to detect some attack patterns. Collecting information in the right manner you could do this following activities: ▪ Identity, Audit and Authenticate Administrator and 3rd Party Access ▪ Control and Audit of all privileged users access ▪ Logging, monitoring user access ▪ Track and Monitor all Access ▪ Access Policy and reporting for Forensics and Investigations on incidents ▪ Continuous Security Training Awareness with Recording Message ▪ Remote Access Session Monitoring and Authentication to Servers ▪ Logging Access, Alert on Unauthorized Access to Sensitive systems ▪ Ports and Services Monitoring, Logging All Server and user activity ▪ Incident Response with Session Replay on Event logs
BP_MON_2	Should implement payload protection policy. Malicious injection in the payload are mitigated using techniques described in chapter 3.4. But the attackers can, for example, use recursive techniques to consume memory resources and other techniques that can compromise the availability of the services. In the optic of a layered approach, the JSON threat protection policies help to protect applications from such intrusions and possible damages. Some example of policy to define are: • Specifies the maximum number of elements allowed in an array. • Specifies the maximum allowed containment depth, where the containers are objects or arrays. • Specifies the maximum number of entries allowed in an object • Specifies the maximum string length allowed for a property name within an object. • Specifies the maximum length allowed for a string value. Some useful reference are MuleSoft documentation [15], Apigee documentation [16] and SAP HANA documentation [17].
BP_MON_3	Should implement protection policy to monitor the traffic against malicious behavior (eg. DOS Attack and spike arrest). • Quotas [18] and rate limits control the number of connections apps can make to the backend via the API • Spike arrest [19] capabilities protect against traffic spikes and denial-of-service attacks

Code	Description
	It is suggested to implement this feature using an API Gateway such as Apigee, SAP API Management.
BP_MON_4	It is suggested to provide real-time monitoring capability o provide real-time API health visibility. Define and implement a list of KPI to monitor the API health status and performance issue. Examples of KPI: 1. If you have more than 5 consecutive errors in the invocation of a service raise an alert 2. If the call for a function takes more than a fixed period ex. 4 second.

4.2 Communication

4.2.1 Transport

Switching between HTTP and HTTPS introduces security weaknesses and the best practice is to use TLS (HTTPS) by default for all the communication.

Table 11 - Best Practices Transport Communication [BP_TCOM]

Code	Description
BP_TCOM_1	Data in transit. The use of TLS should be mandated, particularly where credentials, updates, deletions, and any value transactions are performed. TLS version 1.2 [8] or newer must be utilized.

4.2.2 Data encryption

Encryption should always be used to transfer data or sensitive information between API Client and API Gateway.

Table 12 - Best Practices Transport Encryption [BP_TCRY]

Code	Description
BP_ TCRY_1	PII and sensitive information in general should be encrypted (i.e. JSON encryption [2]).
BP_ TCRY_2	Data at rest. It is necessary to prevent database bypass, which occurs when an attacker threatens to gain access to sensitive data by targeting operating system files and backup media. In this case he may avoid most database authentication and auditing mechanisms. The most common way of preventing this is encrypting the data-at-rest, i.e. whenever it is committed to memory. This has the added benefit of also protecting against improper decommission or theft of drives.

Code	Description
BP_ TCRY_3	Hash-based message authentication code (HMAC) should be used because it's the most secure. (Use SHA-2 and up, avoid SHA & MD5 because of vulnerabilities)
BP_TCRY_4	Any PII and sensitive request parameters passed as query parameters or path variables must be encrypted. Please see section 2.4.5 for more details.

4.2.3 Storage of cryptographic keys and credentials

Some of the best practices for cryptographic design and storage of cryptographic keys are summarized below:

Table 13 - Best Practices for storage of crypto keys [BP_CRPS]

Code	Description
BP_CPRS_1	All protocols and algorithms for authentication and secure communication should be well vetted by the cryptographic community.
BP_CPRS_2	Ensure certificates are properly validated against the hostnames/users i.e. whom they are meant for
BP_CPRS_3	Avoid using wildcard certificates unless there is a business need for it
BP_CPRS_4	Maintain a cryptographic standard to ensure that the developer community knows about the approved ciphersuits for network security protocols, algorithms, permitted use, crypto periods and Key Management
BP_CPRS_5	Store a one-way and salted value of passwords - Use PBKDF2, bcrypt or scrypt for password storage
BP_CPRS_6	Ensure that the cryptographic protection remains secure even if access controls fail - This rule supports the principle of defense in depth. Access controls (usernames, passwords, privileges, etc.) are one layer of protection. Storage encryption should add an additional layer of protection that will continue protecting the data even if an attacker subverts the database access control layer
BP_CPRS_7	Ensure that any secret key is protected from unauthorized access
BP_CPRS_8	Store unencrypted keys away from the encrypted data
BP_CPRS_9	Protect keys in a key vault
BP_CPRS_10	Document concrete procedures for managing keys through the lifecycle
BP_CPRS_11	Under PCI DSS requirement 3, you must protect cardholder data
BP_CPRS_12	Render PAN (Primary Account Number), at minimum, unreadable anywhere it is stored
BP_CPRS_13	Protect any keys used to secure cardholder data against disclosure and misuse. As the requirement name above indicates, we are required to securely store the encryption keys themselves. This will mean implementing strong access control, auditing and logging for your keys. The keys must be stored in a location which is both secure and "away" from the encrypted data. This means key data shouldn't be stored on web servers, database servers etc. Access to the keys must be restricted to the smallest amount of users possible. This group of users will ideally be users who are highly trusted and

