The Identity in *Everyone's* Pocket

**KEEPING
USERS SECURE
THROUGH THEIR
SMARTPHONES**

BY PHIL VACHON

Most every technology practitioner has a smartphone of some sort. Around the world cellular connectivity is more ubiquitous than clean, running water. With their smartphones, owners can do their banking, interact with their local government, shop for day-to-day essentials, or simply keep in touch with their loved ones around the globe.

It's this ubiquity that introduces interesting security challenges and opportunities. Not even 10 years ago, a concept like biometric authentication was a novelty, reserved only for specialized applications in government and the financial services industry. Today you would be hard-pressed to find users who haven't had the experience of unlocking their phones with a fingerprint, or more recently by simply looking at the display. But there's more to the picture than meets the (camera's) eye: Deep beneath layers of glitzy user interfaces, there's a world of secure processors, hardware-backed key storage, and user-identity management that drives this deceptively simple capability.

Newer phones use these security features in many

different ways and combinations. As with any security technology, however, using a feature incorrectly can create a false sense of security. As such, many app developers and service providers today do not use any of the secure identity-management facilities that modern phones offer. For those of you who fall into this camp, this article is meant to leave you with ideas about how to bring a hardware-backed and biometrics-based concept of user identity into your ecosystem.

The goal is simple: Make it as hard as possible for attackers to steal credentials and use them at their leisure. Let's even make it difficult for users to clone their own credentials to share with other users. In addition to this protection, let's ensure that adding extra factors such as biometric authentication provides a stronger assurance of who the user is. Bringing keys and other secrets closer and closer to something that is physically attached to the user provides a stronger assurance of the identity of the user who just authenticated to the device.

WHAT IS A DIGITAL IDENTITY?

In the physical world, proving your identity might involve checking an identity document such as a passport, visa, or driver's license, and matching a photo or other *biometric* printed on that document. The value of forging these documents is quite high, so nation-states and other identity issuers go to great lengths to make this difficult. At the same time they must make it easy for a verifier to catch even the most sophisticated forgeries. There's a whole industry behind designing secure documents (for example, <https://www.jura.hu>), developing anti-forgery

technologies (<https://www.muehlbauer.de>), and producing these documents at scale. Of course, these efforts are not foolproof, and sometimes the most sensitive use cases warrant a closer inspection of the identity document, using “secret” security features embedded in the document itself.

In the realm of technology, an identity is proven through some sort of cryptographic scheme, the identity itself being embodied in a secret key held by the user. Simply possessing this secret, however, is often not enough: Like a physical identity document that doesn’t have a photo to identify the possessor, some cryptographic secrets can be stolen and used by anybody. For most use cases, the *policy* around how a secret is stored becomes critical. A private key stored on a laptop’s hard disk might not be as trustworthy as a private key stored in a smart card.

Consider, for example, the classic evil maid attack in which the attacker uses privileged access to a physical space (such as a private residence) to alter, steal, or simply use a device or credentials in a way that the owner would be unable to detect. Where you store a private key can make a difference. While an evil maid might be able simply to copy a private key stored on a laptop’s disk, he or she could not easily do so on a smart card, which is hard to clone and extract material from. The evil maid would not be able to walk away with a smart card without you noticing, and cloning the card in short order is a difficult task with most modern implementations. It’s also easier to keep a smart card on your person at all times, thus keeping it out of the clutches of the evil maid.

1

MOBILE PHONES AS SECURE KEYSTORES

Many years of development, hard-learned security engineering lessons, and practical experiences have led to extensive security capabilities in most modern smartphones. Before discussing the use of a mobile phone as an identity management device, let's define what this device looks like at a high level.

Figure 1 shows the idealized smartphone. Note the

FIGURE 1: THE IDEALIZED SMARTPHONE

division between the AP (application processor) and SP (secure processor), and how they control different aspects of the phone.

The parts of a smartphone are fairly simple:

- A display.
- A biometric sensor (facial recognition, fingerprint recognition).
- A “secure” processing environment, or SP.
[GlobalPlatform prefers the terminology *trusted execution environment*, or *TEE*. Architecturally, these are similar in concept, but the use of *SP* avoids confusion with terminology that is often used to refer to the Android-specific implementation]. The SP is where specialized security software runs, such as Apple’s SEPOS (Secure Enclave Processor operating system) or Qualcomm’s QTEE; all memory-containing program code and data associated with the SP environment is protected such that not even other CPUs on the same chip can access the SP data (more on this later).
- A secure storage environment, where secrets and other sensitive information for the SP are stored.
- The “application” processing environment, or AP, where apps and the phone’s operating system (such as iOS or Android) run. The AP can communicate with the SP only through a limited channel.

