OpenID Security Analysis and Evaluation

San-Tsai Sun, Kirstie Hawkey, Konstantin Beznosov

Laboratory for Education and Research in Secure Systems Engineering (**LERSSE**)
University of British Columbia

summary

- CSRF attacks
  - single-sign-on CSRF (force victim to login) (70%)
  - account profile CSRF (50%)
  - login CSRF (login as attacker) (73%)
- authentication response interception
  - impersonation (67%)
  - replay attack (6%)

agenda

- background
- approach and evaluation result
- countermeasures

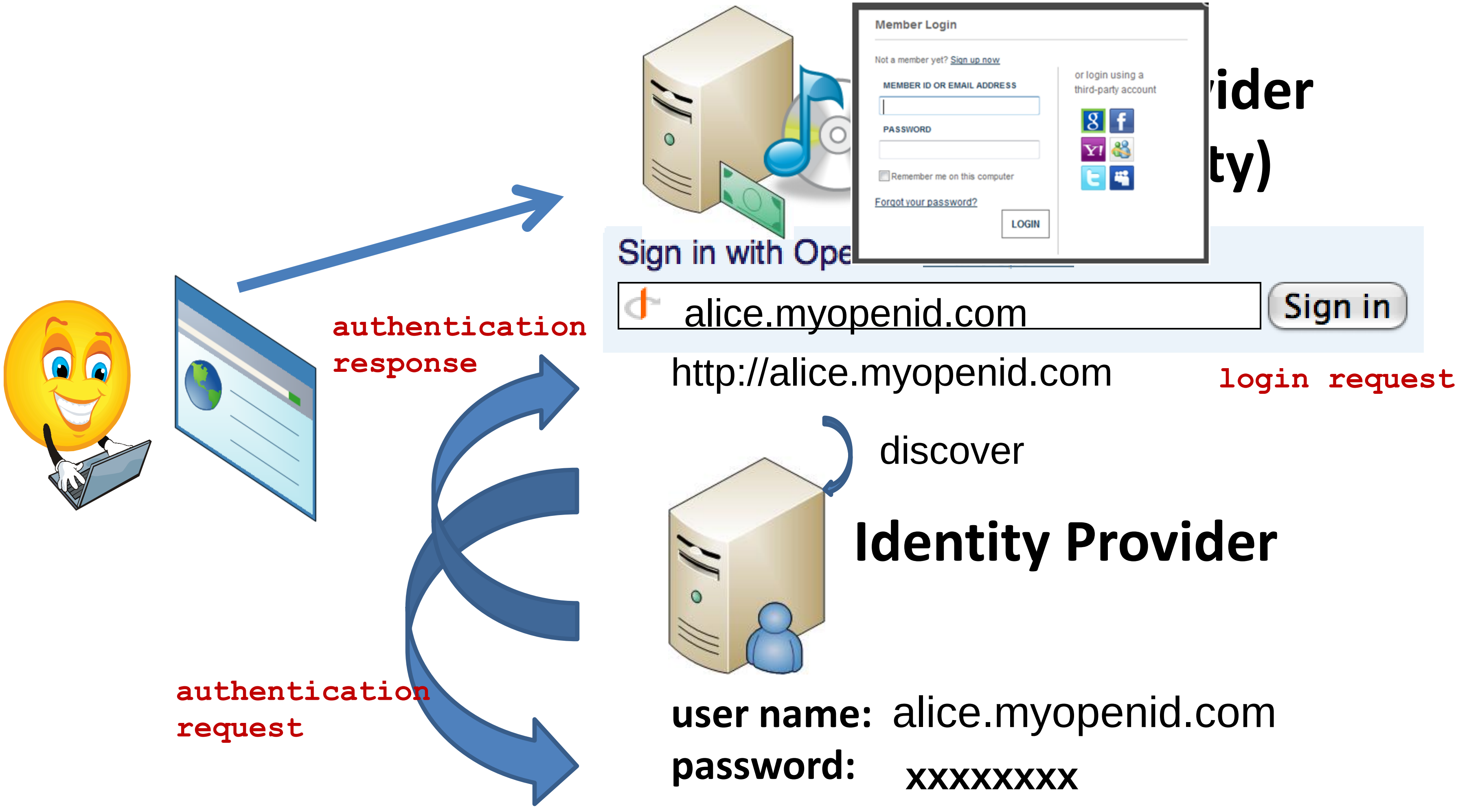
OpenID

- open and user-centric Web single sign-on protocol
- OpenID Foundation (2007) [1]
  - Microsoft, Google, IBM, Yahoo, VeriSign, Facebook, PayPal, PingIdentity
- over **one billion** OpenID enabled user accounts provided by Google, Yahoo, AOL…[1]

---

[1] OpenID Foundation. http://openid.net/foundation/, 2009.

how OpenID works



Member Login

Not a member yet? Sign up now

MEMBER ID OR EMAIL ADDRESS

PASSWORD

☐ Remember me on this computer

Forgot your password?

