

Introduction to SOA Security

Oxford University
Software Engineering Programme
Dec 2013



© Paul Fremantle 2012. Portions © Jeremy Gibbons 2010, © WSO2 2005-2012 used with permission of the author(s).
Licensed under the Creative Commons 3.0 BY-SA (Attribution-Sharealike) license.
See <http://creativecommons.org/licenses/by-sa/3.0/>

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



© Paul Fremantle 2012. Portions © Jeremy Gibbons 2010, © WSO2 2005-2012 used with permission of the author(s).
Licensed under the Creative Commons 3.0 BY-SA (Attribution-Sharealike) license.
See <http://creativecommons.org/licenses/by-sa/3.0/>

Authenticity

- The actual implementation of this is different
- Most interactions
 - only use PKI for one sided authenticity (do I trust the server?),
 - HTTP Basic Auth for the client
- No direct trust of the server (I don't want to store a key for every server I talk to)
 - Certificate Authority model
 - I trust the CA, the CA trusts the Website => I trust the website



HTTP Authorization

- Mutual TLS/SSL
 - Hardly ever used. Requires managing certificates for every client
 - Have you tried to get your mum to install a client cert?!
- Basic Auth
 - Easy, fast, effective
 - Completely insecure except over HTTPS – base64 encoding
- Digest
 - Hash of the password



Basic vs Digest

- Digest looks good on the wire
 - Secure hash of the password is passed over the wire
- How does the server validate?
 - Needs to have the password stored using reversible encryption
- Basic looks bad on the wire
 - Unencrypted password
- Server can store the hash instead
 - Much more likely that a hacker will attack the server and steal 100m passwords, than break a TLS session and steal one!



Integrity

- A Hash Function
 - Takes a large input (e.g. a message)
 - Creates a (hopefully) unique shortened *hash*
 - e.g. a 160-bit, 256-bit, 384-bit or 512-bit output
 - Exact inputs will always end with identical outputs
 - You cannot go from the output back to the input (that would be a wicked compression algorithm)
 - Except maybe by guessing?!



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

- Availability is an important aspect
- Many attacks have been based on bringing down one system before spoofing it
- e.g. famously Mitnick and Shimomura, Christmas Day 1994



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 may change that
- Its 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



TLS pros and cons

- Strongly encrypted two-way data channel
- (+) standard and straightforward
- (+) highly trustworthy
- (–) entirely point-to-point
- (–) no scope for processing by intermediaries (that includes firewalls and filters)
- (–) protocol-based; no scope for fine-grained authorization



OAuth/OAuth2

- Designed to solve the problem of LinkedIn/Gmail
 - I don't want to give LinkedIn my Gmail password so they can scan my contact list
 - I just want to give them 10 mins access to my contact list
- OAuth 1.x requires clients to implement signature algorithms
- OAuth 2.0 allows a vanilla client that implements HTTPS to work



Tokens

- Security Tokens provide an effective way of managing sign-on, authenticity and identity across a network
 - I need to sign on to multiple systems
 - I go to the token server and prove who I am
 - The token server issues me with a token for a given period
 - I present that token to the other systems
 - They validate that the trusted token server has signed this token
 - Since they trust the token server, they now trust the claims made by that token
- The only system that needs to validate my password or identity is the token server