One assumption made here is that the device is normally in the user’s possession. Also, additional protections such as a PIN are assumed to be strong enough to protect the device from an adversary. While it’s interesting to think about the attacks a nation-state with unlimited resources could pull off, designing to such a high standard is not

always practical.

With this smartphone model you can start reasoning about how to construct a security system. Think of the SP and AP as two separate worlds on one phone. For the iPhone, Apple introduced the SEP. Most Android phones either have a completely separate chip (i.e., Google's Titan M chip in the Pixel 3 and later) or implement the SP as a TEE using TrustZone,⁹ an ARM-proprietary secure virtualized state of the application processor CPU.

The SP has a dedicated secure region of memory that is encrypted and usually authenticated.^{3,6} This encryption also protects the secure memory from attackers in physical possession of the phone, as well as preventing the AP from altering or recovering the SP's state. Without access to the keys used to encrypt memory, anyone would have a difficult time recovering the raw memory contents. This is a hardware-enforced control on the modern SoC (system on a chip) such as those from Apple or Qualcomm. Any breach of this control would be catastrophic, allowing the AP free access to any of the SP's sensitive data in memory. (A vulnerability in the SP's software would allow an attacker to gain access to the SP's memory in a way similar to what the secure hardware is trying to prevent. If this poses too large a risk for your application, you might want to think about other hardware secure tokens, such as YubiKeys, or even building your own.)

The SP also has access to its own set of peripherals (such as the fingerprint sensor or secure external devices for payment processing or secure data storage) that are inaccessible to the AP. Certain features of the SoC, such as cryptographic keys that give a smartphone its unique

identity, are also accessible only to hardware available exclusively to the SP. The keys to encrypt all the long-term storage for the secure processor are usually stored using this type of mechanism.

The persistent storage for the SP includes a number of important pieces of data, including secret keys generated by applications, biometric templates representing authorized users, and keys that uniquely identify the phone. In most implementations, these bits of data are cryptographically *wrapped* using the long-term storage keys, making them accessible only to the software running in the SP. The persistent data is then handed back to the AP for long-term storage in flash. This wrapping process keeps this information safe and ensures that no applications running on the AP would be able to pretend they are the SP. More importantly, wrapping prevents malicious parties from extracting secrets from the phone and cloning them (or worse, subtly corrupting these secrets).

The SP, compared with the AP, runs an extremely simple, minimal operating system. Typically third-party apps can't be installed in this environment, and the code that does run is purpose-built—just for the security applications required for the device. This is designed to minimize the exposed attack surface and reduce the probability of software vulnerabilities compromising the integrity of the SP. As you know, software, even in the SP, is never actually perfect.¹⁰

All communication between the SP and AP is highly regimented. By design, the AP cannot access the memory of the SP, but the SP might be able to see some of the AP's memory. All communication between the two worlds is

through RPCs (remote procedure calls), serializing all arguments and data to be passed from the insecure world to the secure world (or vice versa).

Operations defined with this RPC mechanism are usually quite high level. For example: “Generate key pair” generates a new key pair with parameters specified in the command; it then returns the public key and the key ID as a response. “Sign blob with key pair,” which takes the key ID and a pointer to a blob of data, then returns the signature as a response. These operations are inflexible, but that is by design: Flexibility introduces more ways things can go wrong.

Figure 2 shows the logical division between the SP and AP. Note how the SP has its own private encryption hardware and how that hardware is the only way to access key material generated during manufacturing. This protects the key even from software compromise. Most SPs do not have enough flash memory for storing all the keys needed but instead pass their data, encrypted using a key only accessible to the SP, to the AP for long-term storage. Secure memory protection prevents the AP from being able to “see” what is going on in the SP’s memory space, but lets the SP read and write the AP’s memory space.

Figure 3 shows the usage sequence for any key generated and held by the SP. Note how the AP can request use of the key only through specific RPCs and never accesses the private key itself.

One important lesson to take away is that there is a carefully choreographed dance occurring between hardware and software running on both the AP and SP

2

FIGURE 2: THE LOGICAL DIVISION BETWEEN THE SP AND AP

to implement the security features of a modern phone. One mistake, and the entire security model could be compromised.

The final piece of the puzzle to consider is where a phone's identity comes from. Establishing trust requires a bit of proof that the phone was manufactured to the expected security standard. Traceability back to a secure manufacturing process requires that the device have a cryptographic secret programmed into it during

3

FIGURE 3: THE USAGE SEQUENCE FOR KEYS GENERATED BY THE SP

manufacturing, which can be tracked to the manufacturer. A proof of identity signed with this manufacturing secret, combined with knowledge of the physical security of a device holding the key and the software policies around how and where a key you generated might be used, allows you to decide how trustworthy an identity you generated on a device really is.

