Keys to the Cloud: Formal Analysis and Concrete Attacks on Encrypted Web Storage

C. Bansal K. Bhargavan S. Maffeis
A. Delignat-Lavaud*

PROSECCO, INRIA Paris-Rocquencourt

March 18, 2013

In-Browser Encryption

Sensitive data on the Web

- Ever more private data online
- TLS protects data over the wire
- Data at rest on untrusted cloud servers at risk
- Attack on server = disaster

Solution

Client-side encryption

In-Browser Encryption

Sensitive data on the Web

- Ever more private data online
- TLS protects data over the wire
- Data at rest on untrusted cloud servers at risk
- Attack on server = disaster

Solution

Client-side encryption

In-Browser Encryption

Sensitive data on the Web

- Ever more private data online
- TLS protects data over the wire
- Data at rest on untrusted cloud servers at risk
- Attack on server = disaster

Solution

Client-side encryption

In-Browser Encryption

Sensitive data on the Web

- Ever more private data online
- TLS protects data over the wire
- Data at rest on untrusted cloud servers at risk
- Attack on server = disaster

Solution

Client-side encryption

In-Browser Encryption

Sensitive data on the Web

- Ever more private data online
- TLS protects data over the wire
- Data at rest on untrusted cloud servers at risk
- Attack on server = disaster

Solution

Client-side encryption

In-Browser Encryption

Name	Key Derivation	Encryption	Integrity	Metadata Integrity	Sharing
Wuala	PBKDF2	AES, RSA	HMAC	✓	✓(PKI)
SpiderOak	PBKDF2	AES, RSA	HMAC	✓	✓
Mega	AES-ECB	AES, RSA	CBC-MAC	✓	✓(PKI)
BoxCryptor	PBKDF2	AES	None	✗	✗
CloudFogger	PBKDF2	AES, RSA	None	✗	✓(PKI)
1Password	PBKDF2-SHA1	AES	None	✗	✓
LastPass	PBKDF2-SHA256	AES, RSA	None	✗	✓
PassPack	SHA256	AES	None	✓	✓
RoboForm	PBKDF2	AES, DES	None	✗	✓
Clipperz	SHA256	AES	SHA256	✓	✗
ConfiChair	PBKDF2	RSA, AES	SHA1	✓	✓(PKI)
Helios	N/A	El Gamal	SHA256	Zero-Knowledge Proof	N/A

Table : Example encrypted cloud storage websites

In-Browser Encryption

The pros

- Data stored encrypted on cloud
- Server never gets key
- Privacy selling point
- Avoid server-side computations

The cons

- Web client is even more critical
- ...but much easier to attack
- in-browser cryptography highly controversial

In-Browser Encryption

The pros

- Data stored encrypted on cloud
- Server never gets key
- Privacy selling point
- Avoid server-side computations

The cons

- Web client is even more critical
- ...but much easier to attack
- in-browser cryptography highly controversial

In-Browser Encryption

The pros

- Data stored encrypted on cloud
- Server never gets key
- Privacy selling point
- Avoid server-side computations

The cons

- Web client is even more critical
- ...but much easier to attack
- in-browser cryptography highly controversial

In-Browser Encryption

The pros

- Data stored encrypted on cloud
- Server never gets key
- Privacy selling point
- Avoid server-side computations

The cons

- Web client is even more critical
- ...but much easier to attack
- in-browser cryptography highly controversial

In-Browser Encryption

The pros

- Data stored encrypted on cloud
- Server never gets key
- Privacy selling point
- Avoid server-side computations

The cons

- Web client is even more critical
- ...but much easier to attack
- in-browser cryptography highly controversial

In-Browser Encryption

The pros

- Data stored encrypted on cloud
- Server never gets key
- Privacy selling point
- Avoid server-side computations

The cons

- Web client is even more critical
- ...but much easier to attack
- in-browser cryptography highly controversial

In-Browser Encryption

The pros

- Data stored encrypted on cloud
- Server never gets key
- Privacy selling point
- Avoid server-side computations

The cons

- Web client is even more critical
- ...but much easier to attack
- in-browser cryptography highly controversial

In-Browser Encryption

JavaScript cryptography

- Trusted code delivery issue
- Lack of integrity of computations
- Key management

In-Browser Encryption

JavaScript cryptography

- Trusted code delivery issue
- Lack of integrity of computations
- Key management

In-Browser Encryption

JavaScript cryptography

- Trusted code delivery issue
- Lack of integrity of computations
- Key management

Cloud Storage Protocols

Parameters

- u uses browser a to visit website b
- u can derive encryption/MAC keys K/K' and shared login secret $s_{u,b}$ from password p
- b stores for each u a table db of records $m \mapsto (e = Enc_K(x), Mac_{K'}(m, e))$
- a can decrypt and handle its local db synchronized with b using Sync and Update

Cloud Storage Protocols

Parameters

- u uses browser a to visit website b
- u can derive encryption/MAC keys K/K' and shared login secret $s_{u,b}$ from password p
- b stores for each u a table db of records $m \mapsto (e = Enc_K(x), Mac_{K'}(m, e))$
- a can decrypt and handle its local db synchronized with b using Sync and Update

Cloud Storage Protocols

Parameters

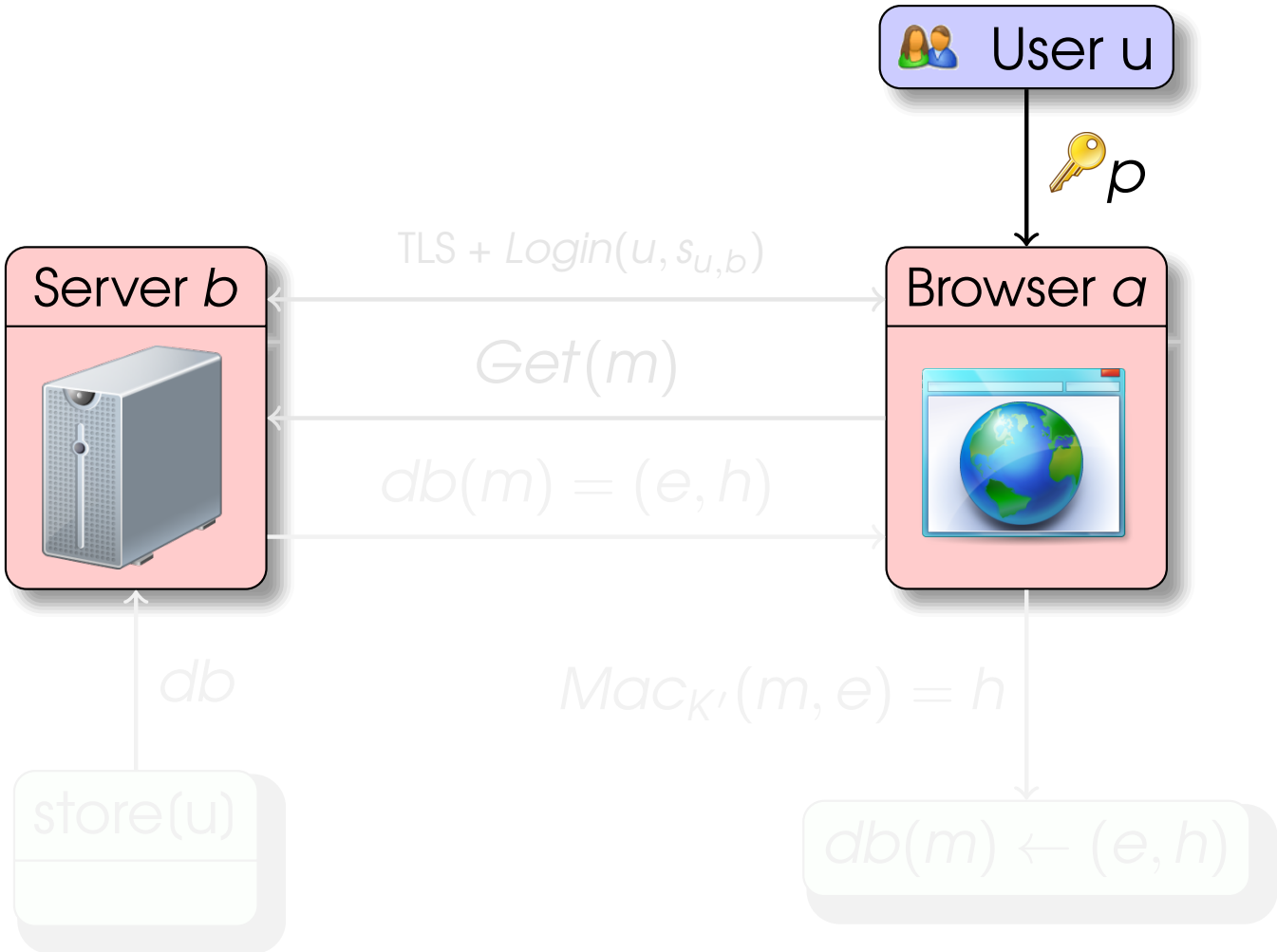
- u uses browser a to visit website b
- u can derive encryption/MAC keys K/K' and shared login secret $s_{u,b}$ from password p
- b stores for each u a table db of records $m \mapsto (e = Enc_K(x), Mac_{K'}(m, e))$
- a can decrypt and handle its local db synchronized with b using Sync and Update

Cloud Storage Protocols

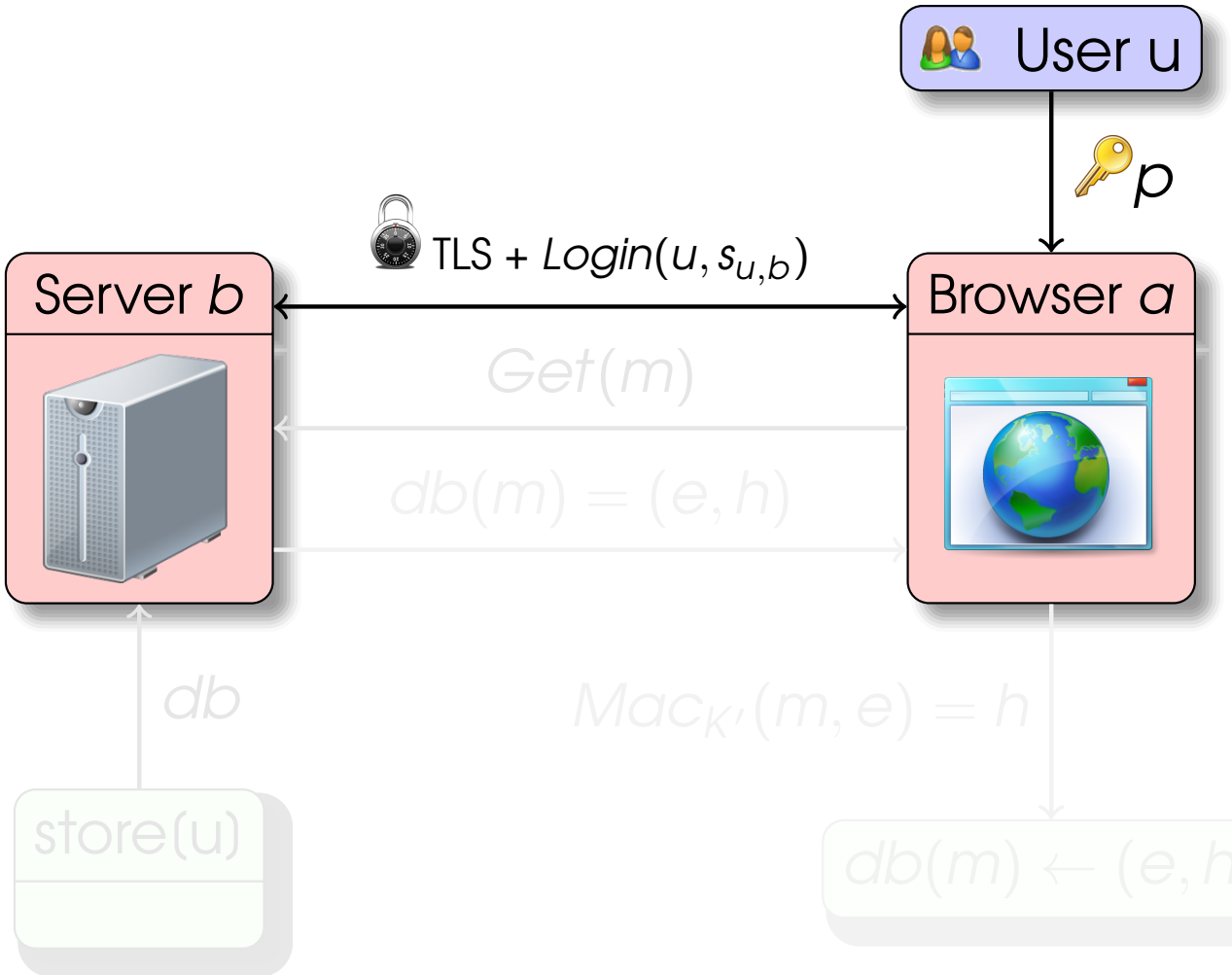
Parameters

- u uses browser a to visit website b
- u can derive encryption/MAC keys K/K' and shared login secret $s_{u,b}$ from password p
- b stores for each u a table db of records $m \mapsto (e = Enc_K(x), Mac_{K'}(m, e))$
- a can decrypt and handle its local db synchronized with b using Sync and Update

