



PKOC Specification

Security Classification:	Protected
Version:	3.0.0
Revision:	Rev0
Control:	Uncontrolled when printed
Date	9/10/2024



Document History

Version	Date	Author	Description
0.0-.99	Oct 8, 2021	Mohammad Soleimani	Initial Document
1.0RC1	Oct 22, 2021	Mohammad Soleimani, Alex Lammers	Changed to PSIA Stationary
1.0RC2	Oct 27, 2021	Mohammad Soleimani	Added ECDHE Perfect Forward Secrecy Flow Updated drawings for ECDHE to remove Normal Flow section
1.0	Nov 5, 2021		Released Specification
2.0RC1.0	OCT 25, 2023	Mohammad Soleimani	Comprehensive updates
2.1	FEB 28, 2024	Jason Ouellette	Changed encryption from AES-GCM to AES-CCM for today providing a path for AES-GCM; Added privacy protection for PKOC bit length to be unique in an PACS
2.2	Jul 19, 2024	Jason Ouellette	Adding BLE UUID updates from the Bluetooth SIG along with protocol version updates for the AES-GCM to AES-CCM support.
2.2.1	Sep 10, 2024	David Devasahayam Edwin	Added transaction timeout for the PKOC over BLE
3.0.0	Sep 30, 2024	Jason Ouellette	Updating version number to 3.0.0



Disclaimer

THIS DOCUMENT IS PROVIDED "AS IS" WITH NO WARRANTIES WHATSOEVER, INCLUDING ANY WARRANTY OF MERCHANTABILITY, NONINFRINGEMENT, FITNESS FOR ANY PARTICULAR PURPOSE, OR ANY WARRANTY OTHERWISE ARISING OUT OF ANY PROPOSAL, SPECIFICATION OR SAMPLE. Without limitation, PSIA disclaims all liability, including liability for infringement of any proprietary rights, relating to use of information in this specification and to the implementation of this specification, and PSIA disclaims all liability for cost of procurement of substitute goods or services, lost profits, loss of use, loss of data or any incidental, consequential, direct, indirect, or special damages, whether under contract, tort, warranty or otherwise, arising in any way out of use or reliance upon this specification or any information herein. You are hereby granted a license to copy and distribute (but not to modify) the information set forth in this document; and to make and sell, including commercially, products using the specification. Any rights that PSIA has not expressly granted are hereby reserved. PSIA encourages you to explore membership in the organization to obtain the full set of benefits to be received from use of its specification.



Table of Contents

- 1 INTRODUCTION: 4**
- 2 GLOSSARY:..... 4**
- 3 RECOMMENDED APP REGISTRATION PROCESS:..... 5**
- 4 REGISTRATION SEQUENCE DIAGRAM: 5**
- 5 PKOC REQUIREMENTS: 6**
 - 5.1 PKOC- CREDENTIAL CREATION AND PROVISIONING 6
 - 5.2 READER 6
 - 5.3 MOBILE DEVICE 7
 - 5.4 CRYPTOGRAPHIC FUNCTIONS 7
- 6 PKOC BLE EXCHANGE:..... 8**
 - 6.1 INITIAL ADVERTISEMENT AND GATT SERVICE SETUP 8
 - 6.2 DATA FOR ADVERTISING..... 8
 - 6.3 CALIBRATION DATA TABLE..... 10
 - 6.4 ACTIVATION RANGE TABLE 10
 - 6.5 PROTOCOL OVERVIEW 11
 - 6.6 MESSAGE TYPES 11
 - 6.7 ERROR CODES..... 14
- 7 PKOC TRANSMISSION FLOWS: 15**
 - 7.1 COMMON HANDSHAKE FOR BOTH THE FLOWS 15
 - 7.2 ECDHE WITH PERFECT FORWARD SECRECY FLOW 16
 - 7.3 UN-OBFUSCATED FLOW: 19
- 8 HIGH LEVEL INTERACTION DIAGRAMS 21**
 - 8.1 UN-OBFUSCATED FLOW 21
 - 8.2 ECDHE WITH PERFECT FORWARD SECRECY FLOW 22
- 9 SEQUENCE DIAGRAM:..... 22**
 - 9.1 UN-OBFUSCATED FLOW 23
- 10 ANDROID PKOC SAMPLE CODE: 25**
- 11 IOS PKOC SAMPLE CODE: 27**



1 Introduction:

This document describes an open air-interface protocol for Bluetooth Low Energy (BLE) interactions for a secure credential exchange with a lock or a Reader for access control. This specification is based on Public Key Open Credential (PKOC) concept, which is a system solution for secure, interoperable credentials by utilizing distributed asymmetric encryption key creation. A phone/smart device will generate an asymmetric key pair using EC (Elliptic Curve) algorithm and store it in a secure element (secure enclave in iOS and key store in android). The Reader/lock will receive the public key from the phone and authenticate it via challenge-response before granting access.

2 Glossary:

Android – An operating system for mobile devices developed and maintained by Google, Inc.

Application (App) – A software application developed to run on a mobile operating system.

