**UNITED STATES DISTRICT COURT FOR THE WESTERN DISTRICT OF TEXAS**

| | |
|---|---|
| PROXENSE, LLC,<br><br>*Plaintiffs*,<br><br>v.<br><br>GOOGLE LLC,<br><br>*Defendants*. | Civil Action No. 6:23-cv-00320-ADA |

**PLAINTIFF'S PRELIMINARY INFRINGMENT CONTENTIONS**

Pursuant to this Court's Standing Order Governing Proceedings in Patent Cases, Plaintiff Proxense, LLC ("Proxense" or "Plaintiff"), hereby provides its final infringement contentions concerning Defendant Google LLC, ("Google", or "Defendant").

Proxense hereby certifies that it undertook reasonable efforts to prepare its preliminary infringement contentions and reserves the right to amend them based on material identified after the invalidity contentions are served and during discovery.

These infringement contentions are based on Plaintiff's current knowledge, understanding, and belief as to the facts and information available to it as of the date of these disclosures. Proxense has not yet completed its investigation, collection of information, discovery, or analysis related to this action. The evidence cited in support of the disclosures herein is necessarily exemplary and is therefore illustrative and not exhaustive. For example, Proxense anticipates that relevant facts and evidence are uniquely within the possession, custody, or control of Defendants. As such, Proxense reserves the right to supplement, amend, or modify the information contained herein and to use and introduce at trial such information and any subsequently-identified or discovered information.



MICROSOFT-1013

These final infringement contentions are based at least in part upon Proxense's present understanding of U.S. Patent No. 8,352,730 (the "730 patent"), U.S. Patent No. 8,886,954 (the "954 Patent"), U.S. Patent No. 9,298,905 (the "905 patent"), U.S. Patent No. 8,646,042 (the "042 Patent"), U.S. Patent No. 9,679,289 (the "289 Patent"), U.S. Patent No. 10,073,960 (the "960 Patent") and the accused products in the absence of discovery. The 730, 954, 905, 042, 289, and 960 Patents are collectively referred to herein as the "Proxense Patents" or the "Patents-in-Suit."

Proxense reserves the right to amend its infringement contentions and asserted claims based on information Proxense obtains through discovery and otherwise as this case progresses. Proxense further reserves the right to amend its final infringement contentions and asserted claims based on any proceedings before the United States Patent and Trademark Office regarding the Proxense Patents.

## I. IDENTIFICATION OF INFRINGED CLAIMS

The asserted claims of the Patents-in-Suit are as follows:

| | **Claims** |
|---|---|
| 730 Patent | 1-3, 5, 15-17 (Google Passwordless) and 1-2, 5 (Google Pay) |
| 954 Patent | 1-3, 5-7, 22-27 (Google Passwordless) and 1-2, 5-7 (Google Pay) |
| 905 Patent | 1-2, 15 (Google Passwordless) and 1, 4-5, 7 (Google Pay) |
| 042 Patent | 10 |
| 289 Patent | 14, 16 |
| 960 Patent | 14, 16 |

Proxense identifies these asserted claims based on its current, preliminary understanding and reserves the right to supplement and/or change its identification as discovery proceeds, including identifying additional claims and withdrawing claims.

## II. IDENTIFICATION OF ACCUSED PRODUCTS, SYSTEMS,

**METHODS ANDSERVICES**

Based on the information currently available, Proxense identifies the following accused products, systems, and methods, including all reasonably similar products, systems, methods, and their variants for each of the Patents-in-Suit.

Hereafter, the term "Accused Instrumentalities" or "Accused Products" refers to the products listed below, (1) Google's Universal Passwordless Architecture, which includes (a) Android Operating System ("OS"), (b) Android SafetyNet Authenticator, (c) Chrome browser, and (d) Google Identity Platform (also known as Google Identity Services or Google Cloud Identity Platform), (2) Google Pay services (also known as Google Wallet) and all Google devices preloaded with Google Pay/Wallet.

## III. CHARTS SETTING FORTH WHERE IN THE ACCUSED PRODUCT(S) EACH ELEMENT OF THE ASSERTED CLAIM(S) ARE FOUND

Subject to ongoing discovery and investigation, Proxense provides claim charts pertaining to the Patents-in-Suit attached hereto as Exhibits A1-A2 for the 730 patent, Exhibits B1-B2 for the 905 patent, Exhibits C1-C2 for the 989 patent, Exhibit D for the 042 patent, Exhibit E for the 289 patent, and Exhibit F for the 960 patent. These infringement contentions serve a notice function and are not required to—and therefore do not—present every possible permutation or theory of Proxense's case.

## IV. LITERAL INFRINGEMENT AND INFRINGEMENT UNDER THE DOCTRINEOF EQUIVALENTS

Proxense contends that Google's products literally infringe the asserted claims of the Patents-in-Suit as more specifically explained in the attached claim charts. Proxense reserves the right to assert infringement under the doctrine of equivalents to the extent that the difference between any component of any product or system, and any step of any service, method, and/or

claim element is insubstantial.

## V. PRIORITY DATES AND ALL DOCUMENTS EVIDENCING CONCEPTION ANDREDUCTION TO PRACTICE FOR EACH CLAIMED INVENTION

The 730 patent matured from application number 11/314,199 ("the 199 application"), which was filed on December 20, 2005. The 905 application claims priority from provisional patent application 60/637,538 which was filed on December 20, 2004. Documents evidencing conception and/or reduction to practice for the claimed inventions of the 730 patent are being collected and will be produced shortly.

The 954 patent matured from application number 13/710,109 ("the 109 application"), which was filed on December 10, 2012. The 905 application claims priority from provisional patent application 60/637,538 which was filed on December 20, 2004. Documents evidencing conception and/or reduction to practice for the claimed inventions of the 954 patent are being collected and will be produced shortly.

The 905 patent matured from application number 14/521,982 ("the 982 application"), which was filed on October 23, 2014. The 982 application claims priority from provisional patent application 60/637,538 which was filed on December 20, 2004. Documents evidencing conception and/or reduction to practice for the claimed inventions of the 905 patent are being collected and will be produced shortly.

The 042 patent matured from application number 13/445,825 ("the 825 application"), which was filed on April 12, 2012. The 825 application claims priority from provisional patent application 60/992,953 which was filed on December 6, 2007. Documents evidencing conception and/or reduction to practice for the claimed inventions of the 042 patent are being collected and will be produced shortly.

The 289 patent matured from application number 14/961,645 ("the 645 application"),

which was filed on December 7, 2015. The 645 application claims priority from provisional patent application 60/992,953 which was filed on December 6, 2007. Documents evidencing conception and/or reduction to practice for the claimed inventions of the 289 patent are being collected and will be produced shortly.

The 960 patent matured from application number 15/595,739 ("the 739 application"), which was filed on May 15, 2017. The 739 application claims priority from provisional patent application 60/992,953 which was filed on December 6, 2007. Documents evidencing conception and/or reduction to practice for the claimed inventions of the 960 patent are being collected and will be produced shortly.

## VI. A COPY OF THE FILE HISTORY FOR EACH PATENT IN SUIT

A copy of the file history for each of the Patents-in-Suit is submitted herewith. The file history of the 730 patent bears production numbers PROX_GOOGLE_000001-PROX_GOOGLE_00805. The file history of the 954 patent bears production numbers PROX_GOOGLE_00806- PROX_GOOGLE_01392. The file history of the 905 patent bears production numbers PROX_GOOGLE_001393- PROX_GOOGLE_001936. The file history of the 042 patent bears production numbers PROX_GOOGLE_001937-PROX_GOOGLE_002433. The file history of the 289 patent bears production numbers PROX_GOOGLE_002434- PROX_GOOGLE_002616. The file history of the 960 patent bears production numbers PROX_GOOGLE_002617- PROX_GOOGLE_002818.

Dated: July 24, 2023

Respectfully submitted,

/s/ David L. Hecht

David L. Hecht (**Co-Lead Counsel**)
dhecht@hechtpartners.com

Maxim Price (*pro hac vice*)
mprice@hechtpartners.com
Yi Wen Wu (*pro hac vice*)
wwu@hechtpartners.com
HECHT PARTNERS LLP
125 Park Avenue, 25th Floor
New York, New York 10017
Telephone: (212) 851-6821

Brian D. Melton (**Co-Lead Counsel**)
bmelton@susmangodfrey.com
Geoffrey L. Harrison
gharrison@susmangodfrey.com
Meng Xi
mxi@susmangodfrey.com
Bryce T. Barcelo
bbarcelo@susmangodfrey.com
SUSMAN GODFREY L.L.P.
1000 Louisiana Street, Suite 5100
Houston, Texas 77002-5096
Telephone: (713) 653-7807
Facsimile: (713) 654-6666

Lear Jiang
ljiang@susmangodfrey.com
SUSMAN GODFREY L.L.P.
1900 Avenue of the Stars, Suite 1400
Los Angeles, California 90067-6029
Telephone: (310) 789-3100
Facsimile: (310) 789-3150

*Counsel for Plaintiff Proxense, LLC*

*Attorneys for Plaintiff Proxense, LLC*

**CERTIFICATE OF SERVICE**

I hereby certify that on July 24, 2023, the foregoing was served on counsel for Defendant Google, LLC.

*/s/ David L. Hecht*
*David L. Hecht*

# Exhibit A1

**U.S. Patent Number 8,352,730 – Preliminary Infringement Contentions[1]**

| | |
|---|---|
| **Assignee:** | Proxense, LLC |
| **Title:** | Biometric personal data key (PDK) authentication |
| **Filing Date:** | 2005-12-20 |
| **Publication Date:** | 2013-01-08 |
| **Inventor:** | Giobbi, John J. |

| **'730 Patent Claim[2]** | | **Accused Instrumentality And Where Each Claim Element Is Found[3]** |
|---|---|---|
| **1** | A method for verifying a user during authentication of an integrated device, comprising the steps of[4]: | Google uses, distributes, sells, and/or offers to sell a Universal Passwordless Architecture ("UPA") (e.g. PROX_GOOGLE_003243, https://developers.google.com/identity/passkeys , which includes: (a) Android OS, (b) Android SafetyNet Authenticator, (c) Chrome, and (d) Google Identity Platform ("GIP"), which together infringe, literally or under the doctrine of equivalents, the claimed limitations as presented below. |

[1] The Preliminary Infringement Contentions (PICS) provided herein are based on information obtained to date and may not be exhaustive. Plaintiff's investigation of Defendants' infringement is ongoing. Plaintiff reserves the right to supplement and/or amend these PICS to identify additional instrumentalities and to further identify where each element of each claim is found in each accused instrumentality, including on the basis of discovery obtained from Defendants, and from third parties during the course of this litigation, pursuant to ¶2 of the Order Governing Proceedings – Patent Cases under Hon. Alan D. Albright.

[2] All PICS set forth herein for any independent patent claims are hereby incorporated by reference into the PICS alleged for any dependent patent claims that depend on such independent claims, as if fully set forth therein.

[3] The Accused Instrumentalities and associated exhibits discussed and/or cited for any claim herein are representative in all material respects of all other accused instrumentalities identified for that claim (e.g., a specified phone or service may be used as a representative example in these charts since the other accused instrumentalities have immaterial differences in their hardware and/or software configuration, the cited references are believed to be illustrative of all such accused devices).

[4] Plaintiff's inclusion of any claim preamble in this claim chart should not be interpreted as an admission that the preamble is limiting. Plaintiff reserves the right to take the position that the claim preambles are limiting or not limiting on a claim-by-claim basis.

| '730 Patent Claim[2] | | Accused Instrumentality And Where Each Claim Element Is Found[3] |
|---|---|---|
| | | Google, through its own actions, and the action of its customers, which Google directs and controls, has created and sells[5] the use of UPA, including Android/Chrome preloaded[6] devices, servers, and software supporting and integrating into the UPA architecture. Servers integrating with and supporting the architecture include GIP providing FIDO and/or OAuth services. Devices integrating and supporting the architecture include devices utilizing the Android OS. Software integrating with and supporting the architecture include the Chrome web browser and Android SafetyNet Authenticator. |
| | Persistently storing biometric data of the user and a plurality of codes and other data values comprising a device ID code uniquely identifying the integrated device and a secret decryption value in a tamper proof format written to a storage element on the integrated device that is unable to be subsequently altered; | <u>Persistently storing biometric data of a user … in a tamper proof format written to a storage element on the integrated device that is unable to be subsequently altered</u><br><br>Utilizing the Android operating system developed and disseminated by Google, Android devices store biometric data of the user in a tamper proof format written to a storage element on the integrated device that is unable to be subsequently altered. Starting with Android 10, Google stored biometric data on Android devices includes fingerprint and facial recognition. PROX_GOOGLE_002913, https://source.android.com/docs/security/features/biometric#android10 ("Android 10: Introduces the BiometricManager class that developers can use to query the availability of biometric authentication. Includes fingerprint and face authentication integration for BiometricPrompt").<br><br>As to protecting biometric data, Android's implementation guidelines require tamper-proof "raw fingerprint data or derivatives (for example, templates) must never be accessible from outside the sensor driver or TEE" (trusted execution environment) and "fingerprint acquisition, enrollment, and recognition must occur inside the TEE". Android Open-Source Project: Fingerprint HIDL, PROX_GOOGLE_003177, https://source.android.com/docs/security/features/authentication/fingerprint-hal. Requiring acquisition and recognition to occur inside the TEE, fingerprint data never leaves the TEE. Android's TEE, called Trusty, "uses ARM's Trustzone™ to virtualize the main processor and create a secure trusted exaction environment" isolated from the rest of the system. Android Open-Source Project: Trusty TEE, PROX_GOOGLE_003283, https://source.android.com/docs/security/features/trusty. Accordingly, fingerprint data, which never leaves the TEE, also never leaves the Trustzone housing Trusty. Keeping biometric data |

[5] *See e.g.* PROX_GOOGLE_003250, https://cloud.google.com/identity-platform/pricing.

[6] For the avoidance of doubt, "preloaded" includes devices that ship with Android and other accused software pre-installed or upon which Android and other accused software is otherwise installed by Defendant (e.g., through operating system upgrades, updates, initial installation, or otherwise).

| '730 Patent Claim[2] | | Accused Instrumentality And Where Each Claim Element Is Found[3] |
|---|---|---|
| | | within the Trustzone, Android devices persistently store biometric data in a tamper proof format.[7]<br><br>On Android devices, access to biometric hardware is controlled by Fingerprint HIDL. "Android uses Fingerprint Hardware Interface Definition Language (HIDL) to connect to a vendor-specific library and fingerprint hardware (for example, a fingerprint sensor)." Android Open Source Project: Fingerprint HIDL, PROX_GOOGLE_003177, https://source.android.com/docs/security/features/authentication/fingerprint-hal. The methods enabled by the Fingerprint HIDL do not permit altering biometric data. *Id*. (providing a listing of methods, none of which allows for the alternation of biometric data.). Keeping fingerprint and other biometric data within a portion of the Trustzone only accessible by Fingerprint HIDL, which lacks a method for altering biometric data, Android devices persistently store biometric data of the user in a tamper proof format written to a storage element on the integrated device that is unable to be subsequently altered.<br><br>Android devices require user consent to enroll a fingerprint. *See* PROX_GOOGLE_003204, How to Add a New Fingerprint to Your Android Device - Technipages (Detailing the process of adding a fingerprint to an Android devices by entering a PIN or password for the device before adding a fingerprint.). As enrolling fingerprints on Android devices require entering PIN / Passcode / Password to evidence user consent, Android devices store biometric data of the user in a tamper proof format unable to be subsequently altered.<br><br>As Android devices store biometric data in a tamper proof format that is not capable of being subsequently altered, Google forces users of the Android OS, exercising control and direction over them by controlling every step that the software performs and every prompt for user action, to persistently store their biometric data in a tamper proof format written to a storage element on an integrated device that is not capable of being subsequently altered. |

[7] In the alternative, to the extent the TEE and Trustzone are considered to be a different security implementation or if limited alterations are possible in limited circumstances and given high levels of security permissions, clearance, or hardware-level permission attribution, the "tamper proof format . . . that is unable to be subsequently altered," limitation is met under the doctrine of equivalents on the grounds that the tamper proofing and limitations on alteration utilized in the Trustzone prevent nearly all alteration by nearly all programs that have not been explicitly approved and, at least, prevent alteration by all regular-world (non-TEE and non-Trustzone) applications and requires a strict level of physical and/or virtual layers of permission to be altered.

