OAuth &
Proof of Possession

The long way round

Dominick Baier
@leastprivilege

Duende.
https://duendesoftware.com

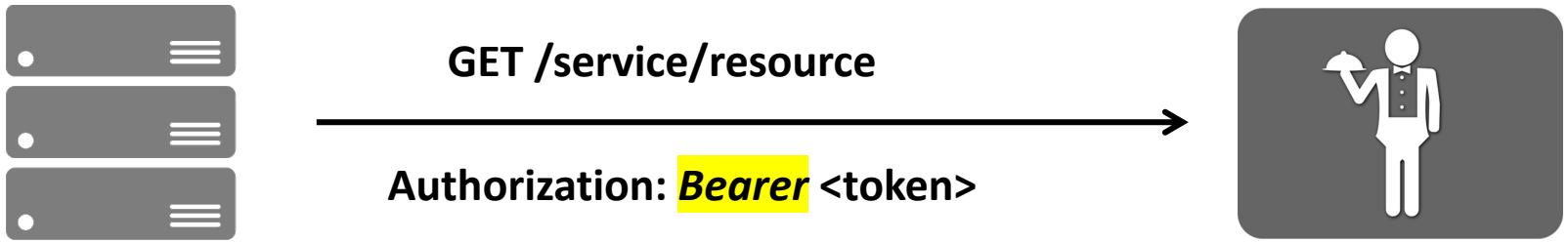
Me

- **Dominick**
  - 15+ years consulting and project work in the identity space
  - co-creator of IdentityServer & IdentityModel OSS projects
  - co-Founder of Duende Software

slides https://speakerdeck.com/duendesoftware

OAuth Bearer Tokens

- **Caller needs access token to call API**
  - transmitted via *Authorization* header using the *Bearer* scheme



https://tools.ietf.org/html/rfc6750

High Security OAuth

norskhelsenett

MITRE

Enterprise Mission Tailored
OAuth 2.0 Profile

FAPI/FAPI2

Do you remember?

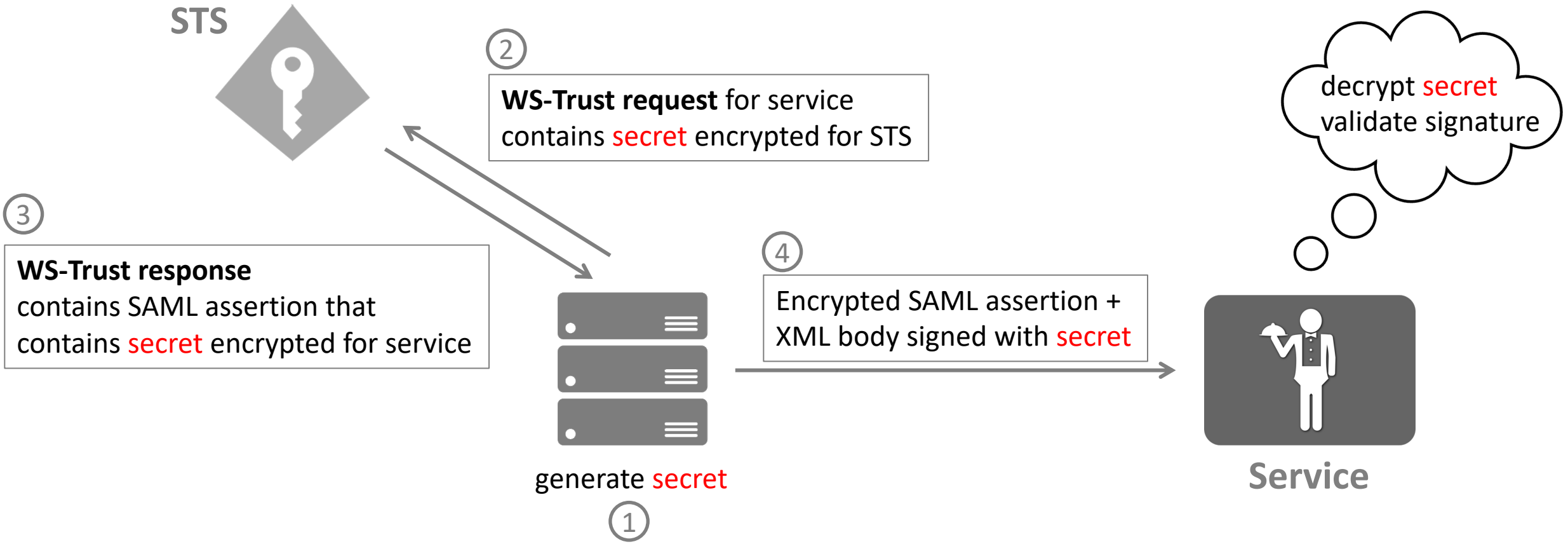
Web Services Security:
SOAP Message Security 1.1
(WS-Security 2004)