BASIC IDENTITY MODEL

By now it should be clear that most recent phones have all the pieces required to create a digital identity. How can developers use those pieces to build up an identity that allows them to authenticate a user running an app on the phone to access a service that runs in the cloud or some on-premises infrastructure?

There is a common lifecycle for any identity you generate to support authenticating a user to your application. The basic steps, whether for a smartphone, stand-alone biometric token, or otherwise, are:

1. *Enrollment*. This kicks off the process to generate required keys.
2. *Attestation and delivery*. This verifies that keys are secure, safely stored, and difficult to extract and clone. If this succeeds, you can deliver some form of identity to the device for future use.
3. *Usage*. The keys are used, perhaps for mutual TLS (Transport Layer Security) authentication or some other out-of-band authentication protocol.
4. *Invalidation*. When something about the user or the phone changes in some way, the user identity keys should be erased, forcing the user to re-enroll.

As we look at the various techniques involved in making such an identity model work, we'll expand on what each of these steps means for your application.

KEY PAIRS

In practical applications, cryptographic identities are represented using asymmetric key pairs. While the details of asymmetric cryptography are well beyond

the scope of this article, the author recommends *An Introduction to Mathematical Cryptography* by Jeffrey Hoffstein, Jill Pipher, and J.H. Silverman [Springer 2008] for a deep dive into how cryptographic schemes work; *Practical Cryptography* by Niels Ferguson and Bruce Schneier [Wiley, 2003] is also a great, albeit slightly dated, reference.

Without going into great detail, asymmetric keys can be said to be composed of two parts: a private key that must be kept secret and can be used to generate cryptographic proofs (in the form of digital signatures), and a public key, which can be used by another party to verify these signatures. The private key must be used under the most controlled circumstances—using the most protected of hardware, ensuring nobody could capture the private key. In the smartphone model of figure 1, this means all operations performed with the private key are done in the SP's environment.

KEYS AND ATTESTATION

It is very difficult to consider the trustworthiness of a private key and the policy with which it is stored without having a way to verify these attributes. How do you gain confidence that the keys you're generating have been stored in an actual SP?

The process of key attestation uses another private key, usually the one installed during manufacturing, to form a proof. This proof is in the form of structured data that contains the public key you generated and the attributes of the key, including whether or not a biometric factor is required to access this key and other policies around its

usage. When this data along with a proof of validity for the key unique to the hardware, plus the detailed security policy of the device, are considered together, you have what is needed to decide whether or not to trust that a key is stored in secure hardware with the specified policy. Let's review the mechanisms for expressing this attestation.

Identity secrets and certificates

Any identity verifier will need a public key to perform some sort of verification. That key can be shared freely, unlike the private key that is held as a secret in the device's SP. Some platforms, such as Apple's SEP, allow the user to generate key pairs on the NIST P-256 elliptic curve. Conversely, Android's hardware-backed keystore implements the RSA [Rivest-Shamir-Adleman] scheme.²

A public key on its own is ambiguous and hard to know anything about; all a public key consists of is a large integer (or pair of integers, in the general case of elliptic curve cryptography) and no other unique metadata. To show the purpose of the key and to show any policy checks that were done as part of creating the key, there must be a container to carry metadata about the key, as well as a signature wrapping this container to allow a recipient to verify the authenticity of this information.

The most common form of this verification is an X.509 certificate, which is an ASN.1 [Abstract Syntax Notation One] DER [Distinguished Encoding Rules]-encoded object that contains many fields, including:

- When a certificate becomes valid to use.
- When the certificate is no longer valid.
- Who verified the authenticity of the party holding the

private key of the certificate.

- The public key itself.
- A signature from the trusted authority who validated the attributes of the key and created the certificate, showing the authenticity of the certificate's contents.

Usually an X.509 certificate is grouped with a set of authority certificates that represent who authenticated the contents of the certificate. This complete set of authority certificates, along with the end entity being authenticated, forms a *certificate chain*.

The key attestation process relies on the SP to perform a series of steps that result in an X.509 certificate chain that shows the provenance of the device back to some authority. An iPhone is tied to an authority run by Apple, while for Android this authority is Google. Figure 4 shows a sample key attestation X.509 certificate chain.

