Exploring Password Authenticated Key Exchange (PAKE) Protocols: A Usability Evaluation

Thomason Zhao
UW–Madison

Thomas Peng
UW–Madison

ABSTRACT

Password-authenticated key exchange (PAKE) protocols offer a more secure alternative to traditional password-based authentication by establishing a secure session key between a client and server without directly exposing the user's password. This project evaluated the usability of three PAKE protocols - Password Over TLS, Secure Remote Password (SRP), and OPAQUE - by assessing their performance, scalability, and user experience. Through experiments measuring server-side resource utilization and a user study with 136 participants, we found that while PAKE protocols provide stronger security guarantees, they can incur higher computational overhead on the server compared to basic password-over-TLS authentication. However, participants generally felt more secure and preferred the PAKE-based authentication, despite the increased response times. These insights contribute to understanding the practical considerations for deploying PAKE protocols in real-world applications, informing decision-makers on balancing security, efficiency, and user-friendliness.

1 INTRODUCTION

The widespread reliance on password-based authentication has long been a contentious topic in the cybersecurity landscape. Traditional password-based authentication systems are susceptible to various attacks, such as password guessing, dictionary attacks, and credential stuffing, which can compromise user accounts and expose sensitive information. To address these vulnerabilities, researchers have developed more robust authentication methods, with password-authenticated key exchange (PAKE) protocols emerging as a promising solution.

PAKE protocols are designed to establish a secure session key between a client and a server without revealing the user's password, even in the face of an adversary with extensive knowledge of the system. These protocols leverage cryptographic techniques to ensure that after a login attempt, whether valid or invalid, the client and server only learn whether the password matched the expected value, without leaking any additional information. This property is particularly valuable in threat models where adversaries may have access to password leaks, password distributions, and substantial computational power.

Conference'17, July 2017, Washington, DC, USA

ACM ISBN 978-x-xxxx-xxxx-x/YY/MM…$15.00
https://doi.org/10.1145/nnnnnnn.nnnnnnn

The primary goal of this project was to evaluate the usability of different PAKE protocols, assessing their performance, scalability, and user experience. By conducting a comprehensive analysis, we aimed to provide insights into the practical implications of adopting PAKE protocols in real-world authentication systems, addressing the tradeoffs between security, efficiency, and user-friendliness.

To achieve this objective, we reimplemented two PAKE protocols - Secure Remote Password (SRP), and OPAQUE - within a web-based application framework utilizing React.js for the front-end, Node.js for the back-end, and MongoDB for the database. We also implemented a baseline password-over-TLS authentication mechanism as a point of comparison. We then performed a series of experiments to measure the protocols' performance and scalability, focusing on metrics such as CPU usage, memory consumption, and storage requirements. Additionally, we conducted a user experience study to gather feedback on the usability, trustworthiness, and perceived safety of the PAKE protocols from a diverse group of participants.

The findings from this project contribute to the understanding of the practical considerations surrounding the deployment of PAKE protocols in real-world settings. By evaluating the tradeoffs between the security benefits and the system-level impact of these protocols, we hope to inform decision-makers and system designers on the feasibility and trade-offs of incorporating PAKE-based authentication into their applications, ultimately enhancing the overall security posture while maintaining a positive user experience.

2 BACKGROUND AND RELATED WORK

Passwords remain the most common form of user authentication on the internet today. However, traditional password-based authentication systems have long been a point of concern due to their inherent vulnerabilities. Passwords can be easily guessed, stolen, or compromised through a variety of attacks, exposing user accounts and sensitive information to unauthorized access. This has prompted the research and development of more robust authentication methods that can better protect user credentials while maintaining a positive user experience.

2.1 Password-Authenticated Key Exchange (PAKE) Protocols

Password-authenticated key exchange (PAKE) protocols are a class of cryptographic techniques designed to establish a secure session key between a client and a server without explicitly revealing the user's password. These protocols leverage the user's password as the primary authentication factor, while ensuring that the password itself is not exposed during the authentication process. PAKE protocols are particularly useful in scenarios where traditional

password-based authentication mechanisms are vulnerable to offline attacks, such as password guessing, dictionary attacks, and credential stuffing.

The core idea behind PAKE protocols is to allow the client and server to mutually authenticate each other and derive a shared session key, without the server ever learning the client's password. This is achieved through a series of cryptographic operations that ensure the password is only used as a key to unlock the authentication process, rather than being directly transmitted or stored on the server. By preserving the confidentiality of the password, PAKE protocols offer a higher level of security compared to traditional password-based authentication mechanisms.

