The Transfer Access Protocol - Moving to New Authenticators in the FIDO Ecosystem

Technical Report UW-CSE-17-06-01

Alex Takakuwa*, Tadayoshi Kohno*, and Alexei Czeskis^

* Paul G. Allen School of Computer Science & Engineering
University of Washington
^ Google Inc.

1. Introduction: Summary of Problem

The FIDO Alliance[1] envisions a world without passwords, providing the tools to revolutionize the way users authenticate on the web. The current ecosystem provides secure standards that promise to improve online account security and simplify the experience for internet connected users. This ecosystem allows users to sign into web services through authenticators (for example, a smartphone or dedicated token) that perform user authentication using an asymmetric cryptographic signature that is resistant to phishing attacks and provides two-factor authentication. Similar to the iPhone's TouchID, users on many platforms will have devices, such as phones, that can serve as FIDO authenticators. For example, imagine that a user is using a phone as an authenticator. This phone has an app that allows the user to view and manage keys. It also allows the user to log-in to websites using FIDO authentication. When the user goes to example.com and selects "log-in with authenticator", the phone alerts the user to scan a fingerprint. The user complies and the server and authenticator app negotiate in a cryptographic protocol to ensure that the user is safely authenticated and consents to the log-in.

There are, however, some unsolved problems in this ecosystem. In particular, in this paper we discuss the problems that arise and propose solutions for the following (likely common) scenario: A user is using a phone as an authenticator but wants to replace that device with a newer model, ceasing to use the old phone. The user, therefore, needs to set up the new phone as an authenticator and remove credentials from the old phone. To solve some of the problems that arise, we propose the Transfer Access Protocol, the concepts of which we describe in this paper.

[1] "FIDO Alliance." https://fidoalliance.org/. Accessed 2 Jun. 2017.

2. Goals of the Transfer Access Protocol

The goal of this work is to transfer account access from the authenticator app on the old phone to the authenticator app on the new phone. However, we would like to do so while preserving desirable properties of the existing FIDO authentication scheme. Here we discuss our primary goals in detail and some of the challenges that arise. In short, we do not want the addition of a Transfer Access Protocol to affect the user experience or the security and privacy properties of the existing FIDO authentication scheme.

2.1 The User Experience

Ideally, when a user buys a new phone, transferring authenticator access should work seamlessly, not adding or changing steps for the user either when they set up the new phone or during the next log-in. For example, during the initial phone setup, one possible implementation could simply attach the Transfer Access Protocol to the Tap & Go[2] feature in Android 5.0+. In the current implementation of Tap & Go, when a user first boots up a new phone, they see the screen sequence from Figure 1. If desired, we could simply ask the user whether they would like to use the new device as an authenticator and remove access from the old phone. In that case, the user would see an extra screen as in Figure 2. We stress that this is just an example implementation and is not necessary for the Transfer Access Protocol described in this paper. One could, for example, choose to make the Transfer Access Protocol completely transparent during the Tap & Go procedure or make the process independent of Tap & Go, instead processing the Transfer Access within the authenticator app itself. In principle, none of the concepts discussed here require any extra steps.

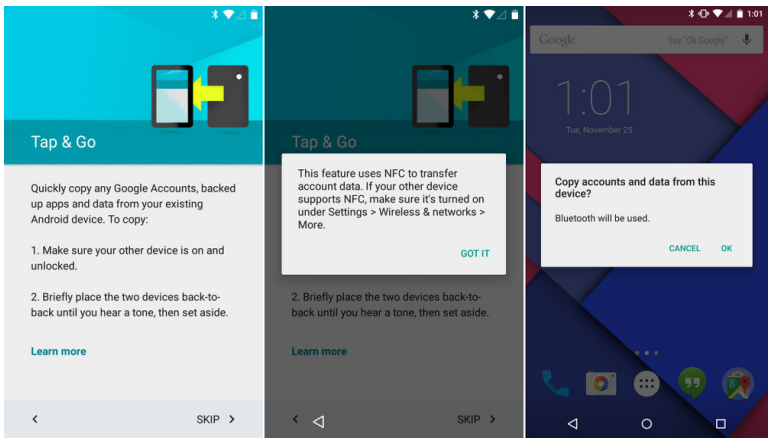


Figure 1: The stock Tap & Go implementation for Android 5.0 and up. This allows a user to quickly set up a new phone by transferring apps and data from the old one. Screenshots from Droid-Life[3].

---

[2] "Set up your new Nexus device - Nexus Help - Google Support." https://support.google.com/nexus/answer/6073630?hl=en Accessed 12 Jun. 2017.

[3] "Android 5.0 Feature: Tap & Go Restore and Restore From ... - Droid Life." 25 Nov. 2014, http://www.droid-life.com/2014/11/25/android-5-0-feature-tap-go-restore-and-restore-from-specific-devices/. Accessed 15 Jun. 2017.

