Safewalk signature API

Release 3.0.x

AltiPeak

Feb 07, 2021

Contents

Safewalk signature API is an OAuth2 Restful API that can be called by a (system) user with an access key.

Note: The Safewalk signature API can be used with a special user license called `Fast:Auth:Sign` that enables to sign content that is delivered to a user mobile application.

1 The Safewalk Fast Auth Signature end user flow (push)

The following flow will take place on the end user side when it is requested to sign content using the push method:

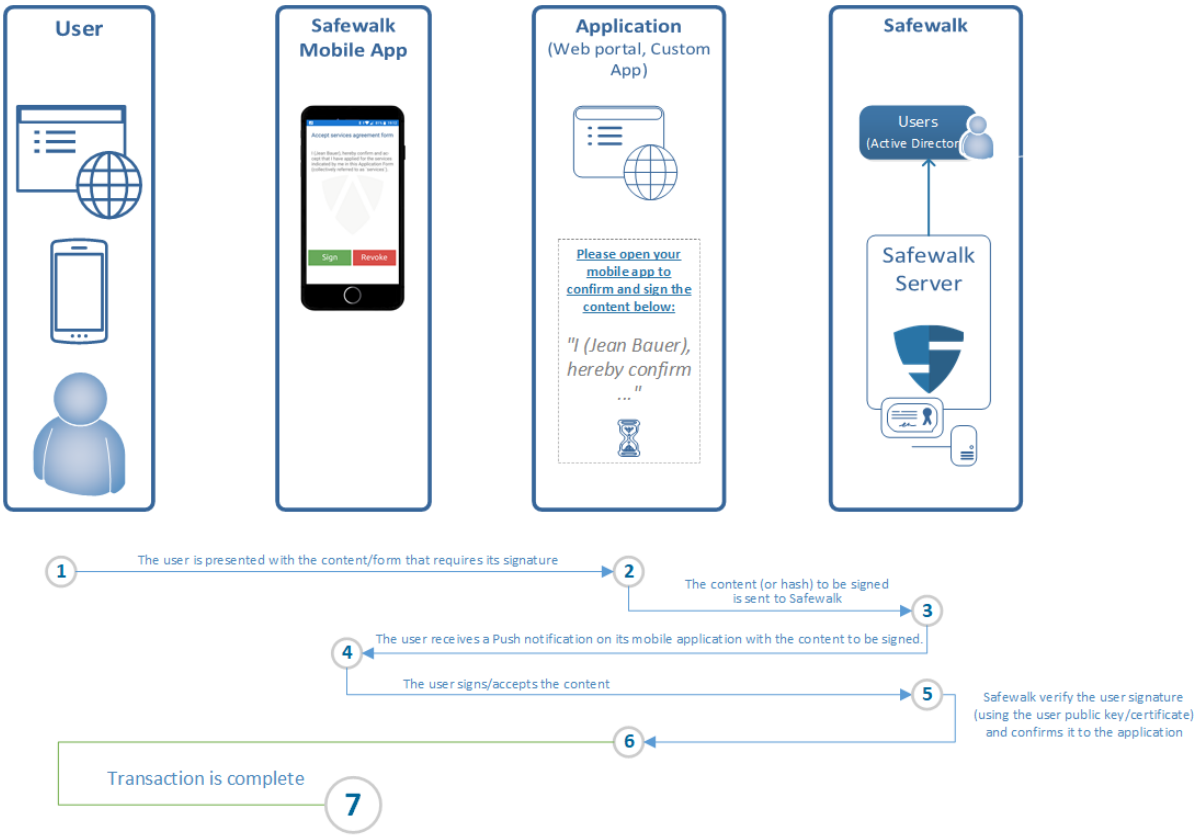

Fig. 1: Safewalk Fast Auth Signature Mobile - Push user flow

2 The Safewalk Fast Auth Signature end user flow (QR/Deep-Link)

The following flow will take place on the end user side when it is requested to sign content using the QR/Deep-link method:

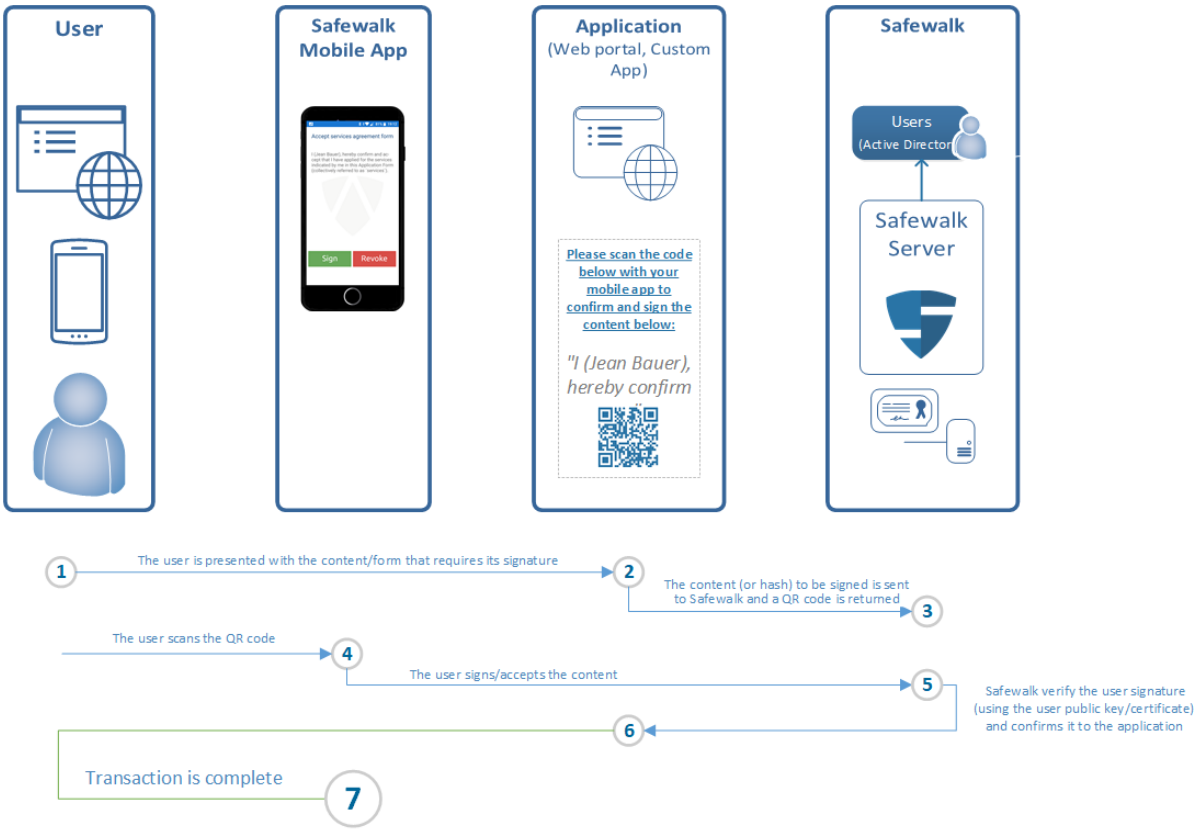

Fig. 2: Safewalk Fast Auth Signature Mobile - QR/Deep-link user flow

3 The Safewalk Fast Auth Signature end to end flow

The overall flow (including the server side interaction) will be the following:

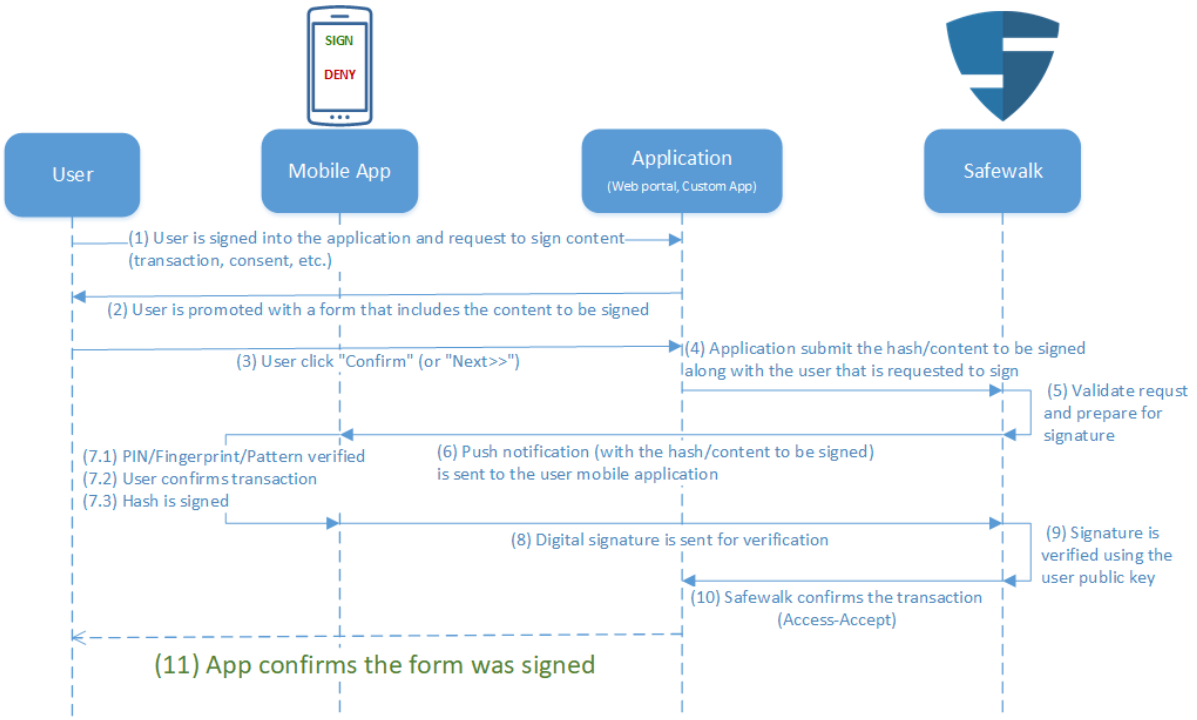

Fig. 3: Safewalk Fast Auth Signature Mobile - End to end flow

4 Creating an OAUTH token for the application account

To be able to query the Safewalk server for signature from a custom application, you will first need to generate an access token that will be used to authenticate to the Safewalk server on behalf of the application on every request.

Follow the steps below to generate an access token:

- Open a browser and sign-in to the Safewalk super-admin console (*https://SAFEWALK_ADDRESS:8443*).
- In the *Internal users & groups* box click the *Users* link and then the *Add user* button at the top right of the screen to create a new internal user.

Note: The password field is required however it is not used by the plug-in for this account so it is recommended to enter a very long and cumbersome password (there is no need to remember it or store it anywhere).

- Click the *Save* button and you will be redirected to a new page.
- In the ‘User Type‘ section, check the box for ‘System user‘ so that this user will be flagged as a system user and will not be manageable from the management console.
- In the ‘Permissions‘ section, remove the user from any group that appear by clicking the ‘Remove all‘button under the ‘Groups‘ combo box.
- In the ‘Access Tokens‘ section select ‘Create Access Token‘
- **Copy the value of the access token that was generated** - You will need it for the configuration of the plug-in/application.

Note: This parameter will be sent to the Safewalk authentication api to identify the application account every time an authentication request is sent.

- Click *Save*

5 The Safewalk Fast Auth Signature API description

`API Reference:`

API to send a signature request from Safewalk.

Method: `POST`

API Path: `api/v1/auth/push_signature/`

Below is an example of such a query using the *curl* command:

```
curl --insecure -X POST -H "Authorization: Bearer ACCESS_KEY" -H "Content-Type: application/json"
--data {"hash":"CONTENT_HASH", "username" : "SIGNING_USER", "data" : "DATA_TO_SIGN_OPTIONAL",
"body" : "BODY_TO_DISPLAY_OPTIONAL", "title" : "TITLE_TO_DISPLAY_OPTIONAL"}
https://SERVER_ADDRESS:PORT/api/v1/auth/push_signature/
```

The parameters values that are provided in the API call are:

`ACCESS_KEY`
The access-key of the application account.

`CONTENT_HASH`
The hash (SHA256) of the content to be signed.

