https REALLY everywhere: AWS Private CA

7 May 2025 - 8 min. read

AWS Private CA *TLS*

The first time I really acknowledged the existence of TLS (it was still SSL back then) was in high school.

One day, we discovered that our school had installed a WiFi network throughout the building, and being the curious kid I was (and still am), I decided to "play" with it. At the time, even the most popular websites were still transitioning from HTTP-only to HTTPS, so after a few queries on Google, I discovered that by installing some sketchy APKs on my Samsung Galaxy S2, I could actually intercept other people's data. (For legal reasons, I have to say that I never actually did this).

Fast forward to college, and I studied SSL/TLS and its benefits. As stated at the beginning of RFC8446 (TLS 1.3), the secure channel created by the TLS protocol provides the following properties:

- **Authentication**: The client is sure of the server's identity. This prevents, for example, a malicious website from impersonating the real website we are trying to visit.
- **Confidentiality**: all information sent over the channel is visible only to the endpoints. An attacker who successfully intercepts the traffic will not be able to read it in clear text.
- **Integrity**: data sent over the channel cannot be modified without detection.

Pretty useful protocol, isn't it?

It's been a while since I was in high school, and now it's crazy to even think about sending login credentials or payment information over an HTTP connection. Modern browsers will even warn you if you just visit a site on HTTP.

Why TLS isn't everywhere

People are now used to seeing TLS on every web page and may be fooled into thinking that every HTTP connection in the world is now secure. Still, the guys working on the other side of the load balancer know that TLS is not even considered for most internal communications. So, should this protocol cease to exist before a proxy/load balancer? Well, in most cases, yes.

There are reasons for this:

- **Internal communications are generally considered secure**: the risk of traffic interception between components of a cloud infrastructure is considered very low, not because of the impact, but because of the low probability of it happening.

- **Management overhead & cost:** managing a public key infrastructure (PKI) is a full-time job. Managing CAs, certificate expiration, and, most importantly, securely managing private keys requires a lot of effort, which in an enterprise translates directly into money.

- **Performance**: while the latest version of TLS has very little performance overhead, it may still be too much for specific workloads.

As you might imagine, the first reason is the most important one. Perceived risk and attitudes toward it strongly influence every other aspect of the preliminary evaluation for adopting TLS on an internal network.

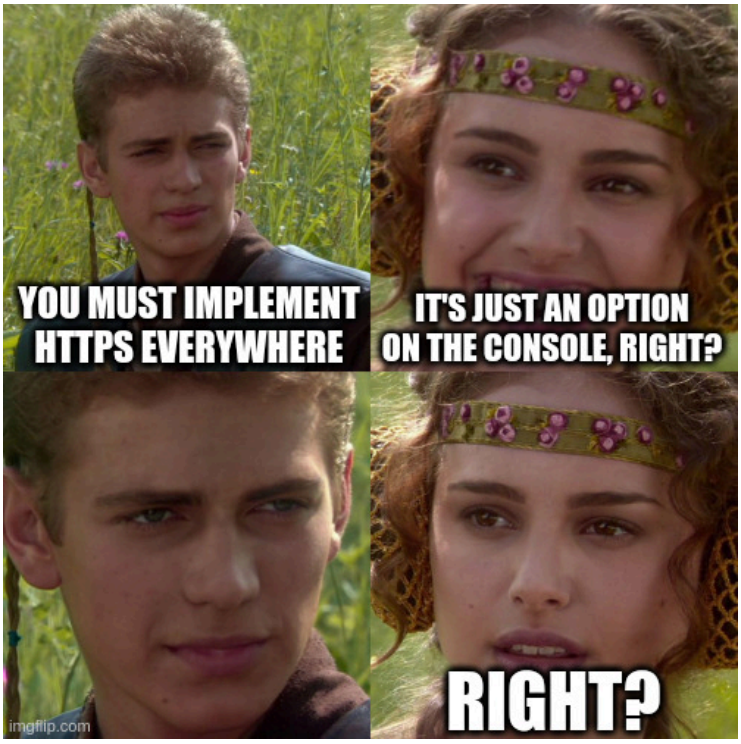
However, different organizations may assign different levels of severity to this risk during the business impact analysis. In most cases, the risk is perceived as low and therefore *accepted*. Still, for organizations that manage highly sensitive data, such as banks, insurance companies, or healthcare organizations, the impact could cause them to cease operations.

