Best Practices: Securing your OpenTok App

TokBox recognizes that security is an essential consideration for any business interested in integrating real time communications into its website, app or service. The OpenTok platform is a reliable and secure platform on which you can build applications that meet your company, industry or client security needs.

Whether you're new to the OpenTok platform, or have years of experience, here is a useful set of best practices you can employ when developing with OpenTok to help you build a secure application.

Best Practice	Details
Personal Information	
• Keep the API key and secret private and secure	The API key and secret are used to create tokens that grant access to sessions, retrieve archive metadata and change archive storage credentials, as well as other administrative operations on your account. To avoid compromising your credentials, you should **always** keep your API key and secret private. Some key measures you can take: - Never save the API key and secret in any public source code repositories - Never save the API key and secret in any client side libraries, or even compiled mobile SDKs - Use only https URLs to make REST calls to the OpenTok servers
• Don't put personal information in URLs	The URLs that you generate are shared with multiple end users and are logged on the OpenTok servers. Therefore any sensitive information that forms part of the URL could be stored and/or accessed by unprivileged users. To avoid this: - Ensure URLs are generic and not intuitive. For example, URLs should not be serially numbered (e.g. abc.com/session1, abc.com/session2), or easily guessed based on the nature of the call (e.g. abc.com/tutoring) - Do not include any information that could be sensitive in the URLs

• Generate a unique Session ID per call and token per participant	You need to generate a Session ID to initiate a call. The tokens that enable the participants to join are unique to a Session ID. The tokens have an expiry but it may be longer than the duration of your call. Therefore, if you have consecutive meetings using the same session ID, earlier users may still be able to connect to the new meeting. To avoid this: - Generate a unique Session ID for each new meeting - Generate a unique token for each participant of that meeting. See here on how to generate tokens and sessions.
• Ensure server generating token is behind authenticated endpoint	It is important to place the server generating the token behind an authenticated endpoint because anyone with access to that server could end up generating new tokens and could abuse the app to generate usage.
• Don't use personal information in token data	The token data is a string containing metadata describing the connection. However, this data is passed to all users in the session and is also readable through the OpenTok client logs. This means you should **never** use unencrypted sensitive or personal information in the toke data. See here on how to add data to your tokens.
Relayed vs Routed Mode	
• End to End Media Encryption	During a routed session, media streams are temporarily decrypted while within the OpenTok Platform cloud servers and then immediately re-encrypted prior to being sent through the internet to the subscribing client. This decryption is necessary for managing group sessions, intelligent quality control, and archiving of sessions (if used). Using routed sessions, your media streams are never transmitted unencrypted on the open internet. However, if your application requires uninterrupted end-to-end encryption of all media, you may choose to use relayed sessions. Be aware that you would not be able to use archiving, and performance will not be managed as well in low bandwidth / high packet loss networks or with groups.
Signaling	
• Encrypt signaling data	The signaling data is transmitted over encrypted websocket connections. This means that data is always transmitted over secure networks and OpenTok servers do not store any signaling data sent. However, to secure your app further, you may also choose to add

	your own encryption and send encrypted data as payload over signals.
Archiving	
• Manage archive deletion	An archive successfully uploaded to your storage will be automatically deleted from the OpenTok archiving server at the time of upload. In case of failure to upload, OpenTok storage is provided as a default fallback option. This means the archive will be stored for 72 hours on the OpenTok server. You will be alerted via email for every archive that fails to reach your storage. You can then use the REST API to download the archive from the S3 bucket. Following the download, you can choose to immediately delete the archive to avoid it being on the OpenTok storage for the remainder of the fallback period. See here for more information on how to delete an archive Note: Customers who do not wish to use the fallback functionality have the option of disabling it. If disabled, following the attempt to upload to a customer's S3 bucket, the archive will automatically be deleted irrespective of whether the upload is successful.
• Encrypt archives	Using REST API it's possible to encrypt archives using partner certificates (the password is stored as part of the metadata on the archive). Upon accessing the archive, you can then use the password to decrypt the archives and access the media. Please note this is currently a beta API.
• Control who is able to begin archiving	You can only archive a session using the REST API. To control who can initiate the archive, you can programmatically decide which view of the application includes the option to start archiving. Those unauthorized would have a limited view with no option to archive. The view with ability to archive can be protected behind a username/password authentication and the use of unique authentication to initiate the archive every time.
• Ensure minimum set of privileges	The minimum set of permissions required to upload archives to our storage is mentioned here. Partners should not provide any additional permissions for the credentials that they store with OpenTok.

• **Turning Archiving Fallback Off**	For some TokBox customers, and in some specific sectors and industries, it is paramount to insure that all information that is discussed within a session is either kept on the customer's data repository or deleted. For some EU customers, due to the nullity of the Safe Harbor agreement, this is also part of a temporary solution to ensure that no customer information is stored outside of the EU. In order to insure this, you can turn Archiving Fallback Off. Here you have a sample of the code that manages the fallback: PUT /v2/partner/{partnerId}/archive/storage Accept: application/json X-OPENTOK-AUTH: YYYYYY (scope: archive.storage.update) | X-TB-PARTNER-AUTH: ZZZZZZ IN {type: "s3", config: {...}, fallback="none | opentok"} OUT {type: "s3", config: {...}, createdAt: ..., updatedAt: ..., fallback="none | opentok"} An example config: might be {"accessKey":"myUsername", "secretKey":"myPassword", "bucket": "bucketName"} In this case you can see that both in IN and OUT there is a fallback option, which can be set to "none" or "opentok". In both cases, you should set the option to "none". Here is an example of disabling fallback if you are using curl: curl -H "X-TB-PARTNER-AUTH: {api_key}:{api_secret}" -H "Content-Type: application/json" -X PUT -d '{"type": "s3", "config": {"accessKey":"{aws_accesskey}", "secretKey":"{aws_secretkey}", "bucket": "{aws_bucket}"}, "fallback":"***none***"}' https://anvil.opentok.com/v2/partner/{partnerId}/archive/storage In this case, you should also make sure that "fallback": is set to "none". Here is an example with a partner set: curl -H "X-TB-PARTNER-AUTH: 44844632:SECRET" -H "Content-Type: application/json" -X PUT -d '{"type": "s3", "config": {"accessKey":"fake", "secretKey":"fake", "bucket": "fake"}, "fallback":"***none***"}' https://anvil.opentok.com/v2/partner/44844632/archive/storage Once this is set up, every time an archived session finishes, it will try to upload it to the respective Amazon S3 or Windows Azure account. If it fails, the archiving data will be lost.
Alerts and Controls	
• **Limit maximum number of users in a session**	In order to have more control over the number of people in a session, it is possible to limit the maximum number of users in a session. This could be useful if you're trying to limit your usage.

• Set up an alert for when a participant is connected but not publishing	For your application, particularly if it requires privacy and confidentiality, you may wish to set up an alert to advise you when a user is connected to the session but is not publishing via camera.
• Set up Moderator permissions to force disconnect	OpenTok platform provides the capability to remove a user from a session. For example, in case of violation of terms of services, you can enable the moderator in the session to remove the violating participant from the session via force disconnect or force unpublish. Find out how here
• Allocate Subscriber-only permissions for those that do not need to publish	In some use cases you may want to limit the number of people who can publish in a session. This is easily implemented by not extending publish permissions to every participant. See here for generating tokens with the right roles.