DIT: De-Identified Authenticated Telemetry at Scale

Sharon Huang Subodh Iyengar Sundar Jeyaraman Shiv Kushwah Chen-Kuei Lee Zutian Luo
Payman Mohassel Ananth Raghunathan Shaahid Shaikh Yen-Chieh Sung Albert Zhang
Facebook Inc.

***Abstract*—Logging infrastructure is a crucial component of any modern service deployed over the internet, particularly with mobile apps. Performance and reliability issues are better understood and fixed through logging, and most often only require aggregate statistics and other non-identifying pieces of data. Redesigning these frameworks to upload logs without user identifiers provides a defense-in-depth protection and mitigates risks of accidental logging and misuse. The unfortunate downside is the opportunity for malicious actors to corrupt, spam, or bias these logs, which is exactly what our solution prevents.**

De-Identified Telemetry (DIT)[1] is intended to be a privacy-focused fraud resistant logging system built using Verifiable Oblivious Pseudorandom Functions (VOPRFs). We report on the first large scale test of this service, scaled to support billions of clients. We discuss requirements that inform our choice of algorithms and designs including careful considerations around system architecture, reidentification resistance, rate limiting, and denial-of-service protections to balance practical anonymity and reliability requirements at scale.

We also motivate and develop attribute-based VOPRFs for use in efficient and transparent key rotation strategies. We design and prove the security of the first pairing-free variant of these VOPRFs at speeds competitive with pairing-based solutions (key derivation in a few ms and oblivious evaluations in under 1 ms). This new scheme is compatible with existing cryptographic libraries shipped with most mobile applications and hence suitable for wider deployment wherever pairings are unavailable.

I. Introduction

Logging infrastructure is a crucial component of any modern application or service. Usage and telemetry data helps developers and product owners evaluate performance and reliability, improve product features and generate reports. In a typical logging scenario every piece of data has identifiers attached to it that are used to authenticate the upload request. In many cases, however, these identifiers are not needed for the downstream analysis which consists of aggregate queries and reporting.

De-identifying data after collection is challenging and prone to mistakes in large scale applications. Data travels through a range of services and distributed infrastructure where it runs the risk of accidental logging and exposure. De-identifying data at the point of collection provides a more private and transparent framework for logging. As a result, the logging requests from clients cannot contain anyone's identity or identifiable information. Doing this naively could leave open an unauthenticated channel prone to potential abuse, which is exactly why we addressed this by providing a de-identified and authenticated channel.

The key idea behind our work is to collect analytics data from devices (or "clients") in a way that is de-identified, but also authenticated, so that we are also proactively protecting our systems. Inspired by the Privacy Pass protocol [1] and the long line of work on blind signatures and anonymous credentials [2, 3, 4, 5, 6], we design and test a new logging framework to enable logging without identifiers while simultaneously ensuring that only requests from legitimate clients are accepted. Privacy Pass-like protocols based on using Verifiable Oblivious Pseudorandom Functions (VOPRFs) were previously used to prevent service abuse from third-party browsers; which distinguishes them from our adaption of similar techniques for large-scale logging frameworks.

At a high level, we solve this problem by splitting the logging workflow into two distinct steps: First, clients use an authenticated connection to the server to obtain an anonymous token in advance. Then, whenever the clients need to upload logs, they send the anonymous token along with the logs in an unauthenticated connection to the server. The anonymous token serves as proof that the client is legitimate.

This new usage and the scale at which the overall system needs to operate raise unique challenges that our work seeks to understand and address. A few examples are the limits on the client application size which restrict the cryptographic algorithms at our disposal, balancing latency and token reusability; the trade-offs between unlinkability and utility and the impact on integrity when restricting network side channels.

We also revisit the question of verifiability in Privacy Pass with the goal of reconciling key transparency and key rotation in a seamless and efficient manner. To do so, we design and explore the deployment of a new pairing-free cryptographic construction called attribute-based VOPRF that is of independent technical interest for other applications of VOPRFs and Privacy Pass.

De-Identified Telemetry (DIT) is built as a scalable standalone infrastructure that can enable more private, verifiable, and authenticated communication between client applications and servers. The privacy benefits of such an infrastructure can be further strengthened using techniques such as local differential privacy (adding noise to data being collected), and global differential privacy (adding noise to aggregate analysis). We are actively exploring these extensions.

[1]DIT was formerly named PrivateStats.

Our Contributions. We design and build DIT, an improved data collection pipeline with strong privacy and integrity safeguards, powered by an anonymous credentialing service, and test it on hundreds of millions of mobile clients. Our contributions include specific design choices to tackle both deployment and scaling concerns such as:

1) The design and deployment of an anonymous endpoint collection service scaled to support large global traffic (up to 200k QPS peak traffic) that might be of independent interest.
2) An extension of the Privacy Pass design that separates the credential service from the application using it for better isolation, security, and privacy guarantees.
3) The use of careful client-side strategies such as batch and randomized credential fetching and limited credential reuse to minimize latencies and balance privacy concerns. As a result, we support sub 1-second end to end latencies for the vast majority of traffic with high internet speeds.
4) Protective measures against side channel reidentification risks, such as IP stripping at the edge, disallowing connection pooling, and measuring the risk of identifiability based on the collected data itself through integration of kHyperLogLog techniques [7] to our data infrastructure.
5) A revised integration of our abuse and denial-of-service protection techniques in light of anonymous collection and side-channel protections (e.g., IP stripping), such as rate limiting through key rotation.

Attribute-based VOPRFs. Motivated by the need for regular key rotation for stronger security and abuse protection, and the challenges with doing so efficiently and transparently, we design and formally prove a new pairing-free attribute-based VOPRF scheme. This enables seamless and transparent key rotation and verification that improves and complements current practices such as storage and retrieval from a public URL. Moreover, the new scheme is compatible with existing cryptographic libraries shipped with most mobile applications which currently do not support pairing-friendly curves. We demonstrate efficiency of this new scheme through benchmarks and experimental deployment in production environment with computation overheads that are competitive with plain schemes as well as pairing-based ones.

II. Preliminaries

Notation. For a function $f : X \to Y$, $\mathrm{Dom}(f) = X$ and $\mathrm{Range}(f) = Y$. $[n]$ denotes the set $\{1, \ldots, n\}$. For $x \in \{0,1\}^n$ and $1 \leq i \leq n$, $x[i]$ denotes the i-th bit of x. $\mathrm{negl}(x)$ denotes a negligible function that is $o(1/x^n)$ for all $n \in \mathbb{N}$. $a \leftarrow S$ denotes the uniform sampling of a from the set S. The notation $\mathcal{A}^f$ denotes an algorithm $\mathcal{A}$ that has black-box (or oracle) access to the function f, with no information about its internals. Efficient algorithms run in time polynomial in their inputs and a security parameter λ (that is often implicit).

A. Setup and Threat Model

Our setup comprises a large number of *clients* that want to report de-identified statistics to a *Data Collection Service*, also referred to as a *Server*. Clients can identify and hence authenticate themselves as valid clients to the Server using some identifying information they share with the Server (such as username/password, cookies, etc.) or through other means (such as device fingerprinting). Clients and servers can communicate via two endpoints. Communication between clients and servers are always encrypted and authenticated (using standard systems such as TLS) but one of the endpoints is designed to be anonymous, i.e., it is not tasked with validating clients that connect to it and does not store any information about the source of the messages it receives (see Section III for how we implement such an endpoint in practice).

We consider two adversaries in the system. First, *rogue clients* that are either compromised or fake clients whose goal is to sabotage the data collection process by injecting fake data into the service. Second, *a semi-trusted* server that aims to deviate from its intended behavior so that it may break the privacy of clients.

Security Goals. Against rogue clients, we design our system to guarantee that clients that are not previously authenticated cannot contribute in any manner to the data collection process. More precisely, any data they send to the anonymous endpoint will be detected and rejected. We mitigate the effect of once authentic but subsequently compromised clients by enabling the servers to rotate keys. That forces clients to authenticate themselves on a frequent basis.

Against the semi-trusted server, we design our system to guarantee that the server cannot use information it possesses for authenticating clients to identify or link the data collected via the anonymous endpoint to any specific client. We also offer a tradeoff between communication and service overhead, and linkability—servers can link *a few* data collection events to each other, but not to any specific client.

Security Non-Goals. We do not protect against authentic clients that choose to upload statistics or other information that is fraudulent. Any information given by a client that is deemed authentic is itself considered accurate. We do not protect against servers deploying this service to a very small set of clients where using auxiliary information such as timing or other side-channel information can plausibly identify clients even through the anonymous endpoint. We test our system in WhatsApp's mobile app which inherently offers a huge pool of users within which individual clients can anonymize themselves.

Finally, we do not offer any secrecy to the logged *data* being uploaded beyond TLS protection which ends at the server endpoints. The logged data itself may contain identifying information but we have built into our system flexibility to address different use cases that could require different levels of privacy (some require device information such as OS or device-type, others might require more information such as country codes for policy reasons). As noted earlier, our

system design can be extended to add local noise to reports offering differential privacy guarantees similar to [8, 9, 10] and ensure some notion of privacy to even the data uploaded by participants.

B. Cryptographic Building Blocks

In this section, we briefly introduce the cryptographic building blocks used in this paper. The first notion we require is a pseudorandom function (PRF). A keyed function $F(k, x)$ is a PRF if the function instantiated with a random key is computationally indistinguishable from a random function over the class of functions $\mathcal{F}$ from $\text{Dom}(F) \rightarrow \text{Range}(F)$. More formally, for security parameter λ, for any efficient adversary $\mathcal{A}$, $\mathbf{Adv}_{\text{prf}}(\mathcal{A})$ defined as $|\Pr[\mathcal{A}^f(1^\lambda) = 1 : f \leftarrow \mathcal{F}] - \Pr[\mathcal{A}^{F(k,\cdot)}(1^\lambda) = 1 : k \leftarrow \mathcal{K}]|$ is $\text{negl}(\lambda)$.

Verifiable Random Functions. Verifiable Random Functions (VRFs), introduced by Micali et al. [11], is a PRF whose outputs can be *verified* given a public key corresponding to a secret key. More formally, for $y = F(sk, x)$, there is a proof π_x that proves (along with a pk) that y is evaluated correctly. The pseudorandomness property requires that the output of $F(sk, x^*)$ on a targeted input x^* remains computationally indistinguishable from a random element in $\text{Range}(F)$ even given $(F(sk, x), \pi_x)$ on adversarially and adaptively chosen inputs x (not equal to x^*, of course). A more formal treatment of the security definition, and the definition of a $\mathbf{Adv}_{\text{vrf}}(\mathcal{A})$ is deferred to Appendix A.

