A private mode for OpenID Connect

Sven Hammann    Ralf Sasse    David Basin

OAuth Security Workshop, July 2017
OAuth 2.0: Authorizing third-party applications (relying parties) to access resources
Brief Introduction to OAuth 2.0 and OpenID Connect

- OAuth 2.0: Authorizing third-party applications (relying parties) to access resources

- Authorization Server hands out access token to the relying party (RP)
Brief Introduction to OAuth 2.0 and OpenID Connect

- OAuth 2.0: Authorizing third-party applications (relying parties) to access resources

- Authorization Server hands out access token to the relying party (RP)
  - RP must be registered at the authorization server
  - Used to access the resources at the resource provider
  - Authorization server and resource provider may be the same
Brief Introduction to OAuth 2.0 and OpenID Connect

- OpenID Connect builds an authentication layer on top (single sign-on)
Brief Introduction to OAuth 2.0 and OpenID Connect

- OpenID Connect builds an authentication layer on top (single sign-on)

- Authorization server is now also an identity provider (IdP)
Brief Introduction to OAuth 2.0 and OpenID Connect

- OpenID Connect builds an authentication layer on top (single sign-on)

- Authorization server is now also an identity provider (IdP)

- IdP hands out an id token
Brief Introduction to OAuth 2.0 and OpenID Connect

- OpenID Connect builds an authentication layer on top (single sign-on)

- Authorization server is now also an identity provider (IdP)

- IdP hands out an id token
  - Signed Json Web Token (JWT)
  - Asserts the user’s identity at the IdP
  - Contains user info
Brief Introduction to OAuth 2.0 and OpenID Connect

- OpenID Connect builds an authentication layer on top (single sign-on)

- Authorization server is now also an identity provider (IdP)

- IdP hands out an id token
  - Signed Json Web Token (JWT)
  - Asserts the user’s identity at the IdP
  - Contains user info

- Can be combined with standard OAuth 2.0
  - Both *token* (access token) and *id_token* handed out
Example *id_token*:

```json
{
    "iss": "https://server.example.com",
    "sub": "24400320",
    "aud": "s6BhdRkqt3",
    "nonce": "n-0S6_WzA2Mj",
    "exp": 1311281970,
    "iat": 1311280970,
    "auth_time": 1311280969
}
```
Motivation of our work

- The IdP learns at which Relying Parties (RPs) the user logs in

- This does not respect the user's privacy
- User's activities over multiple RPs can be tracked
- User might not want anyone to know which service they are using
- Especially if using the RP might provide sensitive information

- Alcoholics Anonymous
- Medical Forums

Our solution: We propose a new mode that hides the RP’s identity from the IdP
Motivation of our work

- The IdP learns at which Relying Parties (RPs) the user logs in
- This does not respect the user’s privacy
Motivation of our work

- The IdP learns at which Relying Parties (RPs) the user logs in

- This does not respect the user’s privacy
  - User’s activities over multiple RPs can be tracked
  - User might not want anyone to know which service they are using
  - Especially if using the RP might provide sensitive information
Motivation of our work

- The IdP learns at which Relying Parties (RPs) the user logs in

- This does not respect the user’s privacy
  - User’s activities over multiple RPs can be tracked
  - User might not want anyone to know which service they are using
  - Especially if using the RP might provide sensitive information
    - Alcoholics Anonymous
    - Medical Forums
Motivation of our work

- The IdP learns at which Relying Parties (RPs) the user logs in

- This does not respect the user’s privacy
  - User’s activities over multiple RPs can be tracked
  - User might not want anyone to know which service they are using
  - Especially if using the RP might provide sensitive information
    - Alcoholics Anonymous
    - Medical Forums

- Our solution: We propose a new mode that hides the RP’s identity from the IdP
Incentives for participants

- Incentives for RPs to support the mode
  - Provide sensitive service to users: Interested in user’s privacy
  - Protect their own data: Number of accesses

- Incentives for IdPs to support the mode
  - Data minimization (fulfill regulatory requirements)
  - Improve public perception
  - Distinguishing feature to attract privacy-interested users
Incentives for participants

- Incentives for RPs to support the mode
  - Provide sensitive service to users: Interested in user’s privacy
  - Protect their own data: Number of accesses

- Incentives for IdPs to support the mode
  - Data minimization (fulfill regulatory requirements)
  - Improve public perception
  - Distinguishing feature to attract privacy-interested users
Incentives for participants

