

# **Distributed Users Authentication and Authorization for Cross Domains Web Applications**

**Sharil Tumin**

University of Bergen,  
IT-Dept., P. O. Box 7800,  
5020 Bergen, Norway

[edpst@it.uib.no](mailto:edpst@it.uib.no)

# Web Applications Security ...

- is about trust relations between pairs
  - client/server, sender/receiver, requester/responder
  - trusted third parties can be used as mediators
- means differently from different prospective.
  - user/system pair established security prospective
    - user ↔ user, user ↔ system, system ↔ system
- is not trivial

# Web-based Applications

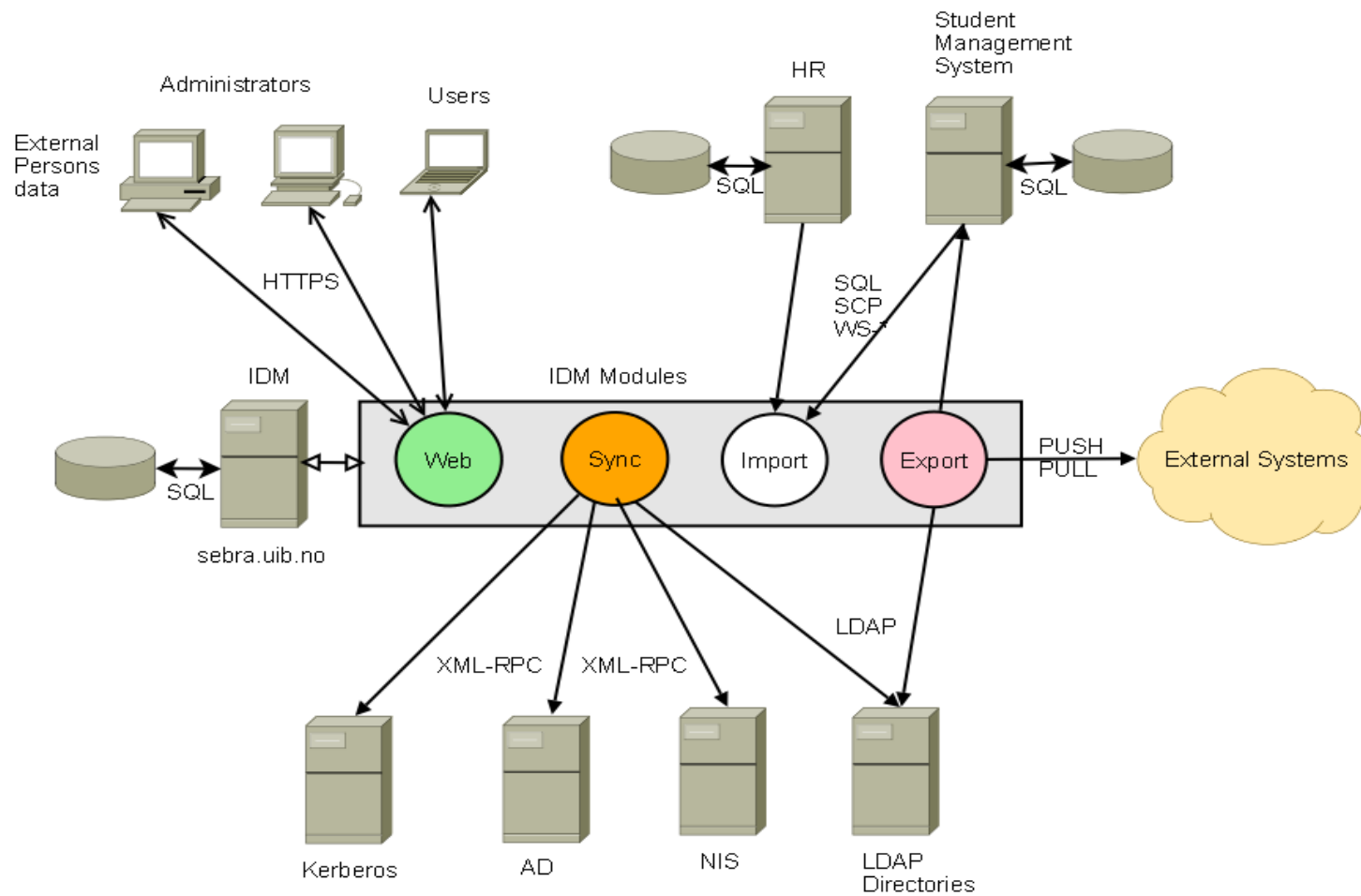
- user interacts with application using Web-browser.
  - any where, any time
  - PC in un-trusted public access environment
  - keyboard grabber spy-ware program
- protected Web applications need
  - user authentication – userid/password
  - user authorization – group memberships
- Independent users, groups and resources managements