OASIS Standard Specification, 1 February 2006

Chairs:
Kelvin Lawrence, IBM
Chris Kaler, Microsoft

Editors:
Anthony Nadalin, IBM
Chris Kaler, Microsoft
Ronald Monzillo, Sun
Phillip Hallam-Baker, Verisign

WS-Security Proof of Possession



What about OAuth 1.0?

Internet Engineering Task Force (IETF) E. Hammer-Lahav, Ed.
Request for Comments: 5849 April 2010
Category: Informational
ISSN: 2070-1721

The OAuth 1.0 Protocol

Abstract

OAuth provides a method for clients to access server resources on behalf of a resource owner (such as a different client or an end-user). It also provides a process for end-users to authorize third-party access to their server resources without sharing their credentials (typically, a username and password pair), using user-agent redirections.

…defaulted to signature-bound tokens

3.4.1. Signature Base String

The signature base string is a consistent, reproducible concatenation of several of the HTTP request elements into a single string. The string is used as an input to the "HMAC-SHA1" and "RSA-SHA1" signature methods.

The signature base string includes the following components of the HTTP request:

- The HTTP request method (e.g., "GET", "POST", etc.).
- The authority as declared by the HTTP "Host" request header field.
- The path and query components of the request resource URI.
- The protocol parameters excluding the "oauth_signature".
- Parameters included in the request entity-body if they comply with the strict restrictions defined in Section 3.4.1.3.

https://www.rfc-editor.org/rfc/rfc5849#section-3.4

What about OAuth 2.0?

OAuth Bearer Tokens are a Terrible Idea

By **Eran Hammer**
Wednesday September 29, 2010

My last post about the **lack of signature support in OAuth 2.0** stirred some very good discussions and showed wide support for including a signature mechanism in OAuth 2.0. The discussions on the working group focused on the best way to include signature support, and a bit on the different approached to signatures (more on that in another post).

OAuth 2.0 and the Road to Hell

By **Eran Hammer**
Thursday July 26, 2012

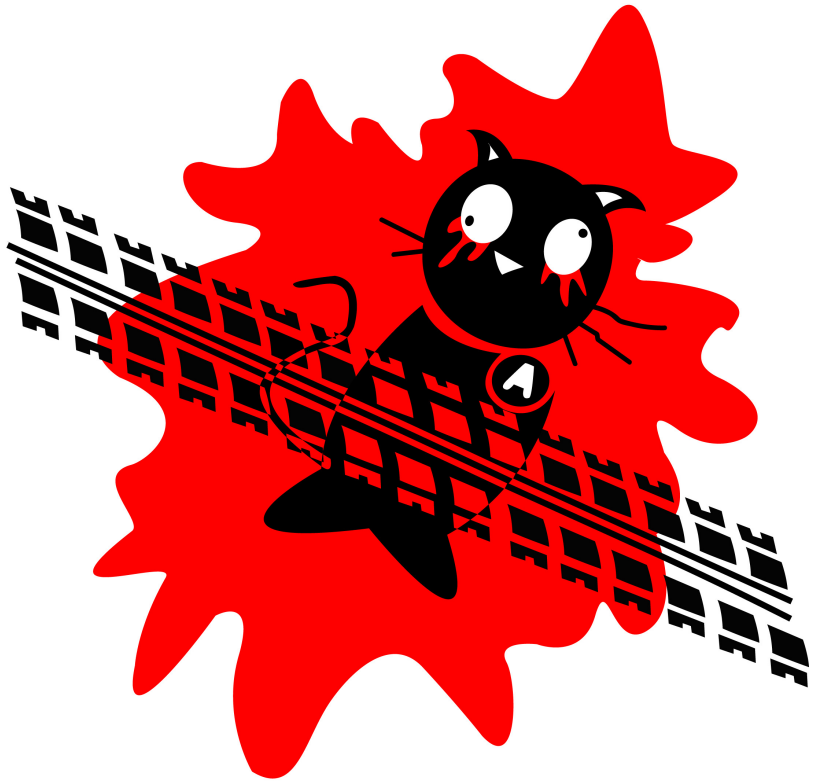
They say the road to hell is paved with good intentions. Well, that's **OAuth 2.0**.

