

Introduction to SOA and Microservices Security

Oxford University
Software Engineering
Programme
April 2021



© Paul Fremantle 2016 except where credited elsewhere. This work is licensed under a Creative Commons Attribution-NonCommercial-ShareAlike 4.0 International License
See <http://creativecommons.org/licenses/by-nc-sa/4.0/>

Security Aims

- Confidentiality
- Integrity
- Availability
- Authenticity
- Non-Repudiation
- Authorization



© Paul Fremantle 2016 except where credited elsewhere. This work is licensed under a Creative Commons Attribution-NonCommercial-ShareAlike 4.0 International License
See <http://creativecommons.org/licenses/by-nc-sa/4.0/>

In Real Life

- The **opacity** of the envelope provides confidentiality
- The **seal** on the envelope provides integrity
- Royal Mail hopefully provides availability
- The **signature** provides authenticity
- The **proof of posting** and “**signed-for**” provide non-repudiation



None of these match up to the standards of electronic security



Confidentiality

- Provided by Encryption
- On the web, 99% of the time using TLS
- TLS is the successor to SSL
 - And often referred to as SSL!
- If you want to understand TLS well, this is about the best resource I've seen:
- <http://www.moserware.com/2009/06/first-few-milliseconds-of-https.html>



Encryption is pointless without authenticity/identity

- If I encrypt the message aimed at the wrong person, I have failed
- Encryption key distribution was the biggest issue until
 - 1973: Ellis, Cocks and Williamson of GCHQ created first “non-secret crypto”
 - Only made public in 1997
 - 1976: Diffie Helman key exchange
 - 1977: Rivest Shamir Adleman (RSA) public key cryptography



A simple analogy

- Padlocks which cannot be explored, re-engineered or re-used
- Keys which cannot be reverse-engineered into a padlock



Public Key Encryption

1. You send out as many padlocks as you like, but you keep the key secret



2. I lock the message in a box with YOUR padlock and send it to you



3. Only you have the key, so only you can read the message



Authenticity

- Need to Sign the message
- Now I send out lots of keys but keep the padlocks secret



- Anything I lock up in a box with my padlock must come from me
- Anyone can unlock it and verify its from me



Hash functions

- Take a message and create a fixed size result
 - E.g. 256 bits of data
- The aim of a hash function is to have no duplicate hits, and a random distribution of responses
 - The only way to find the input from the output is a dictionary attack



Signatures

- Actually the encrypted hash using the private key to encrypt
 - The validator decrypts the message
 - Applies the same hash function
 - Compares the two



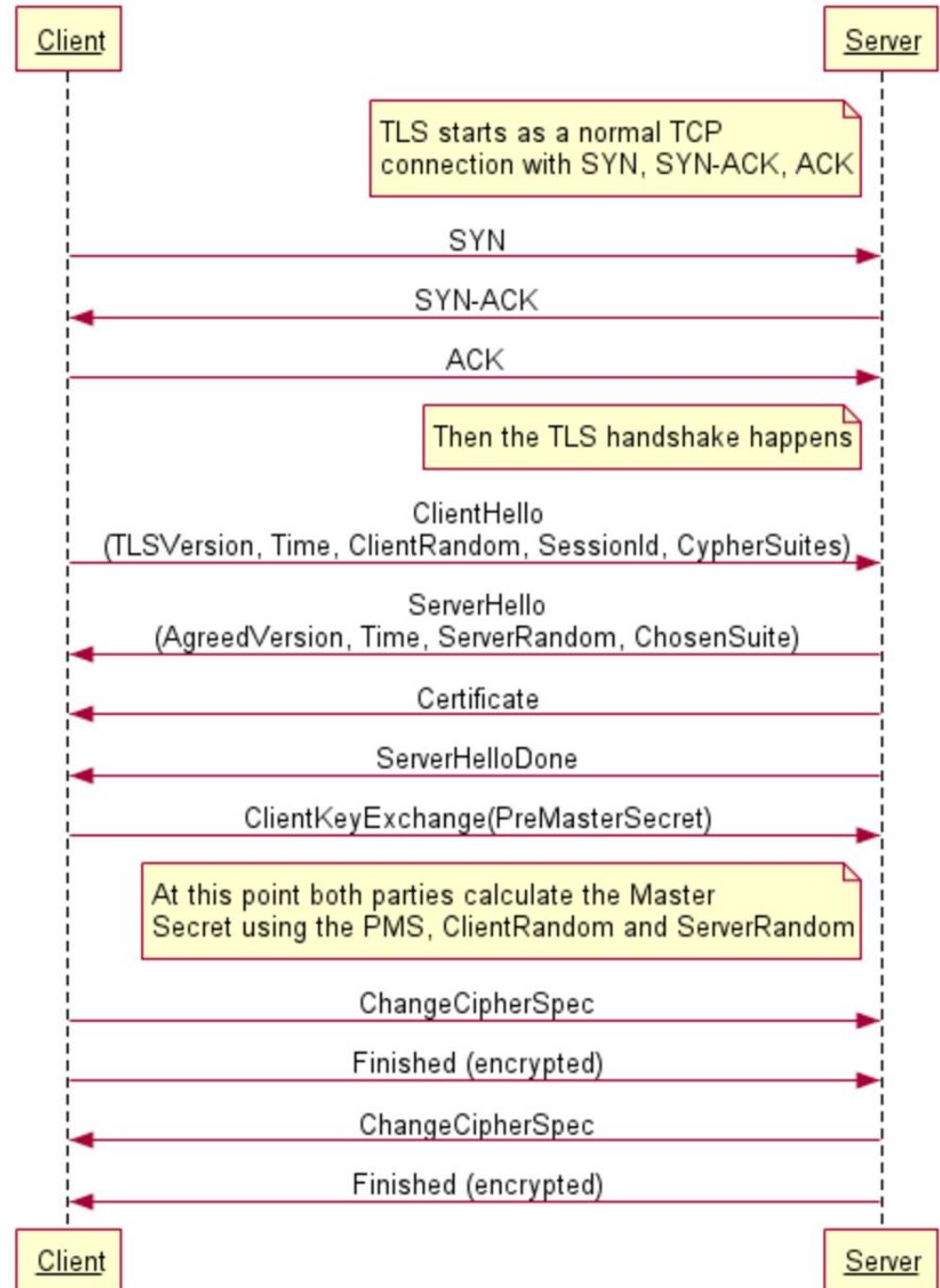
© Paul Fremantle 2016 except where credited elsewhere. This work is licensed under a Creative Commons Attribution-NonCommercial-ShareAlike 4.0 International License
See <http://creativecommons.org/licenses/by-nc-sa/4.0/>

Certificate Authority

- Instead of everybody needing to know about each other's keys, you have a trust hierarchy:
 - I create a key pair (key + padlock)
 - Export the public key (key)
 - Prove who I am to the CA
 - The CA “signs” the key
 - Locks my key in their padlock
 - Anyone who opens it knows that it is verified by them

