

Cryptographic protocols (II)

Myrto Arapinis
School of Informatics
University of Edinburgh

March 17, 2016

Credit card payment protocol

Credit card payment



Credit card payment



- ▶ Is it a real card?

Credit card payment



- ▶ Is it a real card?
- ▶ Is the pin protected?

Behavior in the usual case



1. The waiter introduces the credit card

Behavior in the usual case



1. The waiter introduces the credit card
2. The waiter enters the amount m of the transaction

Behavior in the usual case



1. The waiter introduces the credit card
2. The waiter enters the amount m of the transaction
3. The terminal authenticates the card

Behavior in the usual case



1. The waiter introduces the credit card
2. The waiter enters the amount m of the transaction
3. The terminal authenticates the card
4. The customer enters his secret pin

Behavior in the usual case



1. The waiter introduces the credit card
2. The waiter enters the amount m of the transaction
3. The terminal authenticates the card
4. The customer enters his secret pin
If the amount m is greater than 100 euros (and in only 20% of the cases)
 - 4.1 The terminal asks for authentication of the card
 - 4.2 The bank provides authentication

More details

4 actors: Bank, Customer, Card, and Terminal

Bank owns:

- ▶ a secret signing key sk_B
- ▶ a public verification key pk_B
- ▶ a secret symmetric encryption key per card K_{CB}

Card owns:

- ▶ Data: last name, first name, card's number, expiration date
- ▶ Signature's value $VS = \{\text{hash}(\text{Data})\}_{sk_B}$
- ▶ a secret symmetric encryption shared with the bank K_{CB}

Terminal owns:

- ▶ the public verification key pk_B

Credit card payment protocol (in short)

The terminal reads the card:

1. $Ca \rightarrow T : Data, \{\text{hash}(Data)\}_{sk_B}$

Credit card payment protocol (in short)

The terminal reads the card:

1. $Ca \rightarrow T : Data, \{\text{hash}(Data)\}_{sk_B}$

The terminal asks for the secret pin:

2. $T \rightarrow Cu : \text{pin?}$
3. $Cu \rightarrow Ca : 1234$
4. $Ca \rightarrow T : \text{ok}$

Credit card payment protocol (in short)

The terminal reads the card:

$$1. Ca \rightarrow T : Data, \{\text{hash}(Data)\}_{sk_B}$$

The terminal asks for the secret pin:

$$2. T \rightarrow Cu : \text{pin?}$$

$$3. Cu \rightarrow Ca : 1234$$

$$4. Ca \rightarrow T : \text{ok}$$

The terminal calls the bank

$$5. T \rightarrow B : \text{auth?}$$

$$6. B \rightarrow T : N_B$$

$$7. T \rightarrow Ca : N_B$$

$$8. Ca \rightarrow T : \{N_B\}_{K_{Cb}}$$

$$9. T \rightarrow B : \{N_B\}_{K_{Cb}}$$

$$10. B \rightarrow T : \text{ok}$$

Some flaws

The security was initially ensured by:

- ▶ the cards were difficult to reproduce
- ▶ the protocol (!) and keys were secret

Some flaws

The security was initially ensured by:

- ▶ the cards were difficult to reproduce
- ▶ the protocol (!) and keys were secret

But:

- ▶ cryptographic flaw: 320-bit keys can be broken (1988),
- ▶ logical flaw: no link between the secret code and the authentication of the card,
- ▶ fake cards can be built.

Some flaws

The security was initially ensured by:

- ▶ the cards were difficult to reproduce
- ▶ the protocol (!) and keys were secret

But:

- ▶ cryptographic flaw: 320-bit keys can be broken (1988),
- ▶ logical flaw: no link between the secret code and the authentication of the card,
- ▶ fake cards can be built.

⇒ “YesCard” built by Serge Humpich (France, 1998)

How does the “YesCard” work?

Logical flow

1. $Ca \rightarrow T$: $Data, \{\text{hash}(Data)\}_{sk_B}$
2. $T \rightarrow Cu$: pin?
3. $Cu \rightarrow Ca$: 1234
4. $Ca \rightarrow T$: ok

How does the “YesCard” work?

Logical flow

1. $Ca \rightarrow T : Data, \{\text{hash}(Data)\}_{sk_B}$
2. $T \rightarrow Cu : \text{pin?}$
3. $Cu' \rightarrow Ca' : 5678$
4. $Ca' \rightarrow T : \text{ok}$

How does the “YesCard” work?