| '730 Patent Claim[2] | | Accused Instrumentality And Where Each Claim Element Is Found[3] |
|---|---|---|
| | | persistently storing … a plurality of codes and other data values comprising a device ID code uniquely identifying the integrated device … in a tamper proof format written to a storage element on the integrated device that is unable to be subsequently altered<br><br>The Android OS natively includes the Android SafetyNet Authenticator, which is FIDO certified and accessible via the FIDO2 API for Android. *See* PROX_GOOGLE_003793, FIDO® Certified - FIDO Alliance. In operation, authenticators store and access the cryptographic material underlying FIDO Passkeys. *See* PROX_GOOGLE_002819, AccountManager | Android Developers ("Authenticators (which may be written by third parties) handle the actual details of validating account credentials and storing account information. For example, Google, Facebook, and Microsoft Exchange each have their own authenticator."); PROX_GOOGLE_003088, FIDO Technical Glossary (fidoalliance.org) (Defining a FIDO Authenticator as "responsible for user verification, and maintaining the cryptographic material required for the relying party authentication."); and PROX_GOOGLE_003439, Web Authentication: An API for accessing Public Key Credentials - Level 3 (w3.org) (Defining an authenticator as "A cryptographic entity, existing in hardware or software, that can register a user with a given Relying Party and later assert possession of the registered public key credential, and optionally verify the user, when requested by the Relying Party.").<br><br>Having a native authenticator, Android devices support the use of FIDO Passkeys. *See* PROX_GOOGLE_003187, Google Online Security Blog: Security of Passkeys in the Google Password Manager (googleblog.com) ("Passkeys are the result of an industry-wide effort. They combine secure authentication standards created within the FIDO Alliance and the W3C Web Authentication working group with common terminology and user experience across different platforms, recoverability against device loss, and a common integration path for developers. Passkeys are supported in Android and other leading industry client OS platforms."). Accordingly, instead of the traditional username and password combination, users of Android devices may utilize a passkey to access their Google Account. *See* PROX_GOOGLE_003192, Google Online Security Blog: So long passwords, thanks for all the phish (googleblog.com) ("Starting today, you can create and use passkeys on your personal Google Account."). Not only does the Android OS support the use of Passkeys, but Google also helped to create the standards upon which they are based. *See* PROX_GOOGLE_003192, Google Online Security Blog: So long passwords, thanks for all the phish (googleblog.com) ("Passkeys are built on the protocols and standards Google helped create in the FIDO Alliance and W3C WebAuthn working group. |

| '730 Patent Claim[2] | | Accused Instrumentality And Where Each Claim Element Is Found[3] |
|---|---|---|
| | | This means Passkey support works across all platforms and browsers that adopt these standards. You can store the passkeys for your Google Account on any compatible device or service.”). As the Android OS natively includes a FIDO certified authenticator and supports use the passkeys, Android devices must persistently store passkeys on the integrated device.<br><br>The specific variant of passkeys stored by the native Android OS authenticator includes a device ID code uniquely identifying an integrated device. “[P]asskeys on Android support the proposed Device-bound Public Key WebAuthn extension (*devicePubKey)*. If this extension is requested when creating or using passkeys on Android, relying parties will receive two signatures in the result: One from the passkey private key, which may exist on multiple devices, and an additional signature from a second private key that only exists on the current device.” PROX_GOOGLE_003187, Google Online Security Blog: Security of Passkeys in the Google Password Manager (googleblog.com). The second private key is part of a key pair. PROX_GOOGLE_003439, Web Authentication: An API for accessing Public Key Credentials - Level (pr-preview.s3.amazonaws.com) (“This authenticator registration extension and authentication extension provides a Relying Party with a "device continuity" signal for backup eligible credentials. This is done by creating a user credential-specific hardware-bound device key pair on the authenticator, if such a key pair does not already exist for the user credential being created or exercised, and returning the device public key along with a signature by the device private key to the Relying Party.”). The key pair provided by the Device-Bound Public Key extension is unique to each Android device. PROX_GOOGLE_003439, Web Authentication: An API for accessing Public Key Credentials - Level (pr-preview.s3.amazonaws.com) (“The hardware-bound device key pair is not on its own a user credential and does not have its own credential ID. Instead, the returned device public key is a device-specific contextual attribute of its associated user credential.”). As the key-pair is unique, each key of the pair is a unique device ID. However, only the public key is sent to uniquely identify an Android device. PROX_GOOGLE_003187, Google Online Security Blog: Security of Passkeys in the Google Password Manager (googleblog.com) (“This device-bound private key is unique to the passkey in question, and each response includes a copy of the corresponding device-bound public key.”); and PROX_GOOGLE_003439, Web Authentication: An API for accessing Public Key Credentials - Level (pr-preview.s3.amazonaws.com) (“That is, when that user credential is used—along with the devicePubKey extension—on a particular authenticator, a particular device public key is returned by the extension, along with a signature demonstrating proof-of-possession of the device private key by that device.”). Because device bound-public keys are not synced, "observing two passkey signatures with the same device-bound public key |

| '730 Patent Claim[2] | | Accused Instrumentality And Where Each Claim Element Is Found[3] |
|---|---|---|
| | | is a strong signal that the signatures are generated by the same device. On the other hand, if a relying party observes a device-bound public key it has not seen before, this may indicate that the passkeys has been synced to a new device." Google Online Security Blog: Security of Passkeys in the Google Password Manager, October 12, 2022, PROX_GOOGLE_003187, https://security.googleblog.com/2022/10/SecurityofPasskeysintheGooglePasswordManager.html. Providing an indication of the integrated device generating a signature, the public key of the Device-Bound Public Key extensions key-pair is a device ID code uniquely identifying the integrated device.<br><br>When a Device Bound Public Key is present, the integrated authenticator within the Android OS causes the device to store a private key bound to the device. PROX_GOOGLE_003439 at § 10.2.2.2 (Defining making a credential with a device bound public key as including "5. Let dpk be the newly created or existing device public key, in COSE_Key format in the same fashion as for the user credential's credentialPublicKey when the latter is conveyed in attested credential data. 6. Let devicePrivateKey be the newly created or existing device private key.")) The public key corresponding to the private key is returned to the relying party and saved. PROX_GOOGLE_003439 at § 10.2.2.3.1 (Stating registration at the relying party includes step "6. Create a new device-bound key record … add this device-bound key record to the devicePubKeys member of the new credential record.").) When processing an authentication request received from Google Identity, the native Android OS SafetyNet authenticator creates a signature with the private key associated with the device-bound public key and then sends the signature and the device-bound public key back to Google Identity for verification. *Id.* at § 10.2.2.2 (Defining authentication as including "9. Let dpkSig be the result of signing the assertion signature input with devicePrivateKey. 10. Output dpkSig as the extension's unsigned extension output.").) Upon receipt of the authentication response, Google Identity extracts the device bound public key from the response and uses it to verify the signature. *Id.* at § 10.2.2.3.2 (Defining Relying Party verification of the authentication assertion as including "3. Extract the contained fields from attObjForDevicePublicKey: aaguid, dpk, scope, nonce, fmt, attStmt. 4. Verify that signature is a valid signature over the assertion signature input (*i.e.*, authData and hash) by the device public key dpk.")) If the device bound public key were changed at any time, Google Identity would not be able to verify the signature as it would no longer correspond to the authenticator's device bound private key. Accordingly, by being part of a key pair, the device bound public key is in a format unable to be subsequently altered. |

| '730 Patent Claim[2] | | Accused Instrumentality And Where Each Claim Element Is Found[3] |
|---|---|---|
| | | Of course, if both the private key and public key were altered, then the signature could be nominally verified. In such a situation, however, the altered device-bound public key would not match any of the device-bound public keys stored by the Relying Party (i.e., Microsoft and Google). *See Id.* at § 10.2.2.3.2 (Steps 5 and 6 detailing, during an authentication ceremony, comparing the received device-bound public key to a list of stored device-bound public keys, where the lack of a match indicates a new device.). The Relying Party, accordingly, would treat the altered device-bound public key as representing a new device and not allow the request absent further user verification. Accordingly, by being recorded by the relying party, device-bound public keys are further placed in a format unable to be subsequently altered.[8]<br><br>As Google controls the protocol utilized by its Identity platform, including the ability to sync passkeys across devices, users wanting the benefit of biometric authentication via passkeys have no choice but to store on their device a pair of keys generated by the device bound public key extension that uniquely identifies the integrated device in a tamper proof format written to a storage element on the integrated device that is unable to be subsequently altered.<br><br><u>persistently storing … a secret decryption value in a tamper proof format written to a storage element on the integrated device that is unable to be subsequently altered;</u><br><br>Passkeys in the Google Password Manager are always end-to-end encrypted. See PROX_GOOGLE_003187. When a passkey is backed up, its private key is uploaded only in its encrypted form using an encryption key that is only accessible on the user's own devices. *Id*. Without access to the private key, such an attacker cannot use the passkey to sign into its corresponding online account. *Id*. Accordingly, Android devices must contain a secret decryption value (i.e., the encryption key only accessible on the user's own device) to decrypt and use synced and backed-up passkeys. The decryption values are inherently in a tamper proof format as any alteration would prevent correct decryption of the encrypted passkey and thus render the passkey unusable. As such, Android devices contain a secret decryption value in a tamper proof format unable to be subsequently altered.<br><br>As such encryption is dictated by Google and implemented natively in the Android OS, users wanting the benefit of biometric authentication via passkeys have no choice but to store on their |

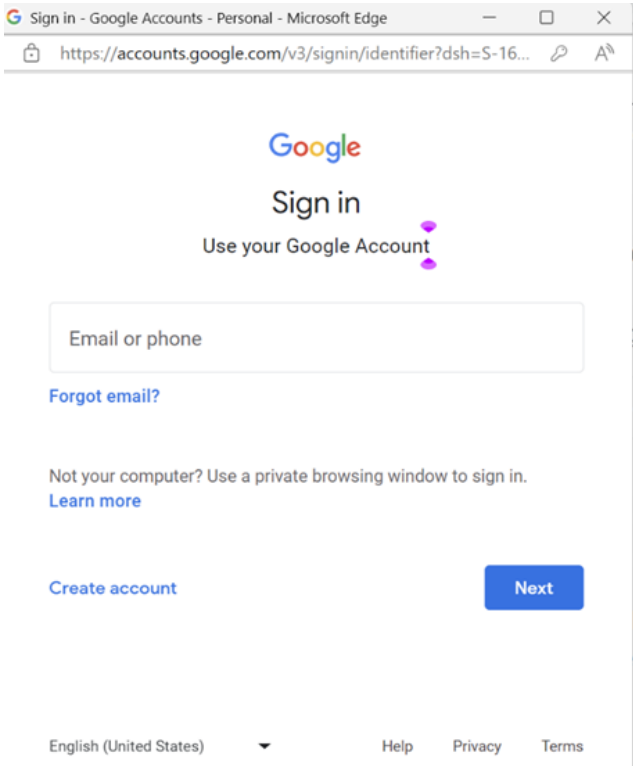
[8] In the alternative, the device-bound key pair performs the same function in the same way and achieves the same result as storage in a format that is unable to be subsequently altered in that alterations are rendered impossible because alterations break the key pair connection and thus would infringe under the doctrine of equivalents as well.

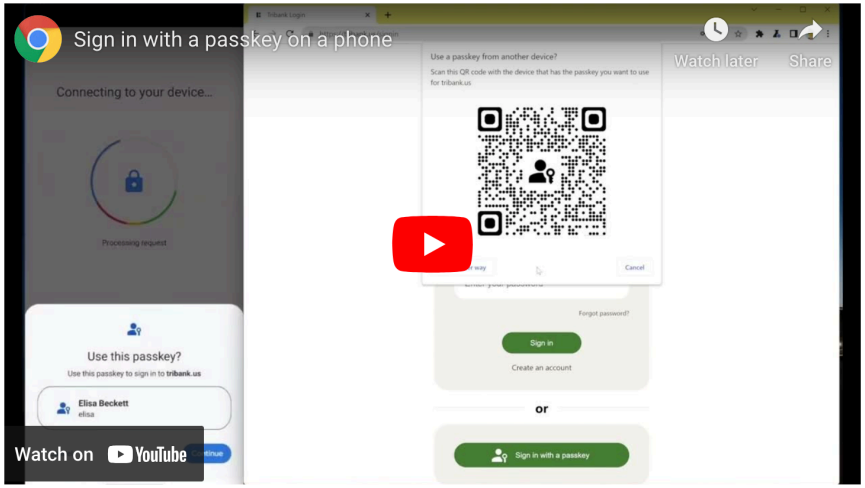
| '730 Patent Claim[2] | | Accused Instrumentality And Where Each Claim Element Is Found[3] |
|---|---|---|
| | | device a secret decryption value in a tamper proof format written to a storage element on the integrated device that is unable to be subsequently altered. |
| | wherein the biometric data is selected from a group consisting of a palm print, a retinal scan, an iris scan, a hand geometry, a facial recognition, a signature recognition and a voice recognition; | Android's development documentation describes that biometric data may include, for example, face recognition or a fingerprint (a subset of a palm print): "One method of protecting sensitive information or premium content within your app is to request biometric authentication, such as using face recognition or fingerprint recognition. This guide explains how to support biometric login flows in your app." PROX_GOOGLE_003271, https://developer.android.com/training/sign-in/biometric-auth |
| | responsive to receiving a request for a biometric verification of the user, receiving scan data from a biometric scan; | Google describes the use of Passkeys as a biometric alternative to remembering passwords. PROX_GOOGLE_003243, Passwordless login with passkeys \| Authentication \| Google for Developers ("Passkeys are a safer and easier alternative to passwords. With passkeys, users can sign in to apps and websites with a biometric sensor (such as a fingerprint or facial recognition), PIN, or pattern, freeing them from having to remember and manage passwords.") Signing in with a biometric sensor necessarily requires prompting a user for a response to an authentication request wherein Google receives scan data from a biometric scan. Google utilizes the biometric sensors in smartphones, tablets, and computers to receive data from those biometric scans. |
| | comparing the scan data to the biometric data to determine whether the scan data matches the biometric data; | Google describes the use of Passkeys as a biometric alternative to remembering passwords. PROX_GOOGLE_003243, Passwordless login with passkeys \| Authentication \| Google for Developers ("Passkeys are a safer and easier alternative to passwords. With passkeys, users can sign in to apps and websites with a biometric sensor (such as a fingerprint or facial recognition), PIN, or pattern, freeing them from having to remember and manage passwords.") Signing in with a biometric sensor necessarily requires comparing the scan data from a biometric scan with data that had been stored previously as the template or baseline of the authenticated user. |
| | responsive to a determination that the scan data matches the biometric data, wirelessly sending one or more codes from the plurality of codes and the other data values for authentication by an agent that is a third-party trusted authority | <u>responsive to a determination that the scan data matches the biometric data, wirelessly sending</u><br><br>As detailed by Google, using passkeys to sign into a service or application is as simple as unlocking their Android device with fingerprint or facial recognition. PROX_GOOGLE_003192, Google Online Security Blog: So long passwords, thanks for all the phish (googleblog.com) ("Passkeys are a more convenient and safer alternative to passwords. They work on all major platforms and browsers, and allow users to sign in by unlocking their |

| '730 Patent Claim[2] | | Accused Instrumentality And Where Each Claim Element Is Found[3] |
|---|---|---|
| | possessing a list of device ID codes uniquely identifying legitimate integrated devices, wherein the one or more codes and other data values includes the device ID code; and | computer or mobile device with their fingerprint, face recognition or a local PIN.”); and PROX_GOOGLE_003243, Passwordless login with passkeys \| Authentication \| Google for Developers (“When a user wants to sign in to a service that uses passkeys, their browser or operating system will help them select and use the right passkey. The experience is similar to how saved passwords work today. To make sure only the rightful owner can use a passkey, the system will ask them to unlock their device. This may be performed with a biometric sensor (such as a fingerprint or facial recognition), PIN, or pattern.”). This ease of access is provided by Google’s FIDO certified SafetyNet Authenticator utilizing the FIDO2 API for Android. *See* **PROX_GOOGLE_003775** (“The FIDO2 API allows Android applications to create and use strong, attested public key- based credentials for the purpose of authenticating users. The API provides a WebAuthn Client implementation, which supports the use of BLE, NFC, and USB roaming authenticators (security keys) as well as a platform authenticator, which allows the user to authenticate using their fingerprint or screen lock.”) In operation, the SafetyNet Authenticator utilizes the native capacities of the Android OS, such as the Android Keystore to store the cryptographic public-private key pairs underlying the device-bound public key extension. *See* **PROX_GOOGLE_003795** ((“The typical way to develop secure authenticator on Android mobile devices (smartphones and tablets) is to use a secured hardware-backed operating environment... This includes technology such as TEE (“Trusted Execution Environment”) that performs cryptographic and other sensitive operations including digital signing and biometric data processing used to support FIDO functionality... Android Keystore allows app developers to store cryptographic keys in a container and use them in cryptographic operations via APIs. Android also offers protection of the fingerprint sensor data via a TEE, which allows for encryption and cryptographic authentication... Adding a small application to complement some functions that are necessary to perform as UAF 1.1 authenticator enables Android Keystore with hardware-backed key attestations and fingerprint sensor to become a secure FIDO UAF 1.1 authenticator."). The Android Keystore securely holds a passkey until it is unlocked by biometric verification. See PROX_GOOGLE_003785; *see also* PROX_GOOGLE_003821 (“When generating or importing a key into the Android KeyStore, you can specify that the key is only authorized to be used if the user has been authenticated. The user is authenticated using a subset of their secure lock screen credentials (pattern/PIN/password, biometric credentials).”). Biometric unlocking within the Android OS is provided by Fingerprint HIDL. *See* PROX_GOOGLE_003177, Fingerprint HIDL \| Android Open Source Project (“On devices with a fingerprint sensor, users can enroll one or more fingerprints and use those fingerprints to unlock the device and perform other tasks. Android uses the Fingerprint Hardware Interface Definition Language (HIDL) to connect to a vendor-specific library and fingerprint hardware |