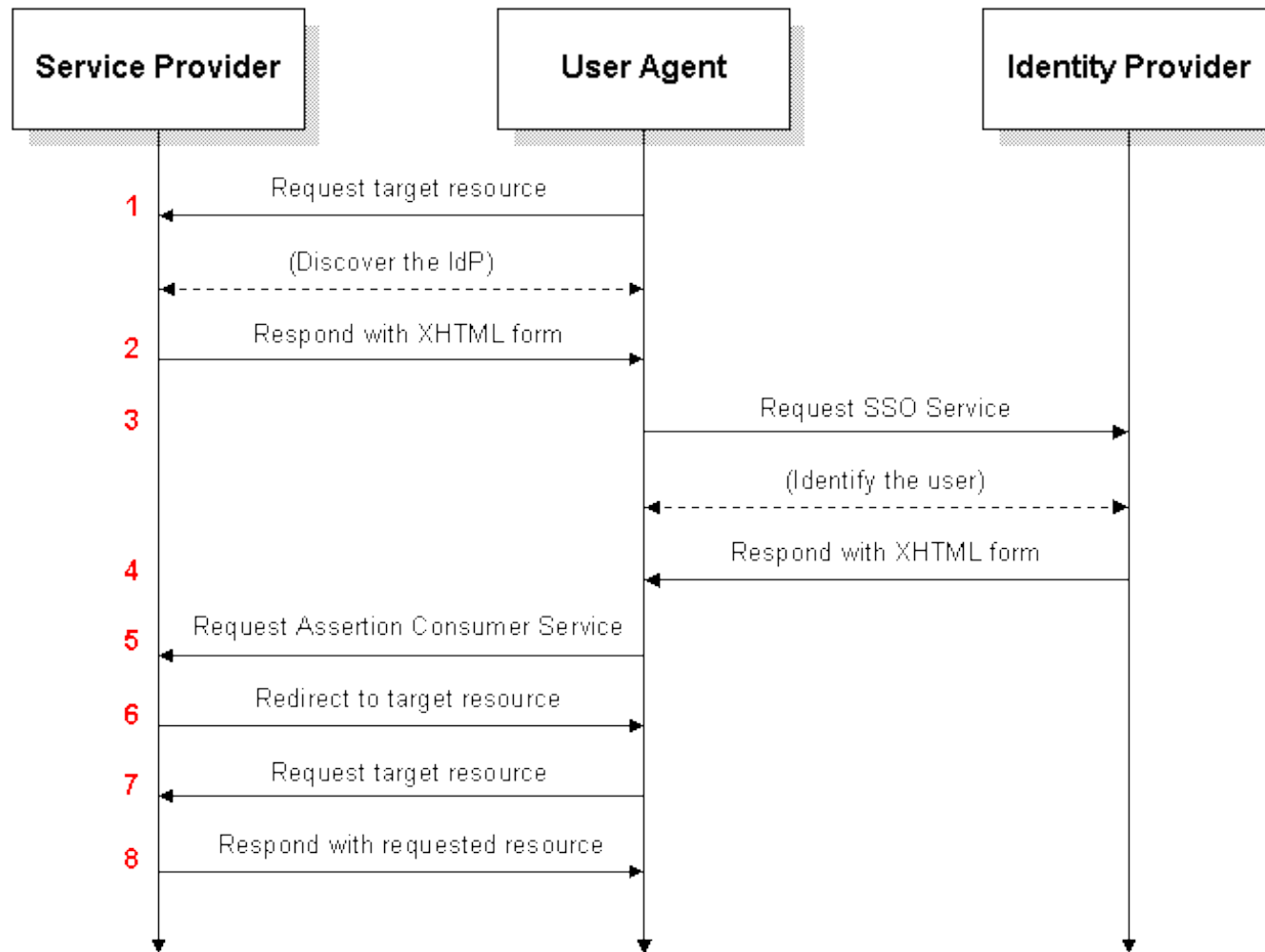


Tokens

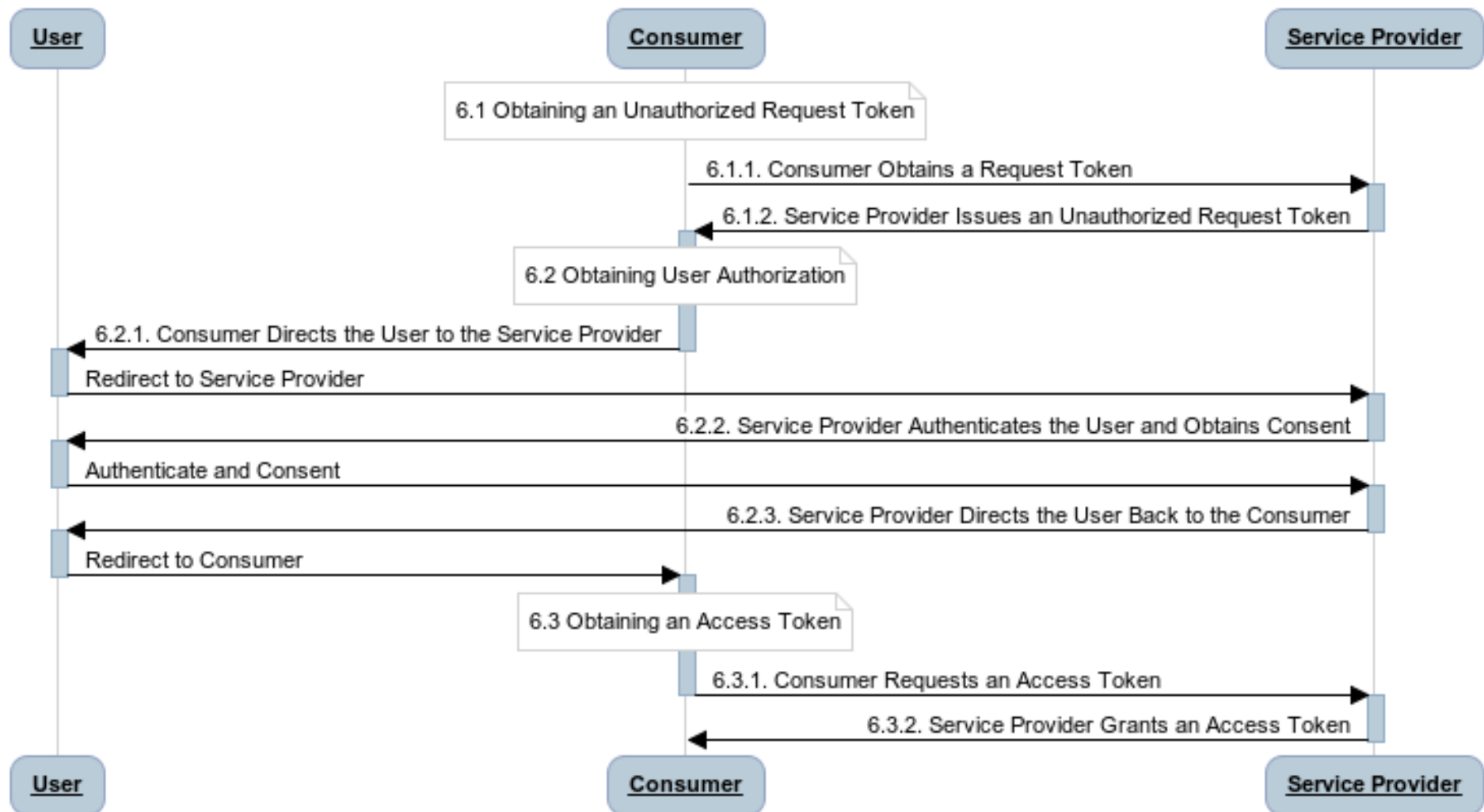
- Kerberos (known as tickets)
 - Developed by MIT in the late 80's onwards
 - Designed to allow lots of campus machines to be secured easily (without having local UNIX password files!)
- SAML/SAML2
 - An XML version that has become a popular way of doing Single Sign On for the Web
 - Used by Google Apps
- OAuth/OAuth2
 - Rapidly gaining popularity with Facebook/LinkedIn/Gmail/Github/etc