Logical flaw

1. $Ca \rightarrow T$: $Data, \{\text{hash}(Data)\}_{sk_B}$
2. $T \rightarrow Cu$: pin?
3. $Cu' \rightarrow Ca'$: 5678
4. $Ca' \rightarrow T$: ok

There is always someone to debit

How does the “YesCard” work?

Logical flaw

1. $Ca \rightarrow T$: $Data, \{\text{hash}(Data)\}_{sk_B}$
2. $T \rightarrow Cu$: pin?
3. $Cu' \rightarrow Ca'$: 5678
4. $Ca' \rightarrow T$: ok

There is always someone to debit
→ creation of a fake card

How does the “YesCard” work?

Logical flow

1. $Ca \rightarrow T$: $Data, \{\text{hash}(Data)\}_{sk_B}$
2. $T \rightarrow Cu$: pin?
3. $Cu' \rightarrow Ca'$: 5678
4. $Ca' \rightarrow T$: ok

There is always someone to debit

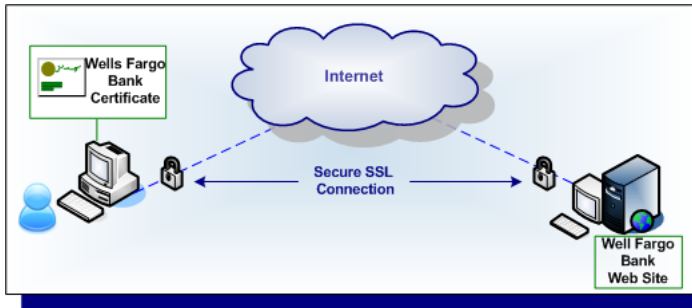
→ creation of a fake card

1. $Ca' \rightarrow T$: $XXXX, \{\text{hash}(XXXX)\}_{sk_B}$
2. $T \rightarrow Cu'$: pin?
3. $Cu' \rightarrow Ca'$: 0000
4. $Ca' \rightarrow T$: ok

The SSL/TLS protocol

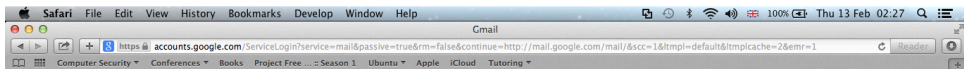
SSL/TLS protocol

Goals: Confidentiality, Integrity, Non repudiation




SSL/TLS use X.509 certificates and hence asymmetric cryptography to exchange a symmetric key. This session key is then used to encrypt subsequent communication. This allows for **data/message confidentiality**, and message authentication codes for **message integrity** and thus, **message authentication**.

SSL/TLS protocol



One account. All of Google.

Sign in to continue to Gmail



Myrto Arapinis
myrto.arapinis@gmail.com

Sign in

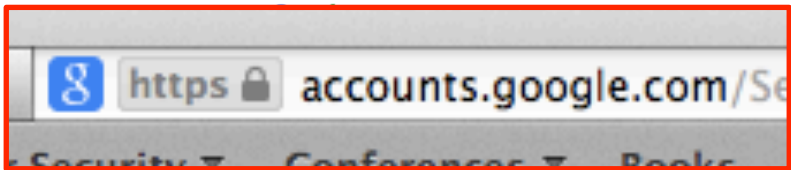
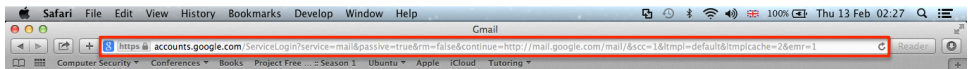
[Need help?](#)

[Manage accounts on this device](#)

One Google Account for everything Google



SSL/TLS protocol



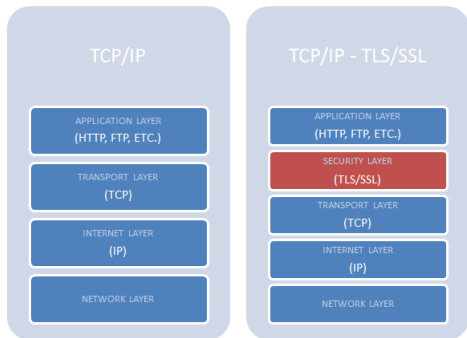
A sign-in form for a Google account. At the top is a blue circular profile picture. Below it, the name 'Myrto Arapinis' and email address 'myrto.arapinis@gmail.com' are displayed. There is a text input field labeled 'Password'. Below the field is a blue 'Sign in' button. A link for 'Need help?' is located below the button.