| '730 Patent Claim[2] | | Accused Instrumentality And Where Each Claim Element Is Found[3] |
|---|---|---|
| | | (for example, a fingerprint sensor)"). Accordingly, by utilizing Fingerprint HIDL in combination with the KeyStore, the Android OS releases a passkey after a determination that the scan data matches the biometric data.<br><br>As detailed supra, Google utilizes a device-bound public key extension, which requires concurrently sending a public-key and a signature generated with a device-bound private key to be verified with the public-key. PROX_GOOGLE_003439, Web Authentication: An API for accessing Public Key Credentials - Level (pr-preview.s3.amazonaws.com) ("That is, when that user credential is used—along with the devicePubKey extension—on a particular authenticator, a particular device public key is returned by the extension, along with a signature demonstrating proof-of-possession of the device private key by that device."). As Android devices possess cellular and/or Wi-Fi capability, the signature and device-bound public key uniquely identifying the integrated device sent to Google Identity Platform will be sent wirelessly – either via Wi-Fi or a cellular network to the internet.<br><br>While the native Android OS authenticator permits the use of a passkey in a variety of situations, each transmits the device-bound public key and corresponding signature over the internet. When a user wishes to use a passkey stored on their Android device to access a resource via a browser the user is presented with either a QR code or push notification. *See* PROX_GOOGLE_003236, Passkey support on Android and Chrome \| Authentication \| Google for Developers ("Depending on the desktop operating system (e.g. ChromeOS, iOS, MacOS, Windows) users may be presented with a QR code to securely use a passkey stored on their mobile device or a notification may be displayed prompting the user to unlock their phone to use the relevant passkey."). In the case of a QR code, the user scans a QR code displayed on the screen of a device. PROX_GOOGLE_003192, Google Online Security Blog: So long passwords, thanks for all the phish (googleblog.com) ("When you do need to use a passkey from your phone to sign in on another device, the first step is usually to scan a QR code displayed by that device."). The proximity between the devices is then verified using Bluetooth. PROX_GOOGLE_003239, Passkeys use cases \| Authentication \| Google for Developers ("The two devices verify that they are in proximity with each other using Bluetooth."). The Bluetooth connection is also utilized to establish a secure connection between the devices via the internet. PROX_GOOGLE_003192, Google Online Security Blog: So long passwords, thanks for all the phish (googleblog.com) ("The device then verifies that your phone is in proximity using a small anonymous Bluetooth message and sets up an end-to-end encrypted connection to the phone through the internet..") Utilizing the internet connection, the Android device sends a one-time |

| '730 Patent Claim[2] | | Accused Instrumentality And Where Each Claim Element Is Found[3] |
|---|---|---|
| | | passkey signature, including the device-bound public key, to the other device, which is used to sign in the user. PROX_GOOGLE_003239, Passkeys use cases | Authentication | Google for Developers ("A one-time passkey signature is transferred back to Safari on the Mac, which the website then uses to sign the user in."). Establishing the connection over the internet would require utilizing either the device's Wi-Fi or cellular capabilities – both of which are wireless protocols. Accordingly, responsive to a determination that the scan data matches the biometric data, Android devices wirelessly send a passkey signature and accompanying device-bound public key uniquely identifying the device.<br><br>Push notifications can originate with the "Sign in with Google" button. To receive the benefit of quickly and easily managing user authentication utilizing "Sign-in with Google", the website must display a "Sign in with Google" button, which is generated by the Google Identity Services JavaScript library, or provide the user the option to sign in with a passkey. PROX_GOOGLE_003779 ("Put it another way, the Sign in with Google button must be generated by the Google Identity Services JavaScript library now. The button rendering API allows you to customize the color, shape, text, and size to meet the branding requirements of your website, whereas still stick to Google's guidelines."); and PROX_GOOGLE_003239. A functional demonstration of the "Sign in with Google" button is provided in Google Identity documentation, as shown below.<br><br>Sign in with Google demo<br><br>Click the button to sign-in to your Google Account.<br><br>Sign in with Google<br><br>*See* PROX_GOOGLE_003232.<br><br>When this button is clicked, the dialog box pictured below is presented to the user. |

| '730 Patent Claim[2] | Accused Instrumentality And Where Each Claim Element Is Found[3] |
|---|---|
| |  <br>The user enters an email and then clicks "next". If the user has activated "Use your phone to sign in" in their Google Account, a dialog box is presented directing the user to complete the sign on with their phone, which has received a push notification. The push notification includes a challenge from Google Identity servers for the FIDO client on the user's Android phone. The FIDO client will invoke the Android SafetyNet Authenticator to complete the challenge by prompting the user to biometrically unlock the stored passkey, use the passkey to sign a challenge, and return the signature and accompanying device-bound public key to Google Identity, as detailed above. Returning the signed challenge in response to the push notification requires establishing the connection over the internet utilizing either the device's Wi-Fi or cellular capabilities – both of which are wireless protocols. Accordingly, responsive to a determination that the scan data matches the biometric data, Android devices wirelessly send a passkey signature and accompanying device-bound public key uniquely identifying the device. *See also:* |

| '730 Patent Claim[2] | Accused Instrumentality And Where Each Claim Element Is Found[3] |
|---|---|
| |  <br>For example, a user visits `example.com` on the Chrome browser on their Windows machine. This user has previously logged into `example.com` on their Android device and generated a passkey. On the Windows machine, the user chooses to sign in with a passkey from another device. The two devices will connect and the user will be prompted to approve the use of their passkey on the Android device, for example, with a fingerprint sensor. After doing so, they're signed in on the Windows machine. Note that the passkey itself isn't transferred to the Windows machine, so typically `example.com` will offer to create a new passkey there. That way, the phone isn't required next time the user wants to sign in. Read Sign-in with a phone to learn more.<br><br>PROX_GOOGLE_003243, Passwordless login with passkeys \| Authentication \| Google for Developers https://developers.google.com/identity/passkeys; Sign in with a passkey on a phone PROX_GOOGLE_003833, (https://www.youtube.com/watch?v=ywQ8bFla-L8 ).<br><br>responsive to a determination that the scan data match the biometric data, … sending one or more codes and other values from the plurality of codes and other data value for authentication … wherein the one or more codes includes the device ID code<br><br>As detailed supra, Google utilizes the device-bound public key extension, which requires concurrently sending a public-key and a signature generated with a device-bound private to be verified with the public-key. PROX_GOOGLE_003439, Web Authentication: An API for accessing Public Key Credentials - Level (pr-preview.s3.amazonaws.com) ("That is when that user credential is used—along with the devicePubKey extension—on a particular authenticator, a particular device public key is returned by the extension, along with a signature demonstrating |

| '730 Patent Claim[2] | | Accused Instrumentality And Where Each Claim Element Is Found[3] |
|---|---|---|
| | | proof-of-possession of the device private key by that device."). The extension allows Google to determine if a sign-in attempt is coming from a new device by observing whether it has previously seen the accompanying device-bound public key. PROX_GOOGLE_003187, Google Online Security Blog: Security of Passkeys in the Google Password Manager (googleblog.com) ("Observing two passkey signatures with the same device-bound public key is a strong signal that the signatures are generated by the same device. On the other hand, if a relying party observes a device-bound public key it has not seen before, this may indicate that the passkey has been synced to a new device.") Making such observations requires that the device-bound public key operates as a device ID.<br><br>As Google controls the protocol utilized by its Identity Platform, including the use of device-bound public key extension, users wanting to sign in to their Google Account without a password via biometric authentication, have no choice but to send a device ID code for authentication upon completion of the biometric challenge.<br><br><u>responsive to a determination that the scan data matches the biometric data, wirelessly sending … for authentication by an agent that is a third-party trusted authority</u><br><br>As detailed *supra*, upon successful completion of the biometric challenge, a device ID in the form of a device-bound public key and a signature is sent from the Android device to the Google Identity Platform for authentication.<br><br>Google Identity utilizes the federation protocol OpenID Connect. PROX_GOOGLE_003181, Google Identity \| Google Developers ("Google's OAuth 2.0 APIs conform to the OpenID Connect specification, are OpenID Certified, and can be used for both authentication and authorization."); and PROX_GOOGLE_003213, OpenID Connect \| Authentication \| Google Developers ("Google's OAuth 2.0 authentication system supports the required features of the OpenID Connect Core specification."). The OpenID Connect protocol includes an OpenID provider (OP) which is a server providing authentication claims to an application or website. *See* PROX_GOOGLE_003103, Final: OpenID Connect Core 1.0 incorporating errata set 1, § 1.2 (Defining OpenID Provider (OP) as "OAuth 2.0 Authorization Server that is capable of Authenticating the End-User and providing Claims to a Relying Party about the Authentication event and the End-User"; and a Relying Party (RP) as "OAuth 2.0 Client application requiring End-User Authentication and Claims from an OpenID Provider."). Providing authentication claims an application relies upon, the OP is a trusted authority. Google Identity Services |

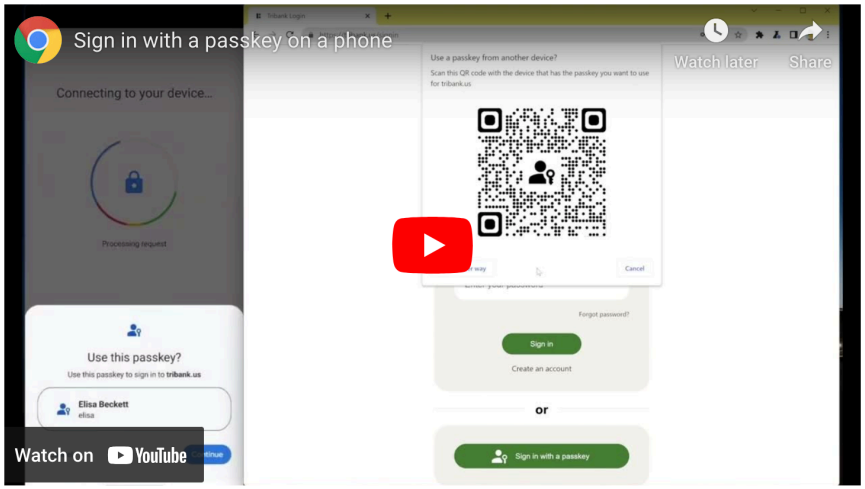
| '730 Patent Claim[2] | | Accused Instrumentality And Where Each Claim Element Is Found[3] |
|---|---|---|
| | | operates as the OpenID Provider by enabling apps and websites to authenticate users using a Google Account. PROX_GOOGLE_003232, Overview \| Authentication \| Google Developers ("Sign in with Google helps you to quickly and easily manage user authentication on your website. Users sign into a Google account, provide their consent, and securely share their profile information with your platform."). As Google Identity is neither the application nor the website being accessed, it is a third party to the transaction. Google Identity thus operates as a third-party trusted authority for authentication.<br><br>possessing a list of device ID codes uniquely identifying legitimate integrated devices<br><br>Google maintains a list of legitimate devices for each user. The FIDO standard utilized by Google Identity incorporates the WebAuthn standard. *See* PROX_GOOGLE_003789 ("Today, WebAuthn is part of the FIDO Alliance's FIDO2 specifications and the FIDO Alliance runs certificate programs to ensure compliance."). One of the use cases for WebAuthn is decommissioning lost, stolen, and disregarded devices such that "assertions signed by this credential are rejected." *See* Web Authentication: An API for accessing Public Key Credentials - Level 3, Sec. 1.3.5 Decommissioning, PROX_GOOGLE_003439. A user may decommission devices they no longer use by signing out of them via their Google Account. *See* PROX_GOOGLE_003791 (Instructing users how to secure lost, disregarded, or stolen devices by signing out of them.). To prevent account access from a device, and mark it as no longer legitimate, a user must sign out of the device and all sessions associated with the device. *Id* ("If you want to make sure there's no account access from a device, sign out of all the sessions with this device name."); and PROX_GOOGLE_003787 ("To not get Google Prompts on a device, sign out of your Google Account on that device."). Accordingly, by maintaining a list of devices the user has selected to remain signed in on, Google Identity maintains a list of legitimate integrated devices.<br><br>Additionally, as noted above, Google Identity utilizes the device-bound public key extension. If this extension is requested when creating or using passkeys on Android, relying parties will receive two signatures in the result: One from the passkey private key, which may exist on multiple devices, and an additional signature from a second private key that only exists on the current device." PROX_GOOGLE_003187, Google Online Security Blog: Security of Passkeys in the Google Password Manager (googleblog.com). The second private key is part of a key-pair. PROX_GOOGLE_003439, Web Authentication: An API for accessing Public Key Credentials - Level (pr-preview.s3.amazonaws.com) ("This authenticator registration extension |

| '730 Patent Claim[2] | | Accused Instrumentality And Where Each Claim Element Is Found[3] |
|---|---|---|
| | | and authentication extension provides a Relying Party with a "device continuity" signal for backup eligible credentials. This is done by creating a user credential-specific hardware-bound device key pair on the authenticator, if such a key pair does not already exist for the user credential being created or exercised, and returning the device public key along with a signature by the device private key to the Relying Party."). The key pair provided by the Device-Bound Public Key extension is unique to each Android device. PROX_GOOGLE_003439, Web Authentication: An API for accessing Public Key Credentials - Level (pr-preview.s3.amazonaws.com) ("The hardware-bound device key pair is not on its own a user credential and does not have its own credential ID. Instead, the returned device public key is a device-specific contextual attribute of its associated user credential."). As the key-pair is unique, each key of the pair is a unique device ID. However, only the public key is sent to uniquely identify an Android device. PROX_GOOGLE_003187, Google Online Security Blog: Security of Passkeys in the Google Password Manager (googleblog.com) ("This device-bound private key is unique to the passkey in question, and each response includes a copy of the corresponding device-bound public key."); and PROX_GOOGLE_003439, Web Authentication: An API for accessing Public Key Credentials - Level (pr-preview.s3.amazonaws.com) ("That is, when that user credential is used—along with the devicePubKey extension—on a particular authenticator, a particular device public key is returned by the extension, along with a signature demonstrating proof-of-possession of the device private key by that device."). Because device bound-public keys are not synced, "observing two passkey signatures with the same device-bound public key is a strong signal that the signatures are generated by the same device. On the other hand, if a relying party observes a device-bound public key it has not seen before, this may indicate that the passkeys have been synced to a new device." Google Online Security Blog: Security of Passkeys in the Google Password Manager, October 12, 2022, PROX_GOOGLE_003187 https://security.googleblog.com/2022/10/SecurityofPasskeysintheGooglePasswordManager.html. To indicate that the integrated device generating a signature has not been seen before, Google Identity must maintain a list of the device-bound public keys code uniquely identifying the integrated device. |
| | responsive to authentication of the one or more codes and the other data values by the agent, receiving an access message from the agent allowing the user access to an | <u>responsive to authentication of one or more codes and the other data values by the agent, receiving an access message from the agent allowing the user access to an application</u><br><br>Authentication is provided by a FIDO server operated by Google in conjunction with OpenID Connect Authentication Server. While OpenID Connect is a federation protocol, it is compatible |

| '730 Patent Claim[2] | | Accused Instrumentality And Where Each Claim Element Is Found[3] |
|---|---|---|
| | application, wherein the application is selected from a group consisting of a casino machine, a keyless lock, a garage door opener, an ATM machine, a hard drive, computer software, a web site and a file. | and complementary with FIDO. "The value of a FIDO authentication capability is amplified by a federated system, where the federation system extends the benefits of a FIDO authentication to applications and services without requiring FIDO to be directly integrated with those applications." PROX_GOOGLE_003072, Enterprise_Adoption_Best_Practices_Federation_FIDO_Alliance.pdf (fidoalliance.org), page 4. When a federated system, such as OpenID Connect, is combined with FIDO, the OpenID Provider (OP) sends a FIDO server challenge to an authenticator which is returned as the FIDO Authentication response. PROX_GOOGLE_003072, Enterprise_Adoption_Best_Practices_Federation_FIDO_Alliance.pdf (fidoalliance.org), page 10 (Detailing the combined use FIDO and federation entails "3… The Authentication Authority determines that the FIDO authentication specified in the previous request is both relevant… 4. The Authentication Authority sends a FIDO Server challenge… 6. FIDO Client returns FIDO authentication response… 7. FIDO Server validates the FIDO response and reports same to Authentication Authority which looks up the user and then redirects the User Agent back to the Application Provider with an authentication assertion.").<br><br>FIDO incorporates the WebAuthn Protocol. PROX_GOOGLE_002926, Client to Authenticator Protocol (CTAP) (fidoalliance.org), § 1.1 ("This specification is part of the FIDO2 project, which includes this specification and is related to the W3C [WebAuthn] specification."); and PROX_GOOGLE_003287, Web Authentication: An API for accessing Public Key Credentials - Level 2 (w3.org), § 1.1 ("Along with the Web Authentication API itself, this specification defines a request-response cryptographic protocol—the WebAuthn/FIDO2 protocol—between a WebAuthn Relying Party server and an authenticator, where the Relying Party's request consists of a challenge and other input data supplied by the Relying Party and sent to the authenticator.") Accordingly, the FIDO server challenge sent will a be WebAuthn authenticatorGetAssertion request. The response to the request received will contain a signature generated by the authenticator's private key and a credential ID. PROX_GOOGLE_002926, Client to Authenticator Protocol (CTAP) (fidoalliance.org), § 6.2.2 (Detailing the response to authenticationGetAssertion as containing "PublicKeyCredentialDescriptor structure containing the credential identifier whose private key was used to generate the assertion" and "an assertion signature".); and PROX_GOOGLE_003287, Web Authentication: An API for accessing Public Key Credentials - Level 2 (w3.org), § 6 (Defining an "assertion signature" as "an assertion signature asserts that the authenticator possessing a particular credential private key has established, to the best of its ability, that the user requesting this transaction is the same user who consented to creating that particular public key credential."). Upon receiving the response, the |