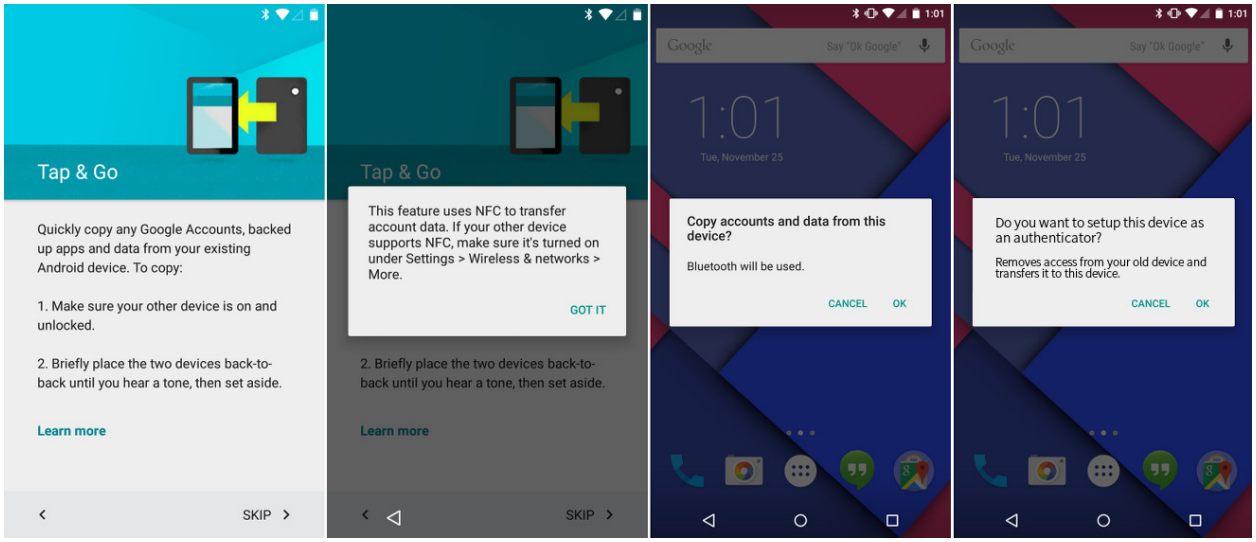


Figure 2: A potential user flow with the Transfer Access protocol added.
Original screenshots from Droid-Life[4]

After setting up the new phone, users will navigate to sites (or open each native application) as they did before. If a user is accustomed to seeing a log-in screen as in Figure 3, we would not like to change that user experience. In fact, we envision that the Transfer Access Protocol would keep the cryptographic transfer transparent to the user as it does during normal FIDO authentication so the next time the user logs in on the new phone, the user experience does not change at all.

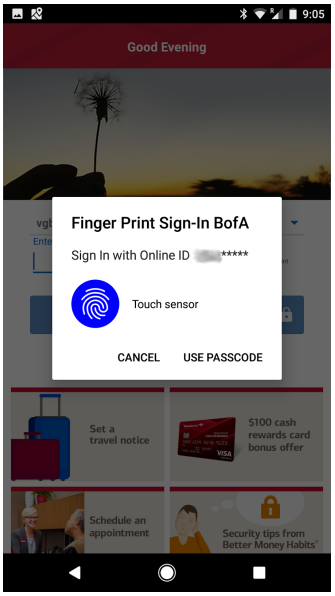
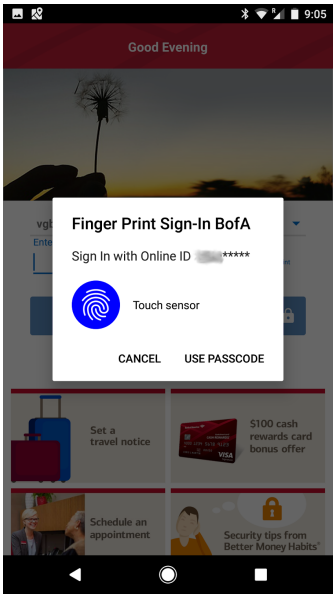
Before:  After: 

Figure 3: An example user experience for an app log-in.
This screenshot is from the Bank of America app on a Google Pixel

[4] "Android 5.0 Feature: Tap & Go Restore and Restore From ... - Droid Life." 25 Nov. 2014, http://www.droid-life.com/2014/11/25/android5-0-feature-tap-go-restore-and-restore-from-specific-devices/. Accessed 15 Jun. 2017.

