Parcae: A Blockchain-Based PRF Service for Everyone

Elizabeth Wyss(✉) and Drew Davidson

University of Kansas, Lawrence, USA
{elizabethwyss,drewdavidson}@ku.edu

Abstract. Pseudorandom function (PRF) services are utilized to cryptographically harden password hashes against offline brute-force attacks. State-of-the-art implementations of PRF services can additionally offer benefits such as detection of online attacks and practical key rotation, but the cost of doing so in a publicly distributed setting is requiring clients to trust a third party service. These third party services are not incentivized to behave honestly and pose as a single point of failure for Denial of Service (DoS) attacks. A successful DoS attack mounted against a deployed PRF service would prevent its clients from authenticating their users' passwords, thus making it impossible for users to log in to those clients' services.

To address these issues, we design and implement Parcae, the first blockchain-based publicly distributed PRF service. Parcae offers all of the additional benefits provided by state-of-the-art PRF services while also providing DoS attack resilience and service auditing capabilities through use of a permissioned blockchain. Performance analysis shows that our implementation of Parcae is practical and can scale to meet the needs of a dynamically growing client base in a publicly distributed setting.

Keywords: Blockchain · Smart contract · Password · PRF

1 Introduction

The breach of industry password databases continues to be a severe and costly issue. In a data breach investigations report conducted by Verizon in 2019, it was found that out of 1,285 examined data breaches from that year, 35% involved the breach of user credentials–including passwords stored in various formats [11]. Industry standard practices in password management involve storing passwords in a hashed and salted format, but compromised passwords of this form are still vulnerable to offline brute-force attacks. A single modern GPU can calculate over seven billion sha-256 hashes per second [1], and at this rate a typical password hash (derived from a password of length eight containing uppercase letters, lowercase letters, digits, and special characters) can be determined by brute-force in a matter of days. An attacker can parallelize this attack across hundreds or even thousands of GPUs and reduce the time needed to brute-force a password hash to a matter of minutes.

P. Gladyshev et al. (Eds.): ICDF2C 2021, LNICST 441, pp. 328–341, 2022.
https://doi.org/10.1007/978-3-031-06365-7_20

Application of pseudorandom function (PRF) services to passwords is an effective form of password hardening that offers protection against offline brute-force attacks even in case of password database compromisation. Clients using a PRF service send password hashes to a remote server where they are further encrypted under a private key known only to that server. The PRF service then replies with the encrypted password hash that is used by the client for password verification. Because these password hashes are encrypted with a remote key, passwords cannot be guessed in an offline brute force attack unless the remote key is known by the attacker. Companies such as Facebook [9] have internally used privately distributed PRF services to harden their users' password hashes against offline brute force attacks.

Deploying a publicly distributed PRF service would offer these security benefits to any and every password-authenticating client. However, state-of-the-art PRF services pose issues of trust and availability in a publicly distributed setting; clients using such a service must expand their root of trust to contain the PRF service even though it is a third party that is not incentivized to behave robustly and truthfully. The PRF service is incentivized to reduce computational costs by using weak cryptographic algorithms or by behaving dishonestly altogether. Additionally, a publicly distributed PRF service is a single point of failure which must remain constantly functional and online, or clients would lose the ability to perform user authentication. If a PRF service were to experience downtime, all of its clients would be unable to perform user logins for the entire duration of the downtime. Due to this single point of failure, any publicly distributed PRF service would be vulnerable to denial of service (DoS) attacks, and a successfully mounted attack would be costly for all clients using the service.

Our key insight is that **the weaknesses of state-of-the-art publicly distributed PRF services can be addressed through the use of a permissioned blockchain**. This approach poses significant challenges in that PRF services need to protect information to remain secure, and yet blockchains need to expose information to remain verifiable. We find that the inherent privacy conflict between PRF services and blockchains can be overcome through clever applications of cryptography within a permission-delegated blockchain. We demonstrate the effectiveness of this approach through a prototype implementation called Parcae, the first publicly distributed PRF service residing on a blockchain smart contract.