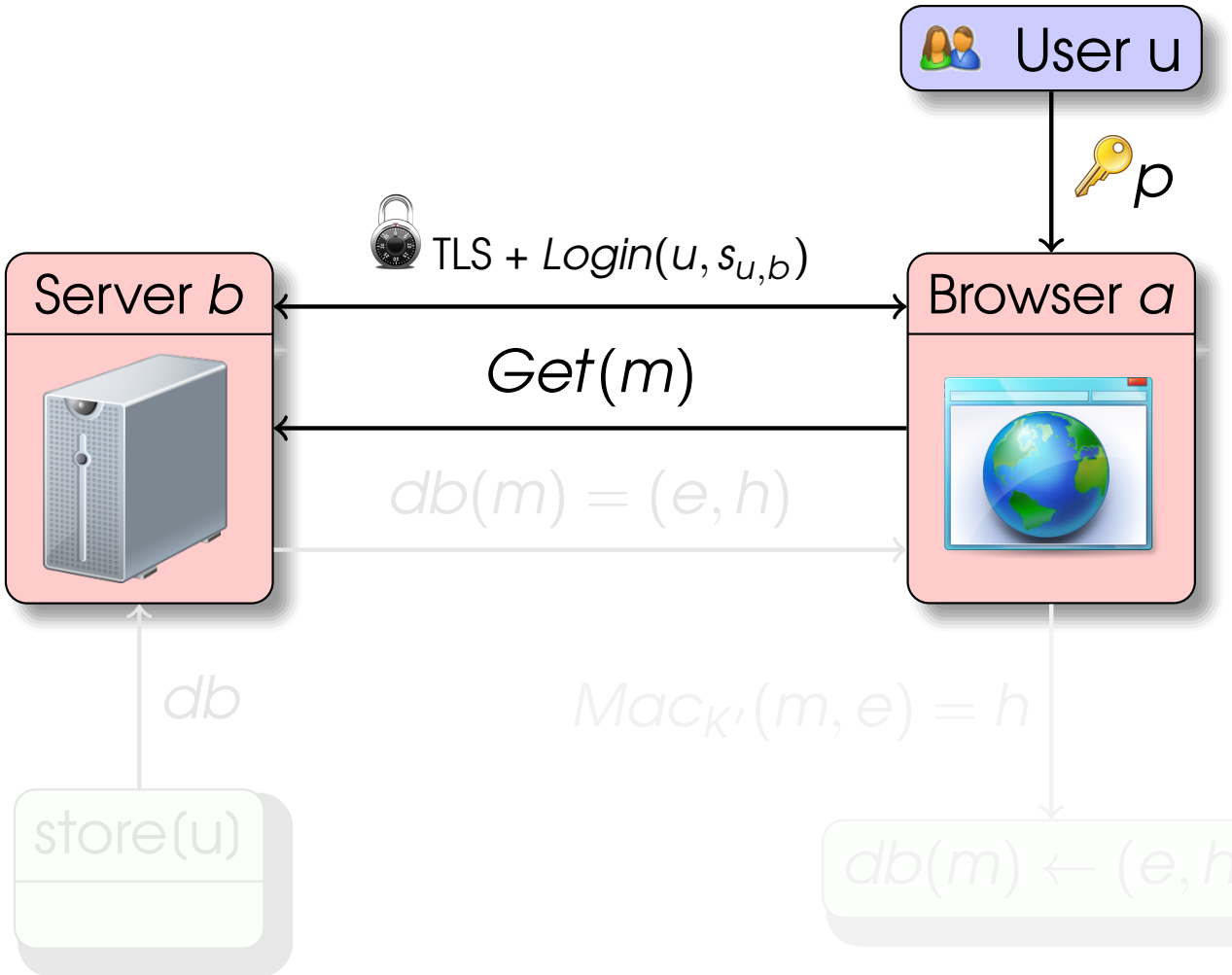
Synchronize Protocol



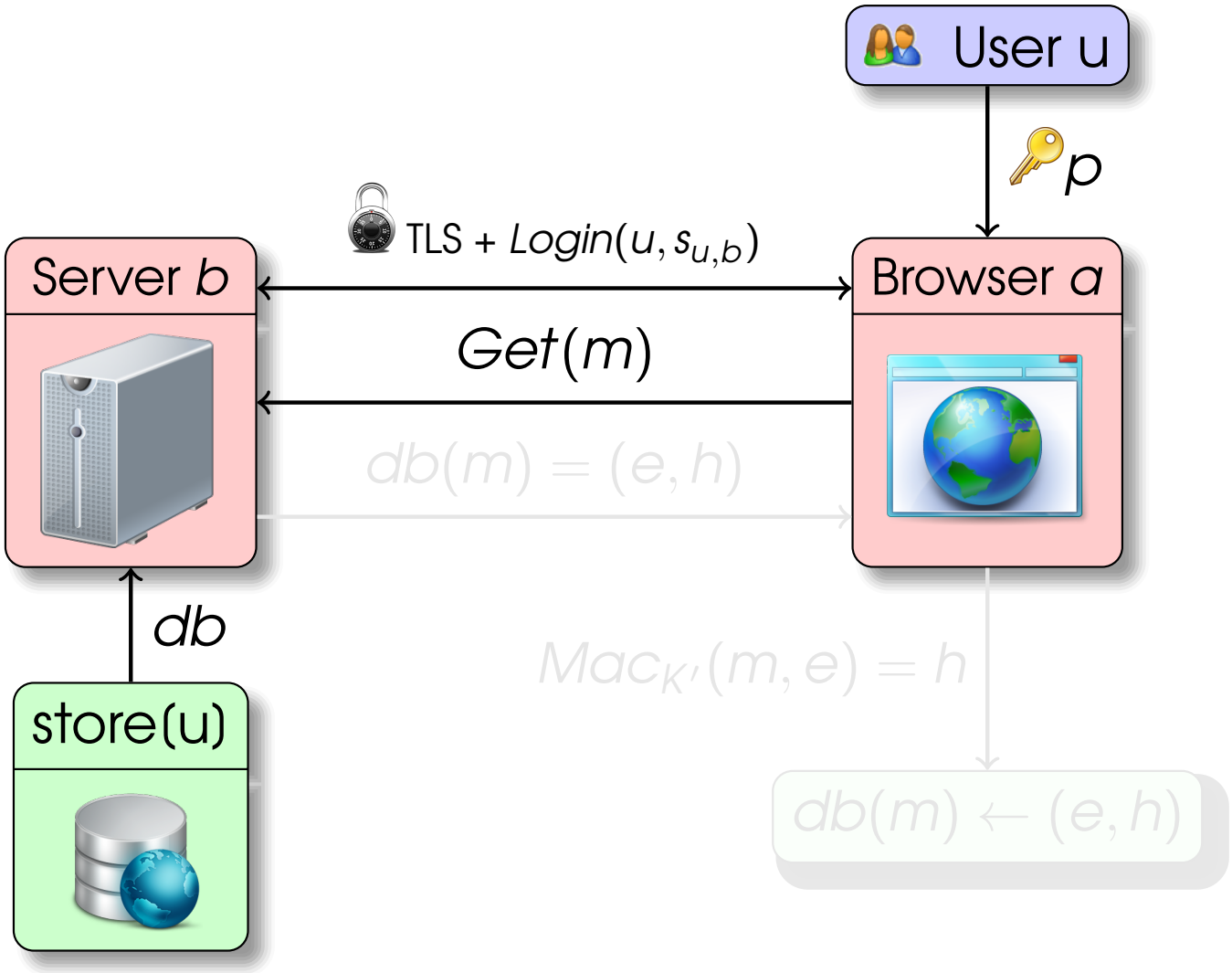
Synchronize Protocol



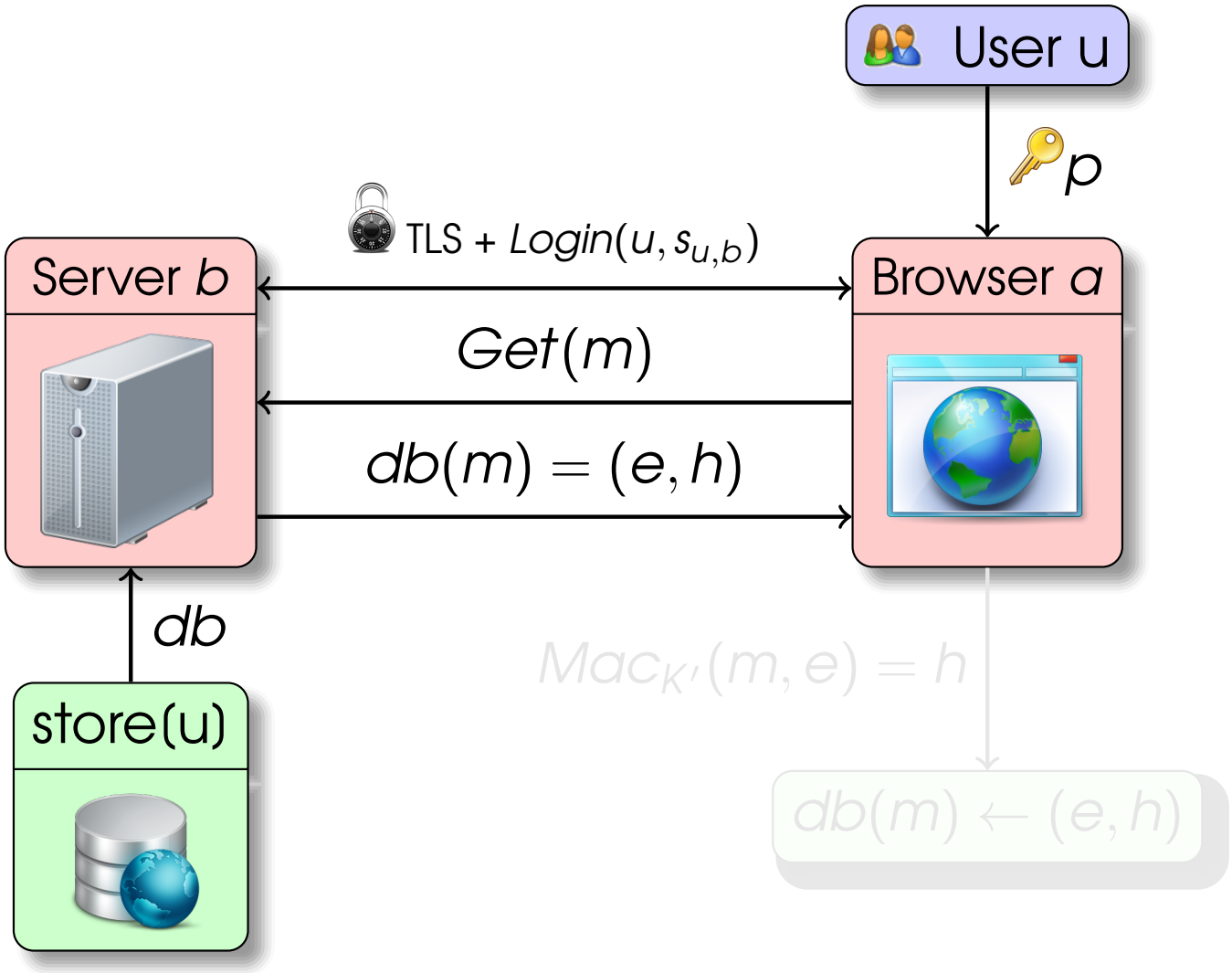
Synchronize Protocol



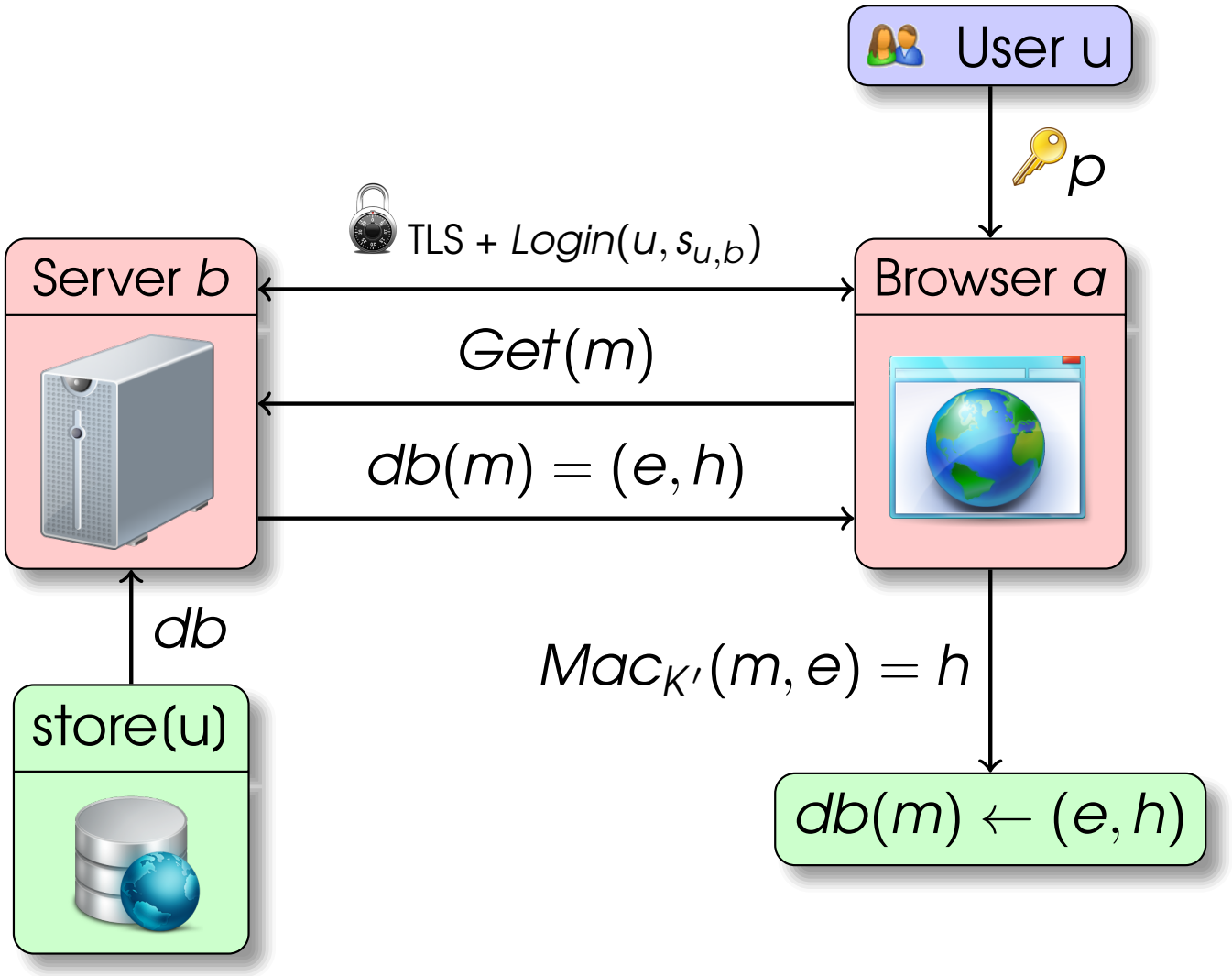
Synchronize Protocol



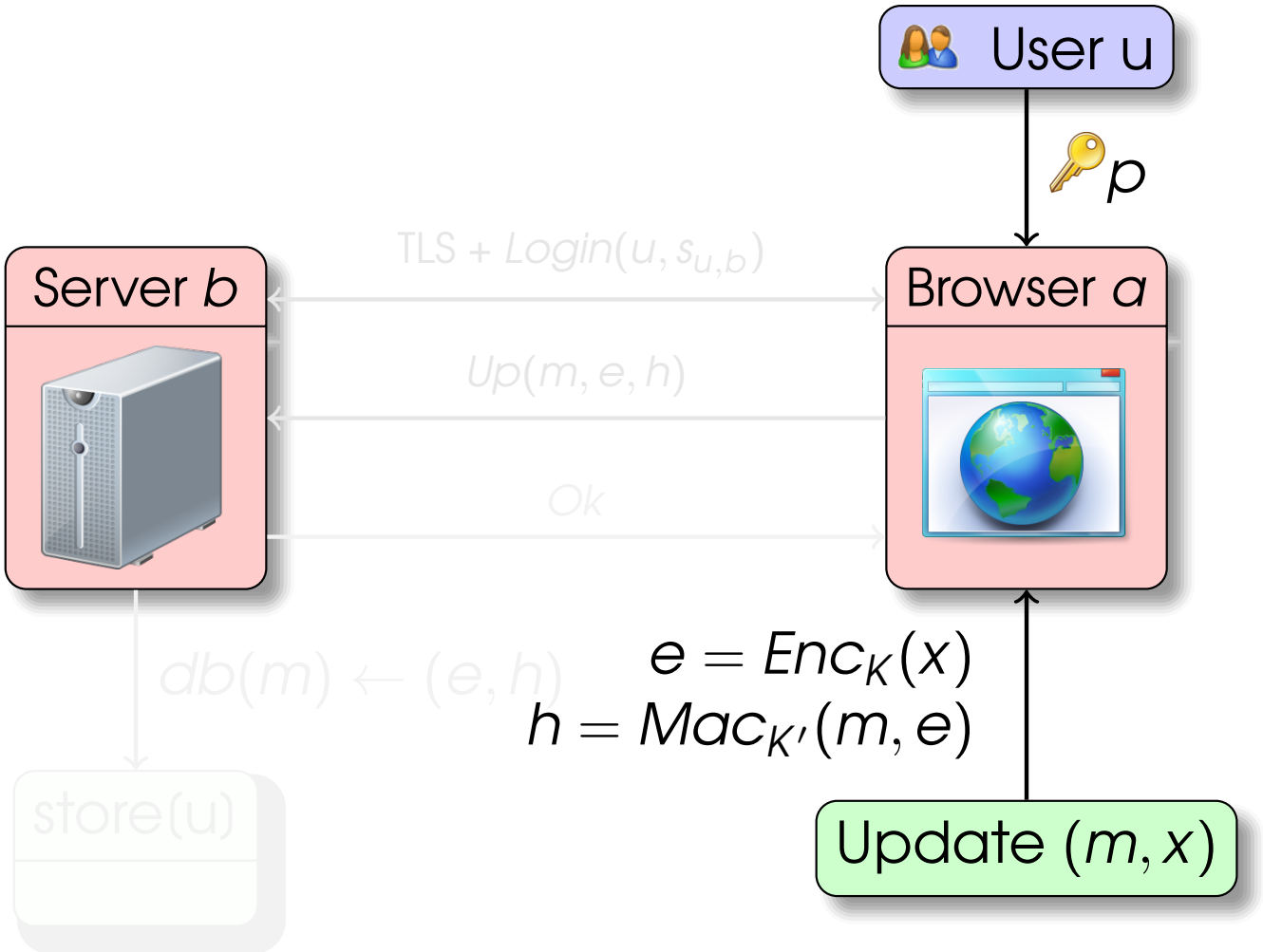
Synchronize Protocol



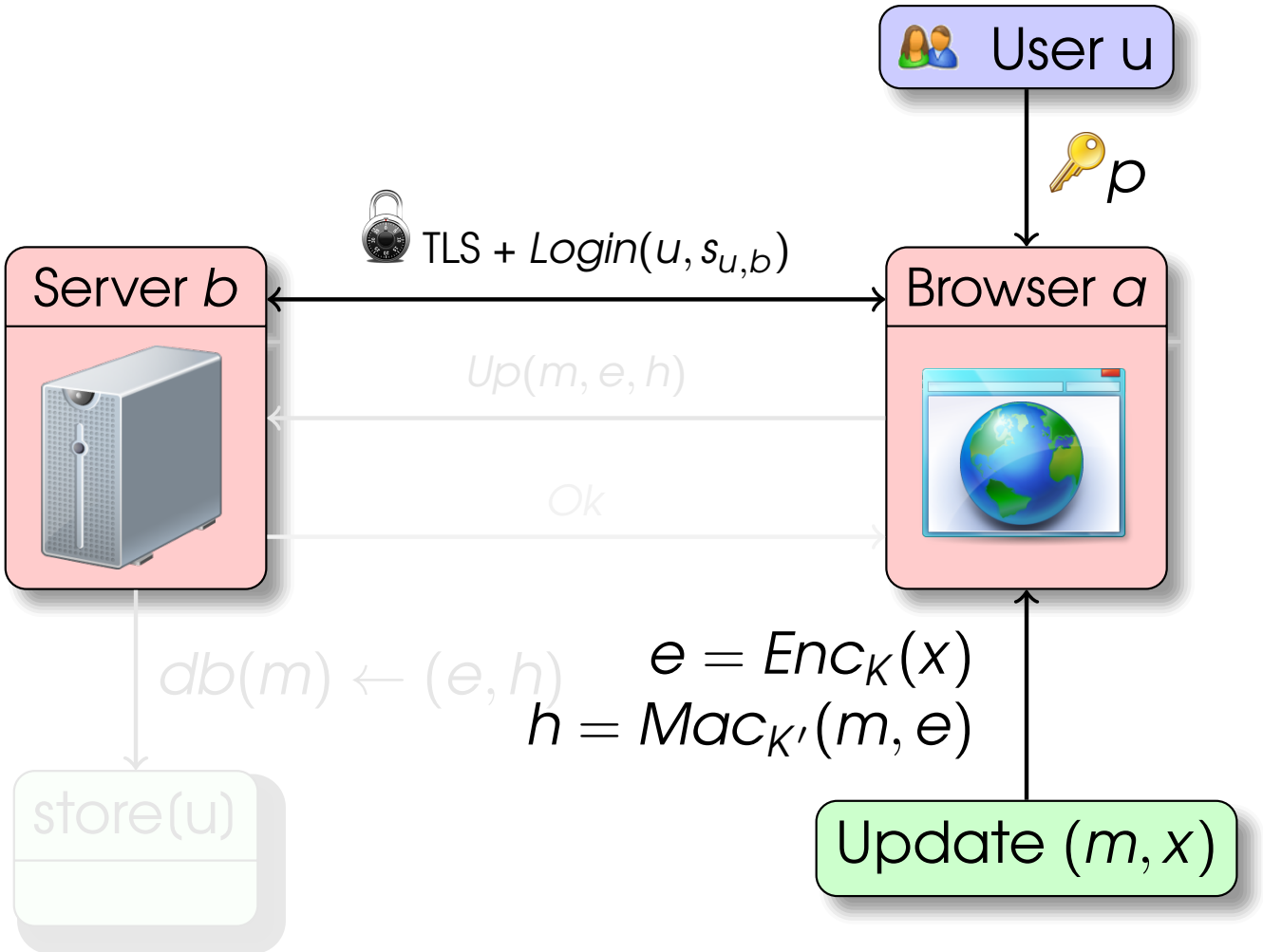
Synchronize Protocol



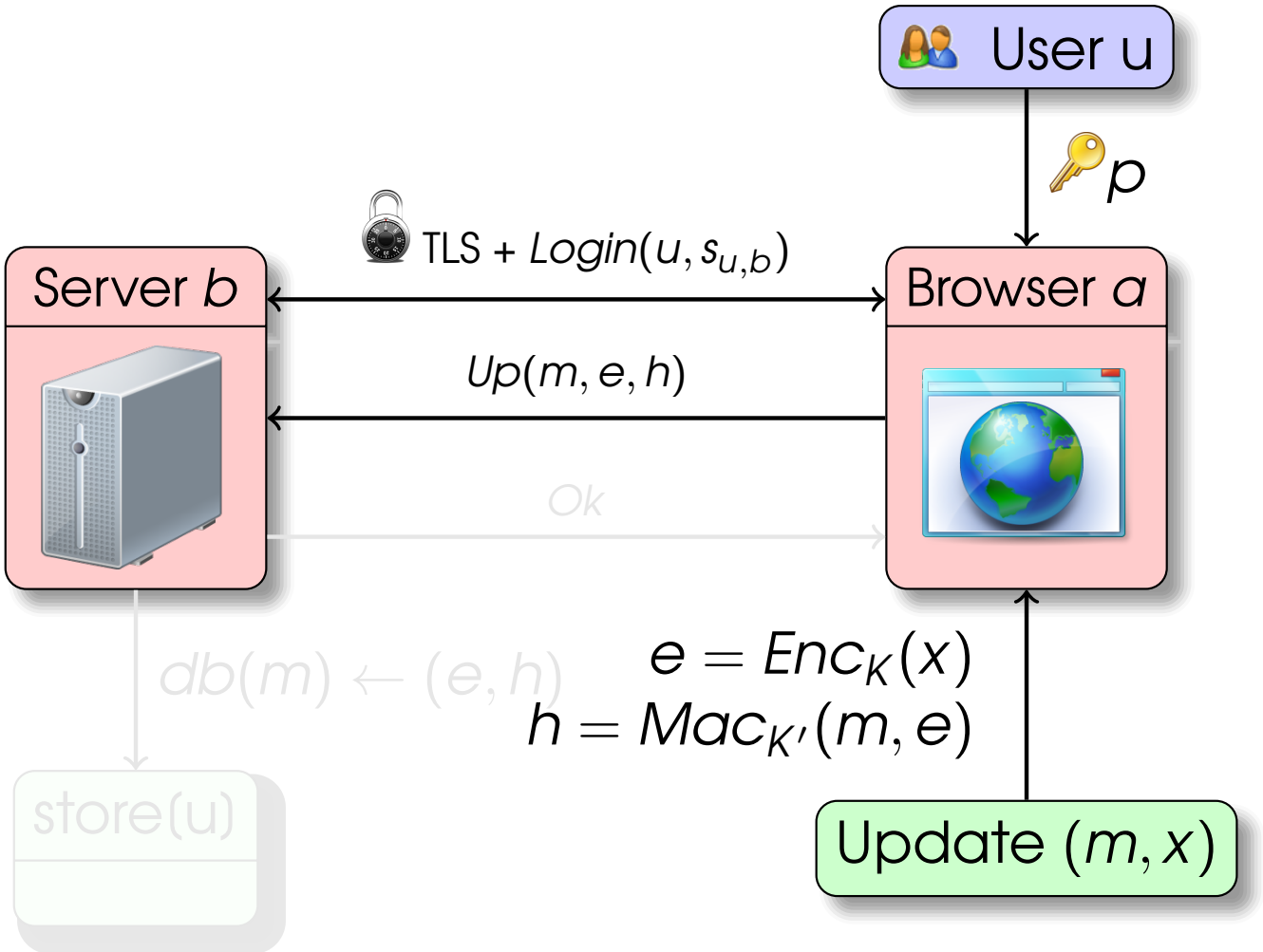
Update protocol



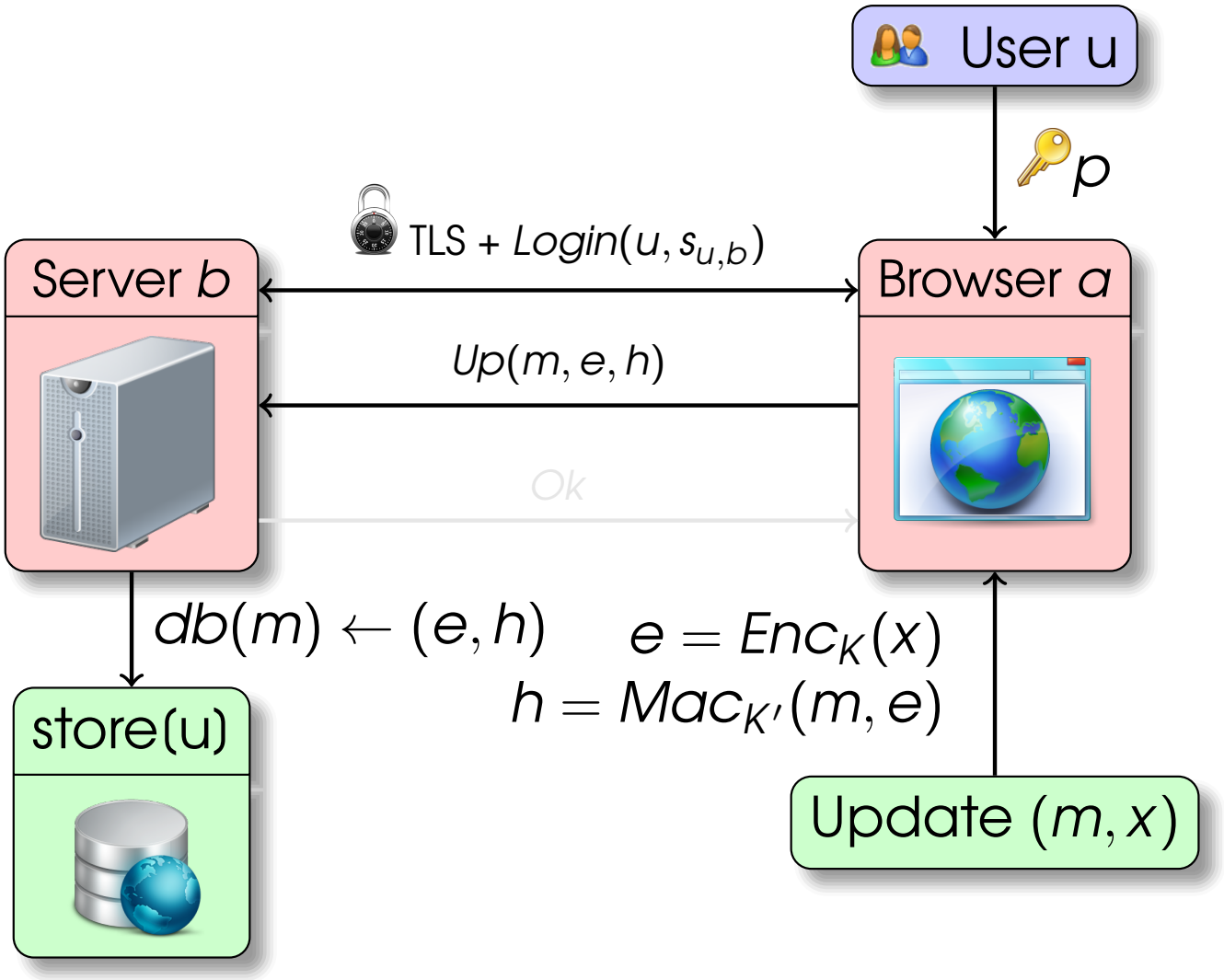
Update protocol



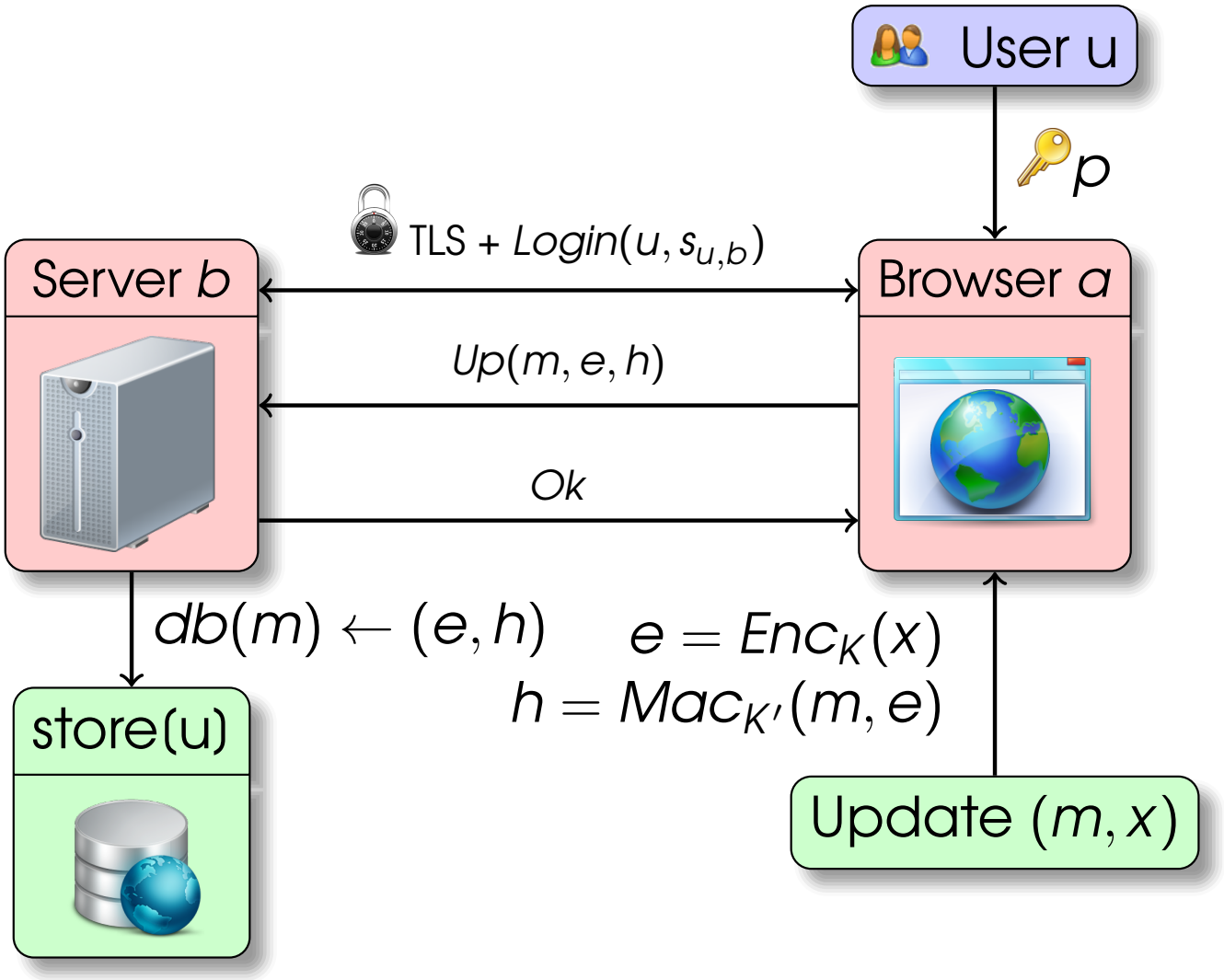
Update protocol



Update protocol



Update protocol



Attacker model

Dolev-Yao attacker

- compromised server
- network attacker
- browser cache inspection

Web login protocol

In the browser

- Must implicitly authenticate user in protocol
- Must avoid asking user for p for every operation
- Where to cache K and K'?

Cookie-based sessions

- Cookie set by server; attached to following requests by browser
