# What are SPHINX and DE-PAKE?

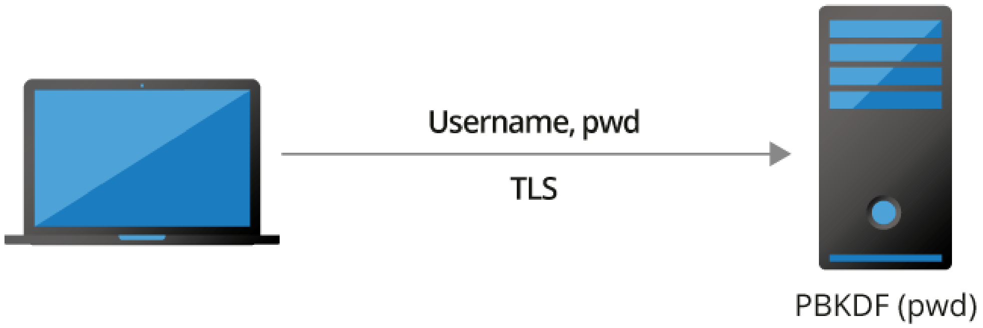
Devolutions

SPHINX SPECIFICATION IS BASED ON THE DE-PAKE SPECIFICATION FROM 2015 WHICH ELABORATES ON THE PKI-FREE PAKE PROTOCOL.

## Introduction

I recently came across the "SPHINX: A Password Store that Perfectly Hides from Itself" white paper that was released by IACR's ePrint repository in July 2018. To be honest, I was not "that" impressed with it at first. SPHINX uses a Password-To-Random (PTR) function on the client's device before sending a "random," but "encoded" password to the server over TLS. It was mainly designed to be compatible with existing authentication mechanisms using a browser extension and some service running on the device or using an online service. The interesting part of the the SPHINX specification is that it is based on the DE-PAKE specification from 2015 which elaborates on the PKI-free PAKE protocol.

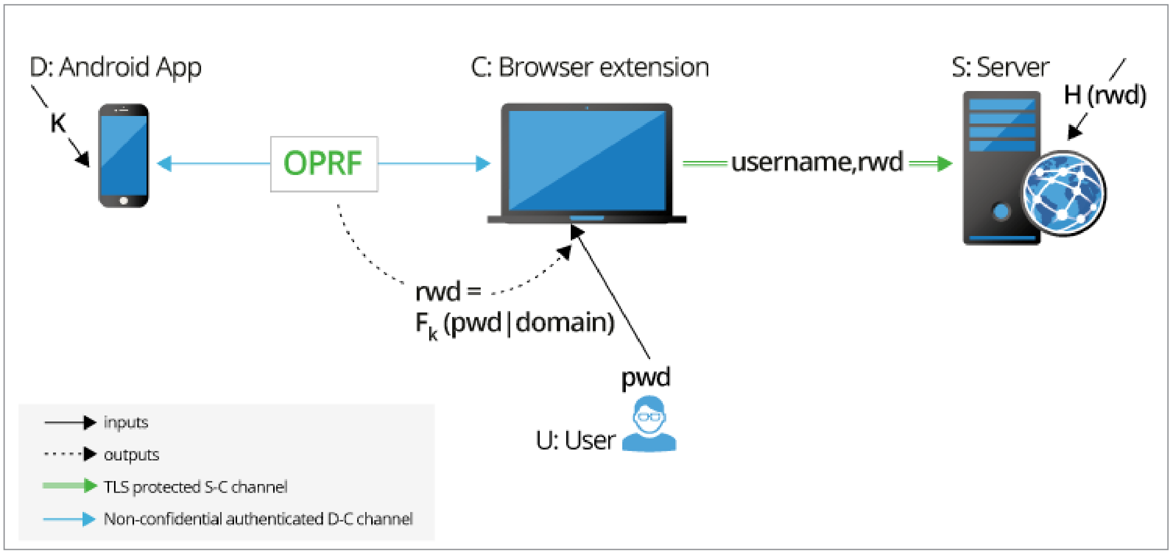
To better understand this technology, we have to look back at how we are doing authentication today. As shown in Image 1, servers usually receive your password directly over TLS and then use some key derivation function such as the "Password-Based Key Derivation Function version 2" (PBKDFv2) to store a derived version of your password. The goal behind this is the following; if the database is compromised, the attacker only has the derived secret which cannot be converted back to the original password. The only way for the attacker to recover the original password is to guess or brute force the password and compute PBKDFv2 to compare the result. While this may sound hard, passwords are generally based on well-known patterns and dictionary words which drastically simplify the recovering process.