| '730 Patent Claim[2] | | Accused Instrumentality And Where Each Claim Element Is Found[3] |
|---|---|---|
| | | WebAuthn/FIDO server will use the credential ID to locate the appropriate public key to verify the signature generated with the private key. PROX_GOOGLE_003287, Web Authentication: An API for accessing Public Key Credentials - Level 2 (w3.org), § 7.2 ("7. Using credential.id (or credential.rawId, if base64url encoding is inappropriate for your use case), look up the corresponding credential public key and let credentialPublicKey be that credential public key… 20. Using credentialPublicKey, verify that sig is a valid signature over the binary concatenation of authData and hash.") Accordingly, a FIDO/WebAuthn server operated by Google Identity will authenticate a device ID (i.e., credential ID). After validating the device ID, the FIDO/WebAuthn portion of the Identity Platform "redirects the user agent back to the Application Provider with an authentication assertion". PROX_GOOGLE_003072, Enterprise_Adoption_Best_Practices_Federation_FIDO_Alliance.pdf (fidoalliance.org), page 10.<br><br>After authentication of the passkey by Google Identity, the user may be allowed access to an app or website. PROX_GOOGLE_002906, Android Developers Blog: Bringing passkeys to Android & Chrome (googleblog.com) ("Developers can build passkey support on their sites for end-users using Chrome via the WebAuthn API, on Android and other support platforms… Our next milestone in 2022 will be an API for native Android Apps."). Google Identity utilizes an OpenID Certified implementation of OAuth 2.0. PROX_GOOGLE_003213, OpenID Connect \| Authentication \| Google for Developers ("Google's OAuth 2.0 APIs can be used for both authentication and authorization. This document describes our OAuth 2.0 implementation for authentication, which conforms to the OpenID Connect (PROX_GOOGLE_003199, https://openid.net/connect/) specification, and is OpenID Certified (PROX_GOOGLE_002921, https://openid.net/certification/).) An authentication request sent to Google Identity elicits a response including an authorization code that can be sent to Google Identity to receive an ID Token and Access Token. PROX_GOOGLE_003213, OpenID Connect \| Authentication \| Google for Developers ("The response includes a code parameter, a one-time authorization code that your server can exchange for an access token and ID token."). The ID Token contains user information sufficient to allow access to the app or website should the user be registered. *Id.,* ("After obtaining user information from the ID token, you should query your app's database. If the user already exists in your database, you should start an application session for that the user if all login requirements are met by the Google API response."). The Access Token provides permission to use other Google APIs on behalf of the user, such as Google Drive or YouTube. *Id.,* ("One of the advantages of using OAuth 2.0 for authentication is that your application can get permission on behalf of the user (such as |

| '730 Patent Claim[2] | | Accused Instrumentality And Where Each Claim Element Is Found[3] |
|---|---|---|
| | | YouTube, Google Drive, Calendar, or Contracts) at the same time as you authenticate the user… The access token that you receive back from Google allows you to access all the APIs related to the scopes of access you requested and were granted."). As such, the authorization code is exchangeable for ID and access tokens providing access to various applications. Being received from Google Identity acting as third-party trust authority and being exchangeable for tokens providing access to various applications, the authorization code is an access message received from the agent allowing the user access to an application.<br><br><u>wherein the application is selected from a group consisting of a casino machine, a keyless lock, a garage door opener, an ATM machine, a hard drive, computer software, a web site and a file.</u><br><br>If Google Identity Services verifies the signed challenge and that the public key corresponds to a legitimate device, an access message is returned in the form of an authorization code, which is used to log into a website or application by being exchanged for the appropriate token. If resources in addition to the website or application are to be accessed, the Authorization Code may be exchanged for an ID Token or an Access Token. To make the exchange, Google requires the client agent (i.e., website loaded in the computer's browser) to send a POST request containing the authorization code received from Google, as well as a client ID and client secret obtained when the developer registered the User Agent with Google Identity Services. *See* PROX_GOOGLE_003213. The information contained in the ID Token is then used to query the developer's database to determine if the user exists, and if so, to start a session for the user given access to the website or application. Id. ("After obtaining user information from the ID Token, you should query your app's database. If the user already exists in your database, you should start an application session for that user if all login requirements are met by the Google API response."). The Authorization Code received from Google Identity Services is a message permitting access to the application or website by being exchangeable for an ID Token used to allow access.<br><br>If, however, the User Agent needs to access a resource such as a file in Google Drive or another resource, the authentication code may be exchanged for an Access Token. *Id*. ("One of the advantages of using OAuth 2.0 for authentication is that your application can get permission to use other Google APIs on behalf of the user (such as YouTube, Google Drive, Calendar, or Contacts) at the same time as you authenticate the user."). To exchange the Authorization Code for an Access Token, Google requires the client agent to send a POST request containing the |

| '730 Patent Claim[2] | | Accused Instrumentality And Where Each Claim Element Is Found[3] |
|---|---|---|
| | | authorization code received from Google, as well as a client ID and client secret obtained when the developer registered the User Agent with Google Identity Services. *Id.* The Authorization Code received from Google Identity Service, thus, is a message permitting access to files and applications associated with Google APIs. *See also:*<br>  <br><br>For example, a user visits `example.com` on the Chrome browser on their Windows machine. This user has previously logged into `example.com` on their Android device and generated a passkey. On the Windows machine, the user chooses to sign in with a passkey from another device. The two devices will connect and the user will be prompted to approve the use of their passkey on the Android device, for example, with a fingerprint sensor. After doing so, they're signed in on the Windows machine. Note that the passkey itself isn't transferred to the Windows machine, so typically `example.com` will offer to create a new passkey there. That way, the phone isn't required next time the user wants to sign in. Read Sign-in with a phone to learn more.<br>PROX_GOOGLE_003243, Passwordless login with passkeys \| Authentication \| Google for Developers https://developers.google.com/identity/passkeys; Sign in with a passkey on a phone PROX_GOOGLE_003833, (https://www.youtube.com/watch?v=ywQ8bFla-L8). |
| *2* | The method of claim 1, wherein the one or more codes and the other data values are transmitted to the agent over a network. | As noted above with respect to claim 1, the signature and device-bound public key uniquely identifying the integrated device is sent to Google Identity Platform either via Wi-Fi or a cellular network to the internet. |

| '730 Patent Claim[2] | | Accused Instrumentality And Where Each Claim Element Is Found[3] |
|---|---|---|
| *3* | The method of claim 1, further comprising: registering an age verification for the user in association with the device ID code. | Resources provided by Google Identity utilize age verification (e.g., YouTube). PROX_GOOGLE_003197, How Google Accounts work when children turn 13 (or the applicable age in your country) - Google For Families Help (Noting that at the age of 13 users will lose access to YouTube Kids.). |
| *5* | The method of claim 1, wherein the biometric data and the scan data are both based on a fingerprint scan by the user. | Starting with Android 10, Google stored biometric data on Android devices includes fingerprint and facial recognition. PROX_GOOGLE_002913, https://source.android.com/docs/security/features/biometric#android10 ("Android 10: Introduces the BiometricManager class that developers can use to query the availability of biometric authentication. Includes fingerprint and face authentication integration for BiometricPrompt"). |
| *15.* | A system, comprising: | This preamble is not limiting.<br><br>Google uses, distributes, sells, and/or offers to sell a Universal Passwordless Architecture ("UPA") (e.g. PROX_GOOGLE_003243, https://developers.google.com/identity/passkeys), a system which includes: (a) Android OS, (b) Android SafetyNet Authenticator, (c) Chrome, and (d) Google Identity Platform ("GIP"), which together infringe, literally or under the doctrine of equivalents, the claimed limitations as presented below.<br><br>Google, through its own actions, and the action of its customers, which Google directs and controls, has created and sells[9] the use of its UPA system, including Android/Chrome preloaded[10] devices, servers, and software supporting and integrating into the UPA architecture. Servers integrating with and supporting the architecture include GIP providing FIDO and/or OAuth services. Devices integrating and supporting the architecture include devices utilizing the Android OS. Software integrating with and supporting the architecture include the Chrome web browser and Android SafetyNet Authenticator. |
| | a biometric key stores biometric data of a user and a plurality of codes and other data values comprising a device ID code uniquely identifying the biometric | <u>a biometric key stores biometric data of user … in a tamper proof format written to a storage element on the biometric key that is unable to be subsequently altered</u><br><br>Utilizing the Android operating system developed and disseminated by Google, Android devices store biometric data of the user in a tamper proof format written to a storage element on |

[9] *See e.g.* https://cloud.google.com/identity-platform/pricing.

[10] For the avoidance of doubt, "preloaded" includes devices that ship with Android and other accused software pre-installed or upon which Android and other accused software is otherwise installed by Defendant (e.g., through operating system upgrades, updates, initial installation, or otherwise).

| | | |
|---|---|---|
| | key and a secret decryption value in a tamper proof format written to a storage element on the biometric key that is unable to be subsequently altered, and if scan data can be verified as being from the user by comparing the scan data to the biometric data, wirelessly sending, one or more codes from the plurality of codes and other data values wherein the one or more codes and the other data values include the device ID code, and the biometric data is selected from a group consisting of a palm print, a retinal scan, an iris scan, a hand geometry, a facial recognition, a signature recognition and a voice recognition; and | the integrated device that is unable to be subsequently altered. Starting with Android 10, Google stored biometric data on Android devices includes fingerprint and facial recognition. PROX_GOOGLE_002913, https://source.android.com/docs/security/features/biometric#android10 ("Android 10: Introduces the BiometricManager class that developers can use to query the availability of biometric authentication. Includes fingerprint and face authentication integration for BiometricPrompt").<br><br>As to protect biometric data, Android's implementation guidelines require tamper-proof "raw fingerprint data or derivatives (for example, templates) must never be accessible from outside the sensor driver or TEE" (trusted execution environment) and "fingerprint acquisition, enrollment, and recognition must occur inside the TEE". Android Open-Source Project: Fingerprint HIDL, PROX_GOOGLE_003177, https://source.android.com/docs/security/features/authentication/fingerprint-hal Requiring acquisition and recognition to occur inside the TEE, fingerprint data never leaves the TEE. Android's TEE, called Trusty, "uses ARM's Trustzone™ to virtualize the main processor and create a secure trusted exaction environment" isolated from the rest of the system. Android Open-Source Project: Trusty TEE, PROX_GOOGLE_003283, https://source.android.com/docs/security/features/trusty. Accordingly, fingerprint data, which never leaves the TEE, also never leaves the Trustzone housing Trusty. Keeping biometric data within the Trustzone, Android devices persistently store biometric data in a tamper proof format.[11]<br><br>On Android devices, access to biometric hardware is controlled by Fingerprint HIDL. "Android uses Fingerprint Hardware Interface Definition Language (HIDL) to connect to a vendor-specific library and fingerprint hardware (for example, a fingerprint sensor)." Android Open Source Project: Fingerprint HIDL, PROX_GOOGLE_003177, https://source.android.com/docs/security/features/authentication/fingerprint-hal. The methods enabled by the Fingerprint HIDL do not permit altering biometric data. *Id*. (providing a listing of methods, none of which allows for the alternation of biometric data.). Keeping fingerprint and other biometric data within a portion of the Trustzone only accessible by Fingerprint HIDL, which lacks a method for altering biometric data, Android devices persistently store biometric |

[11] In the alternative, to the extent the TEE and Trustzone are considered to be a different security implementation or if limited alterations are possible in limited circumstances and given high levels of security permissions, clearance, or hardware-level permission attribution, the "tamper proof format . . . that is unable to b subsequently altered," limitation is met under the doctrine of equivalents on the grounds that the tamper proofing and limitations on alteration utilized in the Trustzone prevent nearly all alteration by nearly all programs that have not been explicitly approved and, at least, prevent alteration by all regular-world (non-TEE and non-Trustzone) applications and requires a strict level of physical and/or virtual layers of permission to be altered.

| | | |
|---|---|---|
| | | data of the user in a tamper proof format written to a storage element on the integrated device that is unable to be subsequently altered.<br><br>Android devices require user consent to enroll a fingerprint. *See* PROX_GOOGLE_003204, How to Add a New Fingerprint to Your Android Device - Technipages (Detailing the process of adding a fingerprint to an Android phone including entering a PIN or password for the device before adding a fingerprint.). As enrolling fingerprints on Android devices require entering PIN / Passcode / Password to evidence user consent, Android devices store biometric data of a user in a tamper proof format unable to be subsequently altered.<br><br>As Android devices store biometric data in a tamper proof format that is not capable of being subsequently altered, Google forces users of the Android OS, exercising control and direction over them by controlling every step that the software performs and every prompt for user action, to persistently store their biometric data in a tamper proof format written to a storage element on an integrated device that is not capable of being subsequently altered.<br><br>the biometric data is selected from a group consisting of a palm print, a retinal scan, an iris scan, a hand geometry, facial recognition, signature recognition and a voice recognition<br><br>Android's development documentation describes that biometric data may include, for example, face recognition or a fingerprint (a subset of a palm print): "One method of protecting sensitive information or premium content within your app is to request biometric authentication, such as using face recognition or fingerprint recognition. This guide explains how to support biometric login flows in your app." PROX_GOOGLE_003271, https://developer.android.com/training/sign-in/biometric-auth<br><br>a biometric key stores … a plurality of codes and other data values comprising a device ID code uniquely identifying the biometric key … … in a tamper proof format written to a storage element on the biometric key that is unable to be subsequently altered<br><br>The Android OS natively includes the Android SafetyNet Authenticator, which is FIDO certified and accessible via the FIDO2 API for Android. *See* PROX_GOOGLE_003793, FIDO® Certified - FIDO Alliance. In operation, authenticators store and access the cryptographic material underlying FIDO Passkeys. *See* PROX_GOOGLE_002819, AccountManager \| Android Developers ("Authenticators (which may be written by third parties) handle the actual details of validating account credentials and storing account information. For example, Google, Facebook, and Microsoft Exchange each have their own |

| | | |
|---|---|---|
| | | authenticator."); PROX_GOOGLE_003088, FIDO Technical Glossary (fidoalliance.org) (Defining an FIDO Authenticator as "responsible for user verification, and maintaining the cryptographic material required for the relying party authentication."); and PROX_GOOGLE_003439, Web Authentication: An API for accessing Public Key Credentials - Level 3 (w3.org) (Defining an authenticator as "A cryptographic entity, existing in hardware or software, that can register a user with a given Relying Party and later assert possession of the registered public key credential, and optionally verify the user, when requested by the Relying Party.").<br><br>Having a native authenticator, Android devices support the use of FIDO Passkeys. *See* PROX_GOOGLE_003187, Google Online Security Blog: Security of Passkeys in the Google Password Manager (googleblog.com) ("Passkeys are the result of an industry-wide effort. They combine secure authentication standards created within the FIDO Alliance and the W3C Web Authentication working group with common terminology and user experience across different platforms, recoverability against device loss, and a common integration path for developers. Passkeys are supported in Android and other leading industry client OS platforms."). Accordingly, instead of the traditional username and password combination, users of Android devices may utilize a passkey to access their Google Account. *See* PROX_GOOGLE_003192, Google Online Security Blog: So long passwords, thanks for all the phish (googleblog.com) ("Starting today, you can create and use passkeys on your personal Google Account."). Not only does the Android OS support the use of Passkeys, but Google also helped to create the standards upon which they are based. *See* PROX_GOOGLE_003192, Google Online Security Blog: So long passwords, thanks for all the phish (googleblog.com) ("Passkeys are built on the protocols and standards Google helped create in the FIDO Alliance and W3C WebAuthn working group. This means passkey support works across all platforms and browsers that adopt these standards. You can store the passkeys for your Google Account on any compatible device or service."). As the Android OS natively includes a FIDO certified authenticator and supports use of passkeys, Android devices must persistently store passkeys on the integrated device.<br><br>The specific variant of passkeys stored by the native Android OS authenticator includes a device ID code uniquely identifying an integrated device. "[P]asskeys on Android support the proposed Device-bound Public Key WebAuthn extension (*devicePubKey)*. If this extension is requested when creating or using passkeys on Android, relying parties will receive two signatures in the result: One from the passkey private key, which may exist on multiple devices, and an additional signature from a second private key that only exists on the current device." PROX_GOOGLE_003187, Google Online Security Blog: Security of Passkeys in the Google Password Manager (googleblog.com). The second private key is part of a key-pair. PROX_GOOGLE_003439, Web Authentication: An API for accessing Public Key Credentials |

| | | |
|---|---|---|
| | | - Level (pr-preview.s3.amazonaws.com) ("This authenticator registration extension and authentication extension provides a Relying Party with a "device continuity" signal for backup eligible credentials. This is done by creating a user credential-specific hardware-bound device key pair on the authenticator, if such a key pair does not already exist for the user credential being created or exercised, and returning the device public key along with a signature by the device private key to the Relying Party."). The key pair provided by the Device-Bound Public Key extension is unique to each Android device. PROX_GOOGLE_003439, Web Authentication: An API for accessing Public Key Credentials - Level (pr-preview.s3.amazonaws.com) ("The hardware-bound device key pair is not on its own a user credential and does not have its own credential ID. Instead, the returned device public key is a device-specific contextual attribute of its associated user credential."). As the key-pair is unique, each key of the pair is a unique device ID. However, only the public key is sent to uniquely identify an Android device. PROX_GOOGLE_003187, Google Online Security Blog: Security of Passkeys in the Google Password Manager (googleblog.com) ("This device-bound private key is unique to the passkey in question, and each response includes a copy of the corresponding device-bound public key."); and PROX_GOOGLE_003439, Web Authentication: An API for accessing Public Key Credentials - Level (pr-preview.s3.amazonaws.com) ("That is, when that user credential is used—along with the devicePubKey extension—on a particular authenticator, a particular device public key is returned by the extension, along with a signature demonstrating proof-of-possession of the device private key by that device."). Because device bound-public keys are not synced, "observing two passkey signatures with the same device-bound public key is a strong signal that the signatures are generated by the same device. On the other hand, if a relying party observes a device-bound public key it has not seen before, this may indicate that the passkeys has been synced to a new device." Google Online Security Blog: Security of Passkeys in the Google Password Manager, October 12, 2022, PROX_GOOGLE_003187, https://security.googleblog.com/2022/10/SecurityofPasskeysintheGooglePasswordManager.html. Providing an indication of the integrated device generating a signature, the public key of the Device-Bound Public Key extensions key-pair is a device ID code uniquely identifying the integrated device.<br><br>When a Device Bound Public Key is present, the integrated SafetyNet authenticator within the Android OS causes the device to store a private key bound to the device. PROX_GOOGLE_003439 at § 10.2.2.2 (Defining making a credential with a device bound public key as including "5. Let dpk be the newly created or existing device public key, in COSE_Key format in the same fashion as for the user credential's credentialPublicKey when the latter is conveyed in attested credential data. 6. Let devicePrivateKey be the newly created or existing device private key.")) The public key corresponding to the private key is returned to |

