VivoKey Scan API

Definitions

Some transponder industry terms are built around the concept of "card" technology. Modern passively coupled chip technology has moved well beyond the card form factor, but within the context of this document, consider the term "card" to mean the passive transponder device.

> Member ID = a unique ID for a ***person*** from hash of VivoKey ID + Salt + Developer ID
> Chip ID = a unique ID for a ***VivoKey implant*** from hash of Chip ID + Salt + Developer ID
> CARD = passive transponder device (chip + antenna + form factor)
> PCD = proximity coupling device (the card reader, NFC reader, smartphone, etc.)
> PICC = proximity integrated circuit card (the chip implant, transponder, etc.)
> Chip Scan = the series of events which occur while PICC and PCD are interacting

API Keys

We use API keys to help secure calls to critical endpoints. To get an API key, you must create a developer account at https://developer.vivokey.com then create your keys. Keys are displayed only once, so it is important you document your key immediately after it is generated.

Member ID

All claimed VivoKey implants associated with an active VivoKey member profile will return a Member ID with a successful call to ***check-result***. A Member ID is a hash value of the person's VivoKey ID (which is a VivoKey internal identifier), a secret salt value, and your developer account ID. This means any API key generated under your developer account will return the same Member ID for the same person, even if they have multiple chip implants associated with their VivoKey member profile. However the Member ID for this person will not be the same for other developers. This protects VivoKey members against unintended data collection or privacy invasion by comparing or matching Member IDs across different services. Sharing of Member IDs, API keys, or developer accounts across multiple organizations **is not allowed**.

Chip ID

The Chip ID for a VivoKey implant is only returned by a successful call to ***check-result*** if your developer account is properly licensed to verify authenticity of VivoKey chip implants directly. Like Member ID, Chip IDs are a hash value of the chip's internal identifier, a secret salt value, and your developer account ID. Sharing of Chip IDs, API keys, or developer accounts across multiple organizations **is not allowed**.

Endpoints

- https://api2.vivokey.com/v1/get-challenge - obtain temporary crypto challenge
- https://api2.vivokey.com/v1/pcd-challenge - submit PCD challenge from PICC
- https://api2.vivokey.com/v1/check-response - check PICC challenge response

get-challenge

Obtains a temporary cryptographic picc-challenge from the server. The intention is to obtain this challenge before any chip scan event occurs so the PCD is ready to initiate the cryptographic process and quickly deliver the challenge to the PICC, rather than a chip scan event starting and then having to initiate an http request to get a challenge.

Accepts json dictionary containing the following

name	length	description
api-key	60	The hex encoded 30 byte API key issued via developer account

Example ***get-challenge*** request

```
{
	"api-key": "3dd9f2c8bd13e803cbc599cf32875c9d9fbf854dc419d9a79c517e98c21a"
}
```

Returns json dictionary containing the following

name	length	description
picc-challenge	32	A 16 byte hex encoded string PICC challenge. Use of all 16 bytes may not be necessary depending on PICC type and expected challenge. LSBs may be truncated to required challenge length.
timeout	int	Integer value equal to the number of seconds the PICC challenge is valid. After timeout, a call to /check-response will fail regardless.

Example ***get-challenge*** response

```
{
	"picc-challenge": "17695fd27eaf8c65833d50cbff12a501",
```

```
    "timeout": 30
}
```

pcd-challenge

For certain chips which only support mutual authentication (Spark 2), the chip will send a challenge to the PCD first. In this scenario we must conduct a call to pcd-challenge during a chip scan event so the backend can process the challenge issued by PICC to the PCD.

Accepts json dictionary containing the following

name	length	description
picc-uid	20	PICC uid represented in hexadecimal notation, all capital letters, no separators. Ex. 0428375BFA44D7 for a 7 byte UID (Spark 2).
picc-challenge	32	The full hex string challenge from /get-challenge (PICC challenge)
pcd-challenge	32	The hex encoded PCD challenge string issued from PICC

Example ***pcd-challenge*** request

```
{
    "picc-uid": "04F2DA739E2BA0",
    "picc-challenge": "17695fd27eaf8c65833d50cbff12a501",
    "pcd-challenge": "55ecc39e823fd6c7c244f0d14a127f28"
}
```

Returns json dictionary containing the following

name	length	description
pcd-response	64	The full hex encoded PCD response string to send back to PICC.

Example ***pcd-challenge*** response

```
{
    "pcd-response": "358dcaf0dfdd7ffa15c74e0ef2eeaa9be376d2ad7d20868c8079a2f545b00522"
}
```

check-response

The server will process the PICC challenge response to check authenticity. If the PICC challenge and PICC response match, the chip scan event will be considered a success and the authenticity of the chip will be validated. If this occurs, the PICC association will be checked to ensure it is associated with an active VivoKey profile. If so, an ID will be returned.