Asymmetric Key Cryptography – A method of communicating where two keys are used, one public which can be transmitted securely in the open, and the other private which should be kept secret. A message can then be either encrypted and/or signed using one key to be decrypted and/or verified by the corresponding key.

Bluetooth Low Energy (BLE) – A subset of the Bluetooth specification designed for low power wireless data exchange without the need to pair devices or maintain a persistent connection.

Challenge-Response – An authentication protocol where a trusted device sends a single-use challenge that an untrusted device performs a cryptographic operation with using a secret to generate a response that establishes that the untrusted device possesses and controls the secret.

Credential – A physical or digital token which contains a unique identifier for the individual to whom it was issued. The credential can then be used to assert the identity of the individual to whom it was issued to systems that recognize the issuer.

Digital Signature – Additional data attached to a message that uses encryption and decryption algorithms to verify the message's origin and contents.

Elliptic Curve Diffie-Hellman Ephemeral (ECDHE) - is an anonymous key agreement protocol that allows two parties, each having an elliptic curve public-private key pair, to establish a shared secret over an insecure channel.

iOS – An operating system for mobile devices developed and maintained by Apple, Inc.

CCM - CBC counter mode, is a mode of operation for cryptographic block ciphers. It is an authenticated encryption algorithm designed to provide both authentication and confidentiality. CCM mode is only defined for block ciphers with a block length of 128 bits.



GCM – Galois/Counter Mode, an authenticated encryption mode for block ciphers like AES. It produces both a ciphertext and a tag used to validate the encrypted data.

Lock – A physical device designed to prevent access without the use of a corresponding key.

Mobile Device – A portable telephone or other network-connected smart device designed to be carried or worn by the owner.

Nonce – A single use number used as a challenge with Challenge-Response.

Panel – A computing device that is maintained within the secured perimeter of a building that electronically locks and unlocks doors or devices based on messages received at the point of access from a Reader.

Public Key Open Credential (PKOC) – A method of permitting the secure authentication and transmission of the credential of an individual using public keys which can be securely passed in the open which are not issued or created by a centralized authority.

Reader – A small device that is mounted near or embedded within a Lock that permits the transmission of a credential to a Panel using a magnetic stripe, RFID, NFC, or Bluetooth to exchange data with a credential issued to an individual.

Secure Element - A hardware component that provides all cryptographic operations for authenticating the user and is designed to be secure even if the operating system kernel is hacked (e.g., Secure Enclave in iOS and Key Store in Android.)

Universally Unique Identifier (UUID) – a 128-bit number used to identify a unique object on a computer system.

Bluetooth SIG UUID – a 16 bit unique identifier specific to the PSIA organization used to advertise PKCO BLE (0xFFFD).

3 Recommended App Registration Process:

In the app, we would have a registration where:

- The app will generate the asymmetric key pair with Elliptic Curve (EC) cryptography and save it to the secure element.
- It will send the public key to the cloud at the time of registration. The public key should be transferred in an encrypted format.

4 Registration Sequence Diagram:

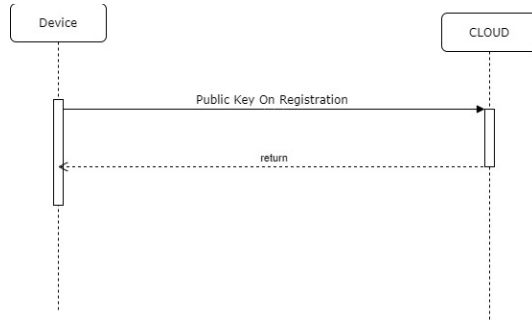


Figure 1 - Registration Sequence

5 PKOC Requirements:

5.1 PKOC- Credential Creation and Provisioning

The device will generate a public/private key pair using the NIST secp256r1 (P256) curve, stored in an internal secure element. The private key never leaves the device, and the public key is used as a user’s Public Key Open Credential (PKOC).

The physical access control system shall define a unique bit length specific for use with PKOC credentials system wide. This prevents an unauthenticated transaction with non-compliant readers. The PACS shall allocate a PKOC credential format of a minimum of 64 bits but should use the largest bit length that can be supported up to the maximum 256 bit “X” portion of the public key.

See below for examples of extracting the credential from the full 65-byte Public Key.

For 256 bits -

04BEA02AA1320054CFF1DFD2F88FA583B5B059833BA87CEC415ABDAE0791F0EC66A913C7104A725F6497B8C0 8FF91217B106FEF7B51ACD4ADF6645E765E4E88D84 For

64 bits -

04BEA02AA1320054CFF1DFD2F88FA583B5B059833BA87CEC415ABDAE0791F0EC66A913C7104A725F6497B8C0 8FF91217B106FEF7B51ACD4ADF6645E765E4E88D84

5.2 Reader

To meet the data transmission requirements for PKOC all Readers must support BLE 4.2¹ or later and utilize the Data Packet Length Extension to permit up to 242 bytes of data to be transmitted. Any reference to Reader in this specification can also refer to a passthrough from a Reader to a Panel where the Panel performs the Reader’s actions.