2.2 Security

For the Transfer Access Protocol, we seek to preserve the security and privacy properties of the existing FIDO authentication scheme. For example, FIDO authentication can prevent web and network attackers from phishing or copying credentials, defend against Man-In-The-Middle attacks, provide clone detection, allow relying parties to revoke access or prevent registration of untrusted hardware, and prevent relying parties from colluding to link user accounts. However, a number of attacks are still out of scope. For example, the FIDO authentication scheme does not explicitly protect against attackers who can simultaneously attack network traffic and the local wireless environment, attackers who can compromise a user's PC and personal device, nor would it protect against a malicious FIDO application or operating system compromise. Because the Transfer Access Protocol transfers FIDO authenticator access, rather than performing an independent security and privacy analysis of each piece of this protocol, we aim to design a protocol that introduces no additional vulnerabilities. Throughout the description of this work, we will discuss some of the relevant and interesting decisions we make through the lens of concerns raised in previous papers on asymmetric authentication schemes. As an example of some of the properties we wish to uphold, we list the following from Lang et al[5].:

- Phishing: The protocol should not be phishable, nor should the resulting credentials.
- Defend Against MITM: Attackers who Man-In-The-Middle the connection, for example between the browser and relying party server, should not gain an advantage by attacking during any step in the Transfer Access Protocol.
- Session-Duplication: The protocol should not aid in the ability for stealing credentials to result in session-duplication, for example, by exposing long-term cookies or passwords.
- Prevent Session Riding: The protocol should not allow an adversary to gain access to an existing session or to a future existing session.
- Trusted Hardware: The protocol should allow the relying party to verify that it trusts the new hardware before allowing access.
- Non-Linkability: Credentials should be site-specific by default so that colluding relying parties can not link credentials to users across sites.
- Detecting Clones: The protocol should allow for the continued detection of potential authenticator clones by keeping a counter.

Threat Model

To determine whether the additional steps in the Transfer Access Protocol uphold these goals, we analyze each step using a threat model based on previous works done in this space. For example, we will use the attackers mentioned in Lang et al.:

- *Web Attackers* who can phish for credentials by setting up forged web pages, including correct TLS certificates for victim sites.
- *Related-Site* attackers where users may have reused the same credentials as the victim site.

---

[5] "Security Keys: Practical Cryptographic Second Factors for the Modern ...." http://fc16.ifca.ai/preproceedings/25_Lang.pdf Accessed 2 Jun. 2017.

- *Network Attackers* who can MITM connections with correct certificates or decrypt traffic.
- *Malware Attackers* who can install malicious applications or take over benign applications.

However, we will also consider other potential attackers through whom we can demonstrate some of the strong security and privacy properties of the FIDO authentication scheme. For example, *Site Attackers* who can dump logs and credentials from the victim site may be able to reveal user passwords, and adversaries who are able to gain physical control of devices at later or earlier times (OEM vs. repurchasing an old phone) can raise some interesting concerns. We also place certain attacks out-of-scope. For example, we do not consider protecting against a malicious FIDO application as the underlying FIDO scheme would not be secure anyway.

We believe that the addition of this work to the FIDO authentication scheme would help secure potential vulnerabilities that result when users transition to new devices.

2.3 Goal Conditions

Before diving into potential workable solutions for transferring access to a new authenticator, we discuss the assumptions and the properties constituting goal conditions for the Transfer Access Protocol. To start, we have the following assumptions:

Assumptions:

- The user has access to an old phone (A) and new phone (B)
- Phone A has keys and associated metadata, each associated with an account
- Phone B may or may not have existing keys
- Phone A and Phone B can create a “secure channel”
    - This secure channel is out of scope for the Transfer Access Protocol. We assume that this channel can only be set up by a legitimate user who explicitly allows the transfer of access from Phone A to Phone B. For the purposes of this paper, we assume this channel allows communication between the two phones that is resilient to all possible attacks, including eavesdropping and Man-In-The-Middle attacks.

Target Goal Conditions (for each transferred account):

At the conclusion of this protocol, we expect the following properties to hold:

- Phone A has deleted the “transferred key”.
- Phone B has the “transferred key”
- Phone B is logged in to the relying party.
- *The relying party removes Phone A’s access*
- *The relying party adds access for Phone B so that it will be able to authenticate in the future using standard FIDO authentication.*
- ***Security Goals***
  Throughout each step of the procedure, we expect the Transfer Access protocol to give attackers no advantage in attacking the FIDO authentication scheme.

3. Solutions

The goal of this work is to transfer access from an old phone, Phone A, to a new phone, Phone B, while preserving the usability, security and privacy properties of the existing FIDO authentication scheme. The solution in the current system would require the user to log-in to each site with Phone A, register a new set of keys for Phone B, remove Phone A's access at the relying party, and delete keys on Phone A (or factory reset the phone). Clearly, this adds multiple steps for the user for *each* existing account, but worse, the user may not have any indication as to how many or which sites require new credentials.

3.1 Simple Solution: Copying Keys

A straightforward solution that simplifies the experience of moving to new devices and eases user burden could merely copy the authenticator data from the old phone to the new phone. However, this violates the security properties of the FIDO protocol in that the relying party (example.com in the example above) would not have a chance to verify the new hardware. Further, if keys are stored in a secure element or trusted execution environment, the OS may not be able to copy them at all. If the OS could copy credentials, it stands to reason that malware could potentially extract keys as well.

