Introduction to SOA Security

Oxford University
Software Engineering
Programme
June 2016

Security Aims

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

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

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

Authentication with Certificates

- 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)

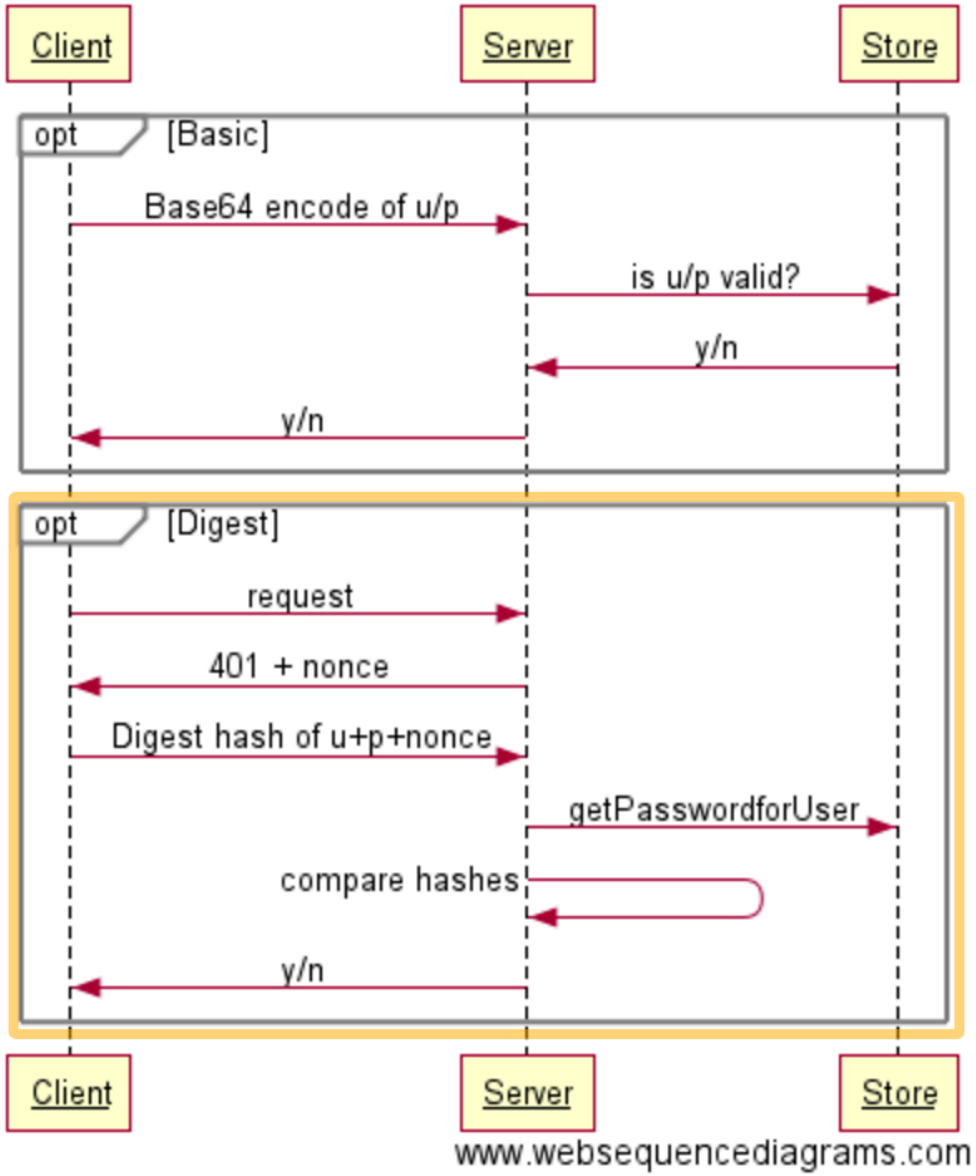
More common authentication

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

HTTP Authentication

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

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

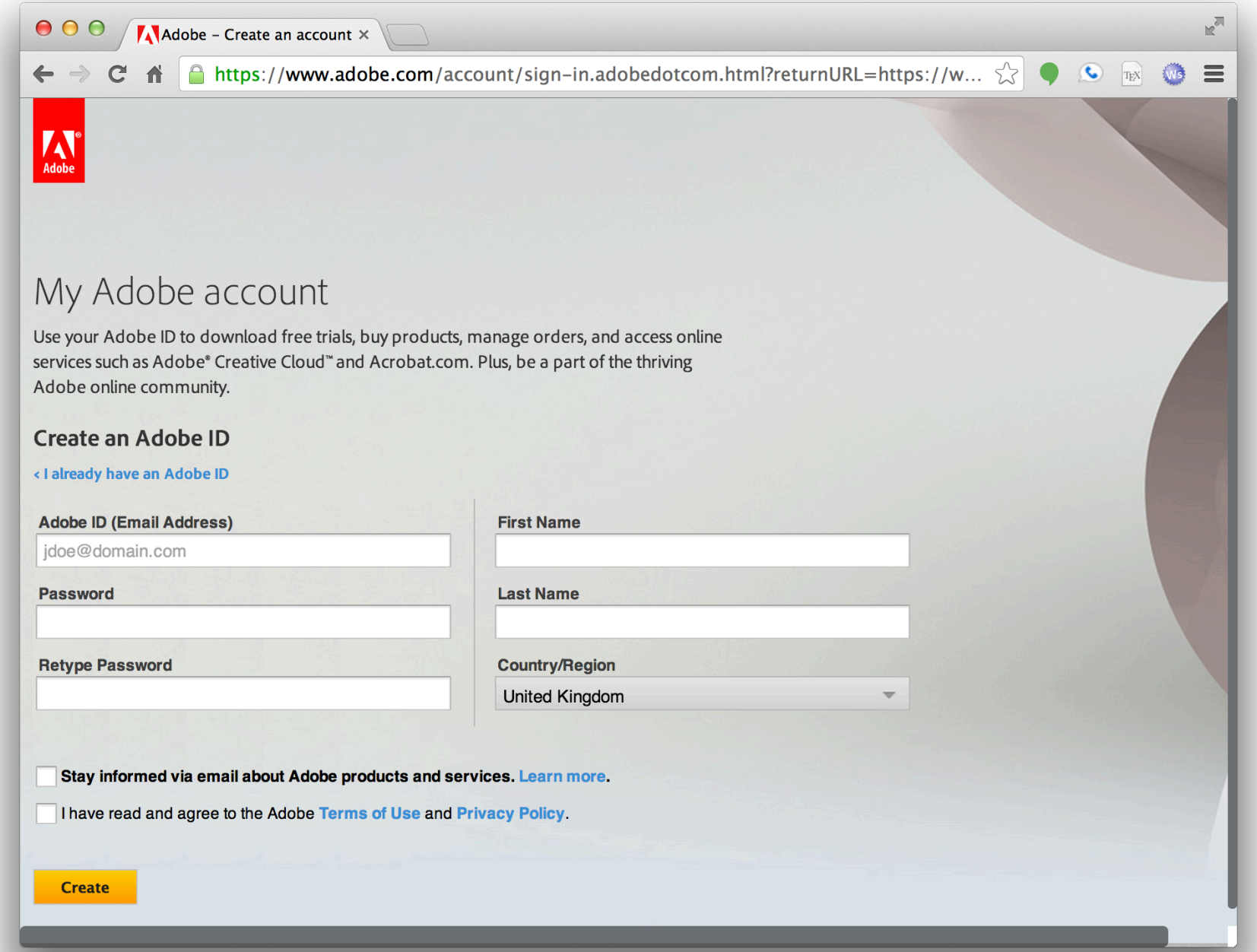
- Public Key Cryptography is based on mathematics of very large prime numbers
- The aim is that to break it will take many many years
  - Though Quantum Computing will change that
- It is slow
- Usually PKC is only used for exchange of a secret key that is then used for a session

TLS

- 99% of time used with only Server-Side authenticity
  - Client validates the cert, and then authenticates via another model (e.g. Basic Auth, OAuth2, SAML2, etc)
- Mutual SSL/TLS is where both sides authenticate
  - Most usually used in server-server communications

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't GiveMeAccess
Sorry, no punctuation



BBC News - Adobe hack: A

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.