- Incentives for RPs to support the mode
  - Provide sensitive service to users: Interested in user’s privacy
  - Protect their own data: Number of accesses

- Incentives for IdPs to support the mode
Incentives for participants

- Incentives for RPs to support the mode
  - Provide sensitive service to users: Interested in user’s privacy
  - Protect their own data: Number of accesses

- Incentives for IdPs to support the mode
  - Data minimization (fulfill regulatory requirements)
  - Improve public perception
  - Distinguishing feature to attract privacy-interested users
**msc OAuth 2 implicit flow**

1. **RP (client)**: Access RP functionality
   - forward: client_id, redirect_uri, ...

2. **User Agent**: Forward:
   - client_id, redirect_uri, ...
   - Ask for user consent
   - User gives consent
   - token (in URI fragment)

3. **Authorization Server**: Checks
   - Extract token

4. **Access resource**

---

**Important Points**

- **client_id**, **redirect_uri**, etc. are typically used in the OAuth flow to identify the client and the redirect URL.
- **Ask for user consent** is a crucial step where the user approves the access.
- **Token** extracted from the URI fragment can be used to access resources.
- **Access resource** is the final step in the flow, allowing the client application to access the requested resource.
**OIDC regular implicit mode**

- **RP A**
  - Authentication Request
    - `client_id, redirect_uri, ...`

- **User Agent**
  - Forward:
    - `client_id, redirect_uri, ...`
  - Checks
  - Ask for user consent
  - User gives consent
    - `id_token (in URI fragment)`
  - Extract `id_token`

- **IdP 1**
  - Check `id_token`
  - Login successful or Error message
Privacy and Security Goals

- Privacy towards IdP: IdP cannot distinguish between logins to different RPs

- Security: Equivalent security to the implicit mode
- All checks are still made and provide the same guarantees

- No RP should be able to use an id token to impersonate the user at another RP
Privacy and Security Goals

- Privacy towards IdP: IdP cannot distinguish between logins to different RPs
  - IdP cannot link repeated logins to the same RP
  - IdP only sees that the user logs in to some RP

- Security: Equivalent security to the implicit mode
  - All checks are still made and provide the same guarantees
  - No RP should be able to use an id token to impersonate the user at another RP
Privacy and Security Goals

- Privacy towards IdP: IdP cannot distinguish between logins to different RPs
  - IdP cannot link repeated logins to the same RP
  - IdP only sees that the user logs in to some RP

- Security: Equivalent security to the implicit mode
Privacy and Security Goals

- Privacy towards IdP: IdP cannot distinguish between logins to different RPs
  - IdP cannot link repeated logins to the same RP
  - IdP only sees that the user logs in to some RP

- Security: Equivalent security to the implicit mode
  - All checks are still made and provide the same guarantees
  - No RP should be able to use an id_token to impersonate the user at another RP
Attacker model

- Honest-but-curious IdP
Attacker model

- Honest-but-curious IdP
  - IdP does not collude with the RP
  - Trusted JavaScript on the IdP frontend
Attacker model

- **Honest-but-curious IdP**
  - IdP does not collude with the RP
  - Trusted JavaScript on the IdP frontend

- **Malicious RPs**
Attacker model

- Honest-but-curious IdP
  - IdP does not collude with the RP
  - Trusted JavaScript on the IdP frontend

- Malicious RPs
  - Regarding security properties, not privacy
  - IdP does not collude even with malicious RPs
Attacker model

- **Honest-but-curious IdP**
  - IdP does not collude with the RP
  - Trusted JavaScript on the IdP frontend

- **Malicious RPs**
  - Regarding security properties, not privacy
  - IdP does not collude even with malicious RPs

- Secure end-to-end channels (TLS)
Problems to Solve

1. *id_token* must only be valid for one RP
Problems to Solve

1. *id_token* must only be valid for one RP
   - Otherwise, any RP could forward it to impersonate the user at another RP
Problems to Solve

1. *id_token* must only be valid for one RP
   ▶  Otherwise, any RP could forward it to impersonate the user at another RP
   ▶  How can the IdP create an *id_token* for one audience RP without knowing for which RP?
Problems to Solve

1. *id_token* must only be valid for one RP
   - Otherwise, any RP could forward it to impersonate the user at another RP
   - How can the IdP create an *id_token* for one audience RP without knowing for which RP?