¹ <https://www.bluetooth.com/specifications/specs/core-specification-4-2/>



5.3 Mobile Device

To interface with a PKOC compliant Reader a mobile device must also support BLE 4.2 or later. Of the most common mobile operating systems in use today this would require IOS 9.2.1 or later, and Android 6.0.1 or later². Developers are strongly encouraged to support Android 9.0 and later to utilize the operating system level secure enclave.

5.4 Cryptographic Functions

PKOC utilizes asymmetric cryptographic functions to securely transmit public keys and other data to Readers. To generate digital signatures on the most common mobile operating systems EcdsaSignatureMessageX962Sha256 is preferred for IOS and SHA256WithECSDA for Android. The signature must follow DER x9.62 encoding but transmit only the 64-byte "real numbers" from the 72-74 byte DER encoded signature. For the Reader, cryptographic functions must be performed by a hardware secure element.

² <https://devzone.nordicsemi.com/f/nordic-q-a/11678/ble-4-2-and-smart-phones-os-compatibility>



6 PKOC BLE Exchange:

6.1 Initial Advertisement and GATT Service Setup

The BLE beacon advertisement and service UUID for PKOC Readers will be the Bluetooth SIG UUID for PKOC Service from v2.2 or newer of the specification. For backwards reader compatibility, PKOC BLE clients may also look for the PKOC GATT Service legacy UUID

The PKOC GATT Service shall have two characteristics:

1. **PKOC Write Characteristic** which shall support: Write without response.
2. **PKOC Read Characteristic** which shall support: Notify

Description of UUID	Value
PKOC GATT Service Legacy UUID	0x41fb60a1-d4d0-4ae9-8cbbb62b5ae81810
PKOC GATT Bluetooth SIG UUID	0xFFFF0
PKOC Write Characteristic UUID	0xfe278a85-89ae-191f-5dde-841202693835
PKOC Read Characteristic UUID	0xe5b1b3b5-3cca-3f76-cd86-a884cc239692

Table 1 PKOC UUID values

6.2 Data for Advertising

The Service UUID for the PKOC GATT Service shall also be advertised in the Advertising Data to be discoverable by passive Bluetooth Scanners.

It is recommended to include the Service Data for the PKOC GATT Service using the Bluetooth SIG 16 bit UUID data element to provide common advertisement-level information about the reader for use in hands-free ranging and reader discovery.



Data Element	Length	Description
Ranging Calibration Data	4 bits	Expected RSSI value at 1 meter distance, as defined by the Calibration Data table
Ranging Activation Range	4 bits	Expected distance to connect, as defined by the Activation Range table
Reader Location Identifier	4 bytes	Least significant 4 bytes of the Reader Location Identifier

Table 2 Service Data for PKOC GATT Service

6.3 Calibration Data Table

Device reference for calibration data is an Apple iPhone 14.

Value	RSSI (dBm)
0	Invalid
1	-52
2	-54
3	-56
4	-58
5	-60
6	-62
7	-64
8	-66
9	-68
10	-70
11	-72
12	-74
13	-76
14	-78
15	-80

6.4 Activation Range Table

Value	Range (centimeters)
0	Invalid
1	5
2	7
3	10
4	14
5	20
6	28
7	40
8	56
9	80
10	113
11	160
12	226
13	320
14	452
15	640



6.5 Protocol Overview

The PKOC over BLE protocol defines 2 roles , a PKOC over BLE GATT (Generic Attribute) Server role and a PKOC over BLE GATT Client role.

On a PKOC over BLE Server, a transaction shall start when a Notification is sent by the Server. A transaction shall complete when the response is received by the server.

On a client, a transaction shall start when a write is sent by the client. A transaction shall complete when the PKOC over BLE Notification is received by the client.

A PKOC transaction not completed within 1 second shall time out. Such a transaction shall be considered to have failed and the local higher layers shall be informed of this failure. No more PKOC protocol requests, commands or notifications shall be sent to the target device on this GATT session. To send another PKOC protocol PDU, a new GATT session must be established between these devices. The existing GATT session may need to be terminated before the new GATT session is established.

6.6 Message Types

- Messages will utilize a Type/Length/Value (TLV) command structure.
 - Type will be fixed at 1 byte.
 - Length will be fixed at 1 byte permitting a maximum 240 bytes to be usable for the value.
 - Value will contain the data needed to fill but not exceed the length specified.
- Packets can contain more than one message using concatenation.
 - Additional messages will begin at the byte immediately following the last message. ○ There is no limit on the number of messages that can be concatenated subject to the maximum transmission unit size of 242 bytes.

Types can be sent or received in any order. Support for the following messages is mandatory to be PKOC compliant. The reader and device must ignore types not included in this list.