Code	Description
	trained to perform Key Custodian duties. There will obviously be a requirement for system/service accounts to access the key data to perform encryption/decryption of data. The keys themselves shouldn't be stored in the clear but encrypted with a KEK (Key Encrypting Key). The KEK must not be stored in the same location as the encryption keys it is encrypting.

4.3 Identity Management

The API Client and/or end user should be authenticated and authorized prior to completing an access control decision. All access control decisions should be logged.

4.3.1 Authentication and session management

Authentication validates if you are the right person who can login to the software system.
A RESTful API should be stateless. This means that request authentication should not depend on cookies or sessions.

Table 14 - Best Practices Identity Management Authentication [BP_IMAU]

Code	Description
BP_IMAU_1	Session-based authentication must be used, either by establishing a session token via a POST or by using an API key as a POST body argument or as a cookie. Username, passwords, session tokens, and API keys must not appear in the URL.
BP_ IMAU_2	Protect session state. Most Web services and APIs are designed to be stateless, with a state blob being sent within a transaction. For a more secure design, consider using the API key to maintain client state if the API is using a server-side cache. It's a commonly used method in Web applications and provides additional security by preventing anti-replay. Replay is where attackers cut and paste a blob to become an authorized user. In order to be effective, include a time-limited encryption key that is measured against the API key, date and time, and incoming IP address
BP_ IMAU_3	All access control decisions shall be logged.
BP_ IMAU_4	Protect against replay attacks. A replay attack (also known as playback attack) is a form of network attack in which a valid data transmission is maliciously or fraudulently repeated or delayed. This is carried out either by the originator or by an adversary who intercepts the data and retransmits it, possibly as part of a masquerade attack by IP packet substitution (such as stream cipher attack). There are a lot of countermeasures that you can take in place that use a time limited encryption key, keyed against the session token or API key, date and time, and incoming IP address. For example, some mechanisms are: session tokens, one time passwords, MAC (Message Authentication Code) and time stamping.

Code	Description
	One common best practice mechanism, also used by OAuth is to have the following combination: • Nonce (number used once): It identifies each unique signed request, and prevent requests from being used more than once. This nonce value is included in the signature, so it cannot be changed by an attacker • Timestamp: We can add a timestamp value to each request .When a request comes with an old timestamp, the request will be rejected. From a security standpoint, the combination of the timestamp value and nonce string provide a perpetual unique value that cannot be used by an attacker. A useful reference is a SANS whitepaper: Four Attacks on OAuth - How to Secure OAuth Implementation [20].

4.3.2 Authorisation

Authorisation validates if you are the right person to have access to the resources.

Table 15 - Best Practices Identity Management Authorisation [BP_IMAZ]

Code	Description
BP_ IMAZ_1	Protect HTTP methods. RESTful API often use GET (read), POST (create), PUT (replace/update) and DELETE (to delete a record) methods. Not all of these are valid choices for every single resource collection, user, or action. Make sure the incoming HTTP method is valid for the session token/API key and associated resource collection, action, and record.
BP_ IMAZ_2	Whitelist allowable methods. For an entity the permitted operations should be defined. For example, a GET request might read the entity while PUT would update an existing entity, POST would create a new entity, and DELETE would delete an existing entity. It is important for the service to properly restrict the allowable verbs such that only the allowed verbs would work, while all others would return a proper response code (for example, a 403 Forbidden).
BP_ IMAZ_3	Protect privileged actions and sensitive resource collections. Not every user has a right to every web service. The session token or API key should be sent along as a cookie or body parameter to ensure that privileged collections or actions are properly protected from unauthorized use.
BP_ IMAZ_4	Protect against cross-site request forgery. For resources exposed by RESTful web services, it's important to make sure any PUT, POST, and DELETE request is protected from Cross Site Request Forgery. Typically one would use a token-based approach.