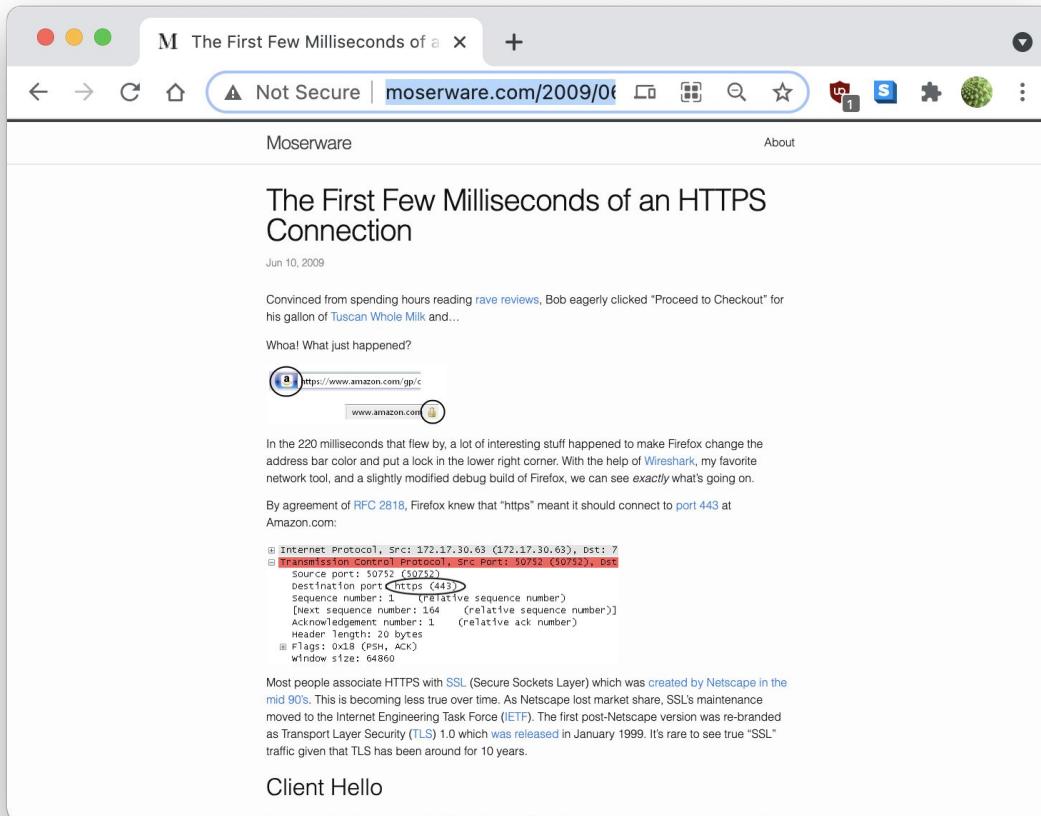


Transport Layer Security (TLS)



More details

<http://www.moserware.com/2009/06/first-few-milliseconds-of-https.html>
<https://tlseminar.github.io/first-few-milliseconds>



The screenshot shows a web browser window with the title "The First Few Milliseconds of a..." and the URL "moserware.com/2009/06/first-few-milliseconds-of-https.html". The page content is titled "The First Few Milliseconds of an HTTPS Connection" and includes a timestamp "Jun 10, 2009". The text discusses the process of establishing an HTTPS connection to Amazon. It includes a screenshot of the Firefox developer tools showing a network request to "https://www.amazon.com/gp/c" with a highlighted destination port of "443". Below this, there is a detailed description of the Client Hello message structure, listing fields like "Source port: 50752 (50752)", "Destination port: https (443)", and "Sequence number: 1 (relative sequence number)". The page also notes the historical context of SSL/TLS, mentioning its creation by Netscape in the mid 90s and its evolution into TLS 1.0.

The First Few Milliseconds of an HTTPS Connection

Jun 10, 2009

Convinced from spending hours reading [rave reviews](#), Bob eagerly clicked "Proceed to Checkout" for his gallon of *Tuscan Whole Milk* and...

Whoa! What just happened?



In the 220 milliseconds that flew by, a lot of interesting stuff happened to make Firefox change the address bar color and put a lock in the lower right corner. With the help of [Wireshark](#), my favorite network tool, and a slightly modified debug build of Firefox, we can see *exactly* what's going on.

By agreement of [RFC 2818](#), Firefox knew that "https" meant it should connect to [port 443](#) at Amazon.com:

```
Internet Protocol, Src: 172.17.30.63 (172.17.30.63), Dst: 7
Transmission Control Protocol, Src Port: 50752 (50752), Dst
Source port: 50752 (50752)
Destination port: https (443)
Sequence number: 1 (relative sequence number)
[Next sequence number: 164 (relative sequence number)]
Acknowledgment number: 1 (relative ack number)
Header length: 20 bytes
Flags: 0x18 (PSH, ACK)
Window size: 64860
```

Most people associate HTTPS with [SSL](#) (Secure Sockets Layer) which was created by [Netscape](#) in the mid 90s. This is becoming less true over time. As [Netscape](#) lost market share, SSL's maintenance moved to the Internet Engineering Task Force ([IETF](#)). The first post-Netscape version was re-branded as Transport Layer Security ([TLS](#)) 1.0 which was [released](#) in January 1999. It's rare to see true "SSL" traffic given that TLS has been around for 10 years.

Client Hello



LetsEncrypt

[Documentation](#)[Get Help](#)[Donate ▾](#)[About Us ▾](#)[Languages](#) ▾

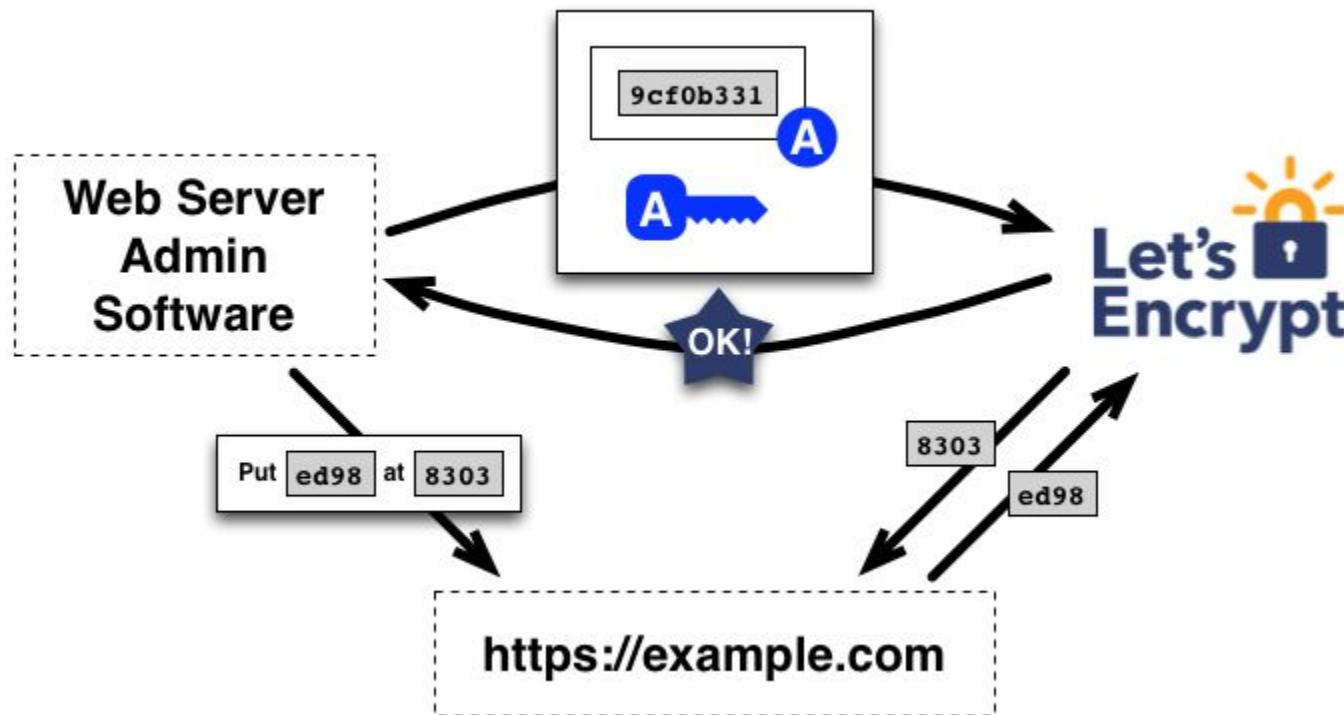
A nonprofit Certificate Authority providing TLS certificates to **240 million** websites.

Read our 2020 [Annual Report](#)

[Get Started](#)[Sponsor](#)

© Paul Fremantle 2016 except where credited elsewhere. This work is licensed under a Creative Commons Attribution-NonCommercial-ShareAlike 4.0 International License
See <http://creativecommons.org/licenses/by-nc-sa/4.0/>

Automatic Certificate Management Environment (ACME)



ACME

Internet Engineering Task Force (IETF)
Request for Comments: 8555
Category: Standards Track
ISSN: 2070-1721

R. Barnes
Cisco
J. Hoffman-Andrews
EFF
D. McCarney
Let's Encrypt
J. Kasten
University of Michigan
March 2019

Automatic Certificate Management Environment (ACME)

Abstract

Public Key Infrastructure using X.509 (PKIX) certificates are used for a number of purposes, the most significant of which is the authentication of domain names. Thus, certification authorities (CAs) in the Web PKI are trusted to verify that an applicant for a certificate legitimately represents the domain name(s) in the certificate. As of this writing, this verification is done through a collection of ad hoc mechanisms. This document describes a protocol that a CA and an applicant can use to automate the process of verification and certificate issuance. The protocol also provides facilities for other certificate management functions, such as certificate revocation.



