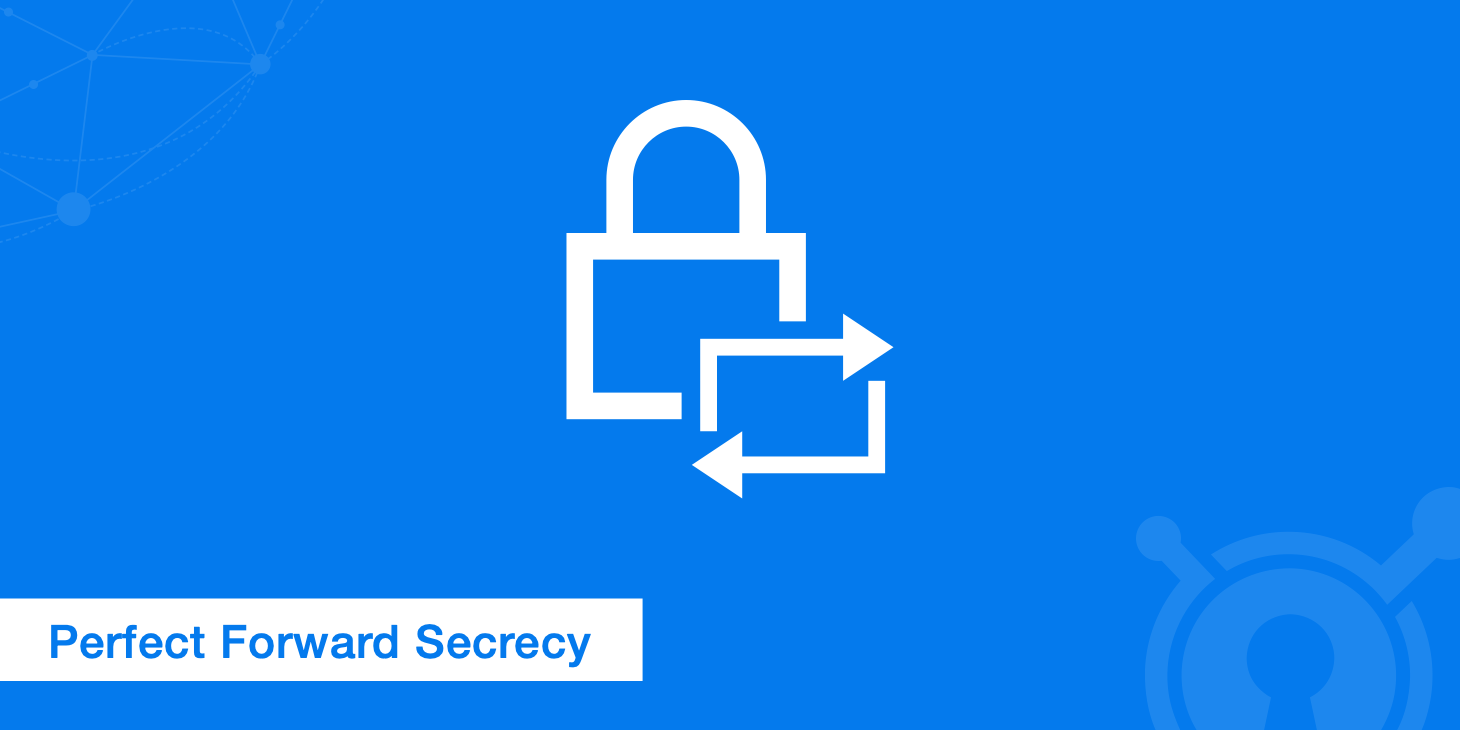
**Perfect Forward Secrecy - Why You Should Be Using It**

By Cody Arsenault

Published on September 14, 2017



Perfect forward secrecy, which is sometimes just referred to as forward secrecy, is a method of ensuring that all transactions sent over the Internet are secure. This method of encryption prevents a hacker from being able to access data from a group of transactions even if they're able to hack the encryption for a single communication sent over the web.

By creating a **unique session key for each transaction** instead of relying on sessions to keep connections open, hackers can't gain access to data from more than a single communication between a server and a user. The benefit is increased security for both the user and the organization running a server, but it's not a perfect system in spite of what its name implies.

If it's not set up or configured correctly, perfect forward secrecy won't work properly or may cause system issues. Further, it can lead to increased costs due to the additional processing power required to generate keys for each communication between a server and a user. In this article, we're going to explore when perfect

In this article, we're going to explore both the benefits and drawbacks of perfect forward secrecy.

**The history of internet transaction security**[**#**](https://www.keycdn.com/blog/perfect-forward-secrecy#the-history-of-internet-transaction-security)

To understand perfect forward secrecy, and how it is different from previous message encryption methods, it helps to have a basic understanding of how online transactions are encrypted as well as why security is so important.

