Alex Davidson, Ian Goldberg, Nick Sullivan, George Tankersley, and Filippo Valsorda

Privacy Pass: Bypassing Internet Challenges Anonymously

Abstract: The growth of content delivery networks (CDNs) has engendered centralized control over the serving of internet content. An unwanted by-product of this growth is that CDNs are fast becoming global arbiters for which content requests are allowed and which are blocked in an attempt to stanch malicious traffic. In particular, in some cases honest users — especially those behind shared IP addresses, including users of privacy tools such as Tor, VPNs, and I2P — can be unfairly targeted by attempted 'catch-all solutions' that assume these users are acting maliciously. In this work, we provide a solution to prevent users from being exposed to a disproportionate amount of internet challenges such as CAPTCHAs. These challenges are at the very least annoying and at their worst — when coupled with bad implementations — can completely block access from web resources. We detail a 1-RTT cryptographic protocol (based on an implementation of an oblivious pseudorandom function) that allows users to receive a significant amount of anonymous tokens for each challenge solution that they provide. These tokens can be exchanged in the future for access without having to interact with a challenge. We have implemented our initial solution in a browser extension named "Privacy Pass", and have worked with the Cloudflare CDN to deploy compatible server-side components in their infrastructure. However, we envisage that our solution could be used more generally for many applications where anonymous and honest access can be granted (e.g., anonymous wiki editing). The anonymity guarantee of our solution makes it immediately appropriate for use by users of Tor/VPNs/I2P. We also publish figures from Cloudflare indicating the potential impact from the global release of Privacy Pass.

Keywords: Anonymity, Blinded tokens, Oblivious PRF, Tor, CAPTCHA, Privacy, DLEQ, Content delivery networks

DOI 10.1515/popets-2018-0026
Received 2017-11-30; revised 2018-03-15; accepted 2018-03-16.

Alex Davidson: Royal Holloway, University of London (work completed during an internship at Cloudflare), E-mail: alex.davidson.2014@rhul.ac.uk

1 Introduction

1.1 Background

An increasingly common trend for websites with globally high visitation is to use content delivery networks (CDNs) to host or cache their resources. According to Cisco, CDNs will serve 71% of all traffic in 2021, up from 52% in 2016 [7]. Some of the most well-known CDNs include Akamai, Cloudflare, Fastly, and Amazon Cloudfront. These CDNs typically house data centers across the globe, meaning that access to websites is sped up by serving from locations geographically near requests.

On top of this, CDNs usually offer protection services to prevent customers from having their websites taken down by distributed denial of service (DDoS) attacks, or vandalized by spam. Protection of this sort takes numerous forms, but a common method is to use public IP-based reputation checking on requests that target protected websites. These checks involve analyzing whether incoming IPs have been used for past malicious activity before granting access. If an IP address is deemed to have a 'poor' reputation existing CDNs may use one of the following options [29] for guarding protected websites:

- block access altogether;
- issue human proof-of-work challenges to prevent access from bots (e.g., CAPTCHAs);
- route requests through a web application firewall.

Unfortunately, these measures disproportionately affect users who happen to be assigned to IPs with poor reputations. In particular, users of Tor and VPN providers are allocated one of a small number of IP addresses (corresponding to exit nodes in the case of Tor). Since different users use these IP addresses, if *any one* of them misbehaves then the IP will be assigned a bad reputa-

Ian Goldberg: University of Waterloo, E-mail: iang@cs.uwaterloo.ca
Nick Sullivan: Cloudflare, E-mail: nick@cloudflare.com
George Tankersley: E-mail: george.tankersley@gmail.com
Filippo Valsorda: E-mail: hi@filippo.io

tion score. Future users may then have limited access to CDN-protected websites in the future, regardless of whether they are honest.

Users of Tor/VPNs are often forced into doing so due to oppressive internet censorship, so access-limiting measures of this nature can further degrade internet accessibility for an important subset of users. It is worth noting that 1.04% of the global traffic served by Cloudflare is challenged. [8] In comparison, the number of requests from Tor users challenged by Cloudflare sits at 17%. Moreover, Cloudflare typically serves over 10 trillion requests per month[1] and approximately 0.05% of this traffic arrives over the Tor network [8]. As such, Cloudflare handles thousands of requests every second, and billions of requests every month, from the Tor network alone, not accounting for the numbers of requests originating from VPNs/I2P shared networks that operate under similar conditions. Reducing the workload for these users will clearly have a large, undeniably positive effect on web accessibility for these users.

In this work, we develop a solution that works in tandem with any system that uses challenges to ascertain whether a client is honest. In particular, we grant an anonymous user, deemed to be honest at some point in the past, the ability to gain access to further web resources without the need for more challenges. Our solution maintains anonymity of the user — this is crucial so that CDNs cannot globally link requests and attempt to subvert anonymization measures.

1.2 Related work

1.2.1 Anonymous e-cash

In essence, our solution grants a user anonymous tokens for each challenge that they solve; these tokens can be redeemed later to bypass future challenges. The setup is reminiscent of anonymous e-cash settings that were popularized by Chaum [3, 4]. Chaum showed that it was possible to modify textbook RSA such that a user could receive signatures from a signer on ‘blinded’ messages. The user is then able to ‘unblind’ the signature such that it is a valid signature for the unblinded message. The blinding prevents the signer from learning the underlying message. These techniques were used by Chaum et al. [5] for an untraceable e-cash system and more recently as the backbone of the taxable, anonymous e-cash service Taler [9] released by the GNU project in 2016. There have been numerous different blind signature schemes introduced since the blind-RSA variant [1, 10–12, 24, 25, 27], though these typically are much more conceptually involved and sometimes require more than two rounds of communication.

1.2.2 Anonymous blacklisting/whitelisting

Anonymous e-cash systems can be useful in granting privacy-preserving access to resources. An alternative method for providing similar functionality is the usage of anonymous blacklisting techniques [14]. In these schemes a user is asked to prove that they are not blacklisted (or that they are part of a whitelist) using some finite resource that they control. This is very similar to the situation that we consider, where owning a token is equivalent to being part of a whitelist or not being blacklisted. In fact, the work of Henry and Goldberg [14] is motivated by a similar scenario where honest users of Tor are denied access to resources. Additionally, they also explicitly consider the potential of using human challenges or CAPTCHAs as a method for whitelisting. Their formalization suggests numerous ways for maintaining an anonymous blacklist including using blind RSA signatures. They also discuss ‘Nymble-like’ [13, 22] systems that leverage the use of a trusted third party for maintaining anonymity during access requests. A similar situation is considered by Liu et al. [21] when proposing their solution TorPolice, a privacy-preserving mechanism that enables access-control policy implementation with respect to connections originating from the Tor network.

We ignore the ‘Nymble-like’ designs as they require an extra trusted party that we do not. While Henry and Goldberg [14] provide a formalization of multiple different anonymous blacklisting methods, with various pros and cons, there is little work detailing the scalability of such systems. Moreover, our solution results in a browser extension that is easily supported by simply running a small bit of code at the service provider. This scales much more efficiently than a service provider implementing their own blacklisting/whitelisting techniques. Our solution benefits all users, not just those originating from Tor, and so we provide a more general browser-based solution than TorPolice [21].

1 https://www.cloudflare.com/case-studies/, accessed on 17 Nov 2017

1.2.3 Privacy-preserving e-ticketing

Another diverging thread of literature focuses on 'privacy-preserving e-ticketing'. The works of Heydt-Benjamin et al. [17] and Sadeghi et al. [26] were the first to propose solutions to privacy-preserving ticketing for public transportation systems. These solutions were based on anonymous e-cash systems such as those listed above and on physically unclonable functions (PUFs). The work of Kerschbaum et al. [20] analyzed Singapore's EZ-Link system and showed that it was easy to extract a traveler's travel records. They proposed an encrypted bill processing procedure that allows for privacy-preserving data mining analysis by the transport company.

While these solutions are similar to the work that we describe, they are heavily targeted at public-transport applications. Thus, their solution may not be optimal when carried across to more generic scenarios involving internet transactions. Additionally, data-mining analysis on the results of the transactions is the primary motivation of these works, which we do not share.

1.3 Our contributions

In this work, we detail a deployable browser-based solution that allows honest users to drastically reduce the number of internet challenges that they are required to solve. Our solution retains anonymity for the user via the use of a cryptographic protocol engaged with a CDN that protects the requested resource. In particular, we provide a detailed case study of the effectiveness of our solution in the form of a public integration with the architecture used by Cloudflare. We refer to this architecture generically as the 'edge' throughout.

Our cryptographic protocol makes use of an adaptation to the 'oblivious pseudorandom function' (OPRF) protocol introduced by Jarecki et al. [18, 19] for password-protected secret sharing. An OPRF protocol, in brief, allows for a user to ask for PRF evaluations on inputs that are hidden from the PRF key holder. An OPRF can be instantiated simply and efficiently using elliptic curve techniques, but the protocol is otherwise similar to that used in a blind-RSA interaction. In our design the OPRF key holder is the edge service provider, while the user requesting a resource sends inputs to the OPRF. The OPRF protocol results in a 1-RTT protocol for both signing and redemption — no current blind signing protocols operate in less than 1-RTT so this is comparable with the most efficient previous work. Our work is also heavily related to recent advances in constructing elliptic curve verifiable random functions by Papadopoulos et al. [23] and constructions of elliptic curve OPRFs by Burns et al. [2].

The work of Jarecki et al. [18] uses a non-interactive zero-knowledge (NIZK) proof of discrete log equality (DLEQ) to provide verification of the OPRF result to the user. Their construction is hence a 'verifiable' OPRF or VOPRF and is proven secure in the random-oracle model. We adapt their construction slightly to use a 'batch' DLEQ proof allowing for much more efficient verification; in short this allows a user to verify a single NIZK proof that states that all of their tokens are signed by the same private key. This prevents the edge from using different key pairs for different users in an attempt to launch a deanonymization attack; we give more details in Section 3.2. The security proofs from the above works [18, 23] immediately imply that our protocol maintains the privacy of the user in the interaction, though we also provide short arguments as to why the protocol meets more specific security goals in Section 5. We also analyse the success of out-of-band attacks on our protocols in Section 8.

