レシート検証プログラミングガイド

Developer

目次

図、表、リスト

レシート検証について

App または App 内課金のレシートは、App および App 内で実行された App 内課金の販売の記録です。レシートの検証コードを App に追加することで、App の不正コピーの実行を防ぐことができます。コピー保護を実装するために App ができることとできないことに関する詳細情報は、使用許諾契約書と審査のガイドラインをご覧ください。

レシート検証では、暗号と各種の安全なコーディング技術についての理解が必要です。App に特有のソリューションを採用することが重要となります。

概要

レシート検証には、ローカルで行う方法と App Store を使用する方法の 2 つがあります。この 2 つの方法を比較し、App とインフラに適した方法を選択してください。また、両方の方法を実行することも可能です。

レシートのローカル検証

ローカルで検証する場合、PKCS #7 署名を読み込んで検証するコードと、署名されたペイロードを解析して検証するコードが必要です。

関連する章: Validating Receipts Locally （6 ページ）, Receipt Fields （24 ページ）

App Store を使用したレシートの検証

App Store を使用して検証する場合、App とサーバの間に安全な接続があることと、App Store を使用してレシートを検証するためのコードがサーバ上にあることが必要です。

関連する章: Validating Receipts With the App Store （19 ページ）, Receipt Fields （24 ページ）

レシートのローカル検証

レシート検証は、Appが起動された直後、ユーザインターフェイスの表示、または子プロセスの生成が行われる前に実行します。このチェックは、`NSApplicationMain`関数が呼び出される前に、`main`関数内で実装します。セキュリティを強化するには、Appの実行中にこのチェックを定期的に繰り返します。

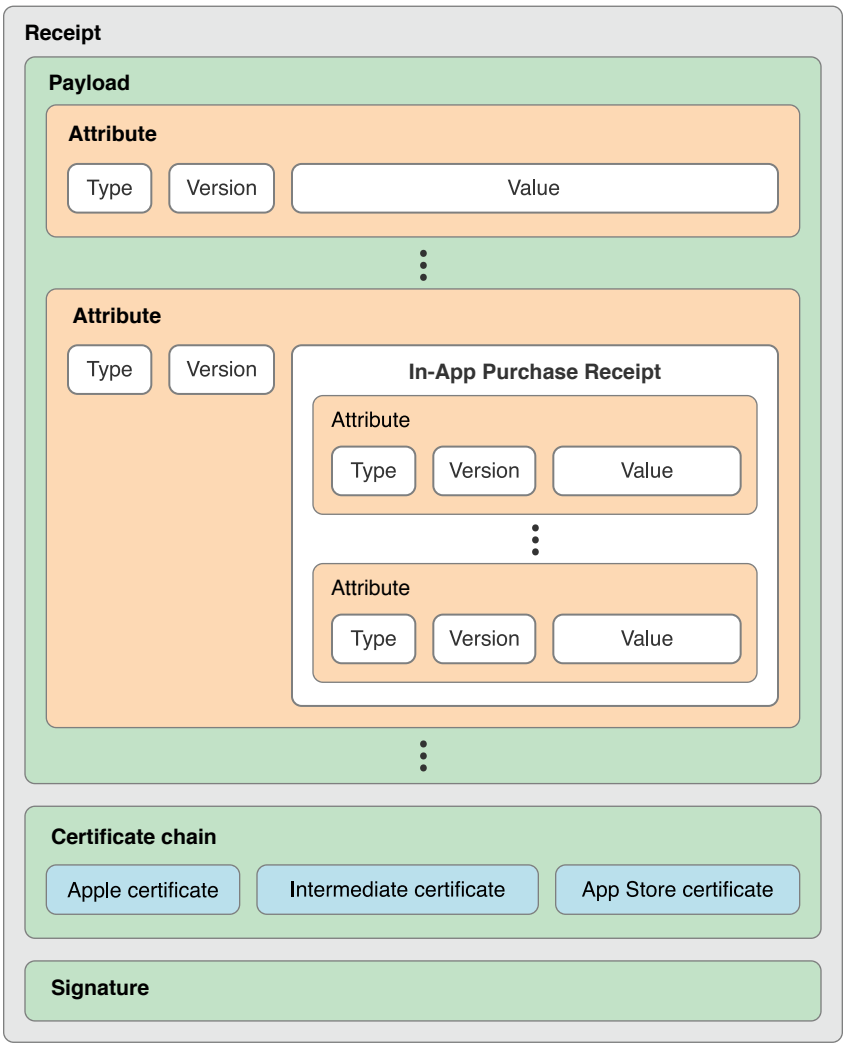
レシートがある場所の特定と解析

App Storeから Appがインストールされると、Appには Appleのみが有効なレシートを作成できることを保証する、暗号署名された Appのレシートが含まれます。レシートは Appバンドル内に格納されます。レシートの場所を特定するには、`NSBundle`クラスの`appStoreReceiptURL`メソッドを呼び出します。

> **注意：：** macOS で、システムが古いため `appStoreReceiptURL` メソッドが使用できない場合、ハードコーディングされたパスにフォールバックすることができます。レシートのパスは`/Contents/_MASReceipt/receipt`で、App バンドル内にあります。
>
> iOS で、システムが古いため `appStoreReceiptURL` メソッドが使用できない場合、App Store で`SKPaymentTransaction` オブジェクトの`transactionReceipt` プロパティの検証にフォールバックすることができます。詳細については、Validating Receipts With the App Store （19 ページ）を参照してください。