In a typical PAKE protocol, the client provides their username and password to the server, and the server responds with a challenge or some form of cryptographic information. The client and server then engage in a series of message exchanges, performing various computations and verifications to establish the shared session key. At the end of the protocol, both the client and server are confident that the other party has successfully authenticated, without revealing the password to either party.

2.2 Threat Model and Research Goal

The primary threat model considered in this project assumes that all adversaries have access to any exposed information, such as password leaks or password distributions, as well as some computational power, such as the ability to precompute password hashes. This threat model reflects the reality of modern cybersecurity challenges, where attackers often possess significant resources and knowledge about the target system.

Within this threat model, we identified two main adversarial roles:

- **Adversarial Client:** A malicious client that attempts to gain unauthorized access by exploiting any vulnerabilities in the authentication process.
- **Adversarial Server:** A malicious server that tries to learn a user's identity and credentials by manipulating the authentication process or storing sensitive information.

The research goal of this project was to evaluate the usability of different PAKE protocols, assessing their performance, scalability, and user experience. By conducting a comprehensive analysis, we aimed to provide insights into the practical implications of adopting PAKE protocols in real-world authentication systems, addressing the tradeoffs between security, efficiency, and user-friendliness.

2.3 Existing PAKE Protocols

To achieve our research goal, we selected two PAKE protocols to evaluate - Secure Remote Password (SRP) and OPAQUE. These protocols represent different approaches to password-authenticated key exchange and offer varying levels of security and usability. We also implemented a baseline - Password Over TLS - authentication mechanism as a point of comparison to assess the performance and security benefits of PAKE protocols.

- Password Over TLS (baseline): This protocol leverages the Transport Layer Security (TLS) protocol to encrypt the communication between the client and server, ensuring that the password is transmitted securely. The server then checks the user's password by comparing it to the stored value in the database.
- Secure Remote Password (SRP): SRP is a PAKE protocol that allows the client and server to authenticate each other and establish a shared session key without explicitly revealing the password. The protocol involves a series of computations and message exchanges to verify the password without transmitting it in plain text.
- OPAQUE: OPAQUE is a more recent PAKE protocol that utilizes an oblivious pseudorandom function (OPRF) to perform the password-based key exchange. This protocol aims to provide stronger security guarantees and protect against a wider range of attacks, such as offline dictionary attacks and server compromise.

By evaluating these different PAKE protocols, we aimed to gain a comprehensive understanding of the tradeoffs between their usability, performance, and security characteristics, ultimately providing insights to guide the adoption of PAKE-based authentication in real-world applications.

3 IMPLEMENTATION

To measure the difference between PAKE protocols and password over TLS, we chose to create three identical applications that utilize different protocols as authentication methods - SRP, OPAQUE, and Password over TLS, where the Password over TLS implementation is used as a control to compare the differences with the PAKE protocols. A simple registration and login page is created for each of the authentication methods, and to minimize the difference between the authentication methods, we tried to develop all three of them using same JavaScript modules. We chose to use React for our client side front end framework, ExpressJS for server side back end, and MongoDB for the database. Lastly, the client will communicate to the server with HTTP requests. The frameworks we chose are commonly see in actual implementation around the world today, which we believe that by using these frameworks we can yield a closer performance to what it would be if these protocols were used in actual production.

Also, to understand how other users would feel about PAKE protocols, we created a questionnaire for users to fill out after trying out our implementation, in the hope for learning not only their experience with our implementation, but also their perspectives towards the promises of PAKE protocols.

3.1 Service design

All of these services are hosted on the cloud, with each component — the front end, back end, and databases — running on virtual private servers with 1 core and 2G memory.

3.1.1 Front end: Since the goal is to analyze the performance and user experience, we aim to design a simple but functional login and register page. For the home page, we only include two buttons that bring users to either register page or login page. In register page and login page, there are two input boxes for users to put username and password respectively, and two buttons, one to submit and one to jump to another page for easier access; the one button on register page will bring users to login page, and vice versa. After clicking

the submit button, the front end will start performing per protocol, and inform users if the register or login attempt is successful or not. Routing in these pages is managed with React-router-dom, and communications with back end is handled with the axios module.