Verifiable Oblivious Random Functions. Verifiable Oblivious Random Functions (VOPRFs) [12, 13] additionally require that there is a protocol that allows one party (client) with input x and pk to evaluate $F(sk, x)$ with another party (server) holding sk, such that this is (a) *oblivious*, i.e., the server learns nothing about x, and (b) *verifiable*, i.e., the client can verify with pk that the output of the oblivious evaluation is correct.

Non-Interactive Zero-knowledge Proofs. Given a Prover with secret information w (or a witness) and a Verifier with x, a non-interactive zero-knowledge proof (NIZK) π is a proof that convinces a Verifier that x satisfies a specific property involving w without leaking any information about w.

For this paper, we require NIZKs on a very specific property of *discrete log equality* [14] that is true for a tuple (g, h, u, v) if there is an α such that $h = g^\alpha$ and $v = u^\alpha$. Camenisch and Stadler [14] construct a NIZK for this property as follows: The prover picks r and computes $t_1 = g^r$, $t_2 = u^r$, $c = H(g, h, u, v, t_1, t_2)$. The proof $\pi = (c, s = r - c \cdot \alpha)$, which can be verified by checking that $c = H(g, h, u, v, g^s \cdot h^c, u^s \cdot v^c)$. In the Random Oracle model [15], where H is modeled as an ideal hash function, it can be shown that this construction is a NIZK for discrete log equality. In the rest of the paper, we refer to this proof in shorthand as DLEQ-$\Pi(g, h, u, v)$.

Discrete-Log VOPRF. In this paper, we use the construction of Jarecki et al. [13] who show that the function $F(sk, x) := H(x)^{sk}$ with a public key $pk = g^{sk}$ that is combined with a discrete log equality proof is a VOPRF in the random oracle model under the Diffie-Hellman assumption (Appendix A). Oblivious evaluation blinds input x as $H(x)^r$ for random r (which can be unblinded by raising to $1/r$). The VRF proof is simply DLEQ-$\Pi(g, pk, H(x), H(x)^{sk})$ with or without blinding applied to $H(x)$.

Hash Functions in the Random Oracle Model. We use the random oracle [15] methodology when analyzing security proofs of cryptographic constructions. If left unspecified, the hash function $H(\cdot)$ is chosen to have an appropriate domain and range, and in practice, there are distinct hash functions $H_1, H_2, \ldots$ with appropriate outputs that are applied to hashes of inputs, hashes of credentials, and hashes used in zero-knowledge proofs (such as DLEQ-Π above).

III. DIT: Design and Testing at Scale

The design of our system is inspired by the Privacy Pass protocol [1] and comprises the following components: (1) A client component that would be shipped with the mobile app. (2) An anonymous credential service (ACS) that runs in a distributed manner in a production fleet. (3) An authentication server whose primary responsibility is to authenticate clients as real users. (4) An application server responsible for receiving logs.

DIT is designed to offer de-identified authenticated logging on behalf of device clients that is resilient to fraudulent reports. Moreover, we ensure that the authentication server, responsible for verifying client identity, is separate from the credential service. This enables us to rate-limit clients (more details to follow) without leaking any information to the credential service. The use of a VOPRF guarantees that the service itself cannot link client identities across the two phases of the protocol. Finally, the application service receiving logs relies on the ACS to verify the authenticity of the reports received, but does so in a manner that does not expose any of the logged information to the credential service. This is an extension of the design of Privacy Pass that only allows a boolean check to claim access to a resource.

Furthermore, by design, our system reports are fully de-identified. This precludes common logging scenarios where linked logs are useful for debugging and other purposes. To facilitate such use cases, clients are responsible for including sufficient information in the logs to link requests, if needed. By completely separating the reporting layer from any additional functionality, we can focus on strengthening anonymity and integrity guarantees.

The protocol proceeds in two phases. In the Request phase, the client component chooses a random input and sends the client identity and a blinded input to the authentication server that verifies the client identity and forwards the blinded input to the ACS. The ACS responds with a signed credential corresponding to the input that is forwarded down to the client component which unblinds the credential. The client component (optionally) verifies that the credential is valid. We use the phrases (anonymous) credential, signed credential, or tokens interchangeably in the paper.

In the Logging phase, the client authenticates any logging message it wants to send by deriving a shared key from the input and the credential and constructing an authentication tag as a MAC on the message with the shared key. The client contacts an application server anonymously with a report comprising the credential input, the message, and the authentication tag. To verify the authenticity of the report, the application server forwards the credential to the ACS, which checks that it is valid, and returns the MAC key corresponding to the credential. This design allows us to hide the input itself from the credential service. The report is authenticated by the application server using the MAC key to verify the message. In what follows, we describe these components in detail. Figure 1 represents the protocol flow and Figure 2 has the cryptographic details of the protocol.

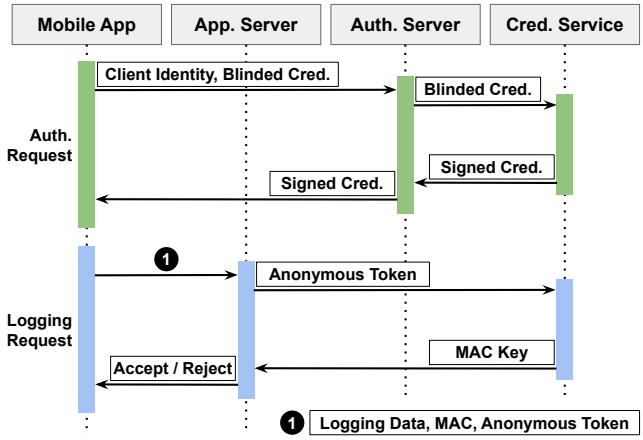

Fig. 1: Protocol flow diagram.

Setup

$(\mathbb{G}, g, h := sk \cdot g, H : \{0,1\}^* \to \mathbb{G}, H_2 : \{0,1\}^* \to \{0,1\}^{128})$.

Authentication Request

1) Client: $r \leftarrow \mathbb{Z}_q, x \leftarrow \{0,1\}^{128}$, sends $(ID, H(x)^r)$.
2) Cred. Service: Receives $y := H(x)^r$ and sends $(z := y^{sk}$, DLEQProof$(g, h, y, z))$.
3) Client: Recovers credential $(x, f(x) := z^{(1/r)} = H(x)^{sk})$.

Logging Request

1) Client: Derives shared key $ssk = H_2(x, c)$ from credential (x, c) and sends $(x, m, t := \text{HMAC}_{ssk}(m))$.
2) App Server: Forwards x to Cred. Service and receives $ssk' := H_2(x, f(x))$. Accepts log m iff $t = \text{HMAC}_{ssk'}(m)$.

Fig. 2: Protocol details. $\mathbb{G}$ is an elliptic-curve group of order q and DLEQ-Π is defined in Section II-B.

A. Client Component

WhatsApp has more than 2 billion users and more than 100 billion messages are sent through WhatsApp every day. Our mobile app fetches the credentials from our server using an authenticated TLS connection, caches the credentials, and then uses cached credentials for logging data to the server. In building DIT into our mobile app, we dealt with several challenges detailed below.

Algorithm choice. The protocol (as well as Privacy Pass) requires the use of prime order groups, a popular choice of which is Ristretto [16]. The client application already has bundled with it support for the x25519 curve used for message encryption. Adding Ristretto support would incur significant overhead with regards to code size, verification, and deployment, and mobile deployment is sensitive to code sizes. We use x25519 instead but mitigate small subgroup attacks by checking if points are on small subgroups before using them. Other threats such as malicious clients executing static DH attacks [17, 18] are mitigated by limiting the number issued credentials before rotating keys [19].

Reliability and efficiency. Several careful considerations go into ensuring the reliability and efficiency of such a system at this scale. With thousands of QPS in peak traffic (see Section V-B for more details) any errors in credential generation will lead to logging failure or add latency that would be harmful to user experience. As this system is built upon existing logging systems that have been tuned over the years to offer a high degree of reliability, we worked on optimizations to the system while fetching credentials to minimize risks. We allow credentials from the client to be reused a small number of times before they are rejected. We randomize when clients fetch credentials to eliminate the possibility of a thundering herd problem. Finally, we allow clients to issue batched requests, with additional batching between the authentication server and the credential service which reduces the number of remote procedure calls required during peak volumes.

Connection Unlinkability. Our mobile app uses the network APIs provided by the mobile platforms in order to log data. However, on some platforms we do not control the connection pooling logic that is implemented at the system level. For certain implementations, it might be possible for the system to use the same connection with other unrelated requests that might arise to our services thanks to connection pooling. To prevent this from happening, we use a dedicated domain name for receiving these requests so that systems that pool connections would be forced to open up separate and new connections for the logging requests.

B. Anonymous Credential Service

The anonymous credential service (ACS) is deployed globally in our data centers to serve traffic from across the world and minimize overall latencies. It is designed to be a standalone service that can serve many use cases and applications in the future. The service is written in C++ and provides an RPC interface [20] used across the company so that application servers can conveniently call into it from different languages. Our motivation for having ACS be a separate service was to isolate the key material for ACS and provide access control and key isolation between multiple tenants as well as a convenient service to be able to rotate and manage keys. Moreover, this separation allows us to design ACS in a manner

that is not exposed to any application specific data. Thus far we have tenants written in Hack as well as Erlang that use ACS via the RPC system.

ACS is built upon several smaller services: a service for managing keys, an internal configuration management system by which we deploy new configurations to ACS, and a counting service for replay protection and rate limiting requests.

Configuration Management System. For each use case supported by ACS, we create a structured configuration file, maintained and reliably pushed to all the ACS server machines. The file covers operational configurations of a ACS use case, including access control list, key rotation schedule, etc. To prevent the configuration from being mutated by an unauthorized party, the management system enforces permission checks upon all the read and write attempts along with change logs for audits.

Counting Service. Malicious clients might try to take advantage of ACS by requesting a large number of credentials or attempting to redeem a credential several times. To prevent such behavior, a real-time, reliable, and secured counting service is important for the overall security of the system. With application servers in datacenters all over the world, ACS can receive requests to issue as well as redeem credentials from different datacenters, and our solution handles this distributed workload securely. We built a counting service that is backed by a highly reliable key-value memory cache service that is widely used in the company [21]. To ensure that only ACS can mutate and read counters from this key-value cache, we use an access control mechanism. We make sure that we provide globally consistent query results by using atomic transactions so that malicious or accidental redemption or issuance of tokens from different datacenters can avoid race conditions.