Of course, X.509 certificates are ubiquitous. Key attestation is only one limited use. An X.509 certificate chain can be used to identify a service or a user uniquely, as is frequently done for web applications. An X.509 certificate issued for a user would be joined with a private key stored in hardware. After verifying that a key is held in secure hardware and that the policy for that key matches expectations, you would then issue your own X.509 certificate to the user for the key you just attested. This means that you don't need to verify a key attestation every time you want to authenticate the user; it also lets other parties that trust you as the identity provider verify your user's identity.

4

FIGURE 4: A SAMPLE KEY ATTESTATION X.509 CERTIFICATE CHAIN

Biometric Factors

Most smartphones include a biometric factor: the bare minimum these days is a fingerprint sensor. Facial recognition is increasingly present in high-end devices, but in the era of COVID-19, where most people are wearing

masks, the convenience of facial recognition has been reduced. In fact, to maintain security guarantees Apple made passcode unlocks easier to perform.⁴

A phone must be personalized for a biometric factor to be usable by an application. This means a user must enroll at least one biometric factor through the system's mechanism. An app developer has to make the critical assumption that the user who personalizes the phone is the owner of the phone or an authorized user.

For many use cases, biometric factors exist as a convenience. Rather than typing a long passcode every time a user wants to unlock a device, the user can simply present a biometric factor to prove who he or she is. Most devices require that the user provide the passcode at least once every seven days, as well as after a reboot or other system events. Reducing the frequency of entering a long passcode means users are more likely to choose long, complex passcodes, improving overall device security.

Both iOS and Android make it possible to set a flag such that a key can be used only if the user has successfully performed a biometric authentication—providing that extra level of confidence and forcing the proof of possession of the biometric factor. (An incident in Malaysia shows the extremes to which car thieves were willing to go to steal a biometric factor to unlock a Mercedes S-class.⁸ Most fingerprint sensors today have some form of liveness detection to thwart this kind of grisly attack.)

Biometric authentication can ensure someone is not sharing a passcode among multiple users or has not had a passcode shoulder-surfed while unlocking the phone. Requiring users to prove who they are before

performing a cryptographic operation provides some assurance that they are at least physically present and authorized to use the device. The SP performs the entire biometric authentication process, ensuring that any operations involving biometric templates occur in a secure environment and that tampering with the templates isn't practical.

Most implementations have a secure channel between the SP and the biometric sensor. This makes stealing a biometric factor and replaying it later difficult to accomplish. Wrapping the measured biometric values in a secure, authenticated channel makes replay attacks, where the communication between the sensor and the SP is captured, impractical. This provides a stronger assurance that the sensor is physically present.

For evidence of why this secure channel is important, you do not have to look far. In 2018, researchers at Technische Universität Berlin demonstrated an attack where they recovered a latent fingerprint image from a physical card, then removed the fingerprint sensor and built a device to simulate the fingerprint sensor transferring that image to the host CPU.⁷ Since there were no security features to authenticate the communication between the fingerprint sensor and the CPU itself, the attackers were able to unlock the card without the original finger being present. This failure shows why it's important for a secure channel to exist between the sensor and the SP, including the ability to authenticate all communications between the two.

Finally, a policy decision is needed: Do you continue to trust an identity you generated if a user has added

new biometric enrollments on his or her phone? This could indicate some sort of compromise of the device. It could also indicate the user was having trouble with the biometric factor and tried to re-enroll, or perhaps the user's children added their own enrollment so they could more easily buy in-game currency. This is a security and usability tradeoff to consider. Both Android and iOS enable a policy that will delete a key if any biometric factors have been added.

Trust is a business decision

When a manufacturer imbues a smartphone with a cryptographic identity on the manufacturing line, it has made an assertion: This phone was manufactured in a trusted environment, to specified standards. At this point, businesses need to make a decision: Do they believe this standard is sufficient for their threat model?

They are deciding whether to trust the assessments that Google and Apple have made of their manufacturing processes. Proving the authenticity of a device is one of the major challenges facing developers today, but it's critical for them to complete the enrollment process and decide if they trust the device to hold on to the secret for normal use.

For modern Android phones, Google provides this assertion. Each Android phone that is built to the requirements Google has set out will receive an X.509 certificate chain, generated for a private key that is held by the secure hardware on the device. The end entity of this chain is a certificate specifically for this device. Thus, this certificate chain can be provided for outside parties so

they can verify the authenticity and trustworthiness of a particular smartphone.

This process is also an attestation—attesting the authenticity and uniqueness of a particular phone. It is worth noting that Google generates an attestation key that is shared among up to 10,000 devices, making it difficult to track users directly (more detail about this later).