Attribution-NonCommercial-ShareAlike 4.0 International License
See <http://creativecommons.org/licenses/by-nc-sa/4.0/>

Authentication with Client Certificates for humans

- Is rarely used
- Client needs a certificate
 - Hard to prove who you are to the CA
 - Implies costly
 - Some countries have implemented this nationally (e.g. Denmark)



More common authentication

- Server-side TLS
 - Every server is proven
- HTTP level authentication
 - Basic Auth
 - Digest
 - OAuth2 / OpenID Connect (OIDC)



© Paul Fremantle 2016 except where credited elsewhere. This work is licensed under a Creative Commons Attribution-NonCommercial-ShareAlike 4.0 International License
See <http://creativecommons.org/licenses/by-nc-sa/4.0/>

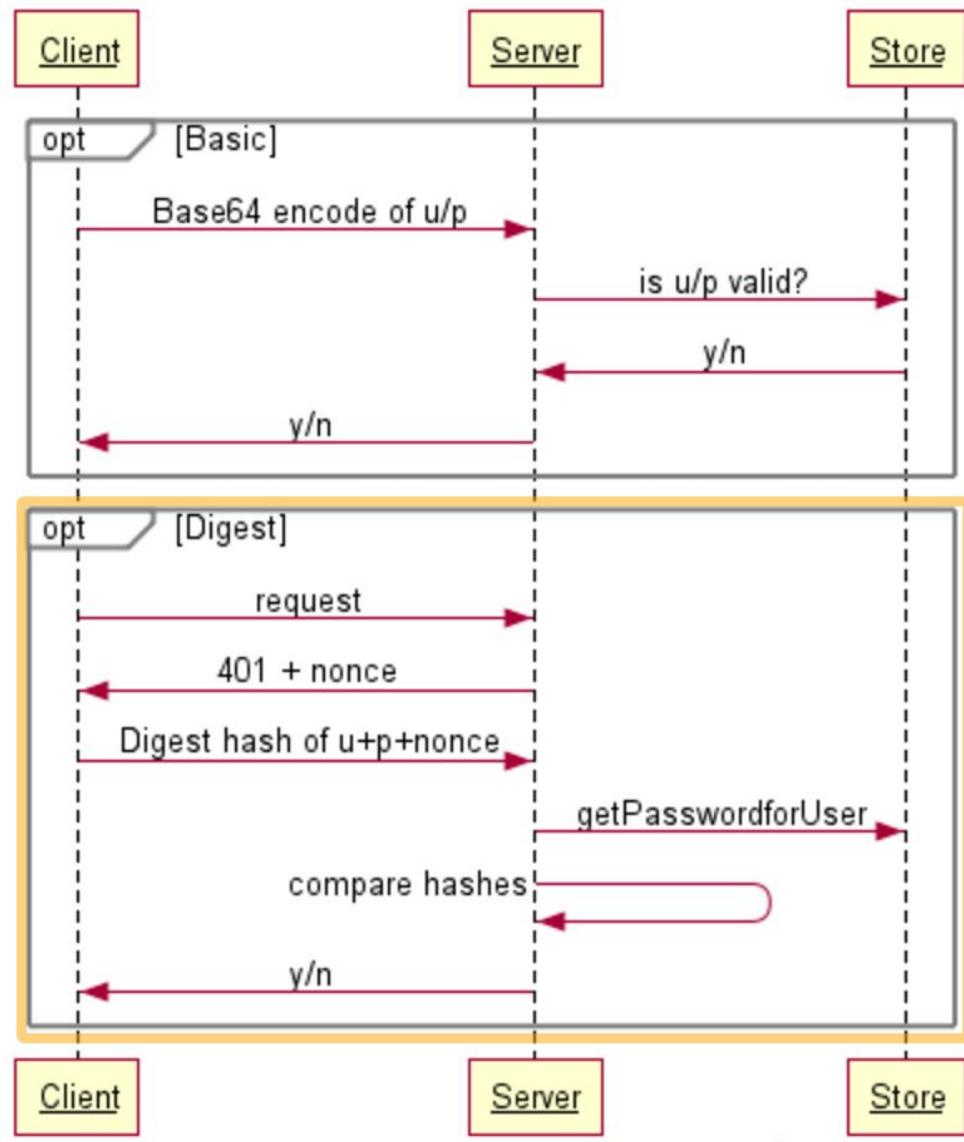
HTTP Authentication

- Basic Auth
 - Easy, fast, effective
 - Completely insecure except over HTTPS
 - base64 encoding
- Digest
 - Hash of the password
- More on OAuth later



© Paul Fremantle 2016 except where credited elsewhere. This work is licensed under a Creative Commons Attribution-NonCommercial-ShareAlike 4.0 International License
See <http://creativecommons.org/licenses/by-nc-sa/4.0/>

Basic vs Digest



Integrity

- Include a signed hash in with the message
- Only the sender could do this
- If the message is modified or manipulated the receiver will calculate a different hash to the sender
- This is used for both integrity AND authenticity
 - The signed hash is equivalent to signing the message



Non-repudiation

- This is done by passing back a signed hash of the signed hash!
- The receiver can prove that the sender sent the message (log the signed hash)
- The receiver then signs this and sends it back
- The sender logs this to prove that the receiver received the message



Availability



Performance

- Public Key Cryptography is based on mathematics of very large prime numbers
- The aim is that to break it will take many many years
 - Though Quantum Computing will change that
- It is slow
- Usually PKC is only used for exchange of a secret key that is then used for a session



The problem with passwords

cabbage

Sorry the password must be more than 8 chars
boiled cabbage

Sorry, you must have a numerical character

1 boiled cabbage

Sorry, no blank spaces

50frickingboiledcabbages

Sorry, at least one upper case

50FRICKINGboiledcabbages

Sorry, the password cannot have more than one upper case consecutively

50FrickingBoiledCabbagesShovedSomewhereIfYouDon'tGiveMeAccess

Sorry, no punctuation



Adobe - Create an account X

<https://www.adobe.com/account/sign-in.adobedotcom.html?returnURL=https://w...>

Adobe

My Adobe account

Use your Adobe ID to download free trials, buy products, manage orders, and access online services such as Adobe® Creative Cloud™ and Acrobat.com. Plus, be a part of the thriving Adobe online community.

Create an Adobe ID

[I already have an Adobe ID](#)

Adobe ID (Email Address) jdoe@domain.com	First Name <input type="text"/>
Password <input type="password"/>	Last Name <input type="text"/>
Retype Password <input type="password"/>	Country/Region United Kingdom

Stay informed via email about Adobe products and services. [Learn more.](#)

I have read and agree to the Adobe [Terms of Use](#) and [Privacy Policy](#).

Create



BBC News - Adobe hack: A X

www.bbc.co.uk/news/technology-24740873

BBC Sign in News Sport Weather iPlayer TV Radio More... Search

NEWS TECHNOLOGY

Home | World | UK | England | N. Ireland | Scotland | Wales | Business | Politics | Health | Education | Sci/Environment | Technology | Entertainment & Arts

30 October 2013 Last updated at 11:43

Share

Adobe hack: At least 38 million accounts breached

Adobe has confirmed that a recent cyber-attack compromised many more customer accounts than first reported.

The software-maker said that it now believed usernames and encrypted passwords had been stolen from about 38 million of its active users.

It added that the attackers had also accessed details from an unspecified number of accounts that had been unused for two or more years.

The firm had originally said 2.9 million accounts had been affected.

Adobe has also announced that the hackers stole parts of the source code to Photoshop, its popular picture-editing program.