4.4 Validating RESTful services

When exposing RESTful service APIs, it is important to validate that the API behaves correctly.

4.4.1 Input validation

Table 16 - Best Practices Input Validation [BP_VALI]

Code	Description
BP_VALI_1	Use a secure parser for parsing the incoming messages.
BP_VALI_2	It is suggested the using of strongly type techniques for incoming data. Limit and define the permitted values for an input parameter.
BP_VALI_3	Validate incoming content-types. The service should never assume the *Content-Type*. When is not present in the header the server should reject the content with a *406 Not Acceptable response*.
BP_VALI_4	Validate response types. It is common for REST services to allow multiple response type (in this case: *application/json*) and the client specifies the preferred order of response types by the Accept header in the request. Do not accept the request if the content type is not one of the allowable types. Reject the request (ideally with a HTTP *406 Not Acceptable response*)
BP_VALI_5	Use some framework (e.g. Jersey) that enable validation constrains to be enforced automatically at request or response time. This kind of framework provide automatic validation after unmarshaling.
BP_VALI_6	To prevent abuse, it is standard practice to add some sort of rate limiting to an API. RFC 6585 introduced a HTTP status code *429 Too Many Requests* to accommodate this. However, it can be very useful to notify the consumer of their limits before they actually hit it.

4.4.2 Output encoding

Table 17 - Best Practices Output Validation [BP_VALO]

Code	Description
BP_VALO_1	Send security headers. To make sure the content of a given resources is interpreted correctly by the browser, the server should always send the *Content-Type* header. The server should also send an *X-Content-Type-Options: nosniff* to make sure the browser does not try to detect different *Content-Type* than what is actually sent (can lead to XSS).
BP_VALO_2	JSON encoding. A key concerns with JSON is preventing arbitrary JavaScript remote code execution within the browser. When inserting values into the browser DOM, strongly consider using *.value/.innerText/.textContent* rather than *.innerHTML* updates, as this protects against simple DOM XSS attacks.
BP_VALO_3	XML as JSON should never be built by string concatenation. It should always be constructed using an appropriate serializer. This should be useful to be sure that the content is parsable and does not contain injected elements.

4.4.3 Error handling

You have to take in consideration that unhandled exceptions could reveal, to an attacker, useful information about your API.

Table 18 - Best Practices Error Handling [BP_ERR]

Code	Description
BP_ERR_1	Utilize error codes. It is highly recommended that error codes are returned whenever an error is encountered. A cautionary note here is to not provide too much information (such that it would provide an adversary an advantage). Successful error codes/messages are a balance between enough information and security.

Annex A REST Security Standard Overview

This section provides a brief overview of open security standards defined by the Internet Engineering Task Force (IETF), the OpenID Foundation (OIDF), and other standards organizations for securing RESTful web interfaces.

Table 19 – Open Security Standards for RESTful Interfaces

Standard	Description
Transport Layer Security (TLS)	IETF standard for secure communications between a client and server, providing transport-layer encryption, integrity protection, and authentication of the server using X.509 certificates (with optional client authentication)
OAuth 2.0	IETF standard for an authorisation framework whereby resource owners can authorize delegated access by third-party clients to protected resources; OAuth enables access delegation without sharing resource owner credentials, with optional limits to the scope and duration of access
JavaScript Object Notation (JSON)	Ecma[19] standard text format for structured data interchange – not a security standard per se, but a key component of several standards listed here
JSON Web Signature (JWS)	IETF standard for attaching digital signatures or Message Authentication Codes (MAC) to JSON objects
JSON Web Encryption (JWE)	IETF standard for encrypted JSON objects
JSON Web Keys (JWK)	IETF standard for representing public and private keys (or sets of keys) as JSON objects
JSON Web Algorithms (JWA)	Specifies cryptographic algorithms to be used in the other JOSE standards
JavaScript Object Signing and Encryption (JOSE)	Collective name for the set of JSON-based cryptographic standards (JWS, JWE, JWK, and JWA)
JSON Web Token (JWT)	IETF standard for conveying a set of claims between two parties in a JSON object, with optional signature and encryption provided by the JOSE standards
OpenID Connect 1.0	OpenID Foundation standard for identity federation based on OAuth 2.0, using JWT to convey signed and optionally encrypted identity claims
User-Managed Access (UMA)	Draft IETF standard for an OAuth 2.0-based access management protocol enabling resource owners to create access policies authorizing requesting parties to access their resources through OAuth clients

Figure 8 below illustrates the dependencies among the security standards, with each standard depending on the others that lie directly beneath it.