Last month I reached the painful conclusion that I can no longer be associated with the OAuth 2.0 standard. I resigned my role as lead author and editor, **withdraw my name from the specification**, and left the working group. Removing my name from a document I have painstakingly labored over for three years and over two dozen drafts was not easy. Deciding to move on from an effort I have led for over five years was agonizing.

When compared with **OAuth 1.0**, the 2.0 specification is more complex, less interoperable, less useful, more incomplete, and most importantly, less secure.

https://web.archive.org/web/20120730164629/http://hueniverse.com/2012/07/oauth-2-0-and-the-road-to-hell/

...specified bearer tokens only



https://vimeo.com/52882780 #fuckoauth

Postponing PoP

RFCs (27 hits)			
RFC 6749 *(was draft-ietf-oauth-v2)* **The OAuth 2.0 Authorization Framework**	76 pages Errata	2012-10	Proposed Standard RFC Updated by RFC 8252, RFC 8996
RFC 6750 *(was draft-ietf-oauth-v2-bearer)* **The OAuth 2.0 Authorization Framework: Bearer Token Usage**	18 pages Errata	2012-10	Proposed Standard RFC Updated by RFC 8996
RFC 6819 *(was draft-ietf-oauth-v2-threatmodel)* **OAuth 2.0 Threat Model and Security Considerations**	71 pages Errata	2013-01	Informational RFC

https://datatracker.ietf.org/group/oauth/documents/

Attempt #1: OAuth PoP

draft-ietf-oauth-pop-key-distribution-07 **OAuth 2.0 Proof-of-Possession: Authorization Server to Client Key Distribution**	17 pages	2019-03-27	Expired WG Document : Proposed Standard
draft-ietf-oauth-signed-http-request-03 **A Method for Signing HTTP Requests for OAuth**	13 pages	2016-08-08	Expired WG Document

RFC 7800 *(was draft-ietf-oauth-proof-of-possession)* **Proof-of-Possession Key Semantics for JSON Web Tokens (JWTs)**	15 pages Errata	2016-04	Proposed Standard RFC

Attempt #2: Token Binding

draft-ietf-oauth-token-binding-08 **OAuth 2.0 Token Binding**	30 pages	2018-10-19	Expired WG Document

Intent to Remove: Token Binding 3839 views

Nick Harper Aug 1, 2018, 7:36:10 PM

to blink-dev, net-dev

Primary eng (and PM) emails
nha...@chromium.org

Links to other Intents threads
Intent to Implement: https://groups.google.com/a/chromium.org/d/topic/blink-dev/jTwWj2Y_IPM/discussion
Intent to Ship (take 1): https://groups.google.com/a/chromium.org/d/topic/blink-dev/r4zE8RKB6l4/discussion
Intent to Ship (take 2): https://groups.google.com/a/chromium.org/d/topic/blink-dev/C-iuVJeDGkk/discussion

Summary
Token Binding is a feature that has been behind a flag for over two years and has never shipped. (No Intent to Ship for Token Binding ever got approved.) I plan to remove all code related to this unused feature.

Motivation
After weighing the security benefit of Token Binding against the engineering costs, maintenance costs, web compatibility risk, and adoption, it does not make sense to ship this feature.