3.2 Chain of Trust

As such, we propose a system that utilizes a secure channel between two phones (tap & go, for example, establishes a secure wireless channel between the new and old phones) to sign a new set of credentials with the old trusted credentials. This creates a chain of trust since the relying party already trusts the old private key. Now Phone A can inform Phone B which accounts the user would like to transfer over the secure channel, at which point Phone B can generate fresh key pairs for each of the sites. This requires Phone A to also send metadata uniquely identifying each key so that when Phone B sends back its new public keys Phone A knows which of its private keys to use to create signatures. Such a signature scheme solves a number of the problems above with the current and simple solutions. Namely, it can be done on initial setup without requiring a user to visit every site and it does not require copying credentials - a poor security practice for private keys. Such a scheme requires two steps. In Stage 1, Phone A and Phone B communicate to exchange necessary information and generate the required signatures. In Stage 2, Phone B negotiates with the relying party to provide assertions that verify trust in the new credentials. For example, in Stage 2, Phone B should send its hardware attestation certificate (and an accompanying signature) so that relying parties can verify that they trust the new hardware. Figure 4 shows the two-stage nature of the Transfer Access Protocol. However, a signature that simply delegates access from Phone A's public key to Phone B's new public key (even while providing hardware attestations for the new phone) still sacrifices a number of security properties provided by the existing FIDO protocol. In the following sections, we discuss how to mitigate these problems.

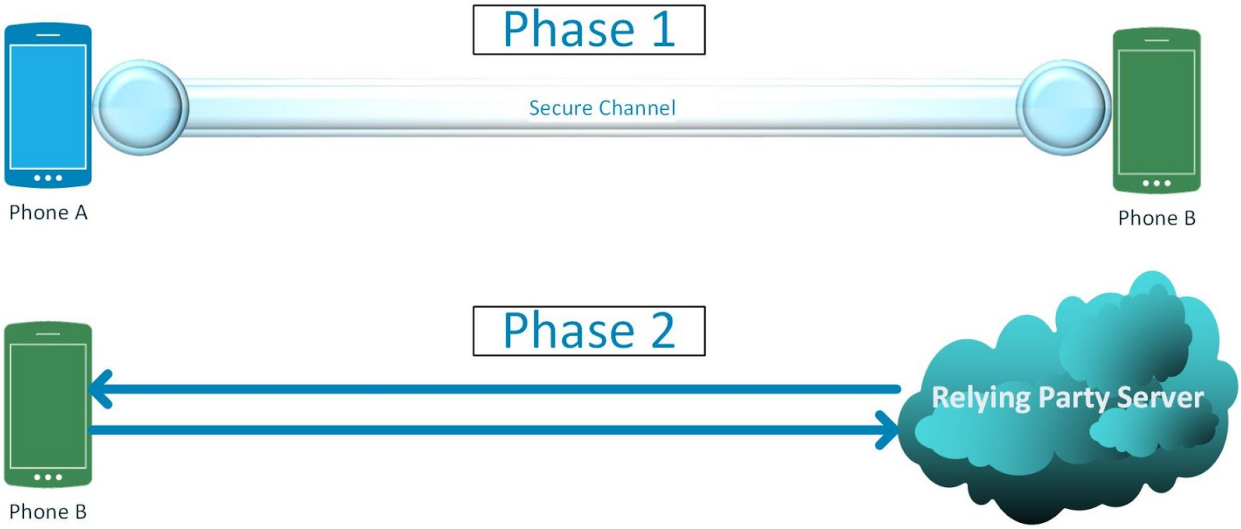


Figure 4: The Transfer Access Protocol requires two stages: In Phase 1, The old phone (Phone A) communicates with the new phone (Phone B) over a secure channel. In Phase 2, Phone B takes the results of that communication and delivers them to the Relying Party server.

3.3 Components of a Transfer Access Message

In the FIDO scheme, relying party servers communicate with authenticators through the browser. The browser can instruct the FIDO authenticator to respond with one of two messages: 1) A registration response or 2) an authentication response. We would like to take the necessary properties from each of these messages to craft a third response, enabling Transfer Access.

Because we do not want to change the user experience, the user's next log-in should serve as both a registration/enrollment for Phone B's new credentials and an authentication. We briefly discuss the registration and authentication messages crafted by the FIDO authenticator, with a focus on the security properties provided by each component of each message.

Registration/Enrollment

During the registration, the relying party (through the browser) asks the authenticator to create a new asymmetric key pair and associate that pair with the relying party. That request contains a challenge parameter which the authenticator can use during the creation of its response. The authenticator creates a certificate in response and sends it to the server so that the relying party can store the necessary credentials for future authentications. The current components of a FIDO registration sent by an authenticator are:

- *Message Header* - Allows for setting flags that can indicate message type, for example Enrollment, Authentication, or Transfer Access.
- *Metadata* - Allows the client and server to efficiently look up keys and binds each key to a specific account.
- *User Public Key* - This is the new public key to be enrolled.