- Session table on server indexed by cookie can maintain state across requests
- Cookies sent even if request caused by third party

Web attacker model

- CSRF: URL of private action triggered on user's behalf using session cookies
- XSS: data executed as JavaScript (`eval`, `innerHTML`)
- Insecure session cookie
- Open redirector
- Phishing
- Clickjacking
- ...

Web attacker model

- CSRF: URL of private action triggered on user's behalf using session cookies
- XSS: data executed as JavaScript (`eval`, `innerHTML`)
- Insecure session cookie
- Open redirector
- Phishing
- Clickjacking
- ...

Web attacker model

- CSRF: URL of private action triggered on user's behalf using session cookies
- XSS: data executed as JavaScript (`eval`, `innerHTML`)
- Insecure session cookie
- Open redirector
- Phishing
- Clickjacking
- ...

Web attacker model

- CSRF: URL of private action triggered on user's behalf using session cookies
- XSS: data executed as JavaScript (`eval`, `innerHTML`)
- Insecure session cookie
- Open redirector
- Phishing
- Clickjacking
- ...

Web attacker model

- CSRF: URL of private action triggered on user's behalf using session cookies
- XSS: data executed as JavaScript (`eval`, `innerHTML`)
- Insecure session cookie
- Open redirector
- Phishing
- Clickjacking
- ...

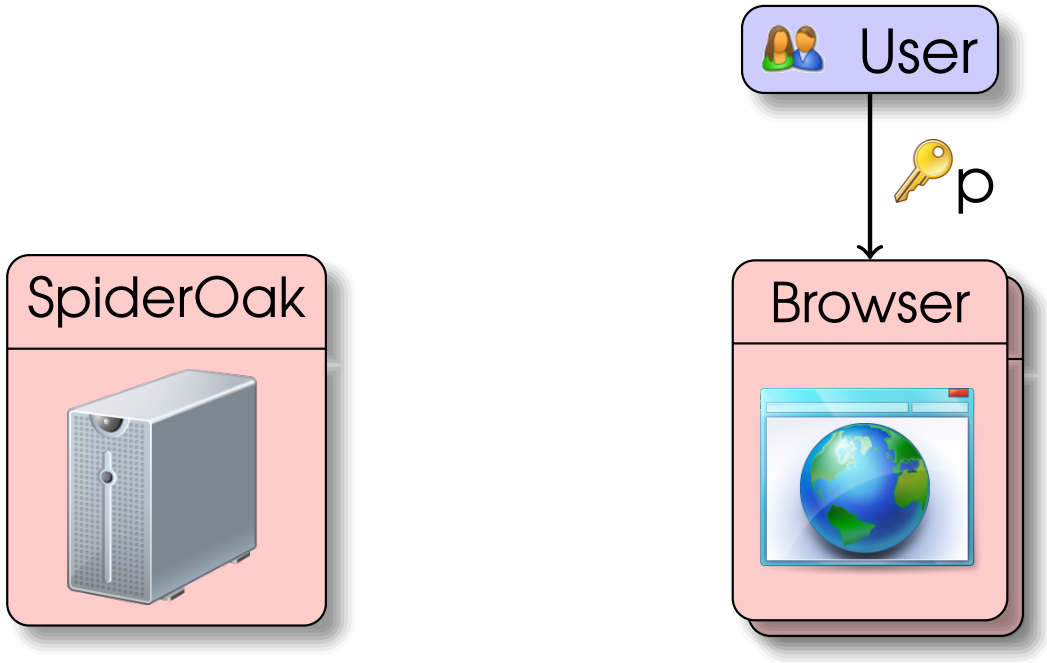
Web attacker model

- CSRF: URL of private action triggered on user's behalf using session cookies
- XSS: data executed as JavaScript (`eval`, `innerHTML`)
- Insecure session cookie
- Open redirector
- Phishing
- Clickjacking
- ...

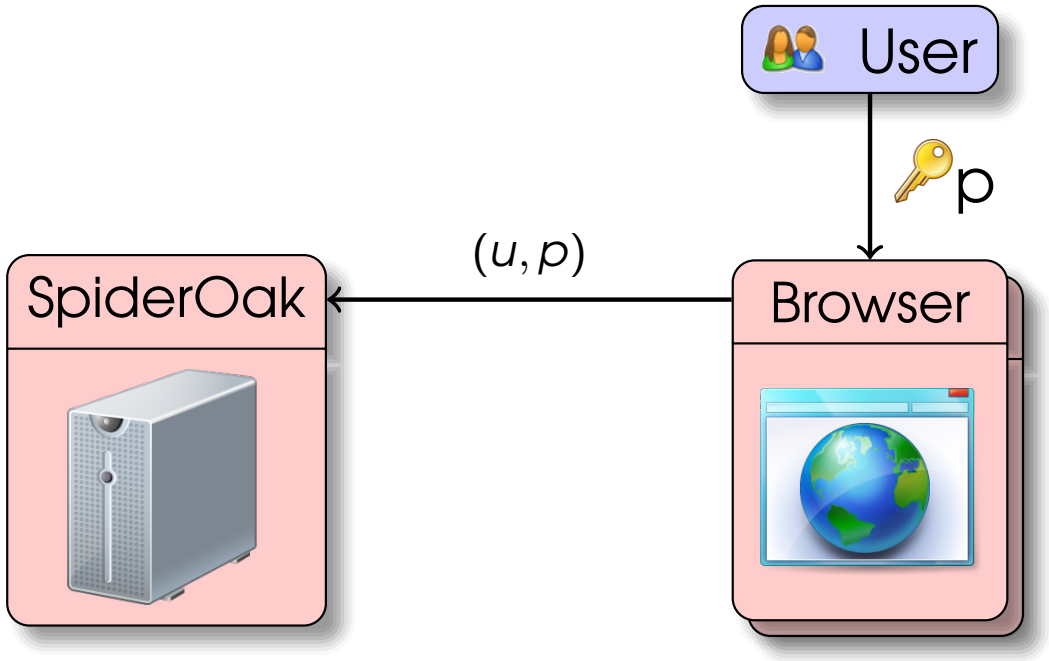
Web attacker model

- CSRF: URL of private action triggered on user's behalf using session cookies
- XSS: data executed as JavaScript (`eval`, `innerHTML`)
- Insecure session cookie
- Open redirector
- Phishing
- Clickjacking
- ...