It had previously revealed that the source code for its Acrobat PDF document-editing software and ColdFusion web-application creation

Adobe Photoshop CS6

Adobe said source code for Photoshop had been stolen

Related Stories

Top Stories

Ukraine warns of Russia 'aggression'

Standard Life quit plan sparks row

Thousands in Sir Tom Finney tribute

Parties 'knew about On the Runs'

Merkel urges 'strong UK' in EU

Features

Looking back

The time when pro-paedo campaigners operated op

Tide of desperation

Picture captures scale of crowd of refugees in Syri

Size matters

Is Gravity too short to wi Picture?



Магазин Аккаунтов #1 в Р

<https://buyaccs.com>

BuyAccs.com

СЕРВИС РЕГИСТРАЦИИ АККАУНТОВ

[Russian Version](#) [English Version](#)

Наш магазин аккаунтов рад предложить аккаунты различных **почтовых служб и бесплатных хостингов** для любых задач. Вы получаете аккаунты **СРАЗУ** после оплаты заказа. Мы принимаем **Webmoney, Perfectmoney и Paypal**.

При покупке аккаунтов менее 1000 штук действует специальный тариф.

[www.FreedomScripts.org - боты для всех соц. сетей](#)
[Mera Софт для дровеев - Zerber](#)

[Twidium - профессиональный инструмент для раскрутки твиттера](#)

[Заработай на продаже аккаунтов](#)

[Купить аккаунты Одноклассников](#)
[Купить аккаунты Вконтакте](#)
[Купить аккаунты Мамба](#)

Сейчас в продаже

Служба	Кол-во акков	Цена за 1К аккаунтов
Mail.ru	34450	до 10K: \$5 от 10K до 20K: \$4.5 от 20K: \$4
Mail.ru Human	13689	до 10K: \$6 от 10K до 20K: \$5.5 от 20K: \$5
Mail.ru Mix	97070	до 10K: \$5 от 10K до 20K: \$4.5 от 20K: \$4
Mail.ru Second Hand	28103	до 10K: \$3 от 10K до 20K: \$2.5 от 20K: \$2
Mail.ru Mix S/H	82526	до 10K: \$3 от 10K до 20K: \$2.5 от 20K: \$2
Yandex.ru	41180	до 10K: \$10 от 10K до 20K: \$9 от 20K: \$8
Rambler.ru	707	до 10K: \$15 от 10K до 20K: \$15 от 20K: \$15
Qip.ru	142	до 10K: \$30 от 10K до 20K: \$30 от 20K: \$30
Hotmail.com	168778	до 10K: \$4 от 10K до 20K: \$3.5 от 20K: \$3
Hotmail.com Plus	159426	до 10K: \$5 от 10K до 20K: \$4.5 от 20K: \$4

Новости

15 Фев 2014
Внимание! С 16 февраля по 2 марта магазин работает в **автоматическом режиме!** Служба поддержки будет работать **один раз в день** в вечернее время. Убедительная просьба отнести с пониманием, максимально четко **формулировать суть проблем**, если таковые будут возникать, а также **не ждать мгновенных ответов**.

03 Фев 2014
 Теперь мы принимаем **Яндекс Деньги** и **Qiwi** через Робокассу.

31 Янв 2014
 Добавлены аккаунты **Яндекс Диск**.

23 Янв 2014
 Добавлены аккаунты **Pinterest.com** по отличной цене - всего **\$70** за **1000 шт.**

15 Янв 2014
 Добавлены аккаунты **Google.com Multi** - самые стабильные аккаунты **Google** на сегодняшний день.

© Paul Fremantle 2016 except where credited elsewhere. This work is licensed under a Creative Commons Attribution-NonCommercial-ShareAlike 4.0 International License
 See <http://creativecommons.org/licenses/by-nc-sa/4.0/>

Third hit on Google

“how to add authentication php”

```
CREATE DATABASE todo;

USE todo;

CREATE TABLE IF NOT EXISTS `users` (
  `user_id` bigint(20) unsigned NOT NULL auto_increment,
  `user_login` varchar(100) NOT NULL,
  `user_password` varchar(64) NOT NULL,
  `user_firstname` varchar(50) NOT NULL,
  `user_surname` varchar(50) NOT NULL,
  `user_email` varchar(100) NOT NULL,
  `user_registered` datetime NOT NULL default '0000-00-00 00:00:00',
  PRIMARY KEY (`user_id`),
  KEY `idx_user_login_key` (`user_login`)
) DEFAULT CHARSET=utf8 AUTO_INCREMENT=1;

CREATE TABLE IF NOT EXISTS `tasks` (
  `task_id` bigint(20) unsigned NOT NULL auto_increment,
  `user_id` bigint(20) NOT NULL REFERENCES `users`(`user_id`),
  `task_name` varchar(60) NOT NULL,
  `task_priority` tinyint(2) NOT NULL default '2',
  `task_color` varchar(7) NOT NULL default '#ffffff',
  `task_description` varchar(150) NULL,
  `task_attendees` varchar(4000) NULL,
  `task_date` datetime NOT NULL default '0000-00-00 00:00:00',
  PRIMARY KEY (`task_id`),
  KEY `idx_task_name_key` (`task_name`)
) DEFAULT CHARSET=utf8 AUTO_INCREMENT=1;
```





Hotel Token

An OAuth 2 access token is like a hotel-room key card.

It gives access, all by itself without further checking, to a particular resource (in this case, room 238 at the Omni Interlocken in Denver.) *Check.*

It's issued to a particular person, who has to be authenticated first (like by showing my driver's license at the check-in.) *Check.*

Nothing on the outside tells you who it's been issued to or what it's for. *Check.*

It's not obscured or encrypted, so you have to take good care of it (if a bad guy got it and knew what it was for, he could get into my hotel room and rob me blind.) *Check.*

You can give it to someone else and have them access the resource for you (like giving a colleague the card and asking them to go up to your room and get the VGA dongle that you stupidly left on the desk.) *Check.*

If you lose it, you can go back to the issuer and get another one which is functionally identical (somehow it wasn't there when you got back from the bar, but the front desk can get you another, assuming you have your wallet and ID.) *Check.*

It expires after a while. (I gave it back to the front desk when I left because I knew it wouldn't be useful any more.) *Check.*



Search

ongoing

What this is ·

Truth · **Biz** · **Tech**

author · **Dad** · **software** ·

colophon · **rights**

May 24, 2013

· **Technology** (76 fragments)

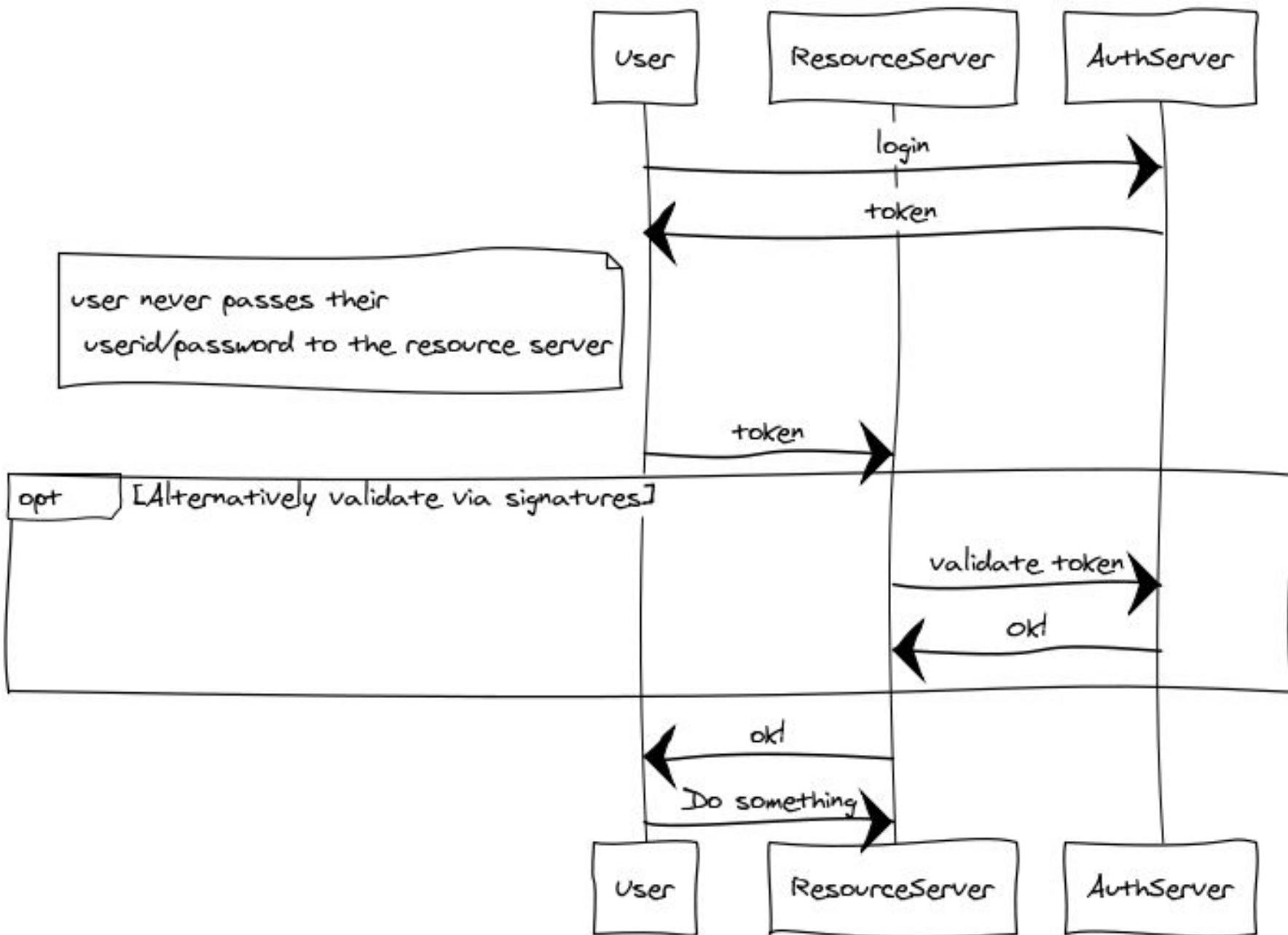
· · **Identity** (39 more)

