智 方 便
iAM Smart

"iAM Smart" Sandbox Programme

Digital Signing

智 方 便 iAM Smart | SANDBOX Programme

Agenda

- Digital Signing Overview
- Identification Code Generation
- “iAM Smart+” Certification
- Different type of digital signing
- UI Requirements
- Quiz

智方便 iAM Smart | SANDBOX Programme

Disclaimer

Please be aware that the video is intended for preliminary introduction. It shall not be followed as the technical instruction. “iAM Smart” Sandbox Programme would not guarantee the correctness and timeliness of data which could be possibly affected by the modification of development. Development team shall follow the guidelines and policies or enquire to related professionals if any safety apprehension.

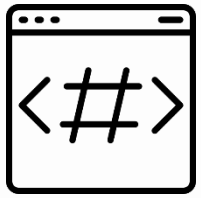
Digital Signing with Service Login

What is Digital Signing with Service Login

Users can use "iAM Smart+" to perform digital signing in accordance with the Electronic Transactions Ordinance (Chapter 553 of the Laws of Hong Kong) to process legal documents and procedures online. The function can be performed after the service login.

PDF Digital Signing / Hash Digital Signing

After iAM Smart receiving the document that requiring signature, OSP need to choose which kind of digital signing would be executed.



Get the document(**PDF, text**, image, etc…)

1234

Generate 4-digit identification code

User verification

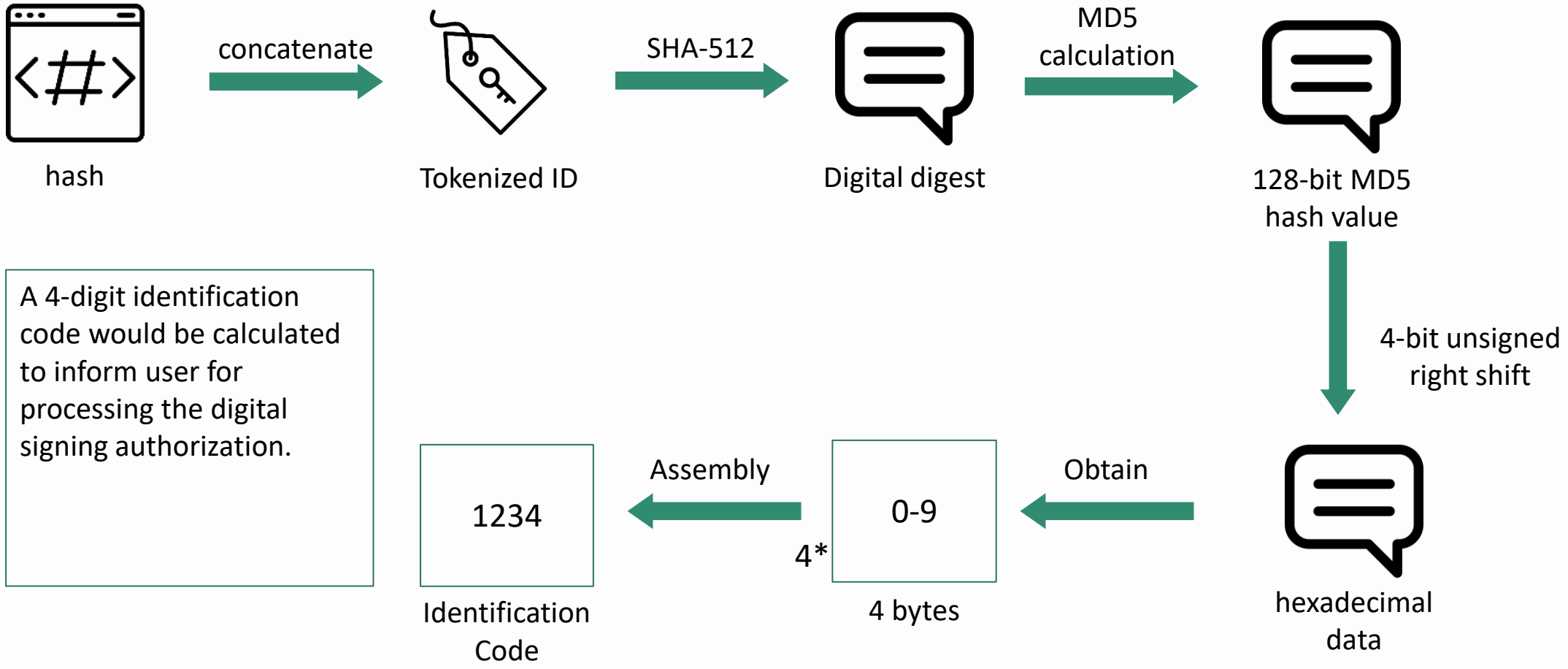
The PDF Digital Signing **only support the digital signature with PDF file type**. Corresponding to other document type, the hash digital signing would be selected.

Result computing

Server acknowledge the result to “iAM Smart” System

Inform client terminal to stop polling and save the accessToken

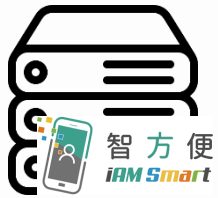
Identification Code for Document to be signed



"iAM Smart+" Users and "iAM Smart" Cert

Certification information

- Originality
  - Issued by Hongkong Post
- Functionality
  - Key factor for digital signing function
- Security
  - Stored in the "iAM Smart" System



Each iAM Smart-Cert is valid for one year.

"iAM Smart" will send a mobile push notification and an email reminder to the relevant user 30 days before the expiry of an iAM Smart-Cert to renew

(Electronic Transactions Ordinance (Cap. 553))

UI Requirements

After an online service sends the digital signing request to "iAM Smart" system, the online service is required to show the online service name, document name, identification code and instruct user to open the "iAM Smart" mobile app in his/her mobile phone referring to format shown on the table with corresponding language.

	Same Device	Different Device
Example		

Here is the example showing the signing step and interface

	Same Device	Different Device
Instructions		
Traditional Chinese	請按照以下步驟： 1. 點擊以下「開啟智方便」按鈕 2. 請確保「智方便」顯示的識別碼相同 3. 點擊「簽署」以完成數碼簽署 開啟智方便	請按照以下步驟： 1. 開啟你手機上的「智方便」應用程式 2. 請確保「智方便」顯示的識別碼相同 3. 點擊「簽署」以完成數碼簽署
Simplified Chinese	请按照以下步骤： 1. 点击以下「开启智方便」按钮 2. 请确保「智方便」显示的识别码相同 3. 点击「签署」以完成数码签署 开启智方便	请按照以下步骤： 1. 开启你手机上的「智方便」应用程式 2. 请确保「智方便」显示的识别码相同 3. 点击「签署」以完成数码签署
English	Please follow the steps below: 1. Click on "Open iAM Smart" below 2. Make sure the identification code shown in "iAM Smart" is the same 3. Tap on "Sign" to complete the digital signing Open iAM Smart	Please follow the steps below: 1. Open "iAM Smart" app in your mobile device 2. Make sure the identification code shown in "iAM Smart" is the same 3. Tap on "Sign" to complete the digital signing

API – Initiate Digital Signing After Service Login – 1

Obtain all the parameters and the CEK as variable of calling API.
Location: RequestSigningController.java (initiateSigning)

Construct the JSON

Compute the hashcode by encoding with SHA-256 method to get a Base64 encoded value