LOGIN

or login using a third-party account

ider

ty)

Sign in with Ope

alice.myopenid.com

Sign in

http://alice.myopenid.com

login request

authentication response

discover

Identity Provider

authentication request

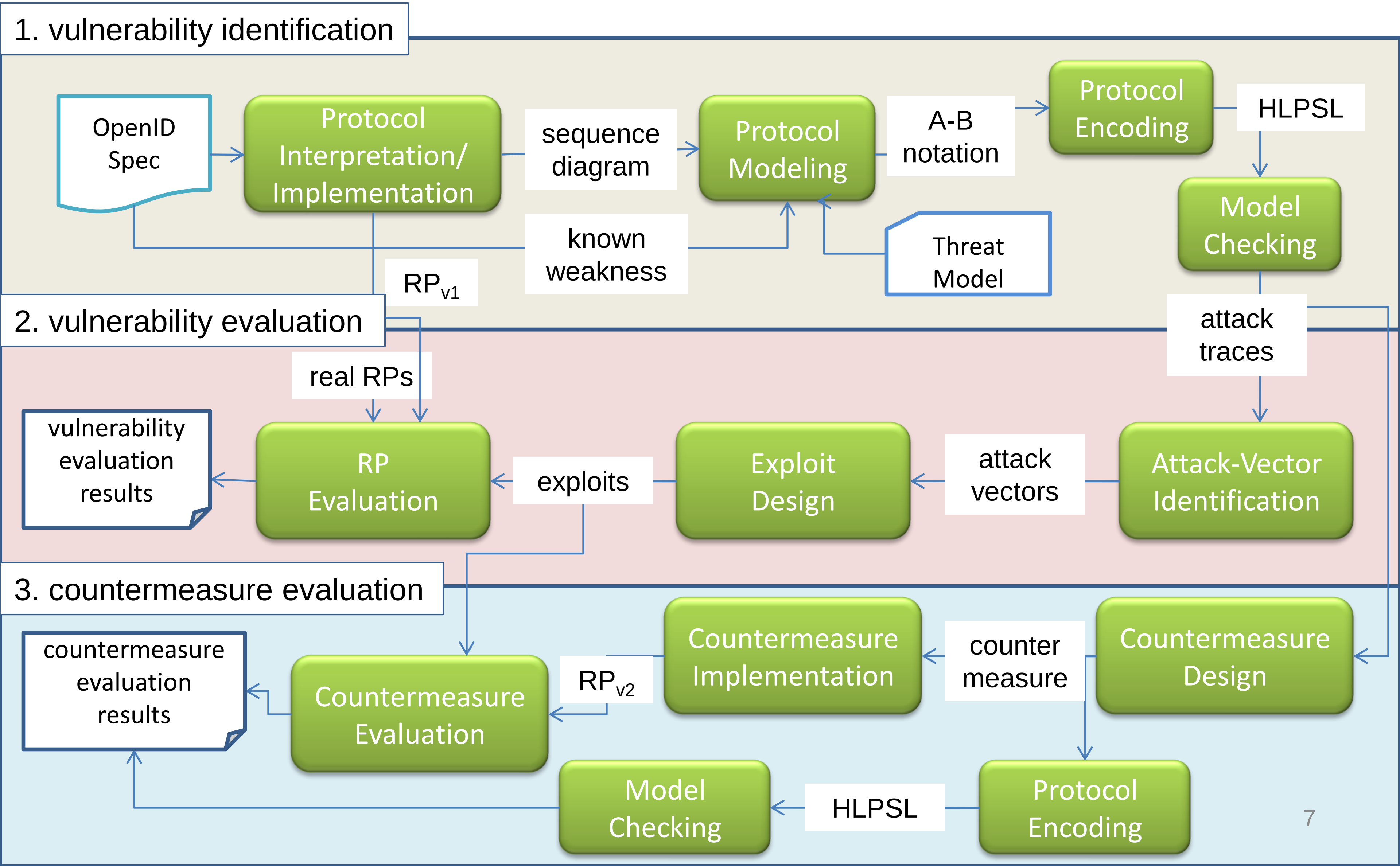
user name: alice.myopenid.com

password: xxxxxxxx

agenda

- background of OpenID
- approach and evaluation results
- countermeasures

methodology



1. vulnerability identification

OpenID Spec

Protocol Interpretation/ Implementation

sequence diagram

Protocol Modeling

A-B notation

Protocol Encoding

HLPSL

Model Checking

known weakness

Threat Model

RP_{v1}

attack traces

2. vulnerability evaluation

real RPs

vulnerability evaluation results

RP Evaluation

exploits

Exploit Design

attack vectors

Attack-Vector Identification

3. countermeasure evaluation

countermeasure evaluation results