Finally, we provide data in Section 7.1 from the deployment of our solution in the form of a browser extension named 'Privacy Pass' for Chrome and Firefox. Privacy Pass interacts with Cloudflare's edge server to store and spend tokens that are received when a user provides a valid Cloudflare CAPTCHA page solution. We show that our solution results in very little change to a user's browsing habits in terms of the time taken for cryptographic operations to complete. To aid presentation of this impact we provide a detailed overview of Cloudflare's edge architecture. While we focus predominantly on Cloudflare, our protocol is sufficiently general that it can be used in any situation where anonymous whitelisting is appropriate.

1.3.1 Alternative CDN-level solutions

Privacy Pass aims to be used in tandem with current challenge-response mechanisms used by CDNs that aim to reduce the influx of malicious requests. As a result, if a CDN implemented no such security measures then the utility of Privacy Pass would be limited. Privacy Pass will decrease the number of challenges presented to honest users by a factor of N, where N is the number of tokens issued for a correct challenge solution. However, a CDN that has a high false-positive rate for a particular class of users (for example, those very users connecting via Tor or a VPN) would require a higher value of N, but this comes with additional security disadvantages, as highlighted in Section 8.

In summary, Privacy Pass does not represent a panacea. The success of our solution is somewhat dependent on the way that a CDN implements security measures and this is considered in our Cloudflare case study in Section 7.1.

2 Current client-edge workflow

To give better intuition for the problem that we are attempting to solve, we illustrate the workflow that is currently initiated between a user/client and an edge provider/CDN. An edge provider is a reverse proxy that examines connections from clients and determines whether they are allowed access to the requested origin site, and if so, serves a cached version (if possible), or forwards the request to the true origin site otherwise (for dynamic content, for example). The access check is one part of the security functionality provided by CDNs such as Cloudflare. These CDNs offer a number of different security and efficiency enhancements; we focus specifically on the mechanism that is used for challenging users who are suspected to be malicious. Typically, the challenge is some manual 'proof-of-work' that is completed by the user. CAPTCHAs are the most common type of such a test, determining whether the user is human or not.

2.1 Gauging reputation

Like many CDNs, Cloudflare uses a reputation gauging system that assigns scores based on activity witnessed from the client IP address. IP addresses are assigned malicious-reputation scores (from 0 to 100) suggesting the confidence that the IP address has been involved in malicious behavior in the past. As an example, IP addresses that have been linked to 'bot-like' behavior (such as spamming or DDoS attacks) will likely have a high score. However, other indicators taken from the client can also be used to change the score.

2.2 Deciding challenges

Let $0 < \tau < 100$ be some threshold malicious-reputation score. Simply, if a user **C** with reputation $\nu > \tau$ attempts to access some website protected by the edge

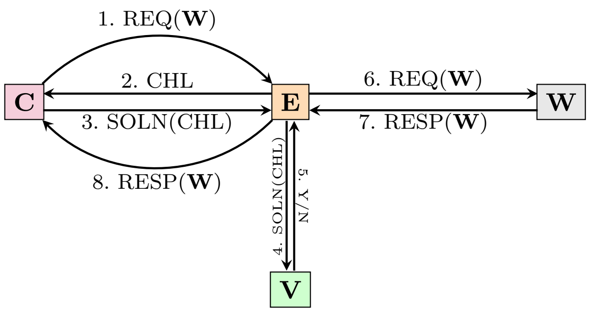

Fig. 1. Current workflow for challenge system

(i.e., a customer of the edge) then **C** will always be shown a challenge.

While this simplification ignores other possibilities for routing requests through CDN architecture, it helps us to generalize the workflow that we consider. Note that we are only interested in reducing the requirement for honest, human users to complete internet challenges; other methods of request routing are not explicitly covered by our solution.

2.3 Allowing access

In Fig. 1, we detail the workflow that occurs when a user **C** with reputation score $\nu_{\mathbf{C}} > \tau$ interacts with the edge provider **E** to access an *origin* website **W**. We also include a party **V** known as a validator; this entity validates challenge solutions for **E**.

In the interaction, we assume that there are messages REQ() and RESP() that are used respectively for requesting and responding with content (mirroring HTTP requests/responses, for example). Furthermore, there is a message CHL requesting that a challenge be solved, and a human-involved task SOLN() that takes a challenge CHL as input and outputs a message containing a (correct) solution that **V** can validate. It is important that SOLN() is not mistaken for an algorithm that can be run on a given challenge instance. This mirrors the complexity of solving a CAPTCHA, for instance. In the case of an incorrect solution (i.e., **V** returns 'N') then steps 6 onwards are not undertaken.

A client in the current workflow has to compute a challenge solution for every request to **E** that they make. If there are multiple origins protected by **E**, then **E** could mitigate this by giving the client a method of authenticating in the future when a challenge is provided. In the HTTP setting, this is typically done using cookies. These (single-origin) cookies prevent a client from having to solve a challenge again for the same origin **W** for some predefined interval of time. However, the same CDN serves a very large number of origins, and the client would still have to solve one challenge for each origin visited. Alternatively, **E** could use *cross-domain* cookies for multiple origins, although at the severe privacy cost of linking the client's requests across all origins.

2.4 Introducing tokens

The rest of the work will focus on how we can use cryptographically blinded 'tokens' to alleviate the burden of the SOLN() step above. In particular, these tokens will be distributed by **E** on acceptance of a valid challenge solution. The tokens will be redeemed and verified by **V** instead of providing a future challenge solution. We present our improved workflow in Section 6.

3 Notation and preliminaries

In the introduction we mentioned that our solution benefits users of a variety of PETs, but for ease of discussion we will refer to this entire set of users as Tor users. We will refer to a 'user' or 'client' as the initiator of a connection to some website. We define the 'edge' server as the entity that decides whether a user's request will be allowed or denied. We use a 'challenge' generally to refer to some task that the edge provides to a user to perform to prove that they are acting in an 'honest' way (for CAPTCHAs this check simply proves the requests originates from a human). By an 'honest' user we simply mean a user that *should* be granted access to a website and conversely for a 'malicious' user. We will use the notion of a 'protected' page to refer to a website that uses the edge to supply users with challenge pages for granting access.

For positive integers m, we write $i \in [m]$ to denote that i takes some value in the set $\{1, \ldots m\}$. For an algorithm $A(\cdot)$ we write $z \leftarrow A(x)$ to denote that z takes the value of the output of A on input x. If we do not say otherwise then $z \in \{0,1\}^{\mu}$ where $\mu = \mathsf{poly}(\lambda)$. For a set S we write $z \leftarrow\!\!\$\, S$ to indicate that z is sampled uniformly from S. We will use the acronym PPT to mean probabilistic polynomial time with respect to algorithms and adversaries $\mathcal{A}$ in security arguments. We denote by $\mathsf{negl}(\lambda)$ a 'negligible' function whereby, for all polynomials $p(\lambda)$, then $\mathsf{negl}(\lambda) < 1/p(\lambda)$ for sufficiently large λ. We write $b \overset{?}{=} b'$ to indicate checking that b equals b' before proceeding with further steps in a protocol. If $b \neq b'$ then we assume that the protocol is aborted at this point (with a message $\perp$ sent to the client indicating as such). We use $||$ to denote concatenation of elements.

In our protocols, we will use the notation **C** and **S** to denote the client and server respectively.

3.1 VOPRF description

The main building block of our construction is a verifiable oblivious pseudorandom function (VOPRF) [18]. An OPRF is a protocol between a server and a user. The server holds a key k for a PRF F, while the user holds some element x that they intend to use as an input. At the end of the protocol the user learns $F_k(x)$ and the server learns nothing (i.e., $\perp$).

A VOPRF protocol extends the functionality to also include a fixed descriptor Y that serves as a commitment to the private key k. Then, much like a digital signature, Y can be used to *verify* that a claimed output $F_k(x)$ is the (unique) correct value, but does not allow the *computation* of $F_k(x)$.

3.2 Discrete log equivalence proofs (DLEQ)

We next detail a zero-knowledge protocol for proving that two pairs of values have the same discrete log relation (e.g., that for $Y = X^{k_1}$ and $Q = P^{k_2}$, then $k_1 = k_2$). We point readers towards prior work [6, 16, 18, 19] for arguments as to why this construction satisfies the soundness and security guarantees required for a NIZK proof.

Let $\mathbb{G}$ be a group with prime order q, let X, P be generators of $\mathbb{G}$, and let $H_3 : \mathbb{G}^6 \to \mathbb{Z}_q$ be a hash function modeled as a random oracle. Let $Y = X^k$ and $Q = P^k$ for some $k \in \mathbb{Z}_q$ and $X, P \in \mathbb{G}$. Chaum and Pedersen [6] showed that it was possible to generate a proof D_k that $\log_X(Y) = \log_P(Q)$. We can think of the server and client that we defined previously as a prover and verifier in this interaction, respectively. Both are given X, Y, P, Q while the prover knows k as a secret.

1. The prover samples a random nonce $t \leftarrow\!\!\$\, \mathbb{Z}_q$ and commits to t with respect to X, P by computing $A = X^t$ and $B = P^t$.
2. The prover computes $c = H_3(X, Y, P, Q, A, B)$ and $s = (t - ck) \mod q$.
3. The prover sends $D_k = (c, s)$ to the verifier.
4. The verifier computes $A' = X^s \cdot Y^c$ and $B' = P^s \cdot Q^c$.