[Manage accounts on this device](#)

One Google Account for everything Google

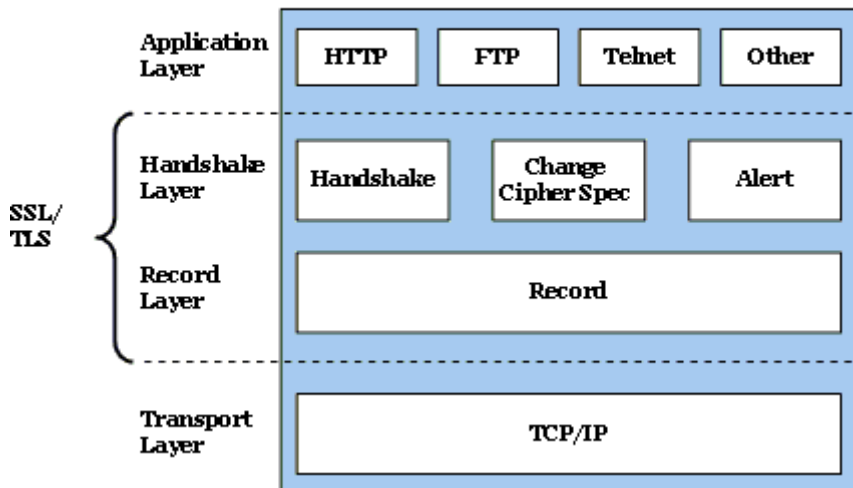


TCP/IP protocol stack



- ▶ TCP/IP provides end-to-end connectivity and is organized into four abstraction layers which are used to sort all related protocols according to the scope of networking involved
- ▶ The SSL/TLS library operates above the transport layer (uses TCP) but below application protocols

SSL/TLS protocol layers



SSL/TLS handshake protocol



SSL/TLS renegotiation

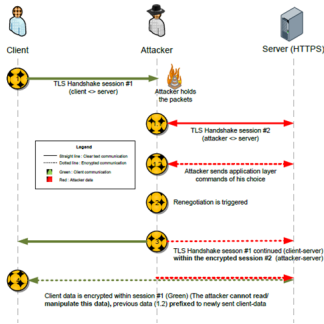
Client and server are allowed to initiate renegotiation of the session encryption in order to:

- ▶ Refresh keys
- ▶ Increase authentication
- ▶ Increase cipher strength
- ▶ ...

Client or server can trigger renegotiation by sending a hello message

SSL/TLS renegotiation weaknesses

- ▶ Renegotiation has priority over application data!
- ▶ Renegotiation can take place in the middle of an application layer transaction!



(Detailed on the board)

Incorrect implicit assumption: the client doesn't change through renegotiation

Marsh Ray's plaintext injection attack on HTTPS

Marsh Ray's plaintext injection attack on HTTPS

Attacker:

```
GET /pizza?toppings=pepperoni;address=attacker_str HTTP/1.1  
X-Ignore-This:(no carriage return)
```

Marsh Ray's plaintext injection attack on HTTPS

Attacker:

```
GET /pizza?toppings=pepperoni;address=attacker_str HTTP/1.1  
X-Ignore-This:(no carriage return)
```

Victim:

```
GET /pizza?toppings=sausage;address=victim_str HTTP/1.1  
Cookie:victim_cookie
```

Marsh Ray's plaintext injection attack on HTTPS

Attacker:

```
GET /pizza?toppings=pepperoni;address=attacker_str HTTP/1.1  
X-Ignore-This:(no carriage return)
```

Victim:

```
GET /pizza?toppings=sausage;address=victim_str HTTP/1.1  
Cookie:victim_cookie
```

Result:

```
GET /pizza?toppings=pepperoni;address=attacker_str HTTP/1.1  
X-Ignore-This:GET /pizza?toppings=sausage;address=victim_str HTTP/1.1  
Cookie:victim_cookie
```

Marsh Ray's plaintext injection attack on HTTPS

Attacker:

```
GET /pizza?toppings=pepperoni;address=attacker_str HTTP/1.1  
X-Ignore-This:(no carriage return)
```

Victim:

```
GET /pizza?toppings=sausage;address=victim_str HTTP/1.1  
Cookie:victim_cookie
```

Result:

```
GET /pizza?toppings=pepperoni;address=attacker_str HTTP/1.1  
X-Ignore-This:GET /pizza?toppings=sausage;address=victim_str HTTP/1.1  
Cookie:victim_cookie
```