# Enterprise IDM

## Identity management system goals

- domain wide one-person ↔ one-identity
  - increase accountability
  - increase IT department productivity
  - increase IT resources usage efficiency
  - increase users' satisfaction
  - increase security
- automatic and dynamic groups/roles management
- management through delegation of responsibilities

# IDM at University of Bergen



# Sharing Web Applications

Need users authentication/authorization

- Within organization
  - across departments
  - across faculties
- Between organizations
  - across organizations
  - library portals
  - federated applications

without exporting users' data into applications' databases

# Basic Terms and Concepts

- Domain – an independent organization
- User – something that have a domain identity, a person, a machine or a named software agent
- Group – a set of users
- Role – a set groups
- Resource – a set of protected Web objects
- Action – a matrix of operations applicable on resources
- Permissions – rights to perform actions

# Collaboration

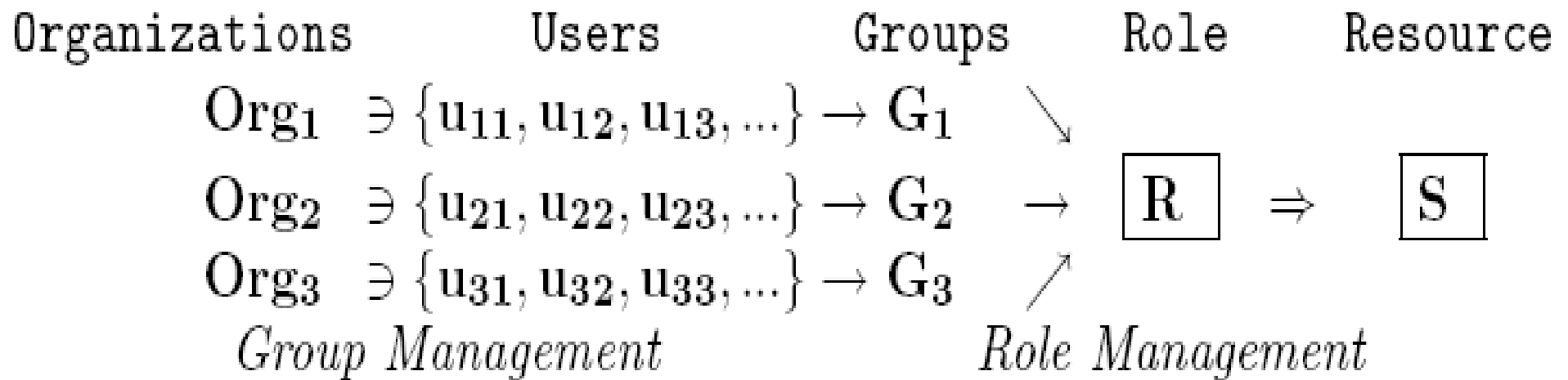
- Cooperation between two or more independent organizations.
- Sharing users, groups data across domains
- Sharing resources to all federated users
- Agreed name spaces:
  - user@domain
  - group@domain
  - host@domain
  - role@domain
  - agent@domain



# Distributed administration

- Client domain manages
  - domain users
  - domain groups
  - users membership to groups
- Provider domain manages
  - domain resources
  - domain roles
  - roles permissions on resources
- Collaborative binding
  - $\text{group@client} \in \text{role@provider}$

# Groups Roles Binding



# Access Control Logic

$user_n@domain_d$  can operate on  $resource_s@domain_e$

if and only if (1)  $\{domain_d, domain_e\} \in federation_f$  and

(2)  $user_n@domain_d$  is authenticated at  $domain_d$  and

(3)  $user_n@domain_d \in group_g@domain_d$  and

(4)  $group_g@domain_d \in role_r@domain_e$  and

(5)  $role_r@domain_e$  has permissions on  $resource_s@domain_e$  and

(6) dynamic constraints are met (e.g.. IP, time, roles-conflict) and

(7)  $user_n@domain_d \notin quarantine@domain_d$  and

(8)  $user_n@domain_d \notin quarantine@domain_e$

# Web-based Applications

- Presentation

- HTML
- XML

- Communication

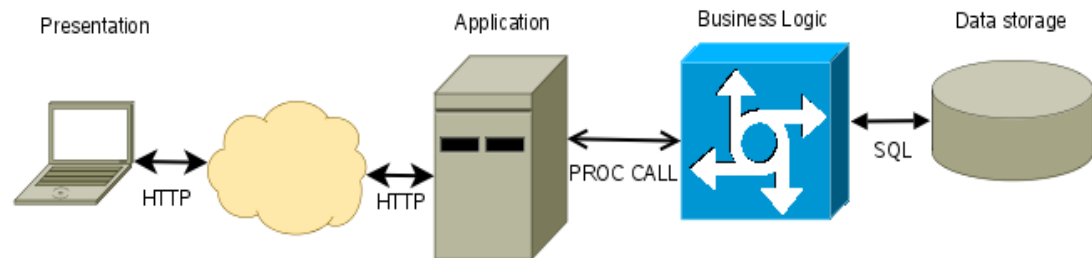
- client/server, request/response messages