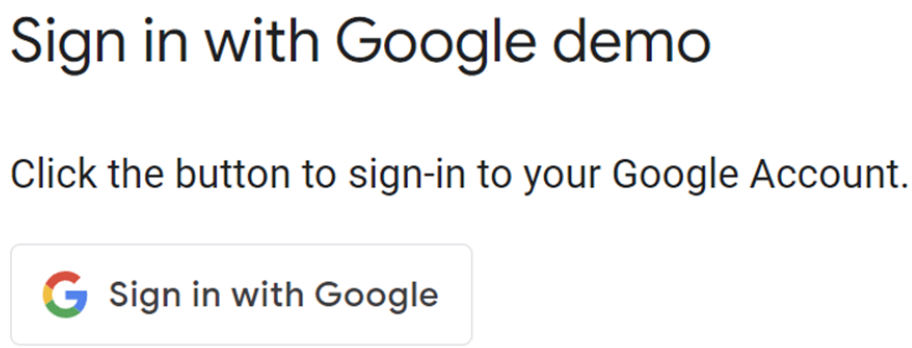
| | | |
|---|---|---|
| | | the relying party and saved. PROX_GOOGLE_003439 at § 10.2.2.3.1 (Stating registration at the relying party includes step "6. Create a new device-bound key record … add this device-bound key record to the devicePubKeys member of the new credential record.").) When processing an authentication request received from Google Identity, the Android OS authenticator creates a signature with the private key associated with the device-bound public key and then sends the signature and the device-bound public key back to Google Identity for verification. *Id.* at § 10.2.2.2 (Defining authentication as including "9. Let dpkSig be the result of signing the assertion signature input with devicePrivateKey. 10. Output dpkSig as the extension's unsigned extension output.").) Upon receipt of the authentication response, Google Identity extracts the device bound public key from the response and uses it to verify the signature. *Id.* at § 10.2.2.3.2 (Defining Relying Party verification of the authentication assertion as including "3. Extract the contained fields from attObjForDevicePublicKey: aaguid, dpk, scope, nonce, fmt, attStmt. 4. Verify that signature is a valid signature over the assertion signature input (i.e., authData and hash) by the device public key dpk.")) If the device bound public key were changed at any time, Google Identity would not be able to verify the signature as it would no longer correspond to the authenticator's device bound private key. Accordingly, by being part of a key pair, the device bound public key is in a format unable to be subsequently altered.<br><br>Of course, if both the private key and public key were altered, then the signature could be nominally verified. In such a situation, however, the altered device-bound public key would not match any of the device-bound public keys stored by the Relying Party (i.e., Google Identity). *See Id.* at § 10.2.2.3.2 (Steps 5 and 6 detailing, during an authentication ceremony, comparing the received device-bound public key to a list of stored device-bound public keys, where the lack of a match indicates a new device.). The Relying Party, accordingly, would treat the altered device-bound public key as representing a new device and not allow the request absent further user verification. Accordingly, by being recorded by the relying party, device-bound public keys are further placed in a format unable to be subsequently altered.[12]<br><br>As Google controls the protocol utilized by its Identity Platform, including the ability to sync passkeys across devices, users wanting the benefit of biometric authentication via passkeys have no choice but to store on their device a pair of keys generated by the device bound public key extension that uniquely identifies the integrated device in a tamper proof format written to a storage element on the integrated device that is unable to be subsequently altered. |

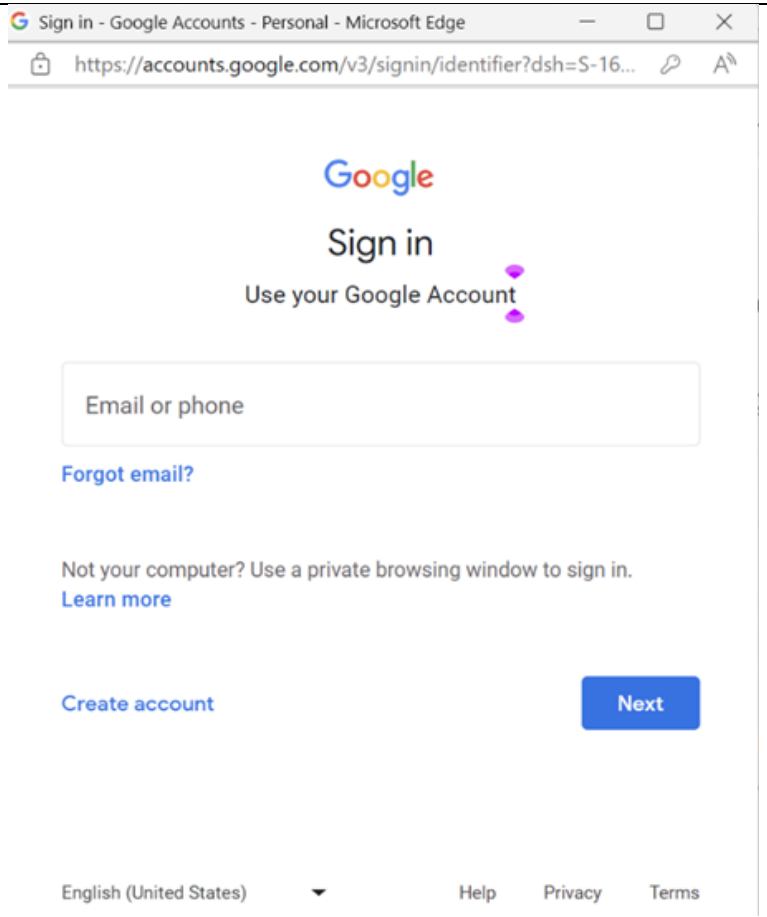
[12] In the alternative, the device-bound key pair performs the same function in the same way and achieves the same result as storage in a format that is unable to be subsequently altered in that alterations are rendered impossible because alterations break the key pair connection and thus would infringe under the doctrine of equivalents as well.

| | | |
|---|---|---|
| | | a biometric key stores … a secret decryption value in a tamper proof format written to a storage element on the biometric key that is unable to be subsequently altered<br><br>Passkeys in Google Password Manager are always end-to-end encrypted. See PROX_GOOGLE_003187. When a passkey is backed up, its private key is uploaded only in its encrypted form using an encryption key that is only accessible on the user's own devices. Id. Without access to the private key, an attacker cannot use the passkey to sign into its corresponding online account. Id. Accordingly, Android devices must contain a secret decryption value (i.e., the encryption key only accessible on the user's own device) to decrypt and use synced and backed-up passkeys. The decryption values are inherently in a tamper proof format as any alteration would prevent correct decryption of the encrypted passkey and thus render the passkey unusable. As such, Android devices contain a secret decryption value in a tamper proof format unable to be subsequently altered.<br><br>As such encryption is dictated by Google and implemented natively in the Android OS, users wanting the benefit of biometric authentication via passkeys have no choice but to store on their device a secret decryption value in a tamper proof format written to a storage element on the integrated device that is unable to be subsequently altered.<br><br>if scan data can be verified as being from the user by comparing the scan data to the biometric data,<br><br>Google describes the use of Passkeys as a biometric alternative to remembering passwords. PROX_GOOGLE_003243, Passwordless login with passkeys \| Authentication \| Google for Developers ("Passkeys are a safer and easier alternative to passwords. With passkeys, users can sign in to apps and websites with a biometric sensor (such as a fingerprint or facial recognition), PIN, or pattern, freeing them from having to remember and manage passwords.") Signing in with a biometric sensor necessarily requires comparing the scan data to the biometric data to determine whether the scan data matches the biometric data.<br><br>wirelessly sending, one or more codes from the plurality of codes and other data values wherein the one or more codes and the other data values include the device ID code<br><br>As detailed by Google using passkeys to sign into a service or application is as simple as unlocking their Android device with fingerprint or facial recognition. PROX_GOOGLE_003192, Google Online Security Blog: So long passwords, thanks for all the phish (googleblog.com) ("Passkeys are a more convenient and safer alternative to |

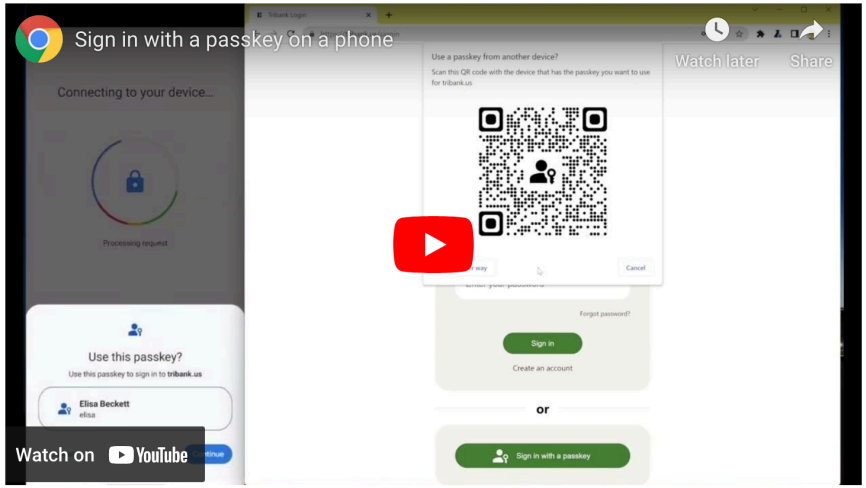
| | | |
|---|---|---|
| | | passwords. They work on all major platforms and browsers, and allow users to sign in by unlocking their computer or mobile device with their fingerprint, face recognition or a local PIN.”); and PROX_GOOGLE_003243, Passwordless login with passkeys | Authentication | Google for Developers (“When a user wants to sign in to a service that uses passkeys, their browser or operating system will help them select and use the right passkey. The experience is similar to how saved passwords work today. To make sure only the rightful owner can use a passkey, the system will ask them to unlock their device. This may be performed with a biometric sensor (such as a fingerprint or facial recognition), PIN, or pattern."). This ease of access is provided by Google's FIDO certified SafetyNet Authenticator utilizing the FIDO2 API for Android. *See* PROX_GOOGLE_003775 (“The FIDO2 API allows Android applications to create and use strong, attested public key- based credentials for the purpose of authenticating users. The API provides a WebAuthn Client implementation, which supports the use of BLE, NFC, and USB roaming authenticators (security keys) as well as a platform authenticator, which allows the user to authenticate using their fingerprint or screen lock.”) In operation, the SafetyNet Authenticator utilizes the native capacities of the Android OS, such as the Android Keystore to store the cryptographic public-private key pairs underlying the device-bound public key extension. *See* PROX_GOOGLE_003795 ((“The typical way to develop secure authenticator on Android mobile devices (smartphones and tablets) is to use a secured hardware-backed operating environment... This includes technology such as TEE (“Trusted Execution Environment”) that performs cryptographic and other sensitive operations including digital signing and biometric data processing used to support FIDO functionality... Android Keystore allows app developers to store cryptographic keys in a container and use them in cryptographic operations via APIs. Android also offers protection of the fingerprint sensor data via a TEE, which allows for encryption and cryptographic authentication... Adding a small application to complement some functions that are necessary to perform as UAF 1.1 authenticator enables Android Keystore with hardware-backed key attestations and fingerprint sensor to become a secure FIDO UAF 1.1 authenticator.”). The Android Keystore securely holds a passkey until it is unlocked by biometric verification. *See* PROX_GOOGLE_003785; *see also* PROX_GOOGLE_003821 (“When generating or importing a key into the Android KeyStore, you can specify that the key is only authorized to be used if the user has been authenticated. The user is authenticated using a subset of their secure lock screen credentials (pattern/PIN/password, biometric credentials).”). Biometric unlocking within the Android OS is provided by Fingerprint HIDL. *See* PROX_GOOGLE_003177, Fingerprint HIDL | Android Open Source Project (“On devices with a fingerprint sensor, users can enroll one or more fingerprints and use those fingerprints to unlock the device and perform other tasks. Android uses the Fingerprint Hardware Interface Definition Language (HIDL) to connect to a vendor-specific library and fingerprint hardware (for example, a fingerprint sensor)”). Accordingly, by |

| | | |
|---|---|---|
| | | utilizing Fingerprint HIDL in combination with the KeyStore, the Android OS releases a passkey after a determination that the scan data matches the biometric data.<br><br>As detailed supra, Google utilizes a device-bound public key extension, which requires concurrently sending a public-key and a signature generated with a device-bound private to be verified with the public-key. PROX_GOOGLE_003439, Web Authentication: An API for accessing Public Key Credentials - Level (pr-preview.s3.amazonaws.com) ("That is, when that user credential is used—along with the devicePubKey extension—on a particular authenticator, a particular device public key is returned by the extension, along with a signature demonstrating proof-of-possession of the device private key by that device."). As Android devices possess cellular and/or Wi-Fi capability, the signature and device-bound public key uniquely identifying the integrated device sent to Google Identity will be sent wirelessly – either via Wi-Fi or a cellular network to the internet.<br><br>While the native Android OS authenticator permits the use of a passkey in a variety of situations, each transmits the device-bound public key and corresponding signature over the internet. When a user wishes to use a passkey stored on their Android device to access a resource via a browser the user is presented with either a QR code or push notification. *See* PROX_GOOGLE_003236, Passkey support on Android and Chrome \| Authentication \| Google for Developers ("Depending on the desktop operating system (e.g. ChromeOS, iOS, MacOS, Windows) users may be presented with a QR code to securely use a passkey stored on their mobile device or a notification may be displayed prompting the user to unlock their phone to use the relevant passkey."). In the case of a QR code, the user scans a QR code displayed on the screen of a device. PROX_GOOGLE_003192, Google Online Security Blog: So long passwords, thanks for all the phish (googleblog.com) ("When you do need to use a passkey from your phone to sign in on another device, the first step is usually to scan a QR code displayed by that device."). The proximity between the devices is then verified using Bluetooth. PROX_GOOGLE_003239, Passkeys use cases \| Authentication \| Google for Developers ("The two devices verify that they are in proximity with each other using Bluetooth."). The Bluetooth connection is also utilized to establish a secure connection between the devices via the internet. PROX_GOOGLE_003192, Google Online Security Blog: So long passwords, thanks for all the phish (googleblog.com) ("The device then verifies that your phone is in proximity using a small anonymous Bluetooth message and sets up an end-to-end encrypted connection to the phone through the internet.") Utilizing the internet connection, the Android device sends a one-time passkey signature, including the device-bound public key, to the other device, which is used to sign in the user. PROX_GOOGLE_003239, Passkeys use cases \| Authentication \| Google for Developers ("A one-time passkey signature is transferred back to Safari on the Mac, which the website then |

| | | |
|---|---|---|
| | | uses to sign the user in."). Establishing the connection over the internet would require utilizing either the device's Wi-Fi or cellular capabilities – both of which are wireless protocols. Accordingly, responsive to a determination that the scan data matches the biometric data, Android devices wirelessly send a passkey signature and accompanying device-bound public key uniquely identifying the device.<br><br>Push notifications can originate with the "Sign in with Google" button. To receive the benefit of quickly and easily managing user authentication utilizing "Sign-in with Google", the website must display a "Sign in with Google" button, which is generated by the Google Identity Services JavaScript library, or provide the user the option to sign in with a passkey. PROX_GOOGLE_003779 ("Put it another way, the Sign in with Google button must be generated by the Google Identity Services JavaScript library now. The button rendering API allows you to customize the color, shape, text, and size to meet the branding requirements of your website, whereas still stick to Google's guidelines."); and PROX_GOOGLE_003239. A functional demonstration of the "Sign in with Google" button is provided in Google Identity documentation, as shown below.<br><br>  <br><br>*See* PROX_GOOGLE_003232.<br><br>When this button is clicked, the dialog box pictured below is presented to the user. |




