Toward Decentralized Code Signing:

A Legal Framework for Ensuring Software Integrity

Sam Kelly, Sc.M. Brown University '17, <samuel_kelly@brown.edu>
DuroSoft Technical Report no. 1, rev 3. Providence, RI. September 2016

Introduction

The widely publicized early 2016 court battle between Apple Inc and the FBI has highlighted disturbing legal and practical limitations of what was once considered the *crowning jewel* of the software engineering industry—the secure signing of software source code. Code signing allows an organization to certify that it is the sole originator of a software distribution, and that this software has not been modified or tampered with by third party actors at any point while in transit or since time of signing. If the U.S. government has the capability to undermine the integrity of this practice by forcing an organization to sign an update against its will (and potentially in secret), even if this "update" is merely scoped towards a specific person or selector, such unchecked authority would shatter the trust-based contract that makes code signing attractive and valuable to consumers in the first place—for the reasonably paranoid, it is currently impossible to trust that software being downloaded from the internet has not been secretly tampered with by a potentially malicious third party.

In this paper we explore the legal and technological feasibility of decentralized code signing, whereby multiple *trust nodes* (distinct legal entities in different countries or legal jurisdictions, each possessing their own private signing key known only to them) must collectively sign off on a software update before it can be accepted by existing installations of the software. In particular, we weigh in on whether such a system would render an organization legally immune to National Security Letters (NSLs) or warrants like the one originally served to Apple by the Department of Justice, and we resolve some notable edge cases, such as what to do in the event that a signer is compromised or is otherwise no longer able to sign.

Background

A basic understanding of encryption technology and information security is important if we are to have a meaningful discussion about Apple v.s. FBI, or code signing in general, and such an understanding is absolutely necessary if our goal is to propose a decentralized replacement for this time-honored industry practice. In this section, we provide an overview of the encryption technologies behind code signing and the full-disk encryption used on Apple iPhones (and many other devices) so we can construct legal arguments regarding these technologies in later sections.

Full-Disk Device Encryption

Most modern smartphones and mobile devices, including the iPhone 5C used by 2015 San Bernardino killer Rizwan Farook [1] make use of full-disk encryption to protect against unauthorized access in the event that the phone is lost or stolen. This entails generating a cryptographically strong private key, and using this key with the 256-bit Advanced Encryption Standard (AES) to encrypt the entire hard drive of the iPhone or device in question, rendering its contents utterly indecipherable without the 256-bit private key. The private key is then stored on a secure chip on the phone from which it cannot be recovered without triggering failsafes that will automatically wipe the phone's hard drive, rendering all data on it unrecoverable [2]. In general, these failsafes makes it extremely difficult to recover the private key without authenticating. That said, the private key is linked with whatever unlock method the user selects, which could be a 4-digit PIN number, a strong, full text password, a lock screen pattern, or even the output from a fingerprint swiper. If an unauthorized user can get past this lock screen, then the private key is automatically recovered and full-disk encryption is bypassed.

In Farook's case, a simple 4-digit PIN was used. Such a PIN is incredibly trivial to brute-force—the FBI would simply need to have someone sit and try all 10,000 possible 4-digit PIN combinations, which would take a matter of hours. An additional failsafe is in place, however, to prevent this kind of abuse. The iPhone 5C will automatically wipe its hard drive in the event of ten consecutive incorrect authentication attempts [2]. Thus, if the FBI were to attempt the brute-force approach, they would only be able to try 10 of the possible 10,000 PIN combinations before the phone wipes itself, giving them a 0.1% chance of success with no ability to try again later if they are wrong.