Initiate digital signing with the JSON and the CEK

```
ObjectNode jsonObj = objectMapper.createObjectNode();
jsonObj.put("businessID", UUID.randomUUID().toString().replace("-", ""));
jsonObj.put("accessToken", accessToken);
jsonObj.put("openID", openId);
jsonObj.put("source", source);
jsonObj.put("redirectURI", iamCallbackEndpoint + Constants.API_SIGNING_CALLBACK);
jsonObj.put("state", state);

// Prepare document hash
String hashCode = "";
try {
    byte[] documentBytes = docxResource.getInputStream().readAllBytes();
    hashCode = Base64.getEncoder().encodeToString(Security.digestToByte(documentBytes, "SHA-256"));
    System.out.println(hashCode);
} catch (IOException e) {
    e.printStackTrace();
}

jsonObj.put("hashCode", hashCode);
jsonObj.put("sigAlgo", "NONEwithRSA");
// jsonObj.put("HKICHash", hkic);
jsonObj.put("department", "MCC");
jsonObj.put("serviceName", "MCC service");
jsonObj.put("documentName", "Signing demo");

String respJson = callApi(Constants.API_SIGNING_REQUEST, jsonObj, cek);
```

API – Initiate Digital Signing After Service Login – 2

After calling the API, it needs to calculate the identification code for acknowledgement.
Location: Security.java (getSignCode)

Calling the method inside security.java

```java
String respJson = callApi(Constants.API_SIGNING_REQUEST, jsonObj, cek);
// ApiRespDecrypted<ApiContentRequest> resp = new ApiRespDecrypted<>(respJson,
// cek);
ApiRespSigning<ApiContentRequest> resp = new ApiRespSigning<>(respJson, cek,
        Security.getSignCode(hashCode, openId));

// System.out.println(Security.getSignCode(hashCode, openId));

String respCode = resp.getCode();
if (respCode != null && respCode.equals(Constants.SUCCESS_CODE)) {
    return ResponseEntity.ok(new Result<>("SUCCESS", resp));
}

return ResponseEntity.status(HttpStatus.BAD_REQUEST).body(new Result<>("iAM Smart error", resp));
```

```java
public static String getSignCode(String hashCode, String openID) {
    // Decode hashCode from Base64
    byte[] hashCodeBytes = Base64.getDecoder().decode(hashCode);

    byte[] openIDBytes = digestToByte(openID, "SHA-512");
    char[] code = new char[4];

    byte[] source = new byte[hashCodeBytes.length + openIDBytes.length];
    System.arraycopy(hashCodeBytes, 0, source, 0, hashCodeBytes.length);
    System.arraycopy(openIDBytes, 0, source, hashCodeBytes.length, openID...length);

    byte[] inData = digestToByte(source, "SHA-512");

    byte[] digestMD5Data = digestToByte(inData, "MD5");

    for (int i = 0; i < 4; i++) {
        code[i] = hexDigits[(digestMD5Data[i * 4] >>> 4 & 0xf) % 10];
    }

    return new String(code);
}
```

Perform the SHA-512 calculation

Perform the MD5 calculation

Final step to assembly into 4-digit code

API – Open "iAM Smart" App For Digital Signing

Online Service App invokes “iAM Smart” Mobile App using URL Scheme with request parameters.

The context links deeply to redirect users to the “iAM Smart” mobile app.

Using the same device as an example

Online service app

<context>

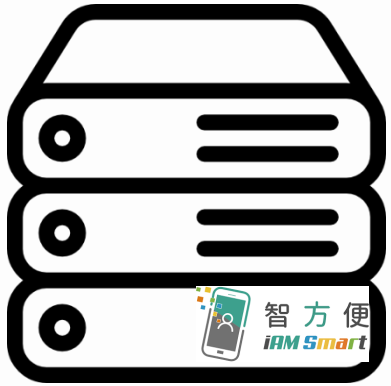
iAM Smart mobile app

API – Digital Signing Callback from "iAM Smart" Server

iAM Smart return the result to e-Service Center

User could accept or reject the digital signing request

"iAM Smart" System invokes Online Service callback API to return the result with "businessID" of the digital signing request to Online Service server. API data decryption is required.



iAM Smart system

return

e-Service Center

API – Acknowledge Digital Signing Result – 1

Online Service Server completes the document digital signing process, verify the result and acknowledges “iAM Smart” System the digital signing result using “iAM Smart” API with the same “businessID” of the digital signing request.
Location: ResponseSigningController.java (repsonseSigning)

```java
@PostMapping(Constants.API_SIGNING_CALLBACK)
public ResponseEntity<Object> responseSigning(@RequestBody CallbackRespBase<String> body) {
    String secretKey = body.getSecretKey();
    String content = body.getContent();

    System.out.println("Signing response");

    if (secretKey == null || content == null || secretKey.isEmpty() || content.isEmpty()) {…

    System.out.println("Secret Key ok");

    byte[] cek = Security.decryptCek(secretKey);
    String respJson = Security.decrypt(content, cek);
    if (respJson == null || respJson.isEmpty()) {…
    System.out.println("CEK ok");
```

Get CEK first

Try to read the content inside the response json

```java
ApiRespSuccess<CallbackContentSigning> resp = null;
try {
    resp = objectMapper.readValue(respJson, new TypeReference<>() {
    });
} catch (JsonProcessingException e) {
    e.printStackTrace();
}
```

API – Acknowledge Digital Signing Result – 2

Put in the businessID for looking for the same request

Create a X509Certificate instance

Check validity

Verify the signature with the original documents

Call the API for the acknowledgement

```
// Verify signature
try {
    // Prepare response
    ObjectNode jsonObj = objectMapper.createObjectNode();
    jsonObj.put("businessID", resp.getContent().getBusinessId());

    // read cert byte
    byte[] certByte = Base64.getDecoder().decode(resp.getContent().getCert());
    // read signature byte
    byte[] signatureByte = Base64.getDecoder().decode(resp.getContent().getSignature());
    if (signatureByte == null) {···

    // create certificate
    CertificateFactory f = CertificateFactory.getInstance("X.509");
    X509Certificate cert = (X509Certificate) f.generateCertificate(new ByteArrayInputStream(certByte));
```

```
    // Check date
    cert.checkValidity(); // check now
    cert.checkValidity(new Date(resp.getContent().getTimestamp())); // check timestamp from iam

    // get public key from cert
    PublicKey publicKey = cert.getPublicKey();

    // get original document
    byte[] documentBytes = docxResource.getInputStream().readAllBytes();
    // Verify it
    boolean isCorrect = Security.verifySignature(publicKey, cert.getSigAlgName(), signatureByte, documentBytes);
    if (!isCorrect) {···

    // Validation success
    jsonObj.put("signingResult", "SR001");
    callApi(Constants.API_SIGNING_ACK_RESULT, jsonObj, cek);
```

Anonymous Digital Signing

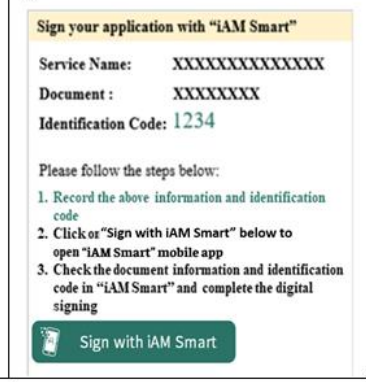
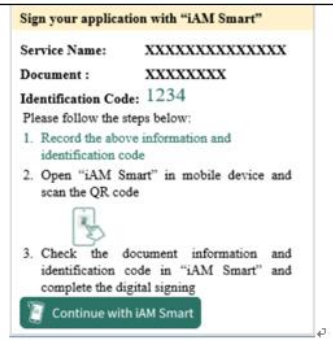
What is Anonymous Digital Signing

Anonymous digital signing allows user to perform digital signing without service login but only perform authentication before signing steps. Users can use "iAM Smart" to perform digital signing in accordance with the Electronic Transactions Ordinance (Chapter 553 of the Laws of Hong Kong) to process legal documents and procedures online.

UI Requirements – 1

After an online service sends the digital signing request to “iAM Smart” system, the online service is required to show the online service name, document name, identification code and instruct user to open the “iAM Smart” mobile app in his/her mobile phone.

By following the format on the right, developers should apply language changes for different users considering to the adaptability. Developers may follow the format shown on the next slide.