By extension, generating another key and associated certificate, subordinate to the device identity certificate, will produce a key attestation certificate. Signing this key attestation certificate with the secret key held in secure hardware could support the claim that all the data held in the certificate is true and valid, assuming the integrity of the software and hardware in the SP has not been compromised. This becomes an authenticated, tamperproof way of transporting data about the state of the world as the SP sees it to the outside world. Short of stealing the private key for device identity, an attacker would have a hard time forging one of these key attestation certificates.

This is not a panacea. As discussed, software bugs might allow attackers to extract secret keys and use them to create seemingly valid, but forged, certificate chains. Hardware bugs could expose sensitive data from the SP to the AP or an outside attacker. Again, you must make a business decision: Is this software and hardware sufficiently well designed to trust with sensitive access credentials for your service? Do you trust Google as an authority to assess whether or not a phone is secure enough to store sensitive secrets used to identify your

users? Is Google storing the attestation keys securely enough to ensure they can't be stolen? One other risk to consider: Could the phone manufacturer have tampered with the software that runs in the SP? That would completely undermine the trust model.

Key attestation has another benefit: By proving a key is stored in secure hardware, you can also have some assurance that an attacker isn't simply emulating trusted hardware. As long as no party is able to extract the device attestation key from the phone, this assertion will hold true. Malware with enough sophistication to emulate cryptographic APIs isn't unheard of, and an attacker who can steal all keys generated by an app would allow that attacker to subvert any sort of trust model. As the secrets are held in secure hardware, even an altered version of your app wouldn't be able to steal these secrets, further protecting your users.

Unfortunately, while Apple implements these capabilities in its Secure Enclave, the ability to leverage this attestation is not yet broadly exposed to third-party app developers. This means that you need to take a leap of faith in the enrollment process for any sort of identity on an iPhone, since verifying that your keys are actually stored in secure hardware is impossible. The iOS 14 developer release introduces app attestation, but this functionality had not been enabled for developer experimentation as of this writing. The new API also does not expose the means to control whether or not a biometric factor is required to unlock the key, limiting its usefulness for many identity management applications.

Google has exposed these features to apps since

Android 7, though generally devices that shipped with Android 8 or later implement all the required capabilities. Since Apple has not revealed how it will ultimately expose key attestation and biometric identity functionality to third-party apps, let's explore how to use Android's features for key attestation and establishing an identity.

ESTABLISHING AN IDENTITY: ANDROID STYLE

On Google Android devices, a hardware-managed identity is created using the hardware-backed keystore, often run in the TEE, the Android implementation of the SP. An application can specify a number of characteristics for the key pair to be used:

- The asymmetric encryption scheme to be used (RSA, EC, etc.) and the size of the key.
- Whether or not the user needs to have a PIN code, biometric factor, or other requirements to be able to generate the key at all.
- Whether or not the user needs to present a biometric factor or simply have recently unlocked the phone (or have not done anything at all) to use the key.
- The purpose or usage of the key (whether it's supposed to be used for signing, encryption, etc.).
- An attestation challenge, a number-used-once (*nonce*) that was generated specifically for this enrollment attempt by your back-end application, to avoid replay attacks. This must *not* be shared between enrollment attempts and must be a cryptographically random string.

After the key is generated, the app can request an attestation for the key. This returns an X.509 certificate chain with the following members:

- The key attestation certificate, the end-entity certificate of this chain, containing the key just generated, the attestation challenge nonce, and the policy associated with the key. This is signed with the device attestation key. This type of certificate is issued to the device only if it has a hardware key store.
- An intermediate certificate, representing and attesting the device attestation key. This is shared, along with the private key, with 9,999 other devices.
- Another intermediate, which is associated with a batch, used to issue the device attestation key certificates.
- Google Hardware Attestation Root certification, representing the root of Android device identities. This is held by Google, hopefully in a very secure location.

These certificates are then passed to your back-end service for verification and to validate that the policies and metadata match expectations. Secure hardware is the only place that the device attestation key resides; this is the only way the end-entity certificate, the key attestation certificate, can be generated. If the certificate chain is rooted in the Google Hardware Attestation Root, then the key is stored in hardware that Google believes to be secure.

Recommendations and sample code on how to perform key attestation for Android are available on the Android developers website.¹

What about the other 9,999 phones with the same certificate?

Sharing the same key attestation certificate among 10,000 devices certainly seems counterproductive from

a security perspective. This means that attestations from any phones in that group are indistinguishable. How can you be assured that an identity is unique? Several mitigating factors lower the risk.