The user enters an email and then clicks "next". If the user has activated "Use your phone to sign in" in their Google Account, a dialog box is presented directing the user to complete the sign on with their phone, which has received a push notification. The push notification includes a challenge from Google Identity servers for FIDO client on the user's Android phone. The FIDO client will invoke the Android SafetyNet Authenticator to complete the challenge by prompting the user to biometrically unlock the stored passkey, use the passkey to sign a challenge, and return the signature and accompanying device-bound public key to Google Identity, as detailed above. Returning the signed challenge in response to the push notification requires establishing a connection over the internet utilizing either the device's Wi-Fi or cellular capabilities – both of which are wireless protocols. Accordingly, responsive to a determination that the scan data matches the biometric data, Android devices wirelessly send a passkey signature and accompanying device-bound public key uniquely identifying the device. *See also:*

| | | |
|---|---|---|
| | |  <br>For example, a user visits `example.com` on the Chrome browser on their Windows machine. This user has previously logged into `example.com` on their Android device and generated a passkey. On the Windows machine, the user chooses to sign in with a passkey from another device. The two devices will connect and the user will be prompted to approve the use of their passkey on the Android device, for example, with a fingerprint sensor. After doing so, they're signed in on the Windows machine. Note that the passkey itself isn't transferred to the Windows machine, so typically `example.com` will offer to create a new passkey there. That way, the phone isn't required next time the user wants to sign in. Read Sign-in with a phone to learn more.<br><br>PROX_GOOGLE_003243, Passwordless login with passkeys \| Authentication \| Google for Developers https://developers.google.com/identity/passkeys; Sign in with a passkey on a phone PROX_GOOGLE_003833 , (https://www.youtube.com/watch?v=ywQ8bFla-L8). |
| | an authentication unit receives the plurality of codes and the other data values and sends the plurality of codes and the other data values to agent for authentication to determine whether one or more codes and the other data values are legitimate, wherein the agent is a third-party trusted authority possessing a list of device ID codes uniquely identifying legitimate | an authentication unit receives the plurality of codes and other data values<br><br>Google Chrome operating on a device forms an authentication unit that receives the plurality of codes and other data values. PROX_GOOGLE_003243, Passwordless login with passkeys \| Authentication \| Google Developers ("When a user wants to sign in to a service that uses passkeys, their browser or operating system will help them select and use the right passkey."); and PROX_GOOGLE_003236, Passkey support on Android and Chrome \| Authentication \| Google Developers ("Chrome on all desktop platforms supports using passkeys from mobile devices."). |

| | | |
|---|---|---|
| | integrated devices, and responsive to the device ID code being authenticated, the authentication unit receiving an access message from the agent allowing the user to access an application, wherein the application is selected from a group consisting of a casino machine, a keyless lock, a garage door opener, an ATM machine, a hard drive, computer software, a web site and a file. | <u>an authentication unit … send the plurality of codes and the other data values to agent for authentication</u><br><br>The responsibility for sending responses received from an authenticator falls upon the WebAuthn Client. PROX_GOOGLE_003287, Web Authentication: An API for accessing Public Key Credentials - Level 2 (w3.org), § 4 ("A WebAuthn Client is an intermediary entity typically implemented in the user agent (in whole, or in part). Conceptually, it underlies the Web Authentication API and embodies the implementation of the [[Create]](origin, options, sameOriginWithAncestors) and [[DiscoverFromExternalSource]](origin, options, sameOriginWithAncestors) internal methods. It is responsible for both marshaling the inputs for the underlying authenticator operations and for returning the results of the latter operations to the Web Authentication API's callers."). Google identifies its Chrome browser as a WebAuthn client by noting it supports the use of passkeys from mobile devices, including permitting the use of an Android phone as a roaming authenticator on Windows. PROX_GOOGLE_003236, Passkey support on Android and Chrome \| Authentication \| Google Developers ("Chrome on all desktop platforms supports using passkeys from mobile devices."); and PROX_GOOGLE_003236, Passkey support on Android and Chrome \| Authentication \| Google Developers (Providing a table indicating "Can sign in with phone" is supported on Chrome running on the macOS, iOS, and Windows.). As such, Chrome will forward the credential received from the Android device to Google Identity.<br><br><u>to an agent for authentication to determine whether one or more codes and the other data values are legitimate</u><br><br>As detailed supra, Google utilizes the device-bound public key extension, which requires concurrently sending a public-key and a signature generated with a device-bound private to be verified with the public-key. PROX_GOOGLE_003439, Web Authentication: An API for accessing Public Key Credentials - Level (pr-preview.s3.amazonaws.com) ("That is, when that user credential is used—along with the devicePubKey extension—on a particular authenticator, a particular device public key is returned by the extension, along with a signature demonstrating proof-of-possession of the device private key by that device."). The extension allows Google to determine if a sign-in attempt is coming from a new device by observing whether it has previously seen the accompanying device-bound public key. PROX_GOOGLE_003187, Google Online Security Blog: Security of Passkeys in the Google Password Manager (googleblog.com) ("Observing two passkey signatures with the same device-bound public key is a strong signal that the signatures are generated by the same device. On the other hand, if a relying party observes a device-bound public key it has not seen before, this may indicate that |

| | | |
|---|---|---|
| | | the passkey has been synced to a new device.") Making such observations requires that the device-bound public key operates as a device ID identifying legitimate devices.<br><br>As Google controls the protocol utilized by its Identity platform, including the use of device-bound public key extension, users wanting to sign in to their Google Account without a password via biometric authentication, have no choice but to send a device ID code to determine whether the one or more codes and other data values are legitimate.<br><br><u>wherein the agent is a third-party trusted authority</u><br><br>As detailed *supra*, a device ID in the form of a device-bound public key and a signature is sent from the Android device to the Google Identity for authentication.<br><br>Google Identity utilizes the federation protocol OpenID Connect. PROX_GOOGLE_003181, Google Identity \| Google Developers ("Google's OAuth 2.0 APIs conform to the OpenID Connect specification, are OpenID Certified, and can be used for both authentication and authorization."); and PROX_GOOGLE_003213, OpenID Connect \| Authentication \| Google Developers ("Google's OAuth 2.0 authentication system supports the required features of the OpenID Connect Core specification."). The OpenID Connect protocol includes OpenID provider (OP) which is a server providing authentication claims to an application or website. *See* PROX_GOOGLE_003103, Final: OpenID Connect Core 1.0 incorporating errata set 1, § 1.2 (Defining OpenID Provider (OP) as "OAuth 2.0 Authorization Server that is capable of Authenticating the End-User and providing Claims to a Relying Party about the Authentication event and the End-User"; and a Relying Party (RP) as "OAuth 2.0 Client application requiring End-User Authentication and Claims from an OpenID Provider."). Providing authentication claims an application relies upon, the OP is a trusted authority. Google Identity operates as the OpenID Provider by enabling apps and websites to authenticate users using a Google Account. PROX_GOOGLE_003232, Overview \| Authentication \| Google Developers ("Sign in with Google helps you to quickly and easily manage user authentication on your website. Users sign into a Google account, provide their consent, and securely share their profile information with your platform."). As Google Identity is neither the application nor the website being accessed, it is a third party to the transaction. Google Identity thus operate as a third-party trusted authority for authentication.<br><br><u>possessing a list of device ID codes uniquely identifying legitimate integrated devices</u> |

| | | |
|---|---|---|
| | | Google maintains a list of legitimate devices for each user. The FIDO standard utilized by Google Identity Service incorporates the WebAuthn standard. *See* **PROX_GOOGLE_003789** ("Today, WebAuthn is part of the FIDO Alliance's FIDO2 specifications and the FIDO Alliance runs certificate programs to ensure compliance."). One of the use cases for WebAuthn is decommissioning lost, stolen, and disregarded devices such that "assertions signed by this credential are rejected." *See* Web Authentication: An API for accessing Public Key Credentials - Level 3, Sec. 1.3.5 Decommissioning, **PROX_GOOGLE_003439**. A user may decommission devices they no longer use by signing out of them via their Google Account. *See* **PROX_GOOGLE_003791** (Instructing users how to secure lost, disregarded, or stolen devices by signing out of them.). To prevent account access from a device, and mark it as no longer legitimate, a user must sign out of the device and all sessions associated with the device. *Id* ("If you want to make sure there's no account access from a device, sign out of all the sessions with this device name."); and **PROX_GOOGLE_003787** ("To not get Google Prompts on a device, sign out of your Google Account on that device."). Accordingly, by maintaining a list of devices the user has selected to remain signed in on, Google Identity maintains a list of legitimate integrated devices.<br><br>Additionally, as noted above, Google Identity utilizes the device-bound public key extension. If this extension is requested when creating or using passkeys on Android, relying parties will receive two signatures in the result: One from the passkey private key, which may exist on multiple devices, and an additional signature from a second private key that only exists on the current device." PROX_GOOGLE_003187, Google Online Security Blog: Security of Passkeys in the Google Password Manager (googleblog.com). The second private key is part of a key-pair. PROX_GOOGLE_003439, Web Authentication: An API for accessing Public Key Credentials - Level (pr-preview.s3.amazonaws.com) ("This authenticator registration extension and authentication extension provides a Relying Party with a "device continuity" signal for backup eligible credentials. This is done by creating a user credential-specific hardware-bound device key pair on the authenticator, if such a key pair does not already exist for the user credential being created or exercised, and returning the device public key along with a signature by the device private key to the Relying Party."). The key pair provided by the Device-Bound Public Key extension is unique to each Android device. PROX_GOOGLE_003439, Web Authentication: An API for accessing Public Key Credentials - Level (pr-preview.s3.amazonaws.com) ("The hardware-bound device key pair is not on its own a user credential and does not have its own credential ID. Instead, the returned device public key is a device-specific contextual attribute of its associated user credential."). As the key-pair is unique, each key of the pair is a unique device ID. However, only the public key is sent to uniquely identify an Android device. PROX_GOOGLE_003187, Google Online Security Blog: Security of Passkeys in the Google Password Manager (googleblog.com) ("This device-bound |

| | | |
|---|---|---|
| | | private key is unique to the passkey in question, and each response includes a copy of the corresponding device-bound public key.”); and PROX_GOOGLE_003439, Web Authentication: An API for accessing Public Key Credentials - Level (pr-preview.s3.amazonaws.com) (“That is, when that user credential is used—along with the devicePubKey extension—on a particular authenticator, a particular device public key is returned by the extension, along with a signature demonstrating proof-of-possession of the device private key by that device.”). Because device bound-public keys are not synced, "observing two passkey signatures with the same device-bound public key is a strong signal that the signatures are generated by the same device. On the other hand, if a relying party observes a device-bound public key it has not seen before, this may indicate that the passkeys has been synced to a new device." Google Online Security Blog: Security of Passkeys in the Google Password Manager, October 12, 2022, PROX_GOOGLE_003187, https://security.googleblog.com/2022/10/SecurityofPasskeysintheGooglePasswordManager.html. To provide an indication that the integrated device generating a signature has not been seen before, Google Identity must maintain a list of the device-bound public keys code uniquely identifying the integrated device.<br><br><u>responsive to the device ID code being authenticated, the authentication unit receiving an access message from the agent allowing the user to access an application</u><br><br>Authentication is provided by a FIDO server operated by Google in conjunction with OpenID Connect Authentication Server. While OpenID Connect is a federation protocol, it is compatible and complementary with FIDO. “The value of a FIDO authentication capability is amplified by a federated system, where the federation system extends the benefits of a FIDO authentication to applications and services without requiring FIDO to be directly integrated with those applications.” PROX_GOOGLE_003072, Enterprise_Adoption_Best_Practices_Federation_FIDO_Alliance.pdf (fidoalliance.org), page 4. When a federated system, such as OpenID Connect, is combined with FIDO, the OpenID Provider (OP) sends a FIDO server challenge to an authenticator which is returned as the FIDO Authentication response. PROX_GOOGLE_003072, Enterprise_Adoption_Best_Practices_Federation_FIDO_Alliance.pdf (fidoalliance.org), page 10 (Detailing the combined use FIDO and federation entails “3… The Authentication Authority determines that the FIDO authentication specified in the previous request is both relevant… 4. The Authentication Authority sends a FIDO Server challenge… 6. FIDO Client returns FIDO authentication response… 7. FIDO Server validates the FIDO response and reports same to Authentication Authority which looks up the user and then redirects the User Agent back to the Application Provider with an authentication assertion.”). |

| | | |
|---|---|---|
| | | FIDO incorporates the WebAuthn Protocol. PROX_GOOGLE_002926, Client to Authenticator Protocol (CTAP) (fidoalliance.org), § 1.1 ("This specification is part of the FIDO2 project, which includes this specification and is related to the W3C [WebAuthn] specification."); and PROX_GOOGLE_003287, Web Authentication: An API for accessing Public Key Credentials - Level 2 (w3.org), § 1.1 ("Along with the Web Authentication API itself, this specification defines a request-response cryptographic protocol—the WebAuthn/FIDO2 protocol—between a WebAuthn Relying Party server and an authenticator, where the Relying Party's request consists of a challenge and other input data supplied by the Relying Party and sent to the authenticator.") Accordingly, the FIDO server challenge sent will a be WebAuthn authenticatorGetAssertion request. The response to the request received will contain a signature generated by the authenticator's private key and a credential ID. PROX_GOOGLE_002926, Client to Authenticator Protocol (CTAP) (fidoalliance.org), § 6.2.2 (Detailing the response to authenticationGetAssertion as containing "PublicKeyCredentialDescriptor structure containing the credential identifier whose private key was used to generate the assertion" and "an assertion signature".); and PROX_GOOGLE_003287, Web Authentication: An API for accessing Public Key Credentials - Level 2 (w3.org), § 6 (Defining an "assertion signature" as "an assertion signature asserts that the authenticator possessing a particular credential private key has established, to the best of its ability, that the user requesting this transaction is the same user who consented to creating that particular public key credential."). Upon receiving the response, the WebAuthn/FIDO server will use the credential ID to locate the appropriate public key to verify the signature generated with the private key. PROX_GOOGLE_003287, Web Authentication: An API for accessing Public Key Credentials - Level 2 (w3.org), § 7.2 ("7. Using credential.id (or credential.rawId, if base64url encoding is inappropriate for your use case), look up the corresponding credential public key and let credentialPublicKey be that credential public key… 20. Using credentialPublicKey, verify that sig is a valid signature over the binary concatenation of authData and hash.") Accordingly, a FIDO/WebAuthn server operated by Google Identity will authenticate a device ID (i.e., credential ID). After validating the device ID, the FIDO/WebAuthn portion of the Identity Platform "redirects the user agent back to the Application Provider with an authentication assertion". PROX_GOOGLE_003072, Enterprise_Adoption_Best_Practices_Federation_FIDO_Alliance.pdf (fidoalliance.org), page 10.<br><br>After authentication of the passkey by Google Identity, the user may be allowed access to an app or website. PROX_GOOGLE_002906, Android Developers Blog: Bringing passkeys to Android & Chrome (googleblog.com) ("Developers can build passkey support on their sites for end-users using Chrome via the WebAuthn API, on Android and other support platforms… Our next milestone in 2022 will be an API for native Android Apps."). Google Identity utilizes an OpenID Certified implementation of OAuth 2.0. PROX_GOOGLE_003213, OpenID |

| | | |
|---|---|---|
| | | Connect \| Authentication \| Google for Developers ("Google's OAuth 2.0 APIs can be used for both authentication and authorization. This document describes our OAuth 2.0 implementation for authentication, which conforms to the OpenID Connect (PROX_GOOGLE_003199, https://openid.net/connect/) specification, and is OpenID Certified (PROX_GOOGLE_002921, https://openid.net/certification/).) An authentication request sent to Google Identity elicits a response including an authorization code that can be sent to Google Identity to receive an ID Token and Access Token. PROX_GOOGLE_003213, OpenID Connect \| Authentication \| Google for Developers ("The response includes a code parameter, a one-time authorization code that your server can exchange for an access token and ID token."). The ID Token contains user information sufficient to allow access to the app or website should the user be registered. *Id.,* ("After obtaining user information from the ID token, you should query your app's database. If the user already exists in your database, you should start an application session for that the user if all login requirements are met by the Google API response."). The Access Token provides permission to use other Google APIs on behalf of the user, such as Google Drive or YouTube. *Id.,* ("One of the advantages of using OAuth 2.0 for authentication is that your application can get permission on behalf of the user (such as YouTube, Google Drive, Calendar, or Contracts) at the same time as you authenticate the user… The access token that you receive back from Google allows you to access all the APIs related to the scopes of access you requested and were granted."). As such, the authorization code is exchangeable for ID and access tokens providing access to various applications. Being received from Google Identity acting as third-party trust authority and being exchangeable for tokens providing access to various applications, the authorization code is an access message received from the agent allowing the user access to an application.<br><br>wherein the application is selected from a group consisting of a casino machine, a keyless lock, a garage door opener, an ATM machine, a hard drive, computer software, a web site, and a file.<br><br>If Google Identity verifies the signed challenge and that the public key corresponds to a legitimate device, an access message is returned in the form of an authorization code, which is used to log into a website or application by being exchanged for the appropriate token. If resources in addition to the website or application are to be accessed, the Authorization Code may be exchanged for an ID Token or an Access Token. To make the exchange, Google requires the client agent (i.e., a website loaded in the computer's browser) to send a POST request containing the authorization code received from Google, as well as a client ID and client secret obtained when the developer registered the User Agent with Google Identity Services. See PROX_GOOGLE_003213. The information contained in the ID Token is then used to query the developer's database to determine if the user exists, and if so, to start a session for the user |