5. The verifier calculates $c' = H_3(X, Y, P, Q, A', B')$ and checks $c' \overset{?}{=} c$.

If $c' = c$, then the verifier can be assured with overwhelming probability (except for negligible soundness error of size $1/q$) that the discrete logs $\log_X(Y)$ and $\log_P(Q)$ are equal. We use this proof system to verify that tokens in our protocol are all signed with the same private key. In the following, we will denote the computation of the proof D_k by $D_k \leftarrow \mathsf{DLEQ}_k(X, Y, P, Q)$ where k is the common exponent.

An important property of NIZK proofs such as this is that an entity who has control over the random oracle H_3 can forge proofs without knowing k: this entity picks random $c, s \leftarrow_{\$} \mathbb{Z}_q$, computes $A = X^s \cdot Y^c$ and $B = P^s \cdot Q^c$, and programs the random oracle H_3 such that $H_3(X, Y, P, Q, A, B)$ outputs c. This is useful in reduction proofs, as we will see later.

3.2.1 Batch DLEQ proofs

Henry [16] showed that it was possible to batch the above proof system to verify many instantiations in one go. We detail a simplified parallelized Schnorr protocol for the common-exponent case from Henry and Goldberg [15] with adaptations for non-interactivity. Let m denote the number of instantiations of the DLEQ proof system above; that is, for all $i \in \{1, \ldots, m\}$ we aim to prove that $D_{k,i} \leftarrow \mathsf{DLEQ}_k(X, Y, P_i, Q_i)$ is a valid proof for each proof instance (P_i, Q_i) with common X and Y, with $Y = X^k$. Assume that the prover and verifier are aware of the same parameters as above.

1. The prover calculates a seed
$$w \leftarrow H(X, Y, \{P_i\}_{i\in[m]}, \{Q_i\}_{i\in[m]})$$
where $H : \{0,1\}^* \to \mathbb{Z}_q$ is a random oracle.
2. The prover seeds a pseudorandom number generator $PRNG : \mathbb{Z}_q \to \mathbb{Z}_q^m$ with w to sample $c_1, \ldots, c_m \leftarrow PRNG(w)$.
3. The prover generates composite group elements:
$$M = (P_1^{c_1}) \cdot \ldots \cdot (P_m^{c_m}),\ Z = (Q_1^{c_1}) \cdot \ldots \cdot (Q_m^{c_m})$$
and sends the proof $(c, s) \leftarrow \mathsf{DLEQ}_k(X, Y, M, Z)$ to the user.
4. The user recalculates w using knowledge of the P_i and Q_i, samples $c_1, \ldots, c_m \leftarrow PRNG(w)$, and recomputes M and Z as in the previous step.
5. The user then verifies that $\log_X(Y) \overset{?}{=} \log_M(Z)$ using (c, s) as in the procedure above.

That this verification procedure is valid is immediate from the fact that $Q_i = P_i^k$ for each $i \in [m]$. That it is sound and zero-knowledge (ZK) follows from Henry [16, Thm. 3.17]. We denote a batched proof, $\bar{D}_k$, for m instances by
$$\bar{D}_k \leftarrow m\text{-}\mathsf{DLEQ}_k(X, Y, \{P_i\}_{i\in[m]}, \{Q_i\}_{i\in[m]})$$
where k is again the common exponent.

4 Overview of our VOPRF protocol

Before we describe our adaptations to the workflow from Section 2 to reduce the number of user challenges, we give details on the underlying cryptographic protocol that we will use. As mentioned previously, our VOPRF is based on the 2HashDH-NIZK realization provided by Jarecki et al. [18] in the random oracle model. We enhance the construction to have two extra features:

– a batch DLEQ proof verifying a commitment to the same private key for all VOPRF evaluations;
– a redemption phase where the user and server establish a shared key based on the VOPRF output and verify a MAC.

We first give an overview of how the protocol is performed; in Section 5 we give a more formal treatment along with correctness and security guarantees.

Let $\mathbb{G}$ be a group of prime order q with a generator denoted by X. The secret key is chosen as an element $k \leftarrow_{\$} \mathbb{Z}_q$ and a 'public key' is computed as $Y = X^k$. Let H_1, H_2, H_3 be three hash functions modeled as random oracles: H_1 hashes into the non-identity elements $\mathbb{G}^*$ of the group $\mathbb{G}$, H_2 hashes into strings of length $\kappa = \kappa(\lambda)$, and H_3 hashes into $\mathbb{Z}_q$. Let $\mathsf{MAC}_K(\cdot)$ denote a secure MAC algorithm, keyed by some key K. The parameters q, X, and Y are known to both parties but only the server has possession of k.

Our protocol has two phases: a signing phase and a redemption phase; the following subsections describe each of these phases respectively. The user seeks access to a particular resource and generates and redeems tokens accordingly. The server signs tokens and verifies redemptions to allow access to said resource.

4.1 Token signing

1. The user samples N random values $t_1, \dots, t_N \leftarrow\!\!\$ \{0,1\}^\lambda$ and N 'blinding factors' $r_1, \dots, r_N \leftarrow\!\!\$ \mathbb{Z}_q$, where N is the size of a 'batch'.
2. The user computes $T_i = H_1(t_i)$ and $P_i = T_i^{r_i}$ and sends P_i to the server for each $i \in [N]$.
3. The server computes $Q_i = P_i^k$ and a batched proof

$$\bar{D}_k \leftarrow N\text{-}\mathsf{DLEQ}_k(X, Y, \{P_i\}_{i\in[N]}, \{Q_i\}_{i\in[N]})$$

which is computed using the hash function H_3.

4. The server sends $(\{Q_i\}_{i\in[N]}, \bar{D}_k)$ to the user
5. The user verifies $\bar{D}_k$ as in Section 3.2
6. The user unblinds each Q_i by calculating $Q_i^{1/r_i} = T_i^k = W_i$ and stores pairs of the form (t_i, W_i).

4.2 Token redemption

1. The user calculates some request binding data R.[2]
2. The user pops a stored, unspent pair (t, W).
3. The user calculates a shared key $K \leftarrow H_2(t, W)$ and sends a pair of the form $(t, \mathsf{MAC}_K(R))$ to the server.
4. The server recalculates their observed request binding data R'.
5. The server checks that t has not yet been redeemed.
6. The server calculates $T' = H_1(t)$, $W' = (T')^k$ and $K' = H_2(t, W')$
7. The server checks that $\mathsf{MAC}_{K'}(R') \overset{?}{=} \mathsf{MAC}_K(R)$ and stores t for checking double-spending later.
8. If verification passes then the server provides the user with the requested resource.

4.3 Overview of requirements

For our protocol to be deemed *correct*, we require that any MAC value that is computed using a key that is derived from (t_i, W_i) received in the signing phase will ultimately be verified successfully by the server in the redemption phase.

For security, we require an *unlinkability* property — informally requiring that any given signing phase is unlinkable to an instantiation of a redemption phase — and a *one-more-token* security property; requiring that the client is unable to forge validly signed tokens using information it learns in the signing phase.

2 It must be possible for the server to generate the same R where R is specifically bound to the resource that the user requests.

5 Concrete protocol

To illustrate our two protocols we use a series of generic algorithms and show how these can be instantiated using our scheme. We can then define a working protocol with correctness and security guarantees based on standard assumptions.

5.1 Scheme

Let $\mathbb{G}$ be a group (written multiplicatively) of prime order q with generator X. Let $H_1 : \mathbb{Z}_q \to \mathbb{G}^*$, $H_2 : \mathbb{Z}_q \times \mathbb{G} \to \{0,1\}^\kappa$ and $H_3 : \mathbb{G}^6 \to \mathbb{Z}_q$ be the hash functions modeled as random oracles previously, and let $\mathsf{sk} = k \in \mathbb{Z}_q$. The hash function H_3 is used in the computation of the batch DLEQ proof below.

- $T \leftarrow \mathsf{GenToken}(t)$: Output $H_1(t)$.
- $(\widetilde{T}, r) \leftarrow \mathsf{Blind}(T)$: Sample $r \leftarrow\!\!\$ \mathbb{Z}_q$ and output $(\widetilde{T}, r) = (T^r, r)$.
- $T' \leftarrow \mathsf{Unblind}(\widetilde{T}', r)$: Output $T' = (\widetilde{T}')^{1/r}$.
- $f_k(\widetilde{T}) \leftarrow \mathsf{Sign}(\widetilde{T}, k)$: Output $f_k(\widetilde{T}) = \widetilde{T}^k$.
- $c_k \leftarrow \mathsf{Commit}(k)$: Output $c_k = Y = X^k$.
- $\pi_k \leftarrow \mathsf{Prove}(k, c_k, \{\widetilde{T}_i\}_{i\in[m]}, \{f_k(\widetilde{T}_i)\}_{i\in[m]})$: Output

$$\bar{D}_k = m\text{-}\mathsf{DLEQ}_k(X, Y, \{P_i\}_{i\in[m]}, \{Q_i\}_{i\in[m]}),$$

  where $P_i = \widetilde{T}_i$ and $Q_i = \widetilde{T}_i^{k_i}$.
- $b \leftarrow \mathsf{VerifyPrf}(\pi_k, c_k, \{\widetilde{T}_i\}_{i\in[m]}, \{f_k(\widetilde{T}_i)\}_{i\in[m]})$: Output the result of verifying $\pi_k = \bar{D}_k$ using $(X, c_k, \{\widetilde{T}_i\}_i, \{f_k(\widetilde{T}_i)\}_i)$.
- $K \leftarrow \mathsf{sKGen}(a, B)$: Outputs $K = H_2(a, B)$, where K is a permissible key for MAC algorithm below.
- $s \leftarrow \mathsf{MAC}_K(R)$: Outputs $\mathsf{HMAC}(K, R)$ where HMAC is the standard HMAC algorithm accepting key $K \in \{0,1\}^\kappa$ and some input data R.
- $b \leftarrow \mathsf{Verify}(t, R, M, k)$: Check that $\mathsf{MAC}_{\mathsf{sKGen}(t, \mathsf{Sign}(\mathsf{GenToken}(t), k))}(R) \overset{?}{=} M$ and output '1' if it verifies, and '0' otherwise.