`SIGNING_USER`
The username that will be prompted to sign.

Note: This is not the application account username but the user that is requested to sign.

`DATA_TO_SIGN_OPTIONAL`
An **optional** parameter, that contains the actual data to be signed by the user.

Note:

- When this field is sent, Safewalk will generate the hash value of the content and compare the hash value with the one given in the *CONTENT_HASH* parameter. If the comparison fails an error will be returned and the signature will not be processed.
- When this field is sent, Safewalk will send to the user mobile application the data to be signed without the *CONTENT_HASH* that was passed. The mobile application will then compute the hash value independently and Safewalk will verify that the expected hash was signed.
- If this field is omitted, Safewalk will send the *CONTENT_HASH* parameter directly to the mobile application to be signed as is.
- Adding this parameter is recommended if you would like to add validation to the signature process and there is no concern that Safewalk will be passed with the data and store it in its audit logs.
- Omitting this field is recommended if you would like to expose minimum information of the data that will be signed.

`BODY_TO_DISPLAY_OPTIONAL`
An **optional** parameter, that contains the content that the user will see on its mobile device when it is requested to sign.

Note:

- This field does not have to match the data that the user will sign (and it will not be signed by the user).

- This field should be used in cases where the *DATA_TO_SIGN_OPTIONAL* parameter was omitted and there is a need to display information about the data that is about to be signed to the user.
- If this parameter is not provided and the *DATA_TO_SIGN_OPTIONAL* is not provided as well, Safewalk will display the value of the *Notification body:* parameter that is set in the multilevel setting.

TITLE_TO_DISPLAY_OPTIONAL

An **optional** parameter, that contains the title of the mobile application page that the user will see on its mobile device when it is requested to sign.

Note:

- This field will not be signed by the user.
- If this parameter is not provided, Safewalk will display the value of the *Notification title:* parameter that is set in the multilevel setting.

SERVER_ADDRESS

The server address that will receive the authentication request.

Note: If you are using the authentication API service on the Safewalk Gateway you will need to provide the Safewalk Gateway address here.

If you are querying the authentication API directly on Safewalk you will need to provide the Safewalk server address here.

PORT

The port the authentication api is listening on (443 for the Safewalk Gateway and 8445 for the Safewalk server).

Below is a sample valid query using the *curl* command:

```
curl --insecure -X POST -H "Authorization: Bearer 3cc70b62cab6fd8e130b68d884e6dc34f43e0a01" -H "Content-
↪Type: application/json"
--data '{"hash":"AD73CA1A90649186D93FBD9FD8433799B3F3D7FC2134155B5160A4A664361961", "username" : "jean.
↪bauer",
"data" : "I (Jean Bauer), hereby confirm and accept that I have applied for the services indicated by me␣
↪in this Application Form (collectively referred to as `services`).",
"title" : "Accept services agreement form"}'  https://192.168.223.29:8445/api/v1/auth/push_signature/
```

5.1 API response

When the call to the API returns with `HTTP CODE : 200 OK` the following parameters are expected:

```
{
  "username": "SIGNING_USER",
  "hash": "SIGNED_HASH",
  "certificate": "SIGNING_USER_CERT",
  "result": "SIGNING_RESULT",
  "signature": "SIGNATURE_BASE64",
  "transaction_id": "TRANSACTION_ID"
}
```

SIGNING_USER

The username of the user who signed the provided hash.

SIGNED_HASH

The hash value that was signed by the user.

Note: You can use this value to confirm that the intended hash was signed by comparing it with the hash that was sent for signature.

SIGNING_USER_CERT
An x509 certificate of the user license (containing the public key). This certificate can be used to verify the signature using standard tools. This public key inside the certificate corresponds to the private key that was used by the user for in the signature.

Note: By default Safewalk creates an untrusted certificate for the user license using a Certificate Signing Request (CSR) that is sent from the mobile application during the registration procedure. The CSR is signed with Safewalk's randomly generated private key that is not trusted or recognized by any Certification Authority (CA).

To generate a recognized CA certificate further integration and customization is required.