Countermeasure Evaluation

RP_{v2}

Countermeasure Implementation

counter measure

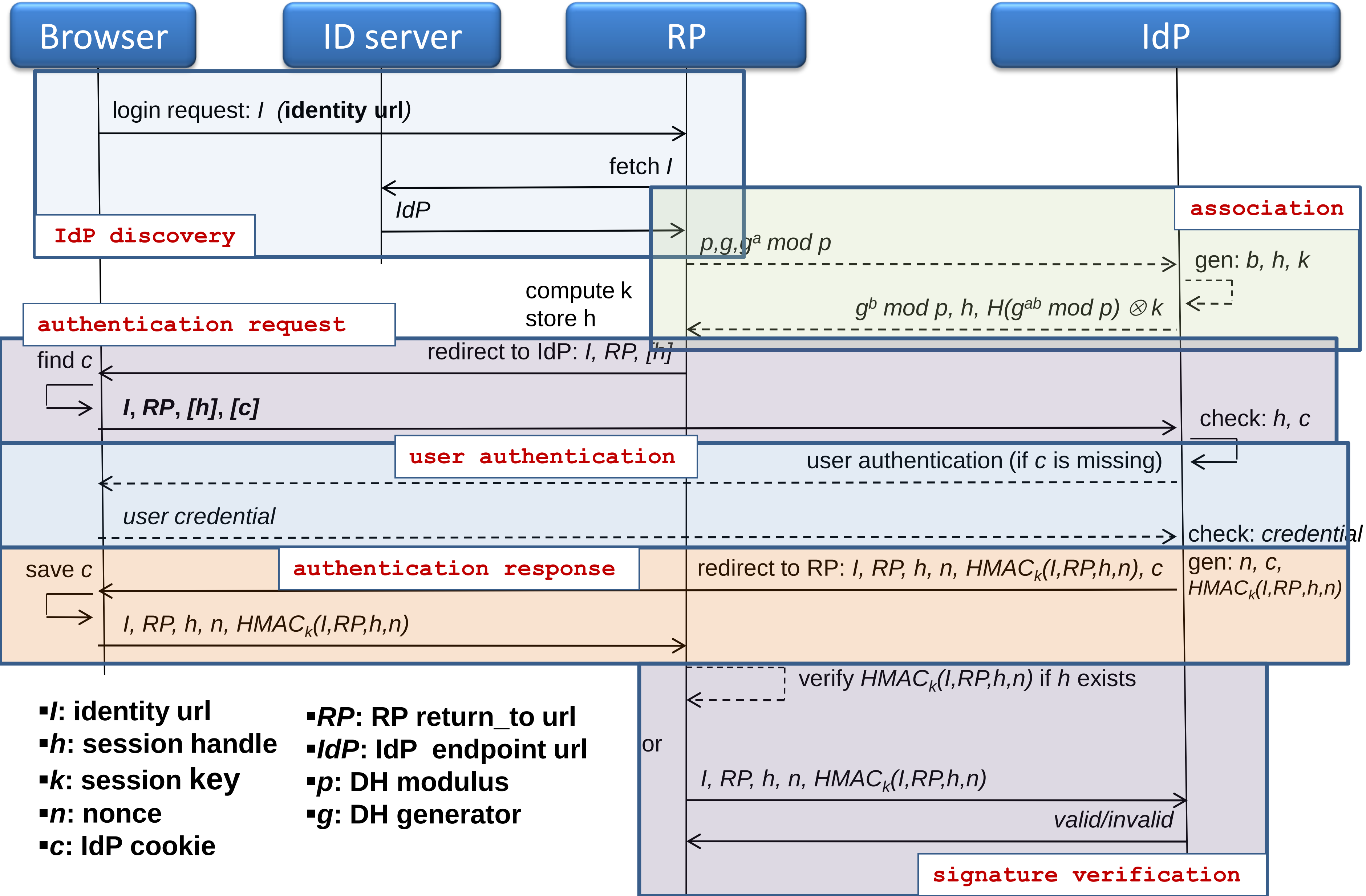
Countermeasure Design

Model Checking

HLPSL

Protocol Encoding

OpenID sequence diagram



Browser

ID server

RP

IdP

login request: *I* (**identity url**)

fetch *I*

IdP

IdP discovery

association

p,g,g^a mod p

gen: *b, h, k*

g^b mod p, h, H(g^{ab} mod p) ⊗ k

compute k
store h

authentication request

find *c*

redirect to IdP: *I, RP, [h]*

I, RP, [h], [c]

check: *h, c*

user authentication

user authentication (if *c* is missing)

user credential

check: *credential*

save *c*

authentication response

redirect to RP: *I, RP, h, n, $HMAC_k$(I,RP,h,n), c*

gen: *n, c, $HMAC_k$(I,RP,h,n)*

I, RP, h, n, $HMAC_k$(I,RP,h,n)

verify *$HMAC_k$(I,RP,h,n)* if *h* exists

or

I, RP, h, n, $HMAC_k$(I,RP,h,n)