Data transmitted over the Internet can easily be intercepted and downloaded, and much of that information is personal or sensitive. Login information, people's credit card numbers, and bank account information are regularly transmitted over the Internet, so it's essential that there is a way to keep this information protected. Additionally, most people aren't comfortable with having information that's not related to their finances easily available.

The solution to keeping data in Internet communications protected was to create encryption that ensures that the data in transactions can't be viewed by a third party not involved in the transaction. This generally works using an [SSH key pair](https://www.keycdn.com/support/create-ssh-key) (public and private keys). When a person attempts to log in to a website, their computer or device will use the website's public key to encrypt a session key. Another key is generated, which is a private key, and that key is used to **keep data sent between the site and the individual secure**.

In theory, the only way to decrypt the key and view data is to have both the public and the private key; having the public key isn't enough to decrypt the session key and see the data being transferred. While the use of two keys has been effective at keeping data protected, the problem is that **encryption keys can often be hacked or discovered with sufficient effort**.

Even the most secure encryption may eventually be able to be [brute force hacked](https://en.wikipedia.org/wiki/Brute-force_attack), which is a process where endless combinations of security keys are tried until the right one is discovered. This can be a very, very long process, but it can be effective given enough time. Up until the development of perfect forward secrecy, encryption keys were used for batches of transactions, or sessions.

What this means is that, when hackers are able to break an encryption key, it's not just the information from one transaction that can be seen; hackers can see data from **all of the transactions that occurred using that encryption key**. One particular transaction sent over the web may not have important data, but the problem is that hackers will frequently download enormous numbers of transactions on the basis that they'll eventually be able to read them.

**Heartbleed highlights the need for perfect forward secrecy**[**#**](https://www.keycdn.com/blog/perfect-forward-secrecy#heartbleed-highlights-the-need-for-perfect-forward-secrecy)

One of the best examples of the need for each online transaction to have its own encryption was the [Heartbleed bug](https://www.eff.org/deeplinks/2014/04/why-web-needs-perfect-forward-secrecy), which was discovered in 2012 and officially announced in 2014 along with the release of a patch that fixed the vulnerability. The bug didn't directly exploit sessions; instead, it **took advantage of a security flaw in servers running OpenSSL**.

The attack allowed hackers to download 64 KB of private memory on a server. Repeatedly running the attack allowed hackers to download a variety of sensitive data from these servers, including cookies, email addresses, and passwords. Additionally, session keys could also leak. The infographic below from [Creately](http://creately.com/blog/announcements/what-is-heart-bleed-bug/) shows how to Heartbleed bug worked in more detail.

While there would have been problems related to having session keys leaked irrespective of whether a server used perfect forward secrecy or not, a single session key from a server not using this type of security led to the potential release of far, far greater amounts of data.

**Benefits of perfect forward secrecy**[**#**](https://www.keycdn.com/blog/perfect-forward-secrecy#benefits-of-perfect-forward-secrecy)

The main reason for implementing perfect forward secrecy is that it provides far better protection due to the fact that it creates a unique session key for each transaction. Even supposing that a hacker is able to obtain a private key for a transaction, it will only allow them to access data related to that single exchange.

This reduces the amount of information that is vulnerable should a hack be successful. Additionally, this type of security makes it less likely that hackers will attempt to access secure data on the server. Even if they are able to crack the private key, the effort required may not be worth the minimal data they'll have access to. It doesn't mean that no one will attempt to gain unauthorized access to your servers, but it does make you a less attractive target.

**Utilizing perfect forward secrecy**[**#**](https://www.keycdn.com/blog/perfect-forward-secrecy#utilizing-perfect-forward-secrecy)

Implementing perfect forward secrecy is reasonably simple since it works on sites that use either [SSL or TLS](https://www.keycdn.com/support/what-is-ssl-tls). Both SSL and TLS are cryptographic protocols that allow secure connections to be created. Neither of which mandate the actual key exchange or determine the encryption cipher to be used.

Instead, the server and the user's machine must agree upon a type of encryption. To make sure that perfect forward secrecy is used, you simply need to set your servers up to make the suitable cipher suites available. Currently, key exchanges that are compliant are:

* Ephemeral Diffie-Hellman (DHE)
* Ephemeral [Elliptic Curve](https://www.keycdn.com/support/elliptic-curve-cryptography) Diffie-Hellman (ECDHE)

These key exchanges need to be set up to be ephemeral, meaning that the keys will only be used once, and after the transaction is complete, the encryption related to the exchange is deleted from the server. This is what ensures that obtaining a session key will be of little to no use to hackers.

**Enabling perfect forward secrecy on your server**[**#**](https://www.keycdn.com/blog/perfect-forward-secrecy#enabling-perfect-forward-secrecy-on-your-server)

If you check the security details of a site and see that it is using "ECDHE" or "DHE" then the server is **already using forward secrecy**. Any key exchange that uses ephemeral keys provides forward secrecy.