Then FAPI comes along…

OpenID

Tokens are Sender Constrained instead of being bearer

Security Levels	Token Types	Notes
	Sender Constrained Token	Only the entity that was issued can used the token. 
	Bearer Token	Stolen tokens can also be used 

PoP at the Transport Layer

Stream:	Internet Engineering Task Force (IETF)
RFC:	8705
Category:	Standards Track
Published:	February 2020
ISSN:	2070-1721

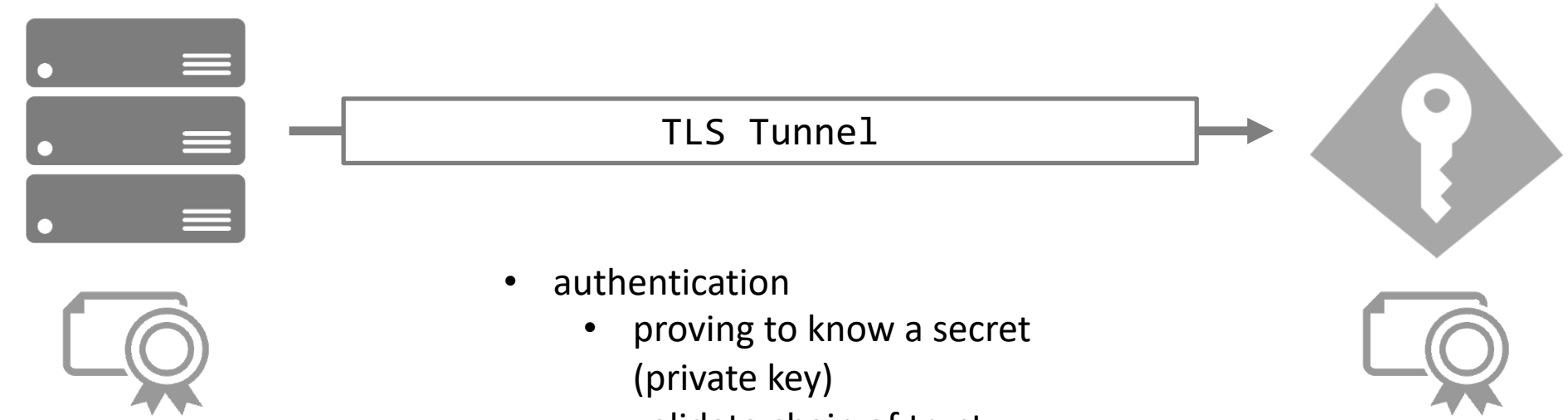
Authors:	B. Campbell	J. Bradley	N. Sakimura	T. Lodderstedt
	Ping Identity	*Yubico*	*Nomura Research Institute*	*YES.com AG*

RFC 8705
OAuth 2.0 Mutual-TLS Client Authentication and Certificate-Bound Access Tokens

Abstract

This document describes OAuth client authentication and certificate-bound access and refresh tokens using mutual Transport Layer Security (TLS) authentication with X.509 certificates. OAuth clients are provided a mechanism for authentication to the authorization server using mutual TLS, based on either self-signed certificates or public key infrastructure (PKI). OAuth authorization servers are provided a mechanism for binding access tokens to a client's mutual-TLS certificate, and OAuth protected resources are provided a method for ensuring that such an access token presented to it was issued to the client presenting the token.

