Agenda

● Core draft update: changes since IETF113 (from -08 to -09)
  ○ Editorial Changes
  ○ Functional Changes

● RS draft update: changes since IETF111 (from -01 to -02)
  ○ A handful of small editorial and structural changes

● Draft roadmap: process issue backlog

● Token model
Differences since IETF112 (Core: -09 to -10)

https://www.ietf.org/rfcdiff
  ?url2=draft-ietf-gnap-core-protocol-10
  &url1=draft-ietf-gnap-core-protocol-09

https://www.ietf.org/rfcdiff
  ?url2=draft-ietf-gnap-resource-servers-02
  &url1=draft-ietf-gnap-resource-servers-01
13 (core) & 2 (RS) Merged Pull Requests

https://github.com/ietf-wg-gnap/gnap-core-protocol/pulls?q=is%3Aclosed+closed%3A2022-03-08..2022-07-12

https://github.com/ietf-wg-gnap/gnap-resource-servers/pulls?q=is%3Aclosed+closed%3A2021-10-26..2022-07-12
40 (core) closed issues

https://github.com/ietf-wg-gnap/gnap-core-protocol/issues
?q=is%3Aissue+is%3Aclosed+closed%3A2022-03-08..2022-07-12

No closed issues on the RS draft
Editorial Changes PR Summary

- **Text consistency:**
  - #426 Added RO policy use case
  - #430 Add note on crypto agility for JOSE methods
  - #432 Reference Florian’s thesis (congrats!)

- **Editorial:**
  - #419 Translate diagrams to SVG
  - #423 Access rights as strings/objects

- **Release and cleanup:**
  - #433 Release -10
Editorial Changes

- New *fancy* diagrams
- Clarity on future-proofing for JOSE signature methods
- Clarity on mapping object rights to strings for APIs
- Discussion on how RO can have a policy in place
“Security Analysis of the Grant Negotiation and Authorization Protocol”

- Thesis paper from Florian Helmschmidt
- 125 pages of security analysis
- Conclusion points out that mitigations have been included in spec
- Future security analysis needed when GNAP is used as an identity protocol

Congratulations, and thank you!
Functional Changes PR Summary

● Security Considerations:
  ○ #415 Add diagram for session attack

● Syntax:
  ○ #418 Collapse user code into string
  ○ #421 Expanded key proofing field to allow parameters
  ○ #429 Reference hash algorithm identifiers from the IANA registry

● Dropped Features:
  ○ #420 Remove split token functionality

● Protocol Flow:
  ○ #427 Added state diagram and discussion
  ○ #428 Allow interactions to time out
User Code Interactions

- Previously used an object with a single string (holdover from old syntax)
- Now uses a string directly

Old mode:
```json
{
    "user_code": {
        "code": "ABC1GHF"
    }
}
```

New mode:
```json
{
    "user_code": "ABC1GHF"
}
```
Dropping split_token

● Functionality to allow single token request to return multiple tokens
● Motivation: allow an AS to split and target different RS’s
● Why drop this?
  ○ No strong demand from community at this time and it was complicated
Parameters for Key Proofs

- Previously used a single string
- Now uses an object when needed, a string otherwise
  - Deterministic mapping from string to object
  - Defaults must be defined for all parameters

Old mode/new mode (collapsed):

```json
{
  "proof": "httpsig"
}
```

New mode (expanded):

```json
{
  "proof": {
    "method": "httpsig",
    "alg": "rsa-pss-sha512",
    "content-digest-alg": "sha512"
  }
}
```
Hash Algorithm Names

- Hash algorithms for interaction hash calculation now point to an IANA registry for cryptographic algorithms

Old:

hash_method=sha2

New mode:

hash_method=sha2-512
Lifecycle and Diagram
Grant Request Lifecycle

- New text added to describe different states of an ongoing grant request
  - What can be done in each state
  - How the system transitions between states
- No protocol changes
  - Clarifications on which request/response fields can be used at which states
  - Guidance for AS developers
GNAP State Diagram

