How to Use Sigstore without Sigstore

Yan-Cheng Chang

ycchang@alumni.harvard.edu

Abstract

Sigstore is a Linux Foundation project aiming to become the new standard for signing software artifacts. It consists of a free certificate authority called Fulcio, a tamper-resistant public log called Rekor, and an optional federated OIDC identity provider called Dex, where Rekor also acts as the timestamping service. Several command line interfaces (CLIs), written in different languages, are available to interact with it for signing software artifacts.

Ironically, we will show in this paper the design of Sigstore eliminates the need of Sigstore, i.e., the key components mentioned above are inessential. Specifically, we will first show how to remove the dependency on Fulcio from existing CLIs while keeping the CLIs work. Next, we will show how to remove the dependency on Rekor from the CLIs. Last, we will explain why relying on Dex, an optional black box with too much power, should be avoided.

As none of Fulcio, Rekor, and Dex is essential to making existing CLIs work, we conclude that they are unnecessary trusted third parties which the open source community should avoid employing. Instead, existing CLIs can be easily adapted to remove the dependency on them while providing the same functionality and user experience. The design of Sigstore is an example of solving a problem with a method which requires the solution as the input.

Keywords: Sigstore, Code Signing, Digital Signature, Applied Cryptography, Security

1 Introduction

Sigstore is an open source project which was launched in 2020 and just reached general availability a few months ago [1]. It is sponsored by the Linux Foundation and aims to become the new standard for signing software artifacts. The core of Sigstore is a free certificate authority (CA) named Fulcio, with other Sigstore components designed and built around it.

The certificates issued by Fulcio are ephemeral, that is, their validity can only last for a few minutes. Accordingly, signers can generate, use, and delete their keys on the fly during the signing process. With this approach, the complexity of key management becomes the past. And to make the validity of a signature last after the ephemeral certificate expires, the signature needs to be timestamped. For Sigstore, the timestamping is done by recording the signature in a tamper-resistant public log named Rekor, which is hosted by Sigstore and mainly serves as a transparency log for the signature data associated with Fulcio certificates.

Fulcio takes the following as inputs and issues public-key certificates accordingly [2]:

- A customizable OpenID Connect (OIDC) token with the Audience claim set to be "sigstore"
- A public key
- A challenge, in the form of a digital signature verifiable by the public key

An OIDC token resembles the concept of an identity card, in a standard JWT format, signed by an OIDC identity provider [3]. A customizable OIDC token, on the other hand, is an OIDC token whose Audience claim can be customized by the token requestor, where the Audience claim, per RFC 7519, is used to let the recipient of the token decide if the token is for the recipient to process. Moreover, the challenge is the digital signature of the Subject claim of the OIDC token. Fulcio will issue a certificate according to the public key and the Subject claim if the challenge can be correctly verified and the OIDC token is signed by an OIDC identity provider that Fulcio trusts. Currently, Fulcio trusts the following OIDC identity providers: Dex, GitHub, Google Cloud, AWS, and Azure, where Dex is a federated OIDC identity provider operated by Sigstore in a black-box manner [4].

For certificate requestors who can obtain customizable OIDC tokens by themselves, they can use Fulcio directly. Typical examples of such certificate requestors are GitHub Actions workflows and Google Cloud virtual machines (VMs), where GitHub Actions is the CI/CD service offered by GitHub. Specifically, when a workflow or a VM needs to authenticate with external services, it can request an OIDC token from its OIDC identity provider (i.e., GitHub or Google Cloud) and use the token to prove who it is. As part of the requesting process, it can customize the Audience claim.

For certificate requestors who cannot obtain customizable OIDC tokens, Sigstore lets Dex issue customizable OIDC tokens for them (with the limit that the Audience claim can only be "sigstore"). Typical examples of such certificate requestors are Gmail users. Specifically, when a Gmail user logs in to some website that supports Google Login, Google will issue an OIDC token, which is not customizable, and pass it to the website so that the website can learn who the Gmail user is. And Sigstore asks Gmail users to log in to Dex's website using Google Login so that Dex can transcribe the users' Google OIDC tokens into the users' Dex OIDC tokens.

