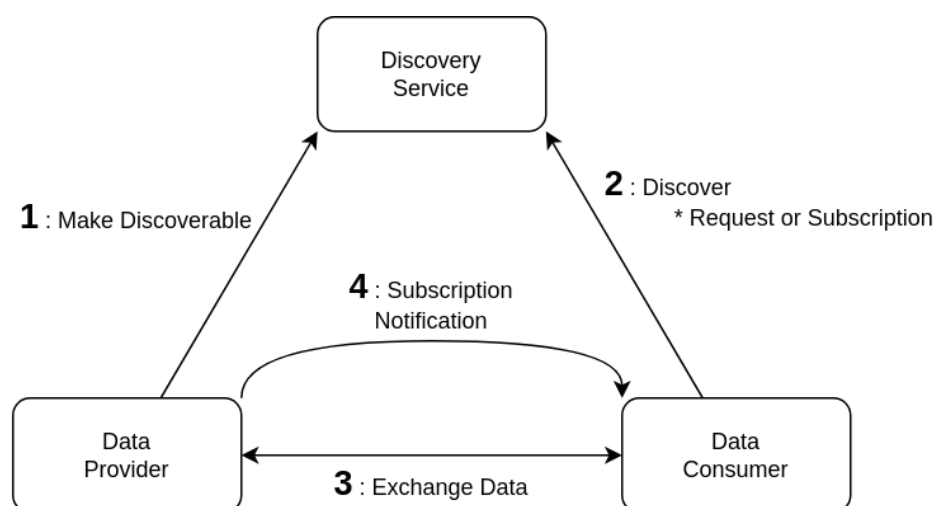


# Autenticação Service Provider

## API Key:

Na comunicação do BR-UTM (imagem abaixo), as requisições devem ser autenticadas e autorizadas. Devido à natureza distribuída da arquitetura, não é viável que cada provedor possua sua lógica de autenticação e autorização.



Portando, a solução proposta pela ASTM é a de um servidor de autenticação central onde os provedores obtém tokens OAuth2 codificados e assinados em JWT. O Validator da documentação abaixo checa a validade da assinatura do token, utilizando a chave pública do Auth Server.