## SPHINX is using the Device

The idea behind SPHINX is to use a PTR function on the client that uses a secret belonging to the device or an online service as pictured in Image 2. This solution ensures that the weak entropy password given by the user as input is derived from the device's secret key. The password is also salted with the domain name of the authentication portal to prevent phishing and provide uniqueness among several application to defeat problems associated with password re-use.

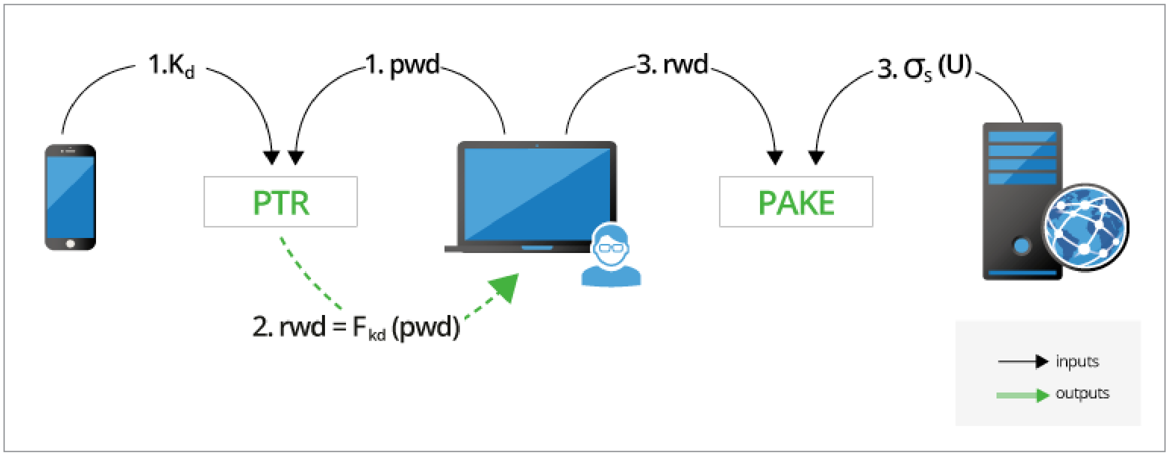
The server which receives the "random password" does not have any clue about the original password unless the device or online service key is leaked. At this stage you may feel the same way as me: "This is cool, but meh!" And you're right, the SPHINX paper is based on the DE-PAKE protocol which was written by the same core researchers.



# DE-PAKE

In 2015, the “Device-Enhanced Password Protocols with Optimal Online-Offline Protection” paper was published in the IACR’s ePrint repository. This paper on “Device-Enhanced Password Authentication Key Exchange” (DE-PAKE) protocol just threw me away by its novelty on solving password authentication and storage flows. Turns out that PAKE protocol has been researched for years, long before the DE-PAKE variant. The “Device-Enhanced” variant is actually the one used by SPHINX. What I haven’t mentioned about it yet is that the device, or online service, has zero-knowledge of the password before encrypting it. The browser extension has also zero-knowledge of the device key, thanks to the “Oblivious Pseudo-Random Function” (OPRF) that is implemented with Elliptic Curve technology in SPHINX. OPRF is a “pseudorandom function that is computed by two parties, one that holds the key to the function and learns nothing from the computation, and one that holds an input and learns the output of the function on that input and nothing else.”

The most interesting part of DE-PAKE is the PAKE free-PKI protocol support where TLS is not required. As shown in Image 3, the idea is to use a Diffie-Hellman key exchange-like protocol which relies on the “random password” as input, but where the server only stores a “state” of the key exchange. This means that even if the database is breached, or if a man-in-the-middle attack occurs, no knowledge of the random password is leaked. Thus, the full entropy key space must be searched by the attacker to achieve compromise of the random password. Furthermore, since the random password is derived from the user’s device or from an online service, no information about the original password is leaked and therefore an attacker would also have to search for the device or online service’s secret full key space. If an attacker listens to the communication between any component, no useful information is leaked. Using strong and modern algorithms such as Elliptic Curves, the key space for random password and device’s secret key should be at least 256 bits, which is considered highly secure as we speak today. Now, if the device and database are compromised, no secret about the password is leaked, but security would fall to the strength of the password.



If the database is compromised, the user can simply rotate their device (or online service) secret key and update their new PAKE state on the server. The password does not need to change and is not affected by common password-reuse weaknesses.

# Conclusion

The DE-PAKE specification on which SPHINX relies on is a very interesting step to securing the authentication flow and mitigating database compromise impacts. However, we might not see them replace the random-generated passwords from modern password managers soon unless backends widely accept the PAKE protocol.

Without PAKE adoption, a password manager does have the ability to generate a random password and send it over TLS which offers almost the same security level as SPHINX. The user's password is only required to access the password manager where all random passwords should be stored. But still, some security properties of SPHINX might be attractive for specific use cases. I'm going to keep a close eye on this research, and so should you!