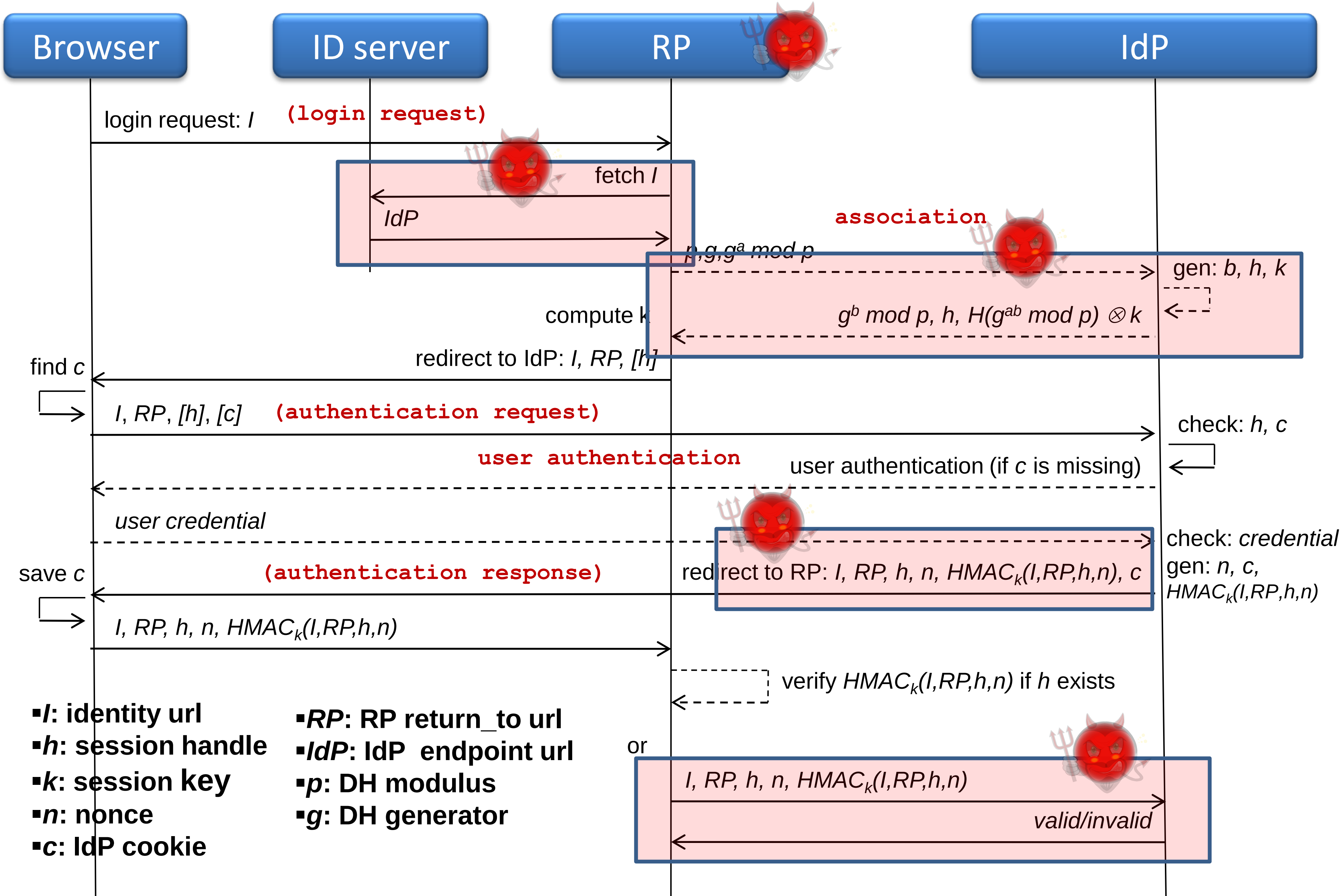
valid/invalid

signature verification

- ***I*: identity url**
- ***h*: session handle**
- ***k*: session key**
- ***n*: nonce**
- ***c*: IdP cookie**

- ***RP*: RP return_to url**
- ***IdP*: IdP endpoint url**
- ***p*: DH modulus**
- ***g*: DH generator**

known weakness


Browser
ID server
RP
IdP
login request: I
(login request)
fetch I
IdP
association
$p, g, g^a \bmod p$
gen: b, h, k
compute k
$g^b \bmod p, h, H(g^{ab} \bmod p) \otimes k$
redirect to IdP: $I, RP, [h]$
find c
$I, RP, [h], [c]$
(authentication request)
check: h, c
user authentication
user authentication (if c is missing)
user credential
check: *credential*
gen: $n, c, HMAC_k(I,RP,h,n)$
save c
(authentication response)
redirect to RP: $I, RP, h, n, HMAC_k(I,RP,h,n), c$
$I, RP, h, n, HMAC_k(I,RP,h,n)$
verify $HMAC_k(I,RP,h,n)$ if h exists
or
$I, RP, h, n, HMAC_k(I,RP,h,n)$
valid/invalid

- ***I***: **identity url**
- ***h***: **session handle**
- ***k***: **session key**
- ***n***: **nonce**
- **c: IdP cookie**

- ***RP***: **RP return_to url**
- ***IdP***: **IdP endpoint url**
- ***p***: **DH modulus**
- ***g***: **DH generator**

threat model

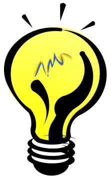
- adversary: non-RP or IdP associated attackers
- adversary goal: unauthorized access/modification to uses’ data hosted on RP
- Web poster: post comments
- Web attacker:
  - setup a malicious website
  - send malicious links via spam
  - deliver malicious content via Ads network
  - exploit web vulnerabilities (i.e., XSS) of benign websites
- network attacker:
  - setup an wireless access point
  - compromise client DNS resolution

threat assumptions

- RP, IdP, user machine, and browser are not compromised
- RP, IdP are not malicious
- user credentials on IdPs are secured
- cookies in the browser are secured (integrity and confidentiality)