	Same Device – 1	Different Device – 2
Example	1) For Online Service Website in Same Device  2) For Online Service App in Same Device 	

UI Requirements – 2

Here are the instruction formats of different languages of Anonymous Digital Signing:

	Same Device – 1	Different Device – 2
Instructions		
Traditional Chinese	1) For Online Service Website in Same Device 請按照以下步驟: 1. 記錄上述文件資料及識別碼 2. 點擊以下「以智方便簽署」按鈕來造訪智方便認證頁面 3. 點擊開啟你手機上的「智方便」應用程式 4. 核對「智方便」內顯示的文件資料及識別碼，然後完成數碼簽署 以智方便簽署 2) For online service App in Same Device 請按照以下步驟: 1. 記錄上述文件資料及識別碼 2. 點擊以下「以智方便簽署」按鈕來開啟你手機上的「智方便」應用程式 3. 核對「智方便」內顯示的文件資料及識別碼，然後完成數碼簽署 以智方便簽署	請按照以下步驟: 1. 記錄上述文件資料及識別碼 2. 點擊以下「以智方便繼續」按鈕來開啟你手機上的「智方便」應用程式 3. 核對「智方便」內顯示的文件資料及識別碼，然後完成數碼簽署 以智方便繼續

	Same Device – 1	Different Device – 2
English	1) For Online Service Website in Same Device Please follow the steps below: 1. Record the above information and identification code 2. Click on "Sign with iAM Smart" below to access the broker page 3. Click to open "iAM Smart" mobile app 4. Check the document information and identification code in "iAM Smart" and complete the digital signing Sign with iAM Smart 2) For Online Service App in Same Device Please follow the steps below: 1. Record the above information and identification code 2. Click on "Sign with iAM Smart" below to open "iAM Smart" mobile app 3. Check the document information and identification code in "iAM Smart" and complete the digital signing Sign with iAM Smart	Please follow the steps below: 1. Record the above information and identification code 2. Open "iAM Smart" in mobile device and scan the QR code 3. Check the document information and identification code in "iAM Smart" and complete the digital signing Continue with iAM Smart

API – Initiate Anonymous Digital Signing – 1

Obtain all the parameters and the CEK as variable of calling API.
Location: RequestSigningController.java (requestAnonymousSigning)

Compute the hashcode by encoding with SHA-256 method to get a Base64 encoded value

Construct the JSON

Initiate digital signing with the JSON and the CEK

```java
@GetMapping("/anonymous/signing/initiate")
public ResponseEntity<Result<ApiRespSuccess<ApiContentRequest>>> requestAnonymousSigning(
        @RequestParam(value = "hkicHash", defaultValue = "") String hkic
    ){
    //if (source.isEmpty()) {
        //return ResponseEntity.status(HttpStatus.BAD_REQUEST).body(new Result<>("source not exist", null));
    //}
    ObjectMapper objectMapper = new ObjectMapper();
    byte[] cek = cekObj.getPrivateKey();
    ObjectNode jsonObj = objectMapper.createObjectNode();
    // Prepare document hash
    String hashCode = "";

    try {
        byte[] documentBytes = docxResource.getInputStream().readAllBytes();
        hashCode = Base64.getEncoder().encodeToString(Security.digestToByte(documentBytes, "SHA-256"));
        System.out.println(hashCode);
    } catch (IOException e) {
        e.printStackTrace();
    }
    // Record current business ID
    String uuid = UUID.randomUUID().toString().replace("-", "");
    Constants.businessIdAnonymousFormFilling.add(uuid);
    jsonObj.put("businessID", uuid);
    jsonObj.put("hashCode", hashCode);
    jsonObj.put("sigAlgo", "NONEwithRSA");
    jsonObj.put("HKICHash", hkic);
    jsonObj.put("department", "DPO");
    jsonObj.put("serviceName", "DPO");
    jsonObj.put("documentName", "Anonymous Digital Signing demo");
    System.out.println("Business ID Added: " + Constants.businessIdAnonymousFormFilling.toString());
    String respJson = callApi(Constants.API_ANONYMOUS_SIGNING_REQUEST, jsonObj, cek);
    ApiRespDecrypted<ApiContentRequest> resp = new ApiRespDecrypted<>(respJson, cek);

    String respCode = resp.getCode();
    if (respCode != null && respCode.equals(Constants.SUCCESS_CODE)) {
        return ResponseEntity.ok(new Result<>("SUCCESS", resp, uuid));
    }

    return ResponseEntity.status(HttpStatus.BAD_REQUEST).body(new Result<>("iAM Smart error", resp));
}
```

API – Initiate Anonymous Digital Signing – 2

After calling the API, it needs to calculate the identification code for acknowledgement.
Location: Security.java (getSignCode)

Calling the method inside security.java

```
String respJson = callApi(Constants.API_ANONYMOUS_SIGNING_PDF_REQUEST, jsonObj, cek);

ApiRespSigning<ApiContentRequest> resp = new ApiRespSigning<>(respJson, cek,
        Security.get4DigitsAnonymousSignCode(docDigest, hkic, clientId));

String respCode = resp.getCode();
if (respCode != null && respCode.equals(Constants.SUCCESS_CODE)) {
    return ResponseEntity.ok(new Result<>("SUCCESS", resp));

eEntity.status(HttpStatus.BAD_REQUEST).body(new Result<>("iAM Smart error", resp));
```

```
public static String get4DigitsAnonymousSignCode(String hashCode, String hkicHash, String clientId) {
    // Decode hashCode from Base64
    byte[] hashCodeBytes = Base64.getDecoder().decode(hashCode);
    byte[] hkicHashBytes = Base64.getDecoder().decode(hkicHash);
    byte[] clientIdBytes = digestToByte(clientId, "SHA-512");
    char[] code = new char[4];

    byte[] source = new byte[hashCodeBytes.length + hkicHashBytes.length + clientIdBytes.length];

    System.arraycopy(hashCodeBytes, 0, source, 0, hashCodeBytes.length);
    System.arraycopy(hkicHashBytes, 0, source, hkicHashBytes.length, hkicHashBytes
    System.arraycopy(clientIdBytes, 0, source, hashCodeBytes.length + hki
                                        .length, clientIdBytes.length);

    byte[] inData = digestToByte(source, "SHA-512");

    byte[] digestMD5Data = digestToByte(inData, "MD5");

    for (int i = 0; i < 4; i++) {
        code[i] = hexDigits[(digestMD5Data[i * 4] >>> 4 & 0xf) % 10];
    }

    return new String(code);
}
```

Perform the SHA-512 calculation

Perform the MD5 calculation

Final step to assembly into 4-digit code

API - GetQR

GetQR would be a usual API for authenticate user by calling out a broker page or QR page. After authorization by user, the page will be redirected to the **redirectURI** with **authCode** and **state** parameters.
Location: GetQrUriController.java (getQrUri)

