Anonymous Credit Tokens

Privacy Pass Integration

Samuel Schlesinger

Updates

- Published draft-schlesinger-privacypass-act
- Partial progress in porting draft-schlesinger-cfrg-act to use Sigma Protocol + Fiat-Shamir drafts
- Chrome and Firefox are exploring how technology like this could be used on the web
- Cloudflare and Google are exploring deployment to rate limit AI agents

Motivation

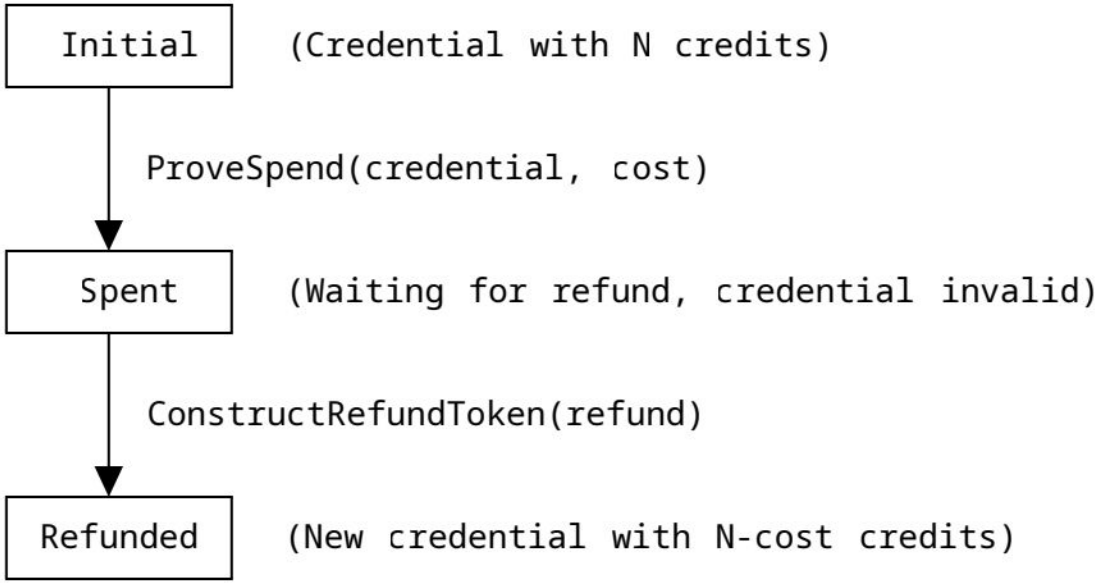
Why not ARC, VOPRF, or Blind RSA?

Boils down to three properties:

- Concurrency control: can only be used by one request simultaneously
- Dynamic revocation: can be revoked by the issuer by refusing to refund the token
- Per-Session Rate Limiting: distinct rate limits to different users (or bots) without impacting privacy

5.2. Client State Transitions

A client managing an ACT credential progresses through the following states:



State transitions:

* **Initial**: Client holds a valid credential with N credits (N > 0). The credential can be spent.

* **Spent**: Client has generated a spend proof and sent it to the origin. The original credential is now invalid and **MUST NOT** be used again. The client is blocked waiting for a refund response.

* **Refunded**: Client has received a valid refund and constructed a new credential with the remaining balance. If the new balance is greater than 0, the client returns to the Initial state with the new credential. If the balance is 0, the credential is exhausted.

