	**Redatto da:** Valerio Bondì
	20 Gennaio 2022
Oggetto	**Stato**
Documentazione progetto SatispayAuthTest	Final Draft

Ambito della seguente documentazione è l'illustrazione del progetto "SatispayAuthTest", consiste nell'implementazione di un meccanismo con la quale i client firmano dei messaggi http per effettuare l'autenticazione ai sistemi Satispay.

Il progetto è sviluppato utilizzando Java 8 e Spring Boot v2.6.2.

Sommario

Interfaccia

L'interfaccia consiste nell'esposizione di un'API *"authenticate"* di tipo GET composta da:

- **Parametri:**
  - Ha un singolo parametro a path parameter **obbligatorio** di nome *type*, che identifica il tipo di chiamata verso l'API di test signature Satispay.
  - Può assumere solo 2 valori: GET o POST;
- **Body:**
  - Ha un body opzionale, dove nel caso di *type=POST* verrà valorizzato ed inserito nella richiesta http verso l'API di Satispay.
- **Output:**
  - Verrà restituito un JSON che sarà la risposta dell'API test signature di Satispay.

Business logic

Tutta la logica dell'applicazione risiede nella classe *SatispayController.java* che implementa l'API GET *authenticate(type, body)*.

Nell'application.properties di Spring sono definite 4 costanti:

- *PRIVATE_KEY_PATH*: percorso relativo del file .pem della chiave privata;
- *TEST_SIGNATURE_URL:* URL dell'API Test Signature Satispay;
- *KEY_ID:* la keyId già autorizzata;
- *ALGORITHM:* l'algoritmo di cifratura utilizzato per la signature.

Gli steps effettuati sono i seguenti:

authenticate(String type, String body):

L'API esposta dal programma.

- Inizialmente si effettuano 2 controlli:
  - Se il parametro *type* non è né GET né POST, allora si restituisce un'eccezione;
  - In caso di body null verrà impostata una stringa vuota, poiché da requisito è richiesto obbligatoriamente *l'header digest*.
    - Anche se la richiesta è GET verrà inserita una stringa vuota. Questo poiché se si forza una GET ad avere un body, *l'header (request-target)* verrà automaticamente riconosciuto come POST, causando una differenza fra il digest e la signed string.
- Viene creata la stringa *digest* effettuando l'hash del body in SHA-256 e poi facendo l'encoding dell'hash in Base64;
- Viene creata la stringa da firmare utilizzando i parametri richiesti per l'autenticazione Satispay;
- Viene firmata la stringa utilizzando crittografia RSA ed encoding Base64;
- Viene creata ed eseguita la richiesta HTTP verso l'API di test signature di Satispay.

parseBody(String body):

Tale metodo effettua 3 controlli per verificare il sistema operativo in uso. Ciò è dovuto al fatto che a seconda dell'OS utilizzato, possono esserci differenti tipi di caratteri di escape (new line e carriage return) che possono influire sul calcolo del digest.
L'unico carattere consentito (verificato utilizzando il tool online di creazione digest di Satispay) è il carattere new line \n.

createSignatureString(String dateFormatted, String digest, String type):

Ritorna la stringa da firmare impostando i parametri quali data (viene impostata la data attuale), digest e tipo di richiesta (get o post).

encodeSignature(String signatureString):

Effettua l'encoding della firma utilizzando l'algoritmo RSA ed un encoding di tipo Base64:

- Si inizializzano gli oggetti relativi alla chiave privata e alla firma;
- Viene poi letta la chiave privata da un file .pem, effettuando un parsing rimuovendo i tag di inizio e fine chiave;
- La chiave viene prima decodificata e codificata in uno standard poter generare la firma;
- Infine si effettua la firma con la chiave decodificata e poi la firma stessa codificata tramite Base64.

executeHttpRequest(String dateFormatted, String digest, String encodedSignature, String method, byte[] byteBody):

Crea ed esegue la richiesta HTTP per l'API test signature Satispay.

- Vengono settate le varie property fra cui data, digest, keyId, algoritmo utlizzato, header utilizzati e signature;
- Solo se il body è valorizzato e il metodo è POST, si aggiunge all'output stream il body;
- Viene letta e ritornata la response dell'API di Satispay.

readHttpResponse(HttpURLConnection http):

Legge dall'InputStream della connessione http tutto il contenuto.

Gestione delle eccezioni

Le eccezioni non hanno una particolare gestione poiché non richiesta. Si usa semplicemente una custom exception *SatispayException* che effettua un wrap delle checked exception con un error code specifico (*DIGEST_EXCEPTION* per errore su MessageDigest, *HTTP_EXCEPTION* per errore sul processo di creazione ed esecuzione della chiamata http, *SIGNATURE_EXCEPTION* per errore sul processo di firma).

Librerie

Vengono utilizzate soltanto librerie interne Java e Spring per la gestione dell'API esposta.

Riferimenti

SHA-256 Hashing in Java | Baeldung

How to Read PEM File to Get Public and Private Keys | Baeldung

Digital Signatures in Java | Baeldung

string - What is the difference between \r\n, \r, and \n? - Stack Overflow