Signed Software Updates

There is, however, one additional software layer that can penetrate the full-disk encryption on the iPhone 5C. Apple designed the 5C's software update subsystem around the assumption that signed system updates from Apple itself are secure and can be trusted (after all, this is the whole point of code signing). If Apple pushes out a signed update via their update servers, a locked 5C with internet connectivity will see this update and automatically begin installing it without the need for the user to authenticate. Since system-level software updates are able to interface with restricted areas of the phone, it would in theory be possible for Apple to push a signed update that disables the lock feature for phones configured with Farook's Apple ID. Alternatively, if Apple refused to push such an update, the FBI would merely need a copy of Apple's private key that it uses to sign software updates. With this key, the FBI could easily create spoofed updates that it can push to arbitrary Apple devices via man-in-the-middle attacks.

The Trust is Gone: Apple v.s. FBI

The 2016 Apple v.s. FBI court case serves as an excellent, and conveniently transparent example of code signing in action. More importantly, though, this case hints at what *could* have happened if these same events had occurred with a smaller, less litigiously endowed organization than Apple Inc. In this section we analyze the legal arguments and precedents brought up over the course of the case, and put forward evidence and analysis suggesting that we can no longer trust conventional (centralized) code-signing methods, in light of these events.

Initial Court Order

Originally, the attorneys representing the U.S. government demanded that Apple "assist" the FBI in executing a search warrant to search the contents of Farook's iPhone 5C [4]. Farook, having died in the attack, could not be compelled to unlock the phone, which is why the FBI involved Apple. The "assistance" the FBI was seeking from Apple is summarized in the following list of demands [5]:

1. Develop (using Apple engineers and resources) a fake update for the iPhone 5C that specifically targets Farook's Apple ID and device ID and disables the 10-time limit on authorization attempts.
2. Sign the "update" using Apple's code signing key so Farook's phone will accept it
3. Allow the FBI to introduce the "update" to Farook's phone in a controlled environment via a man-in-the-middle attack.

These demands, if executed, would have allowed the FBI to trivially brute-force the phone without the PIN attempt safeguard in place.

In the memorandum [4], the U.S. attorneys make the case that the All Writs Act (28 U.S.C. § 1651) imbues the court with the authority to make such a demand, and that asking Apple engineers to develop custom software designed to bypass security protocols for a single user meets the reasonability requirement of the All Writs Act, and doesn't put undue burden on Apple, Inc:

> *Pursuant to the All Writs Act, the Court has the power, 'in aid of a valid warrant, to order a third party to provide nonburdensome technical assistance to law enforcement officers. — Memorandum of Points and Authorities. ED 15-0451M.*

Proceedings

As a large tech company with ample legal resources at its disposal, Apple was able to muster some of the best lawyers in the country [6] to assist them with the case. In response to the original order for assistance, Apple made a motion to vacate the brief

and its supporting documents on the grounds that the government's demands were overly burdensome and ultimately unconstitutional [7].

In response, the government made one last-ditch effort to invoke the All Writs Act, claiming that no, "the burden placed on Apple is not undue and unreasonable," [8] and furthermore that speculation of third party harm does not establish an undue burden. They also invoked *United States v. New York Telephone Co.* as legal precedent for this kind of invocation of the All Writs Act.

In their final rebuttal, Apple reiterated their previous argument that the government's demands were overly burdensome and unconstitutional, and supplemented this argument with the assertion that the "government cannot invoke the All Writs Act here" and that "the invoked New York Telephone precedent does not authorize this order." [9]

Outcome

With an even stronger team of lawyers working Apple's side of the case, and public outcry unfolding in the media decrying the government's laughable invocation of the centuries-old All Writs Act, the FBI's hands were tied. The FBI promptly dropped their entire case against Apple, also making the surprising announcement that an independant contractor had found another way of breaking into the phone and that they no longer required Apple's assistance. As a result of the government dropping the case before a verdict was reached, civil liberties and cybersecurity advocates do not get to reap the benefit a victory for Apple would have spelled had the government stayed the course and lost. Likewise, the government did not win the case, so they did not get to set a legal precedent that would allow them to more easily compel other tech companies to circumvent their code signing protocols in the name of "national security" in the future.