レシートは、図 1-1 に示す構造を持つバイナリファイルです。

図 1-1 レシートの構造



レシートの最も外側の部分（図中の *Receipt* と記された部分）は RFC 2315 で定義されているように、PKCS #7 のコンテナであり、そのペイロードは ITU-T X.690 で定義されているように、ASN.1（Abstract Syntax Notation One）を使用してエンコードされています。ペイロードは、*レシートの属性*のセットで構成されています。各レシートの属性には、タイプ（Type）、バージョン（Version）、値（Value）が含まれます。

ペイロードの構造は、リスト 1-1に ASN.1 記法を用いて定義されています。`asn1c` ツールを使用したこの定義を使用すれば、コードのこの部分を手作業で記述する代わりに、ペイロードをデコードするためのデータタイプ宣言と関数を生成できます。最初に`asn1c`をインストールする必要があります。MacPorts と SourceForge から入手できます。

レシートに記載されているキーについての詳細は、Receipt Fields （24 ページ）を参照してください。

コードを生成するには、リスト 1-1 に記されているペイロードの記述をファイルに保存し、ターミナル（Terminal）で次のコマンドを実行します。

```
asn1c -fnative-types filename
```

`asn1c`ツールによる現在のディレクトリ内のファイル生成が完了したら、生成されたファイルを Xcode プロジェクトに追加します。

リスト 1-1　ASN.1 における ペイロードフォーマットの定義

```
ReceiptModule DEFINITIONS ::=
BEGIN

ReceiptAttribute ::= SEQUENCE {
    type    INTEGER,
    version INTEGER,
    value   OCTET STRING
}

Payload ::= SET OF ReceiptAttribute

END
```

GUID のハッシュ値の計算

macOS では、Get the GUID in macOS （13 ページ） に記載されているメソッドを使用してコンピュータの GUID を取得します。

iOS では、`UIDevice` の`identifierForVendor` プロパティによって返された値をコンピュータの GUID として使用します。

ハッシュ値を計算するには、最初に GUID 値をオペーク値（タイプ 4 の属性）とバンドル ID に連結します。UTF-8 文字列の解釈や標準化を行わずに、レシートの生のバイトを使用します。次に、連結された一連のバイトの SHA-1 ハッシュ値を計算します。

レシートの検証

レシートを検証するには、次のテストを順番に実行します。

1. レシートの場所を特定します。

   レシートが存在しない場合、検証は失敗します。

2. レシートが Apple によって適切に署名されていることを確認します。

   Apple によって署名されていない場合、検証は失敗します。

3. レシート内のバンドル ID が、`Info.plist` ファイル内にあると予想される`CFBundleIdentifier` の値を含むハードコーディングされた定数と一致することを確認します。

   一致しない場合、検証は失敗します。

4. レシート内のバージョン ID 文字列が、`Info.plist` ファイル内にあると予想される `CFBundleShortVersionString` の値（macOSの場合）または CFBundleVersion の値（iOSの場合）を含むハードコーディングされた定数と一致することを確認します。

   一致しない場合、検証は失敗します。

5. Compute the Hash of the GUID （9 ページ）の説明に従って、GUID のハッシュ値を計算します。

   計算結果がレシート内のハッシュ値と一致しない場合、検証は失敗します。

すべてのテストに合格すると、検証は成功です。

> **注意：:** バンドル ID とバージョン ID 文字列は、単なるバイト列ではなく、UTF-8 文字列です。これを踏まえて、比較ロジックをコーディングしてください。

App が Volume Purchase Program に対応している場合、レシートの有効期限を確認します。

レシート検証の失敗への対応

検証は、さまざまな理由で失敗することがあります。たとえば、ユーザがある Mac から別の Mac に App をコピーすると、GUID が一致しなくなり、レシート検証は失敗します。

macOS で検証が失敗したら終了する

macOS で検証が失敗した場合、ステータス `173` で `exit` を呼び出します。この終了ステータスは、レシートが無効であると App が判断したことをシステムに通知します。この時点で、システムは有効なレシートを取得しようとし、ユーザの iTunes の認証情報の入力を求めることがあります。

システムは、有効なレシートを正常に取得した場合、App を再起動します。それ以外の場合は、問題を説明するエラーメッセージをユーザに表示します。

検証が失敗した場合、ユーザにエラーメッセージを表示しないでください。有効なレシートの取得を試みる、またはレシートが無効であることをユーザに通知するのは、システムの役割です。

iOS で検証が失敗したらレシートを更新する

iOS で検証が失敗した場合、`SKReceiptRefreshRequest` クラスを使用してレシートを更新します。

App を終了しないでください。ユーザに猶予期間を与えるか、App 内の機能を制限するかを選択することができます。

Mac App への最小システムバージョン（Minimum System Version）の設定

App の `Info.plist` ファイルに、10.6.6 以上の値を持つ`LSMinimumSystemVersion` キーを含めます。バージョン 10.6.6 より前の macOS でレシート検証に失敗した場合、App はユーザに説明することなく、起動後すぐに終了します。前のバージョンの macOS は終了ステータス `173` を解釈しないため、有効なレシートの取得を試みることや、エラーメッセージを表示することはありません。

バージョン番号をローカライズしない

App がローカライズされている場合、`CFBundleShortVersionString`キーは、App のいずれの `InfoPlist.strings` ファイルにも表示されていてはなりません。ローカライズされていない値は `Info.plist` ファイルからレシートに格納されます。このキーの値をローカライズしようとすると、レシート検証に失敗する場合があります。