Parcae utilizes partially oblivious PRFs (PO-PRFs) [4] to achieve the security guarantees of existing state-of-the-art PRF services and further solves the root of trust and availability issues of a publicly distributed setting. To ensure faithful execution of the Parcae protocol, clients using Parcae may opt to operate a blockchain auditing peer which allows clients to ensure that Parcae's blockchain peers adhere to the PRF service smart contract honestly, all without the need to expose any private information. To address DoS attack vulnerability, Parcae's blockchain peers operate as separate queryable entities that are automatically scaled up in quantity to be as many as is necessary to meet the PRF service needs of all of Parcae's clients.

We show that Parcae achieves both trustworthiness and continual availability by design and through experimental evaluation. A single blockchain peer is able to process up to 498 Parcae PRF queries per second with an average end-to-end latency of 40 ms per query. This latency is far less than the latency of historical password hardening approaches such as iterated hashing and memory-hard functions. For example, a 2016 iterated hashing scheme deployed by Apple's iTunes user verification process [6] incurs more than six times the latency of a single Parcae query. As more and more clients query Parcae, additional blockchain nodes are dynamically deployed to meet the throughput needs of all of Parcae's clients.

Contributions. In summary, we make the following contributions:

- We design Parcae, the first blockchain-based publicly distributed PRF service protocol enabling any client to harden password hashes against both offline and online brute-force attacks. Parcae further improves on past designs of PRF service protocols by providing innate auditing capabilities and DoS attack resilience.
- We present a prototype implementation of Parcae which utilizes PO-PRFs deployed on a blockchain smart contract to realize our design goals.
- We conduct an evaluation of Parcae both in terms of performance metrics and security guarantees to justify the effectiveness of Parcae.

2 Related Work

PRF services and password hardening schemes have been approached using a wide variety of cryptographic protocols and deployment scenarios [2,4,7,8,10]. Despite this, past implementations fail to sufficiently defend against the adversaries of a publicly distributed setting.

Pythia [4] introduced the notion of a PO-PRF as a cryptographic primitive and designed several PRF service implementations to be used in different deployment scenarios. Most relevant to our work is Pythia's use of PO-PRFs in a publicly distributed PRF service protocol. Under Pythia's PO-PRF protocol, clients request hardened password hashes from the Pythia PRF service by sending a blinded password hash and an unblinded salt value to a Pythia server. By blinding password hashes, the server learns nothing about users' passwords. With unblinded salt values–assumed to be unique to each user–the server can perform rate limiting on a per-user basis to detect and prevent online brute-force attacks on a user's password. Additionally, Pythia clients can request secret key rotations to cryptographically invalidate old hardened password hashes in case of a password database breach. Pythia also introduced zero-knowledge proof verifiability in PO-PRFs, enabling clients to request a proof that a hardened password hash is computed correctly with respect to a public key known by the client. Furthermore, Pythia's PO-PRF protocol is shown to be provably secure in an honest environment. All of these features combined synthesizes past works on PRFs into a standard baseline for future PRF services.

Schneider et al. designed another cryptographic primitive dubbed a partially oblivious commitment (PO-COM) [10] and showed that it achieves the same functional properties as a PO-PRF under a simpler security model. Using PO-COMs, Schneider et al. demonstrated the construction of a password hardening service that achieves the same features as Pythia while also nearly halving the latency required to harden a single password. Under security definitions weaker than those used by Pythia, they also show that honest-adherence to their system's protocol is provably secure.

Phoenix [8] extends upon on the work of Schneider et al. [10] to show that the security definitions used to prove the security of PO-COMs are too weak, leaving room for effective offline guessing attacks against Schneider et al.'s system. To address these issues, Phoenix is introduced as a password hardening service that achieves improved security and efficiency over Schneider et al.'s system while still maintaining all of the features of Pythia. Phoenix achieves slightly improved password hardening latency and nearly 50% better throughput compared to Schneider et al.'s system. Phoenix is also shown to be a provably secure protocol under strong security definitions when executed faithfully.

Lai et al. later revisit Phoenix [8] to design Password-Hardened Encryption Services (PHE) [7], an even more efficient password hardening system that also supports hardening additional user data. They achieve both stronger security guarantees and about 40% better throughput compared to Phoenix. Their improved system is further demonstrated to be provably secure when honestly executed.

