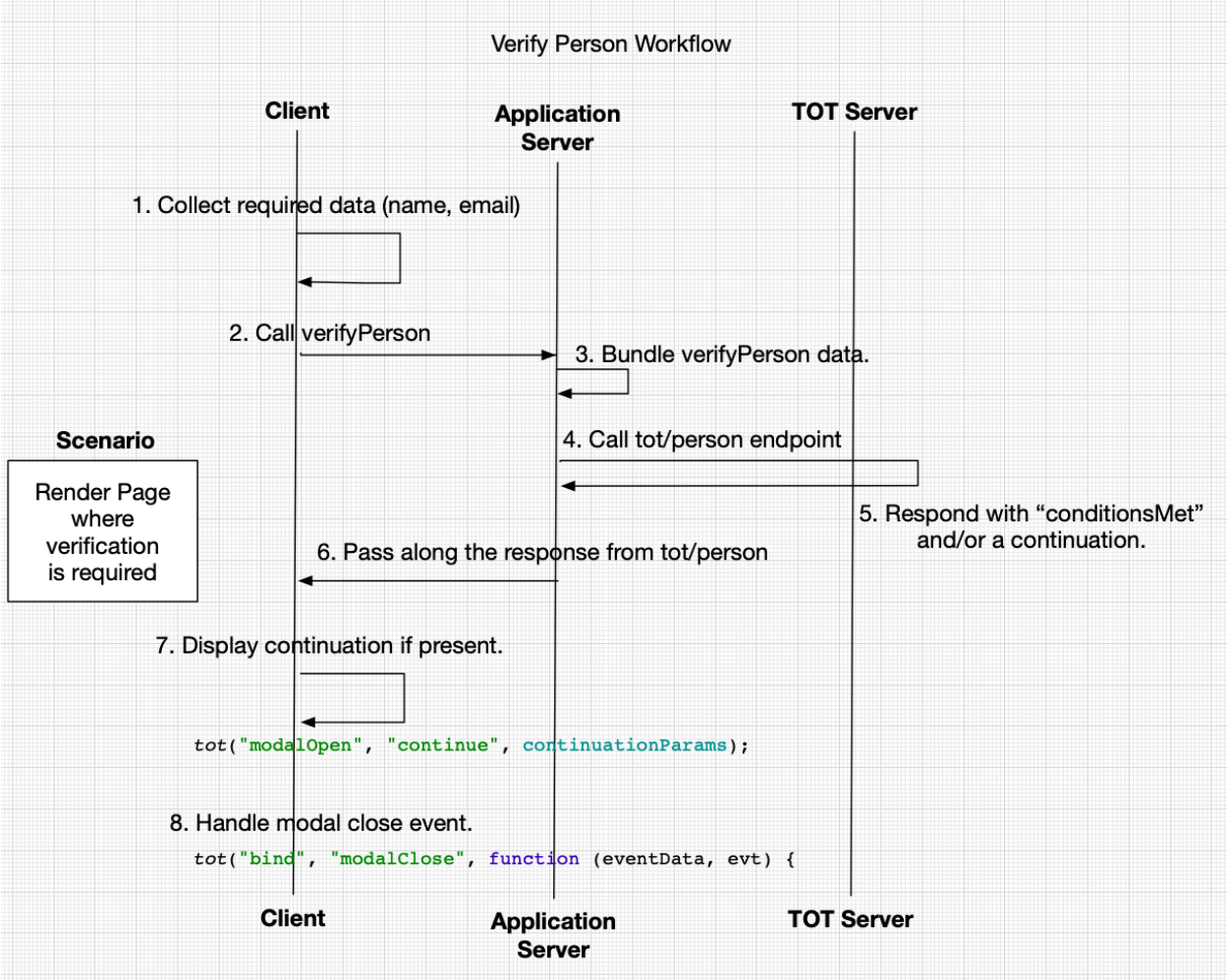
Verify Person API

This API documentation outlines the process of verifying a person using the Token of Trust (TOT) service.

