Token Based Services - Differences from Privacy Pass

Erinn Atwater
erinn@openprivacy.ca

Sarah Jamie Lewis
sarah@openprivacy.ca

DRAFT – Subject to further change and refinement.

1 Context: Prepaid Anonymous Services

We outline an approach to allow anonymous peers to (pre-)purchase services (e.g. message hosting) from anonymous servers based on Privacy Pass [1].

Service providers offer the purchase of "tokens" that can be later redeemed for services in a similar way to how prepaid cards work.

Separating the act of purchasing tokens from using them provides several advantages over purchasing the services directly, even if we assume the purchasing method is completely metadata resistant. This separation allows us to amortize the latency and transaction fees associated with cryptocurrency transactions over several tokens, reducing friction after the initial investment. Also, because tokens are unlinkable and fungible, peers can share tokens across devices or even between group members, opening up the services to a wider audience while preserving the privacy of individual users.

When considering payment channels that do not offer strong privacy protections, separating acquisition and redemption provides another tool for enabling peers to manage risks to their anonymity.

This paper outlines the differences of our protocol from the original Privacy Pass paper [1] and – except where specified – uses the same notation as that paper. It is prerequisite reading for understanding what follows.

2 Differences to Privacy Pass

The major difference in approach from Privacy Pass is in our trigger for signing tokens. Instead of a challenge, we expect peers to purchase tokens directly, executing the signing protocol over the purchasing channel (e.g. using Zcash encrypted memos [4]. The redemption phase can then take place at an unspecified time in the future, free of purchasing metadata that might be used to track usage.

Unlike the original Privacy Pass we require peers/servers to publish $W = t^k$ when redeeming a token in addition to $(t, \mathsf{MAC}_K(R))$. Both the peer and the server know W, as it is necessary for calculating the shared secret K. Once published, there is no longer any need to keep W a secret, and publishing it openly does not harm the privacy of other tokens.

We use W as an additional constraint on the servers ability to sign tokens using different public keys (see section 5.5 Key Consistency in the Privacy Pass paper for details on this attack vector).

2.1 Constraining Token Signing

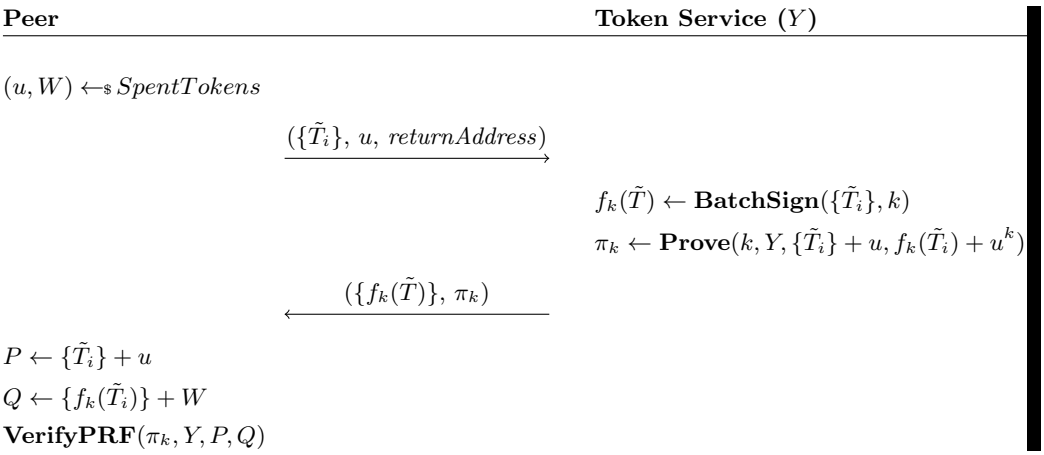

Figure 1: Token Payment and Signing: The client sends a set of blinded tokens $\{\tilde{T}_i\}$ over an encrypted payment channel (e.g. Zcash) along with a return address and a previously publicly redeemed token u. The request must be accompanied by an adequate payment (accounting for both the cost of the tokens themselves and the cost of sending a return message). Once payment is confirmed the server signs the tokens, along with the constraint token u, and sends them back to the client, along with a proof π_k demonstrating that the tokens were signed correctly. Once the client has received the result, π_k is checked after which they can be unblinded and used to redeem services.

Previously published tokens[1] are used to constrain the signing of newer tokens, by forcing the token provider to include them within the discrete-log equivalence proof. We incorporate the token into the **BatchDLEQ** by treating it as an additional blinded token/signed token pair where: $P = T$ and $Q = W = T^k$ (see Figure 1). The token is selected randomly[2] from the set of *SpentTokens* published publicly through the act of redeeming them. As such any valid token for a service, regardless of the originator, can be used to constrain the batch proof signing newer tokens and both sets of tokens are tied to the token servers public key ($Y = X^k$).