3.1.2 Back end: Our back end implementation uses ExpressJS to serve as the central hub for server-side calculations and to facilitate communication with the MongoDB database using the mongoose module. To ensure seamless interaction, Cross-Origin Resource Sharing (CORS) is enabled across all routes, with the origin set to the front end service's IP address. The back end service is designed to listen on a port and handle register or login API calls from the client. There might be multiple routes required for one register or login request due to to design of protocol, but on a successful attempt all the routes should be executed exactly once.

3.1.3 Database: We booted up a MongoDB database using the latest MongoDB docker image, since it is easier to manage than installing and run. We created 3 different databases in MongoDB, with each one dedicated to store data received through corresponding protocol.

3.2 SRP

After searching over Github for a proper implementation for SRP protocols in JavasSript, we chose the implementation by LinusU/secure-remote-password that has actual source code which we can investigate, and the fact that the repository has multiple stars strengthen our belief that it is a good implementation.

3.2.1 Register flow: To register to a SRP service, the client side will start by generating a salt, and derive a private key through the salt, username and password. The client will then produce a verifier with the private key, and both the username and the verifier will be sent to the server and stored in the database.

3.2.2 Login flow: To login a service with SRP protocol, the client side needs to make 2 HTTP requests to the server in one login attempt.

In the first request, the client will generate a pair of secret/public ephemeral pair, and send the public ephemeral to the server alongside with the username. Upon receiving the username, the server can then lookup the salt and verifier stored for the user, and generate the server's secret/public ephemeral pair using the verifier. The salt and server's public ephemeral would be sent back to client.

Since the server cannot save temporary data in the memory after an HTTP request is handled, we store the private ephemeral key back to the database under the username, and make sure to clean up after a certain period of time if it is never used.

When the salt and server public ephemeral arrived at the client side, the client can then reproduce the private key with the salt, username and password. With the private key, the client should be able to derive a shared strong session key using server's public ephemeral, client's secret ephemeral, the private key and password. The client should send another HTTP request to verify the correctness of session key derived.

Upon receiving the request, the server will also derive a session key with the server's secret ephemeral, client's public ephemeral, and the verifier. The server will then send a proof back to the client, allowing the client to check if both parties end up having the same key.

3.3 OPAQUE

We decided to use the implementation by serenity-kit/opaque on Github, which was still maintained up until three months ago, with a website for detailed documentation. Although the source code for the protocol is not provided on Github, the fact that it is based on another Meta project and the thorough documentation prompted our decision to use this implementation.

3.3.1 Setting up the server: OPAQUE requires a long-term secret key to perform any server-side calculations. To load the key, we try to search in the database for the secret key. If it is the server's initial boot-up, we will generate a new secret key with the module's dedicated function and store it in the database.

3.3.2 Register flow: Upon register, the client would generate a registration request with the password, which will be sent to the server and wait for its approval. The server would create a response to the request, using the server secret key and username, and send it back to client. Client will then encrypt the response and the password into an encrypted registration record, send the record to the server, and store it in the database alongside with the username. If it is the first time the server being booted up, we will generate a new private key and store it in the database.

3.3.3 Login flow: To login, client would start with creating a login request with the password, and send the request to the server. A variable that record the client login state is also generated in the process, which will be used in later steps.

The server will query out the user's registration record from the database, and calculate the response with server secret key, username and registration record. Also, this step generates a server login state variable, which will be used in later steps. Same issue as SRP, since data cannot be stored in memory after the HTTP request is handled, we will store the server login state variable in the database under the username, and drop the value if it is not used after a period of time.

After the client received the response, client can calculate the result by using the client login state variable, the response and password. If the login result does not yield a true value, the login fails and should be stopped. Else, client can derive the session key and a variable for finishing login request from the login result. The finishing request will be sent to the server to for session key generation.

Finally, the server will calculate the session key with the client's finishing request and the server login state variable.

3.4 Password over TLS

In the Password over TLS implementation, we chose to use CryptoJS as the source library for our hash functions. Our implementation only uses the SHA256 function in the library for calculating salted password hash.

3.4.1 Register flow: Both username and password are sent from client to the server in plain text. The server would generate a random salt, concatenate the salt to the password, and compute the

SHA256 hash. Finally, the username, password hash and salt will be stored in the database.

3.4.2 Login flow: Both username and password are sent from client to the server in plain text. The server will query out the user's salt, concatenate to the password provided by client, and compute its SHA256 hash. The server will then check if the calculated hash is exactly the same as the one stored in the database. Finally, the result of pass or fail will be sent back to client.