Mutual TLS



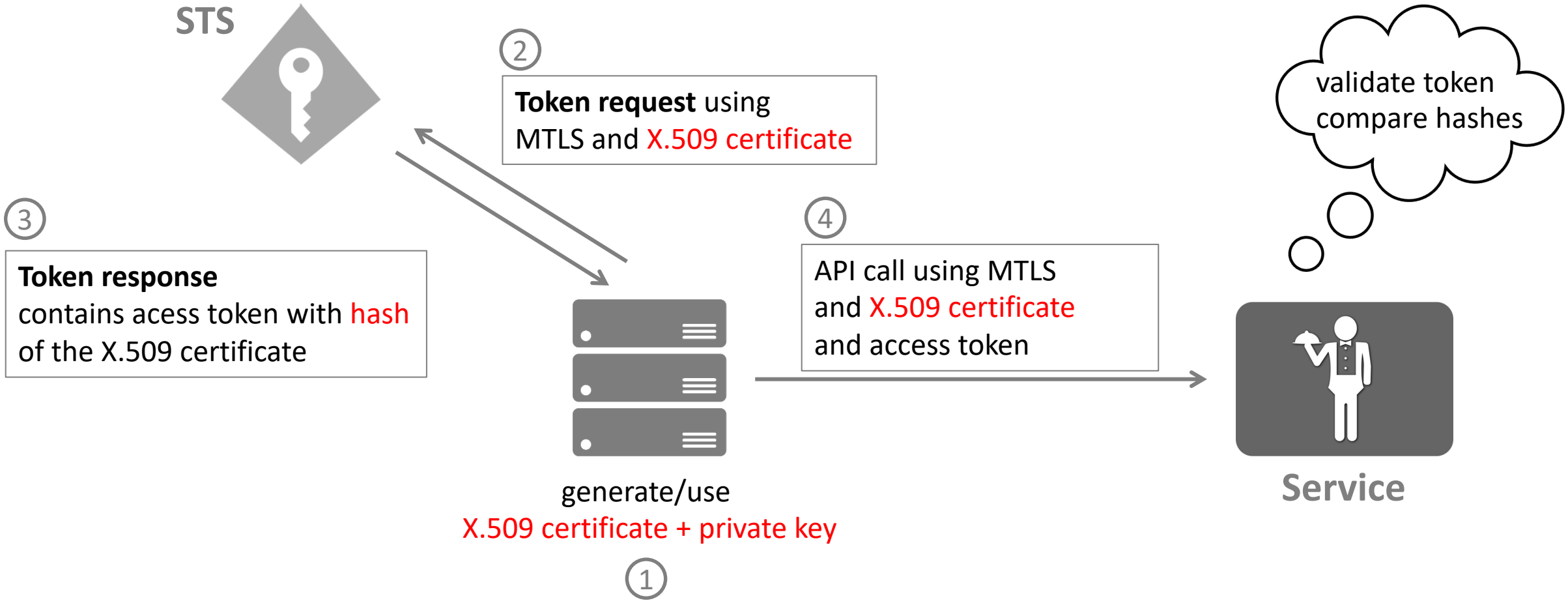
- authentication
  - proving to know a secret (private key)
  - validate chain of trust (optional)
- negotiate key material
  - sign & encrypt traffic

client certificate

server certificate

https://www.youtube.com/watch?v=YteUqj1mmo0

MTLS PoP


STS
2
Token request using
MTLS and X.509 certificate
validate token
compare hashes
3
Token response
contains acess token with hash
of the X.509 certificate
4
API call using MTLS
and X.509 certificate
and access token
generate/use
X.509 certificate + private key
1
Service

Example Access Token

```
{
    header
}.
{
    "iss": "https://server.example.com",
    "client_id": "client1",
    "exp": 1493726400,

    "cnf": { "x5t#S256": "bwcK0esc3ACC3DB2Y5_lESsXE8o9ltc05O89jdN-dg2" }
}.
signature
```

https://tools.ietf.org/html/rfc7800

Creating an X.509 Client Certificate

```
static X509Certificate2 CreateClientCertificate(string name)
{
    var distinguishedName = new X500DistinguishedName($"CN={name}");

    using (var rsa = RSA.Create(2048))
    {
        var request = new CertificateRequest(
            distinguishedName, rsa, HashAlgorithmName.SHA256,RSASignaturePadding.Pkcs1);

        request.CertificateExtensions.Add(
            new X509KeyUsageExtension(X509KeyUsageFlags.DataEncipherment | X509KeyUsageFlags.KeyEncipherment |
                                      X509KeyUsageFlags.DigitalSignature, false));

        request.CertificateExtensions.Add(
            new X509EnhancedKeyUsageExtension(
                new OidCollection { new Oid("1.3.6.1.5.5.7.3.2") }, false));

        return request.CreateSelfSigned(
            new DateTimeOffset(DateTime.UtcNow.AddDays(-1)),
            new DateTimeOffset(DateTime.UtcNow.AddDays(10)));
    }
}
```