Overview of the Workflow



Definitions

Client - assumed to be a web client of your website or service.
Application Server - your server
TOT Server - generally app.tokenoftrust.com - BUT reach out to support@tokenoftrust.com if you need access to a test environment.

1. (On the Client) Collect required data

You need to pass in name and email. If you have it on the server side or in session already no need to pass it here - this is just a suggestion and how you might do this if you're dealing with a form.

```
const person = {
   givenName: firstName,
   familyName: lastName,
   primaryEmail: email,
 };

const clientVerifyPersonBody = {
   person: person,
 };
```

2. (From the Client) Call verifyPerson (an endpoint on YOUR server)

If you're running a client webpage or webview you will need to bounce a request off your server - mainly this is bc you cannot reveal the tot secret key to the client, however this function is generally a relatively simple passthrough.

Assuming you are creating a webview you will have to create an endpoint on your server to service requests from your web pages. An example of this is below - where `'yourVerifyPersonUrl'` would be replaced with your url.

```
   $.ajax({
       url: yourVerifyPersonUrl,
       type: "POST",
       data: clientVerifyPersonBody,
       dataType: "json",
       timeout: 10 * 1000,
       success: function (content, textStatus, theXhr) {
```

3. (On Your Server) Bundle verifyPerson data.

Next you need to handle the incoming ajax request - by assembling the person data from any combination (this depends upon your use case) from the web page or better your backend. This allows you to put it into an outbound request for the next step.

Here's what the payload needs to look like for the verifyPerson ajax request (which is further below).

```
const personFromClient = (body && body.person);
const person = personFromClient || {
```

```
  givenName: firstName,
  familyName: lastName,
  primaryEmail: email,
};

const verifyPersonBody = {
  appDomain, // your product app domain (e.g. 'example.com').
  totApiKey,
  totSecretKey,
  appUserid: getAppUserid(),  // you define - unique user constant.
  person: person,
};
```

4. (From Your Server) Call tot/person endpoint

Using the `verifyPersonBody` assembled here above - call the tot/person endpoint. Here's a snippet of what that might look like:

```
await request({
  method: "POST",
  uri: "https://" + totEndpoint + "/api/person",
  body: verifyPersonBody,
  json: true, // Automatically stringifies the body to JSON
})
.then(function (response) {
  return processResponse(response);
})
.catch(function (err) {
  return processResponse(err.response && err.response.body);
});
}
```

Some things to note:

`totEndpoint` - is app.tokenoftrust.com unless you've requested a sandbox environment. To request a sandbox environment for pre-release testing contact support@tokenoftrust.com.

5. (TOT) Responds with "conditions met" or a continuation.

When TOT is called - check to see if we have already verified or whether they've already cleared. If TOT determines the configured workflow conditions have been met (code 'conditionsMet') then verification is not required.

Otherwise a 'continuation' is returned. A continuation is a link to a TOT workflow that allows the user to interact with Token of Trust to start or continue the verification process.

6. (On Your Server) Pass along the response from tot/person.

Unless you need to do something on the server side with the response you can generally pass the response along to the client. Here's an example where we strip the response down to the essentials:

```
async function processResponse(response) {
 // console.log(JSON.stringify(response));
 var content = (response && response.content) || {};
 if (content.code === "WorkflowService:conditionsMet") {
   return jsonResponse(ctx, { verificationRequired: false });
 } else {
   return jsonResponse(ctx, {
     verificationRequired: true,
     continuation: content.continuation,
   });
 }
}
```

7. (On the Client) Display the continuation if present.