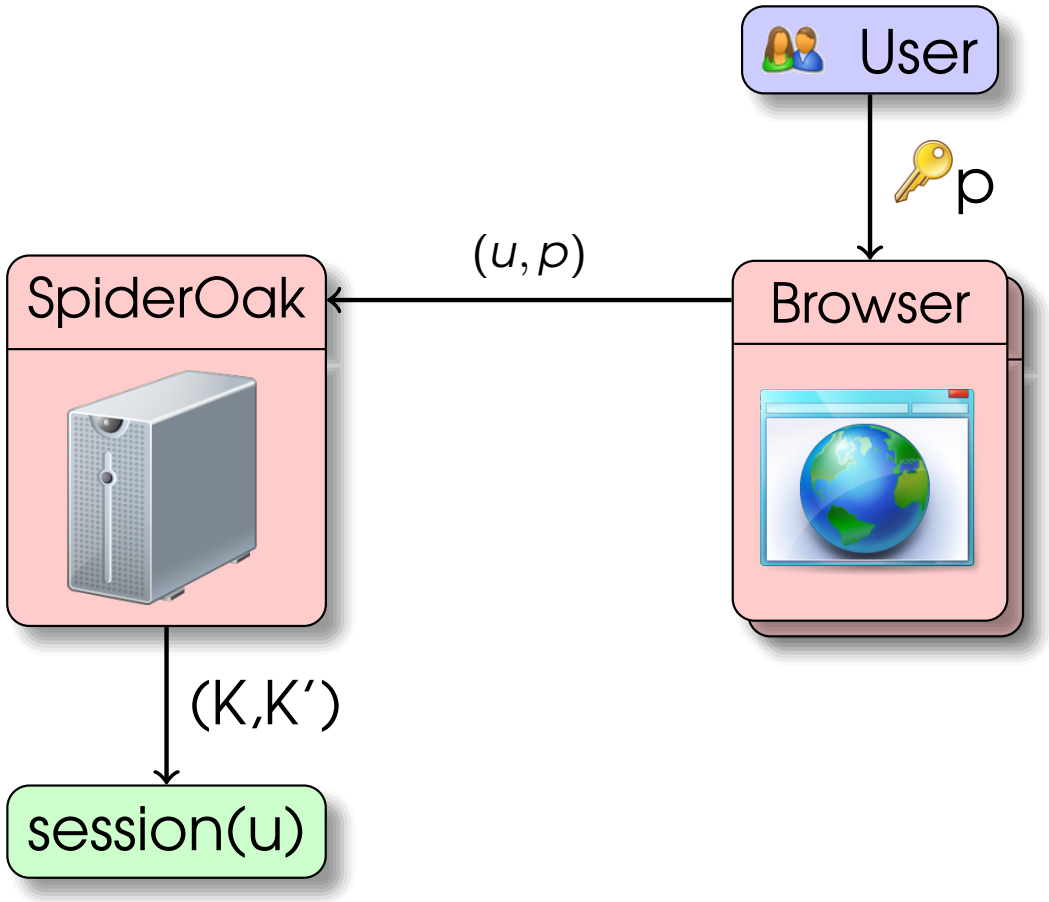
SpiderOak login



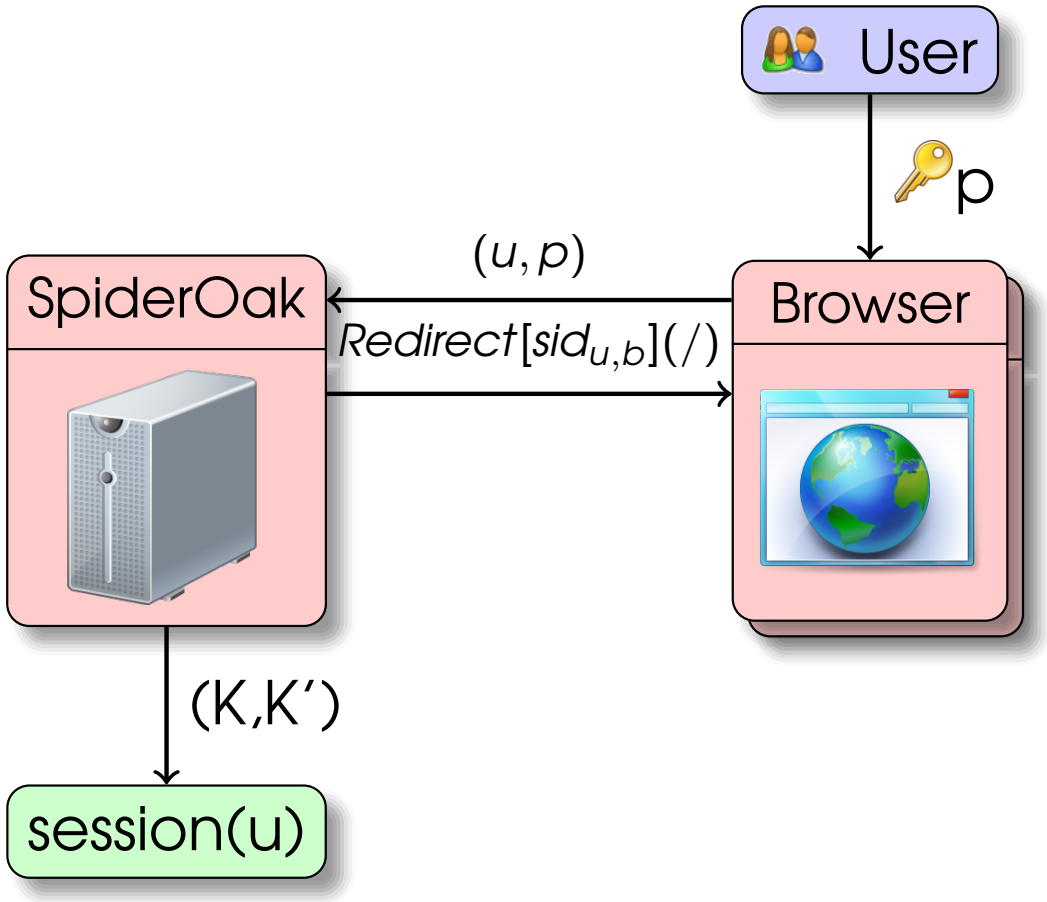
SpiderOak login



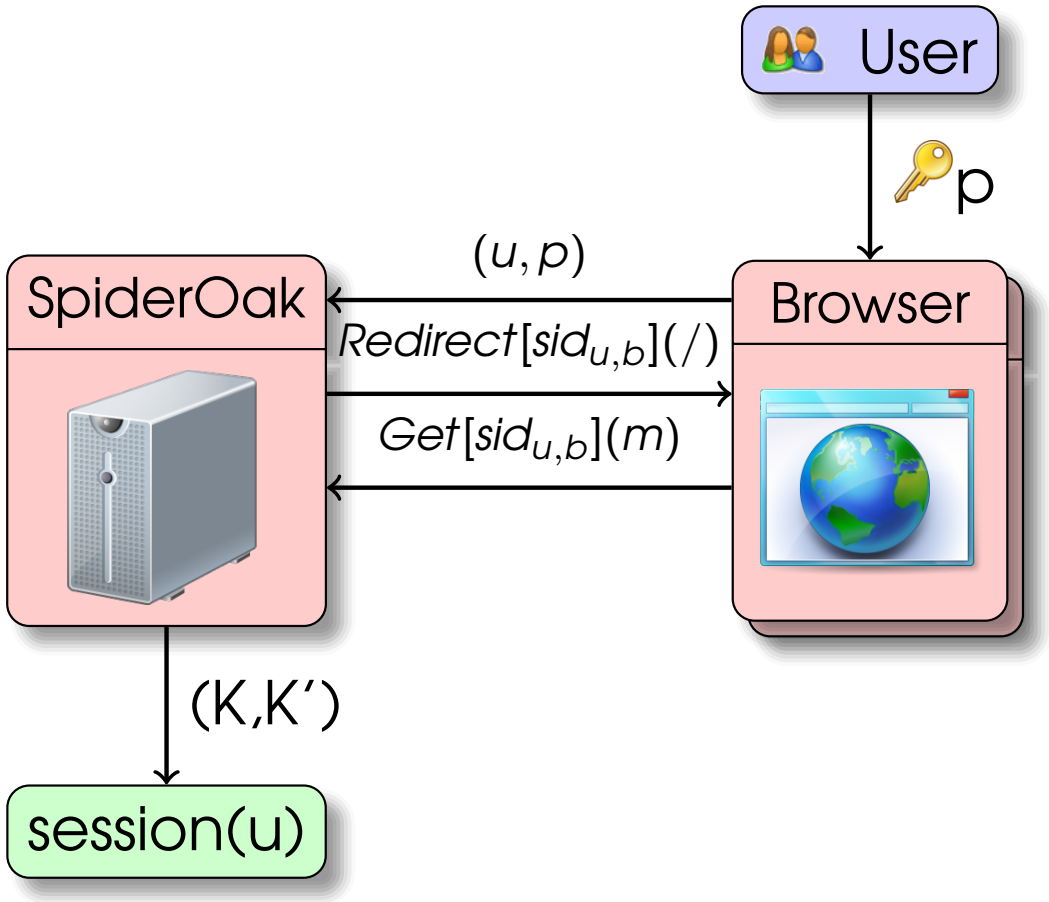
SpiderOak login



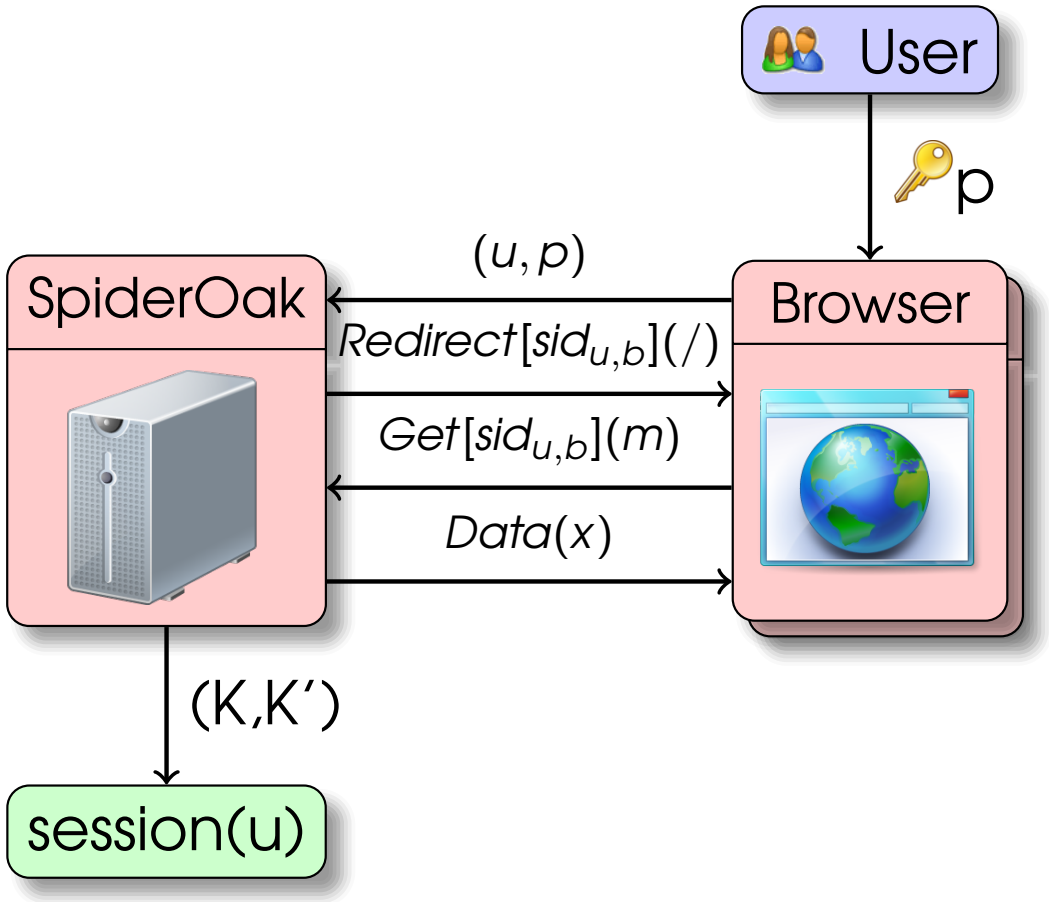
SpiderOak login



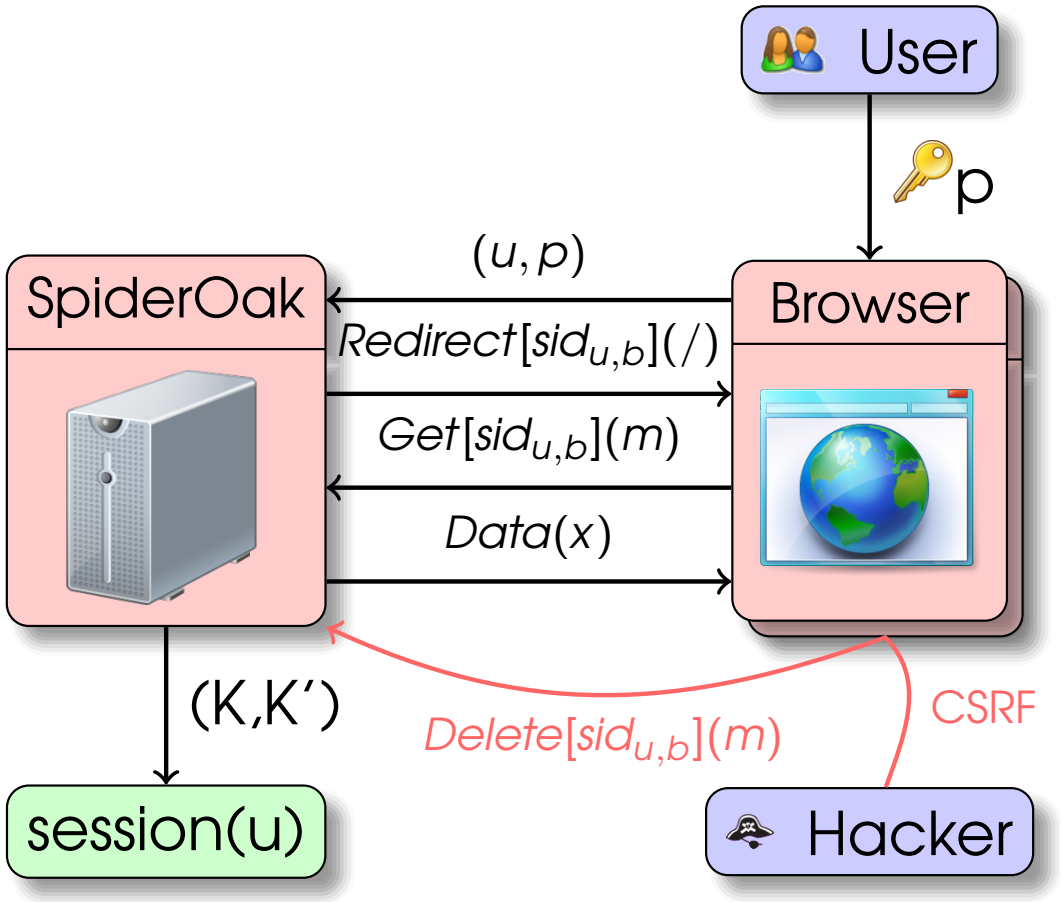
SpiderOak login



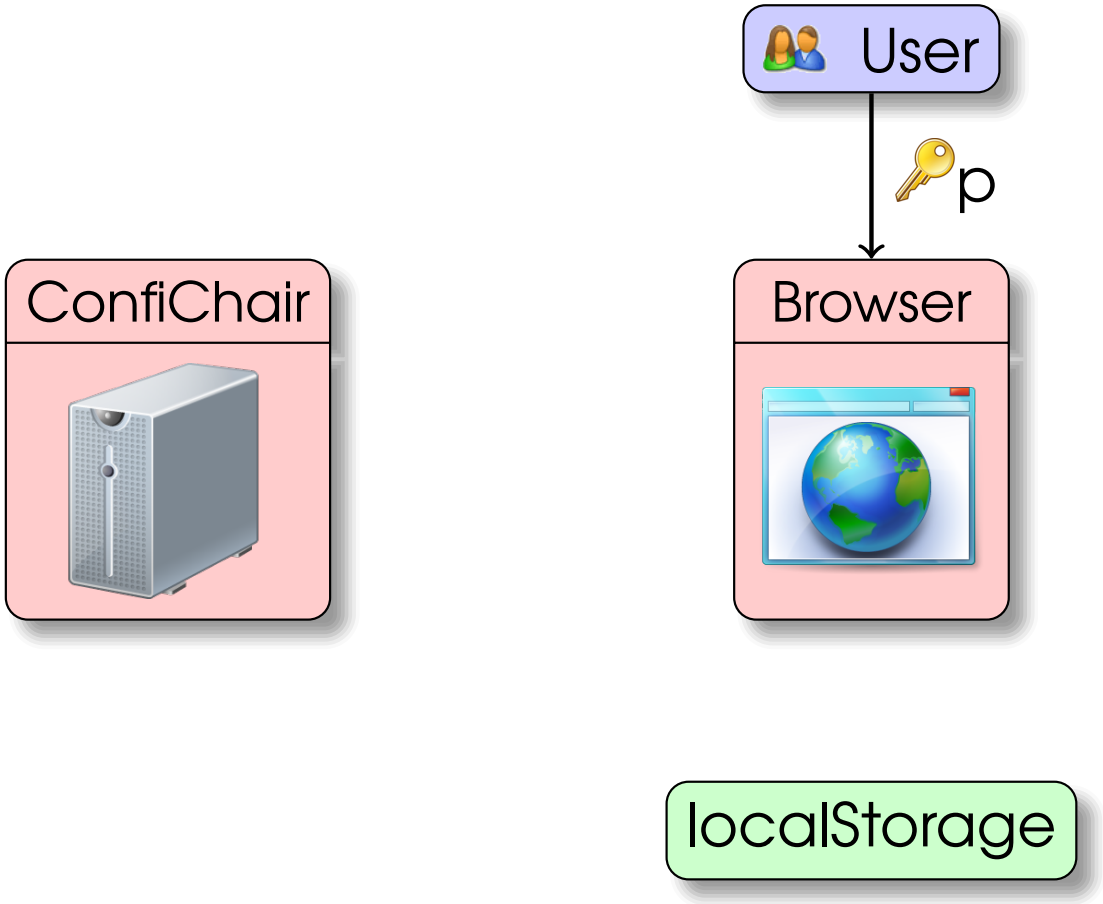
SpiderOak login



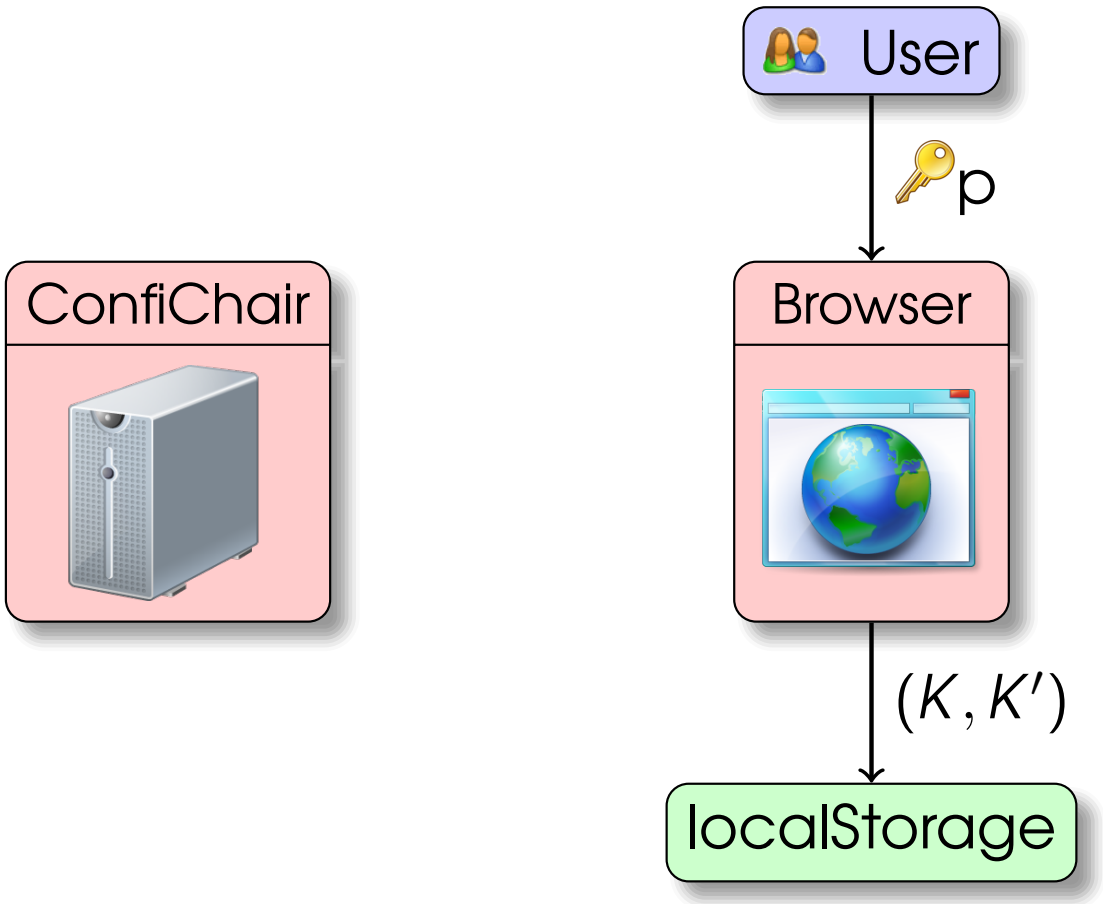
SpiderOak login



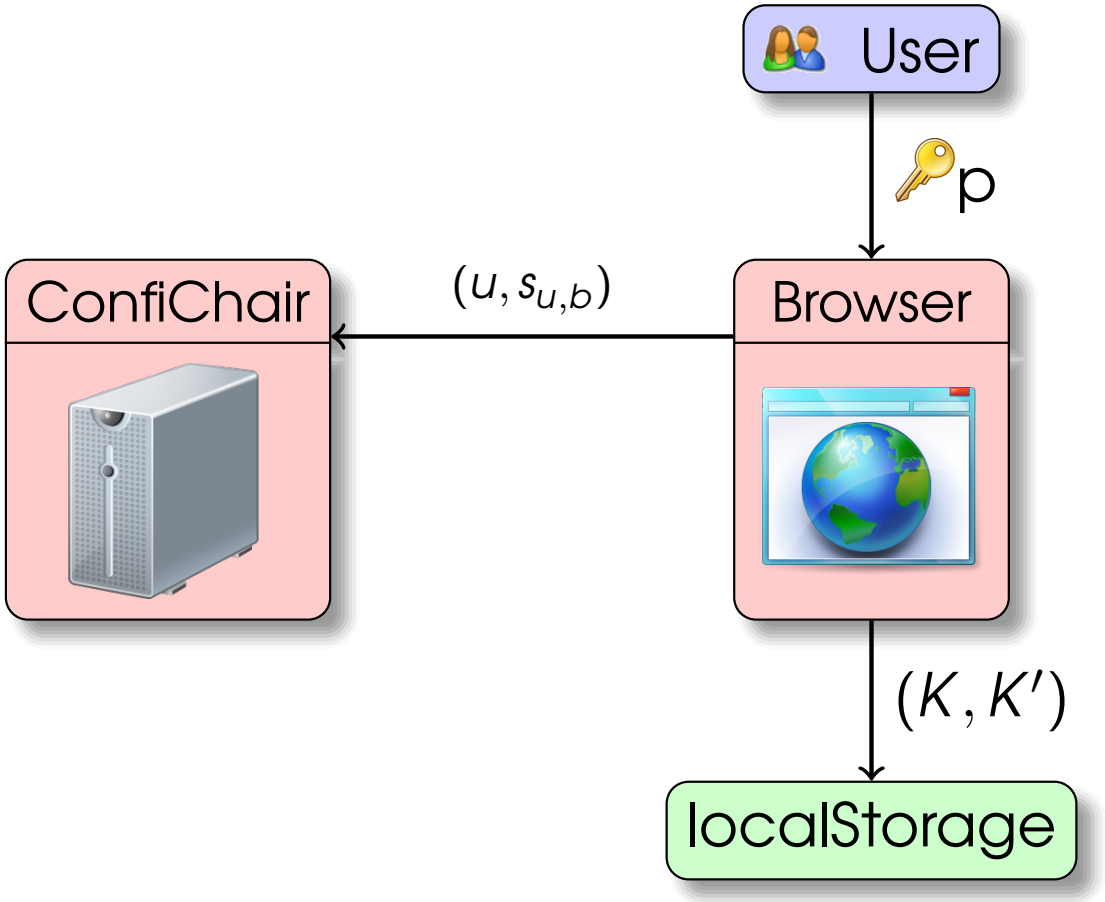
ConfiChair login



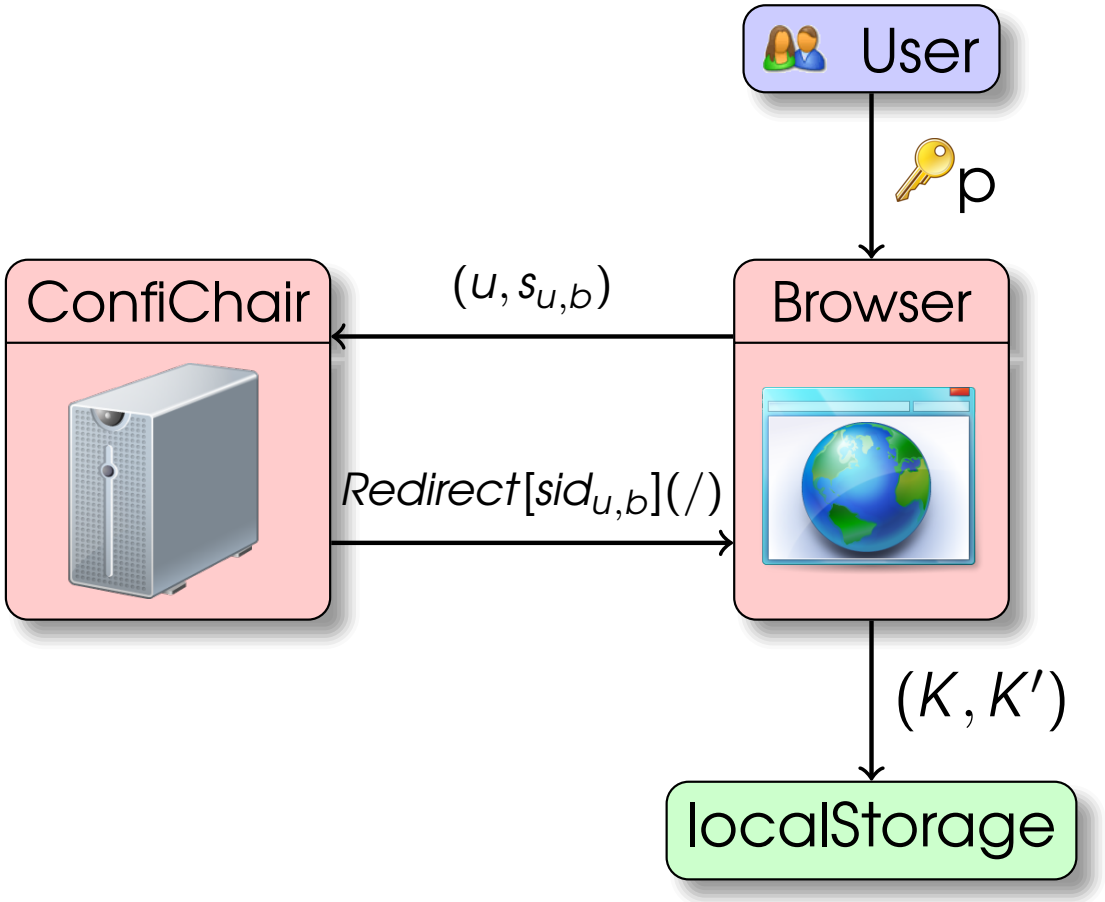
ConfiChair login



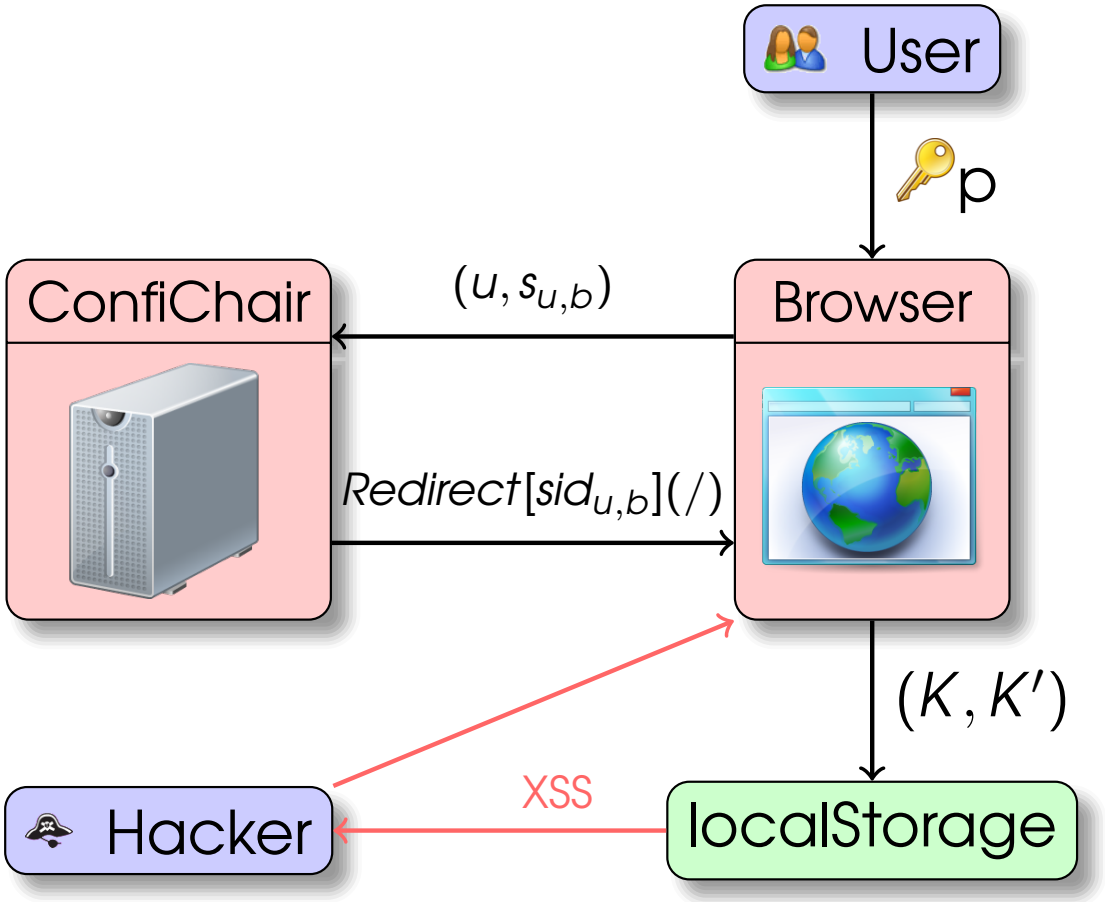
ConfiChair login



ConfiChair login



ConfiChair login



WebSpi

Formal analysis of cloud storage websites

- Applied pi-calculus model can easily reflect the application's cryptographic protocol