Key Management Service. The key management service interacts with the configuration management system to mutate key materials for ACS tenants according to the ciphersuites and key rotation schedules specified in their configuration files. Key management plays a crucial role in ensuring that we can mitigate effects from clients compromised after credential issuance. This is done by rotating keys frequently and discarding reports from old keys. There are challenges in deploying key rotations across a fleet in a consistent manner. Additional challenges around distributing new verification keys to clients that would like to verify credentials also need to be tackled. Systems such as Privacy Pass simply publish new keys at a specific URL. This need for efficient and transparent strategy for key rotation motivates our design of attribute-based VOPRFs, with more details to follow in Section IV.

C. Anonymous Reporting Endpoint

An important component of our solution is the anonymous reporting endpoint that was deployed to handle the scale of incoming reports. The defensibility of the privacy guarantees depend on how this endpoint is built and deployed. We build this by offering a globally available independent web endpoint that can be reached by an unauthenticated (to the company) HTTPS connection from the client. The reporting endpoint has two logical components:

- The network channel that is established once the client presents valid (anonymous) credentials.
- A Logging Layer built on top of this channel that receives and processes logging data.

The credential check layer is separated from the logging layer which allows them to evolve independently. Ideally, variations and upgrades to logging practices should be seamless to the credential check experience. In practice, for simplicity of deployment, we use the same HTTPS POST request to attach the credential in-situ, as an additional parameter to a log message. This allows us to establish the anonymous channel and log data together and to use existing HTTPS libraries provided by the platform as much as possible to reduce application size.

As stated earlier, in our Security Non-Goals, the channel is purely based on the anonymous credential with no identifying information, and any identifying information for functionality must be passed in with the data and processed by the logging layer. For example, events around crash logs might contain identifying information around version numbers where the privacy-utility tradeoff favors more information to enable quicker debugging and performance improvements. Other uses, such as broad aggregate statistics will effectively have little more than the metric payload sent to the logging layer. Other use-cases necessitate unique designs: for counts of distinct users the client writes a special identifier that is consumed by the logging layer. The map of user ID to this identifier is known only to the client app and is frequently rotated.

Building such a logging system faithfully came with several challenges we detail below.

1—Re-identification resistance. While our credentials create an anonymous reporting channel, there are several other channels by which identity could leak, which we worked to protect. One such channel is meta-data, such as IP address, session tickets, or connection pooling. In order to avoid these channels from accidentally being used in the reporting endpoint, we use a separate domain name for the reporting endpoint and we strip IP address identifiers at our Layer 7 load balancers [22]. This prevents the client IP address from reaching our application server endpoint.

As discussed earlier, information in the logs themselves could be used to re-identify users. We use several methods to mitigate these risks. For example, we have a structured client logging system that only allows whitelisted fields to be logged. We also periodically monitor the re-identifiablity and joinability potential of each field that is logged using kHyperLogLog [7]. First, we measure the re-identification potential of each field. We then investigate fields with re-identification potential higher than a configurable threshold for actual re-identification risks. For example, we check if the fields could potentially be joined with fields that we collect through our other telemetry systems. If we detect any actual

risk, we drop the fields from being stored on the server and push out client updates to stop sending those fields to the server. We talk about some of the deployment challenges of this system in Section V-B. In addition to monitoring for re-identifiable data, the privacy benefits of such an infrastructure can be further strengthened using techniques such as local differential privacy (adding noise to data being collected), and global differential privacy (adding noise to aggregate analysis). We are actively exploring these extensions as well as extending re-identification detection to a combination of multiple fields.

2—Rate limiting. To prevent abuse of the anonymous reporting endpoint by a single user while offering the flexibility of reusing credentials *a few times*, we aim to rate limit a user from making too many requests. We cannot use any identifying information to rate limit users when receiving reports, thus our burden of rate limiting shifts to the point of issuing credentials and limiting the number of times a single token is used with the Counting Service discussed earlier. We limit the number of credentials issued per user by limiting the number of times a user fetches credentials and frequently rotating public keys. If n_f denotes the number of fetches before a client is disallowed, b denotes the batch size for each fetch, and t_r denotes the time period afte which we rotate public keys, the effective rate-limit becomes:

$$\text{rate limit} = \frac{n_f \cdot b}{t_r}.$$

Decreasing t_r increases the rate limit but comes at the tradeoff of communicating to clients that the new public keys are rotated legitimately and not as a means to fingerprint or identify specific clients. To overcome this challenge and offer greater flexibility without the need to worry about transparency, we developed attribute-based VOPRFs (Section IV) and plan to deploy them in the future.

3—Denial of service. Stripping client IP addresses at our load balancer prevents accidental use of this data for re-identification, but it also potentially limits our ability to preventing Denial of Service (DoS) attacks on ACS. The service performs several cryptographic operations and is CPU bound. IP address is an important signal to both detect and remediate DoS attacks. Common DoS prevention mechanisms usually drop traffic in order to deal with a DoS attack and are effective only if they are able to shut down attacker traffic without shutting down other forms of legitimate traffic which is hard to do without the IP address. To address this problem, we moved DoS protection from our application servers to our load balancers which retain access to IP address information. This allows us to use a variety of strategies and signals in limiting DoS from dropping packets to dropping requests. We stress the importance of a careful design integrating DoS protection and the anonymous endpoints to avoid catastrophic failures. We detail some of our experiences further in Section V-B.

IV. Attribute-based VOPRFs

As discussed in earlier sections, frequent key rotations provide a rate limiting mechanism for the Anonymous Credential Service and also help minimize the risks associated with static DH attacks [17, 18]. But a malicious server could use these key rotations to segregate and de-anonymize users, e.g., by sending each client a unique key during credential issuance that can be tied back to the user during credential redemption.

Current Privacy Pass implementations [23] address this problem by placing the new keys in a public trusted location [24] accessible to clients and accompanying each OPRF evaluation with a ZK proof that the correct key was used. Frequent key rotations make this approach cumbersome, requiring regular key lookups by clients or complex Key Transparency solutions that keep track of evolution of these keys [25].

Transparent and efficient key rotation can be achieved using an *attribute-based VOPRF* scheme where new secret keys can be derived from attributes (such as strings in $\{0,1\}^\ell$), whose corresponding public keys can be derived from a single master public key. By setting the attributes to refer to the time epoch for which the keys are valid, client verification is easily achieved without the need to fetch new public keys. As a result, we can extend the transparency of the master public key—which can be shipped with client code or posted to a trusted location—to these derived public keys without any additional effort.

Moreover, while we focus on time epochs as attributes, given that attribute can be arbitrary strings one can associate and derive new keys for other types of attributes such as key policies or application identifiers which may be of interest for future use cases of Privacy Pass.

A. Attribute-based VOPRFs

An attribute-based VOPRF thus has two input components: an attribute that is used to derive a secret key from some master secret key, and an input, on which a PRF with this secret key is evaluated (obliviously, if required). In order to guarantee that VOPRF evaluations across different attributes are independently random, we require that the VOPRF outputs be pseudorandom as a function of the attributes as well. However, the VRF is not required to be oblivious in the attributes—in fact, to ensure keys are rotated correctly, the server must know what attribute it is deriving a key for. Therefore, attribute-based VOPRFs are in essence another instantiation of partially-oblivious verifiable random functions as described by Everspaugh et al. [26].

More formally, an attribute-based (AB-VOPRF) is a tuple: (MasterKeyGen, PKGen, PKVerify, Blind, Eval, Unblind, Verify), where the first three algorithms generate the master key, attribute-based public keys, and verify public keys. The client and server can engage in a partially oblivious evaluation with client running Blind, server running Eval, and the client recovering and verifying the PRF evaluation with Unblind and Verify respectively.

We describe the partially oblivious VOPRF from Everspaugh et al. [26] as an AB-VOPRF and then construct a new AB-VOPRF without the use of pairings (under generic Diffie-Hellman assumptions over elliptic curve groups).

KeyGen Algorithms

—MasterKeyGen(1^λ) : Output groups (of order $q \approx 2^\lambda$) with a pairing $\hat{e} : \mathbb{G}_1 \times \mathbb{G}_2 \to \mathbb{G}_T$ and hash functions H_1 and H_2 that map strings to $\mathbb{G}_1$ and $\mathbb{G}_2$ respectively. $msk \leftarrow \mathbb{Z}_q$ and $mpk = g_1^{msk}$.

—PKGen(t, msk) : Outputs $pk_t = H_1(t)$ and proof $\pi = \emptyset$ with $sk_t = H_1(t)^{msk}$.

—PKVerify(pk_t, t, π): is trivial; check that $pk_t = H_1(t)$.

The PRF

$F(sk_t, x) := \hat{e}(sk_t, H_2(x)) = \hat{e}(H_1(t), H_2(x))^{msk}$.

Oblivious Eval Algorithms

—Blind(x): Output $x' = H_2(x)^r$ for $r \leftarrow \mathbb{Z}_q$.

—Eval(sk_t, x'): Output (f', π), where $f' = \hat{e}(sk_t, x')$ and $\pi = \text{DLEQ-}\Pi(g_1, mpk, \hat{e}(pk_t, x'), f')$.

—Unblind(f', r): Recover $F(sk_t, x) = (f')^{1/r} \in \mathbb{G}_T$.

—Verify(pk_t, x, f', π): The discrete log equality proof π is checked in a straightforward fashion; the client knows g_1, mpk and f' and can compute $\hat{e}(pk_t, x')$ needed as input to the proof.

Fig. 3: Pythia's attribute-based VOPRF [26]. Here H_1 and H_2 are modeled as random oracles and DLEQ-$\Pi(g, h, y, z)$ is a random-oracle NIZK proving the equality of discrete logs of h w.r.t g and z w.r.t y.

B. A pairing-free AB-VOPRF

Pairing-friendly curves are not available in most standard cryptographic libraries and consequently are not bundled with most mobile applications. The additional complexity of shipping and managing new cryptographic algorithms and its sever implication on the application size make pairing-based instantiation of AB-VOPRFs a sub-optimal choice.

This motivated us to design and implement a new AB-VOPRF construction that works with standard pairing-free curves that are typically shipped with mobile applications to enable other features such as encrypting messages.