2. User must give consent and *redirect_uri* must be checked
Problems to Solve

1. *id_token* must only be valid for one RP
   ▶ Otherwise, any RP could forward it to impersonate the user at another RP
   ▶ How can the IdP create an *id_token* for one audience RP without knowing for which RP?

2. User must give consent and *redirect_uri* must be checked
   ▶ Requires client (RP) metadata to be looked up by the IdP
Problems to Solve

1. *id_token* must only be valid for one RP
   - Otherwise, any RP could forward it to impersonate the user at another RP
   - How can the IdP create an *id_token* for one audience RP without knowing for which RP?

2. User must give consent and *redirect_uri* must be checked
   - Requires client (RP) metadata to be looked up by the IdP
   - How can this be done if the IdP does not know the RP’s identity?
Solution for the first problem

- Use a hashed pseudonym for the *client_id*
Solution for the first problem

- Use a hashed pseudonym for the $client_id$

- $client_id_hash := h(client_id, rp_nonce, user_nonce)$
Solution for the first problem

- Use a hashed pseudonym for the `client_id`

- `client_id_hash := h(client_id, rp_nonce, user_nonce)`
  - `rp_nonce` is the nonce sent by the RP (also exists in regular mode)
  - `user_nonce` is generated by the user agent
Solution for the first problem

- Use a hashed pseudonym for the \textit{client\_id}

- \textit{client\_id\_hash} := \textit{h(client\_id, rp\_nonce, user\_nonce)}
  - \textit{rp\_nonce} is the nonce sent by the RP (also exists in regular mode)
  - \textit{user\_nonce} is generated by the user agent

- Only the \textit{client\_id\_hash} is sent to the IdP server
Solution for the first problem

- Use a hashed pseudonym for the client_id

- \( \text{client\_id\_hash} := h(\text{client\_id}, \text{rp\_nonce}, \text{user\_nonce}) \)
  - \( \text{rp\_nonce} \) is the nonce sent by the RP (also exists in regular mode)
  - \( \text{user\_nonce} \) is generated by the user agent

- Only the client_id_hash is sent to the IdP server
  - \( \text{client\_id} \) and \( \text{rp\_nonce} \) are sent by the RP in the URI fragment
  - They are not forwarded to the IdP server
  - Hash is computed in IdP JavaScript
Solution for the first problem

- Use a hashed pseudonym for the \textit{client\_id}

- \textit{client\_id\_hash} \( := h(\textit{client\_id}, \textit{rp\_nonce}, \textit{user\_nonce}) \)
  - \textit{rp\_nonce} is the nonce sent by the RP (also exists in regular mode)
  - \textit{user\_nonce} is generated by the user agent

- Only the \textit{client\_id\_hash} is sent to the IdP server
  - \textit{client\_id} and \textit{rp\_nonce} are sent by the RP in the URI fragment
  - They are not forwarded to the IdP server
  - Hash is computed in IdP JavaScript

- IdP hands out a \textit{private\_id\_token}
Solution for the first problem

- Use a hashed pseudonym for the `client_id`

  \[ client\_id\_hash := h(client\_id, \textit{rp}\_nonce, \textit{user}\_nonce) \]

  - \textit{rp}\_nonce is the nonce sent by the RP (also exists in regular mode)
  - \textit{user}\_nonce is generated by the user agent

- Only the `client\_id\_hash` is sent to the IdP server
  - `client\_id` and \textit{rp}\_nonce are sent by the RP in the URI fragment
  - They are not forwarded to the IdP server
  - Hash is computed in IdP JavaScript

- IdP hands out a `private\_id\_token`
  - Contains no \textit{aud} field but a `private\_aud` field containing the `client\_id\_hash`
  - Cannot be confused with a regular `id\_token` since \textit{aud} field is mandatory
Solution for the first problem

Example *private_id_token*:

```json
{
    "iss": "https://server.example.com",
    "sub": "24400320",
    "private_aud": "96f6696e4024d65fcb018a8f71badd313f06e1481f142b29d4ba6f307bfc00e0",
    "exp": 1311281970,
    "iat": 1311280970,
    "auth_time": 1311280969
}
```
Solution for the second problem

- Enable the RP to provide its own client metadata
Solution for the second problem

- Enable the RP to provide its own client metadata

- To ensure its validity, it must be signed by the IdP
Solution for the second problem