MTLS Summary

- **Pros**
  - easy to use if infrastructure exists already
  - proven technology
  - can be also combined with special hardware (e.g. smartcards)

- **Cons**
  - not always easy (or possible) to deploy
    - especially for internet connected scenarios
  - not suitable for browser-based clients

PoP at the Application Layer

OAuth DPoP specification is in the hands of the RFC Editor

The OAuth 2.0 Demonstrating Proof-of-Possession at the Application Layer (DPoP) specification was approved by the IESG and is now in the hands of the RFC Editor in preparation for publication as an RFC. In a related development, the multiple IANA registrations requested by the specification are already in place.

Published: 29 December 2022
Intended Status: Standards Track
Expires: 2 July 2023

Authors: D. Fett *yes.com*, B. Campbell *Ping Identity*, J. Bradley *Yubico*, T. Lodderstedt *yes.com*, M. Jones *Microsoft*, D. Waite *Ping Identity*

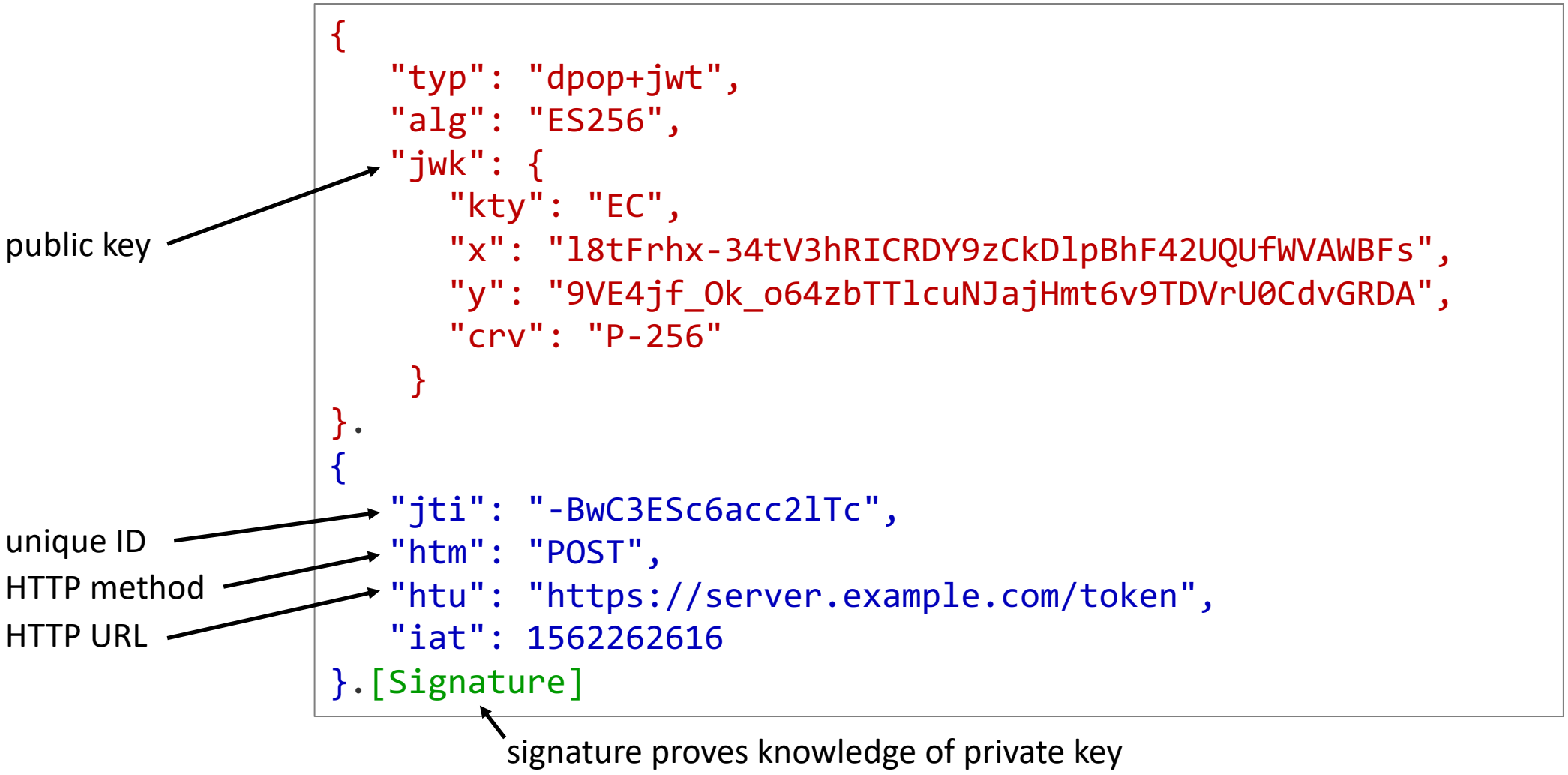
OAuth 2.0 Demonstrating Proof-of-Possession at the Application Layer (DPoP)

