Batched Token Issuance Protocol

`draft-robert-privacypass-batched-tokens`

IETF116, Yokohama
Raphael Robert

Motivation

Privately Verifiable Tokens can be expensive when issued in high numbers.

The primitive choice is conservative (P384) and the protocol doesn’t make use of efficient DLEQ ZK proofs

How?

1. Issuing multiple tokens at once in response to a single TokenChallenge, thereby reducing the size of the proofs required for multiple tokens.
2. Improving server and client issuance efficiency by amortizing the cost of the VOPRF proof generation and verification, respectively.

Token request

```
struct {
  uint16_t token_type = 0x0001; // Privately Verifiable Token
  uint8_t truncated_token_key_id;
  uint8_t blinded_msg[Ne];
} TokenRequest;
```

```
struct {
   uint16_t token_type = 0xF91A; // Batched Token
   uint8_t token_key_id;
   BlindedElement blinded_elements<0..2^16-1>;
} TokenRequest;

struct {
      uint8_t blinded_element[Ne];
} BlindedElement;
```

Token response

```
struct {
   uint8_t evaluate_msg[Ne];
   uint8_t evaluate_proof[Ns+Ns];
} TokenResponse;
```

```
struct {
   EvaluatedElement evaluated_elements<0..2^16-1>;
   uint8_t evaluated_proof[Ns + Ns];
} TokenResponse;

struct {
      uint8_t evaluated_element[Ne];
} EvaluatedElement;
```

Security considerations

“A side-effect of the OPRF protocol variants in this document is that they allow instantiation of an oracle for constructing static DH samples; see [BG04] and [Cheon06]. These attacks are meant to recover (bits of) the server private key. Best-known attacks reduce the security of the prime-order group instantiation by log_2(Q)/2 bits, where Q is the number of BlindEvaluate calls made by the attacker.”

Mitigation strategies:

- Limit issuance (rate-limit BlindEvaluate)
- Rotate keys regularly
- Define token type with larger group

Performance chart (ristretto255)

	Publicly Verifiable	Privately Verifiable	Batched (100)
Server: Generate key pair	122 960 µs	475 µs	37 µs
Client: Issue token request	264 µs	685 µs	52 µs
Server: Issue token response	1 349 µs	2 568 µs	79 µs
Client: Issue token	152 µs	3 480 µs	125 µs
Server: Redeem token	147 µs	725 µs	50 µs

Measured on an M1 using RustCrypto

Implementations

Currently two implementations with interop test vectors exist:

- `privacypass` in Rust
- `pat-go` in Go