state

redirectURI

redirectURI with authCode and state

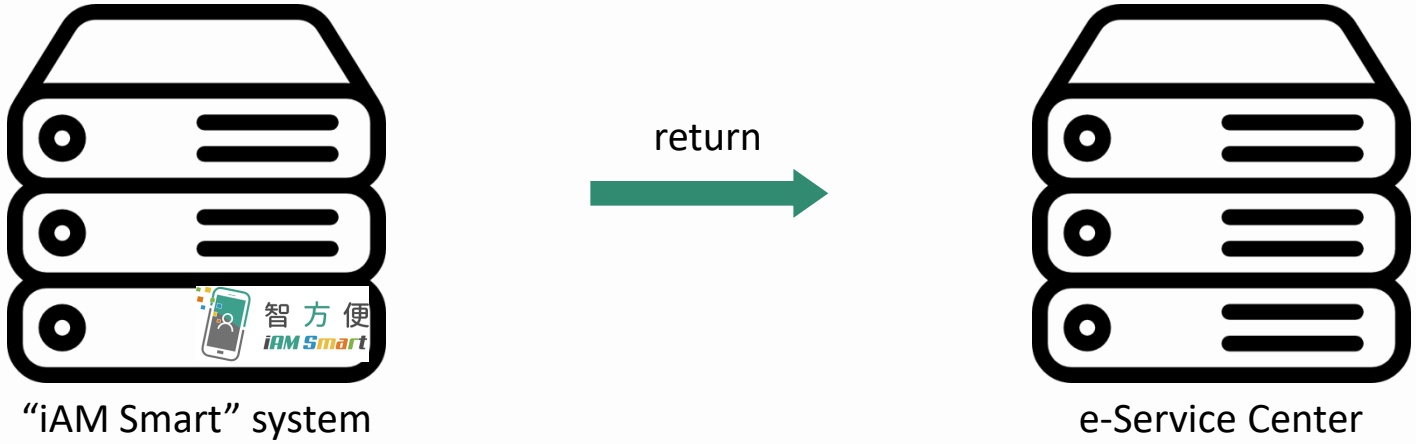
```
String state = UUID.randomUUID().toString().replace("-", "");
boolean isStoredState = storeState(state);
while (!isStoredState) {
    state = UUID.randomUUID().toString().replace("-", "");
    isStoredState = storeState(state);
}

String platform = source.split("_")[0];
boolean isBrokerPage = platform.equals(Constants.SOURCE_PLATFORM_ANDROID) ||
    platform.equals(Constants.SOURCE_PLATFORM_IOS);

String prefix = platform.equals(Constants.SOURCE_PLATFORM_APP) ?
    (ticketID.isEmpty() ? iamUrlScheme + Constants.API_GET_QR_APP :
    iamUrlScheme + Constants.API_ANONYMOUS_FORM_FILLING_REQUEST_APP) :
    iamDomain + Constants.API_GET_QR_WEB;
String scope = URLEncoder.encode(ticketID.isEmpty() ? Constants.TOKEN_SCOPE :
    Constants.TOKEN_SCOPE_ANON, StandardCharsets.UTF_8);
String redirectUri = URLEncoder.encode(redirect, StandardCharsets.UTF_8);

String uri = prefix + "responseType=code&scope=" + scope + "&clientID=" +
    clientId + "&source=" + source + "&redirectURI=" + redirectUri +
    "&state=" + state;
```

API – Anonymous Digital signing Callback from "iAM Smart" Server

"iAM Smart" returns the result to e-Service Center

User could accept or reject the digital signing request

"iAM Smart" System invokes Online Service callback API to return the result with "businessID" of the digital signing request to Online Service server. API data decryption is required.



API – RequestToken

The Anonymous Digital Signing would only request the access token for **ONCE**.

accessToken and openID request only in signing initiate stage

Different from form filling, the anonymous digital signing would convert the token in the response controller.
Location:
ResponseSigningController.java
(initiateAnonymousSigningCallBack)

```java
@GetMapping("/signing/initiate/{state}")
public ResponseEntity<Result<ApiRespSuccess<ApiContentRequest>>> initiateSigning(
        @RequestParam(value = "source", defaultValue = "") String source,
        @PathVariable(value = "state") String state) {

    if (source.isEmpty()) {
        return ResponseEntity.status(HttpStatus.BAD_REQUEST).body(new Result<>("source not exist", null));
    }

    if (state.isEmpty()) {
        return ResponseEntity.status(HttpStatus.BAD_REQUEST).body(new Result<>("state not exist", null));
    }

    IamResponseToken iamResponseToken = findTokenByState(state);
    if (iamResponseToken == null) {
        return ResponseEntity.status(HttpStatus.BAD_REQUEST).body(new Result<>("state invalid", null));
    }

    String accessToken = iamResponseToken.getAccessToken();
    String openId = iamResponseToken.getOpenId();

    if (accessToken == null || openId == null || accessToken.isEmpty() || openId.isEmpty()) {
        return ResponseEntity.status(HttpStatus.BAD_REQUEST).body(new Result<>("state invalid", null));
    }
```

RequestSigningController.java (initiateSigning)

accessToken

openId

```java
JsonNode respNode = objectMapper.convertValue(resp, JsonNode.class);
JsonNode respNodeContent = respNode.get("content");
ApiContentToken content = objectMapper.convertValue(respNodeContent, ApiContentToken.class);
```

API – Acknowledge Digital Signing Result – 1

Online Service Server completes the document digital signing process, verifies the result and acknowledges “iAM Smart” System the digital signing result using “iAM Smart” API with the same “businessID” of the digital signing request.
Location: ResponseSigningController.java (repsonseSigning)

```java
@PostMapping(Constants.API_SIGNING_CALLBACK)
public ResponseEntity<Object> responseSigning(@RequestBody CallbackRespBase<String> body) {
    String secretKey = body.getSecretKey();
    String content = body.getContent();

    System.out.println("Signing response");

    if (secretKey == null || content == null || secretKey.isEmpty() || content.isEmpty()) {…

    System.out.println("Secret Key ok");

    byte[] cek = Security.decryptCek(secretKey);
    String respJson = Security.decrypt(content, cek);
    if (respJson == null || respJson.isEmpty()) {…
    System.out.println("CEK ok");
```

Get CEK first

Try to read the content inside the response json

```java
ApiRespSuccess<CallbackContentSigning> resp = null;
try {
    resp = objectMapper.readValue(respJson, new TypeReference<>() {
    });
} catch (JsonProcessingException e) {
    e.printStackTrace();
}
```

API – Acknowledge Digital Signing Result – 2

Put in the businessID for looking for the same request

Create a X509Certificate instance

Check validity

Verify the signature with the original documents

Call the API for the acknowledgement

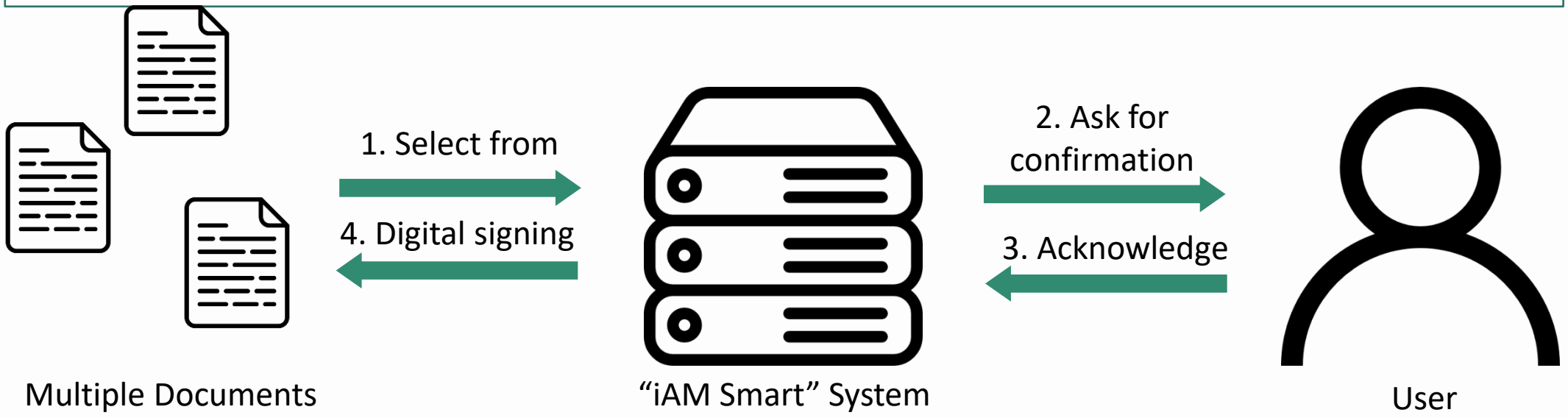
```
// Verify signature
try {
    // Prepare response
    ObjectNode jsonObj = objectMapper.createObjectNode();
    jsonObj.put("businessID", resp.getContent().getBusinessId());

    // read cert byte
    byte[] certByte = Base64.getDecoder().decode(resp.getContent().getCert());
    // read signature byte
    byte[] signatureByte = Base64.getDecoder().decode(resp.getContent().getSignature());
    if (signatureByte == null) {···

    // create certificate
    CertificateFactory f = CertificateFactory.getInstance("X.509");
    X509Certificate cert = (X509Certificate) f.generateCertificate(new ByteArrayInputStream(certByte));
```

```
    // Check date
    cert.checkValidity(); // check now
    cert.checkValidity(new Date(resp.getContent().getTimestamp())); // check timestamp from iam

    // get public key from cert
    PublicKey publicKey = cert.getPublicKey();

    // get original document
    byte[] documentBytes = docxResource.getInputStream().readAllBytes();
    // Verify it
    boolean isCorrect = Security.verifySignature(publicKey, cert.getSigAlgName(), signatureByte, documentBytes);
    if (!isCorrect) {···

    // Validation success
    jsonObj.put("signingResult", "SR001");
    callApi(Constants.API_SIGNING_ACK_RESULT, jsonObj, cek);
```

