OPAQUE: Protecting User Data during Server Breaches

A Technical Report submitted to the Department of Computer Science

Presented to the Faculty of the School of Engineering and Applied Science
University of Virginia • Charlottesville, Virginia

In Partial Fulfillment of the Requirements for the Degree
Bachelor of Science, School of Engineering

Maven Kim
Spring, 2022

Shangtong Zhang, Department of Computer Science

OPAQUE: Protecting User Data during Server Breaches

CS4991 Capstone Report, 2023

Maven Kim
Computer Science
The University of Virginia
School of Engineering and Applied Science
Charlottesville, Virginia USA
mmk6xnb@virginia.edu

ABSTRACT
Ring, an Amazon-owned company that creates home security and smart home devices, must consistently update their data security protocols to preemptively prevent server breaches. To solve this problem, I developed a basic implementation of an oblivious pseudorandom-function password authenticated key exchange (OPAQUE), which secures user data in the event of a server compromise. I created my implementation using Python, Docker, and Amazon Web Services' (AWS) Nitro Enclaves. I collaborated with a fellow intern on this project using GitHub, a version control system. We successfully engineered a basic user registration and authentication application that utilized OPAQUE. In the future, we can scale this application to support multiple users at the same time.

1. INTRODUCTION

Passwords are ubiquitous but fundamentally insecure. When a client transmits a password to a server, there are multiple opportunities for a hacker to steal it. An attacker can eavesdrop on the communication and intercept the password while it is being sent to the server. The hacker could also attack the server itself and steal password data directly. Companies protect against these attack vectors in several ways. During transmission, data is encrypted using a protocol called Transport Layer Security (TLS). When server receives password data, it applies a one-way irreversible function, called a hash function, to the password, making the data unreadable.

These encryption mechanisms have several vulnerabilities. The server must handle a plaintext version of the password to compare it against the encrypted version when a user logs in. During this process, the server could accidentally log the password or become corrupted, making the data vulnerable in the event of a server breach. For example, in 2019, Facebook discovered that it was storing user passwords in an unencrypted format. Users must also trust the server to properly handle their password data. Even if the server is trustworthy, data can be accidentally mishandled, and users cannot know for certain that their passwords are properly stored.

Ring, an Amazon-owned company that creates home security and smart home devices, attempts to solve this issue with its end-to-end encryption. In this protocol, the user's smartphone encrypts the user's videos and data before they are sent and stored in Ring's servers. The only way to access this data is by using a ten-word autogenerated password local to the phone. No unauthorized users, including Ring employees or other potential eavesdroppers, are able to access any user data unless they have this password.

The issue with this implementation is that remembering and using a ten-word password is rather tedious. If the user forgets the password or loses their phone, their data is also lost with no method of recovering it.

The OPAQUE protocol is one possible solution to this problem. By employing an Oblivious Pseudo-Random Function (OPRF) in a Password Authenticated Key Exchange (PAKE), user passwords and encrypted representations of them are never transmitted to the server. The protocol also ensures that users have sole access to their video data without requiring them to remember complex passwords.

2. RELATED WORKS

Bourdrez et. al. (2023) outline a specification of the OPAQUE protocol. They state that OPAQUE has a formal security proof and they list some of the advantages of the protocol. OPAQUE is secure against pre-computation attacks, unlike other PAKE protocols. A pre-computation attack involves a hacker taking a list of common passwords, and applying a hash function to them to create a table of hashes. When a server is compromised and the hacker retrieves a list of hashed passwords, the attacker can instantly look up each of these hashed passwords in the table to find each user's password. With OPAQUE, each user has some server-generated random data, called a salt, that is added to the password before it is hashed. This makes the pre-computed table of hashes useless, since the attacker has no way of knowing the salt before server compromise. Users then have time to change their passwords before an attacker obtains unauthorized access to their data.

Bellovin and Merritt (1992) specify the benefits of PAKE protocols, which are the basis of OPAQUE. In a PAKE protocol, the client and server never reveal their password to each other. They establish a strong shared secret key only when the passwords match. When this occurs, the server can authenticate the client, and the client can authenticate the server. The disadvantage of this early protocol was that it relies on passwords being stored in plaintext on the server, making user data vulnerable in the event of server compromise.

3. PROJECT DESIGN

At the beginning of my internship, my manager assigned a fellow intern and me to work on creating an OPAQUE prototype using Nitro Enclaves. Nitro Enclaves allows for rate-limiting, which prevents any potential brute-force attacks and adds an extra layer of security to the OPAQUE protocol. Our project also needed to use Docker containers, since they serve as a template for an enclave, and Python. Our manager also provided us with a working command-line interface (CLI) implementation of OPAQUE, so the majority of the project involved incorporating this with Nitro Enclaves. Outside of this initial project description, the rest of the implementation details were left to us.

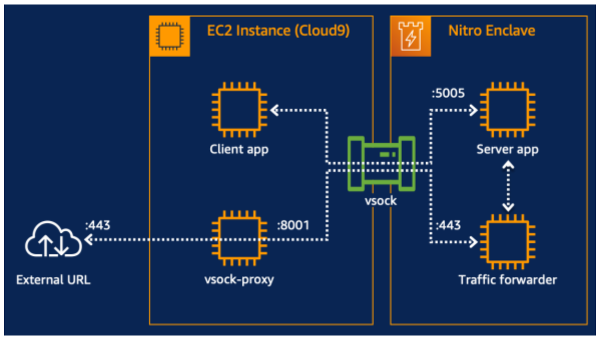

Figure 1: Basic Prototype Structure

As shown in Figure 1, our prototype included three main features: a single client, a single server running in an enclave, and the communication channel between them. The client registers users and lets them log into the server, which secures user information with the OPAQUE protocol. Incorporating multiple servers would have introduced parallelism and a level of complexity that we could not address during our short internship. This entire system ran on a virtual Elastic Compute Cloud (EC2) instance.