5.1.1 Other algorithms

We use the following algorithms without requiring explicit instantiation as they are sufficiently generic.

- $\mathsf{Store}(x, B)$: Stores the value x in the data store B
- $b \leftarrow \mathsf{Check}(x, B)$: Checks if x is stored in B and outputs $b \in \{0,1\}$ where $b = 1$ if x is stored.
- $x_0 \leftarrow B.pop()$: Outputs the item x_0 with index 0 and removes it from a data store B. Also re-orders

C		**S**
$t \leftarrow_{\$} \mathbb{Z}_q$		
$T \leftarrow \mathsf{GenToken}(t)$		
$\widetilde{T}, r \leftarrow \mathsf{Blind}(t)$		
	$\xrightarrow{\widetilde{T}}$	
		$f_{\mathsf{sk}}(\widetilde{T}) \leftarrow \mathsf{Sign}(\widetilde{T}, \mathsf{sk})$
		$\pi_{\mathsf{sk}} \leftarrow \mathsf{Prove}(\mathsf{sk}, c_{\mathsf{sk}}, \widetilde{T}, f_{\mathsf{sk}}(\widetilde{T}))$
	$\xleftarrow{(f_{\mathsf{sk}}(\widetilde{T}), \pi_{\mathsf{sk}})}$	
$b \leftarrow \mathsf{VerifyPrf}(\pi_{\mathsf{sk}}, c_{\mathsf{sk}}, \widetilde{T}, f_{\mathsf{sk}}(\widetilde{T}))$		
$b \overset{?}{=} 1$		
$f_{\mathsf{sk}}(T) \leftarrow \mathsf{Unblind}(f_{\mathsf{sk}}(\widetilde{T}), r)$		
$\mathsf{Store}((t, f_{\mathsf{sk}}(T)), B_{\mathbf{C}})$		

Fig. 2. An overview of the token signing protocol.

C		**S**
$(t, f_{\mathsf{sk}}(T)) \leftarrow B_{\mathbf{C}}.pop()$		
$K \leftarrow \mathsf{sKGen}(t, f_{\mathsf{sk}}(T))$		
$s \leftarrow \mathsf{MAC}_K(R)$		
	$\xrightarrow{(t,s)}$	
		$\mathsf{Check}(t, B_{\mathbf{S}}) \overset{?}{=} 0$
		$T' \leftarrow \mathsf{GenToken}(t)$
		$K' \leftarrow \mathsf{sKGen}(t, f_{\mathsf{sk}}(T'))$
		$s' \leftarrow \mathsf{MAC}_{K'}(R)$
		$s' \overset{?}{=} s$
		$\mathsf{Store}(t, B_{\mathbf{S}})$
	$\xleftarrow{res}$	

Fig. 3. An overview of the token redemption protocol.

B so that the item x_i with index i is moved to index $i-1$ for $i > 0$.

5.2 Formal protocol

We describe a formalized protocol, P, using the algorithms that we have defined above, splitting P into a 'signing' and 'redemption' phase as we have done previously. The notation that we use could be expanded to a generic framework for such protocols using different underlying schemes (e.g. using explicit blind signature schemes).

5.2.1 Signing phase

The signing phase between a server **S** and a client **C** can be constructed as described in Fig. 2. We assume that sk is the signing key known to **S** and that $c_{\mathsf{sk}} \leftarrow \mathsf{Commit}(\mathsf{sk})$ is known to both. The description below is illustrated for the signing of one token but can easily be generalized for more.

We write $f_{\mathsf{sk}}(T) \leftarrow \mathsf{P.SignPhase}(t)$ to indicate a successful signing phase between **C** and **S** using the notation above. If the signing phase is not successful, then the output will be denoted by $\perp$ instead.

5.2.2 Redemption phase

The redemption phase allows **C** to redeem a signed token (obtained in Fig. 2) in order to receive a resource *res*. Let $R \in \{0,1\}^\ell$ be the request binding data associated with this request, known to both parties. We give an explicit description in Fig. 3.

We write $b \leftarrow \mathsf{P.RedeemPhase}(t, f_{\mathsf{sk}}(T))$ to indicate the response of the server in the redemption phase. We say that $b = 1$ if *res* is received by the client and $b = 0$ if $\perp$ is received.[3] If $b = 1$ (resp. 0), then we say that the redemption phase is successful (resp. unsuccessful).

5.3 Correctness

We give a generic correctness definition for protocols P implementing the signing and redemption phases from Fig. 2 and Fig. 3 respectively. Correctness is evaluated with respect to the scheme $\mathbf{\Gamma}$ implementing the algorithms from Section 5.1.

Definition 1. *The protocol* P *is* correct *if:*

$$\Pr\left[1 \leftarrow \mathsf{P.RedeemPhase}(t, f_{\mathsf{sk}}(T)) \,\middle|\, \substack{t \leftarrow_{\$} \mathbb{Z}_q \\ f_{\mathsf{sk}}(T) \leftarrow \mathsf{P.SignPhase}(t)}\right] > 1 - \mathsf{negl}(\lambda)\,.$$

We can therefore prove the correctness of P using the explicit properties of the scheme $\mathbf{\Gamma}$ that we use.

Lemma 1. *Let $\mathbf{\Gamma}$ detail the instantiation of the scheme given in Section 5.1. Then $\mathbf{\Gamma}$ instantiates a protocol* P *satisfying the notion of correctness from Definition 1.*

Proof. Let $(\widetilde{T}, r) \leftarrow_{\$} \mathbf{\Gamma}.\mathsf{Blind}(\mathbf{\Gamma}.\mathsf{GenToken}(t))$ where $t \leftarrow_{\$} \mathbb{Z}_q$, therefore $\widetilde{T} = T^r$.[4] Let $\mathsf{sk} = k$; when **S** receives $\widetilde{T}$ it first computes $\widetilde{T}^k = f_k(\widetilde{T}) \leftarrow \mathbf{\Gamma}.\mathsf{Sign}(\widetilde{T}, k)$. It then computes $1 - \mathsf{DLEQ}_k(X, Y, \widetilde{T}, \widetilde{T}^k) = \bar{D}_k = \mathbf{\Gamma}.\mathsf{Prove}(k, c_k, f_k(T))$ and sends $(\widetilde{T}^k, \bar{D}_k)$ back to **C**.

3 **C** receives $\perp$ if P is aborted due to a failed verification.

4 We prove correctness in the case of one token but this is easily generalizable to m.

Since $\bar{D}_k$ is a valid DLEQ proof as given in Section 3.2, then $b \leftarrow \mathbf{\Gamma}.\mathsf{VerifyPrf}(\bar{D}_k, X, Y, \widetilde{T}, \widetilde{T}^k)$ and $b = 1$ (as long as the commitment X, Y is valid). Therefore, the client computes $T^k \leftarrow \mathbf{\Gamma}.\mathsf{Unblind}(\widetilde{T}^k, r)$ and the signing phase is complete.

For the redemption phase, $\mathbf{C}$ computes a shared key $K \leftarrow H_2(t, T^k)$ and sends $(t, \mathbf{\Gamma}.\mathsf{MAC}_K(R))$ to $\mathbf{S}$.[5] Given t, $\mathbf{S}$ calculates $T^k = \mathbf{\Gamma}.\mathsf{Sign}(\mathbf{\Gamma}.\mathsf{GenToken}(t))$ and computes $\mathbf{\Gamma}.\mathsf{MAC}_{H_2(t,T^k)}(R)$. Since $K = H_2(t, T^k)$ by definition, we conclude that $\mathbf{\Gamma}.\mathsf{MAC}_K(R) = \mathbf{\Gamma}.\mathsf{MAC}_{H_2(t,T^k)}(R)$ and thus $1 \leftarrow \mathsf{P.RedeemPhase}(t, T^k)$, settling correctness. □

5.4 Security properties

Here we give a treatment of the security model and properties that we require the protocol P to uphold.

- We show that our protocol achieves *unlinkability* — a notion that states that the signing and redemption phases are completely independent (Theorem 1).
- Also, we achieve *one-more-token* security — this proves that a malicious client cannot use knowledge of validly signed tokens to forge a signature on a newly generated token (Theorem 2).
- In Section 5.5 we also show that the server must commit to a single signing key — this prevents the server from using client-specific key pairs in an attempt to partition the clients' anonymity set.[6]

We argue that these properties are fundamental to the operation of our protocol. Unlinkability provides the client with the assurance that no server is going to be able to link disparate sessions together — this is the main premise of our solution. One-more-token security prevents the client from blatantly subverting the protocol to commit token forgery and thus bypass the security mechanisms in the current workflow. The proof of unlinkability is unconditional, and the proof of one-more-token security follows from a standard assumption on the El Gamal cryptosystem.

There are undoubtedly more security properties that we could choose to prove here. This is not to say that we cannot prove more interesting properties, such as design modifications that may prevent tokens being used by other identities. We regard analysis of wider security models as an interesting and important avenue for future work.

5.4.1 Security model

An adversary attempting to break the security of the protocol P has oracle access to the functions $\mathsf{P.SignPhase}(\cdot)$ and $\mathsf{P.RedeemPhase}(\cdot)$ and we define the 'view' of the adversary $\mathcal{A}$ to be $\mathcal{A}.view(\mathsf{P.SignPhase}(\cdot))$ and $\mathcal{A}.view(\mathsf{P.RedeemPhase}())$, respectively. By view, we essentially mean the set of all messages and outputs that $\mathcal{A}$ witnesses during a phase execution. Therefore if $\mathcal{A}$ plays the role of the server then $\mathcal{A}.view(\mathsf{P.SignPhase}(t))$ equates to $(\widetilde{T}, f_{\mathsf{sk}}(\widetilde{T}), c_{\mathsf{sk}}, \pi_{\mathsf{sk}})$ for $c_{\mathsf{sk}} \leftarrow \mathbf{\Gamma}.\mathsf{Commit}(\mathsf{sk}), \pi_{\mathsf{sk}} \leftarrow \mathbf{\Gamma}.\mathsf{Prove}(\mathsf{sk}, c_{\mathsf{sk}}, \widetilde{T}, f_{\mathsf{sk}}(\widetilde{T}))$. Additionally, $\mathcal{A}.view(\mathsf{P.RedeemPhase}(t, f_{\mathsf{sk}}(T)))$ equates to $(t, T, f_{\mathsf{sk}}(T))$. We put no restrictions on how $\mathcal{A}$ makes calls to the oracles apart from that it runs in PPT.