Accepts json dictionary containing the following

name	length	description
picc-uid	20	PICC uid represented in hexadecimal notation, all capital letters, no separators. Ex. 0428375BFA44D7 for a 7 byte UID (Spark 2).
picc-challenge	32	The full hex string challenge from /get-challenge (PICC challenge)
picc-response	32	The hex encoded PICC response to PICC challenge*

**Some products only process 10 bytes of a challenge. In those cases the picc-challenge must truncate LSBs to shorten to a length of 10 and append any required flags before sending to the product for processing. For example, the picc-challenge "qcjNbPbcyOCLL3Fc" is passed as "qcjNbPbcyO" to these products. However, the full picc-challenge is always used when calling the API.*

Example ***check-response*** request (if product processes 16 byte challenge/response)

```
{
      "picc-uid": "04F2DA739E2BA0",
      "picc-challenge": "741f849e6650d4d97445ca1384856fd9",
      "picc-response":
"e79b8d9fbbac5cdb484b4851c743c1c530b623483021e16f11e19cf2c71691b3"
}
```

Returns json dictionary containing the following

name	length	description
check-result	11	Simple human readable response code (table 1)
result-data	128	Relevant data returned from check result (table 1)

Table 1. Possible check-result codes

check-result	meaning	result-data
error	Error codes are non-specific. This is to ensure no unintentional state or status data related to the chip scan event is unintentionally leaked.	*<empty>*
member-id	Indicates success, and result-data will contain the 64 byte unique Member ID of the VivoKey member.	Unique 64 byte hex encoded member ID
chip-id	Indicates success, and result-data will contain the 64 byte unique Chip ID of the VivoKey chip.	Unique 64 byte hex encoded chip ID
chip-member	Indicates success, and result-data will be an array consisting of the 64 byte unique Chip ID, then the 64 byte unique Member ID	[chip-id, member-id]

Example ***check-result*** response returns 64 byte hex encoded Member ID

```
{
     "check-result": "member-id",
     "result-data":
"75e95abb37ee6e413ce5431f3785023312093c7fb62deb77dbe4c34a3a108227ae28
e61998029c3edbc6393f84e0422315dc4c87e10fe31db4a435790f051579"
}
```

Example ***check-result*** response returns chip-member response with two IDs; the Chip ID of the scanned chip and the Member ID the of the VivoKey member the chip is associated with

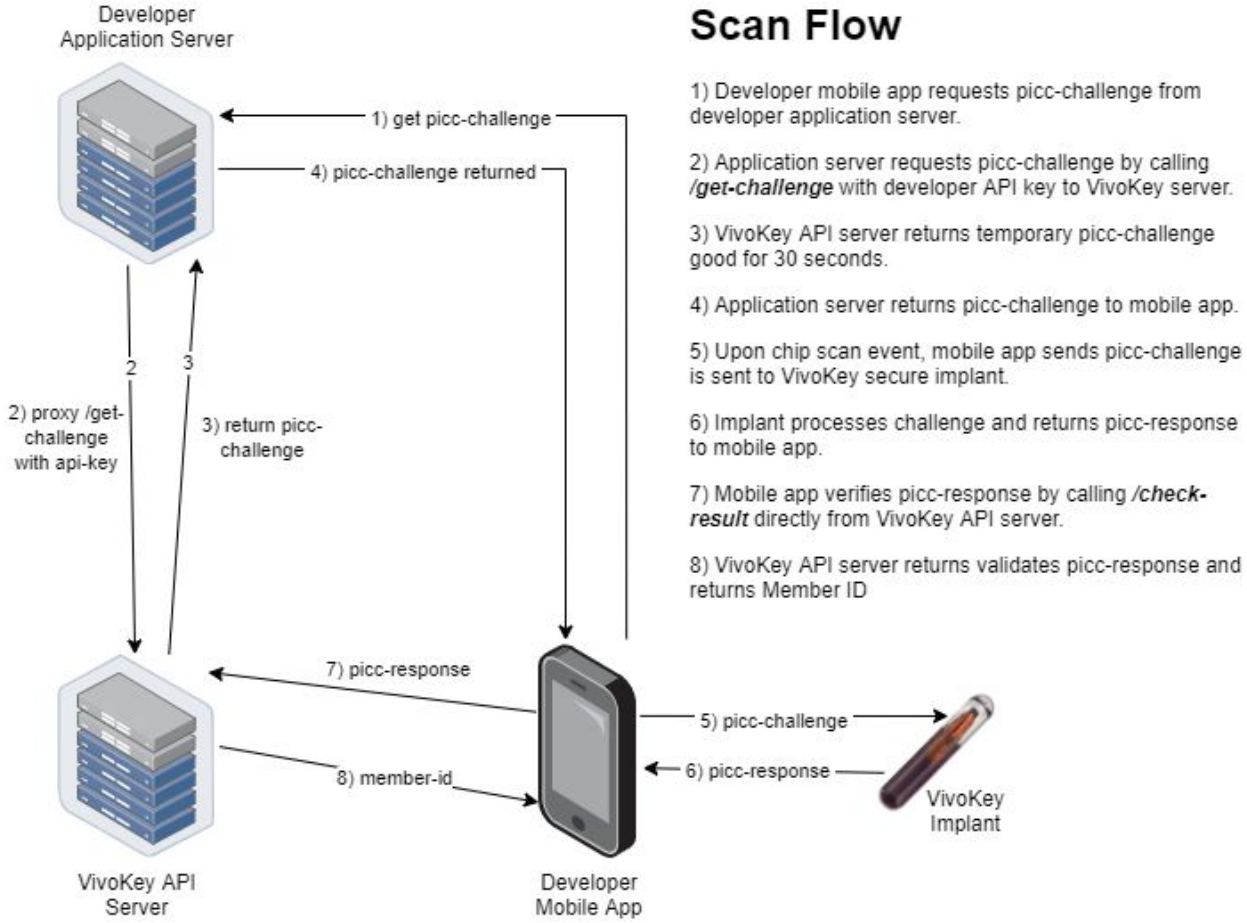
```
{
     "check-result": "chip-member",
     "result-data": [
"1b3e4b680990ba50cb94687868f435a2dedd82af0fc2063a35daeb467c3a276afc57
faacb9b59a3099e2ec0d63aa9d353373f3dc811bceaf313400fad84cf06b",
"75e95abb37ee6e413ce5431f3785023312093c7fb62deb77dbe4c34a3a108227ae28
e61998029c3edbc6393f84e0422315dc4c87e10fe31db4a435790f051579"
     ]
}
```