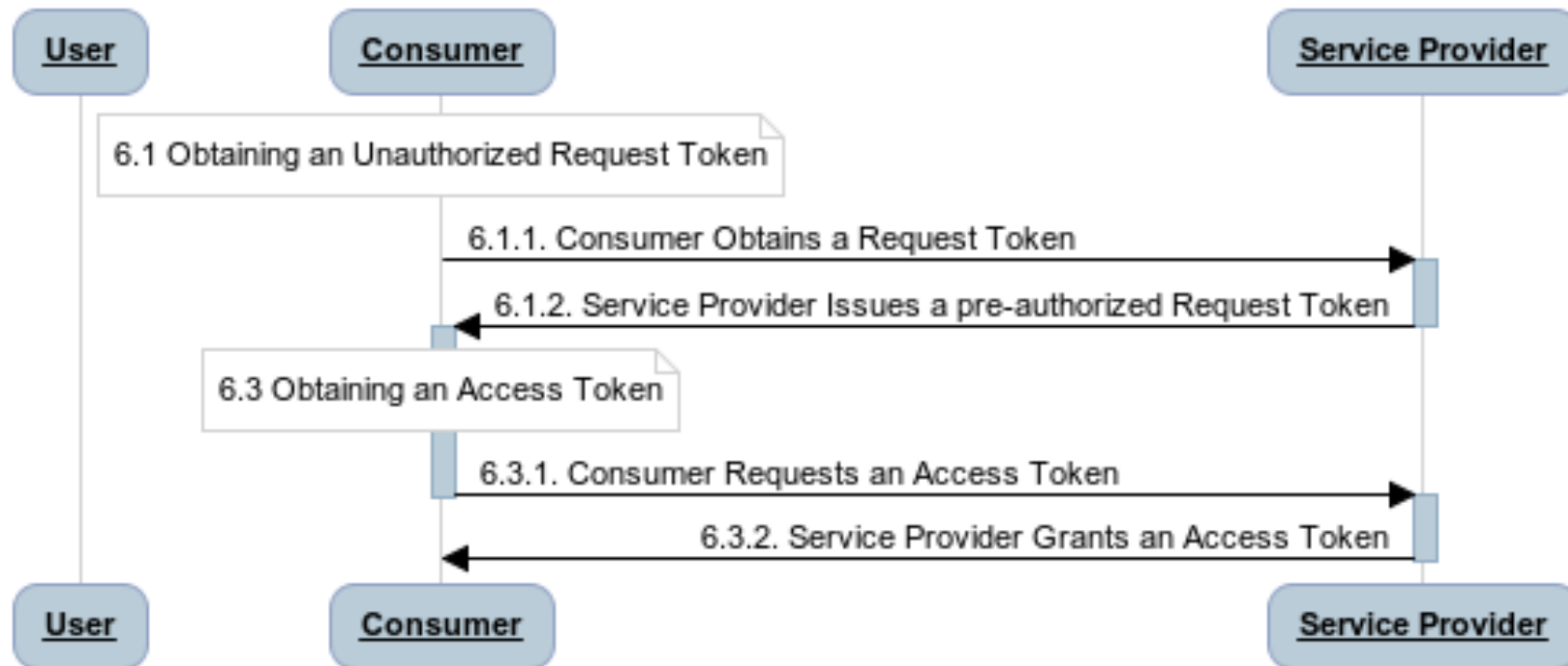
SAML2



Three-legged OAuth



Two Legged OAuth



www.websequencediagrams.com



© Paul Fremantle 2012. Portions © Jeremy Gibbons 2010, © WSO2 2005-2012 used with permission of the author(s).
Licensed under the Creative Commons 3.0 BY-SA (Attribution-Sharealike) license.
See <http://creativecommons.org/licenses/by-sa/3.0/>

REST Security

- HTTPS + Basic Auth
- Rapidly moving to HTTPS+OAuth2
- Does not support signatures, or message level encryption
- New standards emerging
 - <http://tools.ietf.org/html/draft-ietf-jose-json-web-encryption-07> (JWE)
 - <http://tools.ietf.org/html/draft-jones-json-web-signature-07> (JWS)
 - <http://tools.ietf.org/html/draft-ietf-oauth-json-web-token-05> (JWT)



Web Services Security

Covered in detail in another chapter

- 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



Authorization



© Paul Fremantle 2012. Portions © Jeremy Gibbons 2010, © WSO2 2005-2012 used with permission of the author(s).
Licensed under the Creative Commons 3.0 BY-SA (Attribution-Sharealike) license.
See <http://creativecommons.org/licenses/by-sa/3.0/>