- HTTP

- connectionless
- Stateless
- redirection

- States

- Clients 'cookies'



# URL redirects

- Meta refresh

tells the browser to move onto the next page after x seconds.

```
<meta http-equiv="refresh"
      content="0;url=https://sebra.uib.no/logon">
```

- Frame redirect

loads the contents of another page inside a frame

```
<frame name="redirect"
      src="https://sebra.uib.no/registration"></frame>
```

- Header redirect

returned a Location header telling the browser to go to another page as server response

```
HTTP/1.1 200 OK
```

```
...
```

```
Location: https://sebra.uib.no/newPassword
```

# Web Cookies

- Preserving states across stateless communication
  - Client request a Web object, server send cookies and the requested object
  - client save cookies and attach the cookies to its future request to the server if validity check is true

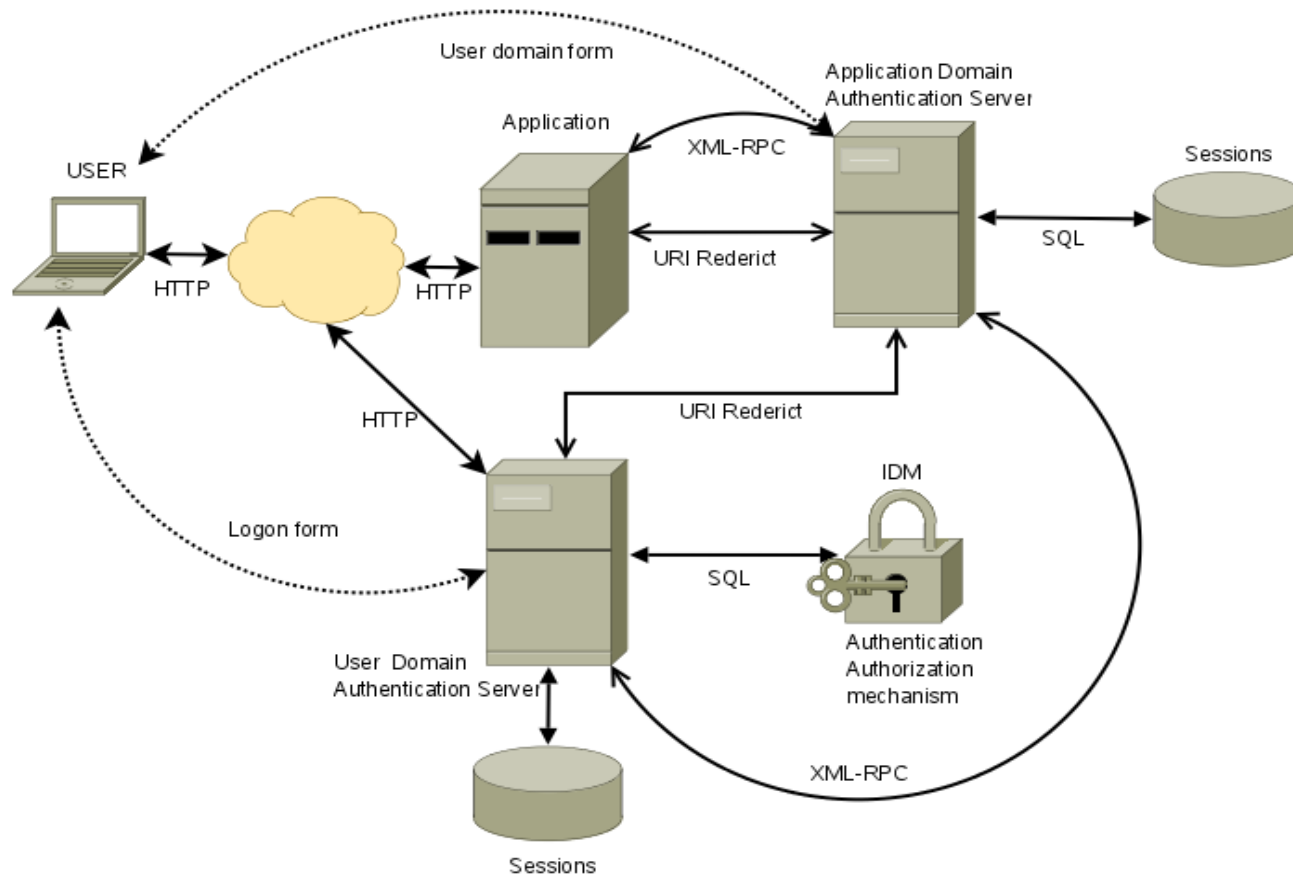
- Server Set-Cookie respond header

```
Set-Cookie: admin=Sharil; expires=Fri,11-Apr-2008 12:05:00;  
path=/reg; domain=sebra.uib.no; secure
```