3.5 Survey design

We created a simple anonymous Google Forms questionnaire to understand whether users understand PAKE protocols, and their experiences with them after trying the protocols out themselves. The questionnaire focuses on three main sections: knowledge about PAKEs, trust towards current Password over TLS implementation, and experience feedback for PAKEs. We sent the link to the Google Forms to our friends and family, and asked them to forward further to their friends and family.

We started by asking if users have ever heard of PAKE protocols, and where they heard about it if they give a positive answer. Then, we asked them if there is a protocol that requires no sensitive information being sent over the internet, how likely will they use the protocol instead of Password over TLS. Lastly, we provided the IP address to our implementations, and asked them to try the three services out themselves. We then ask them if they feel any latency in response time between the three protocols.

4 EXPERIMENTAL MEASUREMENTS AND RESULTS

To evaluate the performance and scalability of the PAKE protocols, we conducted a series of experiments on the server-side implementation. The goal was to measure the impact of the different PAKE protocols on the system's resource utilization, including CPU usage and storage requirements, to understand the practical implications of adopting these protocols in real-world applications.

Our benchmark simulated a high-pressure user registration and login scenario, with 10 concurrent user threads performing 1,000 rounds of registration and login. The results showed a significant difference in the CPU usage patterns between the PAKE protocols and the basic password-over-TLS authentication.

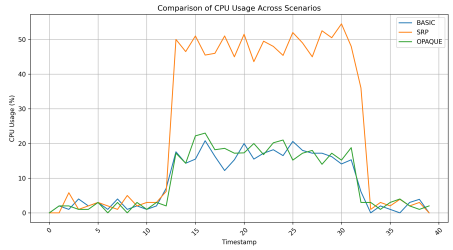

Figure 1: Comparison of server-side CPU usage

4.1 Server-side CPU Usage

One of the key performance metrics we examined was the server-side CPU utilization over time. The PAKE protocols, by nature, involve more complex cryptographic operations compared to a basic password-over-TLS authentication, which could potentially lead to higher CPU consumption.

As shown in Figure 1, the Password Over TLS protocol exhibited a relatively stable and low CPU usage throughout the experiment, as the server-side logic primarily involved hashing the password and comparing it to the stored value in the database.

In contrast, the SRP protocol demonstrated a much higher and more variable CPU usage over time. The server-side computations required for the SRP protocol, such as generating ephemeral keys and deriving the session proof, placed a significantly higher demand on the CPU resources, as evident from the figure. This could be a potential bottleneck for servers handling a large number of concurrent authentication requests.

The OPAQUE protocol also showed elevated CPU usage compared to Password Over TLS, but it was generally lower and more stable than the SRP protocol, as can be observed in Figure 1. The server-side operations in OPAQUE, such as the oblivious pseudorandom function (OPRF) and the encryption/decryption of the client's private key, appeared to be more efficiently implemented, leading to a more manageable CPU utilization.

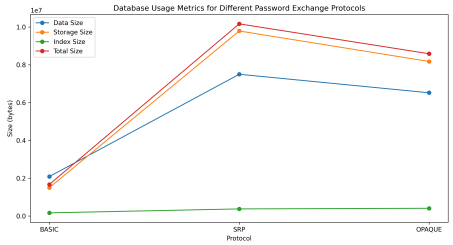

Figure 2: Comparison of server-side storage usage

4.2 Server-side Storage Usage

Another aspect we investigated was the impact of the PAKE protocols on the server-side storage requirements. The storage usage is an important consideration, as PAKE protocols often require storing additional data, such as salts, session information, or encrypted client keys, to facilitate the authentication process.

From Figure 2, our experiments revealed that the PAKE protocols, in general, required more storage space compared to the basic password-over-TLS authentication. The SRP protocol and the OPAQUE protocol had roughly the same storage usage, with the SRP protocol being slightly higher.

For the SRP protocol, the server needed to store the user's username, the generated secret value (V), and the salt (s) for each registered user, as evident from the figure. This additional data overhead amounted to an increase in storage requirements compared to the baseline password-over-TLS protocol.

The OPAQUE protocol, on the other hand, required storing a more complex data structure, including the encrypted envelope (E) containing the client's private key, the client's public key (CKp), and the salt (s) specific to each user's username. The storage usage for the OPAQUE protocol was slightly lower than the SRP protocol, but still significantly higher than the baseline, as can be observed in Figure 2.