non-considered threats

- availability threat
  - DoS by sending massive concurrent auth requests to an IdP
  - DoS by sending massive concurrent auth responses to an RP
- identity spoofing
  - phishing attacks by RP
  - exploits vulnerabilities on IdP
- integrity of IdP discovery process
  - altering discovery information
  - compromise RP DNS resolution



found weakness

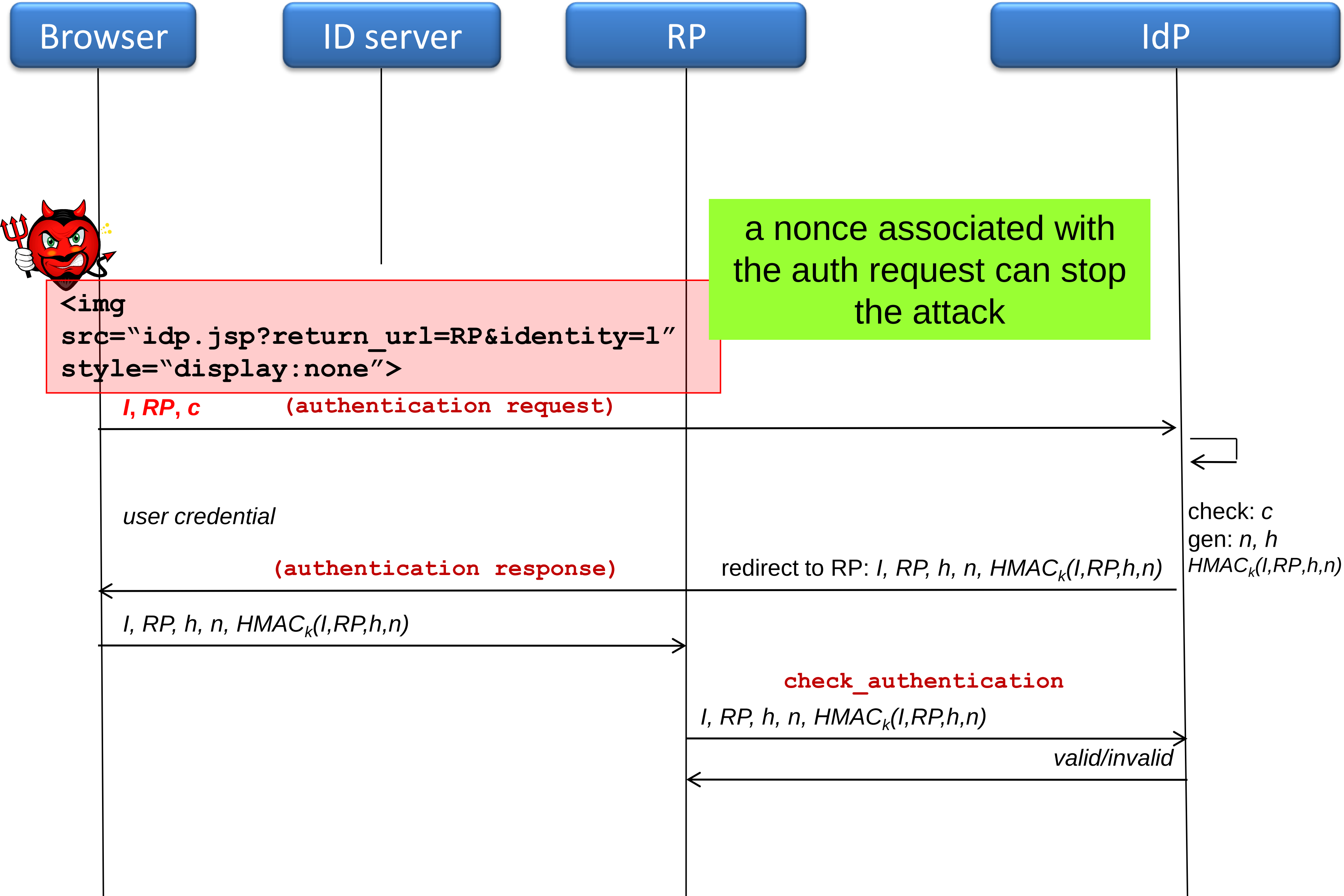
authentication response acts as an one-time access token to an RP, but

- authentication response is not bound to a specific authentication request (non-associate)
- authentication request is not bound to a specific login request
- login request is not bound to the browser session

attack vectors

- CSRF
  - single sign-on (SSO) CSRF (force victim to login)
    - HTTP GET Auth Request CSRF[Web poster, Web attacker]
    - HTTP POST Login CSRF [Web attacker]
    - HTTP GET Login CSRF [Web poster, Web attacker]
  - account profile CSRF [Web poster, Web attacker]
  - login CSRF (login as attacker) [Web attacker]
- authentication response interception
  - impersonation [Network attacker]
  - replay attack [Network attacker]

SSO CSRF: HTTP GET Auth Request



Browser

ID server

RP

IdP

a nonce associated with the auth request can stop the attack

```
<img
src="idp.jsp?return_url=RP&identity=1"
style="display:none">
```