5.4.2 Unlinkability

The notion of unlinkability captures the inability of an adversarial server $\mathbf{S}$ to link together a redemption phase of the protocol to any individual signing phase. A protocol satisfying unlinkability is necessary for meeting the anonymity guarantees that we require.

Definition 2. (Unlinkability) *The signing and redemption phases of the protocol* P *are* unlinkable *if the view of the server $\boldsymbol{S}$ in the signing phase is indistinguishable from a uniformly sampled element in $\mathbb{G}$ and independent of the view of $\boldsymbol{S}$ in the redemption phase.*

Intuitively and in terms of our protocol, when a seed t is revealed in a redemption phase the best chance that $\mathbf{S}$ has in linking this to a particular signing interaction is randomly guessing.

A game-based definition of this security requirement would see the challenger initiate two signing phases with an adversarial server. The challenger would then use the tokens from one of the signing phases in a redemption phase with the adversary. The adversary would output a bit corresponding to which signing phase it thinks the tokens were obtained from. A scheme secure in this game would require that the adversary had a probability of success that was negligibly different from 1/2.

A signing protocol trivially satisfies the unlinkability security property if the output $\widetilde{T}$ of the algorithm

5 We safely assume that the request binding data R is public since it will be available via the request (e.g., specific headers and the HTTP path).

6 We do not prove this explicitly as it follows naturally from the definition of the proof that we use (Section 3.2.1).

$\mathsf{Blind}(T)$ for some token T is uniformly distributed. This makes up the entire view of the server. For this reason, we do not elaborate any further on this formalization.

Theorem 1. P *is unconditionally unlinkable w.r.t.* $\mathbf{\Gamma}$.

Proof. In our scheme, $\widetilde{T} = T^r$ for a randomly selected $r \leftarrow_\$ \mathbb{Z}_q$. Since T is a generator of $\mathbb{G}$, $\widetilde{T}$ is a uniform element of $\mathbb{G}$, and contains no information about T or t. As such, since $\mathbf{S}.view(\mathsf{P.SignPhase}(t))$ is dependent only on $\widetilde{T}$ and $\mathbf{S}.view(\mathsf{P.RedeemPhase}(t, f_{\mathsf{sk}}(T)))$ is dependent on t and T, the phases are independent. □

5.4.3 One-More-Token security

The notion of *one-more-token* security enshrines the inability of the client to access the secret signing key of the server in a 'meaningful' way. By meaningful, we mean being able to use knowledge of signed tokens, received during a signing phase, to sign more tokens.

Definition 3. (One-More-Token security) *Our scheme has* one-more-token security *if a PPT client, after receiving ℓ signed tokens from the server, cannot successfully redeem $\ell+1$ tokens, except with negligible probability.*

We will reduce this security property to a standard assumption on the El Gamal encryption scheme:

Definition 4. (One-More-Decryption security of El Gamal) *A challenger provides to an adversary a description of a group $\mathbb{G}$, its prime order q, a generator X, and a public key $Y = X^k$ where $k \leftarrow_\$ \mathbb{Z}_q$ is known only to the challenger.*

The challenger selects $\ell + 1$ messages $M_i \leftarrow_\$ \mathbb{G}$, encrypts each with El Gamal to form the ciphertexts $\{(C_i, D_i) = (X^{r_i}, Y^{r_i} \cdot M_i\}_{i\in[\ell+1]}$, and sends the ciphertexts to the adversary.

The adversary can make ℓ oracle decryption queries to the challenger with messages of its choice.[7]

The adversary wins if it outputs all $\ell+1$ plaintexts.

El Gamal is secure against a one-more-decryption attack *if any PPT adversary has at most a negligible (in* $\lg q$*) probability of winning the game.*

We note that Schnorr and Jakobsson [28] prove that no PPT adversary in the generic group model has a non-negligible advantage in winning a strictly weaker version of this game.[8] An adversary who can win the above game can easily win the weaker game as well.

Theorem 2. *If El Gamal is secure against a one-more-decryption attack, then our scheme has one-more-token security.*

Proof. Given an adversary $\mathcal{A}$ who can get ℓ tokens signed by the server, and then redeem $\ell+1$, we will produce an adversary $\mathcal{B}$ who wins the one-more-decryption game against El Gamal.

$\mathcal{B}$ starts the game by being given $\mathbb{G}$, q, X, Y, and $\{(C_i, D_i)\}_{i\in[\ell+1]}$. $\mathcal{B}$ initiates $\mathcal{A}$, using $(\mathbb{G}, q, X)$ as the group description and Y as the commitment to the signing key. $\mathcal{B}$ also selects a random permutation π on $\mathbb{Z}_q$, but does not reveal it to $\mathcal{A}$.

$\mathcal{A}$ can make two kinds of queries: polynomially many random oracle queries to H_1, H_2, and H_3, and at most ℓ token signing queries.

When $\mathcal{A}$ makes a token signing query with blinded token $\widetilde{T}_i$, $\mathcal{B}$ selects a random $F_i \leftarrow_\$ \mathbb{G}$, and sends the ciphertext $(\widetilde{T}_i, F_i)$ to the decryption oracle, which returns $M_i = F_i/\widetilde{T}_i^k$. $\mathcal{B}$ then returns $F_i/M_i = \widetilde{T}_i^k$ to $\mathcal{A}$. It provides a forged DLEQ proof, which it can do because it can program the responses of H_3.

When $\mathcal{A}$ makes a random oracle request for $H_1(t)$, $\mathcal{B}$ replies with $\prod_{j\in[\ell+1]} C_j^{\left(a^{j-1}\right)}$, where $a = \pi(t)$. Only with negligible probability $1/q$ will this value be the identity element of $\mathbb{G}$, in which case $\mathcal{B}$ aborts with failure.

When $\mathcal{A}$ makes a random oracle request for $H_2(t, B)$, $\mathcal{B}$ stores (t, B, K) in a table for a randomly selected K, and replies with K. If the same $H_2(t, B)$ is queried again, the same K will be returned.

When $\mathcal{A}$ makes a random oracle request for H_3, $\mathcal{B}$ programs it to forge the DLEQ proof, as above.

At the end of the game, $\mathcal{A}$ will redeem $\ell + 1$ tokens $\{(t_i, R_i, \mathsf{MAC}_{H_2(t_i,B_i)}(R_i))\}_{i\in[\ell+1]}$. $\mathcal{B}$ uses the H_2 table to look up the value of B_i used to generate the MAC key in each token, and it will be the case that $B_i = H_1(t_i)^k = \prod_{j\in[\ell+1]} \left(C_j^k\right)^{\left((\pi(t_i)^{j-1}\right)}$ for each $i \in [\ell + 1]$. If V is the (Vandermonde, and thus invertible) matrix with $V_{ij} = \pi(t_i)^{j-1}$, and U is the inverse of V, then

7 Unlike in a CCA2 game, there is no restriction on querying one of the challenge ciphertexts.

8 In their version, the adversary is given $d > \ell$ ciphertexts, and the correct plaintexts in a permuted order, and must simply match $\ell + 1$ plaintexts to their corresponding ciphertexts using at most ℓ queries to the decryption oracle.

it will be the case that $\prod_{j\in[\ell+1]} B_j^{U_{ij}} = C_i^k$ for each $i \in [\ell+1]$. $\mathcal{B}$ then outputs $M_i = D_i/C_i^k$ for each $i \in [\ell+1]$ and wins the game. □

5.5 Key-consistency

A potential attack on client anonymity is for the server to use different values of the secret key k for different clients. A server that does this could then match a token redemption to the blinded token signing operation that generated it. To thwart this attack, we ensure that the server commits to the key it uses, and that clients are assured that the same commitment is visible to all users (for example, by placing it in the Tor consensus). The batch DLEQ proof then assures the client that all of the tokens are signed using the same key that was used to form the public commitment. Henry [16, Thm. 3.17] establishes the security of this batch proof.

6 Reducing challenges

In Section 5 we described an instantiation of a generic blind signing protocol based on a VOPRF design. In this section, we show how the current client-edge workflow from Section 2 can be augmented to include our privacy-preserving token protocol. The end result is an adapted workflow that results in many fewer challenges for users, specifically a reduction by a factor of N, where a user receives N tokens from a signing phase. We also adapt the role of the validator **V** to include the signing and verification of tokens that are used in the signing and redemption phases of our protocol.

6.1 Adapted workflow

Recall that the workflow from Section 2 requires a user to solve a challenge for every access (modulo the use of cookies). To reduce the number of challenges exposed to the client without sacrificing the unlinkability requirement, we embed the protocol that we constructed in Section 5 into this workflow. This allows a client to receive N unlinkable tokens that can then be used to bypass challenges on future accesses. We present two high-level workflows in Fig. 4 and Fig. 5 embedding the protocols that were discussed in Section 5.