- Request → Processing
- Need Interaction → Pending
- Finish Interaction (approve/deny) → Pending
- Finalize → Finalized
- Update → Approved
- No Interaction → Approved
- Revoke or Finalize → Approved
- Continue → Pending
- Cancel → Finalized
- Continue → Pending
Grant Request Lifecycle Stages

- **Processing**: The AS is actively working on figuring it out
  - AS looks at full state and context of request to determine what’s next
  - Initial state on creation, state when new information is received
  - No incoming updates/continuations can be received until it’s out of this state
  - Need interaction or more information? Move to *pending*.
  - Ready to issue results? Move to *approved*.
  - Can’t process any more? Move to *finalized*.

- **Pending**
- **Approved**
- **Finalized**
Grant Request Lifecycle Stages

- **Processing**
- **Pending**: The AS needs more information from someone
  - Interaction could be taking place right now
  - The client instance can send interaction-finish and other follow-up messages
    - When received, move to *processing*.
  - AS can perform out-of-band authorization
    - When finished, move to *processing*.
  - No modifications to request can be received
    - (receiving one is an error)
  - No tokens can be issued, no subject information is returned
  - Need to cancel? Move to *finalized*.

- **Approved**
- **Finalized**
Grant Request Lifecycle Stages

- **Processing**
- **Pending**
- **Approved**: The AS has what it needs
  - No interaction / authorization is taking place
  - New access tokens can be issued in responses
    - Token rotation happens through token management API
  - Subject information can be returned in responses
  - Client instance can send a modification
    - When received, move to *processing*.
  - Request revoked? Move to *finalized*.
- **Finalized**
Grant Request Lifecycle Stages

- **Processing**
- **Pending**
- **Approved**
- **Finalized**: The grant is done, nothing more can happen
  - No tokens can be issued
  - No subject information can be returned
  - No modifications or continuations can be received
  - Final state, can’t move out of it

- To get more tokens create a new grant request

![Grant Request Lifecycle Diagram]
Future Work

● Prioritize Inline issues (mentioned in the text):
  ○ #44, #54, #58, #69, #71, #78, #85, #87, #89, #88, #91, #103, #104, #105

● Key rotation for tokens

● Management for clients

● RS Draft
  ○ Token Model
Related Work

- HTTP Message Signatures
  - WGLC requested* in HTTP WG
- SECEVENT Subject Identifiers
  - Post-WGLC edits happening now
Key Rotation
Key Rotation for Tokens

- Non-bearer Tokens have keys associated with them
- We want to have ways of associating new keys
- Need a way to prove possession of old and new key simultaneously
  - There doesn’t seem to be a way to do this across all different proof types
- Proposal: allow different kinds of key presentation with different types
- Pattern:
  - sign(new_key, sign(old_key, message-with-new_key_value))
Key Rotation with HTTPSig

- HTTP Message Signatures allow multiple signatures within a single message
- Include the new key in the token rotation message
- Sign the message with the new key
- Sign the new signature with the old key to prove you still have it

Signature: sig1="signature";key="sig2" ... ;keyid= new_key, sig2=(...);keyid= old_key

{
    "proof": "httpsig",
    "new_key": { "jwk" ... } // new key included in message
}
Key Rotation with JOSE

- JOSE-based methods could include embedded JWS objects
- Sign the outer message with the new key and sign the new key value in the message with the old key
- Probably assumes JOSE-based key format (JWK)

Detached-JWS: eyj0... // message signed with new key

{
    "proof": "jwsd",
    "new_key": "eyj0..." // new key signed with old key
}
Key Rotation with MTLS

- Many MTLS systems will rely on certificate management systems (PKI), so rotation is out of scope for GNAP
- Could use ACME’s proposed client certificate extensions
Key Rotation across proofing types

- How do you combine or change methods?
  - What if you start with djws and want to move to httpsig?
  - What if you start with httpsig and want to move to MTLS?