SIGNING_RESULT
The signature result. This value can be one of: - `SIGN_ACCEPT` - To indicate that the signature was processed successfully. - `SIGN_REJECT` - To indicate that the signature was rejected.

SIGNATURE_HEX
The actual signature encoded in hexadecimal format.

TRANSACTION_ID
The transaction id inside the Safewalk server that can be used to trace the transaction at a later time in Safewalk's transaction and audit logs.

Below is a sample valid response:


```
{
  "username": "jean.bauer@directaccess.domain",
  "hash": "AD73CA1A90649186D93FBD9FD8433799B3F3D7FC2134155B5160A4A664361961",
  "certificate": "-----BEGIN CERTIFICATE-----
↪MIICpDCCAYwCAQAwDQYJKoZIhvcNAQELQAwGDEWMBQGA1UEAwwNMDAwMjAwMDc0MDQyOTAeFw0xODEyMTYxMjI5MTVaFw0yMzEyMTUxMjI5MTVaMBgxFjAUBgNV
↪Wo06q1A9RsQoM2nqJIhD0cOnern6d0u6TjSw5o5cbCmIjGb2RpJ8CyczW8zX4i+6hp37IqJhOcYxJa4TUOshbYCYJais7TqT3Wdj4uQg2qAynaZcCGdLgWH6Pi
↪A21GVC2bV0Udc8A+VpxX6aBsdIixqVB2XTnSFMbU8cLAflKgwkQokLmeC8XOghmXlhaiSQVdRi3Mon8AsWGYIUYGKm7JXrtrAq+AefWEBtpJiuvS+QjpR+zn1qN
↪5zccQb4YwgeLLTN0S6VDfXORy8g741IKZDxGEAOYBZe2Gx+pp5I5/
↪xMe+GDOPsunI5g0Y+YCQKV3h8MXNchdRtQjPybjPmWhyh4hIQ0FibADaVKFU6DK/k1/MFQO+ZxbCtTxK3rUQV2vQa0aHyik93A=-----
↪END CERTIFICATE-----",
  "result": "SIGN_ACCEPT",
  "signature":
↪"7b0b92d773da865b00bd4909f30f577c9e1c57314a0116aea76f5e07e56fd151ba963d1ac93d4928d06e97d3cd9190363ef17ab8eb17aa55d28558efe3
↪",
  "transaction_id": "a19354478c4b4b69b55aae0a4f662e64"
}
```


6 Verifying the Safewalk Fast Auth Signature using OpenSSL

Follow the procedure below to verify a signature that was returned by the Safewalk Signature API.

In our procedure we will use the following returned signature to provide an example for the procedure.


```
{
  "username": "jean.bauer@directaccess.domain",
  "hash": "AD73CA1A90649186D93FBD9FD8433799B3F3D7FC2134155B5160A4A664361961",
  "certificate": "-----BEGIN CERTIFICATE-----
↪MIICpDCCAYwCAQAwDQYJKoZIhvcNAQELQAwGDEWMBQGA1UEAwwNMDAwMjAwMDc0MDQyOTAeFw0xODEyMTYxMjI5MTVaFw0yMzEyMTUxMjI5MTVaMBgxFjAUBgNV
↪Wo06q1A9RsQoM2nqJIhD0cOnern6d0u6TjSw5o5cbCmIjGb2RpJ8CyczW8zX4i+6hp37IqJhOcYxJa4TUOshbYCYJais7TqT3Wdj4uQg2qAynaZcCGdLgWH6Pi
↪A21GVC2bV0Udc8A+VpxX6aBsdIixqVB2XTnSFMbU8cLAflKgwkQokLmeC8XOghmXlhaiSQVdRi3Mon8AsWGYIUYGKm7JXrtrAq+AefWEBtpJiuvS+QjpR+zn1qN
↪5zccQb4YwgeLLTN0S6VDfXORy8g741IKZDxGEAOYBZe2Gx+pp5I5/
↪xMe+GDOPsunI5g0Y+YCQKV3h8MXNchdRtQjPybjPmWhyh4hIQ0FibADaVKFU6DK/k1/MFQO+ZxbCtTxK3rUQV2vQa0aHyik93A=-----
↪END CERTIFICATE-----",
  "result": "SIGN_ACCEPT",
  "signature":
↪"7b0b92d773da865b00bd4909f30f577c9e1c57314a0116aea76f5e07e56fd151ba963d1ac93d4928d06e97d3cd9190363ef17ab8eb17aa55d28558efe3
↪",
  "transaction_id": "a19354478c4b4b69b55aae0a4f662e64"
}
```


- First extract the public key from the certificate:
  - Format the certificate so that the `-----BEGIN CERTIFICATE-----` and `-----END CERTIFICATE-----` parts appear each on a separate line.