On the other hand, there are several open source command line interfaces (CLIs) that can interact with Sigstore to help users sign and verify software artifacts:

- cosign, written in Go [5]
- sigstore-js, written in JavaScript [6]
- sigstore-python, written in Python [7]

The CLIs, when used to sign a software artifact, basically run the following procedure:

1) Generate a disposable key pair
2) Obtain a customizable OIDC token (e.g., for a workflow, VM, or Gmail user)
3) Request a public-key certificate from Fulcio using the key pair and the token
4) Digitally sign (the hash of) the software artifact using the private key
5) Timestamp the signature using Rekor
6) Output the certificate, the digital signature from step 4, and the timestamp data from step 5

Then the authenticity of a software artifact can be verified offline with the following proofs:

a) The output from step 6
b) The root certificate of Fulcio
c) The public key that Rekor used to sign the timestamp data

To sum up, in this section we give an overview of Sigstore's technologies. And in the next sections, we will explain why its key components, including Fulcio, Rekor, and Dex, are not essential to making the CLIs work, based on our key observation.

2 Key Observation

It is known that cryptographic designs are subject to small changes in definitions. For example, one-way permutations are very different from one-way functions despite of the resemblance [8]. Similarly, a customizable OIDC token is in fact very different from a normal OIDC token whose Audience claim is set by the OIDC identity provider. Specifically, unlike a normal OIDC token which just carries the identification of the token requestor, a customizable OIDC token not only carries the identification of the token requestor but the intention of the token requestor. To see why, note by setting the Audience claim to be "sigstore" the token requestor in fact shows its intention to request certificates from Fulcio, as anyone who holds such an OIDC token can unconditionally obtain public-key certificates from Fulcio by generating the key pairs and computing the challenges.

This explains why Fulcio cannot accept normal OIDC tokens as inputs, as otherwise a phishing website (or any website that supports Google Login) can easily obtain the public-key certificate that represents a Gmail user. This also explains why Google Login cannot issue an OIDC token with the Audience claim set to be "sigstore" without explicitly getting a user's consent through some formal consent page. After all, the OIDC tokens that Fulcio accepts are in fact very powerful. On the other hand, Google Cloud can issue such powerful OIDC tokens to VMs because by making API calls to request the customizable OIDC tokens (the programs within) the VMs have clearly showed their intentions.

Now that we know a token requestor can show its intention by customizing the Audience claim, we have the following:

Theorem 1. *By setting the Audience claim of a customizable OIDC token to be something like "sigstore?pk=PublicKey" the token requestor can show its intention to use PublicKey to represent itself, where PublicKey is the token requestor's public key in, for example, Base64 or Hex format.*

Proof. Since the Audience claim, per RFC 7519, is just used to let the recipient of the token decide if the token is for the recipient to process, the content of the claim is really up to the interpretation of the recipient. WLOG one can imagine there is an app whose name is "sigstore?pk=PublicKey" and the app, on input a valid OIDC token, will output "The token requestor wants to use PublicKey to represent it" if the Audience claim matches the app's name and will output "No" otherwise.

Corollary 1.1. *The PublicKey in Theorem 1 can be replaced by PublicKeyHash to shorten the claim when the length of the public key is long, where PublicKeyHash is the hash of the public key. The public key itself, in this case, needs to be provided along with the OIDC token for completeness.*

Accordingly, instead of exchanging an OIDC token (and a public key) for a public-key certificate, the OIDC token can be directly used as a proof of intention to use the public key. Since Fulcio has OIDC identity providers in its root of trust, this approach does not introduce any new trust. Instead, it makes Fulcio an unnecessary trusted third party since the proof of intention can be used to replace the certificate issued by Fulcio.

Finally, note this approach can be extended to any kind of intention, up to the interpretation of the recipient application. Therefore, we have the following:

Corollary 1.2. *Any OIDC token customized in the way similar to that described in Theorem 1 and Corollary 1.1 is a proof of intention digitally signed by the OIDC identity provider who witnesses the token requestor's intention encoded in the Audience claim.*

3 Removing the Dependency on Sigstore

Given the theorem in the previous section, we can easily remove the dependency on Sigstore from existing CLIs.