Adobe said source code for Photoshop had been stolen

It had previously revealed that the source code for its Acrobat PDF

Related Stories

Top Stories

Ukraine warns of Russia 'aggressio

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

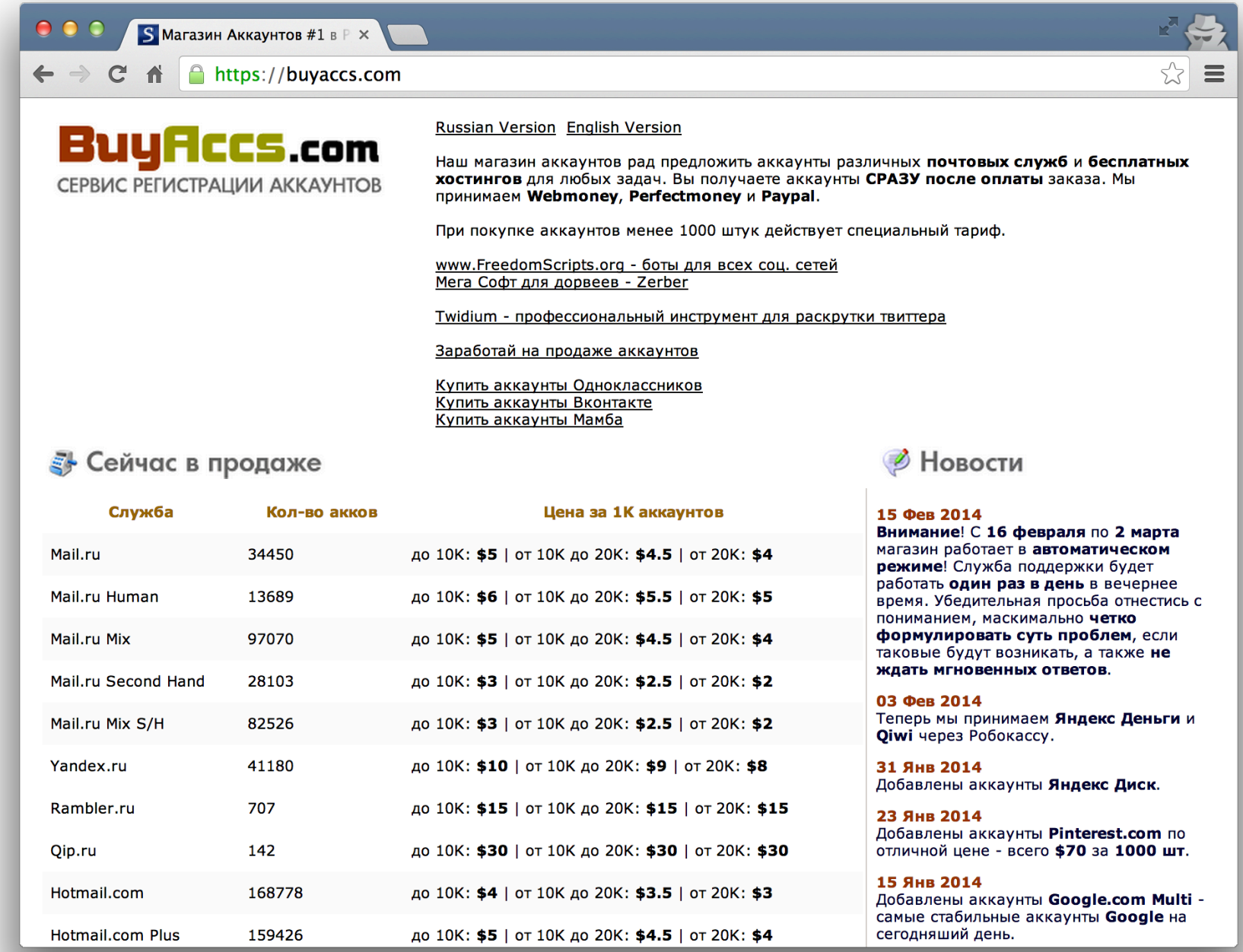
The time when pro-paedo campaigners operated op

Tide of desperation

Picture captures scale of crowd of refugees in Syri

Size matters

Is Gravity too short to wi Picture?