GATT Specific Type

Type	Details	Data	
0x01	Uncompressed Public Key ECC P-256 (65 Bytes)	65-byte Non-Ephemeral Public Key	PKOC Write
0x02	Compressed Ephemeral Public Key	33 byte Compressed Public Key	PKOC Read
0x03	Digital Signature	64-byte ECDSA Signature (R S)	PKOC Write
0x04	Response	See Error Codes	PKOC Write
0x05	Reserved (Deprecated Functionality)		
0x06	Reserved (Deprecated Functionality)		
0x07	Uncompressed Ephemeral Public Key	65-byte Ephemeral Public Key	PKOC Write



0x09	Last Update Time	4-byte unsigned Unix Epoch time. Update time of the most recent change to a PKOC credential or any associated access information.	PKOC Write
0x0A	Reserved (Deprecated Functionality)		
0x0C	Protocol Identifiers	5 bytes. Specification version – 1 byte (Table for Specification Version) Vendor sub version – 2 bytes Feature bits - 2 bytes: 0x xxx1: CCM supported (bit 0) 0x xxx2: GCM supported (bit 1) Bits 2 to 15 are RFU	PKOC Read
0x0D	Reader Location Identifier	16-byte GUID logical identifier for the reader location. It is not hardware specific.	PKOC Read
0x0E	Site Identifier	16-byte GUID logical identifier for the device’s public key. This can be shared between readers in each area/building.	PKOC Read
0x40	Encrypted Data	Variable length ciphertext	PKOC Read/Write
0x80	MFG Specific	3-byte Manufacturer Identifier	PKOC Read/Write

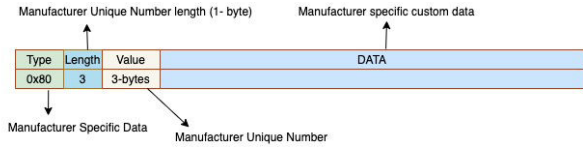
- **Manufacturer Specific Data:** Type 0x80h will be reserved to allow Manufacturer Specific Data (GATT) for the transmission of non-standard data within the standard messaging protocol. This will be in TLV format.
 - Type must be 1 byte with a value of 0x80h for Manufacturer Specific Data.
 - Length will be 1 byte with a value of 0x03h for the Manufacturer Unique Number length.
 - Value must be 3 bytes for the Manufacturer Unique Number³ followed by the rest of the data.

³ Manufacturer’s Unique Number is obtained from the IEEE

<https://standards.ieee.org/products-programs/regauth/oui36/>



PKOC Specification



- Reader/Panel will perform data verification steps and will send the acknowledge byte to the Device.

Specification name	Specification Version
PKOC specification v2.1	0x00
PKOC specification v3.0.0	0x01

Table 3 PKOC BLE Specification Version

6.7 Error Codes

These come from the reader, the Device is not required to send these messages.

Name	Value	Use
Unknown Failure	0x00	Miscellaneous error
Success	0x01	Operation completed successfully. This does not mean that Access was Granted.
Access Denied	0x02	Access was explicitly denied. Readers that cannot determine this information directly do not need to support this feature.
Access Granted	0x03	Access was explicitly granted. Readers that cannot determine this information do not need to support this feature, and Applications should not require it.
Decryption Error	0x04	Decrypted data could not be validated (GCM failure)
Invalid Security Status	0x05	Receipt of an encrypted packet without having derived a secret
Signature Invalid	0x06	Reader could not validate signature
Decryption Error	0x07	Decrypted data could not be validated (CCM failure)



7 PKOC Transmission Flows:

PKOC supports two (2) types of data communication between the PKOC Reader and an App on a mobile device: ECDHE or Un-Obfuscated. The App will scan for a PKOC Service UUID when a Reader is discovered.

NOTE: While both the flows are required in the Reader to support the PKOC specification, the app will select either of the flows.

7.1 Common Handshake for both the flows

The below steps are the handshake procedures before the app selects which flow it will authenticate with.

1. App will scan for PKOC Service UUID, if the Reader is discovered.
2. App will make connection with the Reader.
3. App will discover the PKOC GATT Service.
4. App will discover the PKOC Characteristic UUIDs.
5. App will Request to initiate the transaction to the Reader.
6. Reader will send Protocol identifiers that it supports (0x0C) + Reader Ephemeral Public Key (0x02) + Reader Location Identifier (0x0D) + Site Key Identifier (0x0E) to the App.

Type	Length	Value	Type	Length	Value	Type	Length	Value	Type	Length	Value
0x0C	0x05	Protocol Identifier	0x02	0x21	Reader Ephemeral Public Key	0x0D	0x10	Reader Identifier	0x0E	0x10	Site Identifier

Based on the Reader id in advertisement data, the App will decide whether it can follow the ECDHE or Un-Obfuscated Flow.

NOTE: Going forward, protocol features can be used to indicate shift of encryption mode from AES-CCM to AES-GCM.



7.2 ECDHE with Perfect Forward Secrecy Flow

This protocol primarily utilizes a Full-ECDHE-based flow to establish a secure channel for use in transporting credentials securely.

The ECDHE Flow transmits the PKOC using Elliptic Curve encryption. This flow provides the maximum amount of privacy with Forward Secrecy but requires prior knowledge of the public key of the Site.