Brost et al. design a threshold variant [2] of PHE [7] which seeks to mitigate the single point of failure in password hardening systems. Threshold PHE is a generalization of PHE in which clients can construct a hardened password by combining the outputs of multiple distinct password hardening services. Since the quantity of deployed password hardening services can exceed threshold of distinct services needed to construct a hardened password, clients can still construct hardened passwords even when some password hardening services are not available. With regards to performance, Brost et al.'s system is slower than PHE in terms of throughput and latency within a factor of three. Additionally, Threshold PHE is shown to be provably secure when honestly executed. Despite the increased availability of Threshold PHE, a publicly distributed version of the system would still require clients to expand their root of trust to contain many external third-party password hardening services that are not incentivized to behave honestly.

While past implementations of PRF services and password hardening schemes have been able to achieve high efficiency and numerous security relevant features, none achieve the service auditing capabilities and full DoS attack resilience required to be an effective solution in a publicly distributed setting.

3 Overview

In this section, we show how the design of Parcae overcomes the limitations of past approaches to fully realize a secure publicly distributed PRF service. We

first identify the key threats to PRF services in a publicly distributed setting, then describe the system architecture of Parcae, and lastly show how Parcae is used to defeat those threats.

We identify two primary PRF service adversaries unique to the publicly distributed setting: a *disruption adversary* seeking to deny legitimate access to the PRF, and an *impersonation adversary*, seeking to falsify PRF queries, thus invalidating the user authentication process. We assume adversaries have significant resources, including the ability to issue more PRF queries than the entire throughput of any statically-sized PRF service, spoof DNS, and stand up multiple masquerading PRF servers. Although we discuss these adversaries separately, we note that our model considers an adversary with goals of disrupting and impersonating a PRF simultaneously.

High Level Design of Parcae. Parcae is a PRF service constructed using PO-PRFs [4] implemented on a blockchain smart contract. In the Parcae blockchain, two distinct types of peers are defined: cryptographic peers that are centrally operated by Parcae and auditing peers that are decentrally operated by clients. Cryptographic peers handle all incoming transactions and queries, storing cryptographically sensitive information in a private data store accessible only to them. They are only permitted to handle queries and transactions as specified in the Parcae smart contract, and any deviation from the Parcae smart contract is detected and disallowed by the blockchain. Auditing peers receive hashed versions of the read and write sets of every approved transaction handled by the cryptographic peers. This allows auditing peers to verify that transactions sent to the Parcae blockchain are handled in an honest and timely matter. Additionally, if any changes are made to the Parcae smart contract, the blockchain ensures that auditing peers are notified of the changes. In case of an adversary compromising the entire Parcae blockchain, this prevents them from altering the Parcae protocol without first notifying all auditing peers of their alterations.

In Parcae, clients issue API requests that are received by intermediate servers. These intermediate servers house the cryptographic certificates necessary to issue transactions and queries to the Parcae's cryptographic peers. Upon receiving a client's request, an intermediate server will translate the request into a Parcae smart contract transaction or query, issue the smart contract invocation to the Parcae cryptographic peers, and then return its result to the requesting client. Like the Parcae cryptographic peers, intermediate servers support multiple clients using them simultaneously and are dynamically scaled up in size and quantity to meet the needs of all of Parcae's clients.

Using Parcae. When a new client wants to begin hardening passwords using the Parcae PRF service, they simply install a Parcae API library and they can start issuing requests. Figure 1 displays the basic API details of each request available to clients. Before a client can query Parcae for hardened password hashes, they must first register with the Parcae PRF service. The Parcae registration process requires a new client to provide and verify an email address, and upon successful verification the client receives their unique Parcae identity string–used to identify

Command	Description
Register()	Registers a new client with Parcae and returns the client's unique id and public key
Query(id, pass, salt)	Returns Parcae PRF query result
GenAuth(id)	Sends a key rotation authorization token to client out of band
GenUpdate(id, authtoken)	Rotates the client's Parcae secret key and returns an update token
Apply(hash, updatetoken)	Applies update token to hash and returns the newly encrypted hash
Audit()	Registers a new auditing peer and sends the requesting client its required cryptographic certificates