検証チェックの保護

攻撃者は、Appのバイナリにパッチを当てる、または検証コードが依存するオペレーティングシステムの基本的なルーチンを変更することで、検証コードを回避しようとすることがあります。この種の攻撃に対する耐性を得るには、次のものをはじめとするさまざまなコーディングテクニックが必要です。

- システムが提供する API を使用せずに、暗号チェックのためのコードをインラインする。
- App のバイナリにパッチを当てる際に標的になりやすい、単純なコード構造を避ける。

  たとえば、次のようなコードを書くことは避けてください。

```
if (failedValidation) {
    exit(173);
}
```

- 難読化など、コードを堅牢にするテクニックを実装する。

  複数の App が同じコードを使用して検証を実行している場合、このコードに共通する署名が、App のバイナリにパッチを当てるツールの標的になる場合があります。
- `exit` 関数が、App の終了に失敗した場合でも、App が実行を停止することを確認する。

開発プロセス中のテスト

開発プロセス中にメインAppをテストするには、App を起動するための有効なレシートが必要です。これを設定するには、次の手順を実行します。

1. Apple のサーバに接続できるよう、インターネットに接続されていることを確認します。
2. App をダブルクリックして起動します（または何らかの方法で Launch Services から App を起動します）。

App を起動すると、次の処理が行われます。

- App にレシートが存在しないため、App はレシート検証に失敗し、ステータス `173` で終了します。
- システムは終了ステータスを解釈し、有効なレシートの取得を試みます。App の署名証明書が有効であることを前提に、システムは App に有効なレシートをインストールします。システムは iTunes の認証情報を求めることがあります。
- システムは App を再起動し、App でのレシート検証が成功します。

開発段階でレシートがインストールされていると、`gdb` または Xcode のデバッガを使用するなど、どのような方法でも App を起動できます。

App 内課金の検証

App 内課金を検証するには、App で次のテストを順番に実行します。

1. 前のセクションの説明に従って、App のレシートを解析し、検証します。

   App のレシートが有効でない場合、App 内課金はいずれも有効ではありません。

2. App 内課金のレシート（タイプ 17 の属性の値）を解析します。

   App 内課金の各レシートは、App のレシートと同様、属性のセットで構成されています。これらレシートの構造はリスト 1-2 （12 ページ） で定義されています。レシートを解析する際と同様に、`asn1c` ツールを使用して ASN.1 の記述からコードの一部を生成できます。表に表示されないタイプの属性はすべて無視します。これらの属性はシステムで使用するためのもので、その内容は随時変更される場合があります。

   レシートのフィールドについて詳しくは、Receipt Fields （24 ページ） を参照してください。

3. App 内課金の各レシートの製品 ID を調べ、対応する App の機能やコンテンツを有効にします。購読期間の計算方法について詳しくは、購読についてを参照してください。

App 内課金のレシート検証が失敗した場合、単純に、App はその機能やコンテンツを有効にしません。

リスト 1-2　ASN.1 における App 内課金のレシートフォーマットの定義

```
InAppAttribute ::= SEQUENCE {
    type                    INTEGER,
    version                 INTEGER,
    value                   OCTET STRING
}

InAppReceipt ::= SET OF InAppAttribute
```

元のトランザクション ID と元のトランザクション日の属性は、購入したものが再ダウンロードされる際に使用します。購入したものが再ダウンロードされると、新しいトランザクション ID を与えられますが、元の購入時の ID と日付が含まれています。

実装のヒント

このセクションでは、レシートの検証を実装する際に参考になる、いくつかのコードリストをご紹介します。

macOS での GUID の取得

macOS では、検証コードで GUID を取得する際に使用されるメソッドと App のレシートが作成される際に使用されるメソッドが正確に同じになるよう、リスト 1-3 のモデルに従います（またはこのコードをそのまま使用します）。

リスト 1-3　コンピュータの GUID の取得

```
#import <IOKit/IOKitLib.h>
#import <Foundation/Foundation.h>

// Returns a CFData object, containing the computer's GUID.
CFDataRef copy_mac_address(void)
{
    kern_return_t             kernResult;
    mach_port_t               master_port;
    CFMutableDictionaryRef    matchingDict;
    io_iterator_t             iterator;
    io_object_t               service;
    CFDataRef                 macAddress = nil;

    kernResult = IOMasterPort(MACH_PORT_NULL, &master_port);
    if (kernResult != KERN_SUCCESS) {
        printf("IOMasterPort returned %d\n", kernResult);
        return nil;
    }

    matchingDict = IOBSDNameMatching(master_port, 0, "en0");
    if (!matchingDict) {
        printf("IOBSDNameMatching returned empty dictionary\n");
        return nil;
    }
```

```
    kernResult = IOServiceGetMatchingServices(master_port, matchingDict, &iterator);
    if (kernResult != KERN_SUCCESS) {
        printf("IOServiceGetMatchingServices returned %d\n", kernResult);
        return nil;
    }

    while((service = IOIteratorNext(iterator)) != 0) {
        io_object_t parentService;

        kernResult = IORegistryEntryGetParentEntry(service, kIOServicePlane,
                &parentService);
        if (kernResult == KERN_SUCCESS) {
            if (macAddress) CFRelease(macAddress);

           macAddress = (CFDataRef) IORegistryEntryCreateCFProperty(parentService,
                    CFSTR("IOMACAddress"), kCFAllocatorDefault, 0);
            IOObjectRelease(parentService);
        } else {
            printf("IORegistryEntryGetParentEntry returned %d\n", kernResult);
        }

        IOObjectRelease(service);
    }
    IOObjectRelease(iterator);

    return macAddress;
}
```