Bulk Digital Signing with Service Login

What is Bulk Digital Signing with Service Login

Online Services can make use of Bulk Digital Signing API to enable “iAM Smart” users complete digital signing online for multiple documents with only one digital signing cycle. It can be used in many cases, such as bulk digital signing online application form and bulk digital signing contract and agreement. The function can be performed after the service login.

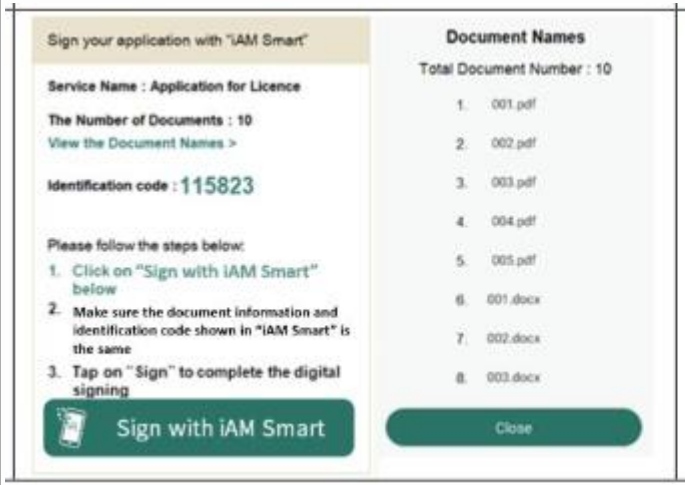
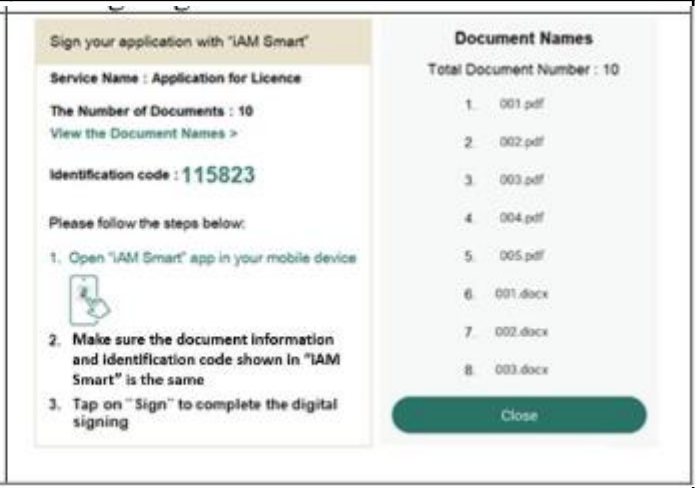


UI Requirements – 1

1. The requirement of sign with “iAM Smart”

	Same Device	Different Device
Instructions		
Traditional Chinese	請按照以下步驟： 1. 點擊以下「以智方便簽署」按鈕來開啟你手機上的「智方便」應用程式 2. 請確保「智方便」顯示的文件資料及識別碼相同 3. 點擊「簽署」以完成數碼簽署 以智方便簽署	請按照以下步驟： 1. 開啟你手機上的「智方便」應用程式 2. 請確保「智方便」顯示的文件資料及識別碼相同 3. 點擊「簽署」以完成數碼簽署
Simplified Chinese	请按照以下步骤： 1. 点击以下「以智方便签署」按钮来开启你手机上的「智方便」应用程式 2. 请确保「智方便」显示的文件资料及识别码相同 3. 点击「签署」以完成数码签署 以智方便签署	请按照以下步骤： 1. 开启你手机上的「智方便」应用程式 2. 请确保「智方便」显示的文件资料及识别码相同 3. 点击「签署」以完成数码签署
English	Please follow the steps below: 1. Click on “Sign with iAM Smart” below 2. Make sure the document information and identification code shown in “iAM Smart” is the same 3. Tap on “Sign” to complete the digital signing Sign with iAM Smart	Please follow the steps below: 1. Open “iAM Smart” app in your mobile device 2. Make sure the document information and identification code shown in “iAM Smart” is the same 3. Tap on “Sign” to complete the digital signing

2. Instruction page with identification code

Same Device	Different Device
	

UI Requirements – 2

3. Two Signing results:
- Request status is completed
  - The bulk digital signing with service login is finished. User could either download the documents or leave this page.
- Request status is cancelled
  - The bulk digital signing with service login is failed. User could leave this page to have another attempt.

API – Initiate Bulk Digital Signing

Obtain all the parameters and the CEK as variable of calling API.
Location: RequestSigningController.java (initiateBulkSigning)

Get accessToken and openId with service login

Construct the JSON

Initiate digital signing with the JSON and the CEK

```java
@GetMapping("/signing/bulk/initiate/{state}")
public ResponseEntity<Result<ApiRespSuccess<ApiContentRequest>>> initiateBulkSigning(
        String openId = iamResponseToken.getOpenId();

        if (accessToken == null || openId == null || accessToken.isEmpty() || openId.isEmpty()) {
            return ResponseEntity.status(HttpStatus.BAD_REQUEST).body(new Result<>("state invalid", null));
        }

        ObjectMapper objectMapper = new ObjectMapper();
        byte[] cek = cekObj.getPrivateKey();
        ObjectNode jsonObj = objectMapper.createObjectNode();
        String[] doc = {"1","2","8","9"};
        ArrayList<ObjectNode> DocumentList = getDocument(doc);

        if(DocumentList!=null) {
        String uuid = UUID.randomUUID().toString().replace("-", "");
        Constants.businessIdAnonymousFormFilling.add(uuid);
        System.out.println("State: "+state);
        //jsonObj.put("Content-Type", "application/json");
        jsonObj.put("businessID", uuid);
        jsonObj.put("accessToken", accessToken);
        jsonObj.put("openID", openId);
        jsonObj.put("source", source);
        jsonObj.put("state", state);
       // jsonObj.put("serverRedirectURI", iamCallbackEndpoint + Constants.API_SIGNING_BULK_SERVER_CALLBACK );
        //jsonObj.put("department", "DPO");
        jsonObj.put("serviceName", "Demo Service (Internal Test)");
        jsonObj.put("maxCallbackSigs", 100);
        jsonObj.put("redirectURI", iamCallbackEndpoint + Constants.API_SIGNING_BULK_SERVER_CALLBACK_TOKEN);
        jsonObj.put("callbackResultURI", iamCallbackEndpoint + Constants.API_SIGNING_BULK_SERVER_CALLBACK +"/"+state) ;
        jsonObj.put("requestName", "DPO");
        jsonObj.putArray("documents").addAll(DocumentList); //call getDocuments Function

        System.out.println("Request Bulk signing: "+jsonObj.toString());
        String respJson = callApi(Constants.API_SIGNING_BULK_REQUEST, jsonObj, cek);
        ApiRespDecrypted<ApiContentRequest> resp = new ApiRespDecrypted<>(respJson, cek);

        String respCode = resp.getCode();
        System.out.println("resp"+resp.getMessage());
        if (respCode != null && respCode.equals(Constants.SUCCESS_CODE)) {

            return ResponseEntity.ok(new Result<>("SUCCESS", resp, uuid));
        }


       return ResponseEntity.status(HttpStatus.BAD_REQUEST).body(new Result<>("iAM Smart error", resp));
    }
        return null;
}
```

API – Open "iAM Smart" App For Bulk Digital Signing

Online Service App invokes “iAM Smart” Mobile App using URL Scheme with request parameters.