Cryptography

To prevent man-in-the-middle attacks, data from both the reader and device are used in the signature computation. The input to the signature function is the following, concatenated:

Data	Length (Bytes)
Site Identifier	16
Reader Location Identifier	16
Phone Ephemeral Public Key X Component	32
Reader Ephemeral Public Key X Component	32

Secure Session

Once the reader and device have shared ephemeral keys, they will each compute a shared secret key using the ECKA-DH key agreement protocol (see BSI TR-03111 section 4.3), using SHA-256 as the final key derivation hash. The output of this function, Z_{AB} , will be used directly as a 256bit AES-CCM key (see NIST SP 800-38D).

Encrypted packets are marked using the Encrypted Data TLV, which contains the encrypted data and the 16-byte Tag. This tag must be validated prior to decryption.

The IV is 12 bytes, in the format of 0x000000000000001AABBCCDD, where AABBCCDD is a 4-byte counter that increments after every encrypted message sent. It starts at 1 and is encoded Big Endian. There is a separate counter for the reader and the device. If the counter rolls over the secure session is invalidated and all encrypted communications should fail until a new session is established.

Flow Description

To establish a secure communications channel, the Device and Reader interact as follows:



1. The Device has recognized the Reader from the Reader's advertisement and already has the Site's public key
2. Device sends a transaction initiation request and creates an Ephemeral Key pair
3. Reader Performs below operation:
 - a Generates Ephemeral EC Keys
 - b Sends below TLVs to the Device:
 - i Protocol identifiers
 - ii Reader Ephemeral Public Key
 - iii Reader Location Identifier
 - iv Site Key Identifier
4. Device Performs below operations:
 - a Send below TLVs to the Reader:
 - i Ephemeral Public Key
 - ii Protocol identifiers
 - b Device generates Shared Secret
5. Reader Performs below operations:
 - a Sends Digital Signature to the Device after performing:
 - i Reader concatenates below data:
 - 1 Site Identifier
 - 2 Reader Location Identifier
 - 3 Device Ephemeral Public Key X component
 - 4 Reader Ephemeral Public Key X component
 - ii And then signs it with site private key
 - b Generates Shared Secret
6. Device verifies the Reader Digital Signature
 - a If verification fails, it closes the connection and throws error
 - b Else:
 - i Creates PKOC Digital Signature using below steps:
 - 1 Device concatenates below data:
 - a Site Identifier
 - b Reader Location Identifier
 - c Device Ephemeral Public Key X component
 - d Reader Ephemeral Public Key X component
 - 2 And then sign with PKOC private key
 - ii Encrypt the below TLVs using Device's shared secret key:
 - 1 Device's PKOC Public Key
 - 2 Digital Signature
 - 3 Last Update Time
 - iii Send the above encrypted data to the Reader, wrapped in an Encrypted Data TLV
7. Reader on receiving the encrypted data:
 - a Decrypts the Encrypted Data TLV using Reader's Shared Secret key



- b Verifies Digital Signature using Device's Public key
- a Sends back the response to the Device

Note: Details of different error codes are mentioned in the Error Codes section.

8. Upon receipt of an encryption error (if any), the Device and Reader *must* close the current connection and re-connect (including new ECDHE negotiation, if necessary).



7.3 Un-Obfuscated Flow:

The Un-Obfuscated flow is a plaintext flow for passing the PKOC credential. For security purposes, this must be limited use:

- PKOC credentials sent over the un-obfuscated flow must be time-limited (temporary access, visitor pass, etc.).
- Applications cannot use the un-obfuscated flow for hands-free operation, the user must actively initiate the connection (select the reader from UI).

Cryptography

To validate the PKOC public key, the device will generate a signature that the reader will validate. The input to that function is the raw bytes of the Reader's Ephemeral Public Key TLV as sent by the reader (33 bytes).

Flow Description

To establish a communications channel, the Device and Reader interact as follows:

1. The Device has recognized the Reader from the advertisement
2. Device sends a transaction initiation request
3. Reader Performs below operations:
 - a Generates Ephemeral EC Keys
 - b Sends below TLVs to the Device
 - i Protocol identifiers
 - ii Reader Ephemeral Public Key
 - iii Reader Location Identifier
 - iv Site Key Identifier
4. Device creates PKOC Digital Signature using below steps:
 - a Generates the digital signature by signing the Reader's Ephemeral Public Key with the PKOC Private Key
 - b Device concatenates below TLVs and sends to the reader
 - i Device's PKOC Public Key
 - ii Digital Signature
 - iii Last Update Time
 - iv Protocol identifiers
5. Reader on receiving the Digital Signature
 - a Verifies Digital Signature using PKOC Public key
 - b Sends back the response to the Device.