In summary, for every signing workflow as in Fig. 4 completed by a client **C**, **C** can engage in N redemption workflows. These N redemptions provide unlinkable to-

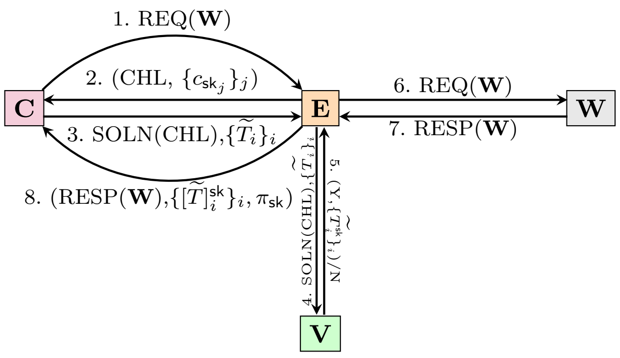
1. REQ(**W**)
2. (CHL, $\{c_{\mathsf{sk}_j}\}_j$)
3. SOLN(CHL),$\{\widetilde{T}_i\}_i$
8. (RESP(**W**),$\{[\widetilde{T}]_i^{\mathsf{sk}}\}_i, \pi_{\mathsf{sk}}$)
C
E
W
V
6. REQ(**W**)
7. RESP(**W**)
4. SOLN(CHL),$\{\widetilde{T}_i\}_i$
5. (Y,$\{\widetilde{T}_i^{\mathsf{sk}}\}_i$)/N

Fig. 4. Adapted workflow for token issuance. Here, T_i = GenToken(t_i) for $t_i \leftarrow\$ \mathbb{Z}_q$, $\widetilde{T}_i \leftarrow$ Blind(T_i), and $\widetilde{T}_i^{\mathsf{sk}} \leftarrow$ Sign($\widetilde{T}_i$, sk) is the signed token returned by the edge. Also, $\pi_{\mathsf{sk}} \leftarrow$ Prove(sk, c_{sk}, $\{T_i\}_i$, $\{f_{\mathsf{sk}}(\widetilde{T}_i)\}\}_i$ is the key-consistency proof (Section 5.5); $\{c_{\mathsf{sk}_j}\}_j$ is a list of commitments corresponding to the currently accepted tokens (Section 6.2).

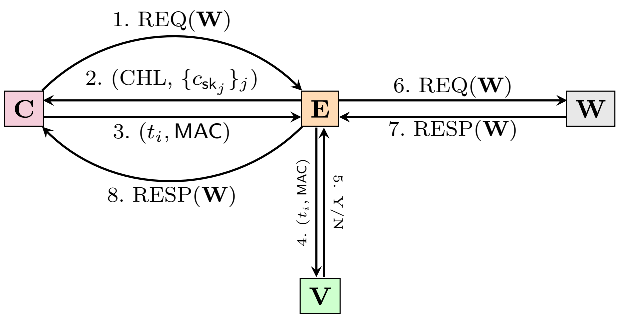

Fig. 5. Adapted workflow for token redemption, notation is as in Fig. 4. The verifier **V** now verifies the pair (t, MAC) instead of the challenge solution.

kens, and do not require human participation to solve challenges. A successful redemption could be exchanged for a single-origin cookie. These cookies allow clients to bypass future challenges for the same domain without using more redemption tokens, and without client browsing being linkable across multiple domains.

6.2 Key rotation

It is important that the server **S** is able to implement a key rotation policy, both in the case that a key has exceeded its lifetime, and to reduce the potency of some attack vectors (Section 8). Key rotation in these workflows requires the server to publish a new commitment $c_{k^*} = Y^* = X^{k^*}$ to each client, where k^* is the new secret key that is used. This is necessary so that the clients can verify the key-consistency proofs that are sent by **S** in the signing phase.

Our workflow in Fig. 4 is adapted so that the server can have multiple keys in play at any one time. This prevents key rotation from immediately rendering all possessed tokens useless. Essentially, **S** sends a list of commitments $\{c_{\mathsf{sk}_j}\}_j$ corresponding to all secret keys that it accepts. The client checks the list of commitments and uses a corresponding pair to verify the key-consistency proof that it receives in the later stages. When a key is phased out for good, the server simply removes the commitment from the acceptance list.

When multiple keys are in use, this acts to siphon the user base into smaller groups and thus increase the deanonymization potential. As a result, the list of keys that a service provider should be kept to a relatively low number; e.g., 2 or 3 at any one time.

7 Browser extension implementation

To instantiate the workflows shown in Section 6 we have created a client-side browser extension named “Privacy Pass” and have written compatible server-side support in Go that can be run on edge servers. The browser extension handles the cryptographic operations that need to be carried out on the client side. Our browser extension is written in JavaScript and is compatible with Chrome and Firefox; additionally, the code is open source and available online.[9]

7.1 Cloudflare deployment case study

Support for Privacy Pass has been written into existing Cloudflare infrastructure in order to reduce the number of CAPTCHAs that are faced by honest users. As a consequence, this reduces the number of CAPTCHAs that need to be solved by a factor directly proportional to the number N of signed tokens received for each initial solution. Currently $N = 30$ but the server supports up to $N = 100$. We use this upper bound to reduce the capacity for malicious clients to hoard tokens; see Section 8.2. The value $N = 30$ was chosen because it provides a balance between usability, performance, and token hoarding considerations.

A second benefit is that sub-resources hosted behind Cloudflare CAPTCHAs will now be accessible for clients

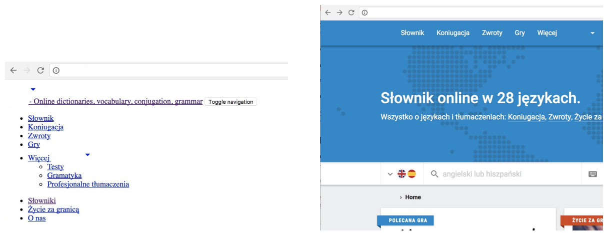

Fig. 6. Left: Website with CSS hosted behind CAPTCHA. **Right:** Website when using Privacy Pass.

— even though clients would not previously have been able to submit CAPTCHA solutions for these URLs. For example, in Fig. 6 we show a website with various CSS files hosted on a different origin, and protected by Cloudflare, before and after using Privacy Pass. Since the user has no way of solving a CAPTCHA for the CSS files (other than locating and navigating to the URL), the website is displayed without the CSS styling and thus appears with clear errors. Privacy Pass is able to redeem signed tokens with these resources instead and thus the user is now able to view the website as it was intended.

While this is undoubtedly a flaw with the implementation of CAPTCHAs in general, it is an example of Privacy Pass being leveraged to make the browsing experience more pleasant and error-free for users.

7.1.1 Signing

Firstly, assume that a client **C** (with sufficiently poor IP reputation $\nu_{\mathbf{C}} > \nu$ for some threshold ν) is attempting to connect via HTTP to a Cloudflare-protected website **W** using a browser with Privacy Pass installed and with no passes available. When a challenge CAPTCHA is solved by **C** then Privacy Pass first generates blinded tokens (as in Message 1 of Fig. 2) and appends these tokens to the body of the HTTP request containing the solution. This amended request is sent asynchronously and the edge verifies whether the CAPTCHA solution is correct. If it is, the blinded tokens are signed and a key-consistency proof is generated. The edge now creates a new HTTP response indicating that the solution was valid (i.e., status code 200) and returns a single-origin clearance cookie in the header, and the signed tokens and the proof in the body of the response.

Privacy Pass parses the signed tokens and the key-consistency proof, which is validated. If the proof is valid, then the signed tokens are unblinded and stored

9 https://privacypass.github.io/

Table 1. Benchmarks (ms) for operations that are instantiated using our implementation from Section 5. N is the number of tokens to be signed in a batch.

	Operation	Timings
Client	Token generation (GenToken() & Blind())	$120 + 64 \cdot N$
	Verify DLEQ (VerifyPrf())	$220 + 110 \cdot N$
	Total signing request	$340 + 180 \cdot N$
	Total redeem request	57
Server	Signing (Sign())	$0.04 + 0.20 \cdot N$
	DLEQ Proof (Prove())	$0.32 + 0.55 \cdot N$
	Total signing	$1.48 + 0.87 \cdot N$
	Total redemption	0.8

for future use in the local storage of the browser. After storage is completed, Privacy Pass reloads the current page, which will now succeed because of the single-domain clearance cookie.

7.1.2 Redemption

Assume that the client **C** has gained tokens as above and that it has attempted to access another edge-protected website $\mathbf{W}'$. A CAPTCHA will be served and the header `cf-chl-bypass` will be present with value '1' in the edge response. Privacy Pass uses the presence of this header to initiate the token redemption procedure when it has signed tokens stored.[10]

Privacy Pass retrieves an unspent token $(t, f_{\mathsf{sk}}(T))$ from local storage and starts forming a new HTTP request for $\mathbf{W}'$. Recall that in the redemption protocol of Fig. 5 the client is required to construct a MAC over some request binding data; the key for the MAC is derived from the unspent token pair that was retrieved (see Fig. 3). The request binding data R in this case is precisely the contents of the 'Host' header for the request concatenated with the HTTP Path.[11] The pair $(t, \mathsf{MAC}_K(R))$ where $K = \mathsf{sKGen}(t, f_{\mathsf{sk}}(T))$ is then appended to the HTTP request as the value of a header named '`challenge-bypass-token`' and the request is then sent to the edge.

The edge takes the request and verifies the value of the '`challenge-bypass-token`' header using the procedure in Fig. 3. If verification succeeds, **E** serves the content of $\mathbf{W}'$ to **C** and a single-origin clearance cookie.

10 This header was also present in the first case but was not actually used.

11 This is not quite a unique identifier for the request but is manageable for preventing various MitM attacks.

Table 2. Size of additional information (bytes) for requests, sent as part of the protocol implementation from Section 5. N is the number of tokens to be signed in a batch.

Operation	Size (bytes)
Signing request ($\mathbf{C} \to \mathbf{E}$)	$57 + 63 \cdot N$
Signing response ($\mathbf{E} \to \mathbf{C}$)	$295 + 121 \cdot N$
Redemption request ($\mathbf{C} \to \mathbf{E}$)	396

7.2 Benchmarks

We illustrate the additional client and server overheads incurred due to the additional structure used for performing the protocol. We compile our data from a series of benchmarks taken over the key cryptographic operations that are computed during a protocol run.