3.1 Removing Fulcio

To remove the dependency on Fulcio from the CLIs, we will adapt the procedure and the proofs mentioned in Section 1 directly. Below we will use strikethrough lines to denote deletions and use underlines to denote additions.

The procedure for the CLIs to sign a software artifact:

1) Generate a disposable key pair
2) Obtain a customizable OIDC token with Audience claim set to be “sigstore?pk=PublicKey” (e.g., for a workflow, VM, or Gmail user)
~~3) Request a public-key certificate from Fulcio using the key pair and the token~~
4) Digitally sign (the hash of) the software artifact using the private key
5) Timestamp the signature using Rekor
6) Output the ~~certificate~~ OIDC token, the digital signature from step 4, and the timestamp data from step 5

The proofs for the CLIs to verify the authenticity of a software artifact offline:

a) The output from step 6
b) ~~The root certificate of Fulcio~~ The public key that the OIDC identity provider used to sign the OIDC token
c) The public key that Rekor used to sign the timestamp data

Obviously, the adaptation not only reduces the trust base but accelerates the signing process. Also, since an OIDC token is as short-lived as a public-key certificate issued by Fulcio, the signature needs to be timestamped, too. Moreover, like the root certificate of Fulcio, the public key of the OIDC identity provider can also be hardcoded in the CLIs as root of trust.

3.2 Removing Rekor

As mentioned, Rekor mainly serves as a transparency log; however, after the removal of Fulcio, Rekor’s function becomes no more than a timestamping service. In other words, it can be replaced by any RFC 3161 Time Stamping Authority (TSA). Free TSAs that are publicly accessible include those provided by established CAs such as DigiCert [9] and Sectigo [10]. The timestamps made by such TSAs can normally last for at least 10 years. Given (1) the progress of quantum computing and (2) the fact that hardly anyone would use a package that was updated 10 years ago, we think such timestamps suffice. Accordingly, to remove the dependency on Rekor from the CLIs, we just need to adapt the step 5) of the above procedure and the item c) of the above proofs:

5) Timestamp the signature using ~~Rekor~~ DigiCert TSA

c) ~~The public key that Rekor used to sign the timestamp data~~ The root certificate of DigiCert TSA

Note if one does not want to trust a single TSA, several TSAs can be employed at the same time.

3.3 Removing Dex

If Dex can confirm a token requestor's intention to request one or more public-key certificates from Fulcio with its website (and issue an OIDC token with Audience claim set to be "sigstore"), it can certainly confirm a token requestor's intention to use a specific public key with its website (and issue an OIDC token with Audience claim set to be "sigstore?pk=PublicKey"), too. In other words, if we just treat Dex as one of the many OIDC identity providers which can issue customizable OIDC tokens, we don't need to deal with it at all.

However, Dex is not just one of them, but an overpowerful one. To see why, note Dex's website, in addition to Google Login, also supports GitHub Login. Since GitHub accounts can be created with any email addresses, we know Dex can issue OIDC tokens that represent any email addresses. Moreover, since Dex is operated in a black-box manner, we have to assume Dex can issue those OIDC tokens even without first seeing the OIDC tokens issued by Google or GitHub.

In the setting of the original Sigstore, this implies Dex can obtain Fulcio certificates that represent any email addresses and can use them to sign any software artifacts. And in the setting of our Fulcio-less Sigstore, Dex is as powerful, too. Thus, as we believe accepting such an overpowerful trusted third party poses a serious risk, we suggest the open source community not rely on Dex and look for better solutions instead. And this implies the removal of the support of Dex from the CLIs.