First, the keys used by the SP to wrap secure material are unique to each phone. Therefore, even if you found two phones with the same key attestation certificate, you would not be able to swap keys generated by one device with the other. This meets the requirement that identity keys must be hard to extract or clone. The key attestation certificate gives you assurance only that the hardware-backed keystore is holding on to your keys—it is not meant to be used as an identity on its own. As discussed earlier, you would want to generate your own X.509 certificate for the key held in secure hardware after verifying the attestation is accurate.

Second, each attestation should be tied to some other identity verification operation during the enrollment process. For example, when a user logs in to your app for the first time, your application would generate a unique challenge. There is a limited horizon for how long this challenge should be valid—likely on the order of tens of seconds. This means that replaying a key attestation from another device with the same key attestation certificate would be difficult; the challenge would limit how long such an attestation could be valid. Of course, once a user has successfully authenticated to your service, that challenge immediately becomes invalid, so even knowing this challenge would make it difficult for an attacker to exploit this property without a user knowing something is up.

With this level of care during the initial enrollment

stage, sharing key attestation certificates among devices should not pose a major threat to your service.

REVISITING THE IDENTITY MODEL

Let's revisit the identity model and fill in some details about how an implementation might work.

First, there is a core assumption that the user has personalized his or her phone by registering a fingerprint, facial recognition, or other biometric factor. Personalization is a prerequisite for any biometric factor to be usable.

Once the user has completed the personalization process, you need to generate an identity unique to your application in the user's secure hardware. Typically this is done the first time a user authenticates to your service, perhaps using an alternative second factor. Let's review this in the context of the lifecycle described previously:

1. Enrollment. After authenticating the user some other way [e.g., username, password, one-time password, challenge on screen as a QR code], you can generate a unique asymmetric key pair that will be used to authenticate the user in the future. The private key is stored by the SP, and the app developer needs to specify the parameters of the key, what is required for the user to unlock this key, and so forth.
2. Attestation and delivery. Verify that the parameters (usage policies, key lengths, etc.) around the secret meet your requirements, performing the final checks on your service back end. This is a comprehensive set of checks that gives your back-end application some assurances of: where the key is held; what the user must do before

the key can be used by a program (e.g., provide biometric proof); what the SP should do when the phone's parameters, such as fingerprint or facial recognition templates, change (e.g., invalidate key); and what type of biometric parameter can be used to unlock the key for use. If the attestation checks match expectations, you can issue an identity certificate chain to the user and store that on the device for future use.

3. Usage. An identity can then be used, based on the policy defined during enrollment and verified during attestation. This could be used along with a client certificate to verify identity when connecting to a back-end service—for mutual authentication of TLS sessions—or used to sign a cryptographic challenge that is provided out of band.
4. Invalidation. Some events can invalidate a user's identity—for example, changing the user's PIN, adding biometric templates, or other changes to policy that affect the security of the phone. These changes would mean there is no way to guarantee that whoever originally generated the identity is still in possession of the phone. The user must re-enroll in order to get out of this state. Return to step 1.

HOW TO USE A DIGITAL IDENTITY

Once the hard work of establishing an identity has been taken care of, the next step is to use that identity. Many use cases might simply be able to directly use the secret held in the trusted hardware as a part of authenticating to a service through mTLS (mutual Transport Layer Security) authentication. The benefits of mTLS are significant:

Requiring any communication with your back-end services to be authenticated using this secret means that no device without a valid, attested identity key pair will even be able to connect. Of course, this comes with a host of other challenges around certificate issuance and management that are outside of the scope here.

This certificate is an attestation of the validity of the generated identity, supplied by you as the service provider. This is done, of course, after the attestation of the validity of the key for this purpose from the phone manufacturer. Lots of trusted authorities are involved in this process.

In this case, you issue an X.509 certificate to authenticate a user through a PKI (public key infrastructure) you run yourself. The private key for the certificate is held exclusively in the phone's secure hardware and can only be acted on in the secure hardware. This means that you have a measure of assurance that users connecting to your service are who they say they are (so long as the integrity of the SP has not been compromised).

Alternatively, a bespoke protocol is an option. A simple challenge-response protocol—where the SP is tasked with signing a nonce—works, especially for legacy environments where OTPs (one-time passwords) have been implemented. Of course, the usual caveat around implementing any cryptographic protocols applies: Here be dragons. The challenges and risks inherent in building such protocols are too numerous to cover in this article, but suffice to say that if you are not aware of how wrong it can go, you should not be considering this approach.

PRIVACY CHALLENGES

With any sort of unique identifier that is tied to hardware, the question of user privacy is inevitable. Some vendors do not want to build devices that make it easier for advertisers, hackers, or nation-state adversaries to track users. Cryptographic identities have the advantage that they are very hard to forge—but this cuts both ways, and privacy advocates are rightfully concerned these identities could be used abusively to identify user behavior patterns.

