Privacy Pass with Token Binding Extension

draft-guo-privacypass-token-binding-00

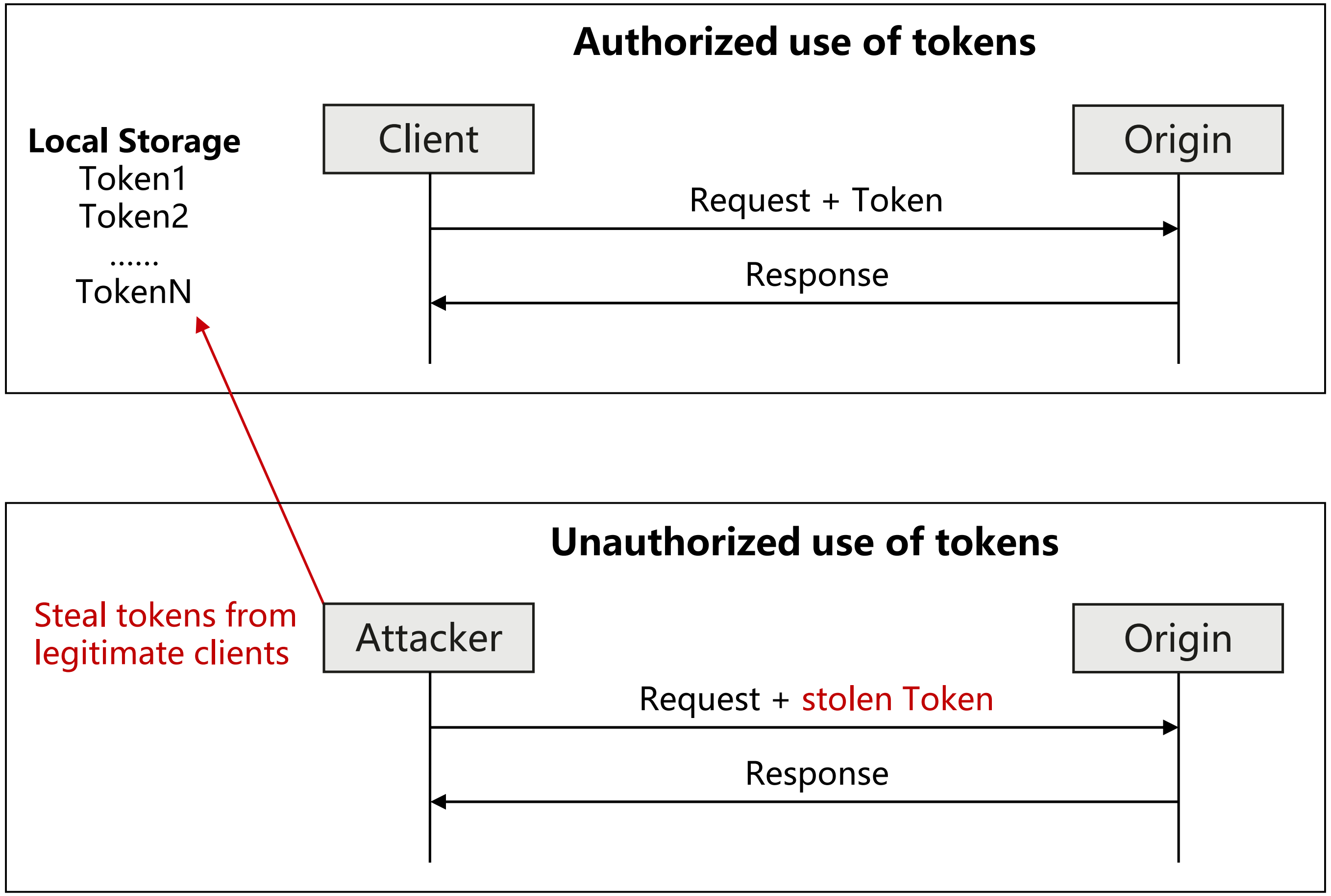
https://datatracker.ietf.org/doc/draft-guo-privacypass-token-binding/00/

Wei Guo, Yong Li, Ji Li and Liang Xia

IETF 122, March 2025, Bangkok

Introduction

- Privacy Pass tokens pertain to **bearer tokens**, and any party in possession of Privacy Pass tokens can also access to the protected resources.
- Therefore, attackers can take advantage of this by **exporting Privacy Pass tokens** from a user's machine (the Client), presenting them to the application server (the Origin), and impersonating the legitimate user (see below).



Privacy Pass with Token Binding Extension

- As shown below, there are some existing works about protecting OAuth tokens from theft attacks, where the tokens are bound to the client's long-term public key.
  - OAuth 2.0 **Token Binding** [RFC8471].
  - OAuth 2.0 **M-TLS** [RFC8705].
  - OAuth 2.0 **DPoP** [RFC9449].

- However, the above solutions **are not applicable to** the Privacy Pass tokens. The reason is that the client's long-term public key can be used to link different tokens, thus **breaking the unlinkability** of Privacy Pass tokens.