4 More Observations

As encouraged as other developers we were happy to see Sigstore emerged as a GPG alternative. However, the more we dig into it, the more questions we have. Below are our observations which were made before our key observation presented in Section 2:

1. Fulcio does not have a certificate policy (CP) [11]
2. Fulcio does not have a certification practice statement (CSP) [11]
3. There is no certificate revocation list (CRL) [12] for Fulcio's intermediate certificates
4. The usefulness of Fulcio's certificate transparency (CT) [13] is limited
5. The usefulness of Rekor's immutable log is limited

Regarding 1 and 2, since CP and CSP are basic requirements for any qualified CA in today's world, the lack of them makes the trustworthiness of the operation of Fulcio questionable. As for 3, though it is understandable Fulcio's leaf certificates, since ephemeral, do not need CRL (or OCSP [14]), the lack of CRL for Fulcio's intermediate certificates is unacceptable in practice and again makes the trustworthiness of the operation of Fulcio questionable. However, all the above can be fixed.

Regarding 4 and 5, first recall CT is a security standard for monitoring and auditing the issuance of digital certificates through recording all certificates issued by publicly trusted CAs in public logs: when anomalies are detected on CT, mistakenly or maliciously issued certificates can be revoked via CRL as a way to notify the relying parties. In the case of Fulcio, as mentioned, there is no CRL. Similarly, Rekor's immutable log can be understood as either "signature transparency (ST)" or "signed hash transparency (SHT)", and there is no implementation of "signature revocation list" or "signed hash revocation list" either. In other words, it is the job of a relying party to identify the anomalies on CT, ST, or SHT, but we are not aware of a feasible method as of today. What's worse is such a method can hardly be run in an offline fashion, which likely destroys the whole point of signature: offline verification (as otherwise software artifacts can just be checked against an online hash list). Accordingly, we claim Fulcio's CT and Rekor's immutable log are of limited usefulness.

5 Conclusion

In this paper, we successfully show the following:

- Fulcio is not essential to making existing Sigstore CLIs work
- Rekor is not essential to making existing Sigstore CLIs work
- Dex, though optional, is overpowerful

Accordingly, we conclude that Fulcio, Rekor, and Dex are unnecessary trusted third parties that the open source community should avoid employing. Instead, existing Sigstore CLIs can be easily adapted to remove the dependency on them while providing the same functionality and the same "keyless" user experience highlighted by Sigstore.

On the other hand, the design of Sigstore serves as an example of solving a problem with a method that requires the solution as the input, where the solution is customizable OIDC tokens. Unlike normal OIDC tokens which are known as identification tokens, customizable OIDC tokens can carry intention information.

Finally, this paper serves as a reminder, that cryptographic designs are subject to small changes in definitions, for future cryptographic protocol designers.

Acknowledgements

We would like to thank Tim Liu for proofreading this paper and providing suggestions. We would also like to thank Brian Chien and William Wang for their help.

References

1. Zachary Newman, John Speed Meyers, and Santiago Torres-Arias, "Sigstore: Software Signing for Everybody," in Proceedings of the 2022 ACM SIGSAC Conference on Computer and Communications Security, 2022, pages 2353–2367
2. "Certificate Issuing Overview," from https://docs.sigstore.dev/fulcio/certificate-issuing-overview/
3. "OpenID Connect," from https://openid.net/connect/
4. "OIDC Usage in Fulcio," from https://docs.sigstore.dev/fulcio/oidc-in-fulcio/
5. Available at https://github.com/sigstore/cosign
6. Available at https://github.com/sigstore/sigstore-js
7. Available at https://github.com/sigstore/sigstore-python
8. Yan-Cheng Chang, Chun-Yuan Hsiao, and Chi-Jen Lu, "The Impossibility of Basing One-Way Permutations on Central Cryptographic Primitives," in Journal of Cryptology 19(1), 2006, pages 97-114
9. "RFC3161 compliant Time Stamp Authority (TSA) server," from https://knowledge.digicert.com/generalinformation/INFO4231.html
10. "Timestamp Server And Stamping Protocols," from https://sectigo.com/resource-library/time-stamping-server
11. "Certificate policy," from https://en.wikipedia.org/wiki/Certificate_policy
12. "Certificate revocation list," from https://en.wikipedia.org/wiki/Certificate_revocation_list
13. "Certificate Transparency," from https://certificate.transparency.dev/
14. "Online Certificate Status Protocol," from https://en.wikipedia.org/wiki/Online_Certificate_Status_Protocol