I, RP, c `(authentication request)`

user credential

check: *c*
gen: *n, h*
$HMAC_k(I,RP,h,n)$

`(authentication response)`

redirect to RP: $I, RP, h, n, HMAC_k(I,RP,h,n)$

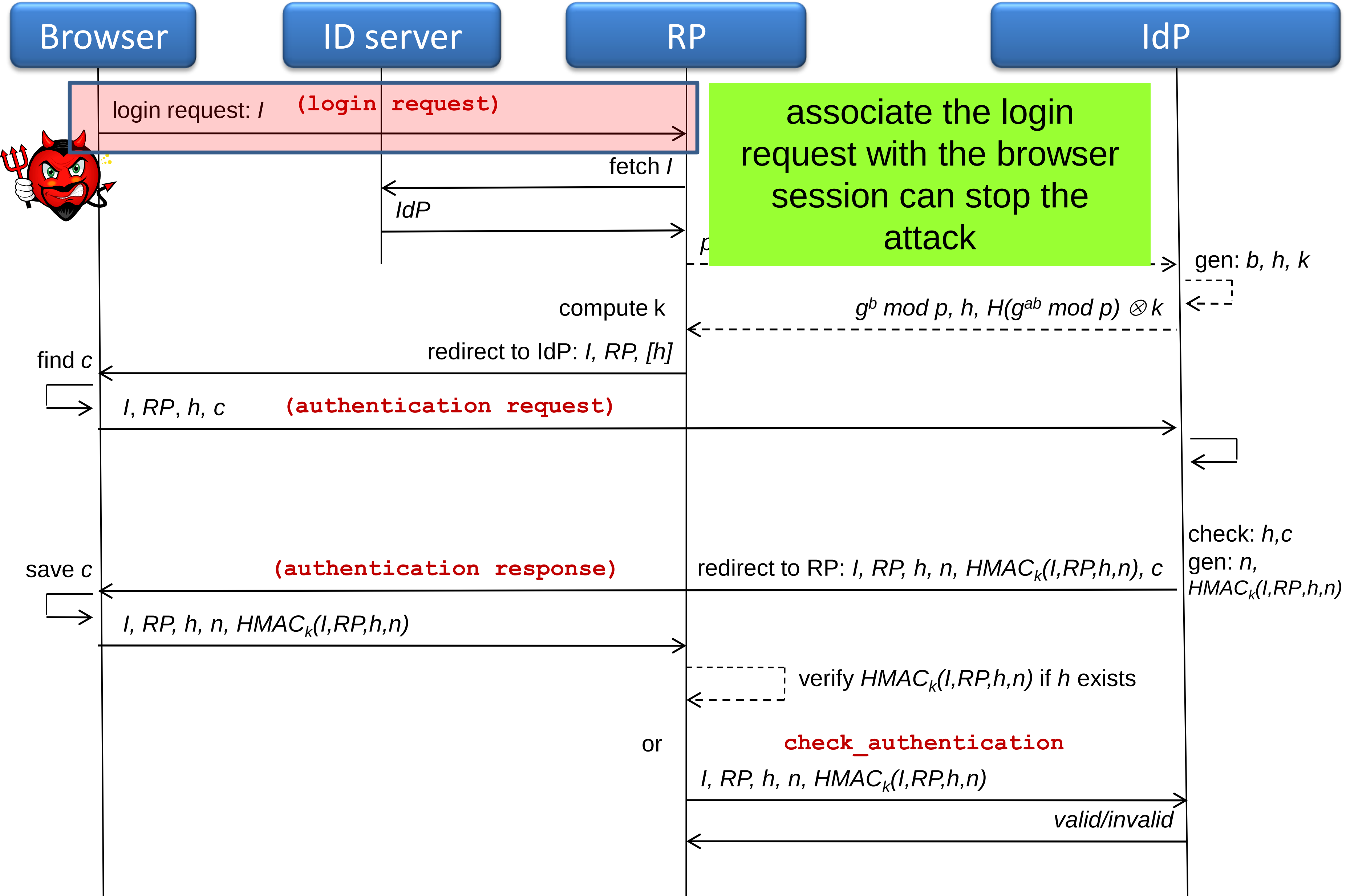
$I, RP, h, n, HMAC_k(I,RP,h,n)$

`check_authentication`

$I, RP, h, n, HMAC_k(I,RP,h,n)$

valid/invalid

SSO CSRF : HTTP POST/GET login



Browser

ID server

RP

IdP

login request: *I* **(login request)**

fetch *I*

IdP

p

associate the login request with the browser session can stop the attack

gen: *b, h, k*

compute k

g^b *mod p, h, H(*g^{ab} *mod p)* ⊗ *k*

redirect to IdP: *I, RP, [h]*

find *c*

I, RP, h, c **(authentication request)**

check: *h,c*

gen: *n*,

$HMAC_k(I,RP,h,n)$

save *c*

(authentication response)