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;
```


http://www.developerdrive.com/2013/05/creating-a-simple-to-do-application-%E2%80%93-part-3

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*.

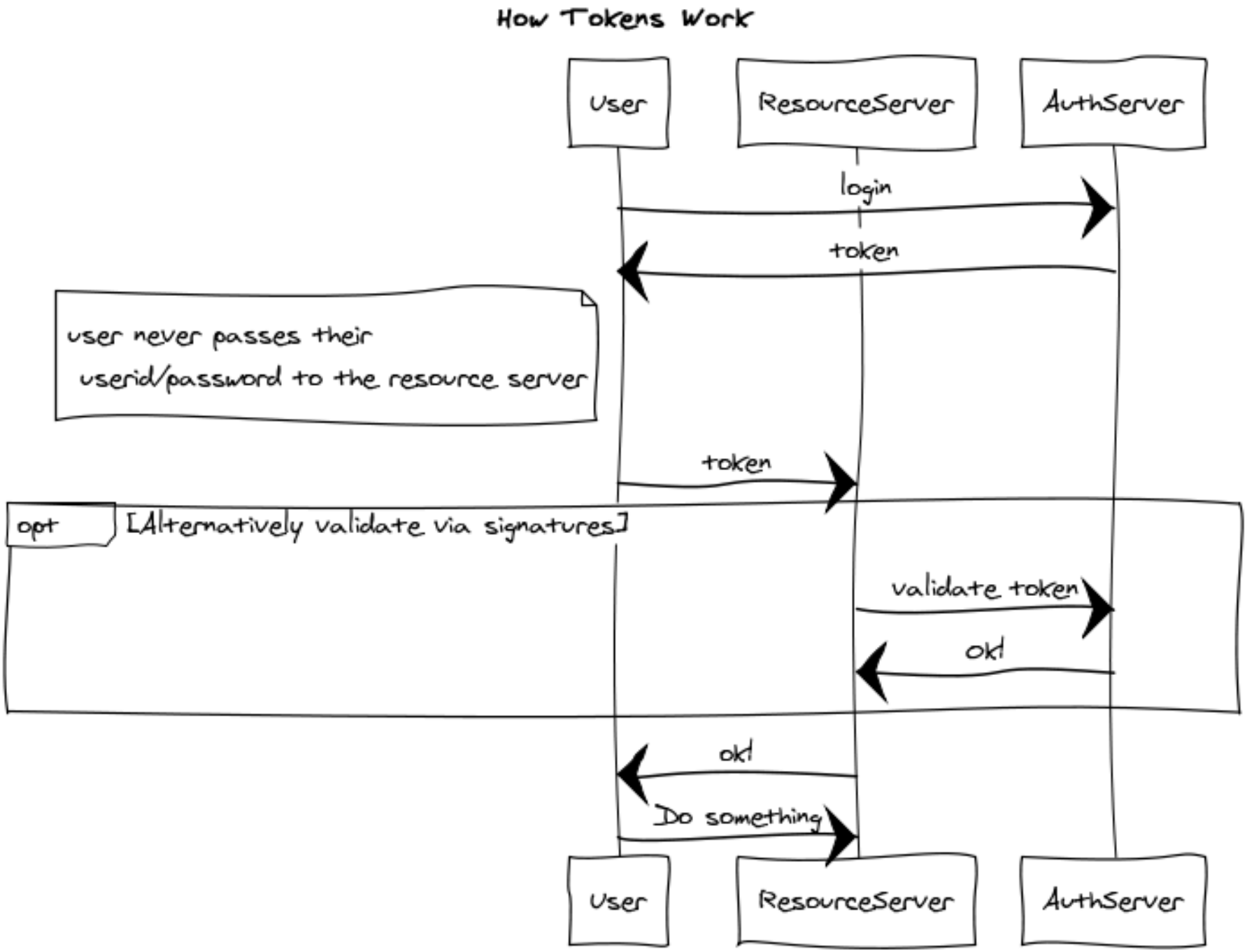
ongoing
What this is ·
Truth · **Biz** · **Tech**
author · **Dad** · **software** ·
colophon · **rights**

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
User
ResourceServer
AuthServer
login
token
user never passes their
userid/password to the resource server
token
opt
[Alternatively validate via signatures]
validate token
ok!
ok!
Do something
User
ResourceServer
AuthServer
www.websequencediagrams.com

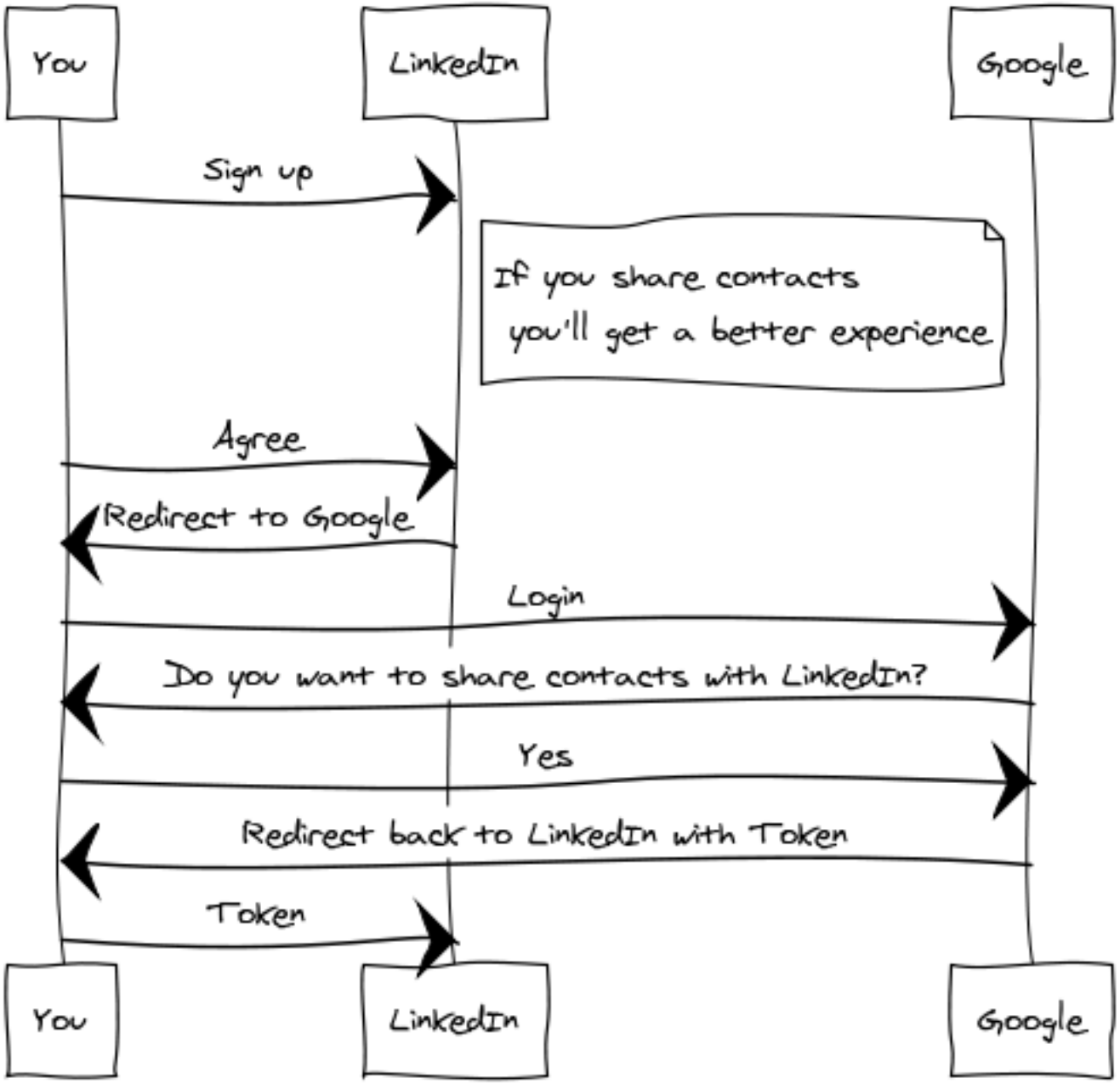
Tokens

- 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

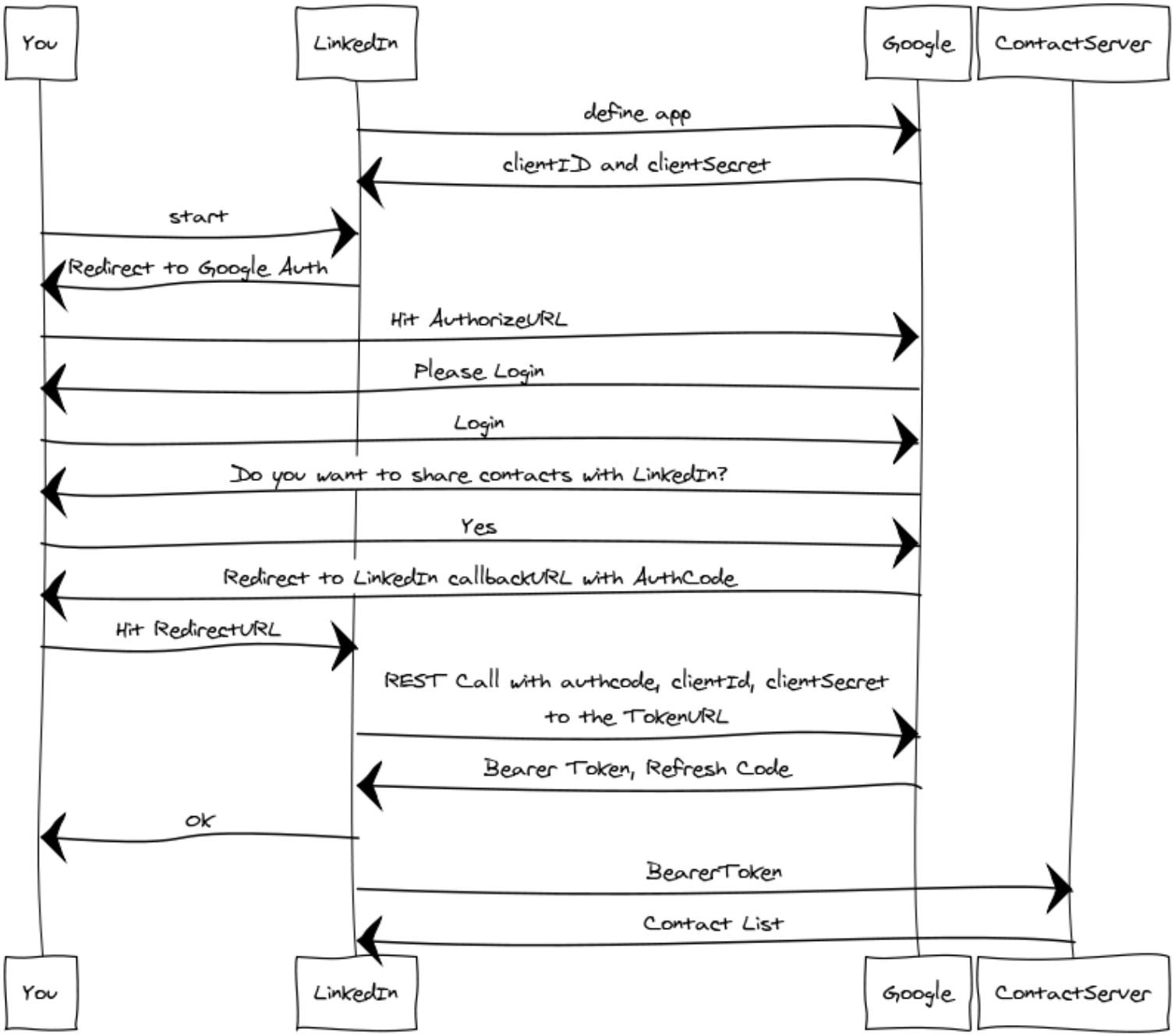