⇒ **Server uses victim's account to send a pizza to attacker!**

Anil Kurmus' plaintext injection attack on HTTPS

Anil Kurmus' plaintext injection attack on HTTPS

Twitter status updates using its API by posting the new status to `http://twitter.com/statuses/update.xml`, as well as the user name and password

Anil Kurmus' plaintext injection attack on HTTPS

Twitter status updates using its API by posting the new status to `http://twitter.com/statuses/update.xml`, as well as the user name and password

Attacker:

POST /statuses/update.xml HTTP/1.1

Authorization: Basic username:password

User-Agent: curl/7.19.5

Host: twitter.com

Accept: */*

Content-Length: 140

Content-Type: application/x-www-form-urlencoded
status=

Anil Kurmus' plaintext injection attack on HTTPS

Twitter status updates using its API by posting the new status to `http://twitter.com/statuses/update.xml`, as well as the user name and password

Attacker:

```
POST /statuses/update.xml HTTP/1.1
```

```
Authorization: Basic username:password
```

```
User-Agent: curl/7.19.5
```

```
Host: twitter.com
```

```
Accept: */*
```

```
Content-Length: 140
```

```
Content-Type: application/x-www-form-urlencoded  
status=
```

Victim:

```
POST /statuses/update.xml HTTP/1.1
```

```
Authorization: Basic username:password...
```


Anil Kurmus' plaintext injection attack on HTTPS

Twitter status updates using its API by posting the new status to `http://twitter.com/statuses/update.xml`, as well as the user name and password

Attacker:

```
POST /statuses/update.xml HTTP/1.1
Authorization: Basic username:password
User-Agent: curl/7.19.5
Host: twitter.com
Accept: */*
Content-Length: 140
Content-Type: application/x-www-form-urlencoded
status=
```

Victim:

```
POST /statuses/update.xml HTTP/1.1
Authorization: Basic username:password...
```

⇒ the attacker gets the user name and password of the victim!

The SAML Single Sign On (SSO) protocol


SAML SSO protocol

Chrome File Edit View History Bookmarks Window Help Thu 13 Feb 00:50


BBC - Homepage www.bbc.co.uk

BBC Sign in News Sport Weather iPlayer TV Radio More Search


THURSDAY 13 FEBRUARY




RT
m 2-3 Liverpool



Storm updates: Get the latest for where you are






Triple killer's accomplices guilty

BBC NEWS

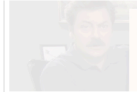
- » UK parties 'will block money union'
- » Comedian Sid Caesar dies at 91
- » Carney adjusts interest rates policy
- » Ketamine to become Class B drug




Fulham 2-3 Liverpool

BBC SPORT


- » Decisive day for Team GB in Sochi
- » Collingwood handed England role
- » Arsenal 0-0 Manchester United
- » Everton fan's 30-year wait goes on



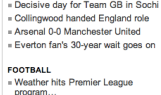
Parks and Rec is bac
Behold the Pyramid!



BBC ONE | COMEDY
Outnumbered
Episode 3




BBC RADIO 2 | FACTUAL
David Attenborough and th...



FOOTBALL

- » Weather hits Premier League program...
- » Newcastle United 0-4 Tottenham Hots...
- » Hodgson rules out Terry return



BBC TWO | University
Episode 1

BBC now Entertainment News Lifestyle Knowledge Sport

https://ssl.bbc.co.uk/1/id/signin

SAML SSO protocol

Chrome File Edit View History Bookmarks Window Help

BBC – Sign in

https://ssl.bbc.co.uk/id/signin

BBC Sign in News Sport Weather iPlayer TV More Search

SIGN IN BBC ID

Don't have a BBC ID? Please register.

Email or username

Password

Forgot your password?

Remember me Untick if you're using a shared computer.

Sign in Cancel

Other ways to sign in

You'll be signed in to the BBC for 30 days.

Facebook Google

Please only use these if you are 16 or over.
We won't share or post your activity to Facebook or Google

About BBC ID

Simple
Register quickly and easily to comment, add favourites, and more...

Safe
We store your information securely, and we never share it without your permission.

Spam-free
We'll only send you emails if you ask for them.

BBC ID help

BBC

iWonder

How did Pack Up Your Troubles become the viral hit of WW1?