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

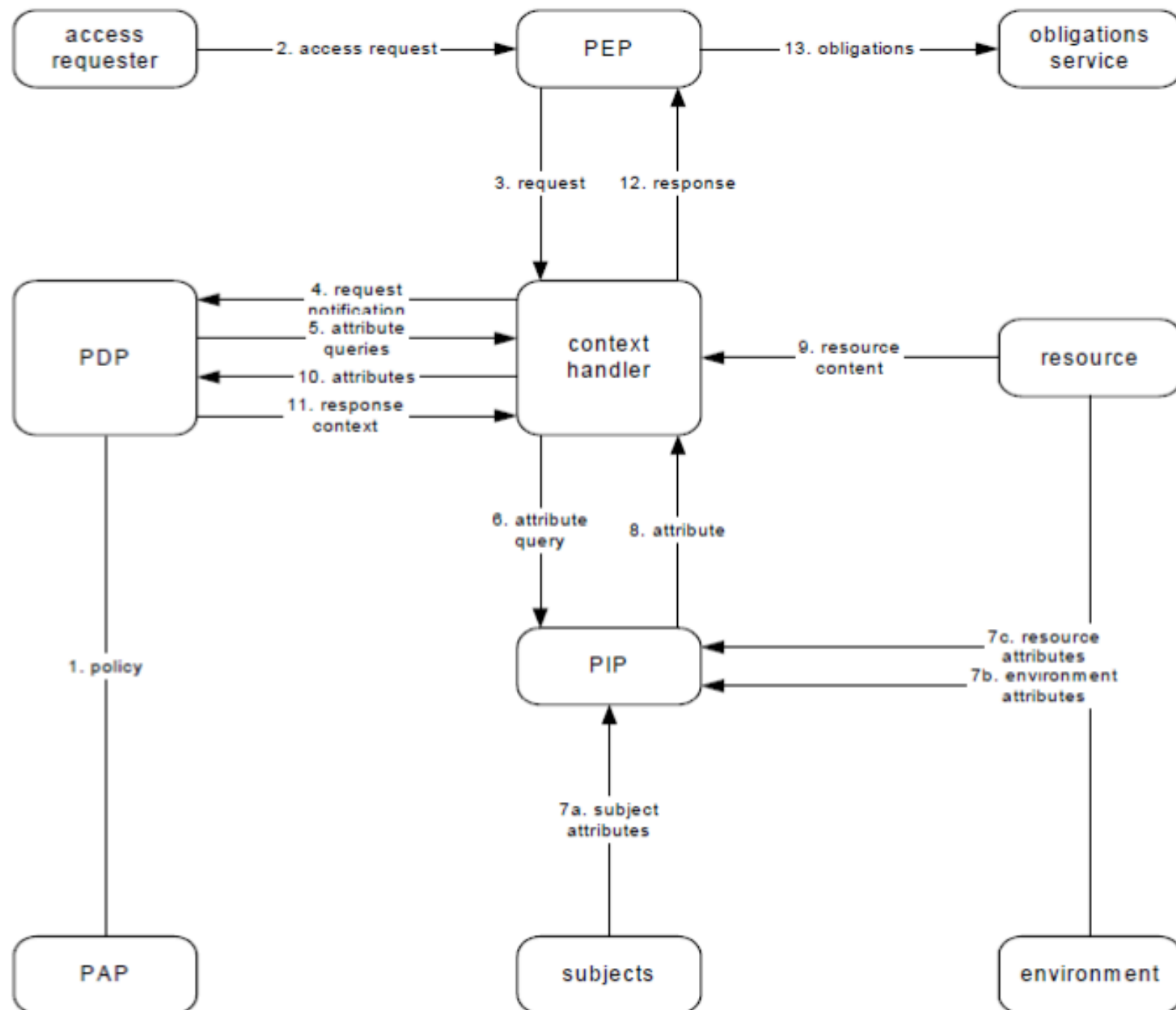
- Delegate access control decisions to a “Policy Decision Point”
- Utilise more information in the decision
- Externalise the decision from the code



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)





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



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!)