By **Tim Bray**.

I work for Google, but the opinions expressed here are my own, and no other party necessarily agrees with them.

A full disclosure of my professional interests is on the **author** page.

How Tokens Work



Tokens

- Kerberos Tickets
 - Developed by MIT in the late 80's onwards
 - Designed to allow lots of campus machines to be secured easily (without having local UNIX password files!)
 - Based on Needham-Schroder
- SAML/SAML2
 - An XML version that has become a popular way of doing Single Sign On for the Web
 - Used by Google Apps/Shibboleth/etc

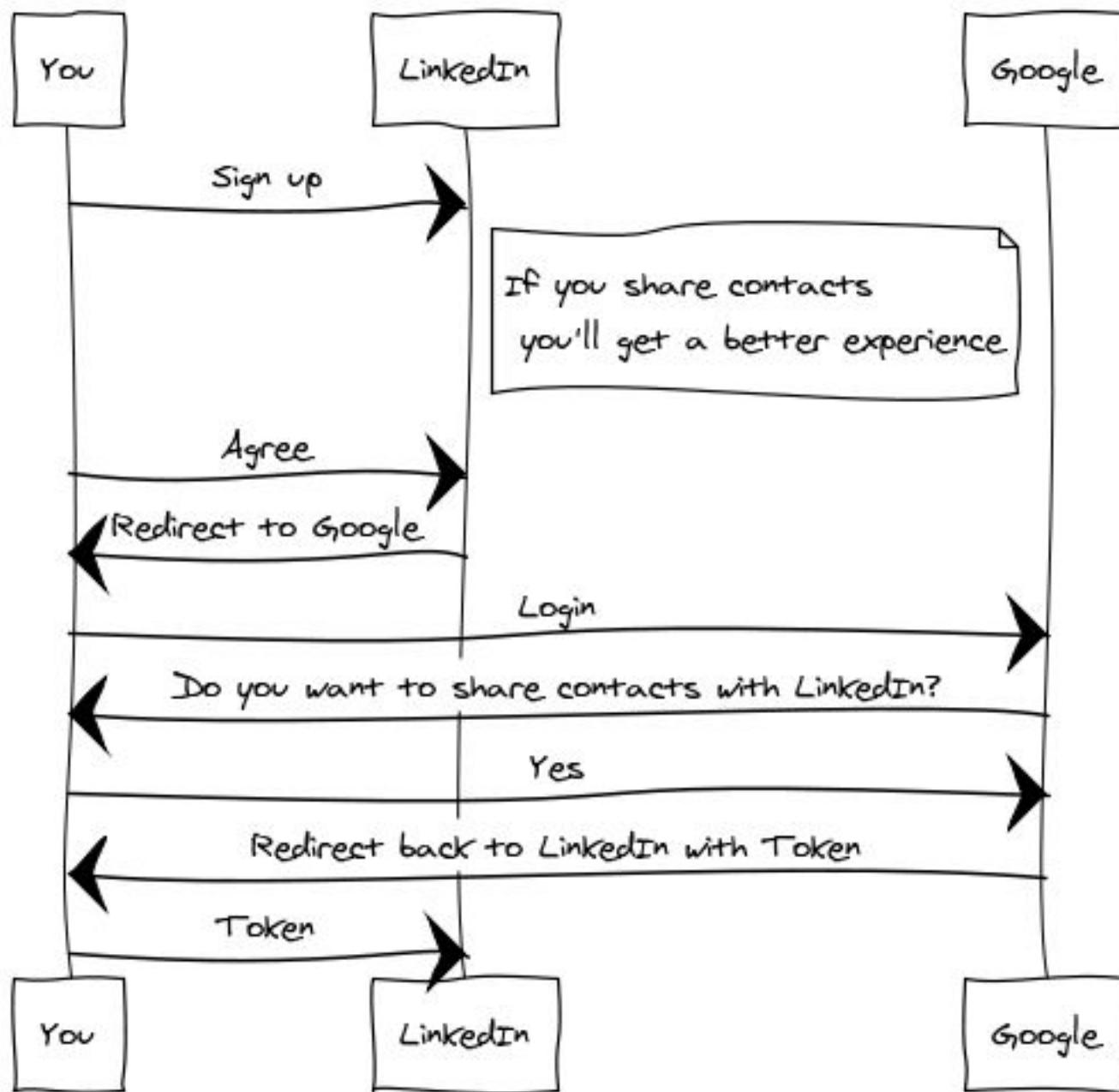


LinkedIn example

- LinkedIn used to ask for your Gmail userid and password
- They said they would only use this to get your contact list for certain purposes and would delete afterwards
- What do you think?



The "LinkedIn" Scenario



OAuth Terminology

- Resource Owner
 - The end user or logical owner of the resource
 - *The gmail user*
- Resource Server
 - The server that contains or controls access to the resource
 - *The contact list*
- Client
 - The system looking for access to the resource
 - *LinkedIn*
- Authorization Server
 - The system that maintains the user identity and issuing tokens
 - *Google identity*