- Client HTTP request

```
GET /reg HTTP/1.0  
From: edpst@it.uib.no  
User-Agent: HTTPTool/1.0  
Cookie: admin=Sharil
```

# User Domain Logon Mechanism by URL-redirect



# Implementation – Symbols

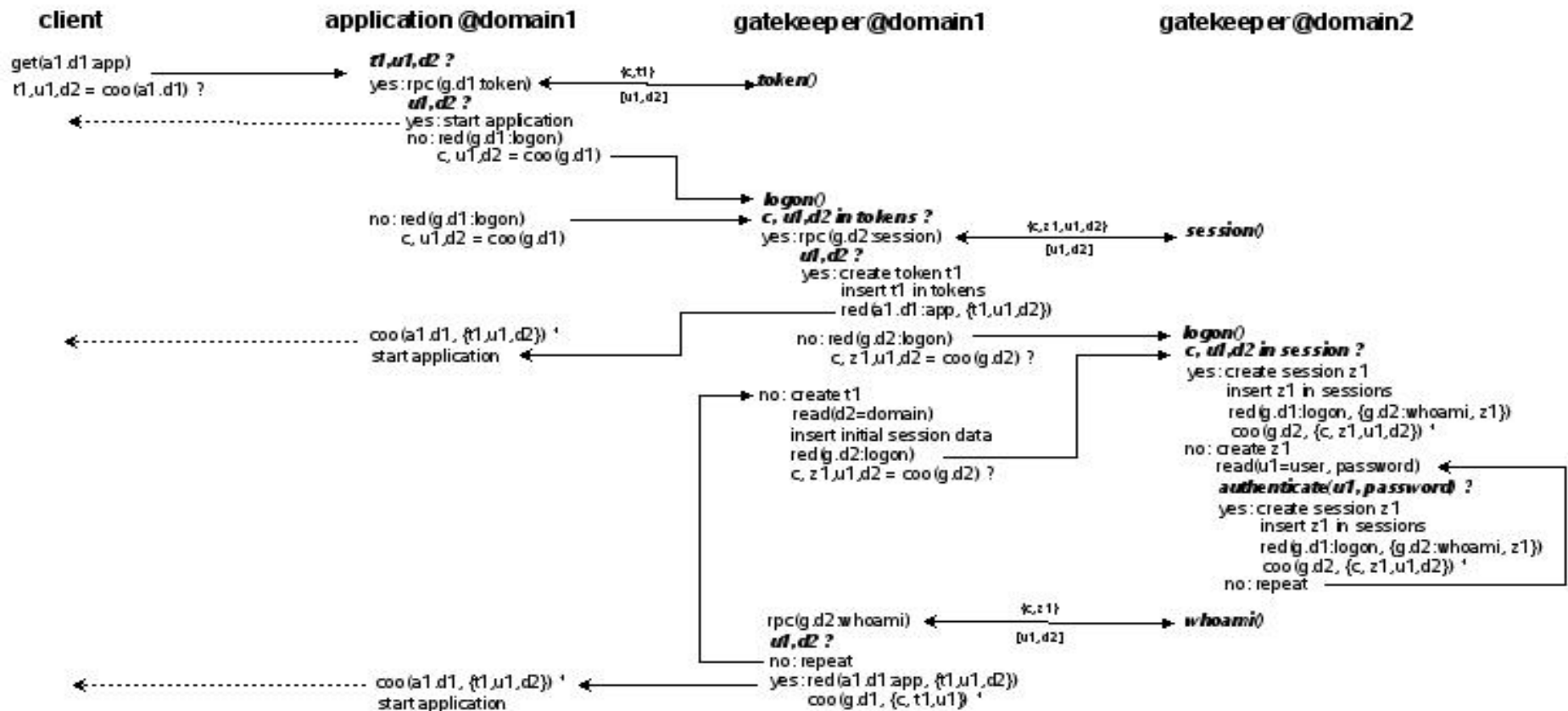
<b><i>d</i></b>	Organization domain name as defined in DNS
<b><i>a</i></b>	Application server hostname e.g. <b><i>a<sub>1</sub>.d<sub>1</sub></i></b>
<b><i>g</i></b>	Logon server hostname e.g. <b><i>g<sub>1</sub>.d<sub>2</sub></i></b>
<b><i>u</i></b>	User name <b><i>u<sub>2</sub>@d<sub>1</sub></i></b>
<b><i>t</i></b>	Web token
<b><i>z</i></b>	Web session
<b><i>c</i></b>	Client IP address