Abstract

This document describes a mechanism for sender-constraining OAuth 2.0 tokens via a proof-of-possession mechanism on the application level. This mechanism allows for the detection of replay attacks with access and refresh tokens.

DPop Token Request

DPop Proof Token (Token Request)



DPop Token Response

- **Access token contains the public key as confirmation method**

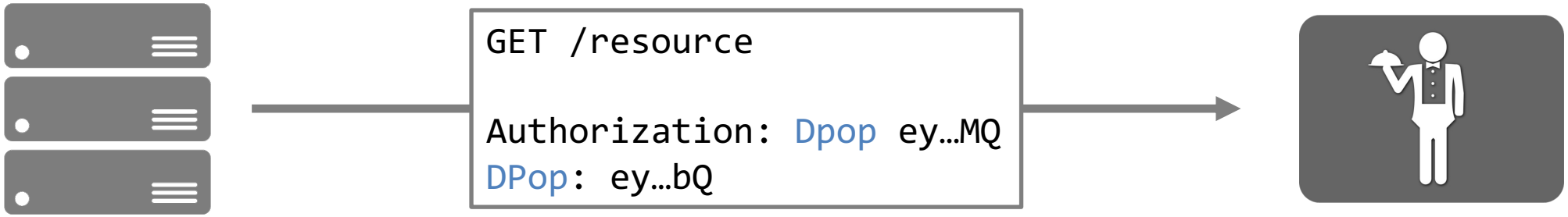
```
{
    "iss": "https://server.example.com",
    "client_id": "client1",
    "exp": 1493726400,

    "cnf": { "jkt":"0ZcOCORZNYy-DWpqq30jZyJGHTN0d2HglBV3uiguA4I" }
}
```

Base64url encoding of the JWK SHA-256 Thumbprint of the public key (RFC 7638)

https://tools.ietf.org/html/rfc7638

Resource Access



DPoP Proof Token (Resource Access)