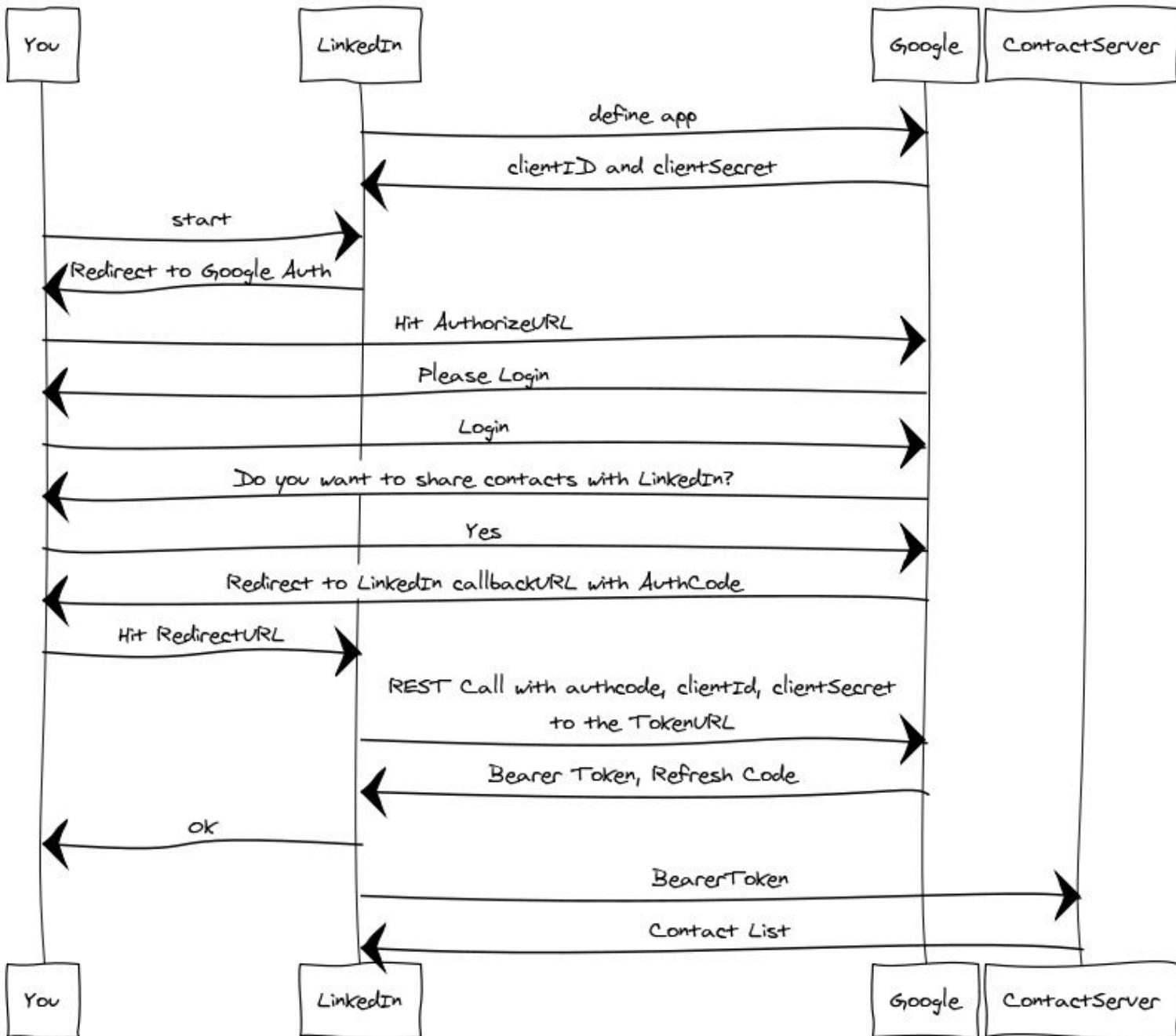


The more detailed version!

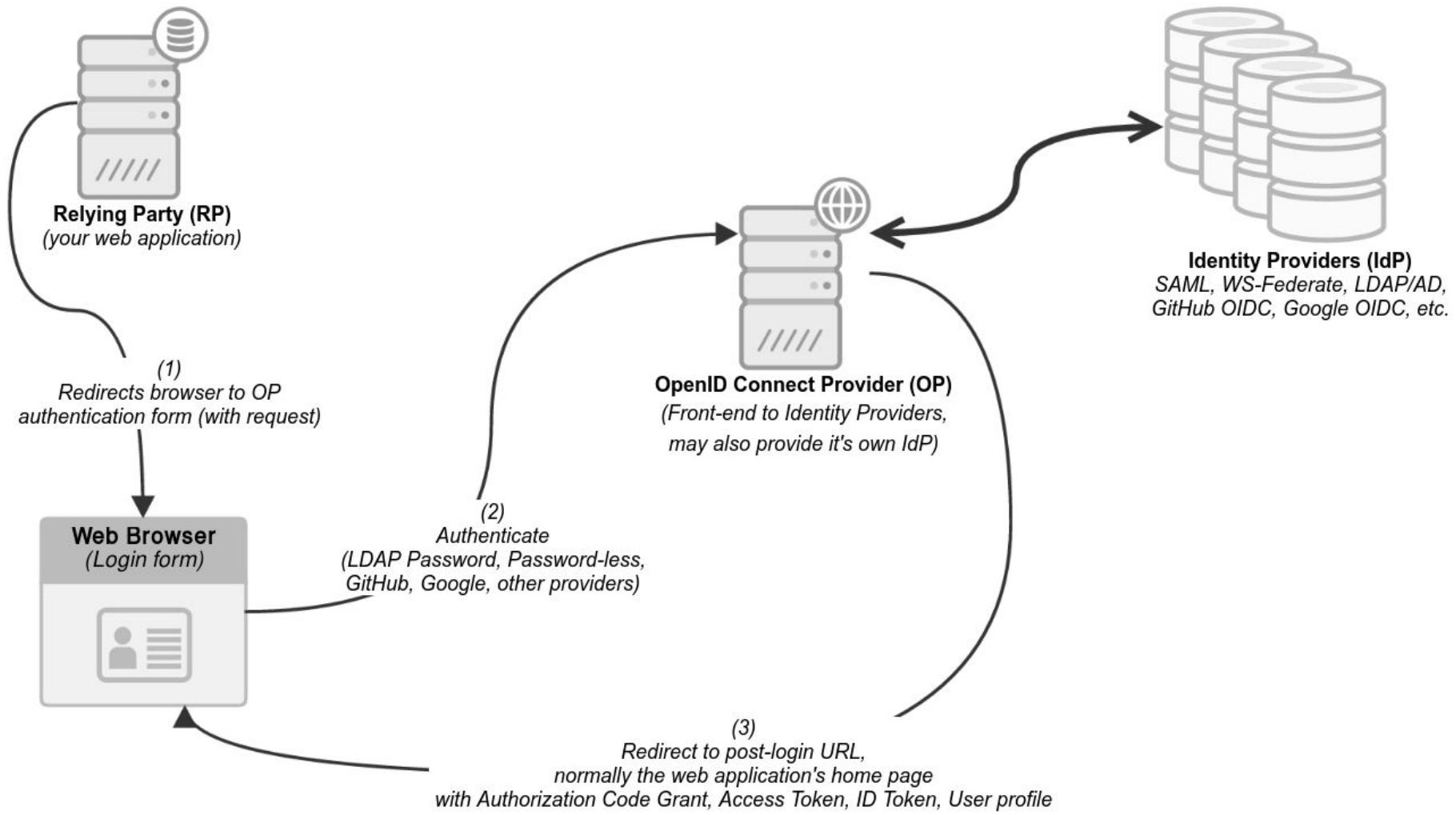


© Paul Fremantle 2016 except where credited elsewhere. This work is licensed under a Creative Commons Attribution-NonCommercial-ShareAlike 4.0 International License
See <http://creativecommons.org/licenses/by-nc-sa/4.0/>

The "LinkedIn" Scenario the details



OpenID Connect (OIDC)



Open Source OIDC systems

- Keycloak
 - Fully featured OIDC/OAuth2/SAML server from Redhat
- WSO2 Identity Server
 - OAuth2/OIDC/SAML IdP and broker
- dex
 - A Kubernetes OIDC provider that only does federated login



Criticisms of OAuth2

- <http://hueniverse.com/2012/07/oauth-2-0-and-the-road-to-hell/>

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

Main concern:

Bearer tokens+TLS vs client signature



A retrospective reasoning for OAuth

- Two significant factors:
 - Billions of users do not fit well into traditional hierarchical models
 - Graph-based, user directed approaches work better – Facebook Friends, Google Circles
 - The Web and REST architecture inherently promotes linked websites, ecosystems, federation
- At the same time the web protocols make this easier to do in a standard, interoperable way



JSON Web Tokens

Encoded

PASTE A TOKEN HERE

```
eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJ  
zdWIiOiIxMjM0NTY3ODkwIiwibmFtZSI6Ikpvag4  
gRG9lIiwiaWF0IjoxNTE2MjM5MDIyfQ.Sf1KxwRJ  
SMeKKF2QT4fwpMeJf36P0k6yJV_adQssw5c
```

Decoded

EDIT THE PAYLOAD AND SECRET

HEADER: ALGORITHM & TOKEN TYPE

```
{  
  "alg": "HS256",  
  "typ": "JWT"  
}
```

PAYOUT: DATA

```
{  
  "sub": "1234567890",  
  "name": "John Doe",  
  "iat": 1516239022  
}
```