OAuth Terminology

- 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!

The "LinkedIn" Scenario the details



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

Web Services Security

Much more powerful but very complex

- WS-Security
  - Provides Message level encryption, signature and authentication
- WS-SecurityPolicy
  - Allows services to publish their security requirements
- WS-SecureConversation
  - Greater efficiency by bootstrapping fast cryptography for a session
- WS-Trust
  - Token-based security (inc Kerberos and SAML)
- WS-Federation
  - Federated security using WS-Trust

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

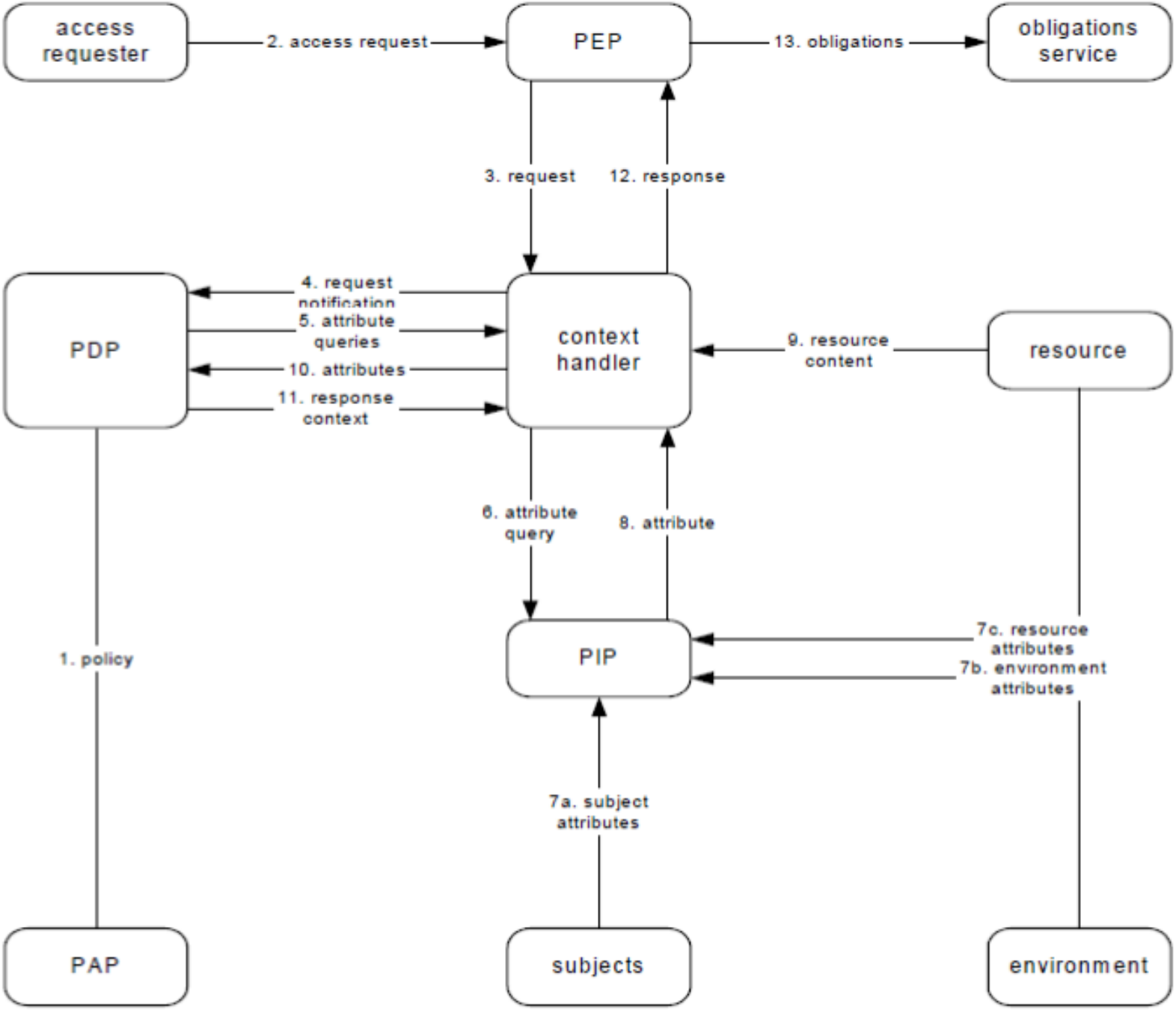
- Problems with this are:
  - Doesn't correctly model the real world