Before we proceed with this new attribute-based VOPRF, we note that a trivial construction of these VOPRFs for attributes from a domain of size N can be constructed by picking N independent public and secret key pairs corresponding to each attribute at setup time. The key sizes will therefore be $O(N)$ each with proofs of size $O(1)$. The construction below achieves attribute-based VOPRFs with key sizes and proofs that are $O(\log N)$ without pairings.

Naor-Reingold Attribute-based VOPRF. Our attribute-based VOPRF construction is inspired by the Naor-Reingold PRF [27] and takes attributes from $\{0,1\}^n$.

- MasterKeyGen($1^\lambda, n$): The algorithm chooses a group $\mathbb{G}$ of appropriate size and samples $a_0, a_1, \ldots, a_n \leftarrow \mathbb{Z}_q$ and $h \leftarrow \mathbb{G}$. We have: $msk = (a_0, \ldots, a_n)$ and $mpk = (\mathbb{G}, g, h, P_0 = g^{a_0}, h_1 = h^{a_1}, \ldots, h_n = h^{a_n})$. Also implicitly in the public parameters is a hash function $H : \{0,1\}^* \to \mathbb{G}$, hashing inputs to the group.
- PKGen(t, msk): The attribute-based public-key generation algorithm proceeds as follows. Set

$$P_i = g^{A(t)_i} \text{ where } A(t)_i = a_0 \cdot \prod_{j \le i} (a_j)^{t[j]}$$

denotes a subset product of a's where the bits of t are 1. Compute

$$\pi_i = \text{DLEQ-}\Pi(h, h_i^{t[i]}, P_{i-1}, P_i) \text{ for } i \in [n].$$

Set $pk_t = P_n = g^{a_0 \cdot \prod a_i^{t[i]}}$, the exponentiation of g to the subset product, and $\pi_t = (P_1, \ldots, P_{n-1}, \pi_1, \ldots, \pi_n)$, the sequence of partial product exponentiations and discrete log equality proofs. The secret key $sk_t = a_0 \cdot \prod a_i^{t[i]}$.
- PKVerify(pk_t, π_t): To verify public keys, verify the sequence of discrete log proofs starting with P_0 and concluding with $P_n = pk_t$.
- **The pseudorandom function.** The function evaluated is:

$$F(msk = (a_0, \ldots, a_n); (t, x)) := H(x)^{a_0 \cdot \prod a_i^{t[i]}} = H(x)^{sk_t}. \quad (1)$$

- Blind(x): Output $x' = H(x)^r$ for $r \leftarrow \mathbb{Z}_q$.
- Eval(sk_t, x'): Output a tuple (f', π_{eval}) where

$$f := (x')^{sk_t}; \quad \pi_{\text{eval}} := \text{DLEQ-}\Pi(g, pk_t, x', f').$$

- Unblind(f', r): Recover $F(msk, (t, x)) = (f')^{1/r}$.
- Verify($pk_t, x, f', \pi_{\text{eval}}$): The discrete log equality proof π_{eval} is checked in a standard fashion, given g, pk_t, x', and f'.

Correctness. Correctness of the oblivious evaluation protocol (Blind, Eval, and Unblind) follows in a straightforward manner from the commutativity of raising to powers r and sk_t. Correctness of both Verify and PKVerify follow from the proofs of correctness of the discrete log equality proofs. In particular, for the latter, the proofs proceed sequentially from the correctness of π_0 through π_n.

Security (Outline). The security notions for an attribute-based VOPRF are closely related to the security notions of a verifiable partially-oblivious PRF [26]. In this presentation, we skip the one-more unpredictability/pseudorandomness approach of Everspaugh et al. [26] but argue that the function F defined in Eq. (1) is a pseudorandom function in both inputs t and x. Further, we show that F is a PRF even after giving an adversary access to proofs for both PKGen and π_{eval} on the adversary's choices of inputs (except for the challenge input, naturally). This is almost identical to security notions for verifiable random functions [11, 28, 29].

Our proof proceeds in two steps. In the first step, we show *selective* security of the VRF, where the adversary chooses its target input (t^*, x^*) before receiving the parameters of the system. This can be converted to *full security* through a standard argument in the random oracle model.[2] The full proof is deferred to Appendix B, but we present an outline.

[2]This is done by hashing t and programming H at t^*, x^* with a q_H loss in the security level, where q_H is the number of queries to H.

The proof of pseudorandomness of F relies on the ℓ-exponent Diffie-Hellman assumption (ℓ-DHE), (stronger) variants of which have been discussed as the gap-Diffie Hellman assumption [28]. ℓ-DHE states that given $(g, g^{\alpha}, \ldots, g^{\alpha^{\ell}})$ for $\alpha \leftarrow \mathbb{Z}_q$ in an elliptic-curve group $\mathbb{G} = \langle g \rangle$, it is computationally infeasible to distinguish between $g^{\alpha^{\ell+1}}$ and a random element in $\mathbb{G}$. Bresson et al. [30] show a connection between the standard Diffie-Hellman assumption and a *group* Diffie-Hellman assumption that can be used to prove pseudorandomness of F in a straightforward manner following our proof outline, but at an exponential loss in tightness.

Recall the selective VRF security game: an adversary $\mathcal{A}$ submits the challenge query (t^*, x^*) to the challenger C and receives the public parameters mpk. $\mathcal{A}$ can query PKGen to receive pk_t for all t and evaluate F on all inputs other than (t^*, x^*). $\mathcal{A}$ also receives corresponding proofs from these queries. Finally, $\mathcal{A}$ receives an output y^* as the challenge output and must distinguish between whether $y^* \leftarrow \mathbb{G}$ or $y^* = F(msk; (t^*, x^*))$.

For $\ell = n$, given an ℓ-DHE challenge $(A_0 = g, A_1 = g^{\alpha}, \ldots, A_{\ell} = g^{\alpha^{\ell}}, A_{\ell+1} = g^{\alpha^{\ell+1}})$ or $A_{\ell+1} \leftarrow \mathbb{G}$, and (t^*, x^*), the challenger sets up msk and mpk as follows. Sample uniformly random values $r_h, r_0, \ldots, r_n \leftarrow \mathbb{Z}_q$ and (implicitly) let $a_0 = r_0 \prod(\alpha + r_i)$ for $t^*[i] = 0$. Also let $h = r_h \prod(\alpha + r_i)$ and $a_i = (\alpha + r_i)$ or $(\alpha + r_i)^{-1}$ for $t^*[i] = 1$ and 0 respectively. Given this, mpk values can be computed "in the exponent" using A_i's via exponentiations and multiplications. Finally, program $H(x^*) = A_1 = g^{\alpha}$.

This actually suffices to work out a proof. With a little work, we can show that all PKGen and F evaluations not on (t^*, x^*) always result in a value of the form $g^{p(\alpha)}$ for some polynomial $p(\cdot)$ of degree $\leq \ell$ and these can be computed "in the exponent" given only $A_1, \ldots, A_{\ell}$. Only when queried on (t^*, x^*) does that result in a value of the form $g^{q(\alpha)}$ for some $(\ell+1)$-degree polynomial $q(\cdot)$ where the real-or-random ℓ-DHE challenge $A_{\ell+1}$ can be embedded—leading to either a real PRF output or a random output depending on the nature of $A_{\ell+1}$ and completing the tight reduction to ℓ-DHE. Further details are fleshed out in Appendix B.

Attribute-based VOPRFs with Hashing. The Naor-Reingold AB-VOPRF can be extended by hashing the attribute first. We denote the resulting AB-VOPRF NR-hashed, and the original construction NR-plain when required to disambiguate between the two constructions. More formally, if $F_{\text{NR-plain}}(msk; (t, x))$ denotes the PRF in (1), we define:

$$F_{\text{NR-hashed}}(msk; (t, x)) := F_{\text{NR-plain}}(msk; (H_2(t), x)),$$

where $H_2 : \{0,1\}^* \rightarrow \{0,1\}^n$ is a hash function mapping strings to n bits.

NR-hashed improves upon NR-plain in two ways: first, as mentioned above, it guarantees *full security* through a standard random oracle argument (see Appendix B); second, it enables attributes t of an arbitrary nature (domain names, timestamps, VOPRF use-cases, etc.) without worrying about a map to $\{0,1\}^n$. NR-hashed however requires larger key sizes and correspondingly (linearly) longer compute times compared to NR-plain with small attribute sizes as $n \geq \lambda = 128$ for the security proof to go through. Selective security is largely sufficient for practical applications where adversarially targeted attributes will presumably be sufficiently independent of the parameters of the system (which is formally captured through NR-hashed). Therefore, for small and realistic attribute sizes ($n \leq 32$, say) NR-plain is much more efficient.

V. Experimental Results

A. Microbenchmarks

We implemented the attribute-based VOPRFs constructed in Section IV to evaluate the runtime of various functions. Our implementation uses the Ristretto implementation of curve25519 [16] and is written in C++ using `libsodium` [31]. Our microbenchmarks are evaluated on machines with Intel Broadwell processors with 24 cores and 64 GB RAM. As a baseline, nistp256 elliptic curve operations for ECDH reported by running `openssl speed ecdhp256` take 0.066 ms (15k operations/sec), as do the Blind and Unblind operations.

Table I contains the timing results of the following steps of the VOPRF construction: MasterKeyGen, PKGen with and without proofs, and PKVerify. The metrics are computed over 1000 runs. Section IV notes the challenges with deploying pairing libraries to clients, but we do note that pairing-based constructions are efficient. In particular, consider the AB-VOPRF in Fig 3 when run with Cloudflare's BN-254 [32] implementation [33]: the PRF evaluation with the pairing operation was benchmarked on our hardware at under 2ms / operation with MasterKeyGen (0.170ms), PKGen (10s of nanoseconds), and PKVerify (10s of nanoseconds) operations being incredibly efficient. What we show in Table I is that even without pairings, these operations can be efficient.

We also report on the simple VOPRF anonymous credential protocol from Privacy Pass to compare to our attribute-based VOPRF benchmarks. Table II contains these results with metrics averaged over 1000 runs. We report on two implementations (ristretto, for a fair head-to-head comparison) and x25519 (the implementation that is run in production).

Finally, we report on the communication overhead of the protocol (notwithstanding other encoding and network communication overheads) in Table III.

B. Testing at Scale

We are currently testing DIT in WhatsApp mobile clients. Given its more than 2 billion users, this is a significant volume and scale. At the peak, the service issues roughly 20,000 credentials per second and over 200,000 credentials per second are redeemed by clients. The current deployment implements simple VOPRFs with the ability for clients to fetch and verify public keys.