redirect to RP: $I, RP, h, n, HMAC_k(I,RP,h,n), c$

$I, RP, h, n, HMAC_k(I,RP,h,n)$

verify $HMAC_k(I,RP,h,n)$ if *h* exists

or

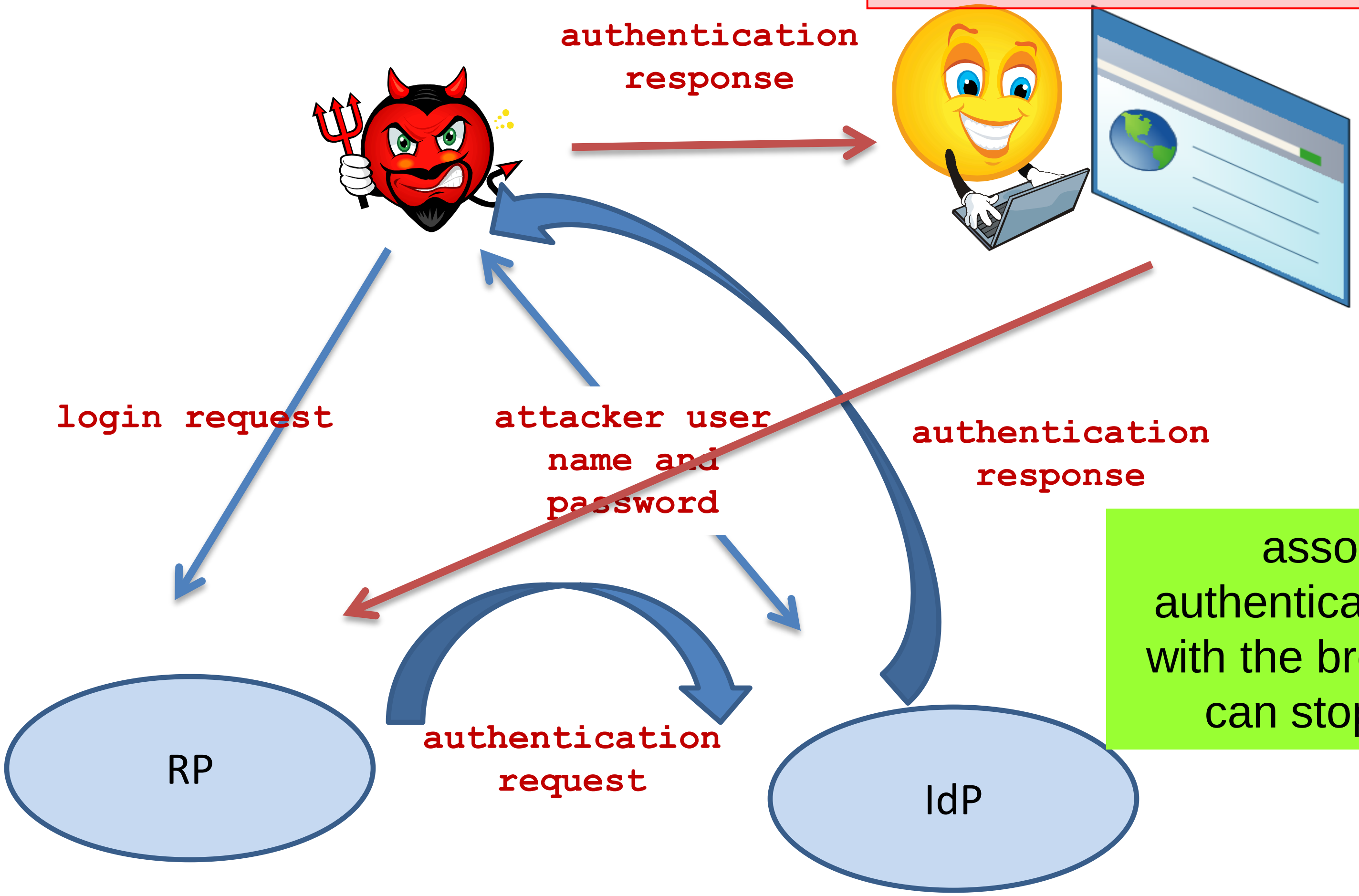
check_authentication

$I, RP, h, n, HMAC_k(I,RP,h,n)$

valid/invalid

login CSRF: login as the attacker

`<img src=“auth response” style=“display:none”>`



authentication response

login request

attacker user name and password

authentication response

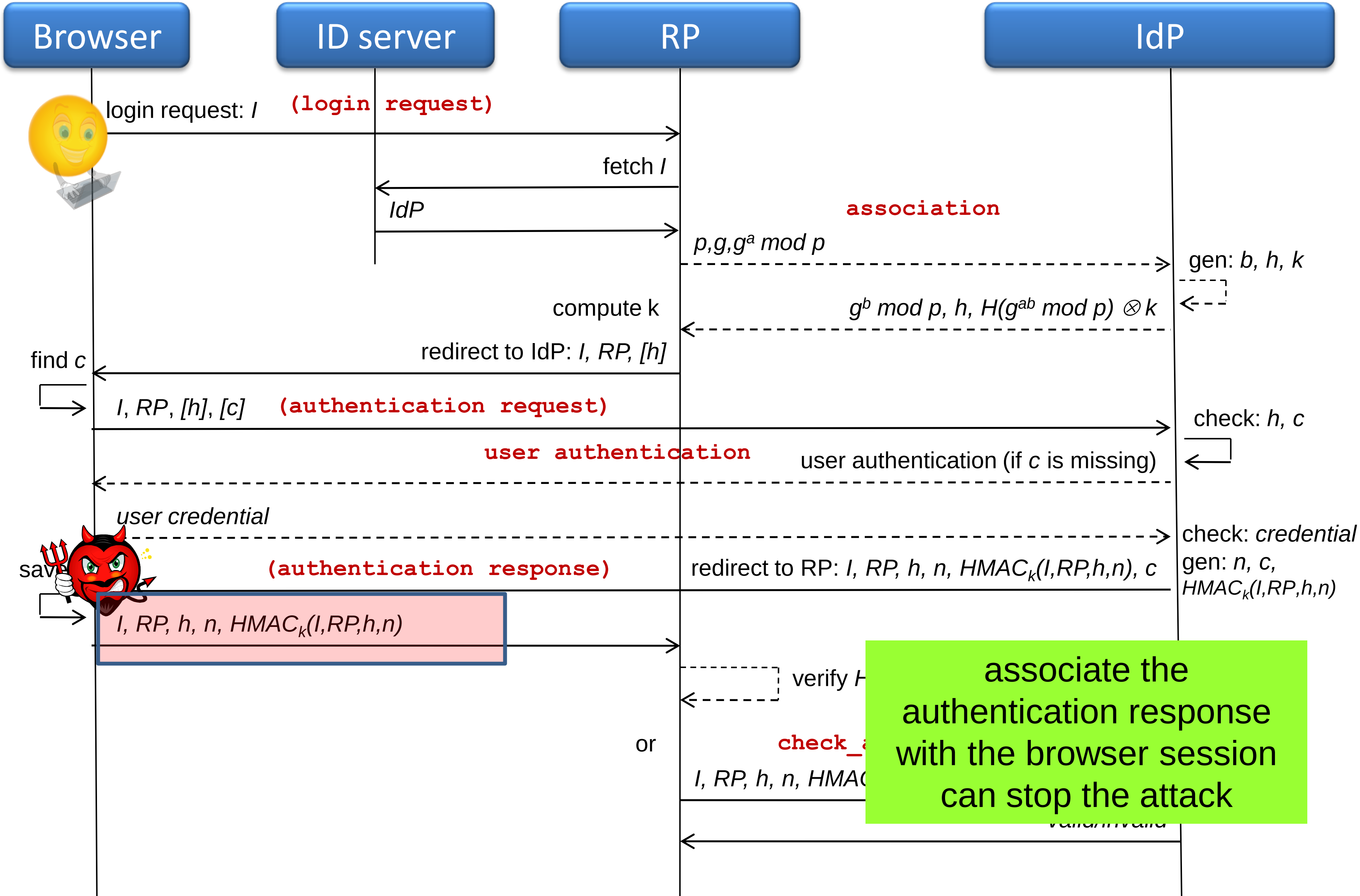
authentication request

RP

IdP

associate the authentication response with the browser session can stop the attack

impersonation and replay attack



Browser

ID server

RP

IdP

login request: *I*

(login request)

fetch *I*

IdP

association

$p, g, g^a \bmod p$

gen: b, h, k

compute k

$g^b \bmod p, h, H(g^{ab} \bmod p) \otimes k$

redirect to IdP: *I, RP, [h]*

find *c*

I, *RP*, *[h]*, *[c]*

(authentication request)

check: *h, c*

user authentication

user authentication (if *c* is missing)

user credential

check: *credential*

gen: n, c, $HMAC_k(I,RP,h,n)$

sav

(authentication response)

redirect to RP: $I, RP, h, n, HMAC_k(I,RP,h,n), c$

$I, RP, h, n, HMAC_k(I,RP,h,n)$

verify *H*

or

check_a

I, RP, h, n, HMAC

valid/invalid

associate the authentication response with the browser session can stop the attack

attack summary

- CSRF attacks
  - single-sign-on CSRF (force victim to login) (70%)