In the case where Token of Trust verification needs user involvement (5 above responds with a continuation) then you'll need to open our modal to continue the process. Here's how that works:

First you'll want to add this script to your site on any page (on load) where you might want to display our continuations or widgets:

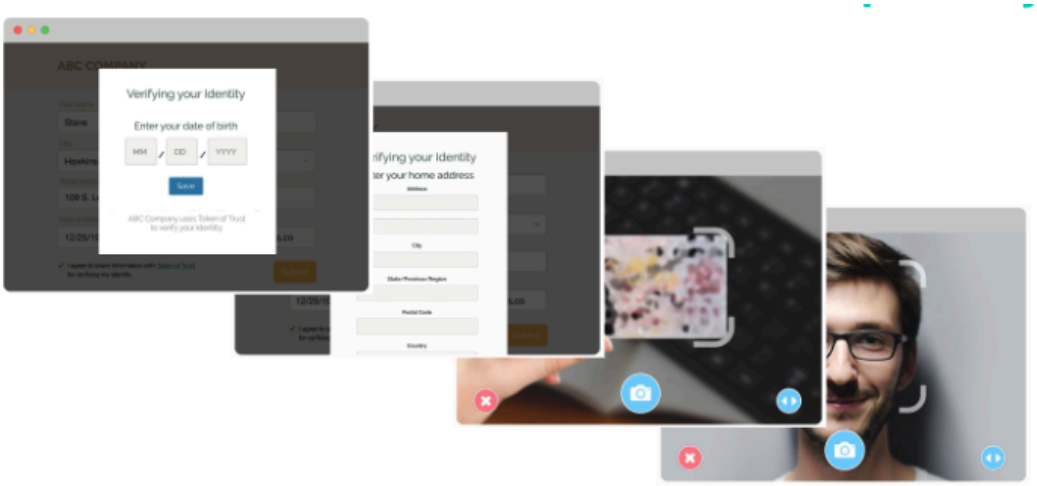
```
function setupTokenOfTrust(bindOptions) {
      // The code below needs to be run on page load.
      var endpoint = "https://" + bindOptions.host + "/embed/embed.js";
      (function (d) {
         var b = window,  a = document; b.tot =  b.tot ||   function () {
      (b.tot.q = b.tot.q || []).push(arguments);  };
         var c = a.getElementsByTagName("script")[0];
         a.getElementById("tot-embed") || ((a = a.createElement("script")),
      (a.id = "tot-embed"),  (a.async = 1), (a.src = d),
      c.parentNode.insertBefore(a, c));
      })(endpoint);
      tot("setPublicKey", bindOptions.apiKey || window.totOpts["apiKey"]);
}
```

If verification is required on the client side you can use the continuation object (from 5 above) and pass it straight into tot to handle like this:

```
function displayContinuation(continuation) {
      continuation.disableClose = true;
      tot("modalOpen", "continue", continuation);
}
```

The first line above prevents the modal from being closed which is the behavior most apps want.

Once you call "modalOpen" - you will see a popup over your app (on mobile we simply appear above your app until done) and then once verification is complete control is seamlessly returned back to your app:



This will create an iframe and pop up a dialog to allow the end user to continue the verification process. Above we're forcing the dialog to remain open until the user finishes the verification process - it's optional but may make sense for you.

8. (On the Client) Handle Modal Close event

Once the end user is done - the modal will close and control will return to your app. To be informed of this you will probably want to register for this event so you can take action. Here's an example that's done:

```
tot("bind", "modalClose", function (eventData, evt) {
   console.log("[CONSOLE] my modalClose event called!");
   verifyPersonInProgress = false;
   if (evt && evt.data) {
       var eventData = {};
       try {
           eventData = JSON.parse(evt.data);
       } catch (err) {
           return false;
       }
       eventData = eventData.data || eventData;
       if (eventData.continue) {
           if (lastEvent.eventTimestamp === evt.timeStamp) {
               return; // don't process invalid data, nor the same event
twice.
           }
           lastEvent.eventTimestamp = evt.timeStamp;
           checkVerificationStatusAndDoSomething(); // your function.
       }
   }
  return false;
});
```

Code Sample

Please note these are untested examples drawn and simplified from working production code. That these code samples do require the tot snippet installed and reference some functions in the Utility Functions defined in the appendix.