- *Attestation Certificate* - Allows the server to decide whether it trusts the hardware. Attestations are batched by device, each device containing a certificate, public key, and matching private key.
- *Challenge* - Contains a nonce to make each registration unique so that it can not be reused. For example, this prevents an attacker from re-registering a previously registered and removed key - for example, after a user realizes a key is compromised and removes it from an account.
- *Signature* - Proves ownership of the attestation private key so that the server knows the device matches the above *Attestation Certificate.* This prevents an untrusted device from falsely providing the *Attestation Certificate* of a trusted device in order to enroll a new private key.

Authentication

When the relying party would like to authenticate an already-registered authenticator, it crafts a request containing a challenge and some key metadata for a previously registered key. The authenticator uses this information to look up the corresponding credentials and craft an authentication response that can convince the relying party to authorize the user. The current components of a FIDO authentication sent by an authenticator are:

- *Test of User Presence* - Requires the user to authorize the authentication, preventing attacks relying on remote surreptitious activation of the authenticator.
- *Counter* - Allows for clone detection. In the case of a cloned authenticator, the server will see consecutive log-ins that don't increment the counter correctly.
- *Metadata* - Binds the credential to the relying party, allowing the authenticator to efficiently look up the key and preventing attacks which seek to determine if some other key is present on the authenticator.
- *Challenge* - Contains a nonce to make each log-in unique, preventing replay and phishing attacks.
- *Signature* - Proves ownership of the private key, the basis for authentication.

Transfer Access

We would like to preserve each of the security protections afforded by the components of the existing registration and authentication messages. To this end, the Transfer Access Protocol should include the following in response to an authentication request from the relying party:

- *Message Header* - We suggest using one of the available bits to inform the server that the message is a Transfer Access Message.
- *Metadata* - Allows the client and server to efficiently look up keys and binds each key to a specific account.
- *New Public Key* - The new public key to be enrolled by Phone B.
- *Challenge* - This makes each registration unique, preventing replay and phishing attacks.
- *New Attestation Certificate* - Allows the relying party to determine whether it trusts the new hardware.
- *Counter* - Notifies the relying party in the case of a cloned authenticator. Given that this is the first log-in on the new device, we don't think it necessary to continue incrementing the

old authenticator. As such, we set this counter to zero, which will alert the relying party if there is a clone in future log-in attempts.

- *Signature (Authentication)* - Proves ownership of an authorized *private key* so that the user can automatically log-in after completing the Transfer Access Protocol. Recall that this Transfer Access Response gets sent in response to an authentication request, so the user expects to log-in.
- *Signature (Attestation)* - Proves ownership of the new *attestation private key*, so that the server knows the credentials have been created by a device with a matching *Attestation Certificate*. This prevents an untrusted device from falsely providing the *Attestation Certificate* of a trusted device in order to enroll a new private key.

Notably absent is the *Test of User Presence.* Recall that this field prevents attacks relying on remote surreptitious activation of the authenticator. Because we assume that setting up a secure channel requires user authorization and that the user intends to move from Phone A to Phone B permanently, we deem the *Test of User Presence* unnecessary for the Transfer Access Protocol. However, one could easily add it to the protocol when the authenticator delivers the Transfer Access response containing the above fields to the server, verifying user presence for that session.

3.4 Creating a Chain Through Multiple Devices

In Section 3.2, we discuss the two-stage nature of the proposed protocol. Although the user experience won't change for sites which the user visits regularly, the user needs to visit and log-in to each relying party with transferred credentials in order to complete the Transfer Access Protocol for each of those credentials. Though this may be reasonable for most sites, it is feasible that users will transfer to yet another new phone (say Phone C) before logging in to less oft-used sites on Phone B. In this case, we would have a situation where Phone B tries to transfer access to Phone C without first registering its credentials with the server. When Phone C finally does visit the relying party and delivers the Transfer Access credential generated by Phone B, the server will not recognize those credentials and will reject the transfer. To solve this problem we propose a protocol that allows for chaining of Transfer Access credentials. Like the original Transfer Access Protocol, a chain delivered to the relying party would require:

- Storing an Identifier for the Original Key
- Final new Public Key
- Metadata for New Public Key
- Final new Attestation Certificate
- Challenge
- Counter
- Signature proving possession of the new Authentication Private Key
- Signature proving possession of the new Attestation Private Key