Note: If the certificate is not well formatted the next command will output an error similar to the following:

unable to load certificate

140635518915840:error:0906D06C:PEM routines:PEM_read_bio:no start line:../crypto/pem/pem_lib.c:691:Expecting: TRUSTED CERTIFICATE

– Extract the public key from the certificate by executing the command below:

```
openssl x509 -pubkey -noout -in cert.pem > public.pem
```

In our example the file `cert.pem` file will contain the following content:

```
-----BEGIN CERTIFICATE-----
MIICpDCCAYwCAQAwDQYJKoZIhvcNAQELBQAwGDEWMBQGA1UEAwwNMDAwMjAwMDc0MDQyOTAeFw0xODEyMTYxMjI5MTVaFw0yMzEyMTUxMjI5MTVaMB
 ↪Wo06q1A9RsQoM2nqJIhD0cOnern6d0u6TjSw5o5cbCmIjGb2RpJ8CyczW8zX4i+6hp37IqJhOcYxJa4TUOshbYCYJais7TqT3Wdj4uQg2qAynaZ
 ↪A21GVC2bV0Udc8A+VpxX6aBsdIixqVB2XTnSFMbU8cLAflKgwkQokLmeC8XOghmXlhaiSQVdRi3Mon8AsWGYIUYGKm7JXrtrAq+AefWEBtpJiuv
 ↪5zccQb4YwgeLLTN0S6VDfXORy8g741IKZDxGEAOYBZe2Gx+pp5I5/
 ↪xMe+GDOPsunI5g0Y+YCQKV3h8MXNchdRtQjPybjPmWhyh4hIQ0FibADaVKFU6DK/k1/
 ↪MFQO+ZxbCtTxK3rUQV2vQa0aHyik93A=
-----END CERTIFICATE-----
```

And after executing the command the `public.pem` file will contain the following content:

```
-----BEGIN PUBLIC KEY-----
MIIBIjANBgkqhkiG9w0BAQEFAAOCAQ8AMIIBCgKCAQEA5wutXOEM7P1qNOqtQPUb
EKDNp6iSIQ9HDp3q5+ndLuk40sOaOXGwpiIxm9kaSfAsnM1vM1+Ivuoad+yKiYTn
GMSWuE1DrIW2AmCWorO06k91nY+LkINqgMp2mXAhnS4Fh+j4pVt16m4Fs0MwwvM1
KA4oPQn12Oe/wNtRlQtm1dFHXPAPlacV+mgbHSIsalQdl050hTG1PHCwH5SoMJEK
JC5ngvFzoIZl5YWokkFXUYtzKJ/ALFhmCFGBipuyV67awKvgHn1hAbaSYrr0vkI6
Ufs59ajfqkUbu2r+TAgq75RMcBDRVEOwCZb56j2PORajBWCLmYQZuY4qnqwOfVKj
jQIDAQAB
-----END PUBLIC KEY-----
```

- Place the signature content in a file and remove the end of line from it by executing the following command:

```
tr -d "\n\r" < data2sign.txt > data2sign2.txt
mv data2sign2.txt data2sign.txt
```

Important: In case the call to the Safewalk Signature API was done without providing the data parameter (DATA_TO_SIGN_OPTIONAL) the signature content that should be provided here is the SHA256 value of the HASH (CONTENT_HASH) parameter that was sent.

This is done in order to avoid using `NONEwithRSA` in the signature process.