Fig. 1. The Parcae API

them to Parcae–and public key–used to verify query proofs generated by Parcae. Once a new client has successfully registered with Parcae, they can begin to invoke query requests and obtain hardened password hashes from Parcae. If a new client has any existing password hash databases that they want to harden with Parcae, they can query the Parcae PRF service with each password hash in their databases to construct one or more hardened password hash databases. After hardening any existing password hash databases, a client updates their user registration and password verification process by adding one additional function call to perform a Parcae query. Now, the client's password hashes are protected by Parcae from both offline and online brute-force attacks. If a client ever detects that a password hash database breach has occured, they can request a key rotation to cryptographically invalidate all previously stored hardened password hashes. To do this, the client first requests a single-use authorization token that is sent to their verified Parcae email address. The client then sends the authorization token in a key rotation request, and Parcae internally replaces the client's secret encryption key with a new one. After a client rotates their Parcae secret key, they receive both a new Parcae public key and an update token that they can use to update the encryption of existing hardened password hashes to be encrypted under their new Parcae secret key. Parcae implements an entirely offline function that clients can use to apply update tokens to old hardened password hashes. Once a client performs a Parcae key rotation, all breached password hashes become cryptographically invalidated since they are encrypted under a secret key that no longer exists anywhere in Parcae.

If a client requests to operate a service auditing peer, Parcae will register a new auditing peer and send its required cryptographic certificates to the client. These certificates are necessary to validate the auditing peer's role and permissions with respect to Parcae's blockchain. After obtaining the certificates, the client simply installs blockchain operation software and executes a single containerized script to bring up their auditing peer which monitors the Parcae service automatically. While auditing Parcae, clients' auditing peers receive all

data blocks distributed on the Parcae blockchain, with private data–such as identity strings and keys–stored in a hashed format. This allows an auditing client to verify their own private data by comparing hashes and furthermore ensures that any change to the Parcae blockchain–such as if the Parcae protocol is altered–is disclosed to all auditors.

4 Parcae Protocol

In this section we give the formal mathematical definitions and security proofs related to the Parcae PRF protocol described in Sects. 3 and 5.

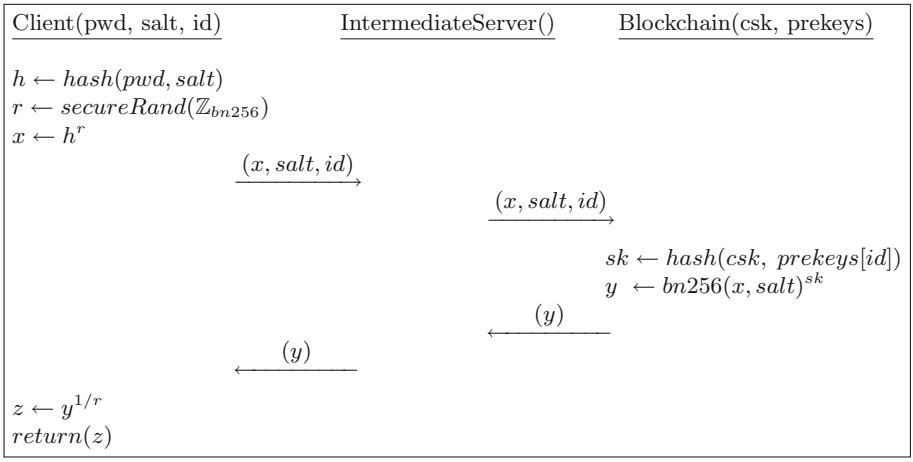

Fig. 2. The Formal Parcae PRF Protocol (Π).

Protocol Definitions. We define the Parcae PRF protocol to be $\Pi = (Client, IntermediateServer,\ Blockchain)$, with the following specifications: *Client* represents the client-side algorithm that takes a password, salt, and identity string as input, calculates a blinded password hash and uses the remaining algorithms to output a hardened password hash. The *IntermediateServer* algorithm simply forwards data from the *Client* algorithm to the *Blockchain* algorithm and vice versa. *Blockchain* is the algorithm which resides on the Parcae Blockchain. It takes a blinded password hash, salt, identity string, central secret key, and set of prekey values as input, and outputs a hardened blinded password hash. Figure 2 depicts Π in mathematical notation.

Protocol Correctness. Π is correct if and only if it deterministically outputs identical z given identical inputs *pwd*, *salt*, *id*, *csk*, *prekeys*. The determinicity of the output z follows directly from the determinicity of hashing algorithms, the invertibility of exponentiation, and the exponent-preserving nature of the bn256 elliptic curve bilinear pairing. Hence, the final output z does not depend

on the only nondeterministic value present in the protocol, r, implying that z only depends on deterministic operations and is itself deterministic for a given set of inputs. Thus, it is proved that Π is correct.