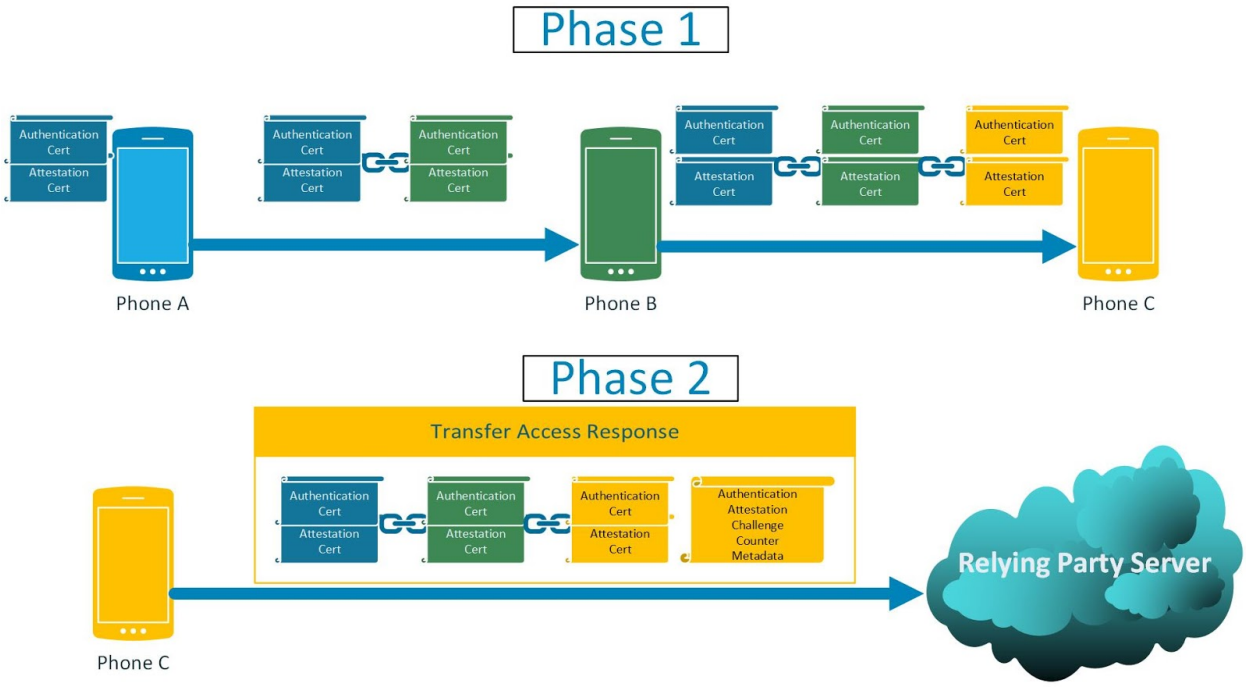


Figure 5: When chaining Transfers of Access, Phase 1 may need to include transfers through many devices (in this figure, Phone A transfers to B, which transfers to C before visiting the relying party. In Phase 2, the final device in the chain (Phone C) delivers the entire Transfer Access Chain to the relying party, along with signatures with its Attestation and Authentication Private Keys. It also includes the Challenge, Counter, and Key Metadata for this and future authentications.

Thus, when Phone B tries to transfer access to Phone C, it would simply add its relevant information to the Transfer Access credential given to it by Phone A, creating a chain. We can improve efficiency within this chain by storing and signing over only those items which the server needs. For example, the server does not need metadata, identifiers, or a counter for Phone B's keys so the chain should neither keep that information nor sign over it. Phone C, when it does eventually deliver the chain to the relying party, can add its counter, metadata for its key, the challenge from the authentication request, and signatures with its Attestation and Authentication Private Keys. With this information the relying party can check to make sure that it trusts the new hardware, the authenticator is not cloned, the response is unique to the authentication request provided, and the device owns the corresponding private key. Figure 5 shows how the chaining works, conceptually.

However, the relying party would also like to check the links in the chain. In the example above (Phone A→Phone B→Phone C), it needs to check that Phone B has a valid attestation certificate and has the matching attestation private key. It also needs to verify that Phone B has the corresponding authentication private key and agrees to transfer access to Phone C. Therefore, we need to store Phone B's Attestation Certificate and Authentication Public Key in order for the server to check those signatures. When Phone B crafts a transfer to Phone C, it will perform

signatures with the corresponding private keys over the included Attestation Certificate and Public Key for Phone C. Such an approach generalizes to a chain of many devices, as the intermediary authenticators can simply use their Authentication Private Keys to sign the next Public Key in the chain and their Attestation Private Keys to sign the next Attestation Certificate in the chain. By chaining the signatures in this manner, the relying party can trust the chain of authentication private key trust, and can trust that none of the devices have been impersonated because each phone signs the next phone's attestation certificate.

3.5 Other Challenges

Looking up the Transfer Access Credential in the Authenticator - When Phone B navigates to a relying party for the first time after receiving a Transfer Access credential, the server will look up the key metadata it knows for the user account and ask for authentication. Because the server does not know about any of the new credentials created by Phone B, Phone B needs to store, along with the Transfer Access credentials, an identifier for the original key from Phone A. In the case where the credential has been transferred through a chain of devices, the last phone in the chain needs to be able to look up the Transfer Access credential using metadata for the original key from Phone A, which started the chain.