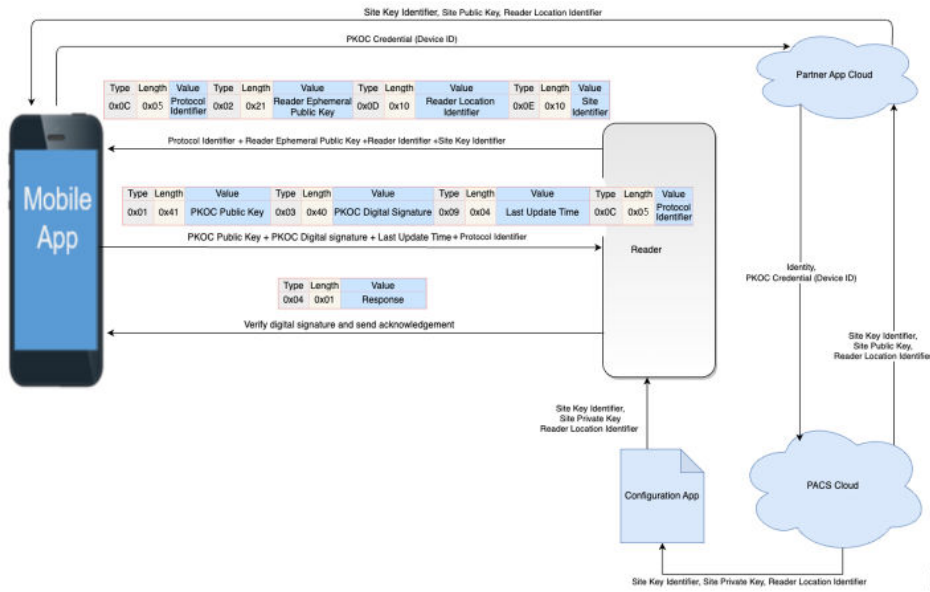


Note: Details of different error codes are mentioned in the Error Codes section.

6. Upon receipt of an encryption error (if any) or success, the Device and Reader must close the current connection and re-connect (including new ECDH negotiation, if necessary).