Protocol Security. We extend upon the proof of PO-PRF security provided in [4] to prove the security of our own scheme, Π. To do this, we define a modified protocol $\Pi' = (Client,\ Blockchain)$ that is identical to the protocol Π detailed in Fig. 2, except the *Client* algorithm is modified to send its intermediate output directly to the *Blockchain* algorithm, and the *Blockchain* algorithm is modified to send its output directly back to the *Client* algorithm, thus foregoing the use of the *IntermediateServer* algorithm entirely. Observe that for a given set of inputs *pwd*, *salt*, *id*, *csk*, *prekeys*, both Π and Π' output identical z since Π' does not modify any of the functional operations of Π. Assuming that the security of the key generation algorithms used to generate inputs to Π' holds (this is a reasonable assumption because secure key generation algorithms are well-studied) and that the Parcae Blockchain in which the *Blockchain* algorithm resides on securely performs operations honestly (this is a reasonable assumption given the guarantees provided by a private and permissioned blockchain), then the Π' algorithm is functionally and securely identical to the partially oblivious algorithm proved to be secure in [4]. Hence, Π must exhibit the same proved security as Π' if the inclusion of the *IntermediateServer* algorithm does not alter the functionality and security of Π'. This is indeed the case since it was shown above that Π and Π' are functionally identical, and since the *IntermediateServer* algorithm is itself secure given that the servers in which the algorithm is hosted exhibit the same security and availability properties as the rest of the Parcae system. Thus, it is proved that the Parcae PRF protocol, Π, is secure.

5 Implementation Details

Parcae uses a private and permissioned blockchain based on Hyperledger Fabric [5]. From the guarantees of Fabric, cryptographic peers cannot deviate from the protocol defined in the Parcae smart contract. Additionally, Hyperledger Fabric is designed to parallelize transactions and dynamically enroll new peers. This allows Parcae to dynamically scale in size to meet the availability needs of all of its clients.

We opted to require clients to connect to Parcae through intermediate servers so that they do not need to possess cryptographic certificates enabling them to directly submit arbitrary transactions to the Parcae blockchain. The intermediate servers exist to simply transform requests into well-formatted blockchain transactions and queries. Because we host both the Parcae blockchain and intermediate servers on the same local area network, communication costs between the two are minimized. This use of intermediate servers in no way weakens the security model of Parcae since intermediate servers are dynamically scaled just like Parcae cryptographic peers; the intermediate servers simply provide input filtering.

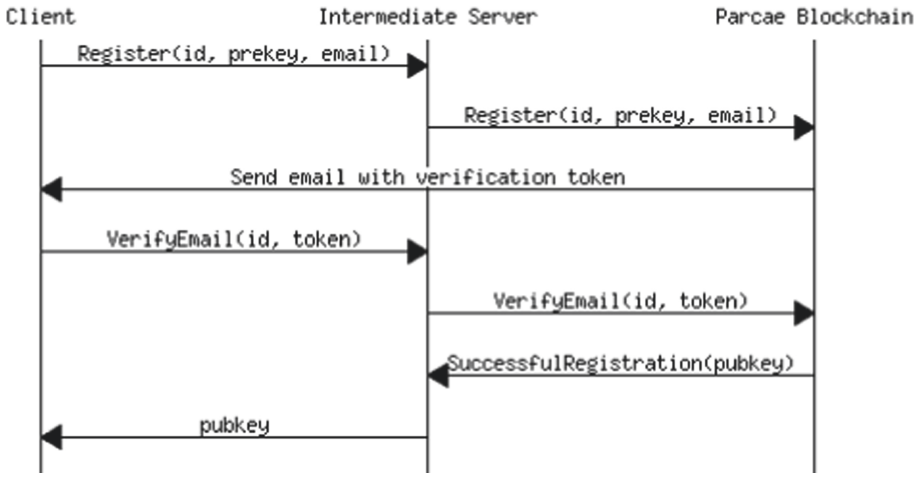

Fig. 3. Parcae client registration protocol diagram.