Implications for Conventional Code Signing

While many were quick to celebrate the "victory" that was won for civil liberties and the software engineering industry when the FBI backed out of the case, that this was truly a victory is unfortunately an incredibly idealistic view. Apple has set itself up in the tech industry as a champion for user privacy, and as stated before, Apple has tremendous resources, both legal and financial, with which to fight the government's court order. Had this happened to any company other than one of the leaders of the tech industry, it is extremely likely the case would have been settled outside of court, and that the company in question would have simply complied with the government's order without challenge.

Even if we assume that any morally upstanding company *would* challenge the government order publically if given the chance (and this is a big if), the sobering fact remains that in 2015 alone there were 9,418 [10] National Security Letters issued to U.S. businesses. These NSLs all contained the usual gag order barring the recipient

from acknowledging the letter publically, meaning we still have no idea which companies have been forced to comply with government demands, and every company that was served with an NSL is legally barred from telling the public. Although the FBI is technically not supposed to ask for this kind of “assistance” in an NSL, there is technically nothing stopping the government from issuing secret NSLs to software companies requiring them to give up their private code signing key or forcing them to sign and push a malicious update. Given the sheer number of NSLs issued every year (for example, this number was over 24,000 in 2010 and 2008 [10]), it would not be unreasonable to assume that this abuse has already occurred in the past, and insiders have informed us that in the past such abuse was commonplace.

In other words, we can no longer trust the integrity of centralized code signing (since in principle any government could issue similar gag orders), but this is especially within the United States. What is needed instead is a *decentralized system above and beyond the influence of any particular legal jurisdiction or government.*

Decentralized Code Signing

Given recent events, it has become clear that the only way to truly restore trust in the code signing industry is to move away from the centralized model entirely, but what would a decentralized model look like, exactly?

In this section we propose a novel decentralized code signing system (which we have dubbed PeerSign). We then discuss the legal implications and the gray areas that would surround widespread adoption of such a system. Finally, we make a case for why we believe this form of decentralized code signing has the potential to eliminate the vast majority of situations where companies (wittingly or unwittingly) sign updates that are bad for user privacy, security, or civil liberties.

Naive Implementation

The seed of the idea behind PeerSign is simple: a centralized code signing system requires a cryptographic signature from a single, easily compromised node—the organization signing the software. Instead of this, why not create a system that requires multiple nodes, or rather, multiple signatures from multiple individuals or organizations in order for a software distribution to be considered signed? What if each signing node was a separate legal entity from the primary organization, meaning signing off on an update was technically (from a legal perspective) beyond the control or even influence of the primary organization? What if we also stipulate that each of these signing nodes has their own private key, which they never share with anyone? Would such a many-headed hydra be able to evade these sorts of government orders, and survive in this cruel world of NSLs cyber terrorists, and secret gag orders?

The short answer is no, not without additional stipulations. All the government has to do in this sort of situation to compel a software company to sign a malicious

update would be to issue orders to (or otherwise compel) *each* of the signing nodes, who in turn would be forced to comply with the government's demands and would collectively sign the update in question. As long as the individuals or organizations designated as required signers operate within a jurisdiction under the control or influence of the same government, then that government can compel all of the signing nodes to sign an update at the same time, just as it can compel any single node to sign an update in the currently popular centralized code signing system.

Transcending Jurisdictions

To overcome the issues outlined above, minimally, it is clear that we must have multiple code signers, and that at least some of the code signers within our system must operate outside the legal jurisdiction of the primary organization's native government. In fact, it is clear that we stand to gain heightened security by placing each signing node in a completely separate legal jurisdiction.

Ironically, we can enhance the benefit of causing our signing nodes to span multiple jurisdictions even further by *intentionally* placing these nodes in countries that are collectively at odds with one other, as this will greatly diminish the chances of international cooperation in forcing a code signer to sign an update against his/her will.