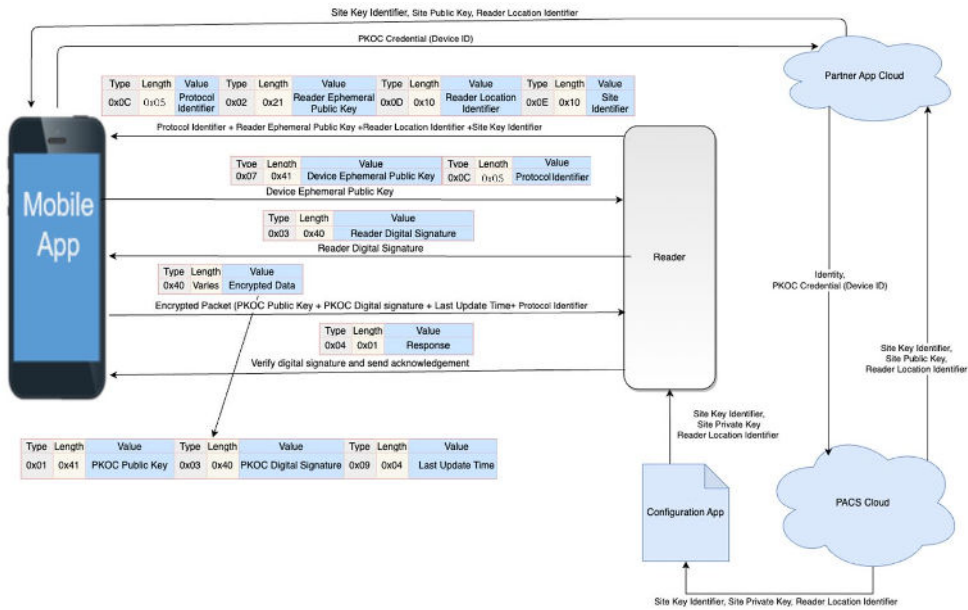
8 High Level Interaction Diagrams

8.1 Un-Obfuscated Flow



Un-Obfuscated Flow - Mobile Lock Interaction

8.2 ECDHE with Perfect Forward Security Flow

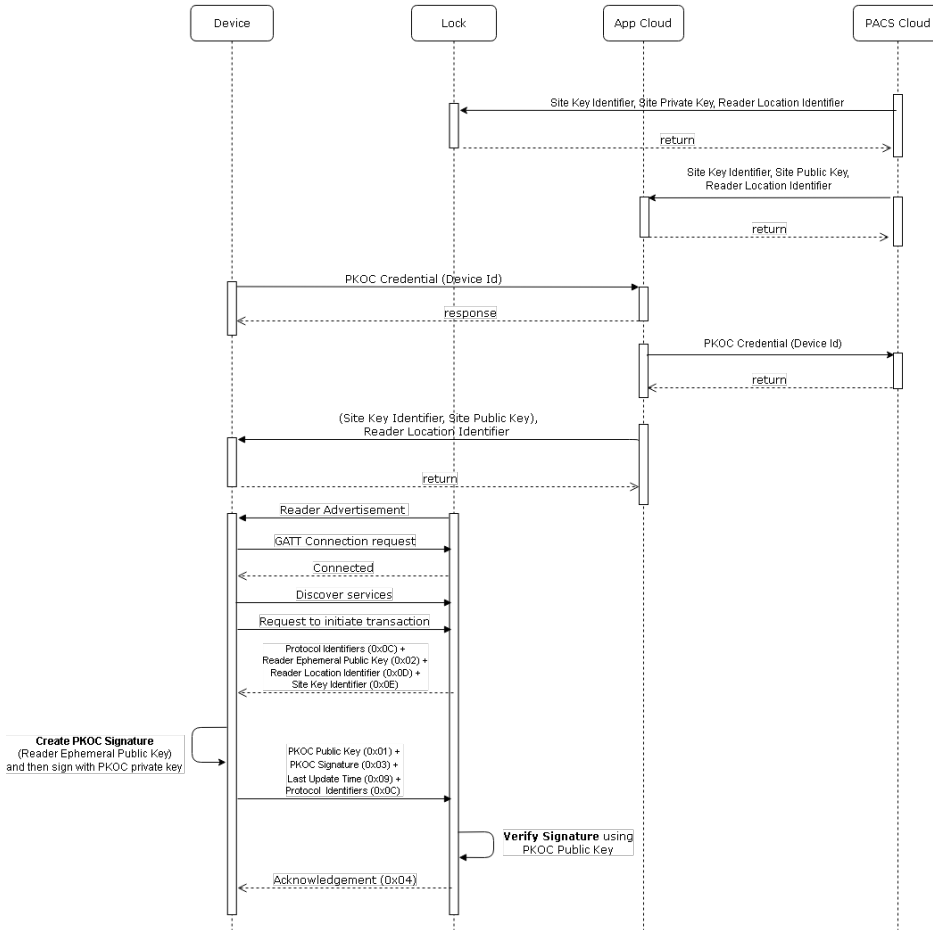


ECDHE with Perfect Forward Security Flow – Mobile Lock Interaction

9 Sequence Diagram:

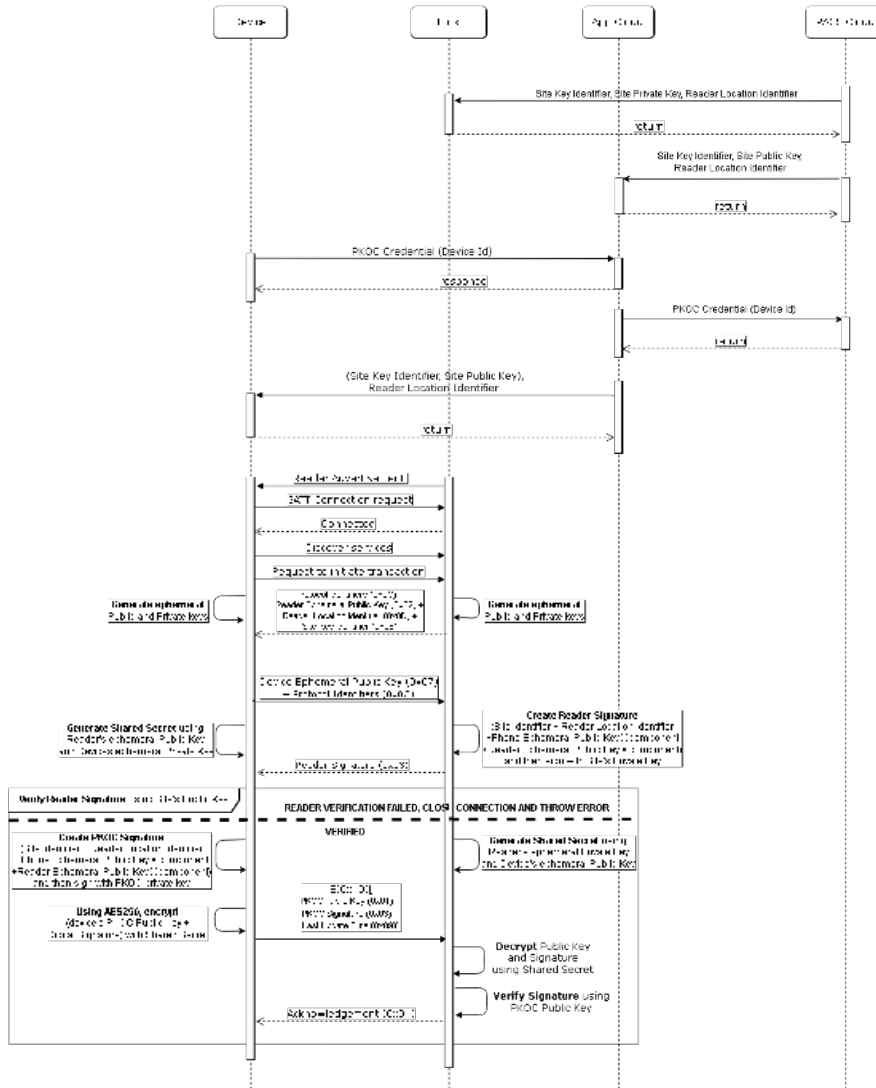
9.1 Un-Obfuscated Flow

LOCK PKOC AUTHENTICATION



Lock PKOC authentication (Forward Secrecy ECDHE)