5.1 Low Level API Details

Parcae Registration. When a new client registers with Parcae, the client first calculates both an identity string and prekey value. Each of these values are computed as secure random integers. Next, the client sends their calculated identity string, prekey value, and an email address to one of many Parcae intermediate servers. The server initiates a blockchain transaction that both records the client's registration information and sends a one-time verification email to the client. Once the client verifies their email, Parcae replies to the client with their unique Parcae public key. Figure 3 depicts the Parcae client registration process as a protocol diagram. Upon successful registration, the client stores both their identity string and Parcae public key in a configuration file needed to identify themselves to Parcae. At this point, the client can begin to query Parcae as a PRF service. Upon successful registration, Parcae's cryptographic peers additionally write the new client's identity string, prekey value, and public key to the Parcae blockchain's private data store. This ensures that Parcae's auditing peers receive a registration block that contains hashed versions of the new client's identity string, prekey value, and public key. With these hashes, the new client can verify that the Parcae blockchain did indeed record all of their registration information correctly.

We note that during registration, Parcae does not need to store a secret key for a newly registered client. This is because Parcae calculates a client's specific secret key as needed by cryptographically combining their identity string and their prekey value with a central secret key known only to Parcae's cryptographic peers.

PRF Queries. To request a PRF query from Parcae, a successfully registered client first hashes the password that they wish to harden and blinds the hash by exponentiating it with a blinding constant–a secure random integer. The client then sends their identity string, the blinded hash, an unblinded salt value, and optionally a proof request to one of many Parcae intermediate servers. The server sends this information to Parcae's cryptographic peers, which calculate a hardened blinded hash by constructing a bilinear pairing of the client's blinded hash and salt value, exponentiated with the calculated client-specific secret key. This hardened blinded hash is then sent back to the client, alongside a verifiable zero-knowledge proof with respect to the client's Parcae public key if requested. Because bilinear pairings preserve exponentiation, the client can then unblind the PRF output by exponentiating it with the inverse of the calculated blinding constant. This use of blinding and unblinding ensures that the final output is deterministic while preventing Parcae from learning anything about passwords passed to it.

Rate Limiting Queries. Rate limiting of PRF queries is performed on a per-user basis, differentiated by the unblinded salt values passed to queries. We assume that salt values are unique to each user since use of unique per-user salts is standard security practice. To perform rate limiting, Parcae records a rate limiting table that is indexed by salt values and stores both a timestamp and count of queries made. Whenever a query is requested, if the rate limiting table shows that for the given salt value, the count of queries exceeds a defined maximum query threshold, the query will be denied and the incident will be reported to the corresponding client. Otherwise, Parcae executes the query as normal and updates the respective timestamp and query count within the rate limiting table. This table is purged on a per-hour basis so that legitimate users should not experience an authentication lockout due to rate limiting.

Generating Authorization Tokens. When a registered client requests a Parcae secret key rotation, they must first obtain an authorization token as to prevent malicious actors from arbitrarily requesting key rotations for legitimate clients. To do this, the client sends their identity string to one of many Parcae intermediate servers. The server then instructs Parcae's cryptographic peers to generate a single-use time-sensitive authorization token and email it to the client's registered email address. The client can then provide this authorization token to request a Parcae secret key rotation.

Generating Update Tokens. After obtaining an authorization token, a client can send their identity string, a freshly calculated prekey value, and their authorization token to one of many Parcae intermediate servers to request a secret key rotation. The server initiates a blockchain transaction that first verifies the client's authorization token. If successful, the cryptographic peers calculate the client's old secret key one last time before replacing their old prekey value with the freshly calculated one. The cryptographic peers then generate the client's new secret key using their new prekey value. From both secret keys, an update

token is generated by multiplying the new secret key by the inverse of the old secret key. The cryptographic peers lastly determine the client's new Parcae public key relative to their new secret key, returning both the new public key and update token to the client. The client can utilize this update token to update previously stored hardened password hashes to be encrypted under their new secret key.

Applying Update Tokens. Update tokens can be applied to existing hardened password hashes entirely offline. To do this, a client simply exponentiates a hardened password hash with a Parcae provided update token. Because bilinear pairings preserve exponentiation, this simultaneously undoes the hardened password hash's exponentiation with the client's old secret key and applies an exponentiation with the client's new secret key. This updated hardened hash is now equivalent to the hardened hash obtained by querying Parcae under the client's new secret key.