- Can a web attack be used to break a provably secure protocol?
- Can we make the cryptographic protocol more robust against common web threats?

D. Akhawe, A. Barth, P.E. Lam, J. Mitchell and D. Song
Towards a formal foundation of web security
CSF 2010

C. Bansal, K. Bhargavan and S. Maffeis
Discovering Concrete Attacks on Website Authrorization by Formal Analysis
CSF 2012

WebSpi

Formal analysis of cloud storage websites

- Applied pi-calculus model can easily reflect the application's cryptographic protocol
- Can a web attack be used to break a provably secure protocol?
- Can we make the cryptographic protocol more robust against common web threats?

D. Akhawe, A. Barth, P.E. Lam, J. Mitchell and D. Song
Towards a formal foundation of web security
CSF 2010

C. Bansal, K. Bhargavan and S. Maffeis
Discovering Concrete Attacks on Website Authrorization by Formal Analysis
CSF 2012

WebSpi

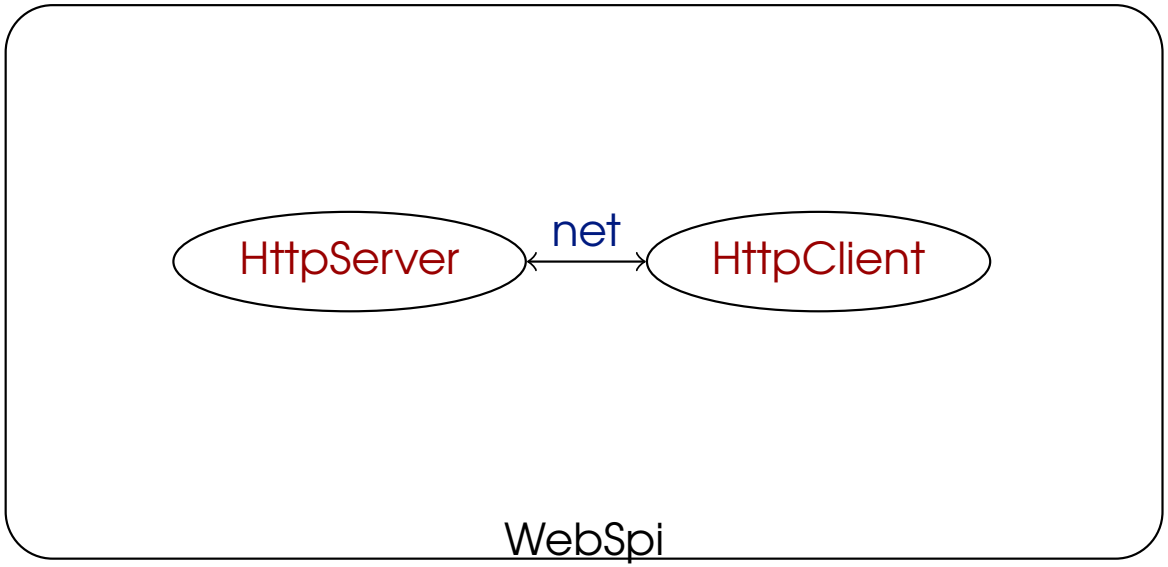
Formal analysis of cloud storage websites

- Applied pi-calculus model can easily reflect the application's cryptographic protocol
- Can a web attack be used to break a provably secure protocol?
- Can we make the cryptographic protocol more robust against common web threats?

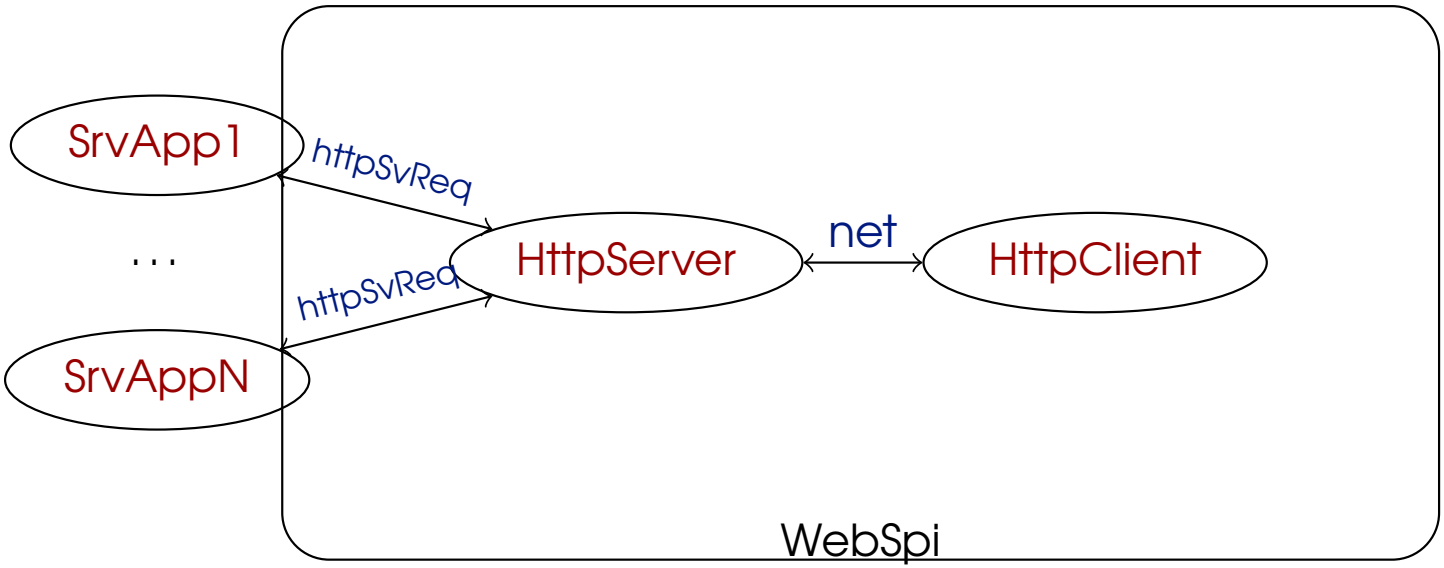

D. Akhawe, A. Barth, P.E. Lam, J. Mitchell and D. Song
Towards a formal foundation of web security
CSF 2010


C. Bansal, K. Bhargavan and S. Maffeis
Discovering Concrete Attacks on Website Authrorization by Formal Analysis
CSF 2012