- Do we need to address this or is the proof type fixed?

- If we do allow changing proof types:
  - Is that a security hole?
  - Does the MxM combination grid need to be complete?
  - Who defines all the combinations?
Could we let a client instance request an access token using a different key/proofing method than its base request to the AS?

Do we require the client prove that it can use the RS-facing key as well?

```json
{
    "client": {
        "proof": "httpsig",
        "key": "AS-facing-key-value"
    },
    "access_token": {
        "access": {
            "value": "123459387345"
        },
        "proof": "httpsig",
        "key": "RS-facing-key-value"
    }
}
```
Client Instance Management
Management for client instances

- When a client instance presents itself dynamically by-value it can receive a client instance identifier in response.
- What if we also gave the client an API and access token to manage itself, like RFC6792 (OAuth Dynamic Client Registration Management)?

```json
{
    "client": {
        "access_token": {"value": "123459387345"},
        "uri": "https://server.com/client/ABCDSA",
        "instance_id": "abc-dsa-123-42"
    }
}
```
Client instance management API

- REST-ish API for updating client properties associated with an instance identifier
- Response can contain a new access token and URI
  - Especially useful if keys are rotated since new token can be bound to new key, old token is thrown out

Authorization: GNAP 123459387345
Signature: sig1=...

```json
{
  "client": {
    "display": {"uri": "https://home.client.example/"},
    "proof": "httpsig",
    "key": { ... rotated client key ... }
  }
}
```
Token model (open discussion)
RS draft - issue #15

- Initial comment from Yaron: “we should specify a minimal, generic format in an appendix, as a non-normative starting point for developers. It would be better than having each implementation make its own mistakes“

- Editors’ response (spoiler): a token model would make sense

- What do you think?
### 6.1.2. Initial Registry Contents

The table below contains the initial contents of the GNAP Token Format Registry.

<table>
<thead>
<tr>
<th>Name</th>
<th>Status</th>
<th>Description</th>
<th>Reference</th>
</tr>
</thead>
<tbody>
<tr>
<td>jwt-signed</td>
<td>Active</td>
<td>JSON Web Token, signed with JWS</td>
<td>[RFC7519]</td>
</tr>
<tr>
<td>jwt-encrypted</td>
<td>Active</td>
<td>JSON Web Token, encrypted with JWE</td>
<td>[RFC7519]</td>
</tr>
<tr>
<td>macaroon</td>
<td>Active</td>
<td>Macaroon</td>
<td></td>
</tr>
<tr>
<td>biscuit</td>
<td>Active</td>
<td>Biscuit</td>
<td></td>
</tr>
<tr>
<td>zcap</td>
<td>Active</td>
<td>ZCAP</td>
<td></td>
</tr>
</tbody>
</table>

*Table 1: Initial contents of the GNAP Token Format Registry.*
Other token formats (not exhaustive)

Sources: podcast, blog (the evaluation isn’t mine and is subjective)
Token format / model

- **Token format** = typically JOSE/JWT, but other formats exist
  - People tend to be opinionated (that’s fine), but the main reason why we want to accommodate various formats are:
    - Evolutivity (see appendix for JOSE examples)
    - Different use cases: JWT (additive) vs macaroon/biscuit (subtractive, e.g. attenuation)

- **Token model** = information typically shared by implementations
  - The Editors suggest focusing on the model in the spec
Where should be describe this model?

● In the core spec, we have Appendix C “component data models”
  ○ Either a basic summary of where the RS fits in the main picture
    ■ Token has RO, user, client, resource list, RS list, etc. (TBD)
  ○ Or detail the token model there

● The RS draft might be a better placeholder
  ○ A model (as a core part of the text)
  ○ A link to supported formats
    ■ with/without explicit schema mapping?
    ■ at least we should make sure it can be done
What’s an access_token value?