Amusingly, a perfect spread of jurisdictions would probably involve nodes in the U.S., Russia, China, Japan, India, Pakistan, Israel, Iran, and Saudi Arabia. Such a contentious spread of quarrelling countries would make it virtually impossible for *any* government agency or malicious third party to convince all governments involved to cooperate in an international effort to force all of these signing nodes to sign a malicious update.

By spreading out the collective jurisdiction of our code signers, and by requiring that our code signers are not direct employees under the primary organization, but rather mere "contractors" that are legally unaffiliated, we can ensure that no government could have any reasonable (or legal) expectation that a primary organization would even be *capable* of complying with an order to sign a malicious update, given that the mechanism by which updates are signed is ultimately beyond their control.

Who Are the Trusted Nodes?

At this point we know we need trusted nodes—whether they be individuals or entire organizations—spread out across multiple countries, each with their own signing key, where all signatures are required before a software update can be pushed out to the public, but how can an organization find such individuals or organizations to perform this code signing in the first place? How can an organization place trust in individuals spread out all over the world, and how should they be compensated?

First and foremost, the trusted nodes should be morally upstanding software engineers, or in the very least, they should be able to understand code. The primary

organization will have a contract with each trusted node instructing him/her to review the source code diff of newly proposed updates, and either approve or deny them based on whether the update causes a breach of user privacy or civil liberties. This way, when the primary organization sends out an update to its trusted nodes to be approved (or denied), each node will have the knowledge, experience, and expertise with which to identify subtle changes in source code that could be used to undermine the security or privacy guarantees of the underlying application. The contract will also stipulate that the trusted node is to ignore any and all additional instructions received from the primary organization that contradict the terms of the original contract. This will prevent the government from compelling a primary organization to tell a trusted node it has "changed its mind" and really just wants to push a particular update through regardless of the privacy implications.

For open-source projects, trusted nodes could be virtually anyone with sufficient expertise, background, and moral authority, and who lives in a desirably remote legal jurisdiction. For closed-source projects, trusted nodes should be compensated for their work, however this compensation seemingly creates a conflict of interest. For example, if a software company threatened to stop paying a trusted node if he/she does not sign a questionable update, this undermines the entire system. To avoid this, trusted nodes should be paid a fixed amount each time they sign *or refuse to sign* a proposed update. In this way, they have no monetary incentive to lean towards or away from signing.

Avoiding Bribes

Bribing is a very difficult to overcome potential problem with decentralized code signing. To avoid situations where a government (or even, a criminal organization or some other malicious third party) attempts to pay off all of the trusted nodes in order to force an update, primary organizations need to be very careful about who they select to be trusted nodes, and how much these trusted nodes are compensated. Realistically speaking, there are very few situations where a government would go to all the trouble of individually bribing each trusted node, and the chances of all nodes accepting such a bribe are low given the fact that these individuals are to be chosen for their high moral character.

Still, there are ways to deal with this situation. To begin with, trusted nodes should be paid an amount that is commensurate with how worried a particular organization is that their trusted nodes will be bribed. A small organization would likely only need to pay $5-$20 per code review, but larger, higher-stakes organization such as Apple or Microsoft would probably pay hundreds or potentially even thousands of dollars to ensure that their trusted nodes can in fact be trusted.

Another potential solution would be for primary organizations to implement a sort of bribe bounty program (think matching donations, but with bribes). Under such a program, trusted nodes would be awarded the full amount of a potential bribe (within reason) if said bribe is credibly reported to the primary organization and is rejected.

It is worth noting that at the end of the day, no existing code signing techniques have a better deterrent to bribes than what we have mentioned above. As a general rule of thumb, the more nodes you add to the system, the more you can collectively trust that all of them have not been bribed or compromised in some way. While it is easy to bribe one person successfully, it is much harder to bribe an *entire* crowd of people.