[19] ECMA was originally an acronym standing for the European Computer Manufacturers Association, but the organization changed its name in 1994 to Ecma International to reflect its global focus

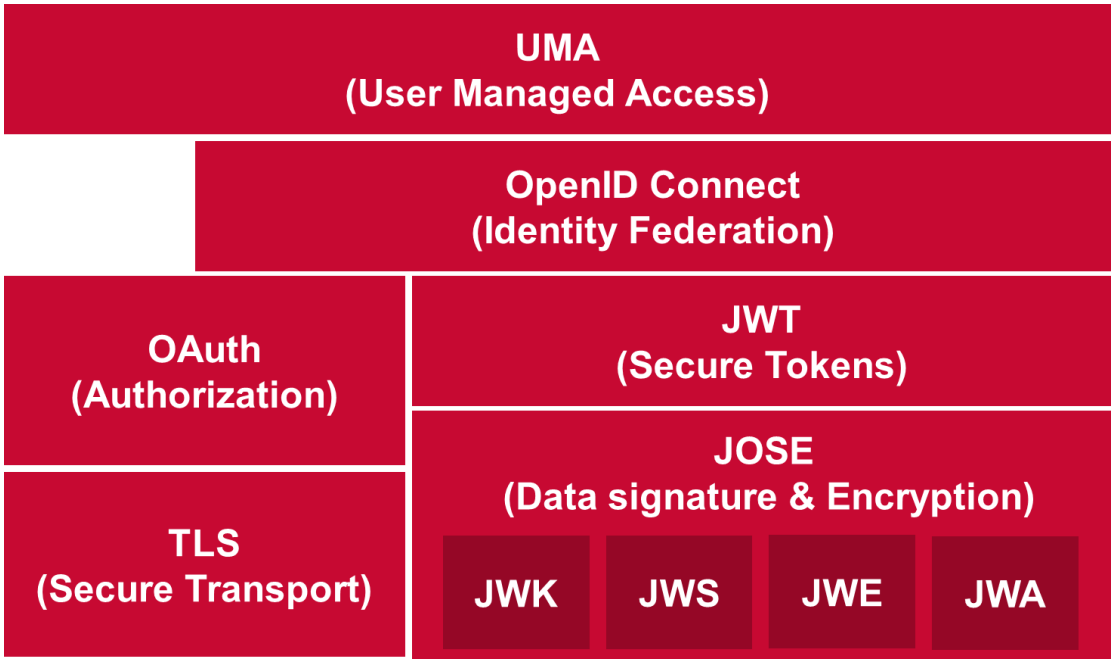

Figure 8: REST Security Standard Dependencies

Annex B Document Management

Document information

Project Owner	Thom Janssen
Project Manager	Mark Zandstra
Project Code	GSMA_2016_0663
Document Title	GSMA Mobile Money API - Security Design and Implementation Guidelines
File Name	GSMA Mobile Money API - Security Design and Implementation guidelines 1_1
Key Words	GSMA, Mobile Money, Harmonized API, Security Design
Classification	Confidential
Status	Draft
Distribution	GSMA

Version history

Version	Date	Status	Author
0.8	**18-05-2016**	**Draft**	**UL**
0.9	**23-05-2016**	**Final Draft**	**UL**
1.0	**24-06-2016**	**Final**	**UL**
1.1	**14-07-2016**	**V1.1**	**GKR Square Ltd**

Change history

Version	Date	Changes
0.8	**Draft**	**First external release to GSMA**
0.9	**Final Draft**	**Completed abbreviations, sections 2.4 and 2.5 and references, Processed comments from Gareth**

1.0	**Final**	**Some changes to reflect call from June 13th and added Best Practices; BP_MON_2, BP_MON_3, BP_MON_4 and updated BP_LOG_2, BP_IMAU_4, BP_VALI_2**
1.1	**14-07-2016**	**Incorporated concept of layer security model comprising of:** **1. Basic Security Model** **2. Advanced Security Model**
1.1	**29-07-2016**	**Added new security summary section**
1.2	**05-08-2016**	**1. Modified summary section as per feedback from Richard Murray** **2. Added a section around protecting API parameters**
1.3	**20-08-2016**	**1. Incorporated feedback from Operators/Vendors** **2. Added a new section on comparison between basic auth and OAuth client credentials flow** **3. Added a new section on secured storage of crypto keys and credentials**
1.4	**25-08-2016**	**1. Introduced the support for end user authentication including end user authentication user 3 legged OAuth flow, delegated authentication using 3rd party IDP and custom authentication using username/MSISDN and PIN**
1.5	**05-09-2016**	**1. Removed API Client certificate authentication based on feedback** **2. Renamed Entry Level option to Development Level option** **3. Clarified use of time zone when performing time deviation check**
1.6	**06-09-2016**	**1. Modified sections 3.2.1 and 3.2.2 based on feedback from Gareth**