LOCK PKOC AUTHENTICATION



10 Android PKOC Sample Code:

1. Create a P-256 ECC public/private key pair

```
public void CreateKeyPair1()
{
    KeyPairGenerator keyGenerator =
    KeyPairGenerator.GetInstance(KeyProperties.KeyAlgorithmEc, "AndroidKeyStore");

    ECGenParameterSpec ecParam = new ECGenParameterSpec("secp256r1");
    var builder = new KeyGenParameterSpec.Builder(_keyName, KeyStorePurpose.Sign |
    KeyStorePurpose.Verify)
        .SetUserAuthenticationRequired(false)
        .SetDigests(KeyProperties.DigestSha256)
        .SetRandomizedEncryptionRequired(false)
        .SetKeySize(256)
        .SetAlgorithmParameterSpec(ecParam);
    keyGenerator.Initialize(builder.Build())
    keyGenerator.GenerateKeyPair(); }
}
```

2. Get the public key

```
public IKey GetPublicKey()
{
    if (!_androidKeyStore.ContainsAlias(_keyName))
    return null;
    return _androidKeyStore.GetCertificate(_keyName)?.PublicKey;
}
```

3. Get the signature from PK and the Hash value

```
// Create Signature instance
Signature s = Signature.GetInstance("SHA256withECDSA");
// Get Private Key
var key = ((PrivateKeyEntry)entry).PrivateKey;
// initialization signature with private key.
s.InitSign(privateKey);
// Update the data to be verified.
s.Update(data); // create
signature var signature =
s.Sign();
```

4. Extract 64 Byte of signature

Formatted: Italian (Italy)



```
public static byte[] RemoveASNHeaderFromSignature(byte[] signature)
{
    if (signature.Length == 64)
    {
        return signature;
    }
    var calculatedSignature = new byte[64];
    int xLength = signature[3];    var arrX =
    new byte[32];                // copy 32 bytes in arrayX
    Array.Copy(signature, xLength == 32 ? 4 : 5, arrX, 0, 32);
    // copy 32 bytes in arrayY
    int yLength = signature[3 + xLength + 2];

    var arrY = new byte[32];
    Array.Copy(signature, yLength == 32 ? (3 + xLength + 2 + 1) :
    (3 + xLength + 2 + 1 + 1), arrY, 0, 32);
    // copy arrayX and arrayY in signature bytes.
    Array.Copy(arrX, 0, calculatedSignature, 0, arrX.Length);
    Array.Copy(arrY, 0, calculatedSignature, arrX.Length, arrY.Length);

    return calculatedSignature; }
}
```



11 iOS PKOC Sample Code:

1. Create a P-256 ECC public/private key pair

```
private bool GenerateKECKeyPair() {
using (var access = new SecAccessControl(SecAccessible.AfterFirstUnlockThisDeviceOnly,
SecAccessControlCreateFlags.PrivateKeyUsage) {
    var secKeyAttributes = new SecPublicPrivateKeyAttrs {
        ApplicationTag = KeyStoreName,
        CanDecrypt = true,
        CanEncrypt = true,
        CanSign = true,
        CanVerify = true,
        IsPermanent = true,
    });
    var secStatusCode = SecKey.GenerateKeyPair(SecKeyType.EC, 256, secKeyAttributes, out SecKey
publicKey, out SecKey privateKey);
    if (secStatusCode == SecStatusCode.Success)
    { return true; }
    else{ return false; }
}
}
```

2. Get the public key:

```
private SecKey GetPublicKey() var Key =
SecKeyChain.QueryAsConcreteType(
new SecRecord(SecKind.Key){
KeyType = SecKeyType.EC,
ApplicationTag = KeyStoreName,
},
out var code);
code == SecStatusCode.Success ? Key as SecKey : null;
return key.GetPublicKey();
```

3. Get the signature from PK and the Hash value

```
NSData generateDigitalSignature (NSData hashData)
{
    NSData signedData = null;
    try {
        SecKey privateKey = getPrivateKey ();
        Console.WriteLine (privateKey);
    } catch (NSError err) {
        if (privateKey == null) {
            GetSecureEnclavePublicKey ();
            privateKey = getPrivateKey ();
        }
    }
}
```

```
signedData = privateKey?.CreateSignature (SecKeyAlgorithm.EcdsaSignatureMessageX962Sha256, hashData,
out err);
Console.WriteLine (signedData);
} catch (Exception ex) {
}
return signedData;
}
```

4. Extract 64 Byte of signature



```
private byte[] RemoveASNHeaderFromSignature(byte[] signature)
{
    if (signature.Length == 64)
    return signature;
    var calculatedSignature = new byte[64];
    int xLength = signature[3];          var arrX =
    new byte[32];
    Array.Copy(signature, xLength == 32 ? 4 : 5, arrX, 0, 32);
    int yLength = signature[3 + xLength + 2];
    var arrY = new byte[32];
    Array.Copy(signature, yLength == 32 ? (3 + xLength + 2 + 1) : (3 + xLength + 2 + 1
+ 1), arrY, 0, 32);
    Array.Copy(arrX, 0, calculatedSignature, 0, arrX.Length);
    Array.Copy(arrY, 0, calculatedSignature, arrX.Length, arrY.Length);
    return calculatedSignature;
}
```



END OF DOCUMENT