The Password Over TLS protocol, being the simplest in terms of server-side implementation, had the smallest storage footprint, as it only required storing the user's hashed password along with the salt. The storage usage for the Password Over TLS protocol was roughly half of the SRP and OPAQUE protocols, as depicted in the figure.

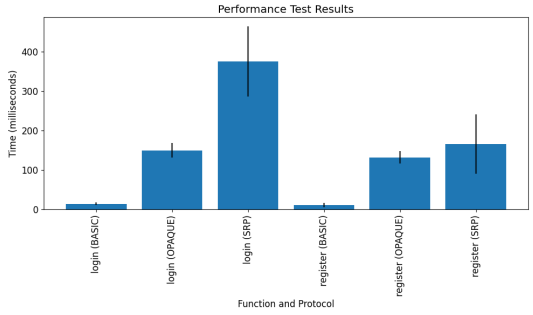

Figure 3: Comparison of client-side per-request response times

4.3 Client-side experience

In addition to the server-side performance metrics, we also examined the client-side experience during the authentication process. This was an important aspect to consider, as the user-perceived responsiveness and efficiency of the authentication flow can significantly impact the overall user experience.

The performance test results, as depicted in Figure 3, showed that the Password Over TLS (baseline) exhibited the fastest and most consistent response times throughout the simulation. The client-side request processing was relatively straightforward, involving the transmission of the username and password, followed by the server's verification and response. This simplicity translated to a smooth and responsive user experience, with minimal latency.

In contrast, the SRP protocol demonstrated significantly longer and more variable response times when compared to the Password Over TLS protocol, as evidenced by the figure. The client-side implementation of SRP required a more complex series of message exchanges, including the generation of ephemeral keys, the computation of the session proof, and the verification of the server's response. This additional computational overhead and the back-and-forth communication between the client and server resulted in a less responsive user experience, with some requests taking considerably longer to complete.

The OPAQUE protocol also exhibited longer response times compared to the Password Over TLS protocol, but the overall performance was better than the SRP protocol, as can be observed in the figure. The client-side operations in OPAQUE, such as the oblivious pseudorandom function (OPRF) and the encryption/decryption of the client's private key, were more optimized than the SRP protocol, leading to a more consistent and responsive user experience. However, the response times for OPAQUE were still noticeably longer than the baseline Password Over TLS protocol.

5 QUESTIONNAIRE RESULT

The questionnaire remained open for a week, during which time it garnered responses from a total of 136 anonymous participants. Since we shared the link out to friends and family, we are expecting the age distribution of the participants to be mainly around 18 to 25, with some participants reaching age 50 and above. From our friend groups and family members, we assume all the participants to the questionnaire has basic knowledge to technology, such as navigate through websites or using social media.

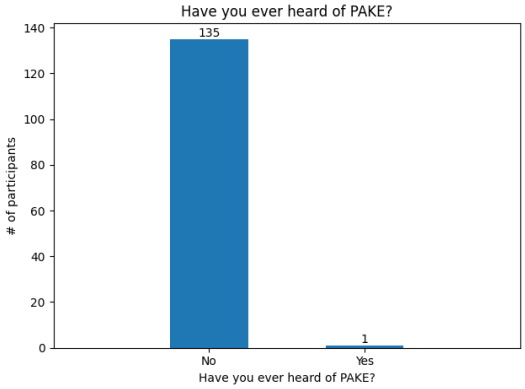

Figure 4: Numbers of participants heard about PAKE

5.1 Knowledge about PAKE

We asked the participants whether they have heard about PAKE protocols or not. Initially, we did not expect any participants to know anything about PAKE, speaking from our experience that none of us knew PAKE until we start working on the project. But surprisingly, one participant out of 136 actually knew about PAKE protocols with prior knowledge, as shown in Figure 4. They highlighted that "PAKE protocols use zero-knowledge proofs to authenticate without transmitting passwords. Good for security," which is the goal PAKE is aiming to achieve.

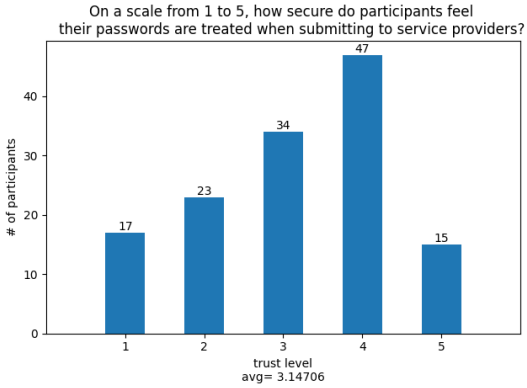

Figure 5: Trust towards service providers with participants' passwords. The intensity is measured on a scale of 1 to 5, where 1 represents not secure, and 5 represents very secure.