The context links deeply to redirect users to the “iAM Smart” mobile app.

Using the same device as an example

Online service app

<context>

“iAM Smart” mobile app

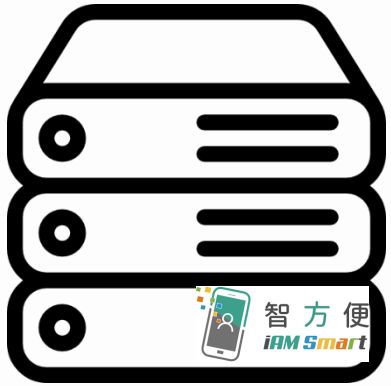
The difference is bulk digital signing allows users to sign multiple documents at once.

API – Bulk Digital Signing Callback from "iAM Smart" Server

"iAM Smart" returns the result to e-Service Center

User could accept or reject the digital signing request

"iAM Smart" System invokes Online Service callback API to return the result with "businessID" of the digital signing request to Online Service server. API data decryption is required.



"iAM Smart" system

return

e-Service Center

API – Acknowledge Bulk Digital Signing Result – 1

Online Service Server completes the bulk digital signing process, verifies the result and acknowledges “iAM Smart” System the digital signing result using “iAM Smart” API with the same “businessID” of the digital signing request.
Location: ResponseSigningController.java (BulksSigningCallBack)

```
public ResponseEntity<Object>BulksSigningCallBack(
        @RequestPart("sigFile") MultipartFile sigFile,
        @RequestPart("sigMetada") CallbackRespBase<String> sigMetadata,
        @PathVariable(value = "state") String state) {
    //System.out.println("sigMetada: "+sigMetadata);
    //System.out.println(sigFile);
    String secretKey = sigMetadata.getSecretKey();
    String constent = sigMetadata.getContent();
    byte[] sk = Security.decryptCek(secretKey);
    byte[] cek = cekObj.getPrivateKey();
    String respJson = Security.decrypt(constent, cek);
```

Get CEK first

Save information of content, code, message, txid into result

```
ObjectMapper objectMapper = new ObjectMapper();
ApiRespSuccess<sigMetaData> resp = null;
ApiRespSuccess<CallbackContentBulkSigning> result = new ApiRespSuccess<CallbackContentBulkSigning>();

try {
    resp = objectMapper.readValue(respJson, new TypeReference<>() {});
    resp.getContent().setState(state);
    result.setContent(new CallbackContentBulkSigning(resp.getContent()));
    //result.getContent().setSigMetaData(resp.getContent());
    result.setCode(resp.getCode());
    result.setMessage(resp.getMessage());
    result.setTxId(resp.getTxId());
} catch (JsonProcessingException e) {
    e.printStackTrace();
}
```

API – Acknowledge Bulk Digital Signing Result – 2

Convert MultipartFile to String

Store the signing result

Put the essential information into jsonObj

Call the API to acknowledge the bulk digital signing result

```
if (resp == null) {
    return ResponseEntity.status(HttpStatus.BAD_REQUEST).body(null);
}
try {
    // Convert MultipartFile to String
    String sigFileContent = FileUtil.convertMultipartFileToString(sigFile);
    String respJson1 = Security.decrypt(sigFileContent, cek);
    JSONObject jsonArray = new JSONObject(respJson1);
    JSONArray signature = jsonArray.getJSONArray("signatures");
    sigFile sigfile = null;
    for (int i = 0; i < signature.length();i++) {
        sigfile = objectMapper.readValue(signature.getString(i), new TypeReference<>() {});
        result.getContent().addSigFile(sigfile);
    }
     storeBulkSigning(objectMapper.writeValueAsString(result), result);

} catch (Exception e) {
    //System.out.println(e.getMessage());
    return ResponseEntity.status(400).body("Error processing callback: " + e.getMessage());

}
```

```
ObjectNode jsonObj = objectMapper.createObjectNode();
String businessID = result.getContent().getSigMetatData().getBusinessID();


if (businessID.isEmpty()) return ResponseEntity.status(HttpStatus.BAD_REQUEST).body(new Result<>("source not exist", null));
jsonObj.put("businessID", businessID);
jsonObj.put("signingResult", "SR001");

String ackResultrespJson = callApi(Constants.API_SIGNING_BULK_ACKRESULT, jsonObj, cek);
ApiRespDecrypted<ApiContentRequest> ackResultresp = new ApiRespDecrypted<>(ackResultrespJson, cek);

String ackResultrespCode = ackResultresp.getCode();
System.out.println("resp"+ackResultresp.getMessage());
if (ackResultrespCode != null && ackResultrespCode.equals(Constants.SUCCESS_CODE)) {

    return ResponseEntity.ok(new Result<>("SUCCESS", resp));
}

return null;
```

API – Enquire Bulk Digital Signing Status

Put necessary information into jsonObj

Calling Api to get the status

Decryption and Convertion

Return the signing result

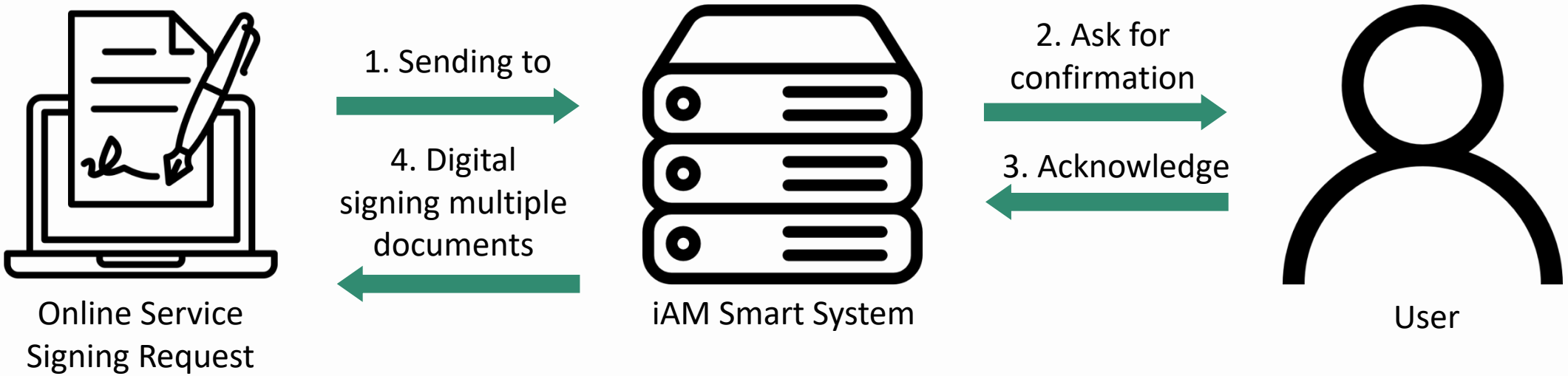
```
//Enquire Bulk Digital Signing Status
jsonObj = objectMapper.createObjectNode();
jsonObj.put("BSQCToken",respResult.getBSQCToken());
jsonObj.put("openID",content.getOpenId());
String respStatusResultJson = callApi(Constants.API_SIGNING_BULK_STATUS, jsonObj, cek);
respSuccess = null;
try {
    respSuccess = objectMapper.readValue(respStatusResultJson, new TypeReference<>() {
    });
} catch (JsonProcessingException e) {
    e.printStackTrace();
}
respDecryptedResultJson = Security.decrypt(respSuccess.getContent(), cek);
if (respDecryptedResultJson == null || respDecryptedResultJson.isEmpty()) {
    return ResponseEntity.status(HttpStatus.BAD_REQUEST).body(null);
}
System.out.println(respDecryptedResultJson);
CallbackContentBulkSigningStatus respStatusResult = null;
try {
    respStatusResult = objectMapper.readValue(respDecryptedResultJson, new TypeReference<>() {
    });
} catch (JsonProcessingException e) {
    e.printStackTrace();
}

if (respStatusResult == null) {
    return ResponseEntity.status(HttpStatus.BAD_REQUEST).body(null);
}
System.out.println(respStatusResult.getStatus()+","+respStatusResult.toString());
respResult.setSigningStatus(respStatusResult.toString());

return ResponseEntity.ok(new Result<>("SUCCESS", respResult));
```