レシートの解析と署名の確認

次のコードリストは、OpenSSL と`asn1c`を使用したレシート検証の 1 つの実装例の概要として使用してください。これらのリストは、関連する API とデータ構造に注目し、ユーザ独自のコードを記述する際の参考となるよう提供されています。コピー＆ペーストして使用するためのものではありません。

OpenSSL を使用する場合は、それに対して静的にバイナリをリンクします。OpenSSL に対する動的なリンクは非推奨であり、ビルド時に警告されます。

リストに概説されているように、コードが次の処理を実行していることを確認してください。

1. 署名を確認する（リスト 1-4 （15 ページ））。
2. ペイロードを解析する（リスト 1-5 （16 ページ））。
3. レシートの属性を抽出する（リスト 1-6 （16 ページ））。
4. GUID のハッシュ値を計算する（リスト 1-7 （17 ページ））。

リスト 1-4 OpenSSL を使用した署名の確認

```
/* The PKCS #7 container (the receipt) and the output of the verification. */
BIO *b_p7;
PKCS7 *p7;

/* The Apple root certificate, as raw data and in its OpenSSL representation. */
BIO *b_x509;
X509 *Apple;

/* The root certificate for chain-of-trust verification. */
X509_STORE *store = X509_STORE_new();

/* ... Initialize both BIO variables using BIO_new_mem_buf() with a buffer and its
 size ... */

/* Initialize b_out as an output BIO to hold the receipt payload extracted during
 signature verification. */
BIO *b_out = BIO_new(BIO_s_mem());

/* Capture the content of the receipt file and populate the p7 variable with the
PKCS #7 container. */
p7 = d2i_PKCS7_bio(b_p7, NULL);

/* ... Load the Apple root certificate into b_X509 ... */
```

```
/* Initialize b_x509 as an input BIO with a value of the Apple root certificate and
 load it into X509 data structure. Then add the Apple root certificate to the
structure. */
Apple = d2i_X509_bio(b_x509, NULL);
X509_STORE_add_cert(store, Apple);

/* Verify the signature. If the verification is correct, b_out will contain the
PKCS #7 payload and rc will be 1. */
int rc = PKCS7_verify(p7, NULL, store, NULL, b_out, 0);

/* You must verify the fingerprint of the root certificate and verify the OIDs of
 the intermediate certificate and signing certificate. The OID in the certificate
 policies extension of the intermediate certificate is (1 2 840 113635 100 6 2 1),
 and the marker OID of the signing certificate is (1 2 840 113635 100 6 11 1). */
```

リスト 1-5　asn1c を使用したペイロードの解析

```
#include "Payload.h" /* This header file is generated by asn1c. */

/* The receipt payload and its size. */
void *pld = NULL;
size_t pld_sz;

/* Variables used to parse the payload. Both data types are declared in Payload.h.
 */
Payload_t *payload = NULL;
asn_dec_rval_t rval;

/* ... Load the payload from the receipt file into pld and set pld_sz to the payload
 size ... */

/* Parse the buffer using the decoder function generated by asn1c.  The payload
variable will contain the receipt attributes. */
rval = asn_DEF_Payload.ber_decoder(NULL, &asn_DEF_Payload, (void **)&payload, pld,
 pld_sz, 0);
```

リスト 1-6　レシートの属性の抽出

```
/* Variables used to store the receipt attributes. */
```

```
OCTET_STRING_t *bundle_id = NULL;
OCTET_STRING_t *bundle_version = NULL;
OCTET_STRING_t *opaque = NULL;
OCTET_STRING_t *hash = NULL;

/* Iterate over the receipt attributes, saving the values needed to compute the
GUID hash. */
size_t i;
for (i = 0; i < payload->list.count; i++) {
    ReceiptAttribute_t *entry;

    entry = payload->list.array[i];

    switch (entry->type) {
        case 2:
            bundle_id = &entry->value;
            break;
        case 3:
            bundle_version = &entry->value;
            break;
        case 4:
            opaque = &entry->value;
            break;
        case 5:
            hash = &entry->value;
            break;
        }
    }
```

リスト 1-7　GUID のハッシュ値の計算

```
/* The GUID returned by copy_mac_address() is a CFDataRef.  Use CFDataGetBytePtr()
 and CFDataGetLength() to get a pointer to the bytes that make up the GUID and to
 get its length. */
UInt8 *guid = NULL;
size_t guid_sz;
```

```
/* Declare and initialize an EVP context for OpenSSL. */
EVP_MD_CTX evp_ctx;
EVP_MD_CTX_init(&evp_ctx);

/* A buffer for result of the hash computation. */
UInt8 digest[20];

/* Set up the EVP context to compute a SHA-1 digest. */
EVP_DigestInit_ex(&evp_ctx, EVP_sha1(), NULL);

/* Concatenate the pieces to be hashed.  They must be concatenated in this order.
 */
EVP_DigestUpdate(&evp_ctx, guid, guid_sz);
EVP_DigestUpdate(&evp_ctx, opaque->buf, opaque->size);
EVP_DigestUpdate(&evp_ctx, bundle_id->buf, bundle_id->size);

/* Compute the hash, saving the result into the digest variable. */
EVP_DigestFinal_ex(&evp_ctx, digest, NULL);
```

App Store を使用したレシートの検証

App Store との通信には信頼できるサーバを使用してください。独自のサーバを使用する場合は、アプリケーションがこのサーバのみを認識し信頼するように設計し、このサーバが App Store のサーバに接続することを確認する必要があります。この接続のいずれの終端もコントロールできないため、ユーザのデバイスと App Store の間で信頼できる接続を直接構築することはできません。