We use the NIST P-256 elliptic curve; for each $N = 5, 10, 15, \ldots, 100$, we have the client generate N tokens at a time and each of these is signed by the server, including generation and checking of batch DLEQ proofs, and redemption of a single token. We repeated this experiment 100 times. In Table 1 we provide a set of benchmarks for the operations that are computed for the client and server. In Table 2 we provide the size of the additional data for requests and responses due to Privacy Pass. We do not include the redemption response size as Privacy Pass does not change this flow.[12]

As is clear from our results, the overheads incurred from using Privacy Pass are quite acceptable. In particular, redemptions are quick for both the client and server and require very little added communication. For signing, the biggest costs appear to be the time taken for the client to generate requests and verify server responses. On the other hand, the initial cost for acquiring signed tokens is warranted given that redemptions are so efficient and client proofs-of-work are then avoided for the lifetime of the tokens. Finally, the additional communication load is less than 6 KB in total for $N = 30$, which is unlikely to trouble the browsing experience.

7.3 Cloudflare adoption results

We released the Chrome/Firefox browser extension Privacy Pass on 8 November 2017, and also released the

12 The signing responses are about double the size of the requests, because for performance reasons, the (JavaScript) client compresses the elliptic curve points in the signing request, but the (Go) server does not compress the elliptic curve points in the response, lest the JavaScript client have to do expensive point decompressions.

open-source code for the extension, which would allow a user to locally install the extension in their browser. In the following we will detail various numbers acquired directly in partnership with Cloudflare [8] (we focus only on distinguishing requests coming from Tor and non-Tor users). The number of Chrome users by 28 November 2017 stood at 8499 and the number of Firefox users stood at 3489 — this does not include users manually installing the extension. Averaging over any given 7 days in this period, Cloudflare accepted 1.6 trillion requests with 780 million of those requests coming over the Tor network. Of these requests, 1.04% were challenged in general globally, though 17% of Tor traffic is challenged in the same period.[13] This illustrates the disproportionate occurrences of challenges being shown to Tor users.

Fortunately, Privacy Pass alleviates the burden of these challenges by a factor of N for honest users who would usually complete the challenge. However, Privacy Pass may not result in a factor of N reduction in the total number of challenges witnessed — i.e., we will not see an optimal drop to $(17/N)$% of requests being served challenges in the Tor case. This is due to the fact that these figures also include challenges that are served to requests from users that have no intention of solving the challenge (such as content scrapers).

In terms of operating numbers, the number of redemption phases occurring peaked globally at 2000 per second and for Tor users at 200 per second. Additionally, there were 515 million requests containing clearance cookies used to bypass challenges globally with 34.5 million of those occurring from the Tor network. As a result, 22.58% of requests coming from Tor users contain these clearance cookies. Using these cookies preserves tokens for use only on unseen domains, which is advantageous for users, and they cannot be tracked across domains due to the single-origin policy.

This demonstrates a system that is clearly useful for clients with the extension installed. In the same time-frame, median request and response sizes for Tor users stood at 700–800 bytes and 5–6 KB respectively while the median size of CAPTCHA submissions and responses were less than 1 KB.[14] Consequently, the additional protocol messages do not result in unmanageable request/response sizes.

13 By challenged, we mean the response to the request displays a Cloudflare CAPTCHA.

14 These figures are slightly different to the benchmarks above as they include information specific to Cloudflare that we did not model in the benchmarking tests.

8 Potential attacks and mitigations

In Section 5 we showed that our protocol satisfied various security properties. However, this does not consider a number of out-of-band attacks that may still be able to effectively subvert the protocol if mitigations are not considered. We consider measures that can be put in place to limit the effectiveness of such attacks here.

8.1 Interception of signing requests

A characteristic of our protocol is that we do not currently encode any useful information into the tokens that are signed by the server. This prevents the client from generating tokens that may inadvertently associate metadata with the tokens that may lead to a greater chance of deanonymization/linkability during redemption. Unfortunately, this leads to the issue that tokens are not associated with any one client. This increases the effectiveness of monster-in-the-middle (MitM) adversaries since transmitted tokens are not cryptographically linked to a specific protocol invocation.

For example, consider a MitM adversary $\mathcal{M}$ in the signing phase of the workflow in Fig. 4. When the client completes the given challenge and sends blinded tokens to the server, this message is hijacked by $\mathcal{M}$ who simply forwards the challenge solution under their own identifier along with a set of their own generated tokens. If the challenge does not have some client-binding hijack-prevention mechanism, then $\mathcal{M}$ will receive signed tokens from the server without completing a challenge.

This attack is particularly effective as it removes the requirement for an adversary to authenticate to a service in order to receive tokens. For example, consider if the challenge mechanism were a CAPTCHA and $\mathcal{M}$ was some bot that was ordinarily not able to solve the CAPTCHA. This attack would allow $\mathcal{M}$ to receive tokens without computing a CAPTCHA solution and then bypass future CAPTCHA invocations. In the case of Cloudflare, the CAPTCHA page is protected by TLS so this seems hard to hijack, however this remains a general attack vector.

While this attack is effective, it requires a renewable source of authentications/challenge solutions for it to carry on indefinitely. In this setting, even without Privacy Pass, such an adversary would essentially have access to a CAPTCHA solving farm and thus the need for acquiring tokens to authenticate would be made redundant anyway (except for a small gain in the speed that authentication could be performed). Although the current system requires an *online* adversary looking to bypass a CAPTCHA maliciously, using Privacy Pass it becomes possible to intercept tokens and use them at a more convenient time in the near future (until the keys are rotated). However, as long as the maximum number of signed tokens is fairly small (in the case of Cloudflare only 100 tokens can be signed for each interaction), then the effectiveness of the attack is not especially increased.

8.2 Token accumulation

A related attack avenue would be for clients to 'farm' signed tokens by repeatedly sending challenge solutions to invoke the signing phase of the protocol. This would allow the client to build up a stockpile of unused tokens that they could then redistribute amongst other clients or redeem tokens for a large period of time. Redistribution of token pairs (t_i, W_i) is possible due to the lack of data encoded into them. Building up a large store of tokens could be useful for launching distributed denial of service (DDoS) attacks on service providers.

Clearly solving challenges incurs much larger overheads than the token redemption process using the browser extension. Therefore, clients can invoke many redemptions in the same time that they could solve one challenge previously. Therefore, it is important that pass verification is very efficient for the server to carry out to reduce the threat of a DDoS attack reducing the availability of the server. Fortunately, as we showed above, verification of a token redemption occurs in less than 1 ms and a few hundred bytes of additional transmitted data. Additionally, carrying out these operations on powerful server hardware may lead to even faster running times. However, depending on the size of the stockpile, a client that amasses a huge number of tokens may still be able to cause problems.

For these attacks we implement a number of possible mitigations. Firstly, the low upper bound of 100 tokens signed at a time by Cloudflare means that it would take a large amount of effort to build up a stockpile of tokens large enough to launch a credible DDoS attack ($1,000,000$ tokens would require minimum $10,000$ CAPTCHA solutions).

Finally, as described in Section 6.2, we recommend regular key rotation so that signed tokens are implicitly related to epochs and are thus invalidated frequently. This prevents stockpiles of tokens from being useful for longer than the epoch they belong to. Ideally key rotation would occur over short time periods (such as 3–4 days); a shorter time period directly reduces the number of useful tokens that can be stockpiled and redeemed. In our initial software distribution, key rotation is handled by updating the commitments directly in the browser extension. Moving forward, CDNs could upload their commitments to beacons or other globally visible and consistent locations, such as the Tor consensus.

8.3 Token exhaustion

Privacy Pass uses a finite list of low-entropy characteristics to determine whether a token should be redeemed or not. In the case of Cloudflare CAPTCHAs, the extension looks for the presence of an HTTP response header and particular status code. Unfortunately, this means that it is easy to recreate the characteristics that are required by the extension to sanction a redemption.

To view the attack at its most powerful, consider a sub-resource that embeds itself widely on many webpages that can trigger token redemptions.[15] Such a resource would be able to drain the extension of all its tokens by triggering redemptions until all the tokens were used. While it is unclear why such an attack would be useful, it is important to acknowledge that it is indeed possible to carry out and would thus render the usage of Privacy Pass useless if the sub-resource was especially prevalent.

Our mitigation for this attack lies in the extension itself. First, the implementation of Privacy Pass prevents token redemptions occurring for the same URL in quick succession by keeping track of where spends have occurred. This data is refreshed when a single-domain cookie is removed to allow re-spending at the domain. This prevents a sub-resource from recursively draining tokens after each spend; it also spreads out the redemptions considerably. The sub-resource would have to force cookies to be removed and then schedule a page reload for each token spend.

A coarser method of stopping these attacks — that has not been implemented at time of writing in the Cloudflare deployment — would be to blacklist URLs from spending after one attempt and then refresh this state every time the extension reaches 0 tokens. While this would be a more effective countermeasure, it would also prevent a client from spending tokens on a URL, even if the cookie for the URL were removed. Other po-

15 For example, a popular analytics script or JQuery code.

tential mitigations could include requiring the edge to sign the appropriate header, or the edge stripping the `cf-chl-bypass` header from origin responses.

8.4 Time-based client deanonymization

As with all systems that require some sort of registration, there is the inherent problem that 'early adopter' clients are part of a much smaller anonymity set than is optimal for enforcing privacy constraints. In particular, consider a scenario where some user is amongst the first five to initiate a signing phase with some given CDN, or after key rotation has occurred. Then when a client comes to redeem a token, the provider will know that the client can be linked to one of the five only clients to have tokens signed. A malicious edge could also artificially reduce the size of the user base by only signing tokens for a small number of users or by using artificially small key windows.

A more detailed analysis may also take into account patterns of behavior that lead to initiating signing and redemption phases in predictable ways. For instance, it may be possible to link token redemptions to signing queries that are linked to a user profile that predictably asks for tokens to be signed at certain times.

