Privacy Pass: Bypassing Internet Challenges Anonymously

(COM-506 Report)

Majdouline Ait Yahia, Lúcás C. Meier

2022-04-04

1 Introduction

To mitigate spam, websites will often make users solve puzzles like CAPTCHAs, which are difficult for bots to solve, or rate-limit access in other ways. To avoid constantly hassling users with these puzzles, websites will also use fingerprinting and other forms of tracking to recognize users. To avoid tracking, some users make use of tools like Tor [DMS04], anonymizing their communication with the website. Unfortunately, this means that users will have to solve a lot of puzzles, since the server doesn't recognize them, or might confuse them with other bad actors.

This paper [DGS+18] addresses this issue through the use of *tokens*, which can be redeemed in order to access the website. The server can issue multiple tokens for each puzzle, and thus the user has to solve fewer puzzles. To prevent tracking users, the tokens are made to be *unlinkable*: the tokens are independent from each-other, and the user redeeming a token can't be linked with the user who was issued that token. Naturally, we also want to make sure that a token can only be used once, and that users can't create new tokens without the help of the server.

2 Implementation

2.1 Core Idea

The core idea of the scheme is that if you have a Pseudo-Random Function (PRF)

$$F : K \times X \to Y$$

Then evaluating $F_k(x)$ should be difficult without knowing the key k. Thus, we can issue a token by choosing a random t, calculating $F_k(t)$, and then sending that to the user. The user will then present (x, y) to us later, and we check that $F_k(x) = y$. It should be difficult for them to forge such a pair without knowing k.

One problem is that the issuer decides what the token is, so they can trivially link the issuance and the usage later on, allowing them to track users.

To get around this, we need to design a protocol for the user and the issuer to work together, in order for the user to learn $F_k(x)$, without learning k, and without the issuer learning x or $F_k(x)$ either. This idea is known as an *Oblivious PRF* (OPRF).

2.2 More Concretely

Now, let's go over their implementation of an OPRF.

A long time ago, in a galaxy far far away, there was a cyclic group $\mathbb{G}$ of prime order q, and with generator $G \in \mathbb{G}$, suitable for use in Cryptography.

2.2.1 A first go

The PRF we use is defined as:

$$F : \mathbb{F}_q \times \mathbb{G} \to \mathbb{G}$$
$$F(k, P) := k \cdot P$$

In English, the issuer take a point P, and multiplies it by its secret scalar k.

The protocol for turning this into an OPRF uses the same principle behind a Diffie-Hellman key-exchange: multiplying by scalars commutes:

1. The user generates $T \xleftarrow{R} \mathbb{G}$, $r \xleftarrow{R} \mathbb{F}_q$, sets $P \leftarrow r \cdot T$, and sends P to the issuer.
2. The issuer computes $Q \leftarrow k \cdot P$, and sends Q to the user.
3. The user computes $W \leftarrow \frac{1}{r} \cdot Q$. The pair (T, W) is their token.

Note that $Q = k \cdot P = kr \cdot T$, so $\frac{1}{r} \cdot Q = k \cdot T$, so our protocol is at least correct.

2.2.2 Malleability

One little issue with the scheme as described is that using a pair $(T, W = k \cdot T)$, a user can calculate another valid pair, as $(\alpha \cdot T, \alpha \cdot W)$. In other words, the tokens are malleable.

To get rid of this malleability, we use the oldest trick in the book: a hash function. We define a hash function $H_1 : \{0,1\}^\lambda \to \mathbb{G}$. Instead of picking a random point T, we pick a random *seed* $t \in \{0,1\}^\lambda$, defining $T := H_1(t)$. Our token then becomes the pair (t, W), which isn't malleable.

2.2.3 Using diferent values for k

Another issue with the scheme so far is that nothing stops the issuer from using different values of k for different users. This would allow them to link tokens together, because they would be able to tell which k was used, by virtue of the token being valid for that k.

To avoid this, we can require the issuer to commit to a public key $Y = k \cdot G$ in advance, via some kind of Public-Key-Infrastructure (PKI).

During step 2, the issuer then proves that they used the same k, in zero-knowledge:

$$\Pi(Y, Q, P; k) := Y = k \cdot G \wedge Q = k \cdot P$$

The paper cites a protocol for this ZK proof by Chaum and Pedersen [CP92]. Maurer generalized this type of proof to arbitrary one-way group homomorphisms, in [Mau09]. In this case, the homomorphism is $\varphi(x) := (x \cdot G, x \cdot P)$, and we prove that $\varphi(k) = (Y, Q)$ in zero-knowledge. This is the same type of proof as in the ubiquitous Schnorr signature [Sch90].