# Implementation - Functions

coo()	coo( <i>s1.d2</i> {v1,v2})	A set of cookies set by a particular server
red()	red( <i>s1.d2:app</i> {parm})	HTTP redirect. Redirect to a particular server and a particular application with a given parameters
get()	get( <i>s1.d2:app</i> {parm})	HTML GET request. A GET request to a particular server and a particular application with a given parameters
rpc()	rpc( <i>s1.d2:app</i> {parm})	HTML-RPC call. Do a RPC call to a particular server and a particular application with a given parameters

# Logon Process Flow



# Logon - Step1 & Step2

1 *client*: get(**a1.d1**:app, *nil*); coo(**a1.d1**, *nil*) →

1.1 red(**g.d1**:logon, {**a1.d1**:app});

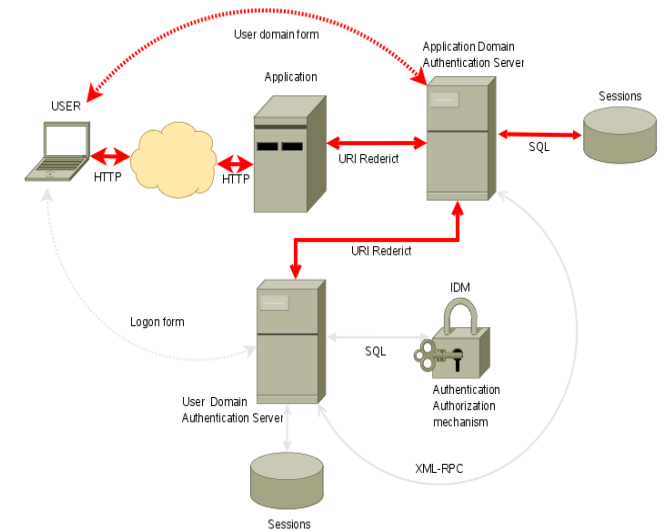
2 *client*: get(**g.d1**:logon, *nil*); coo(**g.d1**, *nil*) →

2.1 logon: **d2** = domain

2.2 creates unique token **t1**;

2.3 save: (**c**, **t1**, session=*nil*, user=*nil*, **d2**, **a1.d1**:app, timestamp)  
into tokens database;

2.4 red(**g.d2**:logon, {**g.d1**:logon, **t1**});



# Logon - Step3

3 client: get(**g.d2**:logon, [user, password],

{**g.d1**:logon, **t1**}); coo(**g.d2**, nil) →

3.1 logon: authenticate(**u1**, password);

3.2 if authenticated then

3.2.1 creates unique session **z1**

associated

with user **u1**;

3.2.2 save: (**c**, **z1**, **u1**, **t1**, **g.d1**, timestamp)

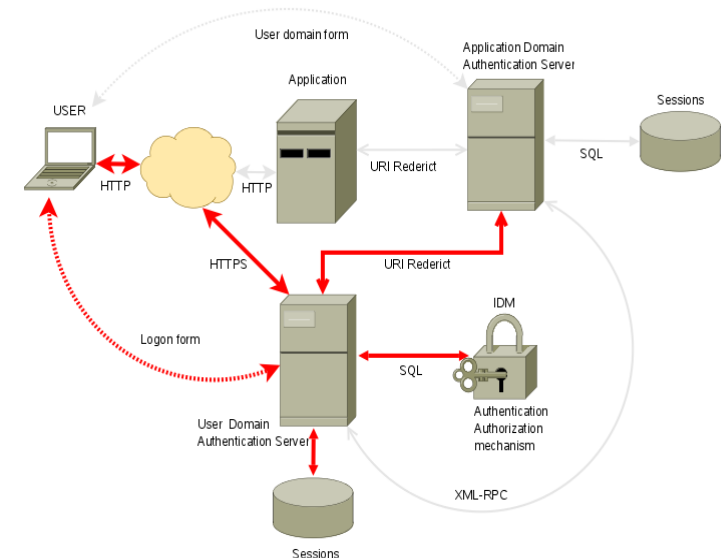
into sessions database;

3.2.3 coo(**g.d2**, {session=**z1**, user=**u1**, domain=**d2**});

3.2.4 red(**g.d1**:logon, {**g.d2**:whoami, **z1**});

else

3.2.5 repeat 3;



# Logon - Step4

4 client: get(**g.d1**:logon, {**g.d2**:whoami, **z1**});  
coo(**g.d1**, *nil*) →

4.1 user, domain, token =  
rpc(**g.d2**:whoami,  
{**c**, **g.d1**, **z1**});