| | | |
|---|---|---|
| | | given access to the website or application. Id. ("After obtaining user information from the ID Token, you should query your app's database. If the user already exists in your database, you should start an application session for that user if all login requirements are met by the Google API response."). The Authorization Code received from Google Identity is a message permitting access to the application or website by being exchangeable for an ID Token used to allow access.<br><br>If, however, the User Agent needs to access a resource such as a file in Google Drive or another resource, the authentication code may be exchanged for an Access Token. Id. ("One of the advantages of using OAuth 2.0 for authentication is that your application can get permission to use other Google APIs on behalf of the user (such as YouTube, Google Drive, Calendar, or Contacts) at the same time as you authenticate the user."). To exchange the Authorization Code for an Access Token, Google requires the client agent to send a POST request containing the authorization code received from Google, as well as a client ID and client secret obtained when the developer registered the User Agent with Google Identity Services. *Id.* The Authorization Code received from Google Identity Service, thus, is a message permitting access to files and applications associated with Google APIs. |
| 16 | The system of claim 15, wherein the biometric key receives an authentication request from the authentication unit, and in response, requests a biometric scan from the user to generate the scan data. | The Android OS natively includes the Android SafetyNet Authenticator, which is FIDO certified and accessible via the FIDO2 API for Android. *See* PROX_GOOGLE_003793, FIDO® Certified - FIDO Alliance. During an authentication ceremony, an authenticator receives an "authenticatorGetAssertion" request to provide cryptographic proof of user authentication. PROX_GOOGLE_002926, Client to Authenticator Protocol (CTAP) (fidoalliance.org), § 6.2 (Defining authenticatorGetAssertion as the method "used by a host to request cryptographic proof of user authentication as well as user consent to a given transaction, using a previously generated credential that is bound to the authenticator and relying party identifier."). The FIDO's CTAP protocol details the processing of this request. As noted in the protocol, when a user verification option is present in the request and set as true, the authenticator will attempt user verification, returning an error if user verification is not successful. PROX_GOOGLE_002926, Client to Authenticator Protocol (CTAP) (fidoalliance.org), § 6.2.2 ("2. If the "uv" option is present and set to true (implying the pinUvAuthParam parameter is not present, and that the authenticator supports an enabled built-in user verification method, see Step 4): 1. Let internalRetry be true. 2. Let uvState be the result of calling performBuiltInUv(internalRetry) 3. If uvState is error: 1. If the error reason is a user action timeout, then return CTAP2_ERR_USER_ACTION_TIMEOUT. 2. If the ClientPin option ID is true and the noMcGaPermissionsWithClientPin option ID is absent or false, end the operation by returning CTAP2_ERR_PUAT_REQUIRED. 3. If the uvRetries counter is <= 0, return CTAP2_ERR_PIN_BLOCKED. 4. Otherwise, end the operation by returning |

| | | |
|---|---|---|
| | | CTAP2_ERR_OPERATION_DENIED. 4. If uvState is success: 1. Set the "uv" bit to true in the response.”).<br><br>As detailed by Google using passkeys to sign into a service or application is as simple as unlocking their Android device with fingerprint or facial recognition. PROX_GOOGLE_003192, Google Online Security Blog: So long passwords, thanks for all the phish (googleblog.com) (“Passkeys are a more convenient and safer alternative to passwords. They work on all major platforms and browsers, and allow users to sign in by unlocking their computer or mobile device with their fingerprint, face recognition or a local PIN.”); and, PROX_GOOGLE_003243, Passwordless login with passkeys | Authentication | Google for Developers (“When a user wants to sign in to a service that uses passkeys, their browser or operating system will help them select and use the right passkey. The experience is similar to how saved passwords work today. To make sure only the rightful owner can use a passkey, the system will ask them to unlock their device. This may be performed with a biometric sensor (such as a fingerprint or facial recognition), PIN, or pattern.”). For such biometric authentication to occur, the authenticator logically must request a biometric scan from the user to generate scan data. |
| 17 | The system of claim 15, wherein if the biometric key cannot verify the scan data as being from the user, it does not send one or more codes and the other data values. | The Android OS natively includes the Android SafetyNet Authenticator, which is FIDO certified And accessible via the FIDO2 API for Android. *See* PROX_GOOGLE_003793, FIDO® Certified - FIDO Alliance. During an authentication ceremony, an authenticator receives an “authenticatorGetAssertion” request to provide cryptographic proof of user authentication. PROX_GOOGLE_002926, Client to Authenticator Protocol (CTAP) (fidoalliance.org), § 6.2 (Defining authenticatorGetAssertion as the method “used by a host to request cryptographic proof of user authentication as well as user consent to a given transaction, using a previously generated credential that is bound to the authenticator and relying party identifier.”). When the native Android OS authenticator receives an authentication request in the form of an authenticatorGetAssertion request with the user verification option present and set as true, the authenticator will attempt user verification, returning an error if user verification is not successful. PROX_GOOGLE_002926, Client to Authenticator Protocol (CTAP) (fidoalliance.org), § 6.2.2 (“2. If the "uv" option is present and set to true (implying the pinUvAuthParam parameter is not present, and that the authenticator supports an enabled built-in user verification method, see Step 4): 1. Let internalRetry be true. 2. Let uvState be the result of calling performBuiltInUv(internalRetry) 3. If uvState is error: 1. If the error reason is a user action timeout, then return CTAP2_ERR_USER_ACTION_TIMEOUT. 2. If the ClientPin option ID is true and the noMcGaPermissionsWithClientPin option ID is absent or false, end the operation by returning CTAP2_ERR_PUAT_REQUIRED. 3. If the uvRetries counter is <= 0, return CTAP2_ERR_PIN_BLOCKED. 4. Otherwise, end the operation by returning CTAP2_ERR_OPERATION_DENIED. 4. If uvState is success: 1. Set the "uv" bit to true in the response.”). As such, only an error will be returned if scan data cannot be verified. |

# Exhibit A2

**U.S. Patent Number 8,352,730 – Preliminary Infringement Contentions**[1]

| | |
|---|---|
| **Assignee:** | Proxense, LLC |
| **Title:** | Biometric personal data key (PDK) authentication |
| **Filing Date:** | 2005-12-20 |
| **Publication Date:** | 2013-01-08 |
| **Inventor:** | Giobbi, John J. |

| **'730 Patent Claim**[2] | | **Accused Instrumentality And Where Each Claim Element Is Found**[3] |
|---|---|---|
| 1 | A method comprising[4]: | This preamble is not limiting.<br><br>Android devices including Google Pay Services (also known as Google Wallet), which is part of Google Play Services, carry out the claimed method, literally or by the doctrine of equivalents, for at least the reasons set forth below: |
| | persistently storing biometric data of the | Persistently storing biometric data of a user … in a tamper proof format written to a storage element on the integrated device that is unable to be subsequently altered |

[1] The Preliminary Infringement Contentions (PICS) provided herein are based on information obtained to date and may not be exhaustive. Plaintiff's investigation of Defendants' infringement is ongoing. Plaintiff reserves the right to supplement and/or amend these PICS to identify additional instrumentalities and to further identify where each element of each claim is found in each accused instrumentality, including on the basis of discovery obtained from Defendants, and from third parties during the course of this litigation, pursuant to ¶2 of the Order Governing Proceedings – Patent Cases under Hon. Alan D. Albright.

[2] All PICS set forth herein for any independent patent claims are hereby incorporated by reference into the PICS alleged for any dependent patent claims that depend on such independent claims, as if fully set forth therein.

[3] The Accused Instrumentalities and associated exhibits discussed and/or cited for any claim herein are representative in all material respects of all other accused instrumentalities identified for that claim (e.g., a specified phone or service may be used as a representative example in these charts since the other accused instrumentalities have immaterial differences in their hardware and/or software configuration, the cited references are believed to be illustrative of all such accused devices).

[4] Plaintiff's inclusion of any claim preamble in this claim chart should not be interpreted as an admission that the preamble is limiting. Plaintiff reserves the right to take the position that the claim preambles are limiting or not limiting on a claim-by-claim basis.

| | | |
|---|---|---|
| | user and a plurality of codes and other data values comprising a device ID code uniquely identifying the integrated device and a secret decryption value in a tamper proof format written to a storage element on the integrated device that is unable to be subsequently altered; wherein the biometric data is selected from a group consisting of a palm print, a retinal scan, an iris scan, a hand geometry, a facial recognition, a signature recognition and a voice recognition; | Utilizing the Android operating system developed and disseminated by Google, Android devices store biometric data of the user in a tamper proof format written to a storage element on the integrated device that is unable to be subsequently altered. Starting with Android 10, stored biometric data on Android devices includes fingerprint and facial recognition. PROX_GOOGLE_002913, https://source.android.com/docs/security/features/biometric#android10 ("Android 10: Introduces the BiometricManager class that developers can use to query the availability of biometric authentication. Includes fingerprint and face authentication integration for BiometricPrompt").<br><br>As to protecting biometric data, Android's implementation guidelines require tamper-proof "raw fingerprint data or derivatives (for example, templates) must never be accessible from outside the sensor driver or TEE" (trusted execution environment) and "fingerprint acquisition, enrollment, and recognition must occur inside the TEE". Android Open-Source Project: Fingerprint HIDL, PROX_GOOGLE_003177, https://source.android.com/docs/security/features/authentication/fingerprint-hal Requiring acquisition and recognition to occur inside the TEE, fingerprint data never leaves the TEE. Android's TEE, called Trusty, "uses ARM's Trustzone™ to virtualize the main processor and create a secure trusted exaction environment" isolated from the rest of the system. Android Open-Source Project: Trusty TEE, PROX_GOOGLE_003283, https://source.android.com/docs/security/features/trusty. Accordingly, fingerprint data, which never leaves the TEE, also never leaves the Trustzone housing Trusty. Keeping biometric data within the Trustzone, Android devices persistently store biometric data in a tamper proof format.[5]<br><br>On Android devices, access to the biometric hardware is controlled by Fingerprint HIDL. "Android uses Fingerprint Hardware Interface Definition Language (HIDL) to connect to a vendor-specific library and fingerprint hardware (for example, a fingerprint sensor)." Android Open Source Project: Fingerprint HIDL, PROX_GOOGLE_003177, https://source.android.com/docs/security/features/authentication/fingerprint-hal. The methods enabled by the Fingerprint HIDL do not permit altering biometric data. *Id*. (providing a listing of methods, none of which allowing for the alternation of biometric data.). Keeping fingerprint and other biometric data within a portion of the Trustzone only accessible by Fingerprint HIDL, which lacks a method for |

[5] In the alternative, to the extent the TEE and Trustzone are considered to be a different security implementation or if limited alterations are possible in limited circumstances and given high levels of security permissions, clearance, or hardware-level permission attribution, the "tamper proof format . . . that is unable to be subsequently altered," limitation is met under the doctrine of equivalents on the grounds that the tamper proofing and limitations on alteration utilized in the Trustzone prevent nearly all alteration by nearly all programs that have not been explicitly approved and, at least, prevent alteration by all regular-world (non-TEE and non-Trustzone) applications and requires a strict level of physical and/or virtual layers of permission to be altered.

| | | |
|---|---|---|
| | | altering biometric data, Android devices persistently store biometric data of the user in a tamper proof format written to a storage element on the integrated device that is unable to be subsequently altered.<br><br>Android devices require user consent to enroll a fingerprint. *See* PROX_GOOGLE_003204, How to Add a New Fingerprint to Your Android Device - Technipages (Detailing the process of adding a fingerprint to an Android device as including entering a PIN or password for the device before adding a fingerprint.). As enrolling fingerprints on Android devices requires entering PIN / Passcode / Password to evidence user consent, Android devices store biometric data of user in a tamper proof format unable to be subsequently altered.<br><br>As Android devices store biometric data of a in a tamper proof format that is not capable of being subsequently altered, Google forces users of the Android OS (through direct prompts), exercising control and direction over them by controlling every step that the software performs and every prompt for user action, to persistently store their biometric data in a tamper proof format written to a storage element on an integrated device that is not capable of being subsequently altered.<br><br>the biometric data is selected from a group consisting of a palm print, a retinal scan, an iris scan, a hand geometry, a facial recognition, a signature recognition and a voice recognition<br><br>Android devices include biometric sensors used by Google for authentication purposes, which provide biometric data. Android's development documentation describes that biometric data may include, for example, face recognition or a fingerprint (a subset of a palm print): "One method of protecting sensitive information or premium content within your app is to request biometric authentication, such as using face recognition or fingerprint recognition. This guide explains how to support biometric login flows in your app." PROX_GOOGLE_003271, https://developer.android.com/training/sign-in/biometric-auth<br><br>persistent storing … a plurality of codes and other data values comprising a device ID code uniquely identifying the integrated device … in a tamper proof format written to a storage element on the integrated device that is unable to be subsequently altered<br><br>Safeguarding credit card information with EMV payment tokens, including Google Pay Services, which is part of Google Play Services, persistently store a device ID code uniquely identifying the smartphone in a tamper proof format written to a storage element on the integrated device that is unable to be subsequently altered. "Google [was] among the first to implement EMV payment tokens in digital wallets that hold credentials for several payments use cases." PROX_GOOGLE_003987, EMV Payment Tokenization Primer and Lessons Learned, U.S. Payments Forum (2019), page 12. "EMV payment tokens are open-loop tokens provisioned by a TSP and, like other tokens, are used to replace |

| | | |
|---|---|---|
| | | the actual payment credential (e.g., PAN) with another numeric value." Id., at 8. Payment tokens, accordingly, are issued ("provisioned") to Android devices exchange for a credit card number by a token service provider (TSP), such as Visa, MasterCard, Discover, and American Express. PROX_GOOGLE_003987, EMV Payment Tokenization Primer and Lessons Learned, U.S. Payments Forum (2019), at 23 (Figure 5 – identifying Visa, MasterCard, American Express and Discover as Token Service Providers); and PROX_GOOGLE_003872, Overview \| Device Tokenization Developer Site \| Google for Developers ("When a user successfully adds their card to Google Pay, Google Pay stores a uniquely generated token on the device that has its own value."). "Payment tokens may vary depending on implementation, but typically there is a unique token for each device, which bears no resemblance to the PAN (e.g., the final four digits do not match)." PROX_GOOGLE_003987, EMV Payment Tokenization Primer and Lessons Learned, U.S. Payments Forum (2019), at 8 (emphasis added); and PROX_GOOGLE_003872, Overview \| Device Tokenization Developer Site \| Google for Developers ("Though each user tokenizes a card using the same PAN, each wallet receives a unique DPAN"). EMV Payment tokens, which are device-specific, are stored by Android devices including Google Pay Services as "device ID codes." PROX_GOOGLE_003872, Overview \| Device Tokenization Developer Site \| Google for Developers ("The Google Pay component of Google Play services manages tokenization, identification and verification (ID&V), token lifecycle management, and payment taps. These features will work, even if the Google Wallet app is not installed on the device assuming the user is in a supported country and is using a compatible device.")<br><br>A device ID is defined in the Specification as a "value that uniquely identifies" a device, which may be "provided during enrollment." U.S. Patent No. 8,352,730, Col. 5, ll. 44-51 ("[I]n one embodiment the code is a device ID or other value that uniquely identifies biometric key 100… In other embodiments, the code is provided during enrollment and/or the biometric data are provided during manufacturing."). EMV payment tokens uniquely identifies devices, as no two devices will have the same payment token. Replacing a provided credit card number during provisioning, EMV tokens are assigned during enrollment of the credit card number on a particular device. Accordingly, EMV payment tokens are values uniquely identifying Android devices provided during card enrollment, and therefore an identified embodiment of device ID codes.<br><br>After receiving device ID codes / EMV payment tokens, Androids with Google Payment Services store tokens with data encryption. Android devices including Google Pay Services support host card emulation (HCE). PROX_GOOGLE_003855, Host-based card emulation overview \| Android Developers ("Android 4.4 and higher provide an additional method of card emulation that doesn't involve a secure element, called *host-based card emulation*.") Devices utilizing HCE store tokens within local device storage. EMV Payment Tokenisation Specification: Technical Framework v2.3, page 41 ("Local device storage: e.g. Payment Token and related data stored using the data storage mechanisms of a Consumer Device, such as when utilising HCE"). Android devices encrypt all data |

| | | |
|---|---|---|
| | | once a device's encryption is enabled. "Once a device is encrypted, all user-created data is automatically encrypted before committing it to disk and all reads automatically decrypt data before returning it to the calling process. Encryption ensures that even if an unauthorized party tries to access the data, they won't be able to read it." PROX_GOOGLE_003853, Android Security Features \| Android Open Source Project. Encrypting data so that it cannot be read prevents meaningful alteration of data, *i.e.*, placing it into a tamper proof format then written to a storage element on the integrated device that is unable to be subsequently altered. In the alternative, this element is met under the doctrine of equivalents. *See* F.N. 5, *supra.*<br><br>In addition to utilizing HCE, Android devices permit the use of secure elements to store tokens. PROX_GOOGLE_003855, Host-based card emulation overview \| Android Developers ("Many Android-powered devices that offer NFC functionality already support NFC card emulation. In most cases, the card is emulated by a separate chip in the device, called a secure element. Many SIM cards provided by wireless carriers also contain a secure element."). Secure elements are recognized as "a dynamic environment to store data securely, process data securely and perform communication with external entities securely," that "will not allow unauthorized access." PROX_GOOGLE_003987, EMV Payment Tokenization Primer and Lessons Learned, U.S. Payments Forum (2019), page 41. Securely storing EMV payment tokens on a secure element not allowing unauthorized access, Android devices including Google Pay Service therefore persistently store a device ID code uniquely identifying the device in a tamper proof format, written to a storage element on the integrated device, that is unable to be subsequently altered.<br><br><u>persistently storing … a secret decryption value in a tamper proof format written to a storage element on the integrated device that is unable to be subsequently altered;</u><br><br>Android's implementation guidelines protect biometric data by requiring "fingerprint templates must be signed with a private, device-specific key." Android Open Source Project: Fingerprint HIDL, https://source.android.com/secruity/authentication/fingerprint-hal. Private key operations include singing and decrypting. Android Open-Source Project: Keymaster Functions, PROX_GOOGLE_004030, https://source.android.com/security/keystore/implementer-ref ("Private key operations (KeyPurpose::DECYPT and KeyPurpose::SIGN)".). The private key utilized to sign a fingerprint template is a secret decryption value. Within the Android OS, "Keystore API and Keymaster components provide hardware-backed cryptography for secure key storage in a secure environment, such as the Trusted Execution Environment (TEE)." Android Open-Source Project: Fingerprint HIDL, PROX_GOOGLE_003177, https://source.android.com/secruity/authentication/fingerprint-hal. Possessing the Keystore, Android devices persistently store a secret decryption value in a tamper proof format written to a storage element on the integrated device that is unable to be subsequently altered. |