2.2.4 Batching

It would be convenient to issue multiple tokens at the same time, which would allow us to reduce the number of puzzles users have to solve. We can do this straightforwardly by having the user generate multiple seeds t_i, send multiple points P_i, and receive multiple points Q_i in return. The zero-knowledge proof

that the same k was used is still necessary, and can also be repeated multiple times. These proofs can also be combined into a more efficient batch proof, as described in [HG13], and explained in the paper. This reduces communication cost.

2.3 Redemption

We've described how to issue tokens, but not how to use those tokens to request content. The idea is that given a token (t, W), hashing this token with a hash function H_2 allows us to derive a shared key $K := H_2(t, W)$ with the issuer. The issuer can derive this key from just t, since W satisfies $W = k \cdot H_1(t)$, and the issuer knows k.

The user can then redeem their token (t, W) to access some content R by sending $(t, \text{MAC}_K(R))$ to the issuer, who can then recalculate K from t, and check that $\text{MAC}_K(R')$ is the same, using the content R' they're responding with. Binding the request to the content R in a way the server can recompute avoids potential issues with someone intercepting a token redemption protocol, and using that to access content. The issuer must also keep a list of tokens t it has seen, in order to prevent users from reusing tokens.

3 Security Notions

There are two sides to consider when it comes to the security of this scheme: the issuer, and the user.

The issuer doesn't want users to be able to generate new tokens without its approval: the tokens should be *unforgeable*. This is denoted as "One-More-Token" security. This notion can be modeled as a game, where an adversary runs the protocol, receiving N tokens, and then attempts to forge an $(N + 1)$th token. Naturally, an adversary should only succeed at this with negligeable probability.

The user, on the other hand, doesn't want the issuer to link their tokens together. The essence here is that what the issuer sees in the redemption phase is independent from what they've seen in the issuance phase. This can also be modeled as a game, in which the user runs two signing phases with the adversary, and then redeems the tokens from one of the two signing phases, with the adversary then guessing which of the phases the tokens came from. An adversary should only guess correctly with negligeable probability.

4 Potential Problems

4.1 Monkey in the middle

A lazy adversary could intercept communication between a user and the issuer, in order to avoid solving puzzles themselves. They would receive a puzzle from the issuer, but then forward that puzzle to the user, pretending to be the adversary. Using the user's response, they would receive the tokens, instead of the user.

4.2 Accumulating Tokens

Since the tokens are unlinkable, and users are presumably anonymous to the issuer, a user can solve many puzzles, accumulating a lot of tokens. They can then redeem these tokens, or even share them with many bots, and conduct a denial-of-service attack on whatever service uses the tokens to restrict spam.

One of the mitigations suggested in the paper is to rotate the k value used in the OPRF, and thus the public key, which invalidates all tokens created with that k. Rotating keys often enough prevents tokens from being accumulated, to a certain extent.

4.3 Weakness in small numbers

Tokens can't be linked with each-other, which makes it difficult to distinguish users making use of the token protocol from eachother. Unfortunately, if few users are using the token protocol at all, there might be enough metadata to link them again. Malicious issuers might even aggravate the situation by intentionally only issuing a small number of tokens in total, or rotating the k value excessively often.

4.4 Key Rotation

The paper briefly mentions rotating the k value. This changes the public key Y. Because of this, users need to be aware of what public key(s) the issuer currently accepts. The paper briefly mentions having the issuer send the user a list of public keys they accept: this is a bit silly, because the issuer can then send each user a different list of public keys, making the tokens linkable. What the authors most likely meant was having the issuer publicly post their keys via some kind of PKI, which would ensure all users would be seeing the same result.

References

[CP92] David Chaum and Torben Pryds Pedersen'. Wallet Databases with Observers. *CRYPTO '92*, 1992.

[DGS+18] Alex Davidson, Ian Goldberg, Nick Sullivan, George Tankersley, and Filippo Valsorda. Privacy Pass: Bypassing Internet Challenges Anonymously. *Proceedings on Privacy Enhancing Technologies*, 2018, 2018.

[DMS04] Roger Dingledine, Nick Mathewson, and Paul Syverson. Tor: The Second-Generation Onion Router:. Technical report, 2004.

[HG13] Ryan Henry and Ian Goldberg. Batch Proofs of Partial Knowledge. In *ACNS*, volume 7954, pages 502–517. 2013.

[Mau09] Ueli Maurer. Unifying Zero-Knowledge Proofs of Knowledge. *Progress in Cryptology - AFRICACRYPT 2009*, 2009.

[Sch90] C. P. Schnorr. Efficient Identification and Signatures for Smart Cards. In *CRYPTO' 89*, 1990.