- Reminder: the access token is opaque to the client

- In the core spec, [section 3.2.1 on access_token](#)
  - value (string): The value of the access token as a string. The value is opaque to the client instance. The value SHOULD be limited to ASCII characters to facilitate transmission over HTTP headers within other protocols without requiring additional encoding. REQUIRED.

- In the AS response, we also have metadata on what that value means (label, associated rights, etc.) and technical data (key, expires_in, flags, manage URL)
Token introspection

- **RS draft, section 3.3**
  - Reminder: a token may be bearer or key-bound (default)

  The RS signs the request with its own key and sends the access token as the body of the request.

  **access_token** (string): **REQUIRED.** The access token value presented to the RS by the client instance.

  **proof** (string): **RECOMMENDED.** The proofing method used by the client instance to bind the token to the RS request.

  **resource_server** (string or object): **REQUIRED.** The identification used to authenticate the resource server making this call, either by value or by reference as described in Section 3.2.

  **access** (array of strings/objects): **OPTIONAL.** The minimum access rights required to fulfill the request. This **MUST** be in the format described in the Resource Access Rights section of [I-D.ietf-gnap-core-protocol].
Relevant IETF documents

- **Cross-domain Identity Management schema**
- **JWT profile for OAuth access tokens (draft)**
  - Header (including “typ” = at+jwt)
  - Claims
    - Note: this draft introduces new IANA values for roles / groups / entitlements
- **JWT BCP**

2.8. **Cross-JWT Confusion**

As JWTs are being used by more different protocols in diverse application areas, it becomes increasingly important to prevent cases of JWT tokens that have been issued for one purpose being subverted and used for another. Note that this is a specific type of substitution attack. If the JWT could be used in an application context in which it could be confused with other kinds of JWTs, then mitigations MUST be employed to prevent these substitution attacks.
Cross-JWT confusion : mitigations

- 3.8 Validate Issuer (“iss”) and Subject (“sub”)
- 3.9 Use and validate Audience (“aud”)
- 3.10 Do not trust received claims : “kid”, “jku" (JWK set URL) or "x5u" (X.509 URL)
- 3.11 Use explicit typing : “typ” = gnapi+jwt?
Other relevant documents

- It can still be useful to compare the format structures to see what’s different, and make sure the mapping is possible
  - Example: **PASETO**:
    - Version ~ equivalent to 4 new JOSE algs (v3.local, v3.public, same for v4, other algs prohibited). Still need to trust the message to know which alg is used.
    - Purpose ~ kind of similar to aud
    - Payload (base64URL encoded)
    - Footer (option, useful for key rotation)
  - Example: **biscuit**
    - Based on ED25519 [https://www.biscuitsec.org/docs/reference/cryptography/](https://www.biscuitsec.org/docs/reference/cryptography/)
    - Datalog engine to (logically) process claims
      - With blocks and base types (int, string, date, bool, byte array, set)
    - UUID / revocation ID (equivalent to jti)
What’s different in GNAP?

- Changes from OAuth2 -> GNAP
  (see core Appendix D.5)
  - Sub : opaque sub_id only ?
  - Clientid -> Client
  - Scope -> Access
  - Flags / keybound ?
  - Expires_in ?
  - RO ? (in practice, often same as user)

(claim part only here)

```json
{
    "iss": "https://as.example.com/",
    "sub": "5ba552d67",
    "aud": "https://rs.example.com/",
    "exp": 1639528912,
    "iat": 1618354090,
    "jti" : "dbe39bf3e1691c4",
    "client": "s6BhdRkqt3"
    "access" : optional/redundant/reference?
}
```
Summary

- Header: new “typ” for GNAP JWT
- Minimal claims: “iss”, “sub”, “aud”, “jti”, “client”
- And?
Next steps

- Submit a PR to review
- Feedback welcome
Appendix : JOSE/JWT evolution ideas

- https://datatracker.ietf.org/group/jwp/about/ (anonymous credentials / selective disclosure)

This isn’t an exhaustive list, it’s only there to show some recent trends.