App Store との通信は、RFC 4627 で定義されている JSON 辞書によって構造化されています。バイナリデータは、RFC 4648 で定義されている base64 でエンコードされています。

レシートデータの読み込み

レシートデータを取得するには、`NSBundle` の `appStoreReceiptURL`メソッドを使用してアプリケーションのレシートの場所を特定し、それからファイル全体を読み込みます。`appStoreReceiptURL` メソッドが利用できない場合は、下位互換性を維持するためにトランザクションの`transactionReceipt` プロパティの値にフォールバックできます。次に、このデータをサーバに送信します。サーバとのやり取りの詳細は、他の場合と同様、開発者が責任を負います。

```
// Load the receipt from the app bundle.
NSURL *receiptURL = [[NSBundle mainBundle] appStoreReceiptURL];
NSData *receipt = [NSData dataWithContentsOfURL:receiptURL];
if (!receipt) { /* No local receipt -- handle the error. */ }

/* ... Send the receipt data to your server ... */
```

App Store へのレシートデータの送信

サーバ上で、以下のキーを使用して JSON オブジェクトを生成します。

キー	値
`receipt-data`	base64 でエンコードされたレシートデータ。

キー	値
password	*自動更新購読が含まれているレシートにのみ使用。* アプリケーションの共有シークレット（16 進数文字列）。
exclude-old-transactions	*自動更新または非更新購読が含まれる iOS7 スタイルの App レシートの場合にのみ使用します。* 値が true の場合、応答には購読の最新の更新トランザクションのみが含まれます。

この JSON オブジェクトを、HTTP POST リクエストのペイロードとして送信します。テスト環境では、URL に `https://sandbox.itunes.apple.com/verifyReceipt` を使用します。本番環境では、URL に `https://buy.itunes.apple.com/verifyReceipt` を使用します。

```
NSData *receipt; // Sent to the server by the device

// Create the JSON object that describes the request
NSError *error;
NSDictionary *requestContents = @{
    @"receipt-data": [receipt base64EncodedStringWithOptions:0]
};
NSData *requestData = [NSJSONSerialization dataWithJSONObject:requestContents
                                                       options:0
                                                         error:&error];

if (!requestData) { /* ... Handle error ... */ }

// Create a POST request with the receipt data.
NSURL *storeURL = [NSURL
URLWithString:@"https://buy.itunes.apple.com/verifyReceipt"];
NSMutableURLRequest *storeRequest = [NSMutableURLRequest requestWithURL:storeURL];
[storeRequest setHTTPMethod:@"POST"];
[storeRequest setHTTPBody:requestData];

// Make a connection to the iTunes Store on a background queue.
NSOperationQueue *queue = [[NSOperationQueue alloc] init];
[NSURLConnection sendAsynchronousRequest:storeRequest queue:queue
```

```
        completionHandler:^(NSURLResponse *response, NSData *data, NSError
*connectionError) {
    if (connectionError) {
        /* ... Handle error ... */
    } else {
        NSError *error;
        NSDictionary *jsonResponse = [NSJSONSerialization JSONObjectWithData:data
 options:0 error:&error];
        if (!jsonResponse) { /* ... Handle error ...*/ }
        /* ... Send a response back to the device ... */
    }
}];
```

応答の解析

応答のペイロードは JSON オブジェクトで、次のキーと値が含まれています。

キー	値
`status`	レシートが有効の場合は `0`。そうでない場合は、表 2-1 （22 ページ）のエラーコードのいずれか。 iOS 6 スタイルのトランザクションレシートの場合、ステータスコードは該当するトランザクションレシートのステータスを表します。 iOS 7 スタイルの App レシートの場合、ステータスコードは該当する App レシート全体のステータスを表します。たとえば、有効な App レシートを送信した場合、期限切れの購読が含まれていても、レシート全体が有効であるため、応答は `0` になります。
`receipt`	検証のために送信されたレシートを JSON 形式で記述したもの。レシートに含まれるキーについて詳しくは、Receipt Fields （24 ページ） を参照してください。
`latest_receipt`	*自動更新購読の iOS 6 スタイルのトランザクションレシートに対してのみ返されます。* 最新の更新の base-64 でエンコードされたレシート。

キー	値
latest_receipt_info	*自動更新購読の iOS 6 スタイルのトランザクションレシートに対してのみ返されます。* 最新の更新のレシートを JSON 形式で記述したもの。
latest_expired_-receipt_info	*自動更新購読の iOS 6 スタイルのトランザクションレシートに対してのみ返されます。* 期限切れの購読のレシートの JSON 形式で記述したもの。
pending_renewal_info	*自動更新購読を含む iOS 7 スタイルの App レシートに対してのみ返されます。* JSON ファイルでは、このキーの値は、各要素が 製品識別子 （28 ページ）によって識別された各自動更新購読の保留中の更新情報を含む配列です。保留中の更新では、今後スケジュールされている更新や、何らかの理由で過去に更新できなかった更新を参照する場合があります。
is-retryable	このレシートの検証を再試行します。*ステータスコード 21100 ～ 21199 にのみ該当*（表 2-1 （22 ページ）に記載）

表 2-1 ステータスコード

ステータス コード	説明
21000	App Store は、提供された JSON オブジェクトを読み込むことができません。
21002	receipt-data プロパティのデータが不正な形式であるか、欠落しています。
21003	レシートを認証できません。
21004	提供された共有シークレットとアカウントのファイルにある共有シークレットが一致しません。
21005	レシートサーバは現在利用できません。