1. Ajax request (verifyPerson) from the client side.

This assumes that you have assembled the verifyPersonBody (see example above).

```
function doVerifyPerson(verifyPersonBody, autoOpen, onVerified, onError) {
   var verifyPerson = window.totOpts["verifyPerson"] || {};
   var verifyPersonUrl = verifyPerson.url || window.totOpts["verifyPersonUrl"];

   if (verifyPersonInProgress) {
       return;
   }
   verifyPersonInProgress = true;
   $.ajax({
       url: verifyPersonUrl,
       type: "POST",
       data: verifyPersonBody,
       dataType: "json",
       timeout: 10 * 1000,
       success: function (content, textStatus, theXhr) {
           var continuation = content && content.continuation || content;
           var continuationParams =
               content.continuation && content.continuation.params || {};

           var verificationMessageCode, verificationRequired;
           if (content && (content.verificationRequired === undefined ||
content.verificationMessageCode === undefined)
               && content.code === "WorkflowService:conditionsMet") {
               const gates =
                   content.details &&
                   content.details.conditionResult &&
                   content.details.conditionResult.gates;
               const isCleared = reasonIsPositive(gates["isCleared"]);
               const isNotRequired = reasonIsPositive(
                   gates["isNotRequired"]
               );
               const isSubmitted = reasonIsPositive(gates["isSubmitted"]);
               const isUpdateRequested = reasonIsPositive(
                   gates["isUpdateRequested"]
               );
```

```
                verificationRequired = false;
                if (isNotRequired) {
                    verificationMessageCode = undefined;
                } else if (isCleared) {
                    verificationMessageCode = "isCleared";
                } else if (isUpdateRequested) {
                    verificationMessageCode = "isUpdateRequested";
                    verificationRequired = true;
                    console.error('TOT: conditionsMet and update requested!',
content);
                } else if (isSubmitted) {
                    verificationMessageCode = "isSubmitted";
                }
            }

            if (verificationRequired !== false && continuationParams.url) {
                if (autoOpen) {
                    // When required - use continuationParams to start the
verification process using the tot function.
                    continuationParams.disableClose = true;
                    tot("modalOpen", "continue", continuationParams);
                }
            } else if (verificationMessageCode && verificationMessageCode ===
"error") {
                verifyPersonInProgress = false;
                return onError(content.errorMessage);
            } else {
                verifyPersonInProgress = false;
                return onVerified(content);
            }
        },
        error: function (theXhr, textStatus, errorThrown) {
            console.error("Failure(%s) in POST to %s", textStatus,
verifyPersonUrl);
            verifyPersonInProgress = false;
        },
    });
}
```

2. Server side example of ‘verifyPerson’ endpoint as implemented in koa / node.js

```
async function verifyPerson(options) {
 const { apiClient } = options;
 const appDomain = apiClient.getAppDomain();
```

```
 const person = {
   givenName: address.first_name,
   familyName: address.last_name,
   primaryEmail: order.email,
 };

// Person - advanced / optional parameters:
//   location: buildLocation(address),
//   shippingLocation: buildLocation(order.shipping_address),
//   billingLocation: buildLocation(order.billing_address),
//   orderNumber: order.name,

 const verifyPersonBody = {
   appDomain,
   totApiKey: apiClient.getApiKey(),
   totSecretKey: apiClient.getSecretKey(),
   appUserid: utils.hashFromString(
     order.email.toLowerCase() + "-" + apiClient.getAppId(),
     "SHA3-256",
     "hex"
   ),
   person: person,
 };

// Additional optional/advanced parameters:
//  guest: true,
//  appTransactionId,
//  appReservationToken: reservationToken,
//  traceId: (cart && cart.token) || checkout.cartToken,
//  browserFingerprint: fingerPrint,

 var vpOptions = {
   method: "POST",
   uri: "https://" + totEndpoint + "/api/person",
   body: verifyPersonBody,
   json: true, // Automatically stringifies the body to JSON
 };

async function processResponse(response) {
 // console.log(JSON.stringify(response));
 var content = (response && response.content) || {};
 if (content.code === "WorkflowService:conditionsMet") {
   return jsonResponse(ctx, { verificationRequired: false });
 } else {
   return jsonResponse(ctx, {
     verificationRequired: true,
     continuation: content.continuation,
```

```
    });
  }
}

  await request(vpOptions)
  .then(function (response) {
    return processResponse(response);
  })
  .catch(function (err) {
    const body = err.response && err.response.body;
    return processResponse(body);
  });
}
```