The vendor as a threat

Remember that this whole security system is predicated on trusting the phone vendor. You are assuming that the vendor, in good faith, is not going to compromise the key storage and usage model such that malicious users can violate your security assumptions. Apple and Google both made different decisions on how to approach this trust model, and both models have tradeoffs. The major differences crop up during the key attestation process, which is critical for the enrollment phase.

Google chose to attest on the device. This means Google has no visibility into what you are attesting, or that you are performing a key attestation at all—all the secrets and attestation certificate generation is performed by the TEE on the phone. What it also means, however, is that the attestation key could be abused by malicious apps to track users, and thus this approach has privacy implications. Reusing this key across 10,000 other devices does make it harder to track a single user based on the attestation key alone. Of course, the value of this is limited in that

other factors about the device could be used to further disambiguate who you are.

Conversely, Apple has a centralized attestation authority. The most naive approach would involve Apple attesting each key in the SEP by asking its centralized authority to generate a certificate. The Secure Enclave encrypts a blob of data containing information about the attributes of the key, the public key itself, the app that requested the attestation, and unique identifiers for the phone. This encrypted blob is then handed to Apple's attestation authority service, which looks up the device, manufacturing details, whether or not it's been marked as stolen (through Find My iPhone), and similar device posture checks. If everything lines up, the service will return an X.509 certificate chain for the key in question.

The upside of this is that the attestation intermediate authority is tied to Apple, not to a secret stored in the phone, a huge benefit for user privacy. This means you're rooting your trust in Apple authenticating the phone, and you know the phone is real—but you don't know exactly which phone this is. You do not know if any attestations for different apps are from the same device, a huge privacy benefit. This naive approach could give Apple a lot of fine-grained information (beyond the scope of App Store telemetry) about how you're using your iPhone and what apps you're using. That's a huge responsibility to hold onto that information.

The details of key attestation for iOS remain to be explored, as Apple has just announced a subset of the functionality to be released in iOS 14. This is not helpful for many user identity use cases, since there is no ability to

require biometric factor verification before using the app attestation key.⁵

THINK OF THE USER!

Any identity management or security feature you add to your app must consider user experience. While factors such as biometric authentication make life easier for users, a poorly planned policy for your app can result in a disappointing user experience. The important part of all this is to make several judgment calls—including whether you need a biometric factor at all for your app to achieve its desired level of proof of user identity.

One common mistake is thinking that proof of identity is required every time a user performs an operation. Requiring biometric verification every time a key is used will have side effects for TLS connections, where users will constantly have to prove identity, multiple times over. This could get awkward as an app is backgrounded and its network activity times out. In these cases, it is better to unlock a key for a longer period of time, such as the duration of the session for which the user is likely going to use the app.

Enrollment is tricky—well-defined user flows are needed, especially once biometric authentication is integrated into the user's login process for your app. Users have to know exactly what they're doing at each stage, especially if there's an out-of-band challenge/response protocol involved in enrolling a user, such as a QR code on a webpage that contains a challenge for the app to prove the user logged in elsewhere. Also, make sure that feature and device state detection is well implemented

and fails rapidly, so users don't get into a process and have confusing failures. Some common states you'll need to check include:

- If you're using these features, is biometric authentication enabled, and are there enrollments that would allow the user to authenticate? Is there even a biometric sensor present at all?
- If you're relying on secure hardware in the phone, have you checked that the hardware is present, enabled, and capable of providing the features you require?
- Can you connect to the services that will perform the enrollment process?

An additional consideration is checking whether or not the enrollment was successful. A dry run at enrollment to ensure that the user knows how to use the biometric capability is always helpful.

Never scare your users. Failures and error messages should be honest and succinct but friendly. A message along the lines of, "Your device is not secure enough," is neither accurate nor appropriate. A vague message could scare and mislead a user. A message that is too technical will train users to ignore error messages, which could be even worse down the road. Messages that explain specific failures in a user-focused fashion are critical, so users can either self-help or know who to contact for support.

Remember that requiring a biometric factor for a key for which you're generating a short-lived certificate will require the user to present the biometric factor just to sign the certificate signing request. This can have an impact on the user experience during certificate renewal, since every time users renew a certificate, they'll have to present

that factor. It might be tempting just to issue a long-lived certificate and wash your hands of the matter—this isn't necessarily the wrong thing to do, but it might not fit with your security model.