Location: ResponseSigningController.java
(initiateAnonymousBulkSigningCallBack)

Anonymous Bulk Digital Signing

What is Anonymous Bulk Digital Signing

Similar to Bulk Digital Signing API, online service can request “Anonymous Bulk Digital Signing” API to complete digital signing online for multiple documents with only one digital signing cycle. Unlike the accessToken obtained from Authentication API, the accessToken received in this API can only be used once. It can be used in many cases, such as digital signing online application forms and digital signing contracts and agreements.



UI Requirements – 1

1. The requirement of sign with “iAM Smart” in Chinese(Traditional)

Same Device - 1	Different Device - 2

Instructions

1) For Online Service Website in Same Device
請按照以下步驟:
1. 記錄上述文件資料及識別碼
2. 點擊以下「以智方便簽署」按鈕來造訪智方便認證頁面
3. 點擊開啟你手機上的「智方便」應用程式
4. 核對「智方便」內顯示的文件資料及識別碼，然後完成數碼簽署

以智方便簽署

2) For Online Service App in Same Device
請按照以下步驟:
1. 記錄上述文件資料及識別碼
2. 點擊以下「以智方便簽署」按鈕來開啟你手機上的「智方便」應用程式
3. 核對「智方便」內顯示的文件資料及識別碼，然後完成數碼簽署

以智方便簽署

請按照以下步驟:
1. 記錄上述文件資料及識別碼
2. 點擊以下「以智方便繼續」按鈕來開啟你手機上的「智方便」應用程式
3. 核對「智方便」內顯示的文件資料及識別碼，然後完成數碼簽署

以智方便繼續

The requirement of sign with “iAM Smart” in English

Same Device - 1	Different Device - 2

1) For Online Service Website in Same Device

Please follow the steps below:
1. Record the above information and identification code
2. Click on “Sign with iAM Smart” below to access the broker page
3. Click to open “iAM Smart” mobile app
4. Check the document information and identification code in “iAM Smart” and complete the digital signing

Sign with iAM Smart

2) For Online Service App in Same Device

Please follow the steps below:
1. Record the above information and identification code
2. Click on “Sign with iAM Smart” below to open “iAM Smart” mobile app
3. Check the document information and identification code in “iAM Smart” and complete the digital signing

Sign with iAM Smart

Please follow the steps below:
1. Record the above information and identification code
2. Open “iAM Smart” in mobile device and scan the QR code
3. Check the document information and identification code in “iAM Smart” and complete the digital signing

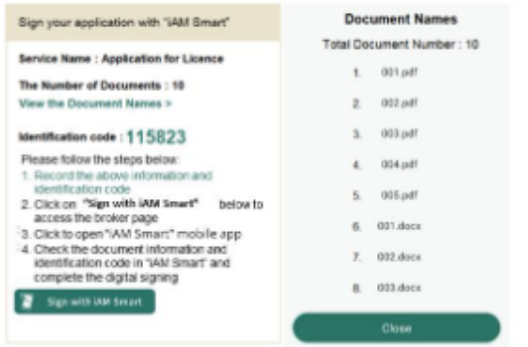
Continue with iAM Smart

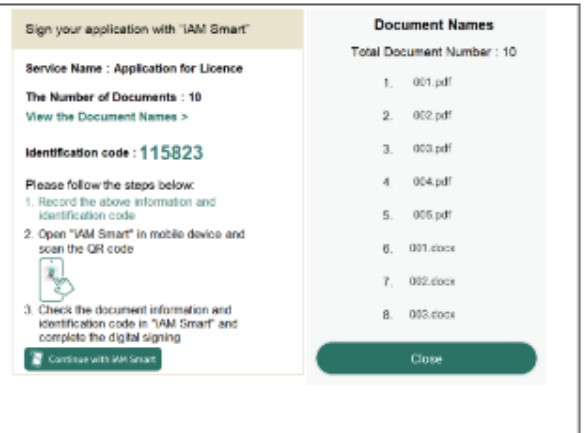
UI Requirements – 2

2. Instruction page with identification code

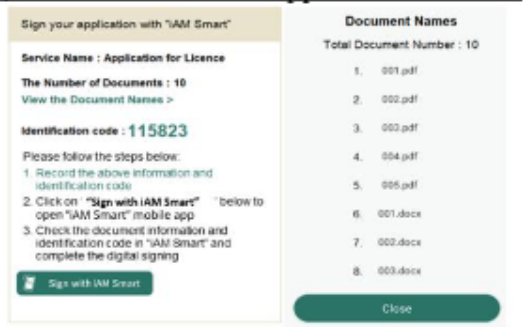
Same Device - 1	Different Device - 2

1) For Online Service Website in Same Device





2) For Online Service App in Same Device



3. Signing results:

API – Initiate Anonymous Bulk Digital Signing

Obtain all the parameters and the CEK as variable of calling API.
Location: RequestSigningController.java (initiateAnonymousBulkSigning)

Example Documents

Construct the JSON

Initiate digital signing with the JSON and the CEK

```java
@GetMapping("/anonymous/signing/bulk/initiate")
    public ResponseEntity<Result<ApiRespSuccess<ApiContentRequest>>> initiateAnonymousBulkSigning(
            @RequestParam(value = "docList", defaultValue = "") String[] docList,
            @RequestParam(value = "hkicHash", defaultValue = "") String hkic,
            @RequestParam(value = "type", defaultValue = "") String type,
            @RequestParam(value = "callbackResultURL", defaultValue = "") String callbackResultURL
            ){
        ObjectMapper objectMapper = new ObjectMapper();
        byte[] cek = cekObj.getPrivateKey();
        ObjectNode jsonObj = objectMapper.createObjectNode();
        String[] doc = {"1","2","8","9"};
        ArrayList<ObjectNode> DocumentList = getDocument(doc);

        if(DocumentList!=null) {
            String uuid = UUID.randomUUID().toString().replace("-", "");
            Constants.businessIdAnonymousFormFilling.add(uuid);
            jsonObj.put("businessID", uuid);
            jsonObj.put("HKICHash", "SZ2WPeWp3d6smN7SRe51OKUj6Ck1GQ3GdNd3U7hku2Q=");
            jsonObj.put("serviceName", "Demo Service (Internal Test)");
            jsonObj.put("maxCallbackSigs", 100);
            jsonObj.put("callbackResultURI", iamCallbackEndpoint + Constants.API_ANONYMOUS_SIGNING_BULK_CALLBACK) ;
            jsonObj.put("requestName", "DPO");
            jsonObj.putArray("documents").addAll(DocumentList); //call getDocuments Function

            System.out.print(jsonObj);
            String respJson = callApi(Constants.API_ANONYMOUS_SIGNING_BULK_REQUEST, jsonObj, cek);
            ApiRespDecrypted<ApiContentRequest> resp = new ApiRespDecrypted<>(respJson, cek);

            String respCode = resp.getCode();
            if (respCode != null && respCode.equals(Constants.SUCCESS_CODE)) {

                return ResponseEntity.ok(new Result<>("SUCCESS", resp, uuid));
            }

            return ResponseEntity.status(HttpStatus.BAD_REQUEST).body(new Result<>("iAM Smart error", resp));
        }
        return null;
    }
```

API – Open "iAM Smart" App For Anonymous Bulk Digital Signing

Online Service App invokes “iAM Smart” Mobile App using URL Scheme with request parameters.