Appendix

verifyPerson Body

Attribute	Value	Description
appDomain	string	Your product app domain
totApiKey	string	API key for Token of Trust
totSecretKey	string	Secret key for Token of Trust
appTransactionId	string	(optional) Unique id for the transaction that is being verified
appUserid	string	(optional) A unique id for the user being verified
traceId	string	(optional) A identifier used to track a verification across multiple API calls.
reservation	boolean	Indicates whether the verification is a reservation.
appReservationToken	string	A token associated with the reservation.

Attribute	Value	Description
guest	boolean	Indicates whether the user is a guest.
appDataConsentRequested	array[string]	(optional) A list of fields to request consent from the user to share. See below for a list of fields that are supported.
person	object	Users details (see below)

Person

Attribute	Value	Description

dateOfBirth	object or string	The user's date of birth, either as a string or an object of the following structure:
		`{`
		`day:1,`
		`month:1,`
		`year:2000`
		`}`

location	object or string	The user's location as a string or an object of the format:
		`{`
		`line1:STRING,`
		`line2:STRING,`
		`locality:STRING,`
		`region or regionCode:STRING,`
		`countryName or countryCode:STRING,`
		`postalCode: STRING`
		`}`

shippingLocation	object or string	User's Shipping location as a string or an object of the format:

Attribute	Value	Description

		{
		line1:STRING,
		line2:STRING,
		locality:STRING,
		region or regionCode:STRING,
		countryName or countryCode:STRING,
		postalCode: STRING
		}

billingLocation	object or string	User's billing location as a string or an object of the format:
		{
		line1:STRING,
		line2:STRING,
		locality:STRING,
		region or regionCode:STRING,
		countryName or countryCode:STRING,
		postalCode: STRING
		}

email	string	The user's email address
phoneNumber	string	The user's phone number
fullName	string	The user's full name as a string

givenName	object or string	The user's given name as a string or object with the format:
		{
		current: STRING
		previous: STRING
		}

familyName	object or string	The user's family name as a string or object with the format: {

Attribute	Value	Description
		maternal: STRING, paternal: STRING current: STRING previous: STRING }
middleName	string	The user's middle name.

User data that can be requested

When appDataConsentRequested has been supplied and the user in fact consents to supplying that data using the TOT UX we can supply these files back via the API.

We currently support either govtId fields or tot (Token of Trust) fields as described below. Please remember that while we supply these fields back you should refer to the reputation report to understand the verification status of any given piece of information. (i.e. just because they supplied a govtId that has the name 'Smith' on it does not mean that the consumer was verified as having that name.

govtId fields

These are fields that are collected from the government id

- govtId.givenName
- govtId.familyName
- govtId.dateOfBirth
- govtId.expirationTimestamp
- govtId.countryCode
- govtId.documentType

tot fields

These fields are set during the initial verification and can be changed by the consumer on their dashboard.

- tot.givenName
- tot.familyName
- tot.location
    - tot.location.line1
    - tot.location.line2
    - tot.location.locality
    - tot.location.region
    - tot.location.countryCode
    - tot.location.postalCode
- tot.shippingLocation
    - tot.shippingLocation.line1
    - tot.shippingLocation.line2
    - tot.shippingLocation.locality
    - tot.shippingLocation.region
    - tot.shippingLocation.countryCode
    - tot.shippingLocation.postalCode
- tot.billingLocation
    - tot.billingLocation.line1
    - tot.billingLocation.line2
    - tot.billingLocation.locality
    - tot.billingLocation.region
    - tot.billingLocation.countryCode
    - tot.billingLocation.postalCode
- tot.dateOfBirth