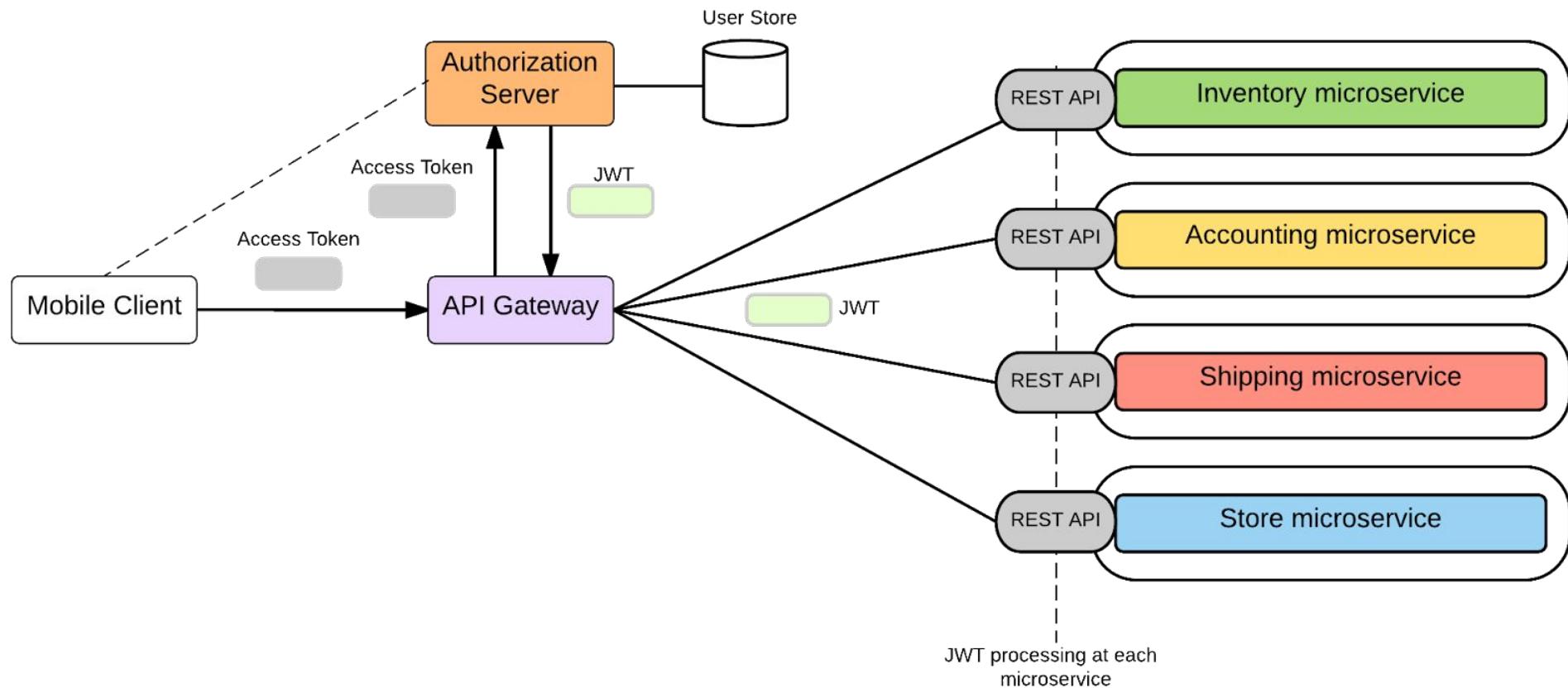
VERIFY SIGNATURE

```
HMACSHA256(  
  base64UrlEncode(header) + "." +  
  base64UrlEncode(payload),  
  your-256-bit-secret  
)  secret base64 encoded
```



© Paul Fremantle 2016 except where credited elsewhere. This work is licensed under a Creative Commons Attribution-NonCommercial-ShareAlike 4.0 International License
See <http://creativecommons.org/licenses/by-nc-sa/4.0/>

JWT in Microservices



mTLS

mutual TLS is where both sides have a certificate

- Has come back into fashion for microservices
- Built in support from Service Meshes



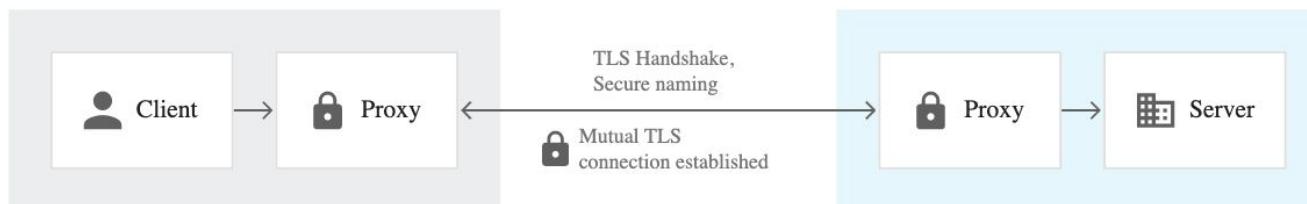
mTLS - mutual TLS by Proxy

Anthos Service Mesh by example: mTLS

[Send feedback](#)

In Anthos Service Mesh 1.5 and later, auto [mutual TLS](#) (auto mTLS) is enabled by default. With auto mTLS, a client sidecar proxy automatically detects if the server has a sidecar. The client sidecar sends mTLS to workloads with sidecars and sends plaintext to workloads without sidecars. Note, however, services accept both plaintext and mTLS traffic. As you [inject sidecar proxies](#) to your Pods, we recommend that you also configure your services to only accept mTLS traffic.

With Anthos Service Mesh, you can enforce mTLS, outside of your application code, by applying a single YAML file. Anthos Service Mesh gives you the flexibility to apply an authentication policy to the entire service mesh, to a namespace, or to an individual workload.



An interesting thread on mTLS

<https://news.ycombinator.com/item?id=25380301>

A screenshot of a web browser window showing a Hacker News post. The browser has multiple tabs open, including 'presentation-source - Google Sheets', '10-security.pptx - Google Slides', and 'My feeling is that Malone h...'. The main content area shows a post by user 'tptacek' from 4 months ago. The post discusses mTLS and its relationship to secure channels and request smuggling attacks. It includes two sections of text: 'First:' and 'Second:'. The post has a 'parent' link and a 'favorite' link.

▲ tptacek 4 months ago | parent | favorite | on: In Defense of Mutual TLS

My feeling is that Malone has misconstrued a bunch of Colm MacCárthaigh's arguments. I will now do the same thing, by attempting to summarize MacCárthaigh's arguments myself.

First: TLS (and mTLS) create secure channels. A channel bears many requests and responses. In many designs, a single channel will bear requests for many different users. The job of a textbook secure channel is to prevent an attacker with control of the underlying network from tampering with requests and responses. Binding identities and authorization claims to requests is a different job. The textbook solution to that problem is authenticated requests, not authenticated secure channels. I wouldn't have used the word "layering violation", which I would contend Is Not A Thing, but the point MacCárthaigh makes is important. So: when MacCárthaigh says SQLI and Request Smuggling are "still things", he's not saying that mTLS introduces SQLI, but rather that authenticating at the level of secure channels means that a request smuggling attack is almost automatically a game-over for your application, because you aren't authenticating the requests independently of the channel. Even Basic Auth can potentially avoid that problem, if an attacker has to know a secret to slip into a request in order to forge a request on a compromised channel.

Second: TLS was designed for the WebPKI, and that's its most important application. A consequence of this is that a lot of TLS



Welcome to cert-manager | cert-manager

cert-manager.io/docs/

Apps Inbox (40,823) - p... Fremantle Family... Weaveworks – Cal... Other Bookmarks Reading List

cert-manager

Welcome to cert-manager

cert-manager is a native [Kubernetes](#) certificate management controller. It can help with issuing certificates from a variety of sources, such as [Let's Encrypt](#), [HashiCorp Vault](#), [Venafi](#), a simple signing key pair, or self signed.

It will ensure certificates are valid and up to date, and attempt to renew certificates at a configured time before expiry.