- Enable the RP to provide its own client metadata
- To ensure its validity, it must be signed by the IdP
- New parameter sent by the RP: `client_id_binding`
Solution for the second problem

- Enable the RP to provide its own client metadata
- To ensure its validity, it must be signed by the IdP
- New parameter sent by the RP: `client_id_binding`
  - JWT signed by the IdP
  - Given to RP when it registers at the IdP
  - Contains a `client_id` with metadata belonging to that RP
Solution for the second problem

- Enable the RP to provide its own client metadata

- To ensure its validity, it must be signed by the IdP

- New parameter sent by the RP: `client_id_binding`
  - JWT signed by the IdP
  - Given to RP when it registers at the IdP
  - Contains a `client_id` with metadata belonging to that RP

- `client_id_binding` is used by user agent
Solution for the second problem

- Enable the RP to provide its own client metadata
- To ensure its validity, it must be signed by the IdP

- New parameter sent by the RP: `client_id_binding`
  - JWT signed by the IdP
  - Given to RP when it registers at the IdP
  - Contains a `client_id` with metadata belonging to that RP

- `client_id_binding` is used by user agent
  - Sent in URI fragment
  - Checks done by IdP JavaScript can access `client_id_binding`
  - No need to look up metadata on the IdP server
Example `client_id_binding`:

```
{
   "client_id": "s6BhdRkqt3",
   "client_name": "Example RP",
   "redirect_uris": [
      "https://rp.example.org/callback",
      "https://rp.example.org/callback2"
   ],
   "logo_uri": "https://rp.example.org/logo.png"
}
```
**msc** OIDC private implicit mode

RP A  

User Agent

IdP 1

**Authentication request**

client_id, redirect_uri,  
client_id_binding, rp_nonce  
(in URI fragment)

client_id_binding checks  
Get user consent

client_id_hash

private_id_token  
(in URI fragment)

Extract private_id_token

private_id_token, user_nonce

Check private_id_token

Login successful  
or Error message
Privacy result

- *client_id_hash* contains randomly generated *user_nonce*
- Looks random to the IdP
Privacy result

- `client_id_hash` contains randomly generated `user_nonce`
  - Looks random to the IdP

- No other parameters sent to the IdP
Security preservation

- *redirect_uri* check equivalent to regular implicit mode
Security preservation

- `redirect_uri` check equivalent to regular implicit mode

- End-user consent equivalent to regular implicit mode
Security preservation

- *redirect_uri* check equivalent to regular implicit mode
- End-user consent equivalent to regular implicit mode
- Check of *private_aud* equivalent to check of *aud* in regular implicit mode
Security preservation

- *redirect_uri* check equivalent to regular implicit mode

- End-user consent equivalent to regular implicit mode

- Check of *private_aud* equivalent to check of *aud* in regular implicit mode

- *rp_nonce* not explicitly part of *private_id_token*, but contained in hash
Security preservation

- *redirect_uri* check equivalent to regular implicit mode

- End-user consent equivalent to regular implicit mode

- Check of *private_aud* equivalent to check of *aud* in regular implicit mode

- *rp_nonce* not explicitly part of *private_id_token*, but contained in hash

- Modes in parallel: Messages cannot be confused
  - *private_id_token* is not a valid *id_token*
Unsupportable OIDC Features

- OAuth access *token*

  ▶ Would require direct communication from RP to IdP
  ▶ Not possible to preserve privacy
  ▶ No UserInfo endpoint: Include all information in the *id token*
  ▶ Parameters that could violate privacy:
    - `max_age`, `acr` values
    - *id token* hint: Allows logins to be linked together
  ▶ Client metadata related to *id token* generation:
    - *e.g.* *id token signed* response
      - IdP cannot look them up
      - Forwarding them could violate privacy
  ▶ Default values are used
  ▶ Pairwise subject identifier
    - Distinct *sub* identifier for each (RP, user) pair
      - To choose the right one the IdP must know the RP
Unsupportable OIDC Features

- OAuth access *token*
  - Would require direct communication from RP to IdP
  - Not possible to preserve privacy
  - No UserInfo endpoint: Include all information in the private_id_token

- Parameters that could violate privacy
  - *max_age, acr* values: Could be (close to) unique for RP
  - *id_token_hint*: Allows logins to be linked together

- Client metadata related to id_token generation
  - e.g. *id_token_signed_response*
    - IdP cannot look them up
    - Forwarding them could violate privacy