Auditing Parcae. In Parcae, any actor can request to operate an auditing peer. They need not already be registered as a client since clients should be able to audit and ensure the validity of their own registration. To accomplish this, the auditor sends their request to one of many Parcae intermediate servers, and the server registers a new auditing peer with the Parcae blockchain. The server then distributes the new auditing peer's cryptographic certificates to the auditor. The auditor then installs publicly available Hyperledger Fabric peer images and joins their peer to the Parcae blockchain network. Once the auditing peer joins the network, receives blocks containing the entire Parcae blockchain ledger (with private data stored in a hashed format), and the auditing peer continuously receives all new blocks of data as they are generated. This allows the auditor to both verify the integrity of the Parcae network and validate hashes of transactions which are initiated by them.

6 Performance Evaluation

In order to analyze the performance of Parcae, we deploy both a Parcae blockchain and a set of intermediate servers on Amazon Web Services (AWS) to test the system's latency and throughput. All Parcae components are hosted in the same datacenter region of AWS. The intermediate servers are deployed on an Amazon Elastic Compute Cloud c4.large instance. The Parcae blockchain is deployed on an Amazon Managed Blockchain with a Hyperledger Fabric backend consisting of one to fifteen c5.4xlarge cryptographic peers.

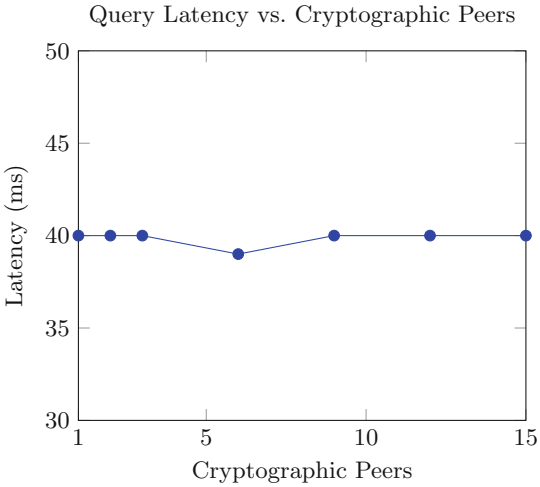

Fig. 4. Parcae query latency vs. number of cryptographic Peers

Latency. To measure the end-to-end latency of a Parcae PRF query, we time the full duration of the Parcae protocol, including client-side blinding and unblinding, checking rate-limiting, performing cryptographic operations, and making all connections between the client, intermediate server, and Parcae blockchain. We find that Parcae achieves an average individual query latency of 40 ms per query with just a single cryptographic peer. Figure 4 shows that as the quantity of cryptographic peers increases, individual query latency remains relatively constant.

Throughput. To measure the total throughput of an individual Parcae cryptographic peer, we measure the total quantity of PRF queries that a peer can process per second while receiving requests parallelized across multiple intermediate servers. From our testing, we find that a single Parcae cryptographic peer can achieve a throughput of up to 498 queries per second. To increase throughput and support more intermediate servers, Parcae dynamically scales its cryptographic peers in quantity to linearly improve the total service throughput. Figure 5 shows our measured linear relationship between the total system throughput and the quantity of cryptographic peers.

Discussion of Performance. Because Parcae strengthens the security measures of past implementations of PRF services and password hardening schemes by using an audited blockchain back-end, Parcae is inherently a slower protocol than past implementations. We find that Parcae's performance metrics are roughly within a factor of two and a half of Pythia's [4] when tested on comparable AWS instances. Despite this, we believe that the performance of Parcae is more than satisfactory to support its use as a practical publicly distributed PRF service. A user incurs only an additional 40 milliseconds between entering their credentials