5.2 Trust to Password over TLS and service providers

We are also curious how users trust service providers with their password. We informed the participants that Password over TLS

implementation requires storing encrypted password in the service providers' databases, and wanted to collect some feedback for this implementation. Most answers surrounded the idea of data breach and leaks, which is one way an adversary gain control of a user's account. One participant mentioned that "widely-used authentication methods, once subject to a data breach, are completely vulnerable: all users' data can be extracted, as user passwords are entirely stored on the server side databases." While this is not entirely correct, storing password in server's database can add additional risks to lose all other personal data stored with the service provider.

Another participant mentioned password reusing, saying that most people would not bother to remember multiple passwords to different services, which can put all the services that use same password in danger if one of them is breached. He also said in order to prevent the incident from happening, users often use password managers or notebook applications to keep track their passwords, but the additional use of programs brings another vulnerability to the potential leak of passwords.

Furthermore, we asked the participants whether they trust service providers with their passwords. The result in Figure 5 shows that part of the uses are confident about their password being handled safely by the service providers, but at the same time, another part of the participants felt uncertain about the situation. We can see a slightly right skewed trend from the figure, but we believe it is not significant enough to show whether a user trust the server would safe their passwords securely or not.

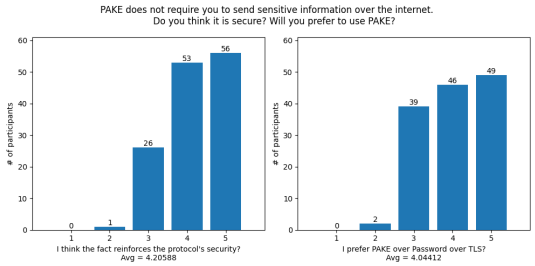

Figure 6: Participants' opinion on PAKE protocols. The intensity is measured on a scale from 1 to 5, where 1 represents most unlikely, and 5 represents most likely

5.3 Preference between PAKE and Password over TLS

In the questionnaire we stressed the point that PAKE protocols do not require passwords to be sent through the internet. We would like to see how participants would response with this information, whether they think it is more secure or less. From Figure 6, it is apparent that the majority of participants view authentication methods that avoid transmitting passwords over the internet as more secure, and have a higher preference to embrace protocols that does not require transmitting.

In Section 5.2, participants pointed out some potential flaws Password over TLS have, but only a slight right skew on Figure 5 is observed. However, in Section 5.3, they showed high interest in using protocols that does not send sensitive information. We assume that the slight right skew comes from the trust towards big companies that most users think are reliable, but when an alternative that does not require sensitive data to be transmitted, users would opt for the second option to guard their security.

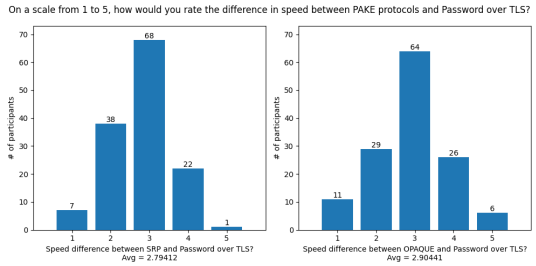

Figure 7: Response time difference between PAKE protocols and Password over TLS. The intensity is measured on a scale from 1 to 5, where 1 represents much slower, and 5 represents much faster.

5.4 User experience analysis

We provided participants with access to the three implementations, enabling them to directly assess the variations in time consumption. Participants were then instructed to log in using all three services, assess their responsiveness, and make comparisons between the two PAKE protocols and the controlled Password over TLS implementation. From Figure 7, participants report similar time consumption when comparing both PAKE protocols to Password over TLS.

This raises our question about why PAKE is rarely implemented. Despite the minimal differences in user experience compared to Password over TLS, we anticipated increased adoption of PAKE across various platforms. However, the reality contradicts our expectations. We assume that developers chose not to implement PAKE protocols becuase the heavy workload that falls on the server side, showed in Section 4. Also, PAKE implementations are much more complicated then Password over TLS, which requires much more effort to trace the logic when building the protocol and maintaining the system.

6 CONCLUSION

REFERENCES