Most modern servers will already have this configured. However, in the event that you do not already have perfect forward secrecy configured, the examples below will demonstrate how to do so.

**Nginx configuration**[**#**](https://www.keycdn.com/blog/perfect-forward-secrecy#nginx-configuration)

1. First, locate your SSL protocol configuration (assuming /etc/nginx is the base directory for your Nginx installation):

grep -r ssl\_protocol /etc/nginx

1. Add the following to your configuration:
2. ssl\_protocols TLSv1.2 TLSv1.1 TLSv1;

ssl\_prefer\_server\_ciphers on;

1. Set the SSL cipher. There are a few options you can choose from here depending on how much compatibility for older browsers you require. The ciper below is currently recommended for modern setups that prefer better security.

ssl\_ciphers 'EECDH+AESGCM:EDH+AESGCM:AES256+EECDH:AES256+EDH';

1. Restart Nginx with:

sudo service nginx restart

**Apache configuration**[**#**](https://www.keycdn.com/blog/perfect-forward-secrecy#apache-configuration)

1. Locate your SSL protocol configuration on Apache with:

grep -i -r "SSLEngine" /etc/apache

1. Add the following to your configuration:
2. SSLProtocol all -SSLv2 -SSLv3

SSLHonorCipherOrder on

1. Set the SSL cipher:

ssl\_ciphers 'EECDH+AESGCM:EDH+AESGCM:AES256+EECDH:AES256+EDH';

1. Restart Apache:

apachectl -k restart

It's important to note that it's easy to set up perfect forward secrecy incorrectly, which can mean that even if it's enabled, it's not the type of security that your server is using. One common mistake is to simply enable support for either DHE or ECDHE.

For these security methods to work, they need to be **given priority by your server**. If another method of security is prioritized by the server, that's the one that will be used. It may be a good idea to disable all other types of security if you decide to use perfect forward secrecy.

The reason for this is that an SSL/TLS vulnerability, called the [FREAK Attack](https://tools.keycdn.com/freak), was discovered in 2015. It allowed hackers to enter secure servers by forcing the server to use a weaker form of encryption than the one that the system gave priority to.

It's also important that you ensure that you disable long duration session IDs or session tickets. These hold information related to a session for extended periods of time after a session has ended on the user's side. In some cases, session tickets can stay in a server's memory until the system is rebooted.

**Downsides of perfect forward secrecy**[**#**](https://www.keycdn.com/blog/perfect-forward-secrecy#downsides-of-perfect-forward-secrecy)

While they generally don't outweigh the benefits, like anything else, there are drawbacks to this type of security. To begin with, it usually requires greater processing power from a server since every transaction requires the generation of a unique encryption key. Ephemeral Elliptic Curve Diffie-Hellman is believed to be the faster of the two, but even it adds up to 20 percent greater SSL/TLS processing requirements to a server.

Another potential downside is that legacy systems, such as older servers and operating systems, aren't able to use perfect forward secrecy. All current browsers support it, and operating systems after Windows XP can use it, so a device has to be fairly old to not work with this type of encryption. Still, it is something to be aware of, and this is especially true if you haven't upgraded your servers in a while.

Finally, it's important to note that if you're using perfect forward secrecy, you're not just keeping hackers unaware of what's going on with communications on your server. You're also keeping network teams from [decrypting traffic](https://www.extrahop.com/company/blog/2017/what-is-perfect-forward-secrecy/). If your servers start to run into slowdowns or has other problems, not being able to decrypt traffic can make troubleshooting an issue significantly more difficult and time-consuming.

Aside from the fact that this can be annoying, server slowdowns, especially extended ones, can start affecting customers negatively as your application or website slows down or fails to work properly. One solution that has been used by some organizations is to put in an intermediary that requires data to travel through it when going from the server to the user. This still keeps data encrypted, but it also allows your network team to know what data is being sent. Of course, this requires additional configuration as well as more server space and processing power to implement.

**Summary**[**#**](https://www.keycdn.com/blog/perfect-forward-secrecy#summary)

According to [SSLLabs](https://www.ssllabs.com/ssl-pulse/), as of 2017, **31.4%** of sites support forward secrecy with modern browsers and **34%** of sites support forward secrecy with most browsers. The move towards stronger TLS encrypt makes this much more available to site owners who want to increase their security. If you're running a legacy system that isn't using forward secrecy yet, it may be time to consider the upgrade. Of course, you'll need to weigh the decision based on the amount of time it will take to upgrade, however, in the end, making the switch will make your own data more secure and reduce the risk of being hacked due to being a less attractive target and one that's harder to gain access to.

One additional thing to consider is the fact that it's almost certain that you'll be required to adopt this type of security at some point in the future. Google was one of the first to start using it, and Gmail and other Google products have been taking advantage of perfect forward secrecy for years.

The question is if you want to start the process now or wait until you absolutely must. If you deal with highly sensitive data or your users expect complete data protection, it's something you should implement sooner than later.