Parsing the message - We mentioned previously that adding a bit in the Message Header to indicate a Transfer Access Response to an authentication request allows the server to easily differentiate between the two possible responses. We further aim to help the server parse the Transfer Access response by providing sequence numbers in the chain of Transfer Access credentials. By appending the existing chain to the *back* of the new Transfer Access credential (inserting it at the front of the chain), the server can immediately know how much space to allocate for storing the chain.

Simplicity - In Section 3.2, we mention that the relying party needs signatures with both the attestation and authentication private keys in order to verify the transfer of trust through the chain. However, we note that because the current FIDO implementation requires a signature using the attestation private key during registration of a new key pair, the relying party already trusts Phone A's attestation. Therefore, signing with Phone A's attestation private key during the first transfer of access from Phone A to Phone B does not add any security properties. However, in a longer chain, we need to ensure that if Phone B transfers to Phone C, and C to D, that both Phone B and in turn Phone C are forced to produce signatures using both the attestation and authentication private keys. For simplicity, we have chosen in our implementation to require the extra signature from Phone A using the already-trusted attestation private key so that the messages throughout the chain are formed using the same algorithm.

Ordering - We note that the chain of credentials that passes trust from old phones to new ones does not necessarily have to be in order. However, in situations where lots of messages are chained in the wrong order, the complexity of figuring out the correct order lies with the relying

party. Instead of forcing the relying party to try all combinations when the chain arrives out of order, we suggest that it simply discard the chain and fail to transfer the credentials.

Deleting Keys - We claim that after the completion of the Transfer Access Protocol, the old Phone, A should delete all transferred keys, but this raises some interesting tradeoffs. Deleting the keys as soon as possible helps protect a user who forgets to factory reset a phone before selling it to a potential attacker (requiring a 2nd factor can help mitigate attacks in this case as well). But in the case of a failure (say one phone runs out of battery during the transfer or the wireless environment becomes disturbed), transfers cannot be rerun. As such, to account for failures during transfer, we suggest waiting until the completion of Stage 1 (where Phone A receives acknowledgement from Phone B for each successfully transferred credential) to delete keys.

As a consequence of deleting keys upon the completion of Stage 1, however, we note that there are potentially times where a user can lose authenticator access. For example, if the user transfers access from Phone A to Phone B, but then loses Phone B before logging in to the relying party, the server will only know about Phone A, but those credentials will have been deleted. Recovery from this situation is a very interesting problem for which we plan to propose solutions in future work.

Furthermore, an attacker can prevent the delivery of the final ACK (acknowledgement from Phone B) so that Phone A will keep the keys. We propose mitigating the harms of this by alerting the user that the protocol has been interrupted, and allowing the user to then delete the keys manually or rerun the protocol.

In the Threat Model in Section 2.2, we discuss some extra attackers beyond the standard Web, Related-Site, Network, and Malware Attackers. Consider, for example, an attacker who obtains temporary access to an unlocked Phone A. This attacker could potentially perform a transfer of access from Phone A to an attacker controlled Phone B. If we did not delete the old keys from Phone A upon the completion of the protocol, the victim may not notice that credentials have been transferred. The attacker can then phish for the second factor and once obtained, can execute a transfer of access to gain access to an account. If the server does not delete access, the original owner may not ever be aware that an attacker has gained access. We suggest mitigating this by deleting keys on the authenticator and at the relying party so the next log-in will fail and the user will be aware of the problem. Further, we suggest designing the authenticator application in a way that allows users to see and manage stored keys. An authenticator app that requires local authentication to make changes would also help mitigate threats from this type of attack.

Log-In CSRF - We note another interesting attack where Phone A is the attacker's phone. Similar to the attack mentioned above, where an attacker gains temporary access, we can have a situation where an attacker gains temporary access to the user's Phone B and attempts to transfer credentials to it. The next time the victim goes to a site, it is possible they won't realize

they are logging in as the attacker, allowing an attacker to collect sensitive data and track activity. As above, requiring some kind of local authentication before using the authenticator application and allowing users to easily manage stored keys can help mitigate this threat.

System Level Malware - Though system level malware on either the old phone or new phone is a serious problem for a user even in the case where the user has a FIDO authenticator, we would like to minimize the effects of a compromise on future log-ins on other uncompromised devices. Assuming that the FIDO application is not compromised (a compromise of the FIDO application is out of scope for this work as it could break every aspect of the existing scheme and the proposed Transfer Access Protocol), the authenticator application cannot necessarily trust the OS to create a secure channel between phones. As a result, we suggest putting the crypto library, keys, and potentially some functionality of the authenticator application into a secure element or trusted execution environment.

4. Summary of the Transfer Access Protocol

In summary, we propose a two-stage Transfer Access Protocol.