Utility Functions

We use a number of utility functions above.

```
// Some utility classes...
function reasonIsPositive(reason) {
   var theReasonValue = reasonValue(reason);
```

```
    return !!(theReasonValue && theReasonValue !== 'false' && theReasonValue !==
'noMatch' && theReasonValue !== 'no_match' && theReasonValue !==
'insufficientData' && theReasonValue !== 'notApplicable');
}

function handleNotApplicable(value) {
    return (value === 'notApplicable') ? undefined : value;
}

function reasonValue(reason) {
    var retVal = reason;
    if (isBoolean(reason)) {
        retVal = reason;
    } else if (isObject(reason)) {
        retVal = reason.value !== undefined ? reason.value : reason.status;
    }
    return handleNotApplicable(retVal);
}

function isObject(possibleObject) {
    return (possibleObject !== undefined) &&
Object.prototype.toString.call(possibleObject) === '[object Object]';
}

function isBoolean(possibleBool) {
    return (possibleBool !== undefined) &&
Object.prototype.toString.call(possibleBool) === '[object Boolean]';
}
```

Verify Person FAQ

How do I get the TOT modal to close?

TOT modals should close when the verification process is complete. Some of the endpoints don't have an auto-close because they're requesting further information.

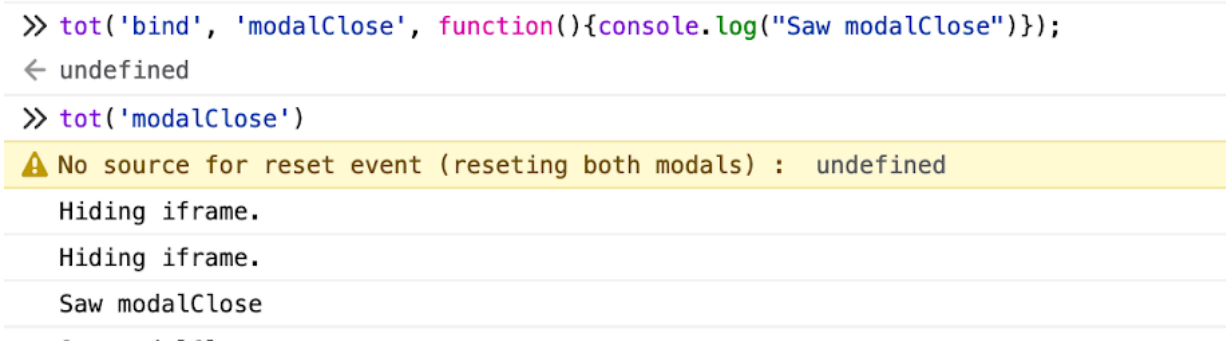
We try hard to ensure these dialogs close on their own BUT if you get to a terminal point where the user gets verified there should be an auto-close. If that's not what you're seeing, please send us a screenshot of the screen you're stuck at and we'll look at that in more detail to success@tokenoftrust.com!

If you think you have a legit reason to not rely upon the auto-close please read on!

Alternatively, you can allow the user to manually close the dialog and we can make this possible by setting disableClose before opening the modal by adding the disableClose attribute to the continuation object before opening the modal (see the bolded code below)... This setting now shows an 'x' to close the TOT dialog. When you close the modal using the 'x' the callback from the tot("bind" call will fire.

```
function displayContinuation(continuation) {
   continuation.disableClose = false;
   tot("modalOpen", "continue", continuation);
}
```

Another way to trigger this in javascript is by calling `tot('modalClose')` from the console. Below I show in the console binding to the modalClose event to print a console message and then I close the TOT window with javascript to give you a sense of how the API works on the web client:



Webhooks Overview

This document covers webhooks and is broken up into a couple parts. The first part is what you're reading and it describes the feature, how it works and what you'll receive. The FAQ describes common questions and problems.