ステータス コード	説明
21006	このレシートは有効ですが、購読の有効期限が切れています。このステータスコードがサーバに返された場合、レシートデータもデコードされ、応答の一部として返されます。 *自動更新購読のiOS6スタイルのトランザクションレシートに対してのみ返されます。*
21007	テスト環境のレシートが、検証のために本番環境に送信されました。テスト環境に送信してください。
21008	本番環境のレシートが、検証のためにテスト環境に送信されました。本番環境に送信してください。
21010	このレシートは許可できませんでした。このレシートはこれまでに購入が行われていない場合と同様に扱います。
21100-21199	内部データアクセスエラー。

`latest_receipt` キーと `latest_receipt_info` キーの値は、自動更新購読が現在アクティブかどうかをチェックする際に便利です。

`latest_expired_receipt_info` キーの値は、自動更新購読の有効期限が切れているかどうかをチェックするときに便利です。Subscription Expiration Intent （30 ページ）の値とともに使用することで、期限切れの理由を特定することができます。

`pending_renewal_info` キーの値は自動更新購読の保留中の更新トランザクションに関する重要な情報を入手する際に便利です。

購読の App レシートまたはトランザクションレシートを表示し、これらの値をチェックすることで、現在アクティブな購読期間についての情報を取得できます。検証されているレシートが最新の更新に関するものであれば、`latest_receipt` の値は `receipt-data`（リクエストの場合）と同じになり、`latest_receipt_info` の値は `receipt` と同じになります。

レシートのフィールド

レシートは、多数のフィールドで構成されています。フィールドの中には ASN.1 形式のレシートでのみ、または JSON 形式のレシートを App Store で検証したときにのみ、ローカルで使用できるものがあります。以下に記載されていないキーは、Apple で使用するために予約されているため、App では無視してください。

App レシートのフィールド

バンドル ID

App のバンドル ID です。

ASN.1 フィールドタイプ：2

ASN.1 フィールド値：UTF8STRING

JSON フィールド名：bundle_id

JSON フィールド値：文字列

このフィールドは、`Info.plist` ファイルの `CFBundleIdentifier` の値に対応します。この値を使用して、App のレシートが実際に生成されたかどうかを検証します。

App バージョン

App のバージョン番号です。

ASN.1 フィールドタイプ：3

ASN.1 フィールド値：UTF8STRING

JSON フィールド名：application_version

JSON フィールド値：文字列

このフィールドは、`Info.plist` の `CFBundleVersion`（iOS の場合）または `CFBundleShortVersionString`（macOS の場合）の値に対応します。

オペーク値

検証の際、他のデータとともに SHA-1 ハッシュを計算するために使う値。

ASN.1 フィールドタイプ：4

ASN.1 フィールド値：バイト列

JSON フィールド名：（なし）

JSON フィールド値：（なし）

SHA-1 ハッシュ

レシートの検証に使用される SHA-1 ハッシュです。

ASN.1 フィールドタイプ：5

ASN.1 フィールド値：20 バイトの SHA-1 ダイジェスト

JSON フィールド名：（なし）

JSON フィールド値：（なし）

App 内課金のレシート

App 内課金のレシートです。

ASN.1 フィールドタイプ：17

ASN.1 フィールド値：App 内課金のレシート属性のセット

JSON フィールド名：in_app

JSON フィールド値：App 内課金のレシートの配列

JSON ファイルの場合は、このキーの値はすべての App 内課金のレシートを格納している配列になります。ASN.1 ファイルの場合は複数のフィールドがあり、そのタイプはすべて17です。各フィールドには App 内課金のレシートが1つ格納されています。

注意：： 空の配列は有効なレシートです。

消耗型製品のApp内課金のレシートは、購入が行われた時点でレシートに追加され、Appでトランザクションが終了するまでの間、レシートに保持されます。その後、レシートが次回更新されたとき（お客様が別の購入を行ったり、Appで明示的なレシートの更新が行われたりした場合）にレシートから削除されます。

非消耗型製品、自動更新購読、非更新購読、無料購読のApp内課金レシートは、無期限に残ります。

App のオリジナルバージョン

App が最初に購入されたときのバージョンです。

ASN.1 フィールドタイプ：19

ASN.1 フィールド値：UTF8STRING

JSON フィールド名：original_application_version

JSON フィールド値：文字列

このフィールドは、最初の購入が行われたときの `Info.plist` ファイルの `CFBundleVersion`（iOS の場合）または `CFBundleShortVersionString`（macOS の場合）に対応します。

サンドボックス環境では、このフィールドの値は常に「1.0」です。

レシートの作成日付

App レシートが作成された日付。

ASN.1 フィールドタイプ：12

ASN.1 フィールド値：IA5STRING（RFC 3339 の日付として解釈されます）

JSON フィールド名：creation_date

JSON フィールド値：IA5STRING（RFC 3339 の日付として解釈されます）

レシートを検証する際、この日付を使ってレシートの署名を検証します。

注意：：多くの暗号処理ライブラリが、PKCS7 パッケージの検証時にデバイスの現在日時をデフォルトで使うようになっていますが、レシートの署名を検証する際、正しい結果が得られない可能性があります。たとえば、有効な証明書を使ってレシートに署名を施しても、その証明書の期限が切れている場合、デバイスの現在日付を使うと誤って無効と判定されてしまうためです。

必ず「レシートの有効期限」フィールドから取得した日付を使って検証するようにしてください。

レシートの有効期限

App レシートが期限切れになる日付です。

ASN.1 フィールドタイプ：21

ASN.1 フィールド値：IA5STRING（RFC 3339 の日付として解釈されます）

JSON フィールド名：expiration_date

JSON フィールド値：IA5STRING（RFC 3339 の日付として解釈されます）

このキーは、Volume Purchase Program 経由で購入された App にのみ存在します。このキーが存在していない場合は、そのレシートに有効期限はありません。