Mobile App Security

The developer's mobile application must employ its own API security measures to secure the API key or keys, and communication between the mobile app and application servers. You should definitely explore technologies like certificate pinning to trust your server connections, Keystore for Android, and Keychain for iOS to enable trusted storage of your mobile app data. You should also explore Mobile App Attestation techniques like SafetyNet for Android and DeviceCheck for iOS to ensure your servers can trust your app is making API requests.

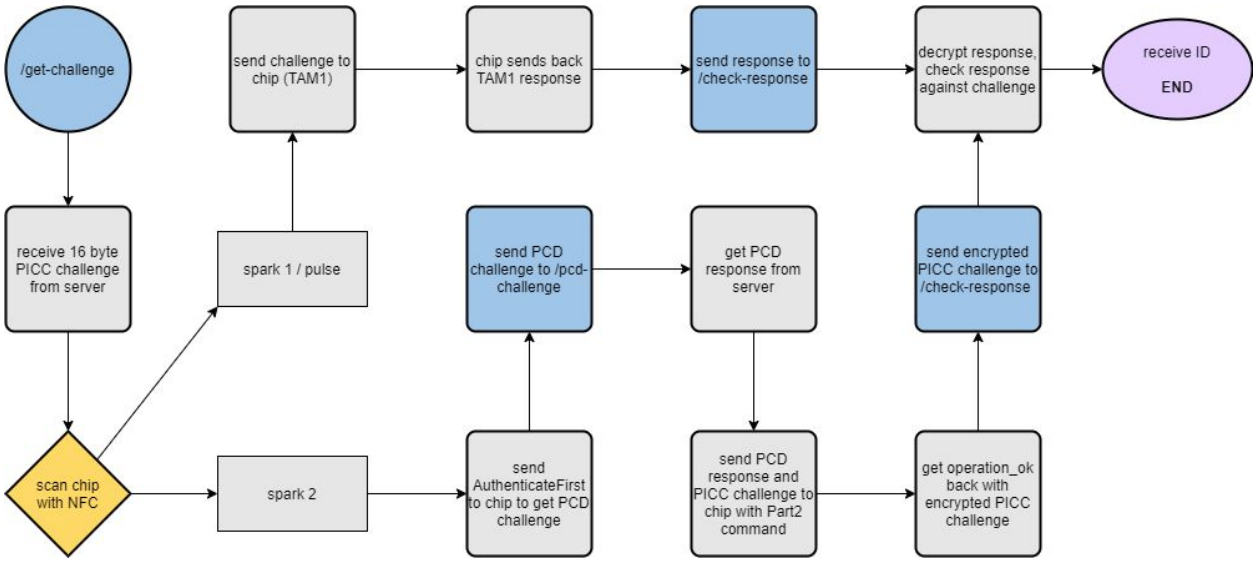
Chip Scan Event Flow

Below is an example of chip scan event flow to obtain Member ID using a Spark 1 implant. The approach uses an application server to proxy the ***get-challenge*** call to shield the api key and keep it from being coded into your application. It is not recommended that your API keys be deployed in source code to your mobile app as they would be vulnerable to decompiling. There are other means you can use to protect your api key, this is just one example.



Chip Scan NFC Flow

The NFC command flow for a chip scan event will depend on the type of PICC being presented to the PCD. The following diagram depicts the flows for various VivoKey products.



NFC Commands

The actual commands used to send and receive data from VivoKey chip implants over NFC will depend on the operating system and NFC framework of the PCD hardware, as well as the VivoKey product being scanned. This section will be expanded in the next version of this document to include code examples for various operating systems. At this time we recommend developers attempt to utilize our easily integrated development libraries for Android and iOS.