If you're looking to add webhooks to your Token of Trust implementation we recommend that you:

1. read through this section to understand how it works and to get your webhooks implemented and tested
2. reach out to us (see configuring your api keys to receive webooks)
3. then of course check out the FAQ to understand common issues.

a. Of particular importance for a developer / tester of webhooks is this item which lets you ask the system to send you mock webhooks for testing and the like: Is it possible to have the Token of Trust server send a test webhook request?

Happy Webhooking and please don't be afraid to contact us at support@tokenoftrust.com if you run into any trouble.

Real time Notification of Gate Changes

TOT will send an update when a verification moves between gates (e.g. submitted, cleared, rejected)

Note: at this time we have limited retry capabilities for webhooks…

Structure of the webhook

The webhooks notification is a POST request with a JSON body and contains the following fields:

- **app_user_id:** verifications appUserId
- **app_transaction_id:** (if applicable) transaction id for the verification
- **operation**: The type of change experienced (Webhook Operations Supported)
- **gates:** list of the gates and their values for the verification
- **reasons:** list of reasons and their values for the verification

Webhook Operations Supported

- isCleared - verification has cleared based on your configuration.
- isUpdateRequested - Reviewer has requested updated information about the user.
- isSubmitted - User has submitted all documentation and is waiting for a review.
- isRejected - User has failed to meet the requirements to get verified based on your configuration.

Reason and Gate Values

- fullMatch, paritalMatch, true -Indicates that the verification meet the requirements of the reason or gate. A partialMatch indicates that the verification didn't meet the requirement perfectly. Some examples of partialMatch are: image was a little blurry, user used a nickname instead of exact name, government id will expire soon.
- noMatch, false - Verification did not match the requirements for this reason or gate.

Example


```
{
	"app_transaction_id": "R:MXAOM3NVK9yUv62V",
	 "app_user_id": "vYRsWXlypTzRAS42gbZrQR1i/m+bO9K0z59w00Fz9IU=",
	"app_domain": "mySite.com",
	"reasons": {
		"appReviewNotRejected": "noMatch",
		"govtIdAppReviewerApproved": "noMatch",
"govtIdTotReviewerApproved": "noMatch",
		"govtIdPositiveAppReview": "noMatch",
		"selfieIsGoodHeadShot": "fullMatch",
		"community_isOnlineVerified": "noMatch",
		"community_accountDataIsConsistent": false,
		"ageVerified": "noMatch",
	},
	"gates": {
		"isConditionallyVerified": "noMatch",
		"isApproved": "noMatch",
		"isPositiveReview": "noMatch",
		"isCleared": "noMatch",
		"isSubmitted": false,
		"isUpdateRequested": "noMatch",
		"isRejected": "fullMatch"
	},
	"eventId": "txeQPiToZz",
	"name": "verification.isRejected",
	"operation": "isRejected",
	"timestamp": "Mon, 18 Jul 2022 13:03:54 GMT"}
```


HMAC Signatures

Every webhook request is signed with an HMAC digest and set into the 'Authorization' http header. Doing this both protects the request from being tampered with and ensures that the request came from Token of Trust.

Here's the code used to generate the HMAC in node.js. Integrators should verify all incoming requests by regenerating them with their webhookSecretKey and comparing them against the contents of the Authorization header. Any requests that do not match should be discarded.

```
/**
* Takes the webhook body and creates a hmac based on the secret key.
* @param {string} body - The returned body.
* @param {string} webhookSecretKey - The current webhook secret access key.
* @returns {string} hmac - the hmac generated.
*/
function generateSignature(body, webookSecretKey) {
  var hmac = crypto.createHmac('sha1', webookSecretKey);
  hmac.update(body, 'utf8');
  return hmac.digest('base64');
}
```

Recommendations for Processing and Storage

First and foremost we ask that you follow best practices related to web hooks:

a. Dispatch our post requests quickly - saving the authorization header body.
b. Close the request to free resources.
c. Then process the hook - i.e. check the signature, make database requests, etc.