In our example the data that was signed has the following content:

```
I (Jean Bauer), hereby confirm and accept that I have applied for the services indicated by me in␣
 ↪this Application Form (collectively referred to as `services`).
```

- Make sure the hash in the file is the same as the HASH that was sent to the API.

```
sha256sum data2sign.txt
```

In our example the hash value should be the following:

```
ad73ca1a90649186d93fbd9fd8433799b3f3d7fc2134155b5160a4a664361961
```

- Place the hexadecimal signature that was returned in a file and convert the signature to binary format by executing the following command:

```
cat data2sign.txt.hex.signed | sed -e 's/.*= \([^ ]\+\)$/\1/' | xxd -r -p > data2sign.txt.bin.signed
```

In our example the signature file `data2sign.txt.hex.signed` will contain the following content:

```
7b0b92d773da865b00bd4909f30f577c9e1c57314a0116aea76f5e07e56fd151ba963d1ac93d4928d06e97d3cd9190363ef17ab8eb17aa55d28558e
```

- Verify the signature by executing the following command:

```
openssl dgst -sha256 -verify public.pem -signature data2sign.txt.bin.signed data2sign.txt
```

If the signature verification is successful you will see a message similar to the one below:

```
Verified OK
```

6.1 Verifying the checksum signature

Whenever a successful call to Safewalk `sign_push` API is made, the Safewalk server keeps a track of the transaction in its audit log. The audit log contains the signature that was requested as well as an additional signature for the following accumulated content:

- The original data that was sent for signing (this value can be empty if the caller sent only the hash)
- The data hash (this value will be empty if the original data for signing was sent)
- The Safewalk server transaction id
- The UTC timestamp as recorded by Safewalk when the API call was made
- The server URL (this value is optional and is used if it is configured in the `Fast auth` section of the superadmin console)

The accumulated content is called `checksum_data`, the signature on this parameter is called `checksum` and the generated SHA256 of the accumulated content that was signed is provided in the `checksum_hash` parameter.

An example of such audit log row is shown below:

```
{
       'body': u'Confirm?',
       'checksum_hash': ' 13D32EA49252FB69C517615BE5637BB3A8398F9115C16951D37534AAFF1E2964',
       'hash': u'AD73CA1A90649186D93FBD9FD8433799B3F3D7FC2134155B5160A4A664361961',
       'title': u'Confirm by signing',
       'checksum': u
↪'4416497a0d404e852eb336db5863af55489790e9d6cb7caf415e404f855438071fd861612a9f4529963ec41d342bb1ac0e053288742b5d98f582c88796
↪',
       'signature': u
↪'7b0b92d773da865b00bd4909f30f577c9e1c57314a0116aea76f5e07e56fd151ba963d1ac93d4928d06e97d3cd9190363ef17ab8eb17aa55d28558efe3
↪',
       'checksum_data': [u'I (Jean Bauer), hereby confirm and accept that I have applied for the services␣
↪indicated by me in this Application Form (collectively referred to as `services`).', u'', u
↪'9d793aadc3d544f8a4048cc99b8c9e96', u'1544963472', u'https://sign.safewalk.technology']
}
```

The accumulated content that is sent to the user mobile for signature is a concatenation of the `checksum_data` that appears in the audit log. As such before verifying the signature on the `checksum_data` it first needs to be concatenated by removing the `[` and `]` characters from the beginning and the end of the message, the `u` character from the beginning of each parameter in the json array and the `'` character from the beginning and end of each parameter in the json array.

In our example the value of:

```
[u'I (Jean Bauer), hereby confirm and accept that I have applied for the services indicated by me in this␣
↪Application Form (collectively referred to as `services`).', u'', u'9d793aadc3d544f8a4048cc99b8c9e96', u
↪'1544963472', u'https://sign.safewalk.technology']
```

Would need to be converted to the following:

```
I (Jean Bauer), hereby confirm and accept that I have applied for the services indicated by me in this␣
↪Application Form (collectively referred to as `services`).
↪9d793aadc3d544f8a4048cc99b8c9e961544963472https://sign.safewalk.technology
```

After concatenating the `checksum_data` parameter you will be able to verify the signature on this data using the procedure that is described above. Using the concatenated `checksum_data` as the data that was signed,

the `checksum` as the hexadecimal signature and the `checksum_hash` as the expected SHA256 value for the concatenated `checksum_data` parameter.