4.1 Stage 1

Phone A and Phone B communicate over a shared secure channel. The specifics of such a channel are out of scope for this paper, but we assume that it does not add an attack surface for any in-scope attackers. Ideally, this phase would not impose extra work for the user, for example, it could be done during the initial phone setup when transferring apps and data from the old device. During this phase:

1. Phone A tells Phone B which credentials it would like to transfer. In practice this would be indicated by a unique identifier for each key that Phone B can understand. It should also attach a version number to ensure compatibility. In our implementations, we only accept one valid version number for simplicity.
2. Phone B sends its Attestation Certificate to Phone A. Phone B also generates new credentials for all the valid transferred key identifiers and sends the corresponding public keys back to Phone A. Phone B needs to mark each new public key with the original identifier so that Phone A knows which of its keys to use for signing.
3. Phone A generates a Transfer Access credential, and sends that back to Phone B. That credential may contain a chain, so Phone A must also send the original key metadata (the only one the server knows about) so that Phone B can look up the Transfer Access credential upon the next log-in. The Transfer Access Credential is a function of:
    - New Public Key
    - The Relying Party Site
    - New Attestation Certificate
    - Old authentication private key
    - Old attestation private key
4. Phone B acknowledges receipt of the Transfer Access credentials for each transferred key identifier so that Phone A can delete the corresponding credentials.

4.2 Stage 2

Phone B navigates to a relying party as normal over TLS. During this phase:

5. The relying party asks for the user account, which the user supplies. This can be done in the browser (for example by typing in a username, etc. and then having the user or authenticator select a key) or it could be done in the authenticator app, which would present credentials by account. The user could, for example, select “log-in with authenticator” and simply select from the accounts with matching domains within the authenticator. We leave the implementation of the authenticator app out of the scope of this paper.
6. Once the relying party knows which key it would like to ask for an authentication, it sends a standard authentication request containing:
    - Challenge
    - Metadata for the selected key
7. Instead of responding with a standard authentication response, Phone B responds with its stored Transfer Access credential chain.
8. The server parses all credentials in the chain and can decide whether to allow or deny access. If it decides to deny access it is up to the relying party whether it wishes to delete old keys or keep them. If it succeeds, it deletes access for Phone A, authorizes the authentication attempt, and adds the credentials for Phone B so that the user may log-in with a normal FIDO authentication in the future.

These changes would require subtle changes on both the relying party servers and authenticators to process the messages associated with the Transfer Access Protocol.

4.3 Summary of Proposed Changes

Here we summarize the concrete changes we propose in the Transfer Access Protocol.

- Relying Party Servers

  The server should be updated to handle both authentication and Transfer Access Responses to authentication requests during log-in. We suggest the following changes:
    - Activate a bit in the Message Header to differentiate between authentication and Transfer Access responses.
    - When processing the Transfer Access response, verify the chain of authentication key trust as well has hardware trust.

- Authenticator Clients
    - Allow authenticator to create, store, and send Transfer Access credential chains in addition to traditional FIDO authentication credentials and authentication responses.

- Store metadata for the original key so that the authenticator can look up Transfer Access credential chains when prompted by the key identifier known to the relying party.
- Expand the API to allow authenticator applications to talk directly to each other and perform the steps from Stage 1 in the Transfer Access Protocol.

4.4 Implementation

We have implemented the concepts described in this paper on top of the public FIDO-U2F protocol from Google (https://github.com/google/u2f-ref-code). Our changes are available for download from our fork (https://github.com/alextaka/u2f-ref-code). The server was implemented in Java; the client was implemented in software in javascript.

5. Future Directions

Device loss - The natural next step for this work is to perform Transfer of Access when the user no longer has access to Phone A. This is also likely a very common scenario. We have some proposals for this space including keeping a backup authenticator, or pre-registering backup keys with an authenticator and relying party. We plan to explore the security and usability tradeoffs of some of these solutions in future work.

Second Factor - There are a number of proposed second factors for the FIDO authentication scheme, ranging from iris scans, to fingerprints, to pins, to passwords. We hope to discuss the security and usability tradeoffs of many of these second factors, as well as how they should be best implemented in the framework of a FIDO authentication application.

Version Negotiation - For the purposes of our discussion and implementation, we assume that all clients are using the same version. If the version differs, we simply stop the protocol. This simplifies the decision making process during the protocol, however, we expect that the FIDO ecosystem will move to new versions in the future. The Transfer Access Protocol should be able to handle transfers to devices that require new versions.

References:

Czeskis, A., Dietz, M., Kohno, Y.. Wallach, D., and Balfanz, D. “Strengthening user authentication through opportunistic cryptographic identity assertions.” *ACM Conference on Computer and Communications Security* (2012).

Lang, J., Czeskis, A., Balfanz, D., Schilder, M., and Srinivas, S. “Security Keys: Practical Cryptographic Second Factors for the Modern Web.” *Financial Cryptography and Data Security* (2016).

"FIDO Alliance." *FIDO Alliance*. N.p., n.d. Web. 02 June 2017. <https://fidoalliance.org/>.

"Set up Your New Nexus Device - Nexus Help." *Google*. Google, 2017. Web. 12 June 2017.

"Android 5.0 Feature: Tap & Go Restore and Restore From Specific Devices." *Droid Life*. N.p., 25 Nov. 2014. Web. 15 June 2017.