The context links deeply to redirect users to the “iAM Smart” mobile app.

Using the same device as an example

Online service app

<context>

“iAM Smart” mobile app

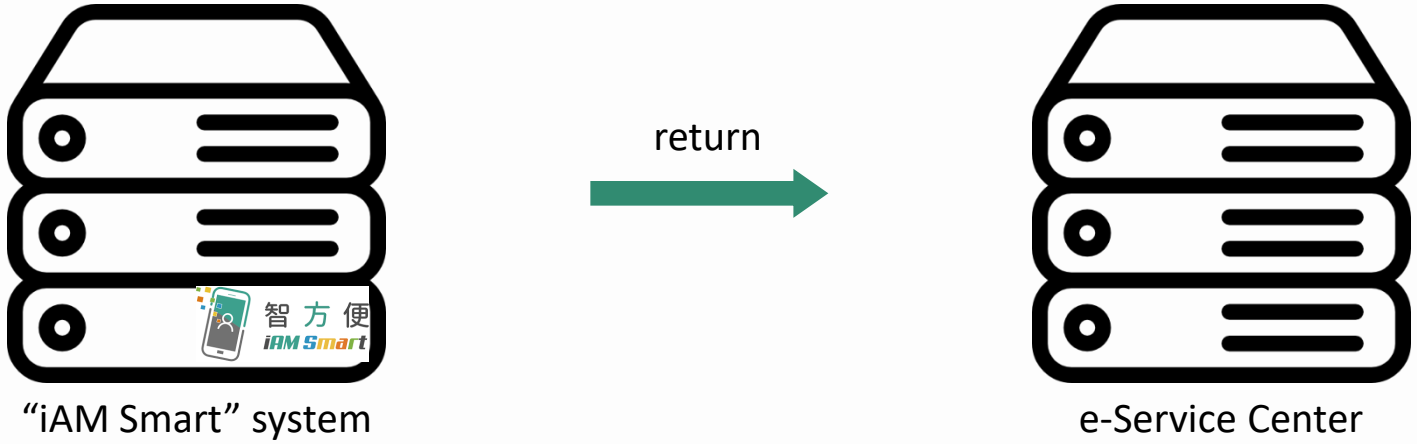
The difference is anonymous bulk digital signing allows users to sign multiple documents at once.

API – Anonymous Bulk Digital Signing Callback from "iAM Smart" Server

"iAM Smart" returns the result to e-Service Center

User could accept or reject the digital signing request

"iAM Smart" System invokes Online Service callback API to return the result with "businessID" of the digital signing request to Online Service server. API data decryption is required.



API – Acknowledge Anonymous Bulk Digital Signing Result – 1

Online Service Server completes the bulk digital signing process, verifies the result and acknowledges "iAM Smart" System the digital signing result using "iAM Smart" API with the same "businessID" of the digital signing request.
Location: ResponseSigningController.java (initiateAnonymousBulkSigningCallBack)

```
//Callback
@GetMapping("/anonymous/signing/bulk/callback")
    public ResponseEntity<Result<CallbackContentBSQCToken>> initiateAnonymousBulkSigningCallBack(
            @RequestParam(value = "businessID", defaultValue = "") String businessID,
            @RequestParam(value = "code", defaultValue = "") String code,
            @RequestParam(value = "state", defaultValue = "") String state,
            @RequestParam(value = "error_code", defaultValue = "") String error_code){
        ObjectMapper objectMapper = new ObjectMapper();
        ObjectNode jsonObj = objectMapper.createObjectNode();
        byte[] cek = cekObj.getPrivateKey();
```

Get CEK first

jsonObj create for converting value

```
jsonObj.put("code", code);
jsonObj.put("grantType", Constants.GRANT_TYPE);
System.out.println(jsonObj);
String respJson = callApi(Constants.API_GET_TOKEN, jsonObj, cek);
System.out.println("call api ok" + state);
System.out.println("businessID:"+businessID);
ApiRespDecrypted<ApiContentToken> resp = new ApiRespDecrypted<>(respJson, cek);
System.out.println("plaintext:" + resp);

1 reference
JsonNode respNode = objectMapper.convertValue(resp, JsonNode.class);
JsonNode respNodeContent = respNode.get("content");
1 reference
ApiContentToken content = objectMapper.convertValue(respNodeContent, ApiContentToken.c
```

API – Acknowledge Anonymous Bulk Digital Signing Result – 2

jsonObj create for converting value

Get BSQC Token by the jsonObj

Acknowledge bulk digital signing result

Save the signing status

```
System.out.println("resp:"+content);
System.out.println("accessToken:"+content.getAccessToken());
System.out.println("openID:"+content.getOpenId());

jsonObj = objectMapper.createObjectNode();
jsonObj.put("accessToken",content.getAccessToken());
jsonObj.put("openID", content.getOpenId());

String respResultJson = callApi(Constants.API_ANONYMOUS_SIGNING_BULK_GET_BSQC_TOKEN, jsonObj, cek);
System.out.println(respResultJson);
ApiRespSuccess<String> respSuccess = null;
try {
    respSuccess = objectMapper.readValue(respResultJson, new TypeReference<>() {
    });
} catch (JsonProcessingException e) {
    e.printStackTrace();
}
```

```
//Online Service acknowledges bulk digital signing result
jsonObj = objectMapper.createObjectNode();
jsonObj.put("businessID",businessID);
jsonObj.put("signingResult", "SR001");
String respAckResultJson = callApi(Constants.API_SIGNING_BULK_ACKRESULT, jsonObj, cek);
System.out.println(respAckResultJson);
System.out.println(respResult.getBSQCToken());
//Enquire Bulk Digital Signing Status
jsonObj = objectMapper.createObjectNode();
jsonObj.put("BSQCToken",respResult.getBSQCToken());
jsonObj.put("openID",content.getOpenId());
String respStatusResultJson = callApi(Constants.API_SIGNING_BULK_STATUS, jsonObj, cek);
respSuccess = null;
```

API – Enquire Anonymous Bulk Digital Signing Status

Put necessary information into jsonObj

Calling API to get the status

Decryption and Conversion

Return the signing result

```
//Enquire Bulk Digital Signing Status
jsonObj = objectMapper.createObjectNode();
jsonObj.put("BSQCToken",respResult.getBSQCToken());
jsonObj.put("openID",content.getOpenId());
String respStatusResultJson = callApi(Constants.API_SIGNING_BULK_STATUS, jsonObj, cek);
respSuccess = null;
try {
    respSuccess = objectMapper.readValue(respStatusResultJson, new TypeReference<>() {
    });
} catch (JsonProcessingException e) {
    e.printStackTrace();
}
respDecryptedResultJson = Security.decrypt(respSuccess.getContent(), cek);
if (respDecryptedResultJson == null || respDecryptedResultJson.isEmpty()) {
    return ResponseEntity.status(HttpStatus.BAD_REQUEST).body(null);
}
System.out.println(respDecryptedResultJson);
CallbackContentBulkSigningStatus respStatusResult = null;
try {
    respStatusResult = objectMapper.readValue(respDecryptedResultJson, new TypeReference<>() {
    });
} catch (JsonProcessingException e) {
    e.printStackTrace();
}

if (respStatusResult == null) {
    return ResponseEntity.status(HttpStatus.BAD_REQUEST).body(null);
}
System.out.println(respStatusResult.getStatus()+","+respStatusResult.toString());
respResult.setSigningStatus(respStatusResult.toString());

return ResponseEntity.ok(new Result<>("SUCCESS", respResult));
```

Location: ResponseSigningController.java
(initiateAnonymousBulkSigningCallBack)

Quiz

Question:
What is the purpose of identification code ?

A. Useless
B. For user to verify the identification code are the same before authorising the signing request
C. Reference number
D. For login

Answer: B