The client, which my partner primarily worked on, consisted of a command-line interface that registered and authenticated users. Users could register an account using a username and password. After registration, users could log into their account using the same username or password. Behind the scenes, the client was securely performing the client-side portion of the OPAQUE protocol, which hinges on an OPRF. The OPRF takes in two values, a secret key provided by the server and the client's password. The client can use the OPRF to compute the result of this function without learning the secret key, and the server can provide the key without learning the password or the result. Along with this result, the client generates a public and private key pair specific to the newly created account and uses the OPRF result to encrypt an envelope containing this key pair.

The server, which is running on the Nitro Enclave, handles both the storage of the envelope and the server-side portion of the OPAQUE protocol. Each user's envelope along with the secret key provided by the server was stored on AWS DynamoDB and encrypted with AWS Key Management Service (KMS). When the user authenticates, the server sends back the envelope along with the secret key linked to the user's username. If the client generates the correct OPRF result to unlock the envelope, it initiates a PAKE with the server so that they can mutually authenticate each other.

The communication between the client and the server was formatted using JavaScript Object Notation (JSON) and done using Transport Layer Security (TLS) over the vsock channel. This channel is the only way the enclave can communicate with external services. When the enclave needs to communicate with an external service, such as DynamoDB or KMS, it sends a document containing measurements that are specific to the enclave. These services can then verify the enclave using these measurements. This process, called cryptographic attestation, fails if the enclave's source code is changed or if the code is run outside of the enclave, further securing the secret key and envelopes used in the OPAQUE protocol.

4. RESULTS

At the end of the internship period, my partner and I created a script that automatically set up the OPAQUE prototype we created. The script downloaded all of the client code, server code, and any other dependencies our project relied on. We demoed this script in front of the entire AWS Cryptography team. We also documented our code so that other members of the team could continue to expand on the prototype we created. Our prototype template became an AWS sample, which gives other developers at AWS open access to our code. This allows other developers to further develop the prototype into a system Ring can use at scale.

We also extended cryptographic attestation functionality for AWS KMS. Originally, KMS could only attest decrypt operations from KMS, meaning it could only verify if the enclave ran a decrypt operation. We enabled the encrypt operation to also send the attestation document, allowing KMS to verify the enclave during encryption.

5. CONCLUSION

We developed a basic implementation of the OPAQUE protocol using AWS Nitro Enclaves. The OPAQUE protocol addresses the inherent vulnerabilities of traditional password authentication by employing an OPRF during a PAKE. These cryptographic protocols, combined with Nitro Enclaves, secure user data even in the event of server compromise. The prototype was built using Python, Docker, and AWS Nitro Enclaves, and we demoed this project to the AWS Cryptography team.

6. FUTURE WORK

In the future, there are several avenues we could explore to further expand on our project. We can work on scaling the application to support multiple users and servers simultaneously, which would help make the system more robust and practical for real-world usage. Without this support, companies like Ring cannot use our OPAQUE implementation.

To further optimize the user experience, we can also investigate ways to move the user registration and authentication process to a more user-friendly user interface, instead of a command line interface. We also need to conduct more rigorous testing and benchmarking to identify and address any potential performance bottlenecks or security vulnerabilities, ensuring that the system remains reliable and secure as it evolves.

7. ACKNOWLEDGMENTS

I would like to thank my manager, Matthew Campagna, for providing me with guidance Be throughout my internship period. I would also like to thank my fellow intern, Alex Richman, for working on this project with me.

REFERENCES

[1] Bellovin, S. and Merritt, M. 1992. Encrypted key exchange: password-based protocols secure against dictionary attacks. In Proceedings 1992 IEEE Computer Society Symposium on Research in Security and Privacy, IEEE Comput. Soc. Press, Oakland, CA, USA, 72–84. DOI:https://doi.org/10.1109/RISP.1992.213269

[2] Bourdrez, D., Krawczyk, H., Lewi, K., and Wood, C. 2023. The OPAQUE Asymmetric PAKE Protocol. Internet Engineering Task Force. Retrieved April 7, 2023 from https://datatracker.ietf.org/doc/draft-irtf-cfrg-opaque

[3] Bradley, T. 2020. OPAQUE: The Best Passwords Never Leave your Device. The Cloudflare Blog. Retrieved April 7, 2023 from http://blog.cloudflare.com/opaque-oblivious-passwords/

[4] Canahuati, P. 2019. Keeping Passwords Secure. Meta. Retrieved April 7, 2023 from https://about.fb.com/news/2019/03/keeping-passwords-secure/

[5] Nitro Enclaves. n.d. secure-local-channel-arch.png (1130×631). Retrieved April 7, 2023 from https://nitro-enclaves.workshop.aws/images/secure-local-channel-arch.png

[6] Ring.com. n.d. Understanding Video End-to-End Encryption (E2EE). Ring Help. Retrieved April 7, 2023 from https://support.ring.com/hc/en-us/articles/360054941511-Understanding-Video-End-to-End-Encryption-E2EE-