    - HTTP GET Auth Request (25%)
    - HTTP POST Login (50%)
    - HTTP GET Login (35%)
  - account profile CSRF (50%)
  - login CSRF (login as attacker) (73%)
- authentication response interception
  - impersonation (67%)
  - replay attack (6%)

agenda

- background of OpenID
- approach and evaluation result
- countermeasures

countermeasure

- when a new browser session initialized
  - RP generates a nonce N = HMAC(browser session id)
  - issues a new cookie C_N = N
  - append a parameter P_N=N to the OpenID login form
- when receive a login request
  - check if $P_N = C_N$ and C_N = HMAC(browser session id)
  - initiate a new authentication request
  - append a parameter R_N=N to the **`return_to`** URL
- when receive an authentication response
  - check if $R_N = C_N$ and C_N = HMAC(browser session id)

characteristics of countermeasure

- compatible with existing OpenID
- do not require any additional storage on RP
- would not reveal browser session id
- protect from cookie overwrite

future work

- evaluate more RPs
- apply our methodology to other Web single sign-on protocol
  - Facebook connect
  - Microsoft Live ID

OpenID Security Analysis and Evaluation

san-tsai sun <santsais@ece.ubc.ca>

Department of Electrical and Computer Engineering
Laboratory for Education and Research in Secure Systems Engineering (**LERSSE**)