    - I should only be able to see my team's salaries, and only while participating in the salary review process
  - Hard to evolve
  - Hard to manage compliance
    - How do I find out who can make a trade of $30m?

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)


access requester
2. access request
PEP
13. obligations
obligations service
3. request
12. response
PDP
4. request notification
5. attribute queries
10. attributes
11. response context
context handler
9. resource content
resource
6. attribute query
8. attribute
1. policy
PIP
7c. resource attributes
7b. environment attributes
7a. subject attributes
PAP
subjects
environment

What can you do with XACML?

- Encode rules such as:
- X can access record Y if:
  - X is the patient for record Y
  - X is the doctor of the patient and working on the patients behalf
  - X is the guardian or parent of the patient
- How does this data get to the decision point?
  - In claims (e.g. there may be a claim saying that the doctor is logged into the hospital records system)
  - In further information available to the PIP
    - LDAP lookup shows I am Z's guardian

WSO2 Identity Server

- An Open Source identity gateway and identity management system
  - Works with existing user systems including LDAP, ActiveDirectory and RDBMS user stores
  - Provisioning by self-signup as well as SCIM
  - Single Sign on with SAML2 and OpenID Connect
  - Authorization and access control via XACML 3.0
  - Kerberos, WS-Trust, OpenID, etc
  - http://wso2.com/products/identity-server/

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!)