The server side components are deployed on machines located in our data centers across different regions all around the world, as application jobs managed by our efficient and reliable cluster management system [34].

Scheme	n	MasterKeyGen	PKGen w/o proof	PKGen w/ proof	PKVerify
NR-Plain	16	1.34 ms (2.27 ms)	0.23 ms (0.22 ms)	1.60 ms (2.49 ms)	3.06 ms (4.79 ms)
	32	2.83 ms (4.42 ms)	0.46 ms (0.43 ms)	3.15 ms (4.97 ms)	5.91 ms (9.57 ms)
	64	5.33 ms (9.04 ms)	0.92 ms (0.86 ms)	6.34 ms (9.64 ms)	12.12 msi (19.14 ms)
NR-Hashed	128	11.02 ms (18.09 ms)	1.80 ms (1.71 ms)	13.38 ms (20.81 ms)	24.05 ms (37.74 ms)
	256	20.99 ms (35.75 ms)	3.48 ms (3.42 ms)	25.35 ms (39.60 ms)	46.81 ms (75.47 ms)

TABLE I: Running times of various steps in the attribute-based VOPRF constructions in this paper implemented with both Ristretto and x25519 curves. n here denotes the attribute length and we note that reported values are both efficient and scale linearly in n as expected. x25519 numbers reported in parentheses.

Step	time/iter (x25519)	iter/s (x25519)
client-side blinding	70 μs (77 μs)	14.22k (13.02k)
server-side eval (w/o proof gen)	89 μs (196 μs)	11.18k (11.18k)
server-side eval (w/ proof gen)	244 μs (436 μs)	4.1k (2.29k)
client-side unblinding	117 μs (289 μs)	8.52k (3.46k)
client-side verify	364 μs (813 μs)	2.75k (1.23k)

TABLE II: Microbenchmarks on the *non*-attribute based VOPRF constructions with Ristretto and x25519 implementations. The x25519 values are in parentheses.

Step	Communication overhead
Fetch public key	$32 \cdot (n+2)$ bytes
Fetch pk_t	32 bytes
Fetch π_t	$64 \cdot n$ bytes
Upload blinded input	32 bytes
Receive evaluation	64 bytes

TABLE III: Communication overhead for operations over a 256-bit elliptic curve targeted at 128 bits of security. We note that only communications related to fetching and verifying the public key depend on n, communications with receiving credentials are independent of n, as with *non*-attribute based VOPRFs.

Client metrics. We measure and report end-to-end latencies for credential requests in Table V. For a period of a week, we report on the min and max 50th, 95th, and 99th percentile latencies to account for variations over requests and resources over different days. Both at median and 95th percentile values, round trip latency is under a few seconds. The higher latencies at the 99th percentiles can be attributed to variations in networking speeds around the world and to illustrate this further, we break up some of this information based on country code (and note that further variations can be accounted for based on the high-end or low-end devices that are used in different geographies).

While the microbenchmarks from Table II suggest that client-side operations for fetching credentials (blinding and unblinding) steps are fast, we measured runtime for these operations in the wild. Across a wide variety of client devices, we measured median runtime for blinding and unblinding to be under a few milliseconds and even at 99th percentiles were well under a second. Table IV has more information. We found that overall latency for fetching credentials is dominated by networking latencies rather than by compute time. If the client does not have a credential when trying to upload a log message, it would need to fetch one. At the 99th percentile, not only can it can take a long time to fetch a credential, but it could run into errors like network errors or server timeouts. In order to reduce the impact of network errors and latencies, we decided to deploy a few optimizations to reduce the number of cases where a client might not have a credential. For example, we pre-fetch new credentials when old credentials are about to expire. This increases the likelihood of having a credential available when trying to upload a log message. Additionally, when a client runs out of credentials, for the sake of reliability, we allow for the reuse of an already used credential until a new credential has been retrieved from the server. Limiting the number of re-used credentials was also important for rate limiting requests, so we made a reasonable estimate of the number of uses based on data from our existing system of how many client upload events take place in one day. Our metrics suggest that only a small fraction of credentials are used more than once, possibly by clients in the long-tail of the estimated number of events. We believe that these long-tail clients might be located in unreliable network conditions where this optimization helps significantly. Given the volume of data (hundreds of millions of events) processed by our anonymous endpoints, the linkability concerns with reused credentials are minimal. We believe that anyone deploying an anonymous logging system will face similar issues during deployment and these optimizations might be important to obtain the reliability needed.

Step	Time (p50)	Time (p95)	Time (p99)	Time (p99.9)
Blind	1 ms	9 ms	47 ms	680 ms
Unblind	3 ms	25 ms	120 ms	850 ms

TABLE IV: Client blinding and unblinding latency.

Server metrics. Over a 7-day period, our credential service, ACS, recorded roughly between 10,000 and 20,000 QPS of credential fetching traffic. Scaling up our service to meet these demands required a careful allocation of resources across different geographic locations to minimize cross-region traffic both between requests and the service as well as the authentication server that first authenticates the user before querying the credential service. When we first deployed our service,

Percentile	Latency Min	Latency Max
p50	310 ms	360 ms
p95	2.2 s	2.9 s
p99	12.2 s	15.5 s

Country	Latency (p50)	Latency (p95)	Latency (p99)
NA-1	120 ms	559 ms	2.69 s
EU-1	208 ms	783 ms	3.00 s
EU-2	224 ms	889 ms	4.82 s
EU-3	267 ms	1.15 s	6.14 s
EU-4	277 ms	1.10 s	6.00 s
NA-2	221 ms	2.10 s	12.1 s
AS-1	288 ms	1.58 s	8.84 s
SA-1	330 ms	1.99 s	11.8 s
SA-2	282 ms	2.24 s	13.4 s
AF-1	488 ms	4.24 s	17.5 s
AS-2	413 ms	4.11 s	18.1 s
AF-2	570 ms	11.8 s	27.2 s

TABLE V: Credential request end-to-end latencies over a 7-day period, overall (above), and split by a few select countries (below). For anonymity reasons, we refer to countries by continent (NA–North America, EU–Europe, AS–Asia, SA–South America, and AF–Africa). Countries NA-1 to EU-4 have high internet speeds, while the other two buckets have lower typical internet speeds. Note that median latencies are not affected significantly by geography but p95 and p99 latencies vary considerably suggesting tail networking latency effects.

we deployed it to a set of initial data-center regions. When we started testing our system for a larger population of users, we noticed an increase in latency for fetching and uploading log messages. We initially assumed that this increase was because of poor network latency from the clients to our servers; however, on further examination, we found that a large percentage of the latency was due to latency between our services in the data-centers.

While there are several causes of this latency, one of the major ones was due to requests from application servers in one region making requests cross-region to ACS in another region. Rolling out to larger populations caused users to make requests to application servers in a diverse set of regions which did not have a credential service. A lesson we learnt from this is that misallocation of resources across regions can also increase client latencies and degrade user experience. As we deployed ACS to more regions, the cross-region latency reduced to an acceptable level. One of the other sources of latency that we have observed between our servers is the latency between the credential service and our counting service. The overhead of managing atomic counters across regions can be of the order of hundreds of milliseconds.

We also observe that due to our optimization strategies at the client allowing reuse, we not only expected, but also observe an order of magnitude of more requests redeeming credentials with roughly 80,000 to 200,000 QPS depending on the time of the day. Thus, redemptions are more common than issuances in our system, and so focusing on rate limiting redemptions is important. Figure 4 illustrates typical variations across days.

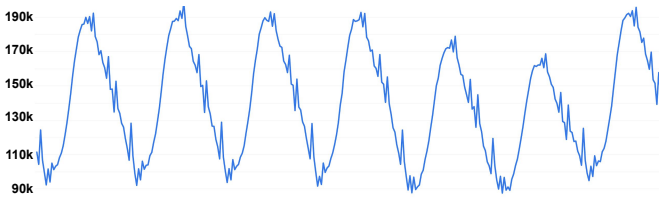

Fig. 4: Typical variation of credential fetch and redemption requests. The y-axis corresponds to the number of redemption requests. The x-axis is removed to anonymize information around what times precisely correspond to peaks and troughs in traffic.

Denial-of-Service protection and anonymous reporting. When we started rolling out DIT, we noticed that clients experienced an unexpectedly high error rate, even in countries that had good network connectivity. On investigating, we realized that this error was due to rate limits at the application server triggered by a lower layer rate limiting library that would rate limit all requests. Normally, this library would rate limit requests based on IP address of the client. But inour system we strip the IP address at our load balancer. As a result the rate limiter applied a global rate limit which would result in errors for everyone when load spiked. To work-around this issue, we experimented with disabling the rate limit for the DIT endpoint. After disabling the limit, we quickly ran into another issue where we saw that our credential service would run into periodic spikes in queries every night at precisely the same time. A spike in workload caused ACS to take longer to respond to requests, and increased latency and errors during the spike, degrading user experience. We found that this was caused due to a faulty timer logic on the client which would trigger events from multiple clients to be scheduled for upload at the same time. As a result, clients from all over would upload events in a synchronized manner to the server at once, triggering more requests for redeeming credentials and causing a minor self denial of service. We deployed a change to the client to dither the time of the upload of logs to spread the queries out a longer period of time.

We realized that we needed some protection against denial of service attacks, even if accidentally caused by our own clients. Additionally, application updates take some time to get to all clients. As a result, we decided to deploy DoS protections for our endpoint on our load balancer. Deploying rate limiting at our load balancer had several advantages. The load balancer already needs the client's IP address to function correctly. We are also able to use several rate limiting mechanisms in order to drop requests, from dropping individual requests, to dropping TCP or QUIC packets from our clients, and also dropping per IP or globally. As shown in Figure 5, after we deployed DoS protection on our load balancers, we were able to limit the sudden spikes of traffic and requests to our servers were smoothed out. This resulted in a reduction in the overall

errors and latency. We believe that organizations deploying anonymous reporting channels should consider deploying rate limiting at the load balancer layer.

Fig. 5: An illustration of spiky and then subsequently smoothed out behavior at the anonymous endpoint.