レシートの検証時に、この日付と現在の日付とを比較して、レシートを期限切れにするかどうかを判別します。この日付は、有効期限までの残り時間など、他の情報の計算に使用しないようにしてください。

App 内課金のレシートのフィールド

数量

購入したアイテム数です。

ASN.1 フィールドタイプ：1701

ASN.1 フィールド値：INTEGER

JSON フィールド名：quantity

JSON フィールド値：整数として解釈できる文字列

この値は、トランザクションの `payment` プロパティに保存されている `SKPayment` オブジェクトの `quantity` プロパティに対応します。

製品 ID

購入したアイテムの製品IDです。

ASN.1 フィールドタイプ：1702

ASN.1 フィールド値：UTF8STRING

JSON フィールド名：product_id

JSON フィールド値：文字列

この値は、トランザクションの `payment` プロパティに保存されている `SKPayment` オブジェクトの `productIdentifier` プロパティに対応します。

トランザクション ID

購入したアイテムのトランザクション ID です。

ASN.1 フィールドタイプ：1703

ASN.1 フィールド値：UTF8STRING

JSON フィールド名：transaction_id

JSON フィールド値：文字列

この値は、トランザクションの `transactionIdentifier` プロパティに対応します。

以前のトランザクションを復元したトランザクションの場合、この値は、元の購入トランザクションのトランザクション ID とは異なります。自動更新購読のレシート内のトランザクション ID の新しい値は、購読が自動的に更新されたり、新しいデバイスに復元されたりするたびに生成されます。

オリジナルのトランザクション ID

以前のトランザクションを復元したものである場合、元のトランザクションのトランザクション ID です。そうでない場合は、トランザクション ID と同一になります。

ASN.1 フィールドタイプ：1705

ASN.1 フィールド値：UTF8STRING

JSON フィールド名：original_transaction_id

JSON フィールド値：文字列

この値は、元のトランザクションの `transactionIdentifier` プロパティに対応します。

この値は、特定の購読に生成されたすべてのレシートの値と同じです。この値は、同じ個人であるお客様の購読に関する複数の iOS 6 スタイルのトランザクションレシートを関連付ける場合に便利です。

購入日

アイテムが購入された日付と時刻です。

ASN.1 フィールドタイプ：1704

ASN.1 フィールド値：IA5STRING（RFC 3339 の日付として解釈されます）

JSON フィールド名：purchase_date

JSON フィールド値：文字列（RFC 3339 の日付として解釈されます）

この値は、トランザクションの `transactionDate` プロパティに対応します。

以前のトランザクションを復元するものである場合、購入日はオリジナル購入日と同じです。元のトランザクションの日付を取得するには、Original Purchase Date （29 ページ） を使用します。

自動更新購読のレシートでは、購入日は購読が購入または更新（失効したかに関わらず）された日付になります。現在の期間の有効期限満了日に自動更新される場合、購入日は次の期間の開始日となります。これは現在の期間の終了日でもあります。

オリジナル購入日

以前のトランザクションを復元したトランザクションの場合、元のトランザクションの日付になります。

ASN.1 フィールドタイプ：1706

ASN.1 フィールド値：IA5STRING（RFC 3339 の日付として解釈されます）

JSON フィールド名：original_purchase_date

JSON フィールド値：文字列（RFC 3339 の日付として解釈されます）

この値は、元のトランザクションの `transactionDate` プロパティに対応します。

自動更新購読のレシートでは、購読が更新されている場合であっても、購読期間が開始されたときを示します。

購読の有効期限

購読の有効期限が、1970 年 1 月 1 日 00:00:00 GMT からのミリ秒単位で表されています。

ASN.1 フィールドタイプ：1708

ASN.1 フィールド値：IA5STRING（RFC 3339 の日付として解釈されます）

JSON フィールド名：expires_date

JSON フィールド値：文字列（RFC 3339 の日付として解釈されます）

このキーは、自動更新購読のレシートにのみ存在します。この値を使用して、購読の更新または期限切れになる日付を識別し、お客様がコンテンツやサービスにアクセスできるかどうかを判断します。最新のレシートを検証し、最新の更新トランザクションの購読の有効期限が過去の日付である場合は、購読の有効期限が切れていることになります。

購読期限切れの意図

期限切れの購読の場合は、その購読の期限切れの理由を示します。

ASN.1 フィールドタイプ：（なし）

ASN.1 フィールド値：（なし）

JSON フィールド名：expiration_intent

JSON フィールド値：整数として解釈できる文字列

「1」 - お客様が購読をキャンセルした。

「2」 - 課金エラー。お客様の支払い情報が有効ではなくなった場合など。

「3」 - お客様が、直近の値上げに同意しなかった。

「4」 - 更新時に製品が販売されていなかった。

「5」 - 不明なエラー。

このキーは期限切れの自動更新購読を含むレシートにのみ存在します。この値を使用して、お客様が再購読をする際に App 内に適切なメッセージを表示するかどうかを判断できます。

購読の再試行フラグ

期限切れの購読に対し、Apple がまだ自動的に購読を更新しようとしているかどうかを示します。

ASN.1 フィールドタイプ：（なし）

ASN.1 フィールド値：（なし）

JSON フィールド名：is_in_billing_retry_period

JSON フィールド値：整数として解釈できる文字列

「1」 - App Store はまだ購読の更新を試みている。

「0」 - App Store は購読の更新の試みを停止した。

このキーは、自動更新購読のレシートにのみ存在します。App Store がトランザクションを完了できなかったため、お客様の購読が更新できなかった場合は、この値は、App Store でまだ購読を更新しようとしているかどうかを反映します。