same public key

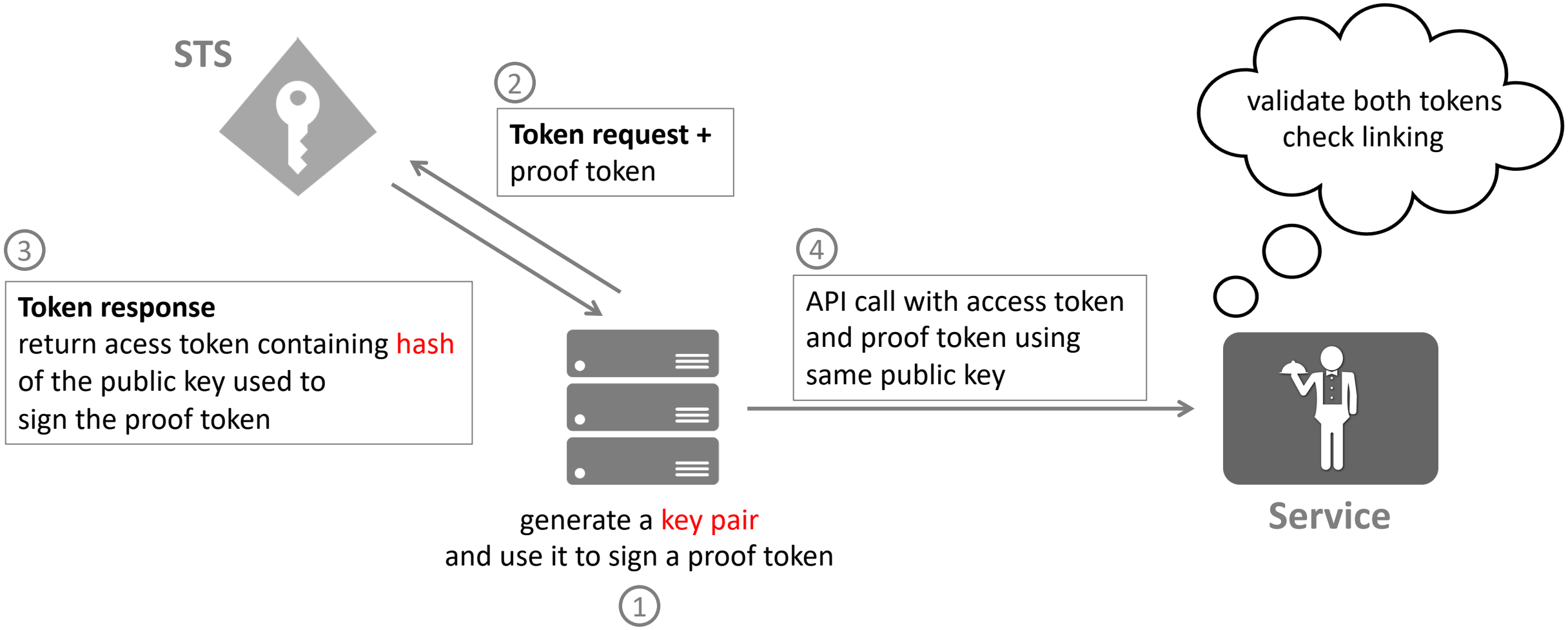
resource must create *jkt* and compare with access token

hash of access token

```
{
   "typ": "dpop+jwt",
   "alg": "ES256",
   "jwk": {
      "kty": "EC",
      "x": "l8tFrhx-34tV3hRICRDY9zCkDlpBhF42UQUfWVAWBFs",
      "y": "9VE4jf_Ok_o64zbTTlcuNJajHmt6v9TDVrU0CdvGRDA",
      "crv": "P-256"
   }
}.
{
   "jti": "e1j3V_bKic8-LAEB",
   "htm": "GET",
   "htu": "https://resource.example.org/resource",
   "ath": "lcuNJajHmt6v9TDVrU0Cd",
   "iat": 1562262618
}.[Signature]
```

signature proves knowledge of private key

DPoP


STS
2
Token request +
proof token
validate both tokens
check linking
3
Token response
return acess token containing hash
of the public key used to
sign the proof token
4
API call with access token
and proof token using
same public key
generate a key pair
and use it to sign a proof token
1
Service

Sometimes "simple" is too simple

- **Might need to sign more request data than URL and method**
  - implementation and canonicalization format is up to you
- **Proof token receiver needs to determine their own acceptance policy**
  - "issued at" timestamp + replay cache
- **Client generated time stamp has potential issues**
  - allows generating tokens for future use
  - clock skew

Mitigation: server-issued nonces

DPoP Summary

- **Pros**
  - does not need special infrastructure
  - can be implemented with widely available tooling (JWT, crypto etc.)

- **Cons**
  - new technology
  - requires code changes everywhere
    - proof token creation and validation
    - replay caches
    - nonce handling

Summary

- **Sometimes bearer tokens are not secure enough**
  - sensitive data
  - traversing untrusted networks
  - defense in depth needed

- **PoP increases security guarantees considerably**
  - does not come for free
  - infrastructure vs application complexity