Handling Compromised or Incapacitated Nodes

It is always possible that a government or malicious entity will attempt to exploit the decentralized code signing system itself by compelling a trusted node *not* to sign a routine update, thereby holding the entire software company's ability to push out updates hostage. This same problem occurs if a trusted node decides to quit his/her job, or if for some reason we want to add or remove additional trusted nodes. How can we allow for these types of changes, but still avoid the bad effects of putting code signing solely in the hands of the primary organization?

A clever answer lies in the mathematics of code signing. Instead of requiring all n signatures to consider an update to be fully signed, we could instead merely require signatures from n-1 trusted nodes—or more, if there are a truly large number of trusted nodes. This is technologically possible by storing public keys for each possible n-1 permutation of the set of trusted nodes, instead of just a single public key containing *all* private keys. To verify an n-1 update, we simply try all the public keys. If any one of them is valid, then we know that at least n-1 trusted nodes approved the update, and we should consider it safe.

To allow for modifications to the set of trusted nodes, we could ask the trusted nodes to sign an update that alters the set of public keys currently being used to verify the integrity of updates. If a trusted node is added, the existing trusted nodes approve an update adding the new node's public key to the set In this way, an organization can continue to securely push out updates in the event that a trusted node dies or abruptly quits. By using n-2 or n-3 instead of n-1, an organization can be more confident that they will not be hamstringed in the event that more than one trusted nodes dies or quits at the same time, however this will also decrease the overall security of the system, as it now would take more than one trusted node objecting to be able to cause an update to be rejected.

Conclusion

A decentralized code signing system such as PeerSign, if created, would represent the first time in the history of software engineering that an organization could no longer be held legally responsible for their own inability (or tacitly, their refusal) to comply with an order (secret or otherwise) to push malicious updates to their existing software products. Widespread adoption of such a system would be a major victory for civil liberties and privacy advocates (as well as those wise enough to recognize its value

in the cybersecurity industry), as this adoption would lead to greatly increased trust in the integrity of software updates.

Works Cited

1. "FBI: Investigating CA Massacre as Act of Terrorism." *CNBC*. 04 Dec. 2015. Web. 11 May 2016.

2. Person, and AppleInsider. "In Depth Review: Apple's IPhone 5s Running IOS 7." *AppleInsider*. Web. 11 May 2016.

3. Lek, Kamol, and Naruemol Rajapakse. *Cryptography: Protocols, Design, and Applications*. Hauppauge, NY: Nova Science, 2012. Print.

4. "Government's Ex Parte Application for Order Compelling Apple Inc. to Assist Agents In Search; Memorandum of Points and Authorities." *United States District Court Proceedings*. Central District of California. ED No. 15-0451M.

5. "Order Compelling Apple, Inc. to Assist Agents in Search." *United States District Court Proceedings*. Central District of California. ED 15-0451M.

6. Leswing, Kif. "Meet Ted Olson, Apple's Insanely High-powered Lawyer in the FBI Case." *Business Insider*. Business Insider, Inc, 23 Feb. 2016. Web. 11 May 2016.

7. "Apple Inc's Motion to Vacate Order Compelling Apple Inc. to Assist Agents in Search, and Opposition to Government's Motion to Compel Assistance." *United States District Court Proceedings*. Central District of California. ED No. CM 16-10 (SP)

8. "Government's Reply in Support of Motion to Compel and Opposition to Apple Inc.'s Motion to Vacate Order." *United States District Court Proceedings*. Central District of California. ED No. CM 16-10 (SP)

9. "Apple Inc.'s Reply to Government's Opposition to Apple Inc.'s Motion to Vacate Order Compelling Apple Inc. to Assist Agents in Search." *United States District Court Proceedings*. Central District of California. ED No. CM 16-10 (SP)

10. "EPIC - Foreign Intelligence Surveillance Act Court Orders 1979-2015." *Electronic Privacy Information Center*. Web. 11 May 2016.