購読の試用期間

購読に対し、その購読が無料試用期間中であるかどうかを示します。

ASN.1 フィールドタイプ：（なし）

ASN.1 フィールド値：（なし）

JSON フィールド名：is_trial_period

JSON フィールド値：文字列

このキーは、自動更新購読のレシートにのみ存在します。現在、お客様の購読が無料試用期間中である場合、このキーの値は `"true"`、無料試用期間中でない場合は、`"false"` になります。

キャンセル日

Apple カスタマーサポートによってキャンセルされたトランザクションで、キャンセルが行われた日時です。

ASN.1 フィールドタイプ：1712

ASN.1 フィールド値：IA5STRING（RFC 3339 の日付として解釈されます）

JSON フィールド名：CANCELLATION_DATE

JSON フィールド値：文字列（RFC 3339 の日付として解釈されます）

キャンセルされたレシートは、購入が行われなかったものとして扱います。

> **注意：:** キャンセルされた App 内課金は、無期限にレシートに残ります。返金が、非消耗型製品、自動更新購読、非更新購読、無料購読に対して行われた場合のみが対象となります。

キャンセル理由

キャンセルされたトランザクションに対する、キャンセル理由を示します。

ASN.1 フィールドタイプ：（なし）

ASN.1 フィールド値：（なし）

JSON フィールド名：cancellation_reason

JSON フィールド値：整数として解釈できる文字列

「1」- App に問題が生じたか、問題が生じる可能性があったため、お客様がトランザクションをキャンセルした。

「0」- お客様が誤って購入した場合など、別の理由でトランザクションがキャンセルされた。

キャンセル日と一緒にこの値を使用することで、お客様が Apple カスタマーサポートに連絡する原因となった App の問題を識別することができます。

App のアイテム ID

トランザクションを作成した App を一意に識別するために、App Store が使用する文字列です。

ASN.1 フィールドタイプ：（なし）

ASN.1 フィールド値：（なし）

JSON フィールド名：app_item_id

JSON フィールド値：文字列

サーバが複数の App をサポートする場合は、この値を使用して区別できます。

App に ID が割り当てられるのは本番環境に限られるため、テスト環境で作成されたレシートにはこのキーは存在しません。

このフィールドは、Mac 用の App には存在しません。

詳しくは、Bundle Identifier （24 ページ） をご覧ください。

外部バージョン ID

App の改訂を一意に識別する任意の番号です。

ASN.1 フィールドタイプ：（なし）

ASN.1 フィールド値：（なし）

JSON フィールド名：version_external_identifier

JSON フィールド値：文字列

このキーはテスト環境で作成されたレシートには存在しません。この値を使用すると、お客様が購入した App のバージョンを特定できます。

Web 注文明細行 ID

購読の購入を識別するための主キーです。

ASN.1 フィールドタイプ：1711

ASN.1 フィールド値：INTEGER

JSON フィールド名：web_order_line_item_id

JSON フィールド値：文字列

この値は、購読の更新購入イベントなど、購入イベントをデバイス間で識別する固有の ID です。

購読の自動更新ステータス

自動更新購読の現在の更新状況。

ASN.1 フィールドタイプ：（なし）

ASN.1 フィールド値：（なし）

JSON フィールド名：auto_renew_status

JSON フィールド値：整数として解釈できる文字列

「1」 - 現在の購読期間の終了時に購読が更新される。

「0」 - お客様が購読の自動更新をオフにした。

このキーは、アクティブまたは期限切れの購読に対する自動更新購読のレシートにのみ存在します。このキーの値は、お客様の購読ステータスとして解釈すべきではありません。この値は、お客様が現在のプランからダウングレード可能な下位レベルの購読プランなどの、App内の代替購読製品を表示するために使用できます。

購読の自動更新環境設定

自動更新購読の現在の更新の環境設定。

ASN.1 フィールドタイプ：（なし）

ASN.1 フィールド値：（なし）

JSON フィールド名：auto_renew_product_id

JSON フィールド値：文字列

このキーは、自動更新購読のレシートにのみ存在します。このキーの値は、お客様の購読が更新される製品の `productIdentifier` プロパティに対応します。この値は、現在の購読期間が終了する前に、お客様に代わりのサービスレベルを提示するために使用できます。

購読価格の同意ステータス

購読価格の値上げ時の現在の価格の同意状況。

ASN.1 フィールドタイプ：（なし）

ASN.1 フィールド値：（なし）

JSON フィールド名：price_consent_status

JSON フィールド値：整数として解釈できる文字列

「1」 - お客様は値上げに同意した。購読は値上げされた価格で更新されます。

「0」 - お客様は値上げに関してアクションを取っていない。お客様が更新日までに同意しない場合は、購読の有効期限が切れます。

このキーは、アクティブな購読ユーザの既存の価格が維持されずに購読価格が値上げされた場合、自動更新購読レシートに存在します。この値は、お客様の新価格への同意の有無を追跡し、適切なアクションを取るために使用できます。

Apple Inc.
1 Infinite Loop
Cupertino, CA 95014
U.S.A.

Apple Japan合同会社
〒106-6140 東京都港区六本木 6丁目10番1号
六本木ヒルズ
http://www.apple.com/jp/

Draft copy. Trademarks go here.