Storage

Most people will want to save the result of the hooks to a database. What we recommend is that you record the state of the reasons that come back in a table with user metadata. How you do so is up to you but a simple solution is to store the following:

Attribute	Value	Description
`timestamp`	UTC Timestamp String	You want to store this because it is possible for webhooks to be delivered out of order. You'll want to process an incoming webhook only if the timestamp of the incoming request is > the timestamp stored.
`totStatus`	String	The gates (referenced above) are good status indicators.

Webhooks FAQ

How do I start receiving webhooks?

Webhooks are turned on by request. When you think you're ready please contact support@tokenoftrust.com and let us know what your test and live endpoints will be for your webhooks. We'll get you setup.

Testing Webhooks - How do I know my hooks work?

We've seen customers with problems identifying why they aren't receiving our hooks. One of the great things about webhooks is that you can easily test them. We encourage you to both unit test them and integration test them.

Curl is a handy way to do this:

```
curl -k -H "authorization:foo" -X POST -d 'test' \
 https://yourserver.com/webhooks/tot-reputation
```

If you find this works for you but you're still not getting our hook notifications please let us know!

See "Is it possible to have the Token of Trust server send a test webhook request?"

How do I check the webhook signature?

Checking Webhook Signature: NodeJS

```
/**
* Confirms that a given signature matches the body.
* @param {string} req - The incoming webhook request.
* @param {string} webhookSecretKey
* @returns {string} hmac - the hmac generated.
*/
function checkReqSignature(req, webhookSecretKey) {
  if (!(req && req.headers && req.body) || !webhookSecretKey) {
    return false;
  }

  var body = utils.isObject(req.body) ? JSON.stringify(req.body) : req.body; // In case already parsed the body...
  var signature = req.headers['authorization'];

  if (!signature || !body || !webhookSecretKey) {
```

```
    return false;
  }
  var hmac = crypto.createHmac('sha1', webhookSecretKey);
  hmac.update(body, 'utf8');
  var calculatedSignture =  hmac.digest('base64');

  return calculatedSignture === signature;
}
```

Checking Webhook Signature: PHP

```
$auth_header = $request->get_header('Authorization');
$calculated_signature =
       base64_encode(hash_hmac('sha1', $request->get_body(), TOT_LICENSE_KEY, true));
```

Is it possible to have the Token of Trust server send a test webhook request?

Yes! We've built an API (api/webhooks/test) for this purpose - here's an example curl request for illustration. This method is dumb and will send out exactly the webhook body told it to send out. Note this functionality is deprecated in our production environment so here we are using the sandbox endpoint. If you'd like access to sandbox please reach out to us at support@tokenoftrust.com.

```
curl -X POST \
  https://sandbox.tokenoftrust.com/api/webhooks/test \
  -d '{
  "appDomain" : "YOUR_APP_DOMAIN",
  "totApiKey" : "public_test-YOUR_KEY",
  "totSecretKey" : "secret_test-YOUR_SECRET",
  "webhook" : {
        "name" : "reputation.created",
        "body" : {"reputation" : {"foo" : "bar"}}
  }
}'
```

The webhook.name parameter is required. The webhook.body parameter is not validated and may contain anything you wish to simulate whatever conditions you like.

Example Response:

```
{
  "content": {
    "sentWebhookBody": {
      "reputation": {
        "foo": "bar"
      },
      "testMode": true,
      "name": "reputation.created",
      "resource": "reputation",
      "operation": "created",
      "timestamp": "Sat, 14 Apr 2018 03:46:12 GMT"
    },
    "webhookConfig": {
      "reputation.created": {
        "hookUrl":
"http://my.findaplace.xyz:32080/webhooks/reputation.created"
      },
      "reputation.updated": {
        "hookUrl":
"http://my.findaplace.xyz:32080/webhooks/reputation.updated"
      }
    }
  },
  "metadata": {
    "version": "1.0.1",
    "status": 200
  }
}
```


As usual it follows the normal TOT api response.

content.sentWebhookBody - is what was actually sent to the client.
content.webhookConfig - is your current webhook configuration (on this server).