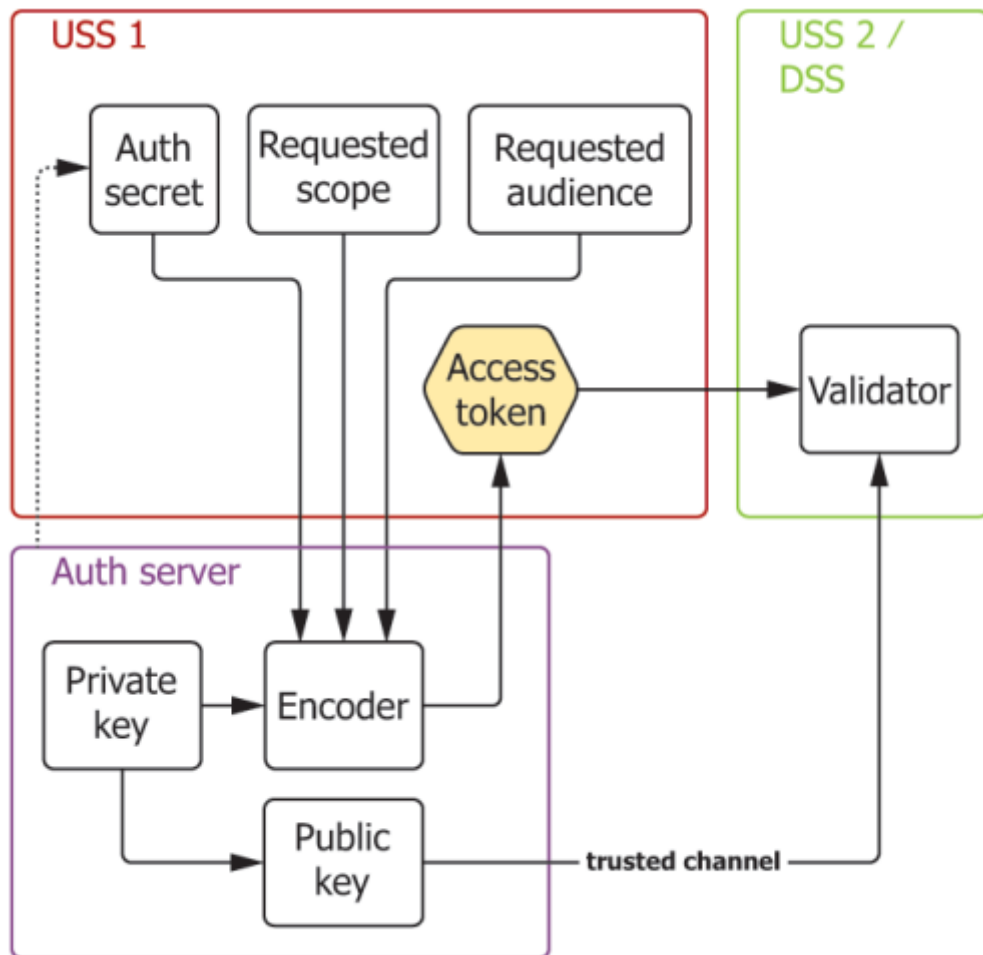
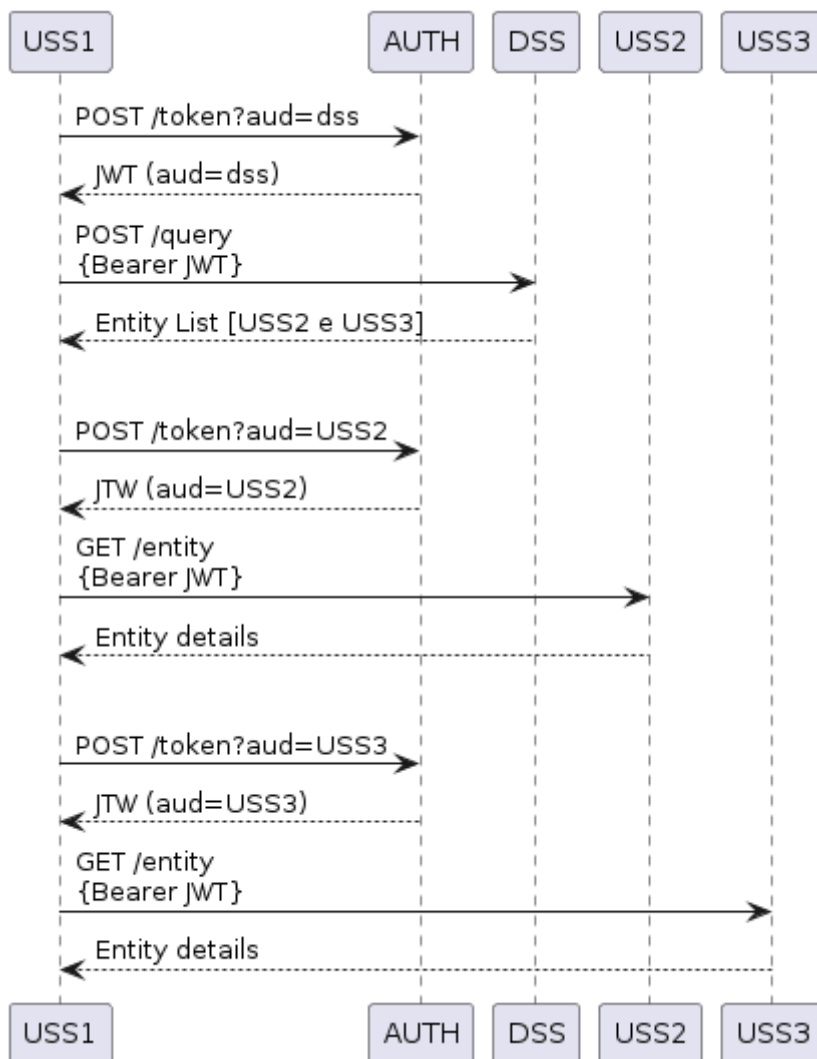


FIG. X1.1 Access Tokens With Audience Claims

Um exemplo de troca de mensagens autenticadas é:

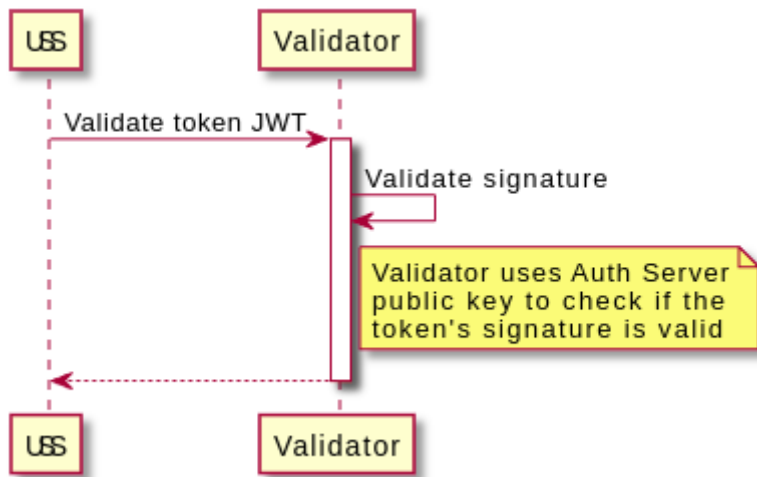


## Uso da API Key

Com sua API Key, você pode realizar ações programaticamente no ECO-UTM:

1. **Insira a sua API Key no header da requisição;**
2. **Insira o** `scope`
3. **Insira o** `intended_audience`
  1. **Em caso de comunicação com outro USS, o preencha com o conteúdo do campo** `manager` **da resposta do DSS.**

## Validator



## Implementação

Código de exemplo para início da implementação do Auth Server e do Validator em Go

### Código exemplo

```
package main

import (
    "encoding/json"
    "flag"
    "log"
    "net/http"
    "os"
    "strings"

    "github.com/golang-jwt/jwt"
)

var (
    keyFile      = flag.String("private_key_file", "auth.key", "OAuth private key file")
    publicKeyFile = flag.String("public_key_file", "auth.pem", "OAuth public key file")
)

func verifyToken(token string) (bool, error) {
    bytes, err := os.ReadFile(*publicKeyFile)
    if err != nil {
        log.Panic(err)
    }
```

```
}
}
```

```
pubkey, err := jwt.ParseRSAPublicKeyFromPEM(bytes)
```

```
if err != nil {
```

```
    log.Panic(err)
```

```
}
```

```
parts := strings.Split(token, ".")
```

```
err = jwt.SigningMethodRS256.Verify(strings.Join(parts[0:2], "."), parts[2], pubkey)
```

```
if err != nil {
```

```
    return false, nil
```

```
}
```

```
return true, nil
```

```
}
```

```
func main() {
```

```
    http.HandleFunc("/validate", func(w http.ResponseWriter, r *http.Request) {
```

```
        tokenString := r.URL.Query().Get("token")
```

```
        valid, err := verifyToken(tokenString)
```

```
        if err != nil {
```

```
            log.Panic(err)
```

```
        }
```

```
        log.Println(valid)
```

```
    })
```

```
    http.HandleFunc("/token", func(w http.ResponseWriter, r *http.Request) {
```

```
        //
```

```
        token := jwt.NewWithClaims(jwt.SigningMethodRS256, jwt.MapClaims{
```

```
            "aud": "aud",
```

```
            "scope": "scope",
```

```
            "iss": "iss",
```

```
            "exp": "exp",
```

```
            "sub": "sub",
```

```
        })
```

```
        // Read private key
```

```

    []bytes, err := os.ReadFile(*keyFile)
    []if err != nil {
    []    []log.Panic(err)
    []}
    []privateKey, err := jwt.ParseRSAPrivateKeyFromPEM(bytes)
    []if err != nil {
    []    []log.Panic(err)
    []}

    []// Sign and get the complete encoded token as a string using the secret
    []tokenString, err := token.SignedString(privateKey)
    []if err != nil {
    []    []log.Panic(err)
    []}

    []resp := make(map[string]string)
    []resp["access_token"] = tokenString
    []jsonResp, err := json.Marshal(resp)
    []if err != nil {
    []    []log.Fatalf("Error happened in JSON marshal. Err: %s", err)
    []}
    []w.WriteHeader(http.StatusOK)
    []w.Header().Set("Content-Type", "application/json")
    []w.Write(jsonResp)
    []return

    []})

    []log.Fatal(http.ListenAndServe(":9096", nil))
}

```

## Eco-UTM Autenticator Public Key

-----BEGIN PUBLIC KEY-----

MIGeMA0GCSqGSIb3DQEBAQUAA4GMADCBiAKBgHkNtpy3GB0YTCI2VCCd22i0rJwl

GBSazD4QRKvH6rch0IP4igb+02r7t0X//tuj0VbwtJz3cEICP8OGSqrDTSCGj5Y0

30a2gPkx/0c0V8D0eSXS/CUC0qrYHnAGLqko7eW87HW0rh7nnl2bB4Lu+R8fOmQt

5frCJ5eTkzwK5YczAgMBAAE=

-----END PUBLIC KEY-----

## Lista de endpoints

A lista completa de *endpoints* também está disponível neste [link](#), [neste arquivo OpenAPI](#) e [nesta coleção no Insomina](#).

## ECO-UTM

A URL base para os seguintes *endpoints* é <http://montreal.icea.decea.mil.br:64235/>

GET

/token

Aprovar token de autenticação do provedor associado ao usuário

### Path Param

intented_audience	user da entidade de destino da mensagem
scope	escopo da requisição
apikey	chave recebida do ICEA

### Bearer

token	Bearer token gerado
-------	---------------------

### Código exemplo python

```
response = requests.get(
    f"{AUTH_URL}/token",
    params={
        "grant_type": "client_credentials",
        "intended_audience": "localhost",
        "scope": "utm.constraint_management",
        "apikey": "brutm",
    },
)
```

### Response

200	Success
403	Non-Authoritative Information

### Response Body

access_token	token de acesso ao ECO-UTM
--------------	----------------------------



Revision #20  
Created 25 June 2024 13:21:50 by Cenato  
Updated 1 July 2024 14:16:30 by Cenato