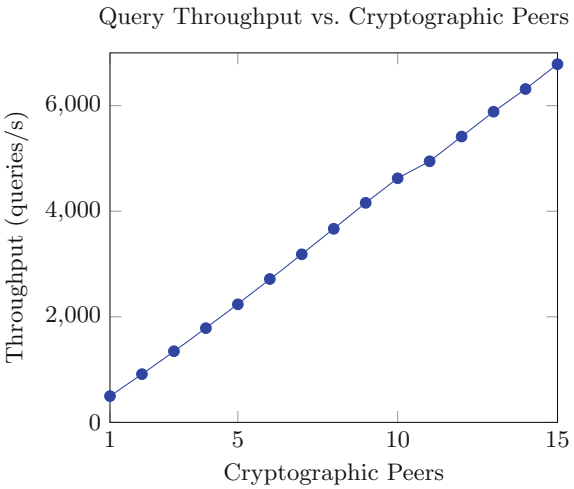

Fig. 5. Parcae total query throughput vs. number of cryptographic peers

and logging in to a client's service; this latency is typically far less than the network latency incurred by connecting to a remote client's service. Additionally, Parcae's throughput is more than reasonable with respect to scalability. With a Parcae blockchain consisting of just a single cryptographic peer that handles 498 queries per second, over 43 million Parcae queries can be processed in a single day. At this throughput, it would take a single Parcae cryptographic peer just over two days to harden the passwords of every one of the 98 million Facebook users that registered new accounts during the second quarter of 2020 [3]. With this level of throughput, it is practical to scale Parcae's cryptographic peers to meet the needs of all of Parcae's clients. For these reasons, we believe that Parcae's performance is both reasonable and practical for deployment in a publicly distributed setting.

7 Conclusions

We proposed the design and implementation of Parcae, a PRF service that fully realizes the needs of a dynamically sized client base in a publicly distributed setting. While past implementations of PRF services have focused on formalizing security definitions and improving efficiency, none have achieved the dynamic scalability and service auditing capabilities required to provide the trust and availability that is needed in a publicly distributed setting. We presented an implementation of Parcae that utilizes the security guarantees of a private permissioned blockchain to create a trustworthy and continually available PRF service. We showed that our system Parcae strengthens the security guarantees of past implementations of PRF services and password hardening schemes to protect against adversaries unique to a publicly distributed deployment scenario.

We lastly showed that our implementation of Parcae is both practical and reasonable in terms of query latency and scalable throughput.

References

1. Binary1985: Gigabyte RTX 2080ti Hashcat benchmarks. GitHub (2018). https://gist.github.com/binary1985/c8153c8ec44595fdabbf03157562763e
2. Brost, J., Egger, C., Lai, R.W.F., Schmid, F., Schröder, D., Zoppelt, M.: Threshold password-hardened encryption services. In: Proceedings of the 2020 ACM SIGSAC Conference on Computer and Communications Security, CCS 2020, pp. 409–424. Association for Computing Machinery, New York, NY, USA (2020). https://doi.org/10.1145/3372297.3417266
3. Clement, J.: Number of monthly active facebook users worldwide as of 3rd quarter 2020. Brandwatch (2020). https://www.statista.com/statistics/264810/number-of-monthly-active-facebook-users-worldwide/
4. Everspaugh, A., Chaterjee, R., Scott, S., Juels, A., Ristenpart, T.: The pythia PRF service. In: 24th USENIX Security Symposium (USENIX Security 15), pp. 547–562. USENIX Association, Washington, D.C. (2015). https://www.usenix.org/conference/usenixsecurity15/technical-sessions/presentation/everspaugh
5. Fabric, H.: A blockchain platform for the enterprise. Hyperledger-Fabric (2020). https://hyperledger-fabric.readthedocs.io/en/release-2.2/
6. Inc, A.: ios security. Apple (2016). https://www.apple.com/business/docs/iOS_Security_Guide.pdf
7. Lai, R.W.F., Egger, C., Reinert, M., Chow, S.S.M., Maffei, M., Schröder, D.: Simple password-hardened encryption services. In: 27th USENIX Security Symposium (USENIX Security 18), pp. 1405–1421. USENIX Association, Baltimore, MD, August 2018. https://www.usenix.org/conference/usenixsecurity18/presentation/lai
8. Lai, R.W.F., Egger, C., Schröder, D., Chow, S.S.M.: Phoenix: rebirth of a cryptographic password-hardening service. In: 26th USENIX Security Symposium (USENIX Security 17), pp. 899–916. USENIX Association, Vancouver, BC, August 2017. https://www.usenix.org/conference/usenixsecurity17/technical-sessions/presentation/lai
9. Muffet, A.: Facebook: password hashing and authentication. Real World Crypto (2015)
10. Schneider, J., Fleischhacker, N., Schröder, D., Backes, M.: Efficient cryptographic password hardening services from partially oblivious commitments. In: Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security, CCS 2016, pp. 1192–1203. Association for Computing Machinery, New York (2016). https://doi.org/10.1145/2976749.2978375
11. Verizon: Data breach investigations report (2019). https://www.key4biz.it/wp-content/uploads/2019/05/2019-data-breach-investigations-report.pdf