In these cases, using TLS, even for internal services, is a non-negotiable requirement.

Self-managed or AWS managed?

Now management decided that every connection must be secured, every microservice should communicate only on HTTPS and everyone who uses plain HTTP will be fired.

But how should it be implemented in an AWS environment?



There are two main ways to manage private certificates on AWS. Actually, there are many more, but most of them are a bit “hacky” and, therefore, not suitable for an enterprise organization.

Self-managed internal CA

There are many commercial and open source projects that can help build and manage a private CA. They provide a great way to manage the certificate lifecycle. Unfortunately, the main problem with managing certificates is keeping them secure. An internal CA is supposed to be more secure than a domain controller. TLS completely loses its value if certificates are not handled properly.

It's a thing sometimes overlooked; we're used to creating tons of public certificates through Let's Encrypt, which manages this complexity for us, but maintaining certificates requires a lot of attention.

The primary danger here is that if someone compromises your CA, they could generate valid certificates for any service in your organization, effectively breaking your entire security model. This is why using intermediate CAs (which can be revoked if compromised) rather than directly using the root CA is considered a best practice.

On the financial side, even though the certificates are completely free to generate, the time and care required to manage an internal CA and its certificates will likely make it an expensive choice. Remember always to include the time needed for maintenance during a cost analysis.

Finally, let's not try to hide it: it's also a liability issue. The need of managing such a mission-critical infrastructure component doesn't appeal to anyone...

AWS Private CA

AWS Private CA is an AWS service that allows you to create an entire CA hierarchy in a few clicks. The true power of this service lies not only in the ease of creating root and subordinate CAs from the AWS Console but also in the fact that AWS Private CA is deeply integrated with other services. All certificates are created through ACM (AWS Certificate Manager), which means that any service supported by ACM will automatically support the private certificates created by AWS Private CA.

Of course, since AWS Private CA issues end-entity X.509 certificates, these certificates can also be used for other areas besides TLS. For example, we recently used them with IAM Roles Anywhere for authentication. Also, if you use the API or AWS CLI to generate the certificates or export a private certificate from ACM, the certificates can be installed anywhere.

AWS Private CA solves most of the inherent criticalities of a self-managed PKI infrastructure:

- Security: certificate handling is managed by AWS, so there are no more worries about securing private keys. This also solves the liability issue discussed earlier.
- Setup time: (excluding architecture planning), it's a matter of a few clicks on the console or a few API calls.
- Maintenance time: ACM certificates signed by a private CA from AWS Private CA are eligible for managed renewal.

Unfortunately, all of these features come at a cost. Depending on the size of your organization, AWS Private CA pricing can be daunting or quite manageable: $400 per private CA per month for general-purpose mode. Keep in mind that a good CA architecture includes a root CA and multiple subordinates, so you will need to multiply this price for each CA created.

Certificates are not free either; the price is $0.75 for each certificate issued per month, dropping to $0.35 after the first 1000 and to $0.001 after 10000.

It's definitely not an economical choice, but for the value it brings to a business, it's affordable.

Here is a summary table of the comparison.

	Setup internal CA	AWS Private CA
Security	🔒🔒	🔒🔒🔒
Setup time	🕒🕒🕒	🕒
Maintenance time	🕒🕒	🕒
Cost	💰/💰💰 (OSS/commercial)	💰💰💰

How to implement it

Regardless of your choice, the implementation is very similar. The only difference is that certificates created by a self-managed CA need to be uploaded to ACM in order to be used by most AWS services. This will also require the certificates to be manually renewed once they are approaching expiration.