Attribute-based VOPRFs. We also deployed the protocol that uses the NR-Plain and NR-Hashed Attribute-based VOPRFs from Section IV to a small amount of shadow traffic and evaluated benchmarks in production. In the future, we plan to design and deploy attribute-based VOPRFs with time-based epochs that enable clients to easily verify public keys on-demand without fetching additional keys per time epoch. Our experiment with shadow traffic shows promising results. While slower than plain VOPRFs, even in production, AB-VOPRFs perform just as well, as shown by the microbenchmarks, suggesting that overheads that are a few milliseconds should do little to hurt production throughput (see Table VI). Additionally, we can cache the public keys between requests so that the evaluation of the AB-VOPRF does not have to occur on every request, which amortizes the overhead of deriving keys. One of the main advantages of the AB-VOPRF scheme is that since it uses the same algorithms for the VOPRF as the existing scheme, it does not require changes to clients to deploy. It can seamlessly work with older clients who interpret the AB-VOPRF public key as a regular public key.

Length of attribute	Microbenchmark	Production
—	**42.3** μs	**61.9** μs
16	220 μs	307 μs
128	1.71 ms	2.86 ms
256	3.42 ms	5.85 ms

TABLE VI: Table showing overheads of generating attribute keys at a production scale. The first row refers to a plain VOPRF and the columns refer to key generation times in the stand-alone microbenchmark and with production traffic.

Re-identification resistance measurement. As mentioned previously, we use kHyperLogLog [7] to measure field level re-identifiability. When we initially implemented kHyperLogLog, we used a naive approach of using regular SQL commands to query our datasets to determine the re-identifiability of each field. However, we quickly found that the queries would take a long time, and be killed by the system due to using too many resources like CPU time and memory. To work around this, we implemented kHyperLogLog as a native function in Presto [35], a very popular data query engine. Implementing kHyperLogLog natively sped up queries by 4 times and made the queries practical to perform. Some results of speedups achieved by our system in practice for production datasets of different sizes are presented in Table VII. While we haven't found any re-identifiable fields in DIT, because fields are whitelisted in our solution, we evaluated the same techniques on unrelated datasets and have been able to identify re-identifiable fields. We think that implementing these functions in Presto will help others in implementing processes to assess re-identifiability of data in their own datasets.

Rows (in millions)	Time (Naive HLL)	Time (Native HLL)
25	12s	3s
347	2min	38s
3000	1hr	20min

TABLE VII: Table showing differences between the runtime of native kHyperLogLog implementation and the naive implementation in SQL for tables of different sizes.

VI. Related Work

Privacy Pass. The work closest to our paper is Privacy Pass [1]. Using VOPRFs as a main building block, it enables clients to provide proof of trust without revealing when and who obtained this trust. The primary application of Privacy Pass deployed in practice is to replace cumbersome CAPTCHA interactions with seamless trusted token verification for privacy conscious users who use anonymization services such as Tor [36] and VPNs to mask their IP address. As discussed in this paper, the unique requirements of the anonymous logging framework and the scale at which it needs to operate raise new design and deployment challenges that we address in this paper. We also extend this discussion to new attribute-based VOPRFs to tackle the issue of rotating keys transparently, an issue noted by Privacy Pass [1] but left open.

Kreuter et al. [37] extend Privacy Pass by augmenting anonymous tokens with a private metadata bit that remains unforgeable and private to the clients. The similarity between this metadata bit and what we call attributes in our attribute-based VOPRFs is peripheral. Attributes can be arbitrarily long and correspond to public information such as timestamps or key policy and are intended to be public and verifiable by clients for transparency reasons.

Blind Signatures. Starting with the seminal work of Chaum [2], blind signatures and more advanced anonymous credential schemes [3] have been used to design anonymous electronic cash [38] and direct anonymous attestation frameworks [4], and to launch systems such as Idemix [5] and U-Prove [6]. While we share the same fundamental goals of unlinkability and authenticity, and are inspired by this line of work, these systems typically use more complex cryptographic schemes and primitives that are not suitable for our use case. What makes our schemes more efficient are also a reduced set of requirements. We do not worry about revoking credentials

of individuals as we rotate keys frequently and any impact of compromised clients (particularly on aggregate statistics) is somewhat bounded. We also do not require that credentials be publicly verifiable with only a single party that is interested in issuing and verifying credentials.

Oblivious PRFs. Oblivious PRFs [12], the symmetric-key variant of blind signatures and their verifiable variants are used in a variety of applications from private set intersection [39, 40] to password protocols [26, 13]. Partial VOPRFs [26] are closely related to our notion of attribute-based VOPRF and as discussed in Section IV, a pairing-based instantiation of attribute-based PRF can be obtained using the construction of Everspaugh et al. [26]. While the authors discuss key rotation, they approach it from the perspective of key-updatable schemes, something that does not apply to our use case. Additionally, the use of partial obliviousness in Everspaugh et al. [26] is motivated by the service rate-limiting client requests with something that is not "blindable". In contrast, a similar notion is needed in our schemes to enable the service to evaluate the PRF on the right input in a transparent manner (the input leading to transparent key rotation) – very different requirements on very similar functionality. The partial OPRF construction of Jarecki et al. [41] operates over standard elliptic curves but requires expensive zero-knowledge proofs to be made verifiable.

Private Analytics Systems. Finally, there are several other techniques to collect data for analytics purposes with privacy [42, 43, 44, 45] and references therein. Differential privacy [8, 9, 10] offers solutions when there are no anonymous upload endpoints. They do not tackle integrity although DP guarantees can inherently place limits on malicious clients and their inputs. DP techniques (for privacy from values themselves) can be incorporated into our solution and are complementary to its design. When values are aggregated after verifying credentials, clients can choose to upload these values with local noise to make them differentially private.

Prio [46] tackles integrity by providing proofs that can be verified in zero-knowledge with two non-colluding servers. The values are themselves sent in plaintext and require no integrity protection a la Prio. Systems like ours can be composed with Prio by running the credential redemption and logging service on two independent endpoints guaranteeing that the logging endpoints only learn aggregate, anonymous reports from (previously) authenticated clients.

The Tor Network [36] is a widely deployed anonymous communication network via onion routing through relays that shares similar anonymity goals. Tor is primarily designed for web browsing and two-way communication, and does not have an integrity requirement on clients connecting to anonymous endpoints through relays. In fact, this absence of integrity and the frequent CAPTCHA challenges motivated the work in Privacy Pass. Our system requires a service that can authenticate clients. Additional differences include one-way communication in our system as well as the scale at which reports are received.

VII. Conclusions

In conclusion, we present DIT, a privacy forward authenticated logging platform for receiving reports from billions of clients. We describe specific design choices to scale up our system, extensions, our experience with testing it at scale, and the motivation as well as construction of a novel attribute-based verifiable and oblivious PRF without the use of heavyweight pairings. We leave open the possibility of extending our system to support input and output privacy via techniques such as differential privacy and two-party secret sharing and secure computation.

References

[1] A. Davidson, I. Goldberg, N. Sullivan, G. Tankersley, and F. Valsorda, "Privacy pass: Bypassing internet challenges anonymously," *Proceedings on Privacy Enhancing Technologies*, vol. 2018, no. 3, pp. 164–180, 2018.

[2] D. Chaum, "Blind signatures for untraceable payments," in *Advances in cryptology*. Springer, 1983, pp. 199–203.

[3] J. Camenisch and A. Lysyanskaya, "Signature schemes and anonymous credentials from bilinear maps," in *Annual International Cryptology Conference*. Springer, 2004, pp. 56–72.

[4] E. Brickell, J. Camenisch, and L. Chen, "Direct anonymous attestation," in *Proceedings of the 11th ACM conference on Computer and communications security*, 2004, pp. 132–145.

[5] J. Camenisch and E. Van Herreweghen, "Design and implementation of the idemix anonymous credential system," in *Proceedings of the 9th ACM conference on Computer and communications security*, 2002, pp. 21–30.

[6] C. Paquin and G. Zaverucha, "U-prove cryptographic specification v1. 1," *Technical Report, Microsoft Corporation*, 2011.

[7] P. H. Chia, D. Desfontaines, I. M. Perera, D. Simmons-Marengo, C. Li, W.-Y. Day, Q. Wang, and M. Guevara, "Khyperloglog: Estimating reidentifiability and joinability of large data at scale," in *Proceedings of the 2019 IEEE Symposium on Security and Privacy*, 2019.

[8] C. Dwork, F. McSherry, K. Nissim, and A. Smith, "Calibrating noise to sensitivity in private data analysis," in *Theory of cryptography conference*. Springer, 2006, pp. 265–284.

[9] Ú. Erlingsson, V. Pihur, and A. Korolova, "Rappor: Randomized aggregatable privacy-preserving ordinal response," in *Proceedings of the 2014 ACM SIGSAC conference on computer and communications security*, 2014, pp. 1054–1067.

[10] Apple Differential Privacy Team, "Learing with privacy at scale," 2017.

[11] S. Micali, M. Rabin, and S. Vadhan, "Verifiable random functions," in *40th annual symposium on foundations of computer science (cat. No. 99CB37039)*. IEEE, 1999, pp. 120–130.

[12] M. J. Freedman, Y. Ishai, B. Pinkas, and O. Reingold, "Keyword search and oblivious pseudorandom functions," in *Theory of Cryptography Conference*. Springer, 2005, pp. 303–324.

[13] S. Jarecki, A. Kiayias, and H. Krawczyk, "Round-optimal password-protected secret sharing and t-pake in the password-only model," in *International Conference on the Theory and Application of Cryptology and Information Security*. Springer, 2014, pp. 233–253.

[14] J. Camenisch and M. Stadler, "Proof systems for general statements about discrete logarithms," *Technical Report/ETH Zurich, Department of Computer Science*, vol. 260, 1997.

[15] M. Bellare and P. Rogaway, "Random oracles are practical: A paradigm for designing efficient protocols," in *Proceedings of the 1st ACM conference on Computer and communications security*, 1993, pp. 62–73.

[16] "Ristretto." [Online]. Available: https://ristretto.group/ristretto.html

[17] D. R. Brown and R. P. Gallant, "The static diffie-hellman problem." *IACR Cryptol. ePrint Arch.*, vol. 2004, p. 306, 2004.

[18] J. H. Cheon, "Security analysis of the strong diffie-hellman problem," in *Annual International Conference on the Theory and Applications of Cryptographic Techniques*. Springer, 2006, pp. 1–11.

[19] K. Thomas, J. Pullman, K. Yeo, A. Raghunathan, P. G. Kelley, L. Invernizzi, B. Benko, T. Pietraszek, S. Patel, D. Boneh *et al.*, "Protecting accounts from credential stuffing with password breach alerting," in *28th USENIX Security Symposium (USENIX Security 19)*, 2019, pp. 1556–1571. [Online]. Available: https://storage.googleapis.com/pub-tools-public-publication-data/pdf/33bc2203e7bcb5c0abe289f7432e11563fb2a238.pdf

[20] D. Watson, "Under the hood: Building and open-sourcing fbthrift," https://engineering.fb.com/2014/02/20/open-source/under-the-hood-building-and-open-sourcing-fbthrift/.