- Default values are used

- Pairwise subject identifier
  - Distinct sub identifier for each (RP, user) pair
  - To choose the right one the IdP must know the RP
Unsupportable OIDC Features

- OAuth access *token*
  - Would require direct communication from RP to IdP
  - Not possible to preserve privacy
  - No UserInfo endpoint: Include all information in the
    private_id_token

- Parameters that could violate privacy
Unsupportable OIDC Features

- OAuth access *token*
  - Would require direct communication from RP to IdP
  - Not possible to preserve privacy
  - No UserInfo endpoint: Include all information in the *private_id_token*

- Parameters that could violate privacy
  - *max_age, acr_values*: Could be (close to) unique for RP
  - *id_token_hint*: Allows logins to be linked together
Unsupportable OIDC Features

- OAuth access *token*
  - Would require direct communication from RP to IdP
  - Not possible to preserve privacy
  - No UserInfo endpoint: Include all information in the *private_id_token*

- Parameters that could violate privacy
  - *max_age, acr_values*: Could be (close to) unique for RP
  - *id_token_hint*: Allows logins to be linked together

- Client metadata related to *id_token* generation
Unsupportable OIDC Features

- OAuth access *token*
  - Would require direct communication from RP to IdP
  - Not possible to preserve privacy
  - No UserInfo endpoint: Include all information in the *private_id_token*

- Parameters that could violate privacy
  - *max_age, acr_values*: Could be (close to) unique for RP
  - *id_token_hint*: Allows logins to be linked together

- Client metadata related to *id_token* generation
  - e.g. *id_token_signed_response_alg*
  - IdP cannot look them up
  - Forwarding them could violate privacy
  - Default values are used
Unsupportable OIDC Features

- OAuth access token
  - Would require direct communication from RP to IdP
  - Not possible to preserve privacy
  - No UserInfo endpoint: Include all information in the private_id_token

- Parameters that could violate privacy
  - max_age, acr_values: Could be (close to) unique for RP
  - id_token_hint: Allows logins to be linked together

- Client metadata related to id_token generation
  - e.g. id_token_signed_response_alg
  - IdP cannot look them up
  - Forwarding them could violate privacy
  - Default values are used

- Pairwise subject identifier
Unsupported OIDC Features

- OAuth access *token*
  - Would require direct communication from RP to IdP
  - Not possible to preserve privacy
  - No UserInfo endpoint: Include all information in the *private_id_token*

- Parameters that could violate privacy
  - *max_age, acr_values*: Could be (close to) unique for RP
  - *id_token_hint*: Allows logins to be linked together

- Client metadata related to *id_token* generation
  - e.g. *id_token_signed_response_alg*
  - IdP cannot look them up
  - Forwarding them could violate privacy
  - Default values are used

- Pairwise subject identifier
  - Distinct *sub* identifier for each *(RP, user)* pair
  - To choose the right one the IdP must know the RP
Questions for the Audience

▶ Are there any fundamental problems with the approach?
Questions for the Audience

▶ Are there any fundamental problems with the approach?

▶ How important are the unsupported features?
Questions for the Audience

- Are there any fundamental problems with the approach?

- How important are the unsupported features?
  - Are there optional parameters that are often used in practice?
  - What privacy do we lose by not supporting the pairwise subject identifier?
Questions for the Audience

▶ Are there any fundamental problems with the approach?

▶ How important are the unsupported features?
  ▶ Are there optional parameters that are often used in practice?
  ▶ What privacy do we lose by not supporting the pairwise subject identifier?

▶ How realistic is the assumption to trust the JavaScript on the IdP frontend?
Questions for the Audience

- Are there any fundamental problems with the approach?

- How important are the unsupported features?
  - Are there optional parameters that are often used in practice?
  - What privacy do we lose by not supporting the pairwise subject identifier?

- How realistic is the assumption to trust the JavaScript on the IdP frontend?
  - Is there a better way to accomplish the checks in the user agent? (Browser extension probably not feasible)
Questions for the Audience

▶ Are there any fundamental problems with the approach?

▶ How important are the unsupported features?
  ▶ Are there optional parameters that are often used in practice?
  ▶ What privacy do we lose by not supporting the pairwise subject identifier?

▶ How realistic is the assumption to trust the JavaScript on the IdP frontend?
  ▶ Is there a better way to accomplish the checks in the user agent? (Browser extension probably not feasible)

▶ Would people (users, RPs, IdPs) be interested in this?