4.3 if not (token == *nil*) then

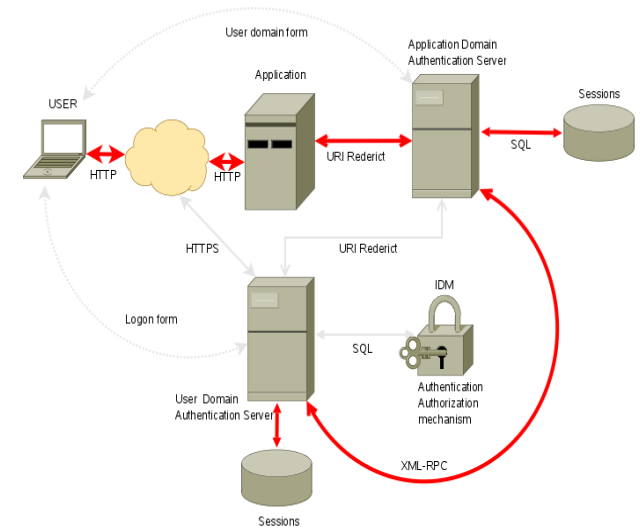
4.3.1 update: (**c**, **t1**, **z1**, **u1**, **d2**, **a1.d1**:app, timestamp);

4.3.2 coo(**g.d1**, {token=**t1**, user=**u1**, domain=**d2**});

4.3.3 red(**a1.d1**:app, {**g.d1**:session, **t1**});

else

4.3.4 red(**g.d2**:logon, {**g.d1**:logon, **t1**});



# Logon - Step5

5 client: get(**a1.d1**:app, {**g.d1**:session, **t1**}); coo(**a1.d1**, **nil**) →

5.1 user, domain = rpc(**g.d1**:session, {**c**, **a1.d1**, **t1**});

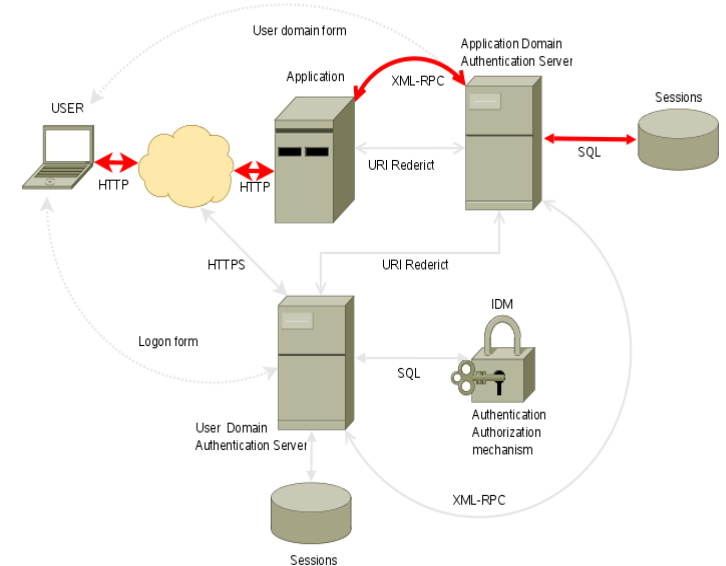
5.2 if not (user@domain == **nil**) then

5.2.1 coo( **a1.d1**, {token=**t1**, user=**u1**, domain=**d2**}):

5.2.2 start application;

else

5.2.3 red(**g.d1**:logon, {**a1.d1**:app})



# Sessions

- User sessions and tokens can be used to implement SSO
- Sessions can be transferred between domains
- Sessions and authorization data can be used to implement access control on protected resources
- Session data can be used to do local or global sign-off procedure.

# PASS Card

- Web logon without using domain users' credentials
- minimize domain-wide security breach
- one can request PASS cards as often as one will using userid/password credential
- PASS Card requests only within the organization local-network




# PASS Card Request

Es	xN	jd
GW	eG	zQ
ya	uJ	mx
ht	ih	Rg
1118905713-9		

Your nickname: **Happy Monkey**

Your PASS card will provide you with keys to be used during login to system using PASS system.

Use your choosen nickname and PASS keys combination.




# Logon using PASS card

		12
34		
56		

**Nickname:**

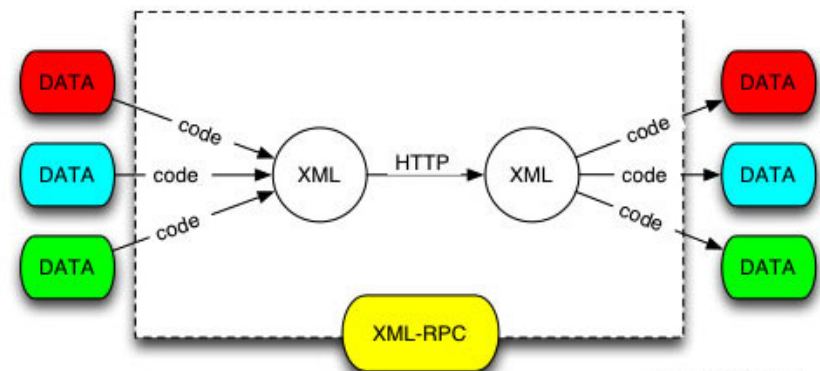
**PASS keys:**

Write your nickname and keys combination, click Login.