[1] e.g. In one of the applications we are considering, the server maintains a publicly auditable log of messages, including the token used to pay for those services. We expect peers to select from this set of tokens.

[2] For now; in the future, we envision including spent tokens with server invitations in order to create a gossip-esque mechanism to help prevent key partitioning attacks by the server.

2.1.1 Considerations for Key Rotation

Constraining signing proofs to previously used tokens obviously has an impact on the ability to rotate keys. This can be handled much as described in the original Privacy Pass paper, by having the server commit to a rolling set of public keys, keeping the set size relatively low e.g. 2 or 3 at any one time. This allows clients to use keys from a larger range in time, without restricting server key rotation.

2.2 Request Binding

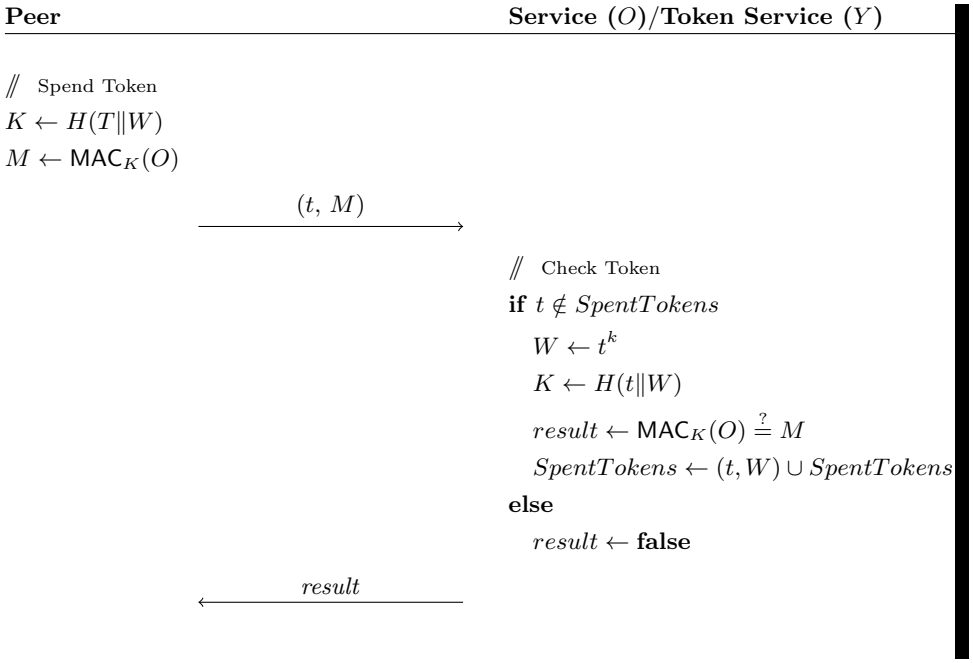

Figure 2: Token Redemption: We require the server to maintain a public set *SpentTokens* of previously used tokens. The redemption requests are bound to the public key of the service O. On receipt of a redemption request the server checks that the token t has not previously been spent, and it has not it checks the MACagainst the derived key and the public key of the service. If all the checks succeed, t is added to the set of *SpentTokens*.

In contrast to the Privacy Pass paper wherein redemptions are bound to a particular requested resource, we bind requests to the public key[3] of the service being paid for (see Figure 2).

This explicit binding to the service prevents a rogue token server from acting as middleperson for an unrelated service (e.g. by forwarding on all signing/redemption requests and responses while charging a premium) by having services reject any token redemption that isn't bound to their public identifier.

[3]For clarity, this is not the public key Y used for token signing, but an identity key used to identify the service over some anonymous communication network e.g. a Tor v3 onion address

3 Parameterization

For our implementation[4] we use Ristretto255 [3] as the cyclic group of prime order. In addition, we use Merlin transcripts [2] to generate the challenges used in the protocol rather than a standalone hash of the public inputs directly.

References

[1] Alex Davidson, Ian Goldberg, Nick Sullivan, George Tankersley, and Filippo Valsorda. Privacy pass: Bypassing internet challenges anonymously. *Proceedings on Privacy Enhancing Technologies*, 2018(3):164–180, 2018.

[2] Henry de Valence. Merlin transcripts. *https://merlin.cool/*, 2018.

[3] Henry de Valence, Jack Grigg, George Tankersley, Filo Valsorda, and Isis Lovecruft. The ristretto255 group. Technical report, IETF CFRG Internet Draft, 2019.

[4] Daira Hopwood, Sean Bowe, Taylor Hornby, and Nathan Wilcox. Zcash protocol specification. *Tech. rep. 2016–1.10. Zerocoin Electric Coin Company, Tech. Rep.*, 2016. Section 3.2.1.

[4] https://git.openprivacy.ca/openprivacy/zcashtokenservice