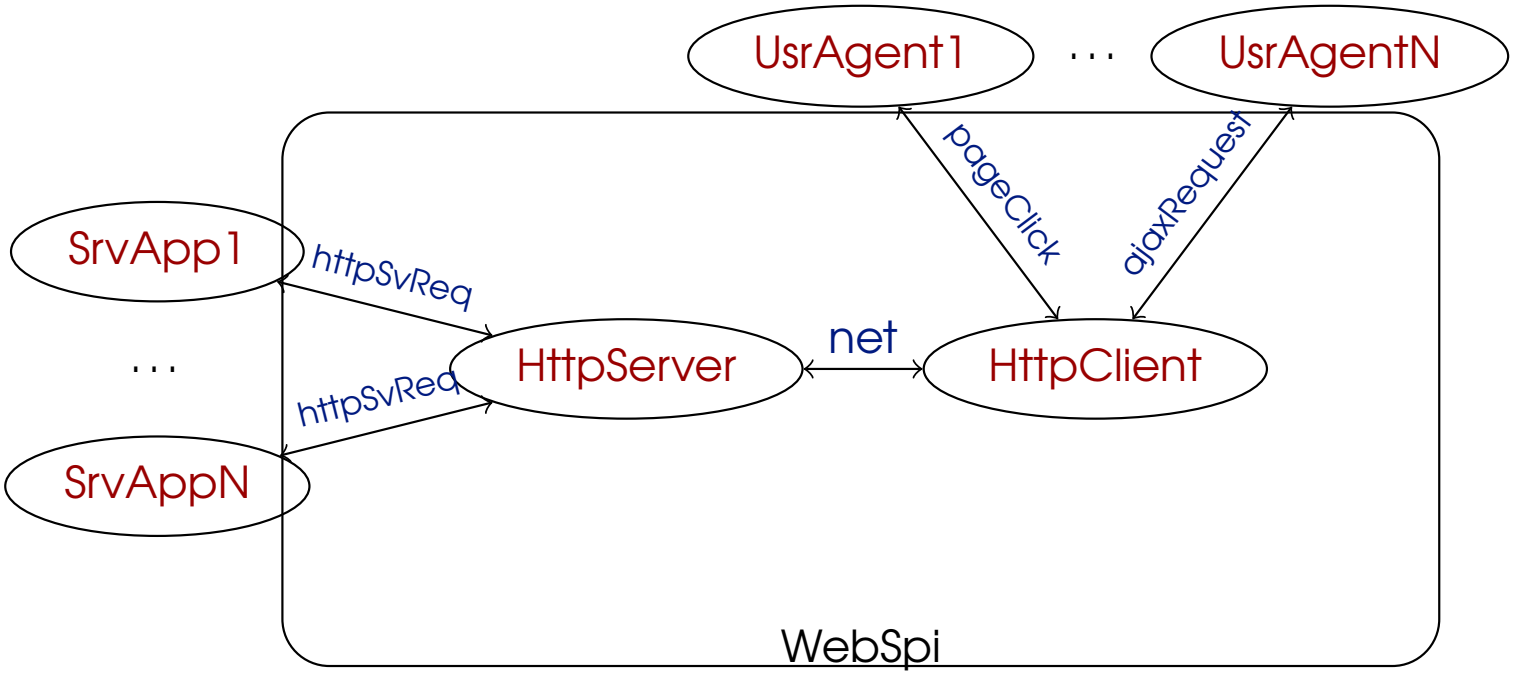
WebSpi overview



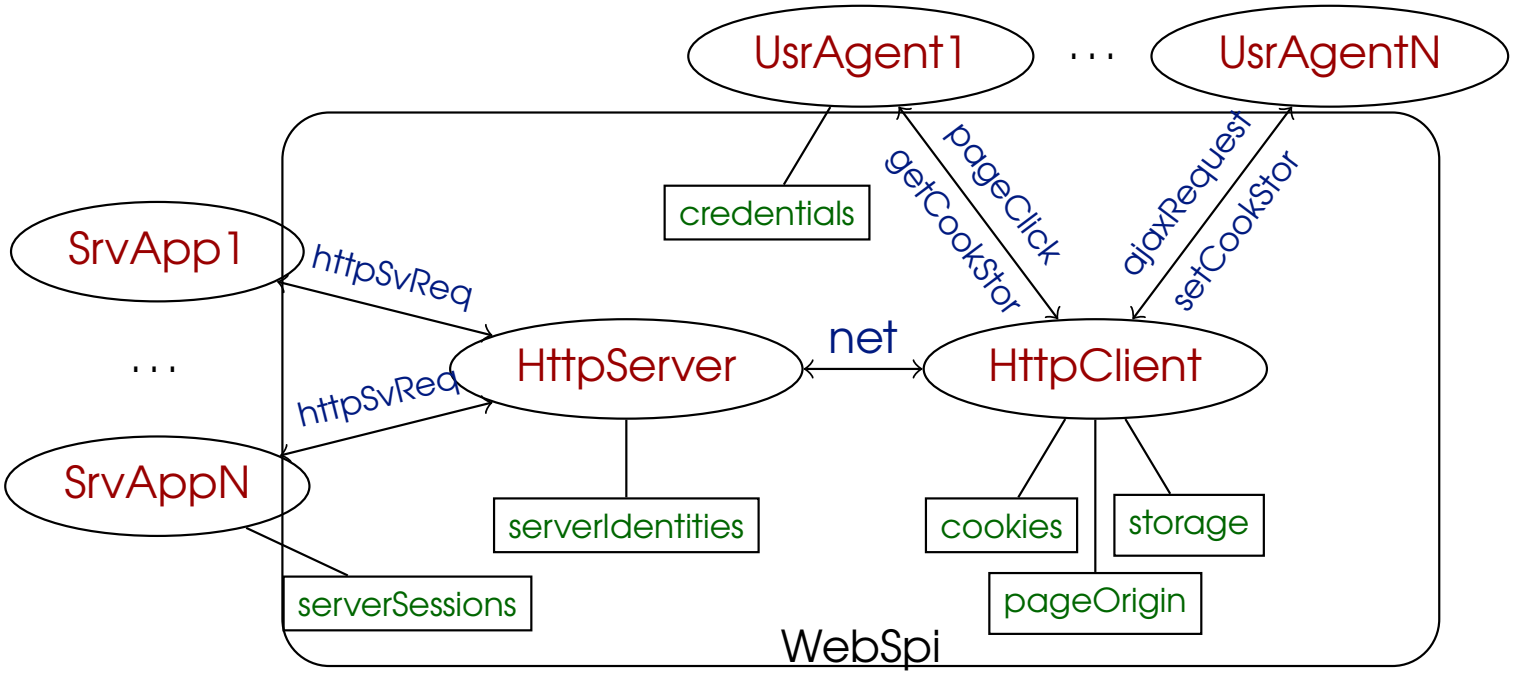
WebSpi overview



WebSpi overview



WebSpi overview



WebSpi model

Web client model

Client-side: process accepting requests on browser channels pageClick, ajaxRequest, getCookieStorage, setCookieStorage

Attacker model

- public net enables the standard Dolev-Yao *network attacker*
- a *compromised server* has its private key released
- *XSS* flaws are modeled by an AttackerProxy from net to the (secret) browser channels

WebSpi model

Verification

Security goals: correspondence assertions between user-defined events:

$$\forall M_1, \ldots M_k.\, e(M_1, \ldots M_k) \Rightarrow \varphi$$

Incompleteness

WebSpi not a complete model of the web.

- Still captures most common classes of attacks
- Extensible and actively developed

WebSpi model

Verification

Security goals: correspondence assertions between user-defined events:

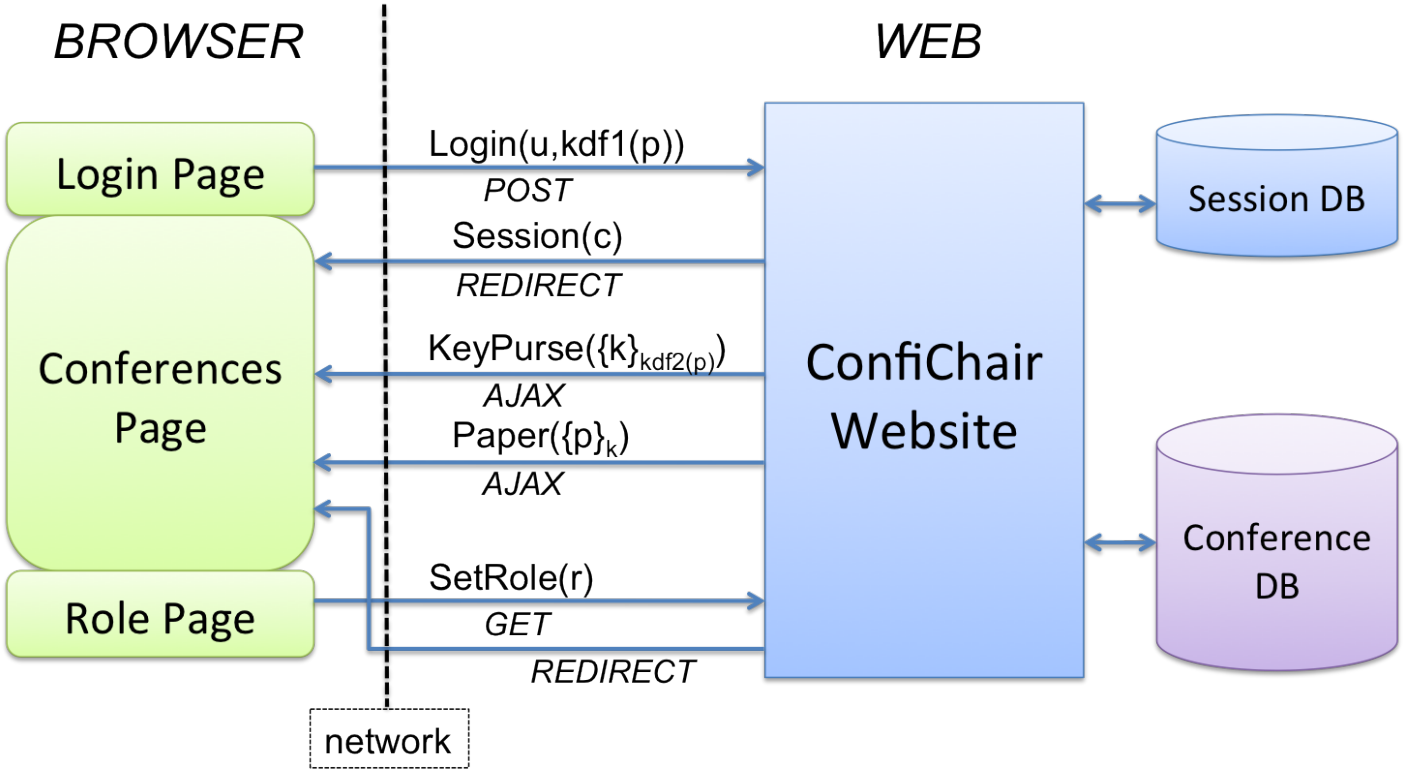
$$\forall M_1, \ldots M_k.\, e(M_1, \ldots M_k) \Rightarrow \varphi$$

Incompleteness

WebSpi not a complete model of the web.

- Still captures most common classes of attacks
- Extensible and actively developed

ConfiChair


BROWSER
WEB
Login Page
Conferences Page
Role Page
Login(u,kdf1(p))
POST
Session(c)
REDIRECT
KeyPurse({k}kdf2(p))
AJAX
Paper({p}k)
AJAX
SetRole(r)
GET
REDIRECT
network
ConfiChair Website
Session DB
Conference DB

ConfiChair login page

Login page

- LoginApp: server listening for requests on `/login`
- LoginUserAgent: JS and HTML of login page.
- waits for user to type username and password
- derives credential and sends it with username to LoginApp over HTTPS

```
let loginURI = uri(https(), confichair, loginPath(), noParams()) in
out(browserRequest(b),(loginURI, httpGet()));
in (newPage(b),(p:Page,=loginURI,d:bitstring));
get userData(=confichair, uid, pwd, paper) in
let cred = kdf1(pwd) in
in (getCookieStorage(b),(=p,cookiePair(cs,ch),od:Data));
out (setCookieStorage(b),(p,ch,storePassword(pwd)));
event LoginInit(confichair, b, uid);
out(pageClick(b),(p,loginURI,
        httpPost(loginFormReply(uid,cred))))
```

ConfiChair conference pages

Conference pages

- server-side ConferenceApp and client-side ConferenceUserAgent processes
- ConferencesUserAgent first retrieves user's keypurse using AJAX
- keypurse decrypted with stored key and stored in local storage

```
let keypurseURI = uri(https(), confichair,
    keyPursePath(), nullParams()) in
out (ajaxRequest(b),(p,keypurseURI,httpGet()));
in (ajaxResponse(b),(=p,=keypurseURI,JSON(x)));
in (getCookieStorage(b),
   (=p,cookiePair(cs,ch),storePassword(pwd)));
let keypurse(k) = adec(x, kdf2(pwd)) in
out (setCookieStorage(b),(p,ch,storeKeypurse(k)))
```

ConfiChair conference pages

Conference pages

At any point, user may request a paper to be downloaded and decrypted using key in keypurse

```
let paperURI = uri(https(), h, paperPath(), nullParams()) in
out (ajaxRequest(b),(p,paperURI,httpGet()));
in (ajaxResponse(b),(=p,=paperURI,JSON(y)));
in (getCookieStorage(b),
   (=p,cookiePair(cs,ch),storeKeypurse(k)));
let paper = adec(y,k) in event PaperReceived(paper))
```

ConfiChair security goals

Security goals

- Login authentication:

```
event(LoginAuthorized(confichair,id,u,c))
  ⟹event(LoginInit(confichair,b,id))
```

- Secrecy of papers:

```
in(paperChannel, paper:bitstring);
get userData(h, uId, k, =paper) in
 event PaperLeak(uId,paper).
query u:Id,p:bitstring; event(PaperLeak(id,p))
```

ConfiChair attacker

XSS vulnerability in Role Page

`http://confichair.org/?set-role=<script>S</script>`

If requested role is invalid, returned error page contains the unsanitized requested role name.

In RoleUserAgent, the page identifier is released

```
let roleURI = uri(https(), h, changeRolePath(), roleParams(x)) in
out(browserRequest(b),(roleURI, httpGet()));
in (newPage(b),(p:Page,=roleURI,y:bitstring));
out(pub, p)
```

Result of verification

Authentication goal is broken

Attacker can read the user's password from local storage and leak it to a malicious website

Paper privacy is broken

Attacker can read the paper decryption key from local storage and leak it to a malicious website

... in previous ProVerif analysis, same goals were valid against cloud attacker model.

Robustness against XSS

Robustness against XSS

- Can fix XSS, but there could be others
- As a first step, store (K, K') rather than p
- Server wraps all keys in keypurse with a fresh, short-lived key.
- The decryption script unwraps the input key before decryption
- Decryption function may not leak the short-lived wrapping key, this requires advanced language-based techniques to achieve

Robustness against XSS

Robustness against XSS

- Can fix XSS, but there could be others
- As a first step, store (K, K') rather than p
- Server wraps all keys in keypurse with a fresh, short-lived key.
- The decryption script unwraps the input key before decryption
- Decryption function may not leak the short-lived wrapping key, this requires advanced language-based techniques to achieve

Robustness against XSS

Robustness against XSS

- Can fix XSS, but there could be others
- As a first step, store (K, K') rather than p
- Server wraps all keys in keypurse with a fresh, short-lived key.
- The decryption script unwraps the input key before decryption
- Decryption function may not leak the short-lived wrapping key, this requires advanced language-based techniques to achieve

Robustness against XSS

Robustness against XSS

- Can fix XSS, but there could be others
- As a first step, store (K, K') rather than p
- Server wraps all keys in keypurse with a fresh, short-lived key.
- The decryption script unwraps the input key before decryption
- Decryption function may not leak the short-lived wrapping key, this requires advanced language-based techniques to achieve

Robustness against XSS

Robustness against XSS

- Can fix XSS, but there could be others
- As a first step, store (K, K') rather than p
- Server wraps all keys in keypurse with a fresh, short-lived key.
- The decryption script unwraps the input key before decryption
- Decryption function may not leak the short-lived wrapping key, this requires advanced language-based techniques to achieve

Towards automatic model generation

From implementation to model

- Writing WebSpi model can be difficult for website authors
- Current line of work investigates direct translation of concrete PHP+JavaScript websites into WebSpi processes
- Model extraction from traces (AuthScan)
- Only the security goals and events need to be added manually

Towards automatic model generation

From implementation to model

- Writing WebSpi model can be difficult for website authors
- Current line of work investigates direct translation of concrete PHP+JavaScript websites into WebSpi processes
- Model extraction from traces (AuthScan)
- Only the security goals and events need to be added manually

Towards automatic model generation

From implementation to model

- Writing WebSpi model can be difficult for website authors
- Current line of work investigates direct translation of concrete PHP+JavaScript websites into WebSpi processes
- Model extraction from traces (AuthScan)
- Only the security goals and events need to be added manually

Towards automatic model generation

From implementation to model

- Writing WebSpi model can be difficult for website authors
- Current line of work investigates direct translation of concrete PHP+JavaScript websites into WebSpi processes
- Model extraction from traces (AuthScan)
- Only the security goals and events need to be added manually

Thank you. Questions?

Maffeis is supported by EPSRC grant EP/I004246/1.
Bhargavan and Delignat-Lavaud are supported by ERC starting grant CRYSP
Bansal is supported by Microsoft Research India Travel Grant