# XML-RPC Based Communication

- simple protocol – XML encoded call over HTTP
- basic data types
  - nil, integer, double, boolean
  - base64, string, date/time
  - array, struct
- commands
  - methodCall
    - methodName, params
  - methodRespond
    - params, fault

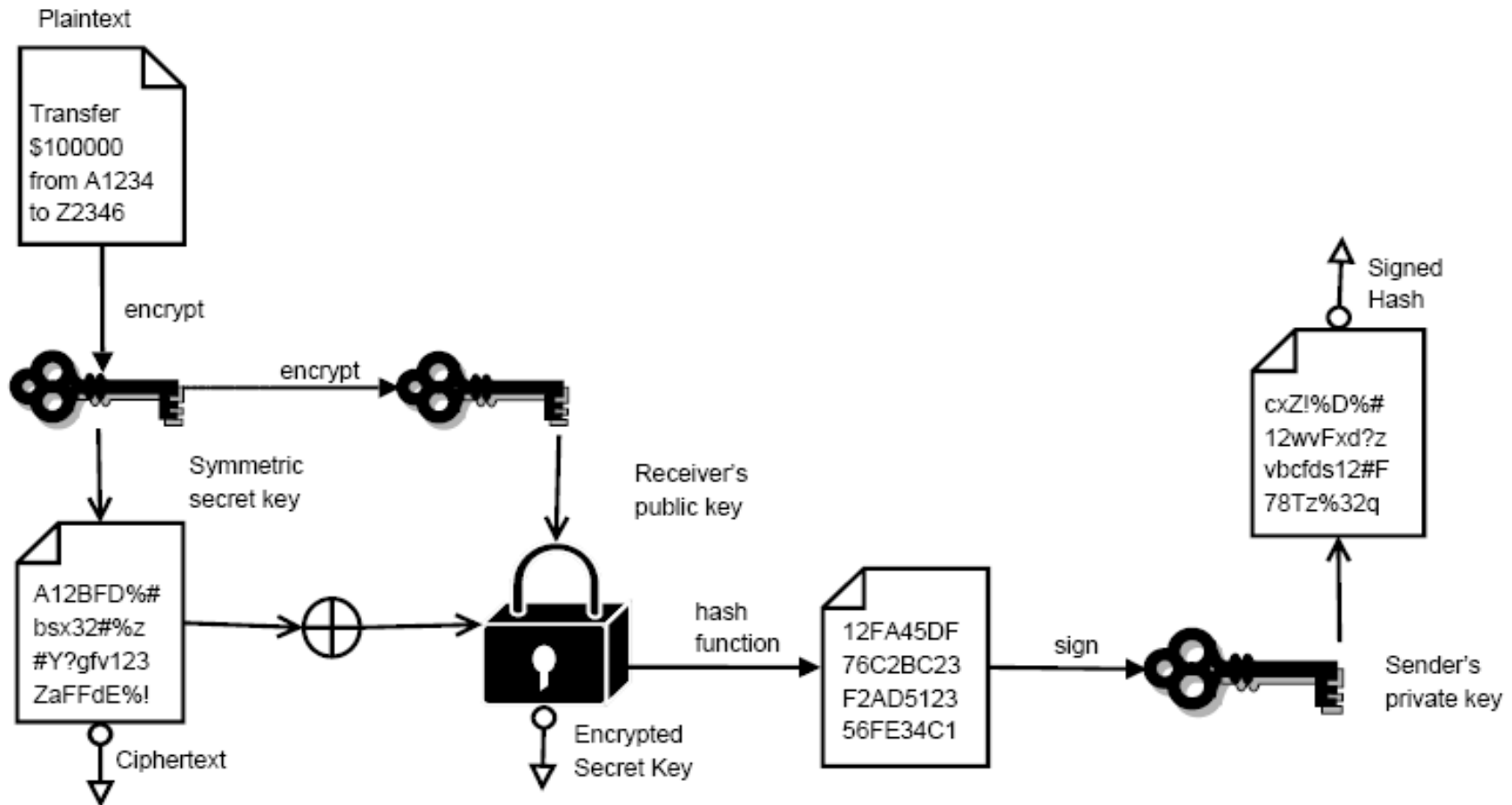


Source: JY Stervinou

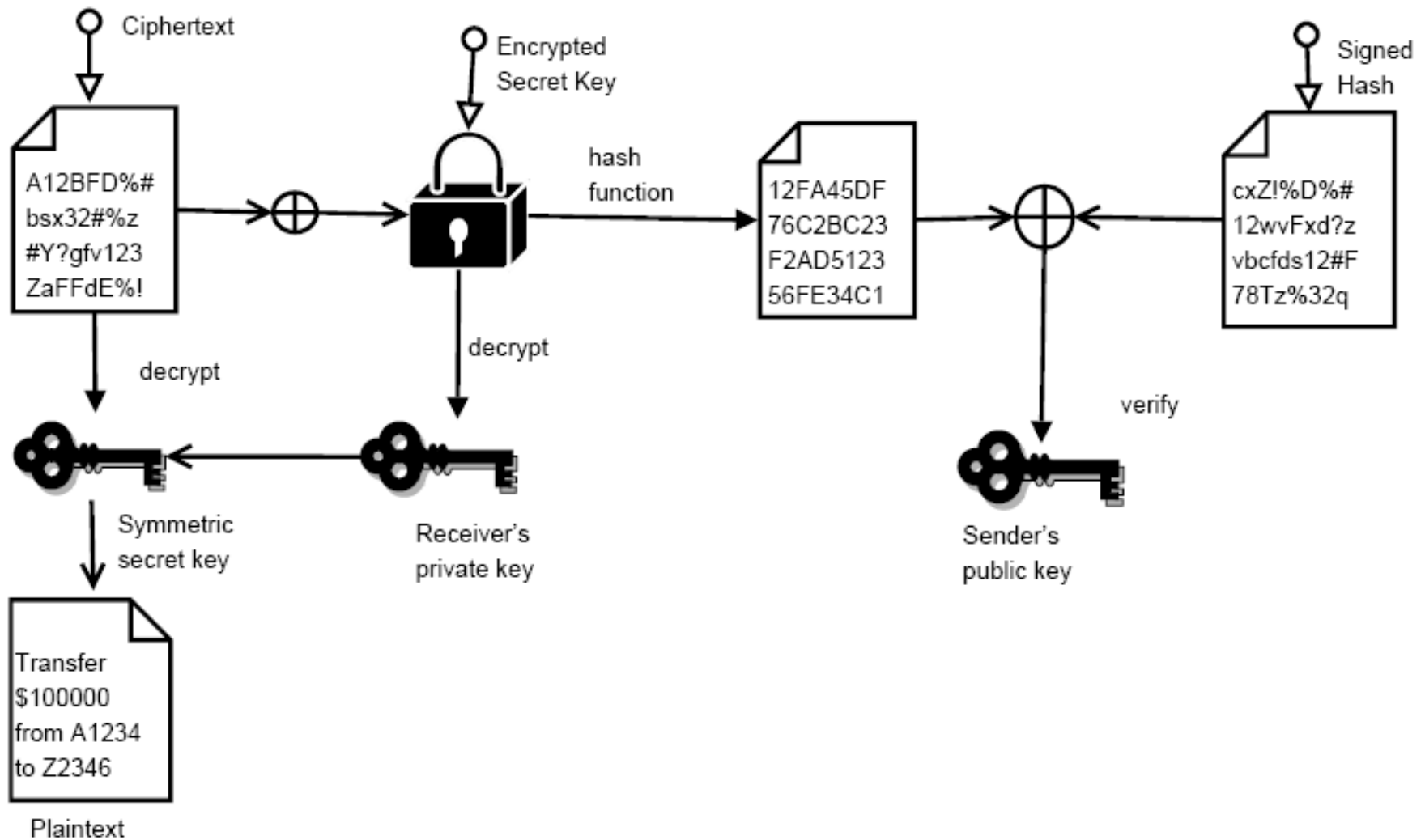
# Signed Digital Envelope

- use secret-key cryptography to encrypt XML request/response block into payload
- use receiver public key to encrypt the secret-key into skey
- hash-value = hash-function(payload+skey)
- use sender private key to sign hash-value
- send payload, skey and hash-value as XML request/response param block
- reverse the process at the receiving end

# Package Sending



# Receiving Package



# Conclusion

- security is important and is based on trust
- security is a balance between accessibility and control
- split (users, groups) and (roles, resources) cross domains managements
- URL-redirect and clients cookies to implement users sessions for SSO and roles-based access control
- PASS card logon minimize the risk of stolen credentials
- signed digital envelopes protect both privacy and authenticity of message exchange

# Research Publication Points

Data taken from NSD data:

<http://dbh.nsd.uib.no/pub/>

Vitenskapelig publisering  
publikasjonspoeng/forfatterandeler

	USIT UiO	IT-Avd UiB
2004	0.2	
2005	1.0	0.5
2006	1.3	2.0
2007	2.6	<b>10.4</b>



Thanks

Tusen takk

Gracias

Obrigado