[21] M. Annamalai, K. Ravichandran, H. Srinivas, I. Zinkovsky, L. Pan, T. Savor, D. Nagle, and M. Stumm, "Sharding the shards: managing datastore locality at scale with akkio," in *13th {USENIX} Symposium on Operating Systems Design and Implementation ({OSDI} 18)*, 2018, pp. 445–460.

[22] D. Sommermann and A. Frindell, "Introducing proxygen, facebook's c++ http framework," https://engineering.fb.com/2014/11/05/production-engineering/introducing-proxygen-facebook-s-c-http-framework/, 2014.

[23] "Privacy pass key rotation." [Online]. Available: https://blog.cloudflare.com/supporting-the-latest-version-of-the-privacy-pass-protocol/

[24] "Privacy pass trusted key location." [Online]. Available: https://github.com/privacypass/ec-commitments

[25] "Certificate transparency." [Online]. Available: https://www.certificate-transparency.org/

[26] A. Everspaugh, R. Chaterjee, S. Scott, A. Juels, and T. Ristenpart, "The pythia PRF service," in *24th USENIX Security Symposium (USENIX Security 15)*, 2015, pp. 547–562.

[27] M. Naor and O. Reingold, "Number-theoretic constructions of efficient pseudo-random functions," *Journal of the ACM (JACM)*, vol. 51, no. 2, pp. 231–262, 2004.

[28] S. Hohenberger and B. Waters, "Constructing verifiable random functions with large input spaces," in *Annual International Conference on the Theory and Applications of Cryptographic Techniques*. Springer, 2010, pp. 656–672.

[29] D. Boneh, H. W. Montgomery, and A. Raghunathan, "Algebraic pseudorandom functions with improved efficiency from the augmented cascade," in *Proceedings of the 17th ACM conference on Computer and communications security*, 2010, pp. 131–140.

[30] E. Bresson, O. Chevassut, and D. Pointcheval, "The group diffie-hellman problems," in *International Workshop on Selected Areas in Cryptography*. Springer, 2002, pp. 325–338.

[31] "Libsodium—the modern, easy-to-use software library for encryption, decryption, signatures, password hashing, and more." https://libsodium.gitbook.io/doc/.

[32] P. S. Barreto and M. Naehrig, "Pairing-friendly elliptic curves of prime order," in *International workshop on selected areas in cryptography*. Springer, 2005, pp. 319–331.

[33] "Cloudflare bn-254." [Online]. Available: https://github.com/cloudflare/bn256

[34] K. Yu and C. Tang, "Efficient, reliable cluster management at scale with twine," https://engineering.fb.com/2019/06/06/data-center-engineering/twine/, 2019, accessed: 2020-12-01.

[35] "Presto: Distributed sql query engine for big data." https://prestodb.io/docs/current/functions/khyperloglog.html.

[36] P. Syverson, R. Dingledine, and N. Mathewson, "Tor: The secondgeneration onion router," in *Usenix Security*, 2004, pp. 303–320.

[37] B. Kreuter, T. Lepoint, M. Orrù, and M. Raykova, "Anonymous tokens with private metadata bit," in *Annual International Cryptology Conference*. Springer, 2020, pp. 308–336.

[38] D. Chaum, A. Fiat, and M. Naor, "Untraceable electronic cash," in *Conference on the Theory and Application of Cryptography*. Springer, 1988, pp. 319–327.

[39] M. J. Freedman, Y. Ishai, B. Pinkas, and O. Reingold, "Keyword search and oblivious pseudorandom functions," in *Theory of Cryptography Conference*. Springer, 2005, pp. 303–324.

[40] V. Kolesnikov, R. Kumaresan, M. Rosulek, and N. Trieu, "Efficient batched oblivious prf with applications to private set intersection," in *Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security*, 2016, pp. 818–829.

[41] S. Jarecki, H. Krawczyk, and J. Resch, "Threshold partially-oblivious prfs with applications to key management," Cryptology ePrint Archive, Report 2018/733, 2018, https://eprint.iacr.org/2018/733.

[42] G. Danezis, C. Fournet, M. Kohlweiss, and S. Zanella-Béguelin, "Smart meter aggregation via secret-sharing," in *Proceedings of the first ACM workshop on Smart energy grid security*, 2013, pp. 75–80.

[43] A. Bittau, Ú. Erlingsson, P. Maniatis, I. Mironov, A. Raghunathan, D. Lie, M. Rudominer, U. Kode, J. Tinnes, and B. Seefeld, "Prochlo: Strong privacy for analytics in the crowd," in *Proceedings of the 26th Symposium on Operating Systems Principles*, 2017, pp. 441–459.

[44] T. Elahi, G. Danezis, and I. Goldberg, "Privex: Private collection of traffic statistics for anonymous communication networks," in *Proceedings of the 2014 ACM SIGSAC Conference on Computer and Communications Security*, 2014, pp. 1068–1079.

[45] L. Melis, G. Danezis, and E. De Cristofaro, "Efficient private statistics with succinct sketches," *arXiv preprint arXiv:1508.06110*, 2015.

[46] H. Corrigan-Gibbs and D. Boneh, "Prio: Private, robust, and scalable computation of aggregate statistics," in *14th {USENIX} Symposium on Networked Systems Design and Implementation ({NSDI} 17)*, 2017, pp. 259–282.

[47] M. Hamburg, "Decaf: Eliminating cofactors through point compression," in *Annual Cryptology Conference*. Springer, 2015, pp. 705–723.

[48] W. Diffie and M. Hellman, "New directions in cryptography," *IEEE transactions on Information Theory*, vol. 22, no. 6, pp. 644–654, 1976.

APPENDIX
CRYPTOGRAPHIC FOUNDATIONS

A. Preliminaries

In this section, we introduce formal notions of security for verifiable random functions (VRFs) and the Diffie-Hellman assumptions required for the proofs in Appendix B.

VRF Security Games. Recall the definition of a verifiable random function from Section II-B. The security of a VRF is defined via a VRF security game between a Challenger and an adversary $\mathcal{A}$. First, the Challenger samples a uniform (sk, pk) from the set of all keypairs and publishes pk for $\mathcal{A}$ to use. Next, $\mathcal{A}$ queries the Challenger with inputs x and receives $(F(sk, x), \pi_x)$. This proceeds interactively with $\mathcal{A}$ able to choose which x's to query based on previous responses. At any point, $\mathcal{A}$ can request a "real-or-random" challenge on a challenge input x^* subject to x^* not being in the any of the queries (either prior or subsequently).

The Challenger, on input a challenge input x^* responds with either $F(sk, x)$—"real"—or uniformly at random from the outputs of $F(sk, \cdot)$—"random". For shorthand, we denote the two behaviors as either Chal-Real or Chal-Rnd. The game ends when $\mathcal{A}$ is done with its queries and outputs 0 or 1 with the aim of distinguishing between Chal-Real and Chal-Rnd. The adversary's *advantage* in the VRF game is defined as:

$$\mathbf{Adv}_{\text{vrf}}(\mathcal{A}) := \left|\Pr\left[\mathcal{A}^{\text{Chal-Real}}(1^\lambda) = 1\right] - \Pr\left[\mathcal{A}^{\text{Chal-Rnd}}(1^\lambda) = 1\right]\right|.$$

Observe that this is an extension of $\mathbf{Adv}_{\text{prf}}(\mathcal{A})$ defined in Section II-B except that the adversary only interacts with the random function on its challenge input (otherwise the proofs π_x would necessarily have to fail on other queries).

A weaker notion of security for VRFs, denoted *selective* security, is also useful. While in practice an adversary can choose to break the pseudorandomness of the function by querying it on a carefully constructed target input x^*, it is likely that any such inputs are independent of the public and secret keys in the system. Selective security captures this notion by modifying the real-or-random VRF security game as follows.

Instead of the Challenger first sampling a keypair (sk, pk) and publishing pk, the Challenger first receives the target query x^* from $\mathcal{A}$. Then, it samples the keypair, publishes pk, and responds to all $\mathcal{A}$ queries $x \neq x^*$. It finishes the game responding with either $F(sk, x^*)$ or a random value as before. If Sel-Chal-Real and Sel-Chal-Rnd are shorthands for the two behaviors of the challenger in the selective security game, then $\mathbf{Adv}_{\text{sel-vrf}}(\mathcal{A})$ is naturally defined as:

$$\mathbf{Adv}_{\text{sel-vrf}}(\mathcal{A}) := \Big|\Pr\left[\mathcal{A}^{\text{Sel-Chal-Real}}(1^\lambda) = 1\right] - \Pr\left[\mathcal{A}^{\text{Sel-Chal-Rnd}}(1^\lambda) = 1\right]\Big|,$$

and it is easy to show that $\mathbf{Adv}_{\text{sel-vrf}}(\mathcal{A}) \leq \mathbf{Adv}_{\text{vrf}}(\mathcal{A})$.

Finally, we note that our formal security analysis considers the VRF operating in a non-oblivious mode. While the core arguments remain identical, a formal treatment of security (including security notions and proofs) will require the introduction of a "one-more" version of pseudorandomness along the lines of Appendix B.2 in [26]. This captures the notion that with oblivious queries, the notion of $x^* \neq x$ becomes subtle and the concept of a challenge query is one that allows the adversary to gather the output of the VRF on *one more* evaluation than the number of oblivious evaluation queries. The results in Appendix B should extend to this scenario along the lines of Everspaugh et al. [26].

Diffie-Hellman Assumptions. The Diffie-Hellman problem, first mentioned in the seminal work of Diffie and Hellman [48], asks the question: how hard is it to compute g^{ab} given g^a and g^b for two random a and b drawn from the size of the group and g being a generator of this group? A variant of the problem, called the decisional-Diffie Hellman problem (DDH) defines $\mathbf{Adv}_{\text{ddh}}(\mathcal{A}) := |\Pr[\mathcal{A}(g, g^a, g^b, g^c) = 1 : a, b, c \leftarrow \mathbb{Z}_p] - \Pr[\mathcal{A}(g, g^a, g^b, g^{ab}) = 1 : a, b \leftarrow \mathbb{Z}_p]$, the advantage an adversary has of guessing whether a given value is g^{ab} or simply a random value in $\mathbb{G}$.