- The **token binding** extension for the Privacy Pass protocol is to prevent token theft attacks by cryptographically binding a Privacy Pass token to the Client generated **one-time public key** and requiring the Client presenting the token to additionally **prove possession of** the corresponding one-time private key.

Privacy Pass with Token Binding Extension

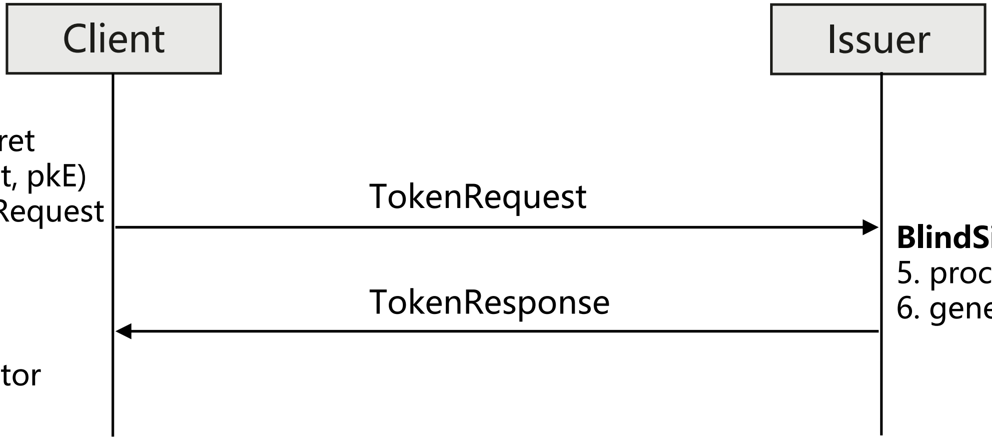
- The token binding extension **does not introduce** additional cryptographic primitives. Specifically, we choose the **Schnorr NIZKP**, that is currently utilized the Privacy Pass protocol, to achieve token binding for both privately verifiable tokens and publicly verifiable tokens because of the following reasons.
    - The keypair of the Schnorr NIZKP can be **generated from a seed**, which is not supported by the RSA signature.
    - The keypair generation of the Schnorr NIZKP is **much faster than** that of the RSA signature.

- Optionally, the Schnorr NIZK proof generation can support the property of **channel binding**, which guarantees that the token and the binding proof for one channel that obtained by an attacker cannot be used in another channel. **Three variants** for channel binding are defined.
    - **No** channel binding.
    - **TLS** [RFC8466] channel binding.
    - **HPKE** [RFC9180] channel binding.

Privacy Pass with Token Binding Extension

Issuance Protocol with Token Binding

Blind function:
1. creates token_input as normal
2. generates a keypair (skE, pkE) and keeps skE secret
3. creates bound_token_input = concat(token_input, pkE)
4. blinds bound_token_input and generates TokenRequest

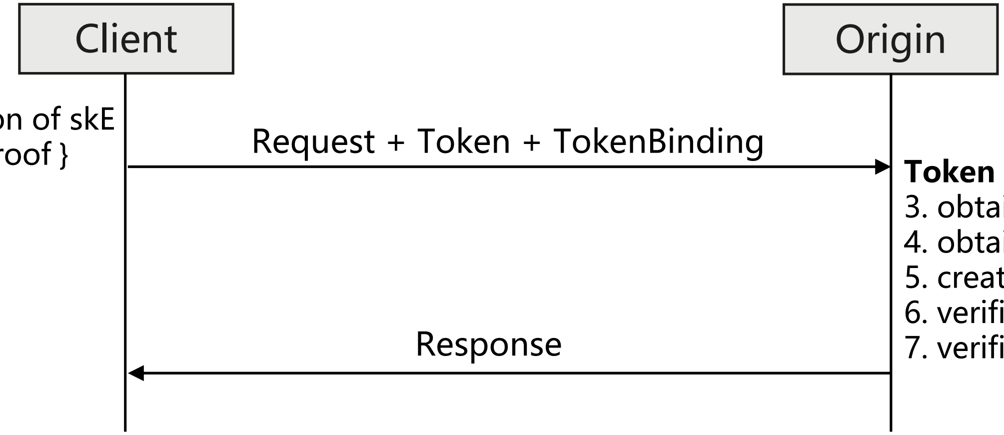


BlindSign function:
5. processes TokenRequest using the Issuer private key
6. generates TokenResponse

Finalize function:
7. unblinds TokenResponse to generate authenticator
8. generates Token, stores Token and pkE

Redemption Protocol with Token Binding

TokenBinding generation:
1. generates binding_proof to prove possession of skE
2. generates TokenBinding = { pkE, binding_proof }



Token and TokenBinding verification:
3. obtains token_input and authenticator from Token as normal
4. obtains pkE and binding_proof from TokenBinding
5. creates bound_token_input = concat(token_input, pkE)
6. verifies authenticator against bound_token_input
7. verifies binding_proof using pkE

Thank You

Any Questions