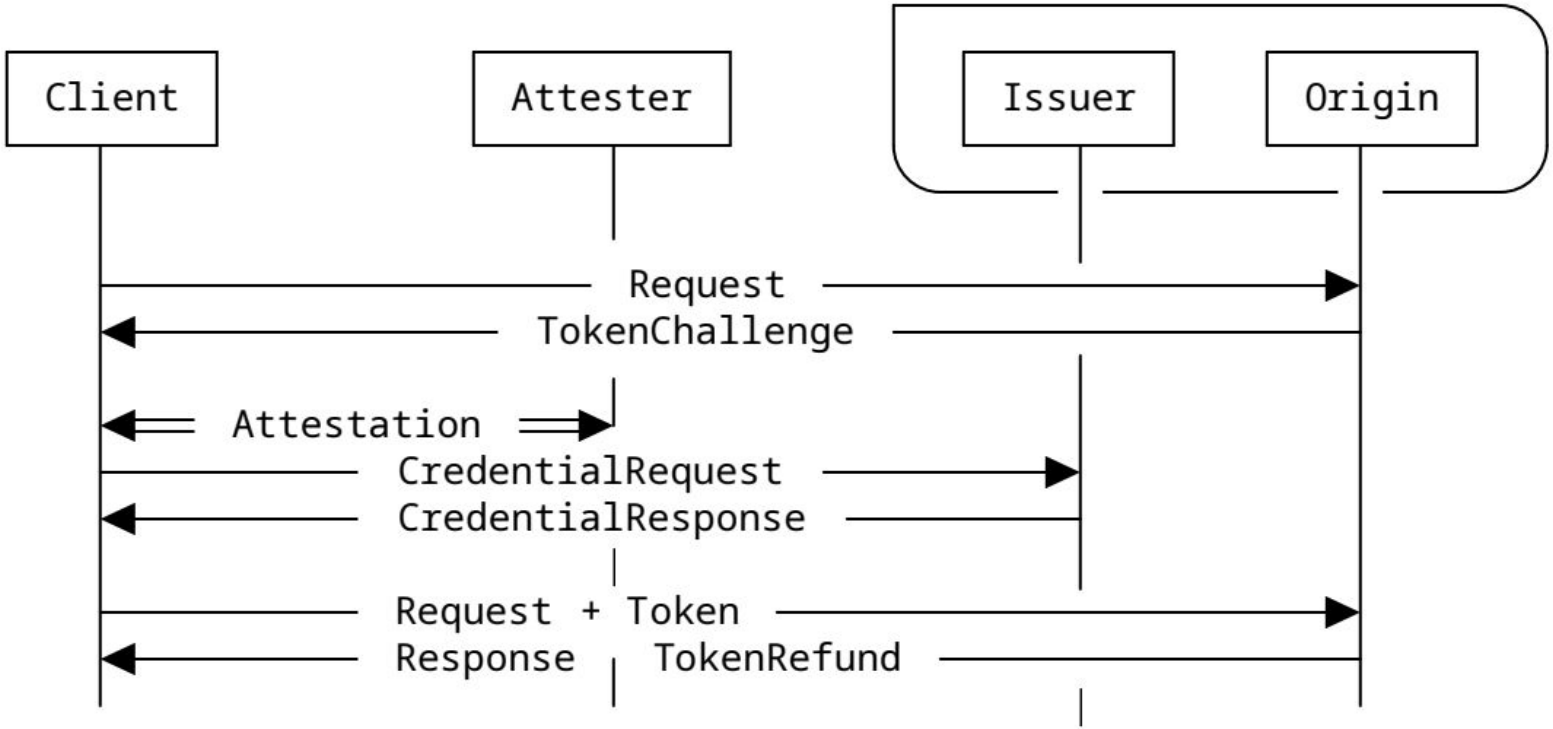


Figure 1: Issuance and Redemption Overview

```
(clientSecrets, request) = IssueRequest()
```

The Client then creates a TokenRequest structure as follows:

```
struct {
  uint16_t token_type = 0xE5AD; /* Type ACT(Ristretto255) */
  uint8_t truncated_issuer_key_id;
  uint8_t encoded_request[Nrequest];
} TokenRequest;
```

```
request_context = concat(tokenChallenge.issuer_name,
  tokenChallenge.origin_info,
  tokenChallenge.credential_context,
  issuer_key_id)
response = IssueResponse(skI, request, initial_credits, request_context)
```

The Issuer then creates a TokenResponse structured as follows:

```
struct {
   uint8_t encoded_response[Nresponse];
} TokenResponse;
```

Note: request_context should be set in the IssueRequest, not here, for privacy reasons the client needs to know request_context anyways. Pleasantly, this removes the need for a new field in the credential.

Given a TokenChallenge value as input, denoted challenge, a cost, denoted cost, and a previously obtained credential that is valid for the Issuer identifier in the challenge, denoted credential, containing at least cost credits, Clients compute a spend request as follows:

```
spend_proof, state = ProveSpend(credential, cost)
```

Each credential instance **MUST** only ever be used for a single spend request. When the client receives the refunded credential from the server, the client uses that new credential instance for the next spend. If the same credential instance is used more than once, the privacy assumptions of ACT are violated by presenting the same nullifier twice.

The resulting Token value is then constructed as follows:

```
struct {
    uint16_t token_type = 0xE5AD; /* Type ACT(Ristretto255) */
    uint8_t challenge_digest[32];
    uint8_t issuer_key_id[Nid];
    uint8_t encoded_spend_proof[Nspend_proof];
} Token;
```

```
request_context = concat(tokenChallenge.issuer_name,
  tokenChallenge.origin_info,
  tokenChallenge.credential_context,
  issuer_key_id)
refund = VerifyAndRefund(skI, request_context, spend_proof)
```

Next Steps

- Porting the remaining bespoke Sigma Protocols and Fiat-Shamir implementation to the emerging CFRG standards
- Adding request context, needed for compatibility with the Privacy Pass integration
- Other issues in draft-act/issues

Questions