It is loosely based upon the work of [kube-lego](#) and has borrowed some wisdom from other similar projects such as [kube-cert-manager](#).

```
graph TD; subgraph Issuers [Issuers]; I1[letsencrypt-staging]; I2[letsencrypt-prod]; I3[vault-prod]; I4[venafi-tpp]; end; subgraph Certificates [Certificates]; C1[foo.bar.com  
Issuer: venafi-tpp]; C2[example.com  
www.example.com  
Issuer: letsencrypt-prod]; end; subgraph KubernetesSecrets [Kubernetes Secrets]; SK1[Signed keypair]; SK2[Signed keypair]; end; Issuers --> cert-manager; Certificates --> KubernetesSecrets;
```



Access Control

- Traditionally Authorization has been
 - Role-based
 - If I am a manager then I can look at salaries
 - Based only on user
 - Hard-coded
 - Authorization rules encapsulated in code and applications



Problems with RBAC

- Problems with this are:
 - Doesn't correctly model the real world
 - I should only be able to see my team's salaries, and only while participating in the salary review process
 - Hard to evolve
 - Hard to manage compliance
 - How do I find out who can make a trade of \$30m?



Policy Based Access Control

- Delegate access control decisions to a “Policy Decision Point”
- Utilise more information in the decision
- Externalise the decision from the code

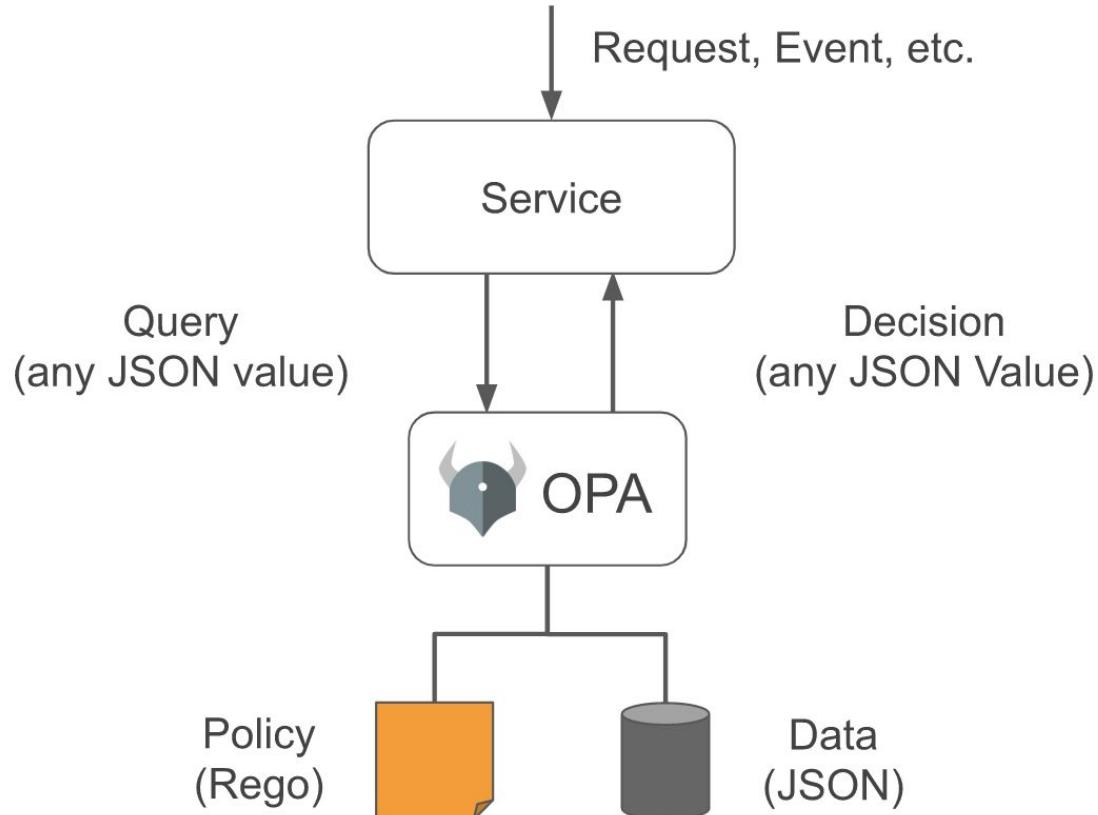


XACML 3.0

- An XML language for capturing authorization and entitlement
- Together with a powerful model
- Terminology
 - Policy Decision Point (PDP)
 - Policy Enforcement Point (PEP)
 - Policy Administration Point (PAP)
 - Policy Information Point (PIP)



Open Policy Agent (OPA)



Policy Decoupling



© Paul Fremantle 2016 except where credited elsewhere. This work is licensed under a Creative Commons Attribution-NonCommercial-ShareAlike 4.0 International License
See <http://creativecommons.org/licenses/by-nc-sa/4.0/>

OPA in Kubernetes



OPA for HTTP APIs

```
authz.rego — ~/src/policies/httpapi/acmecorp

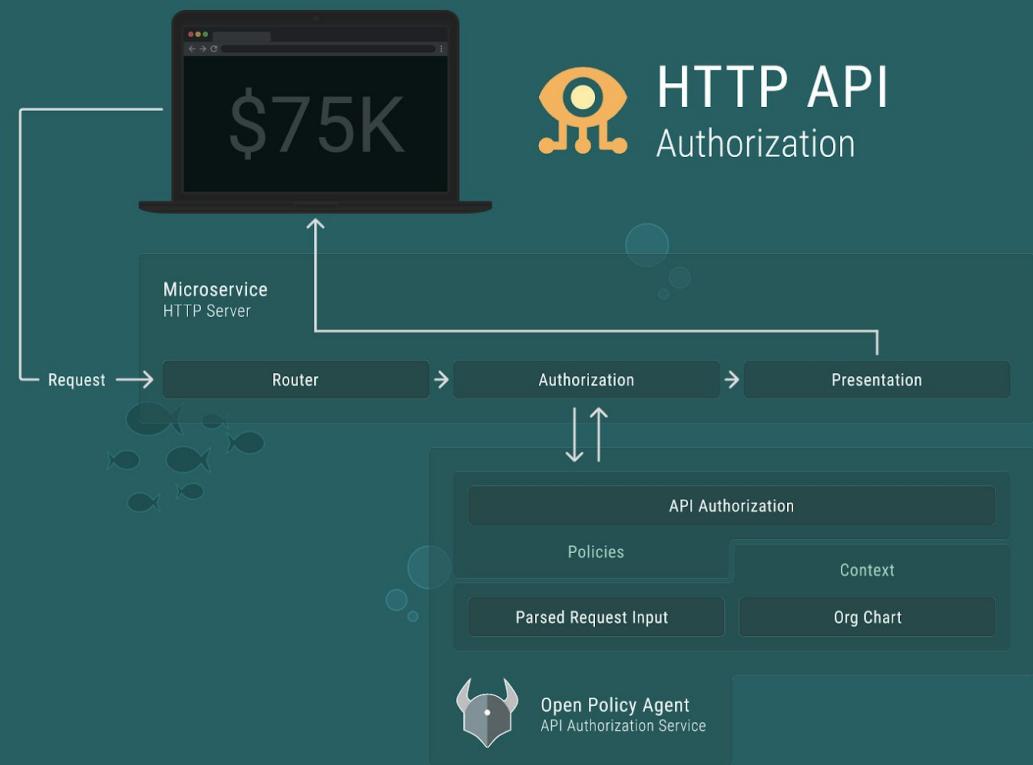
# HTTP API Authorization

package acmecorp.authz

default allow = false

# Allow people to read their own salaries.
allow {
    input.method = "GET"
    input.path = ["salaries", employee_id]
    input.user = employee_id
}

# Also allow managers to read the salaries of people they manage.
allow {
    input.method = "GET"
    input.path = ["salaries", employee_id]
    input.user = data.manager_of[employee_id]
}
```



Resources

- Applied Cryptography, Second Edition, Bruce Schneier, John Wiley & Sons, 1996, ISBN 0-471-11709-9
- Web Services Security, Mark O'Neill, 2003, ISBN 0072224711
- Google (sorry but there are too many links!)



© Paul Fremantle 2016 except where credited elsewhere. This work is licensed under a Creative Commons Attribution-NonCommercial-ShareAlike 4.0 International License
See <http://creativecommons.org/licenses/by-nc-sa/4.0/>