Services integrated with ACM

When a service is integrated with ACM, certificate configuration is trivial. Typically, the certificate ARN passed as a parameter during creation or update is all that is required for it to work.

The list of supported services includes most front-facing services, such as Amazon CloudFront, Amazon API Gateway, AWS Elastic Beanstalk, and Elastic Load Balancing, to name a few.

Other services

We will now briefly cover the most common compute services that may require the installation of a private CA to consume our secured services.

EC2

The installation of a private CA in an EC2 instance is of course depending on the operating system. There are plenty of tutorials on the web for each OS. We will use Ubuntu as an example, but the procedure is similar for every OS:

```
sudo apt-get install -y ca-certificates
sudo cp private-ca.crt /usr/local/share/ca-certificates
sudo update-ca-certificates
```

In summary the procedure requires copying the certificate in the OS certificate store. Pretty easy.

Lambda

Now it's time to get a little more creative. There are several ways to do this:

- **Install the private CA directly in the runtime**: this works the same way as with EC2. The CA certificate is added directly to the trust store, so it's recognized system-wide. Here is a (very unoptimized, but easy to read) example of a Lambda Custom Runtime with the certificate installed:

```
FROM public.ecr.aws/lambda/python:3.12
RUN dnf install -y ca-certificates
RUN update-ca-trust force-enable
COPY certs/* /etc/pki/ca-trust/source/anchors/
RUN update-ca-trust extract
...
```

- **Ship the certificate with your code:** another option is to include the public key directly in your repository and customize requests to the secured service to reference it. Here is an example using the Python *requests* library:

```
requests.post(url, data=data, verify='/path/to/public_key.pem')
```

- **Downloading the certificate at runtime**: an alternative to including the key in the repository. It's less efficient at runtime, but caching the file between function calls can make it a valid alternative.

ECS

The options here are the same as for Lambda, but this time, packing the certificate into the trust store is probably more appropriate since the effort to create a “custom

runtime" is now a requirement because of the Docker file.

Conclusions

Implementing TLS for internal AWS communications is becoming increasingly necessary, especially for organizations handling sensitive data. While AWS Private CA offers a streamlined solution with excellent service integration, its cost must be weighed against your security needs. Self-managed CAs are less expensive but require ongoing expertise and maintenance.

Whichever option you choose, extending TLS protection throughout your infrastructure adds crucial authentication, confidentiality, and integrity. The effort and cost may seem daunting initially, but the security benefits make it a worthwhile investment. Taking these preventative measures now helps avoid potentially serious consequences later.

Bonus

"What if I want to have HTTPS everywhere, but don't want the hassle of managing all that?"
Disclaimer: This solution is not for enterprises, maybe for small businesses or temporary solutions.

You can absolutely use public certificates (e.g., issued by Let's Encrypt) for private services, you just need to invoke the service with the correct domain name. You can do this using a private DNS or simply creating public records with your service's private address.

That's what I do in my homelab ;)

About Proud2beCloud

Proud2beCloud is a blog by beSharp, an Italian APN Premier Consulting Partner expert in designing, implementing, and managing complex Cloud infrastructures and advanced services on AWS. Before being writers, we are Cloud Experts working daily with AWS services since 2007. We are hungry readers, innovative builders, and gem-seekers. On Proud2beCloud, we regularly share our best AWS pro tips, configuration insights, in-depth news, tips&tricks, how-tos, and many other resources. Take part in the discussion!

Andrea Pusineri

DevOps Engineer @ beSharp. I love solving problems and I'm back belt of finding them. Linux enthusiast and security guy wannabe, I like to play CTFs, but in my spare time I'm an avid comic/manga/book reader. btw I use Arch

---