The DDH assumption over a class of groups $\mathbb{G}$ states that for all efficient adversaries, $\mathbf{Adv}_{\text{ddh}}(\mathcal{A})$ is $\text{negl}(\lambda)$, where λ is the security parameter and $\log(|\mathbb{G}|)$.

There are several variants of the DH assumption [30] and for this paper, we consider one variant that is denoted the n Diffie-Hellman Exponent assumption (n-DHE). The n-DHE problem requires an adversary to distinguish the $n+1$-th power of a secret α hidden in the exponent from a random element in $\mathbb{G}$. More formally,

$$\mathbf{Adv}_{\text{nDHE}}(\mathcal{A}) := \Big|\Pr\left[\mathcal{A}(g, g^\alpha, g^{\alpha^2}, \ldots, g^{\alpha^n}, g^{\alpha^{n+1}}) = 1 : \alpha \leftarrow \mathbb{Z}_p\right] - \Pr\left[\mathcal{A}(g, g^\alpha, g^{\alpha^2}, \ldots, g^{\alpha^n}, g^r) = 1 : \alpha, r \leftarrow \mathbb{Z}_p\right]\Big|.$$

The n-DHE assumption simply states that this advantage $\mathbf{Adv}_{\text{nDHE}}(\mathcal{A})$ remains $\text{negl}(\lambda)$ for all efficient adversaries $\mathcal{A}$. Bresson et al. [30] show a reduction from the nDHE-problem to the standard DDH problem which would enable our proof of security from Theorem 1 to be directly derived from the DDH assumption but unfortunately the reduction adds an overhead that grows exponentially in the size of n rendering it vacuous for larger values of n. We leave open the question of proving the security of our AB-VOPRF from DDH in a tight manner.

B. Proofs

Theorem 1. *In the random oracle model, the function F defined by Eq.* (1) *is a selectively-secure verifiable random function under the hardness of the n-DHE assumption. Moreover, the reduction preserves the adversary's advantage, i.e., for every VRF adversary $\mathcal{A}$, there exists a n-DHE adversary $\mathcal{B}$ such that $\mathbf{Adv}_{\text{sel-vrf}}(\mathcal{A}) \leq \mathbf{Adv}_{\text{nDHE}}(\mathcal{B}) + \text{negl}(\lambda)$.*

Proof. The proof proceeds in two steps. In the first step, given a n-DHE challenge $(A_0 = g, A_1 = g^\alpha, \ldots, A_n = g^{\alpha^n}, A_{n+1})$ where $A_{n+1} = g^{\alpha^{n+1}}$ or random, we setup parameters to simulate the selective-security game to a VRF adversary $\mathcal{A}$. In the second step, we show that this simulation is (a) indistinguishable from a real security game, and (b) returns either a real or random output whenever $A_{n+1} = g^{\alpha^{n+1}}$ or random respectively. The theorem and the tightness of the reduction follows immediately.

To simulate the selective-security game, we first receive the challenge input (t^*, x^*) from $\mathcal{A}$. Consider the indices in t^* that are zero and one respectively: $\mathcal{I}_0 = \{i : t^*[i] = 0\}$ and $\mathcal{I}_1 = \{i : t^*[i] = 1\}$.

Public Parameters: Sample $r_h, r_0, \ldots, r_n \leftarrow \mathbb{Z}_q$. Program the random oracle $H(x^*) = A_1 = g^\alpha$. Implicitly, we set:

$$a_0 = r_0 \cdot \prod_{i \in \mathcal{I}_0} (\alpha + r_i),$$

$$a_i = \begin{cases} (\alpha + r_i), & \text{if } i \in \mathcal{I}_1 \\ (\alpha + r_i)^{-1}, & \text{if } i \in \mathcal{I}_0 \end{cases}$$

$$h = g^{r_h \cdot \prod_{i \in \mathcal{I}_0} (\alpha + r_i)}.$$

Note that g^{a_0} and $h_j = h^{a_j}$ can be computed "in the exponent"—i.e., knowing only g raised to powers of α and not α itself—as follows: first, compute the polynomial $p(\alpha)$ corresponding to the exponent (either $r_0 \cdot \prod(\alpha + r_i)$ for $i \in \mathcal{I}_0$ or $r_h \cdot \prod(\alpha + r_i)$ for i's depending on j and $\mathcal{I}_0$ respectively). Next, compute $g^{p(\alpha)}$ as $\prod(A_i)^{p_i}$ where $p(\alpha) = \sum_{i \leq n} p_i \cdot \alpha^i$ is a $\leq n$-degree polynomial. h can be computed in a similar fashion. Thus, we can publish the public parameters:

$$mpk = (\mathbb{G}, g, h, g^{a_0}, h_1 = h^{a_1}, \ldots, h_n = h^{a_n}).$$

The selective-security game proceeds with PKGen and PRF Eval queries. Given the setup above, we describe how to answer both sets of queries using the same "in the exponent" computation technique starting with the DHE challenge and programming the random oracle to generate corresponding NIZKs.

PKGen queries: To simulate pk_t consider the polynomial in the exponent. $p(\alpha) = a_0 \cdot \prod a_i^{t[i]} = r_0 \prod(\alpha + r_i)$ for indices i that satisfy one of these two conditions: either $i \in \mathcal{I}_1$ and $t[i] = 1$ or $i \in \mathcal{I}_0$ and $t[i] = 0$. As $\deg(p) \leq n$, pk_t can be simulated as explained above. The partial products P_i can be

computed in an almost identical manner (by considering the product up to index $1, 2, \ldots, n-1$). Note that these values $P_1, \ldots, P_n$ are correctly computed, just without access to raw values of α (and hence a_i's). Therefore, by programming the random oracle, DLEQ proofs $\pi_1, \ldots, \pi_n$ can be simulated. If one of these points has been queried before, the simulation aborts, but that happens with probability $< nq_H \cdot |\mathbb{G}|^{-1}$ (where q_H is the total number of queries to H) which is $\mathrm{negl}(\lambda)$.

PRF queries: For every query $(x, t) \neq (x^*, t^*)$, we simulate the output. We consider two scenarios: $x = x^*$ and $x \neq x^*$.

When $x = x^*$, recall that $H(x) = g^{\alpha}$. The PRF output is $H(x)^{a_0 \prod a_i^{t[i]}}$ which is of the form $g^{p(\alpha)}$ for a polynomial

$$p(\alpha) = \alpha \cdot r_0 \prod_{i \in \mathcal{I}(t)} (\alpha + r_i),$$

where $\mathcal{I}(t) := \{i : (i \in \mathcal{I}_0 \wedge t[i] = 0) \vee (i \in \mathcal{I}_1 \wedge t[i] = 1)\}$, the set of indices in $t \oplus t^*$ set to 1. As $t \neq t^*$ the size of this set is $\leq n-1$ and therefore $\deg(p) \leq n$. As discussed before, the PRF output $g^{p(\alpha)}$ can therefore be computed given A_i's from the DHE challenge.

When $x \neq x^*$, we program the random oracle $H(x) = g^{r_x}$ for uniform and independently sampled $r_x \leftarrow \mathbb{Z}_q$ for each x. In similar lines to $x = x^*$, the resulting polynomial $p(\alpha)$ is of the form $r_0 \cdot r_x \prod_{i \in \mathcal{I}(t)} (\alpha + r_i)$. Even if $t = t^*$, the degree of $p(\alpha)$ is at most n and therefore $g^{p(\alpha)}$ can be computed "in the exponent."

As with PKGen queries, by programming the random oracle, the output of F is evaluated correctly with respect to the public parameters, therefore any NIZKs can be appropriately simulated. It is also easy to see that these simulations are distributed as required by the definition of F. The randomness $r_0, \ldots, r_n$ in the public parameters ensures their correct distribution even if they have additional structure, and both from the randomness of the nDHE assumption as well as additional randomness r_x, the outputs of $H(x)$ on arbitrary x are correctly simulated.

Challenge input: The final task is to simulate the challenge output on input (x^*, t^*) that is either a real evaluation of the PRF or a random value in $\mathbb{G}$. Our setup lets us do this in a straightforward manner. The PRF output is $H(x^*)^{a_0 \prod a_i^{t[i]}}$ which is of the form $g^{p(\alpha)}$ for:

$$p(\alpha) = \alpha \cdot r_0 \prod_{i=1}^{n} (\alpha + r_i).$$

Note that this is the full product of all $(\alpha + r_i)$'s because $t = t^*$. $p(\alpha)$ is therefore a degree $n+1$ polynomial that can be written as $r_0 \cdot \alpha^{n+1} + q(\alpha)$, a degree $\leq n$ polynomial $q(\cdot)$. Therefore, we set the PRF output to be $g^{q(\alpha)} \cdot A_{n+1}^{r_0}$ where $g^{q(\alpha)}$ is computed "in the exponent" using only $A_1, \ldots, A_n$ from the challenge.

In the final step of the proof, consider the case when $A_{n+1} = g^{\alpha^{n+1}}$. The challenge output is exactly the output of the pseudorandom function on (x^*, t^*). When $A_{n+1} \leftarrow \mathbb{G}$, the PRF output is a random element raised to a power and multiplied by another element. As we haven't used A_{n+1} prior to this, A_{n+1} continues to be uniformly random and the output $g^{p(\alpha)} \cdot A_{n+1}^{r_0}$ is a uniformly random element in $\mathbb{G}$, exactly as required.

From this, it follows that, except in the cases of an abort when simulating NIZKs (which happens with negligible probability), an adversary that can break the selective-security PRF game by distinguishing the challenge output from uniformly random can be used to break the DHE challenge directly. This implies that $\mathbf{Adv}_{\text{sel-vrf}}(\mathcal{A}) \leq \mathbf{Adv}_{\text{nDHE}}(\mathcal{B}) + \mathrm{negl}(\lambda)$, as required, completing the proof. □

The following corollary implies that any selectively secure VRF can be efficiently transformed into adaptively secure one in the random oracle model using a small modification where the original attribute string t is replaced by its hash $H(t)$.

Corollary 1. *In the random oracle model, the function F' defined $F'(msk; (t, x)) := F(msk; (H(t), x))$ is a secure verifiable random function. For every VRF adversary $\mathcal{A}$ against F' making atmost q_H queries to $H(\cdot)$, there exists a selective-secure VRF adversary $\mathcal{B}$ against F such that $\mathbf{Adv}_{\text{vrf}}(\mathcal{A}) \leq q_H^2 \cdot \mathbf{Adv}_{\text{sel-vrf}}(\mathcal{B})$.*