This threat bypasses the unlinkability of the protocol and exploits the fact that linkability is possible based on the timing of certain requests or via early registration. The pervasiveness of the potential to deanonymize clients this way means that, like many PETs including Tor itself, such privacy solutions are only really effective when the number of users is quite large. As we mentioned previously, 20 days since it was first deployed the number of Chrome users of Privacy Pass stands at 8499, and at 3489 for Firefox (November 28 2017)[8]. This suggests that linking browsing sessions is likely to incur significant overheads and high error rates.

8.5 Double-spending protection

It is vital that services use their own private keys for signing tokens so that tokens cannot be spent multiple times across different services. Moreover, each service that supports Privacy Pass should implement double-spend protection to ensure that tokens cannot be spent multiple times across their own architecture. Not checking double-spends properly essentially implies that the stockpile of unspent passes for any client is much larger. An effective way of managing a large double-spend index would be to use a hash table, Bloom filter or other similarly efficient data structure.

9 Conclusion

In this work we have detailed the use of a VOPRF construction that can be instantiated using elliptic curves to achieve the functionality of a blind signing protocol at much lower cost than a traditional blind-RSA-based construction. We have implemented the protocol and created a browser extension Privacy Pass to offer a generic method of providing easier access for Tor users to web resources by reducing the amount of necessary human participation in completing CAPTCHA challenges. The protocol does not compromise the anonymity of the user by ensuring that the signing and redemption of tokens are cryptographically unlinkable. In addition, support for Privacy Pass within the Cloudflare infrastructure shows that the extension works in global-scale deployments and does not add any significant overheads to either the client or server.

We believe that Privacy Pass represents an innovative and effective way to improve the usability of privacy enhancing technologies on the internet, while maintaining protections against abuse for content providers. We hope that support for Privacy Pass will increase as its benefits, and more use cases, become more apparent.

Acknowledgements

The work benefited from the use of the CrySP RIPPLE Facility at the University of Waterloo. Davidson was supported by the EPSRC and the UK Government as part of the Centre for Doctoral Training in Cyber Security at Royal Holloway, University of London (EP/K035584/1). Goldberg gratefully acknowledges the support of NSERC for grant RGPIN-03858. We would like to thank Eric Tsai, Dan Boneh, Zi Lin, Peter Wu, and Blake Loring for their participation in various stages of the project. We would also like to acknowledge helpful feedback from Sharon Goldberg, Christopher Wood, Peter Eckersley, Brian Warner, Zaki Manian, Tony Arcieri, Prateek Mittal, Zhuotao Liu, Isis Lovecruft, Henry de Valence, Mike Perry, Trevor Perrin, Justin Paine, Marek Majkowski, Eoin Brady, Aaran McGuire, Yan Zhu, Roger Dingledine, Georg Koppen, and the anonymous reviewers.

References

[1] Dan Boneh, Ben Lynn, and Hovav Shacham. Short signatures from the Weil pairing. In Colin Boyd, editor, *ASIACRYPT 2001*, volume 2248 of *LNCS*, pages 514–532. Springer, Heidelberg, December 2001.

[2] Jonathan Burns, Daniel Moore, Katrina Ray, Ryan Speers, and Brian Vohaska. EC-OPRF: Oblivious pseudorandom functions using elliptic curves. Cryptology ePrint Archive, Report 2017/111, 2017. http://eprint.iacr.org/2017/111.

[3] David Chaum. Blind signatures for untraceable payments. In David Chaum, Ronald L. Rivest, and Alan T. Sherman, editors, *CRYPTO'82*, pages 199–203. Plenum Press, New York, USA, 1982.

[4] David Chaum. Blind signature system. In David Chaum, editor, *CRYPTO'83*, page 153. Plenum Press, New York, USA, 1983.

[5] David Chaum, Amos Fiat, and Moni Naor. Untraceable electronic cash. In Shafi Goldwasser, editor, *CRYPTO'88*, volume 403 of *LNCS*, pages 319–327. Springer, Heidelberg, August 1990.

[6] David Chaum and Torben P. Pedersen. Wallet databases with observers. In Ernest F. Brickell, editor, *CRYPTO'92*, volume 740 of *LNCS*, pages 89–105. Springer, Heidelberg, August 1993.

[7] Cisco. The zettabyte era: Trends and analysis, 2017. https://www.cisco.com/c/en/us/solutions/collateral/service-provider/visual-networking-index-vni/vni-hyperconnectivity-wp.html. Accessed Sep 2017.

[8] Cloudflare. Personal communication, 2017.

[9] Florian Dold and Christian Grothoff. GNU Taler: Ethical online payments for the internet age. *ERCIM News*, 2016(106), 2016.

[10] Georg Fuchsbauer, Christian Hanser, Chethan Kamath, and Daniel Slamanig. Practical round-optimal blind signatures in the standard model from weaker assumptions. Cryptology ePrint Archive, Report 2016/662, 2016. http://eprint.iacr.org/2016/662.

[11] Sanjam Garg and Divya Gupta. Efficient round optimal blind signatures. In Phong Q. Nguyen and Elisabeth Oswald, editors, *EUROCRYPT 2014*, volume 8441 of *LNCS*, pages 477–495. Springer, Heidelberg, May 2014.

[12] Sanjam Garg, Vanishree Rao, Amit Sahai, Dominique Schröder, and Dominique Unruh. Round optimal blind signatures. In Phillip Rogaway, editor, *CRYPTO 2011*, volume 6841 of *LNCS*, pages 630–648. Springer, Heidelberg, August 2011.

[13] Ryan Henry and Ian Goldberg. Extending Nymble-like systems. In *2011 IEEE Symposium on Security and Privacy*, pages 523–537. IEEE Computer Society Press, May 2011.

[14] Ryan Henry and Ian Goldberg. Formalizing anonymous blacklisting systems. In *2011 IEEE Symposium on Security and Privacy*, pages 81–95. IEEE Computer Society Press, May 2011.

[15] Ryan Henry and Ian Goldberg. Batch proofs of partial knowledge. In Michael J. Jacobson Jr., Michael E. Locasto, Payman Mohassel, and Reihaneh Safavi-Naini, editors, *ACNS 13*, volume 7954 of *LNCS*, pages 502–517. Springer, Heidelberg, June 2013.

[16] Henry, Ryan. *Efficient Zero-Knowledge Proofs and Applications*. PhD thesis, University of Waterloo, 2014. http://hdl.handle.net/10012/8621.

[17] Thomas S. Heydt-Benjamin, Hee-Jin Chae, Benessa Defend, and Kevin Fu. Privacy for public transportation. In George Danezis and Philippe Golle, editors, *Privacy Enhancing Technologies: 6th International Workshop (PET 2006)*, pages 1–19. Springer, 2006.

[18] Stanislaw Jarecki, Aggelos Kiayias, and Hugo Krawczyk. Round-optimal password-protected secret sharing and T-PAKE in the password-only model. In Palash Sarkar and Tetsu Iwata, editors, *ASIACRYPT 2014, Part II*, volume 8874 of *LNCS*, pages 233–253. Springer, Heidelberg, December 2014.

[19] Stanislaw Jarecki, Aggelos Kiayias, Hugo Krawczyk, and Jiayu Xu. Highly-efficient and composable password-protected secret sharing (or: How to protect your bitcoin wallet online). In *EuroS&P*, pages 276–291. IEEE, 2016.

[20] Florian Kerschbaum, Hoon Wei Lim, and Ivan Gudymenko. Privacy-preserving billing for e-ticketing systems in public transportation. In *Proceedings of the 12th ACM Workshop on Workshop on Privacy in the Electronic Society*, WPES '13, pages 143–154, New York, NY, USA, 2013. ACM.

[21] Zhuotao Liu, Yushan Liu, Philipp Winter, Prateek Mittal, and Yih-Chun Hu. Torpolice: Towards enforcing service-defined access policies for anonymous communication in the tor network. In *25th IEEE International Conference on Network Protocols, ICNP 2017*, pages 1–10, 2017.

[22] Peter Lofgren and Nicholas Hopper. BNymble: More anonymous blacklisting at almost no cost (a short paper). In George Danezis, editor, *FC 2011*, volume 7035 of *LNCS*, pages 268–275. Springer, Heidelberg, February / March 2012.

[23] Dimitrios Papadopoulos, Duane Wessels, Shumon Huque, Moni Naor, Jan Včelák, Leonid Reyzin, and Sharon Goldberg. Making NSEC5 practical for DNSSEC. Cryptology ePrint Archive, Report 2017/099, 2017. http://eprint.iacr.org/2017/099.

[24] David Pointcheval and Jacques Stern. Provably secure blind signature schemes. In Kwangjo Kim and Tsutomu Matsumoto, editors, *ASIACRYPT'96*, volume 1163 of *LNCS*, pages 252–265. Springer, Heidelberg, November 1996.

[25] Markus Rückert. Lattice-based blind signatures. In Masayuki Abe, editor, *ASIACRYPT 2010*, volume 6477 of *LNCS*, pages 413–430. Springer, Heidelberg, December 2010.

[26] Ahmad-Reza Sadeghi, Ivan Visconti, and Christian Wachsmann. User privacy in transport systems based on RFID e-tickets. *Proceedings of the 1st International Workshop on Privacy in Location-Based Applications (PilBA)*, 2008.

[27] Claus-Peter Schnorr. Efficient identification and signatures for smart cards. In Gilles Brassard, editor, *CRYPTO'89*, volume 435 of *LNCS*, pages 239–252. Springer, Heidelberg, August 1990.

[28] Claus-Peter Schnorr and Markus Jakobsson. Security of signed ElGamal encryption. In Tatsuaki Okamoto, editor, *ASIACRYPT 2000*, volume 1976 of *LNCS*, pages 73–89. Springer, Heidelberg, December 2000.

[29] Tor. List of services blocking Tor, 2017. https://trac.torproject.org/projects/tor/wiki/org/doc/ListOfServicesBlockingTor. Accessed Sep 2017.