| | | |
|---|---|---|
| | responsive to receiving a request for a biometric verification of the user, receiving scan data from a biometric scan; | Google Pay requires that a device is unlocked in order to function, which on Android may occur responsive to receiving a request for biometric verification of the user and in turn receiving that scan data from a biometric scan. "If a device is locked, the Google Pay app can't be opened." PROX_GOOGLE_003879, Set up a screen lock for the Google Pay app – Android – Google Pay Help. Android devices must be unlocked to permit the use of Google Pay, which can be done with biometrics, including fingerprint, Iris Scan, and 3D face scan. *See* PROX_GOOGLE_003879 , Set up a screen lock for the Google Pay app – Android – Google Pay Help (Presenting a list of acceptable screen locks that work with the Google Pay App). When utilizing a fingerprint to unlock an Android device, the user touches their fringe to a sensor. *See* PROX_GOOGLE_003880, Unlock IIndroid device to verify it's you - Google Account Help ("**Fingerprint**: Touch your finger to a sensor."). On Android devices, "the fingerprint sensor of [the] device is generally idle", but "in response to a call to authenticate … the fingerprint sensor listens for a touch". Android Open Source Project: Fingerprint HIDL, PROX_GOOGLE_003177, https://source.android.com/security/authentication/fingerprint-hal. As the device may be unlocked with a touch of fingerprint sensor, when at the lock screen the Android device's lock must request biometric verification of the user. After the user places their fingerprint on the sensor, a "vendor-specific library determines if there is a fingerprint match in the current set of enrolled fingerprint templates." *Id*. Determining whether there is a fingerprint match necessarily requires the vendor-specific library on the Android device to receive scan data from the biometric (fingerprint) scan. |
| | comparing the scan data to the biometric data to determine whether the scan data matches the biometric data; | Google Pay requires that a device is unlocked in order to function, which further involves comparing scan data received from a biometric scan to a baseline.<br><br>"If a device is locked, the Google Pay app can't be opened." PROX_GOOGLE_003879, Set up a screen lock for the Google Pay app - Android - Google Pay Help. Android devices must be unlocked to permit the use of Google Pay, which can be done with biometrics, including fingerprint, Iris Scan, and 3D face scan. *See* PROX_GOOGLE_003879, Set up a screen lock for the Google Pay app - Android - Google Pay Help (Presenting a list of acceptable screen locks that work with the Google Pay App. When utilizing a fingerprint to unlock an Android device, the user touches their fringe to a sensor. *See* PROX_GOOGLE_003880, Unlock your Android device to verify it's you - Google Account Help ("**Fingerprint**: Touch your finger to a sensor."). On Android devices, "the fingerprint sensor of [the] device is generally idle", but "in response to a call to authenticate … the fingerprint sensor listens for a touch". Android Open Source Project: Fingerprint HIDL, https://source.android.com/security/authentication/fingerprint-hal. On Android devices, "the fingerprint sensor of [the] device is generally idle", but "in response to a call to authenticate … the fingerprint sensor listens for a touch". Android Open Source Project: Fingerprint HIDL, https://source.android.com/security/authentication/fingerprint-hal. After the user places their fingerprint on the sensor, a "vendor-specific library determines if there is a fingerprint match in the current set of enrolled fingerprint templates." *Id*. Determining if there is a fingerprint match, requires the vendor-specific library on |

| | | |
|---|---|---|
| | | Google Pay preloaded smartphones compare scan data from the sensor to biometric data to determine whether the scan data matches the biometric data. |
| | responsive to a determination that the scan data matches the biometric data, wirelessly sending one or more codes from the plurality of codes and the other data values for authentication by an agent that is a third-party trusted authority possessing a list of device ID codes uniquely identifying legitimate integrated devices, wherein the one or more codes and other data values includes the device ID code | <u>responsive to a determination that the scan data matches the biometric data, wirelessly sending one or more codes from the plurality of codes and the other data values</u><br><br>Google Pay requires that a device is unlocked in order to function, which further involves, after determining that the scan data matches the biometric data, wirelessly sending codes, including a device ID code.<br><br>Before Google Pay can be accessed, the Android device needs to be unlocked. PROX_GOOGLE_003879, Set up a screen lock for the Google Pay app - Android - Google Pay Help ("If a device is locked, the Google Pay app can't be opened."). The inability to use Google Pay while the device is locked is determined by Google Pay's service manifest declaration having the requireDeviceUnlock attribute set to true. *See* PROX_GOOGLE_003855, Host-based card emulation overview \| Android Developers ("The <host-apdu-service> tag must contain an <android:description> attribute that contains a user-friendly description of the service that you can show in the app UI. You can use the requireDeviceUnlock attribute to specify that the device is unlocked before you invoke this service to handle APDUs."). Because Google Pay has the requireDeviceUnlock attribute set to ture, when a user taps the reader with their device locked, they will be prompted to unlock their device. PROX_GOOGLE_003855, Host-based card emulation overview \| Android Developers ("If you set the android:requireDeviceUnlock attribute to true for your HCE service, Android prompts the user to unlock the device when the following happen: the user taps an NFC reader. the NFC reader selects an AID that is resolved to your service.").<br><br>As detailed *supra*, Android devices may be unlocked with a fingerprint. When utilizing a fingerprint to unlock an Android device, the user touches their fringe to a sensor. *See* PROX_GOOGLE_003880, Unlock your Android device to verify it's you - Google Account Help ("**Fingerprint**: Touch your finger to a sensor."). On Android devices, "the fingerprint sensor of [the] device is generally idle", but "in response to a call to authenticate … the fingerprint sensor listens for a touch". Android Open Source Project: Fingerprint HIDL, https://source.android.com/security/authentication/fingerprint-hal. On Android devices, "the fingerprint sensor of [the] device is generally idle", but "in response to a call to authenticate … the fingerprint sensor listens for a touch". Android Open Source Project: Fingerprint HIDL, https://source.android.com/security/authentication/fingerprint-hal. After the user places their fingerprint on the sensor, a "vendor-specific library determines if there is a fingerprint match in the current set of enrolled fingerprint templates." *Id*. Determining if there is a fingerprint match, requires the vendor-specific library on Google Pay preloaded smartphones compare scan data from the sensor to |

| | | |
|---|---|---|
| | | biometric data to determine whether the scan data matches the biometric data. Only when the scan data matches the biometric will the device by unlocked as to allow the use of Google Pay.<br><br>After Android device has been unlocked, Google Pay service wirelessly sends one or more codes, including device ID code. PROX_GOOGLE_003872, Overview \| Device Tokenization Developer Site \| Google for Developers ("The Google Pay component of Google Play services manages tokenization, identification and verification (ID&V), token lifecycle management, and payment taps. These features will work, even if the Google Wallet app is not installed on the device assuming the user is in a supported country and is using a compatible device."). "Google [was] among the first to implement EMV payment tokens in digital wallets that hold credentials for several payments use cases." PROX_GOOGLE_003987, EMV Payment Tokenization Primer and Lessons Learned, U.S. Payments Forum (2019), page 12. Accordingly, Android devices with Google Pay service contain payment tokens. EMV payment tokens, which, as detailed supra, are values uniquely identifying Android devices provided during card enrollment, and thus are an identified embodiment of device ID codes.<br><br>Tapping a credit card is an example of contactless payment. "In a contactless payment transaction, the consumer holds the contactless card, device, or mobile phone in close proximity (less than 2-4 inches) to the terminal and the payment account information is communicated wirelessly (via radio frequency [RF]) or NFC." PROX_GOOGLE_003987, EMV Payment Tokenization Primer and Lessons Learned, U.S. Payments Forum (2019), page 39. Payment taps are managed by Google Pay services. PROX_GOOGLE_003872, Overview \| Device Tokenization Developer Site \| Google for Developers ("The Google Pay component of Google Play services manages tokenization, identification and verification (ID&V), token lifecycle management, and payment taps. These features will work, even if the Google Wallet app is not installed on the device assuming the user is in a supported country and is using a compatible device."). Thus, by utilizing NFC, Android devices with Google Pay Service wirelessly transmit payment information to in-store terminals. Furthermore, as Google Pay can only be used after the device is unlocked, only responsive to a determination that the scan data matches the biometric data will the Android device wirelessly EMV payment token (i.e., device ID code) from the plurality of codes and other data values stored on the device.<br><br><u>responsive to a determination that the scan data match the biometric data, … sending one or more codes and other values from the plurality of codes and other data value for authentication … wherein the one or more codes includes the device ID code</u><br><br>When shopping, the merchant requires payment. But first, the merchant must seek authorization for payment. The series of requests and responses permitting the merchant to receive payment, as summarized in Figure 10.1 of PROX_GOOGLE_003881, the EMV Payment Tokenisation Specification: Technical Framework, v2.2 (2020), begins with the merchant submitting a Token |

| | | |
|---|---|---|
| | | Payment Request for transaction routing. The data included within the Token Payment Request, as detailed in Table 10.1 of PROX_GOOGLE_003881, EMV Payment Tokenisation Specification: Technical Framework, v2.2 (2020), is required to contain the payment token.<br><br>During transaction routing within the payment network, the Token Payment Request is transformed to a Token Authorization Request, as detailed in Figure 10.1 of PROX_GOOGLE_003881, the EMV Payment Tokenisation Specification: Technical Framework, v2.2 (2020). As detailed in Table 10.3 of PROX_GOOGLE_003881, the EMV Payment Tokenisation Specification: Technical Framework, v2.2 (2020), a required field of the Token Authorization Request generated during routing through the payment network is the payment token. Accordingly, the payment token sent in the Token Payment Request from the merchant continues to persist during the token authorization request process. EMV payment tokens, as detailed supra, are values uniquely identifying Google Pay preloaded smartphones provided during card enrollment, and thus are an identified embodiment of device ID codes. Accordingly, an embodiment of device ID codes are included within the one or more codes and other data wireless sent.<br><br><u>responsive to a determination that the scan data matches the biometric data, wirelessly sending … for authentication by an agent that is a third-party trusted authority possessing a list of device ID codes uniquely identifying legitimate integrated devices</u><br><br>"The Token Authorisation request process continues until De-Tokenisation has been completed." PROX_GOOGLE_003881, EMV Payment Tokenisation Specification: Technical Framework, v2.2 (2020), page 86. De-Tokenisation is performed by the token service provider. Visa, MasterCard, American Express and Discover each take on the role as token service providers. PROX_GOOGLE_003987, EMV Payment Tokenization Primer and Lessons Learned, U.S. Payments Forum (2019), page 23 (Figure 5 – identifying Visa, MasterCard, American Express and Discover as Token Service Providers). "Token Service Providers are responsible for a number of discrete functions which may include, but are not limited to: Maintenance and operation of a Token Vault … [and] DeTokenisation". PROX_GOOGLE_003881, EMV Payment Tokenisation Specification: Technical Framework, v2.2 (2020), page 19.<br><br>Maintaining the token vault and providing de-tokenisation, token service providers keep a list of device ID codes uniquely identifying legitimate integrated devices. The token vault maintained by Visa, MasterCard, and other token service providers is a "repository that maintains the established Payment Token / Token Expiry Date mapping to the underlying PAN / PAN Expiry Date and includes Payment Token related data". Maintaining the mapping between payment tokens and underlying primary account numbers (PAN), the token vault represents a list of legitimate tokens. EMV payment tokens, as detailed supra, are values uniquely identifying Google Pay preloaded smartphones provided during card |

| | | |
|---|---|---|
| | | enrollment, and thus are an identified embodiment of device ID codes. As tokens are embodiments of device ID codes, the listing of legitimate payment tokens maintained by token service providers as part of the token vault is an identified embodiment of a list of device ID codes uniquely identifying legitimate integrated devices. Being responsible for maintaining such as listing as a token vault, token service providers are one embodiment of a third-party trusted authority possessing a list of device ID codes uniquely identifying legitimate integrated devices.<br><br>Opening the token vault to perform de-tokenisation occurs in response to receiving the token authorization request containing the payment token. De-Tokenisation is "the process of converting a Payment Token and Token Expiry Date to its underlying PAN and PAN Expiry Date based on the Payment Token / Token Expiry Date mapping to the underlying PAN / PAN Expiry Date stored in the Token Vault." Id, at 6. "The Payment Token SHALL be de-tokenised to the underlying PAN in the incoming Token Authorisation prior to sending the PAN Authorisation to the Card Issuer." Id., at 91. Detokenizing in response to an incoming token authorization requests requires the token service provider be sent the token authorization request. As noted supra, the token authorization request sent to the token service provider contains the payment token (i.e. device ID code) wirelessly sent from an Android device including Google Pay Services after successful biometric authentication. Authenticating a user via biometrics, as noted above, entails a vendor-specific library on the Google Pay preloaded smartphones comparing scan data from a sensor to biometric data to determine whether the scan data matches the biometric data. Detokenizing in response to an incoming token authorization request containing a payment token provided by Android devices including Google Pay Services following successful biometric authentication, therefore, requires the token service provider is wirelessly sent one or more codes from the plurality of codes and the other data values for authentication responsive to a determination that the scan data matches the biometric data. |
| | responsive to authentication of the one or more codes and the other data values by the agent, receiving an access message from the agent allowing the user access to an application, wherein the application is selected from a group consisting of a casino machine, a keyless lock, | <u>responsive to authentication of the one or more codes and the other data values by the agent … allowing the user access to an application, wherein the application is selected from a group consisting of a casino machine, a keyless lock, a garage door opener, an ATM machine, a hard drive, computer software, a web site and a file</u><br><br>As it travels through the payment network to the token service provider, the payment token wirelessly transmitted from a Google Pay preloaded smartphone becomes part of the token payment request sent by the merchant, which is transformed to a token authorization request, prior to de-tokenization. See PROX_GOOGLE_003881, EMV Payment Tokenisation Specification: Technical Framework, v2.2 (2020), Figure 10.1 and page 81 ("The basic authorisation flow is shown in Figure 10.1."). De-Tokenisation is "the process of converting a Payment Token and Token Expiry Date to its underlying PAN and PAN Expiry Date based on the Payment Token / Token Expiry Date mapping to the underlying PAN / PAN Expiry Date stored in the Token Vault." Id, at 6. "Token Service Providers are responsible |

| | | |
|---|---|---|
| | a garage door opener, an ATM machine, a hard drive, computer software, a web site and a file. | for a number of discrete functions which may include, but are not limited to: Maintenance and operation of a Token Vault … [and] De-Tokenisation". Id, at 19.<br><br>Maintaining the token vault and performing de-tokenization, a token service provider receives the token authorization request. Upon receiving the request, "the Payment Token SHALL be de-tokenised to the underlying PAN in the incoming Token Authorisation prior to sending the PAN Authorisation to the Card Issuer." Id., at 91. As shown in Figure 10.1, "Once a Payment Token has been de-tokenised, the final request step is to initiate a PAN Authorisation, destined for the Card Issuer's authorisation system". Id. at 92. The PAN authorization request sent to the Card Issuer, contains the "underlying PAN from token vault". Id., Table 10.9, first row, at 93. Obtaining the underlying primary account number from the token vault, as to prepare the PAN authorization request, requires gaining access to the file containing the account number. Converting the payment token to its underlying account number based on the Payment Token / Token Expiry Date mapping to the underlying PAN / PAN Expiry Date stored in the token vault, as to determine which file access, the token service provider necessarily authenticates the payment token received from the Google Pay preloaded smartphone. By sending an authorization request for payment containing the account number from the token vault, after mapping the payment token to the to the account number, the token service provider authenticates the payment token and the other data values and allows the user access to the corresponding file within the token vault.<br><br>Upon receipt of the PAN authorization request, i.e., responsive to authentication of the one or more codes and other data values by the token service provider, the issuer completes final authorization and sends an authorization response back to the token service provider. . Accordingly, the PAN authorization request sent from the token service provider (acting as third party trust authority) is an access message allowing the user to access the issuer's computer software necessary to process and authorize the payment. <u>responsive to authentication of the one or more codes and the other data values by the agent, receiving an access message from the agent allowing the user access to an application</u><br><br>After the purchase has been authorized, Android devices including Google Pay Services smartphones receive a push notification with details of the transaction. PROX_GOOGLE_003870, Manage your Google Pay settings - United States - Google Pay Help ("**Important:** You'll always get payment notifications."). As the purchase can only be made after the primary account number mapped to the token was released from the token vault and placed in an authorization request sent to the card issuer, the push notification received indicates allowed access to the file containing the account number and the issuer's software utilized process and authorize the payment. As shown in Figures 6, 7 and 11 of PROX_GOOGLE_003987, EMV Payment Tokenization Primer and Lessons Learned, U.S. Payments Forum (2019), this confirmation of access is either sent directly from the token service provider or indirectly from the token service provider as one of series messages flowing back to the Android device including Google Pay Services. Accordingly, Android devices with Google Pay will receive an access |

| | | |
|---|---|---|
| | | message in the form a push notification sent from in the token service provide acting as a third-party trust authority (i.e., agent) which allowed access to the issuer's payment authorization software and the token vault file containing the actual account number. |
| 2 | The method of claim 1, wherein the one or more codes and other data values are transmitted to the agent over a network | The payment token (i.e., Device ID Code of the one or more codes and other data values) will be transmitted to the token service provider acting as third party trusted authority (i.e., agent) over the payment network. |
| 5 | The method of claim 1, wherein the biometric data and the scan data are both based on a fingerprint scan by the user. | Starting with Android 10, stored biometric data on Android devices includes fingerprint and facial recognition. PROX_GOOGLE_002913, https://source.android.com/docs/security/features/biometric#android10 ("Android 10: Introduces the BiometricManager class that developers can use to query the availability of biometric authentication. Includes fingerprint and face authentication integration for BiometricPrompt"). |