Make sure that such tradeoffs are considered carefully when integrating identity into your system. A longer-lived identity certificate might make sense for an app that users are expected to interact with daily. A short-lived authorization might be preferred if the app is used infrequently, and an attestation only has to happen during these rare interactions.

Related articles

Hack for Hire

Investigating the emerging black market of retail email account hacking services

Ariana Mirian

<https://queue.acm.org/detail.cfm?id=3365458>

A Threat Analysis of RFID Passports

Do RFID passports make us vulnerable to identity theft?

Alan Ramos, et al.

<https://queue.acm.org/detail.cfm?id=1626175>

Rethinking Passwords

Our authentication system is lacking. Is improvement possible?

William Cheswick

<https://queue.acm.org/detail.cfm?id=2422416>

WHERE DOES THIS LEAVE US?

There's no easy answer when it comes to creating a usable, durable, and secure user identity. Mobile phones offer a compelling option, especially where the right features and capabilities are available, but these aren't consistent across the major smartphone platforms today. When building these systems you will always have to make tradeoffs, ranging from user experience challenges to limitations of the platforms themselves.

As you build such a system, you will find that the devil truly is in the details: How correct is your

service in validating the attestation certificate chain? How do you adjust your policy as mobile-phone technology changes (think Apple's migration to Face ID from Touch ID)? How do you handle various types of partially malformed attestation certificates? Do you want to trust Apple and Google with the crown jewels—how users access and authenticate to your service? Do you even have a choice if you want to leverage smartphones as identity devices?

Stealing a user's credentials with such a scheme is now difficult. Maliciously using the credentials is even more difficult if you require biometric authentication. To use the keys representing your user, the user would have to be prompted—how convenient is that?

Unfortunately, until Apple provides such capabilities as a part of iOS and makes these features available to apps in its App Store, we're going to be a long way off from making strong, hardware-backed identity ubiquitous. Google, on the other hand, has provided this capability for several years, allowing apps to take advantage of the attestation capabilities.

References

1. Android Developers. 2020. Verifying hardware-backed key pairs with key attestation; <https://developer.android.com/training/articles/security-key-attestation>.
2. Android Open Source Project. 2020. Hardware-backed keystore; <https://source.android.com/security/keystore>
3. Apple Inc. 2020. Apple platform security; https://manuals.info.apple.com/MANUALS/1000/IMA1902/en_US/apple-platform-security-guide.pdf.
4. Apple Inc. 2020. About iOS 13 updates: iOS 13.5; <https://>

- support.apple.com/en-us/HT210393#135.
5. Apple Inc. Establishing your app's integrity; https://developer.apple.com/documentation/devicecheck/establishing_your_app_s_integrity.
 6. Cai, L. 2019. Guard your data with the Qualcomm Snapdragon mobile platform. Qualcomm; <https://www.qualcomm.com/media/documents/files/guard-your-data-with-the-qualcomm-snapdragon-mobile-platform.pdf>.
 7. Fietkau, J., Starbug, Seifert, J.-P. 2018. Swipe your fingerprints! How biometric authentication simplifies payment, access and identity fraud. In *Proceedings of the 12th Usenix Workshop on Offensive Technologies*; <https://www.usenix.org/conference/woot18/presentation/fietkau>.
 8. Kent, J. 2005. Malaysia car thieves steal finger. BBC News; <http://news.bbc.co.uk/2/hi/asia-pacific/4396831.stm>.
 9. Ngabonziza, B., Martin, D., Bailey, A., Cho, H., Martin, S. 2016. TrustZone explained: architectural features and use cases. IEEE 2nd International Conference on Collaboration and Internet Computing, 445-451; <https://ieeexplore.ieee.org/document/7809736/definitions>.
 10. Ryan, K. 2019. Hardware-backed heist: extracting ECDSA keys from Qualcomm's TrustZone. NCC Group; <https://www.nccgroup.com/globalassets/our-research/us/whitepapers/2019/hardwarebackedhesit.pdf>.

Phil Vachon is the manager of the Security Analytics and Identity Architecture team in the office of the CTO at Bloomberg. He leads a team of engineers working on

problems related to network and infrastructure security, human and machine identity management, and data science as it applies to emerging threats and the behavior of malicious actors. Prior to his return to Bloomberg, Vachon co-founded a startup focused on high-speed packet capture and analysis, developed high-frequency trading systems, designed and implemented firmware for identity and security infrastructure devices, built synthetic aperture radar data-processing tools, and worked on data plane traffic engineering for carrier routers. His main interests are developing threat models that are relevant to business problems, designing secure embedded systems, and working to improve user privacy protection in the increasingly connected world.

Copyright © 2020 held by owner/author. Publication rights licensed to ACM.