Gareth Malone explains why the song was a success

News Sport Weather iPlayer TV Radio

Comedy Food History Learning

https://ssl.bbc.co.uk/id/statecookie/google.com

SAML SSO protocol

The image shows a Chrome browser window displaying the Google Accounts sign-in page. The browser's address bar shows the URL: `https://accounts.google.com/ServiceLogin?service=iso&passive=1209600&continue=https://accounts.google.com/o/oauth2/auth?scope%3Dhttps://www.googleapis.com...`. The page features the Google logo at the top, followed by the text "Sign in with your Google Account". Below this is a sign-in form with a profile icon placeholder, an "Email" input field, a "Password" input field, and a blue "Sign in" button. A link for "Need help?" is located below the button. At the bottom of the form area, there is a link to "Create an account". Below the form, the text "One Google Account for everything Google" is displayed, accompanied by icons for various Google services: Search, Gmail, Drive, YouTube, Maps, and Photos. The footer of the page contains the text "Google Privacy & Terms Help".

Chrome File Edit View History Bookmarks Window Help

Sign in - Google Accounts

<https://accounts.google.com/ServiceLogin?service=iso&passive=1209600&continue=https://accounts.google.com/o/oauth2/auth?scope%3Dhttps://www.googleapis.com...>

Google

Sign in with your Google Account

Email

Password

Sign in

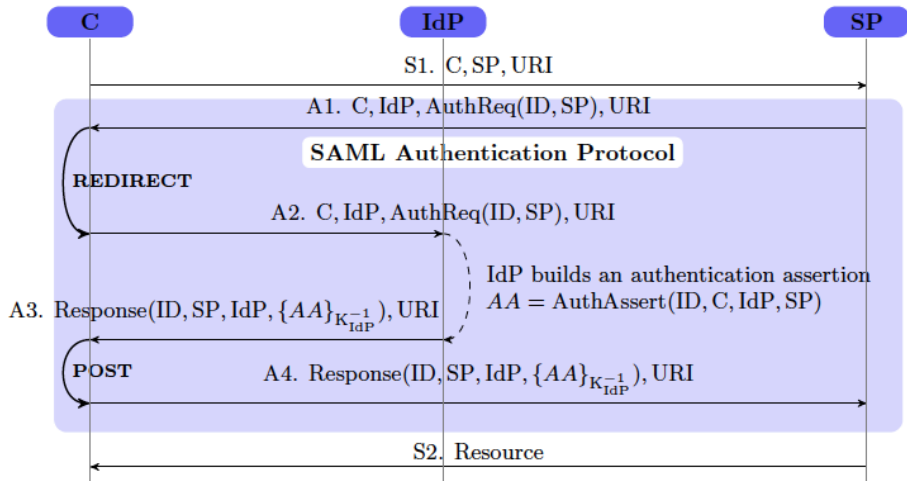
Need help?

[Create an account](#)

One Google Account for everything Google

Google Privacy & Terms Help

SAML SSO protocol (OASIS 2005)



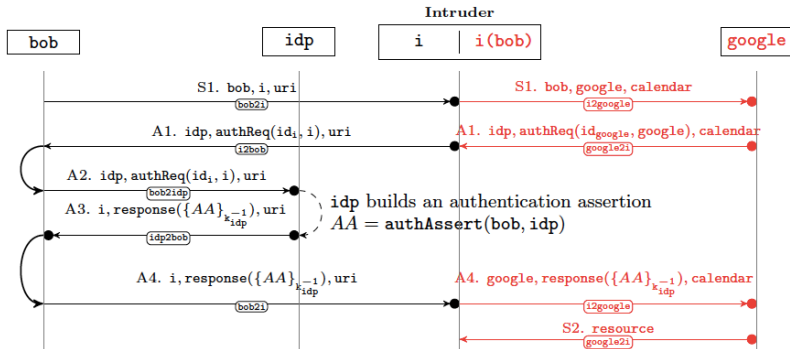
Google's implementation of SSO

Google's SAML-based Single Sign-On for Google Applications deviates from the above protocol for a few, seemingly minor simplifications in the messages exchanged:

- G1. ID and SP are not included in the authentication assertion, *i.e.* $AA = \text{AuthAssert}(C; IdP)$ instead of $\text{AuthAssert}(ID; C; IdP; SP)$;
- G2. ID , SP and IdP are not included in the response, *i.e.* $\text{Resp} = \text{Response}(\{AA\}_{K_{IdP}^{-1}})$ instead of $\text{Response}(ID; SP; IdP; \{AA\}_{K_{IdP}^{-1}})$.

Attack Google's SSO implementation

[A. Armando, R. Carbone, L. Compagna, J. Cullar, L. Tobarra, "Formal analysis of SAML 2.0 web browser single sign-on: breaking the SAML-based single sign-on for google apps", (FMSE'08)]



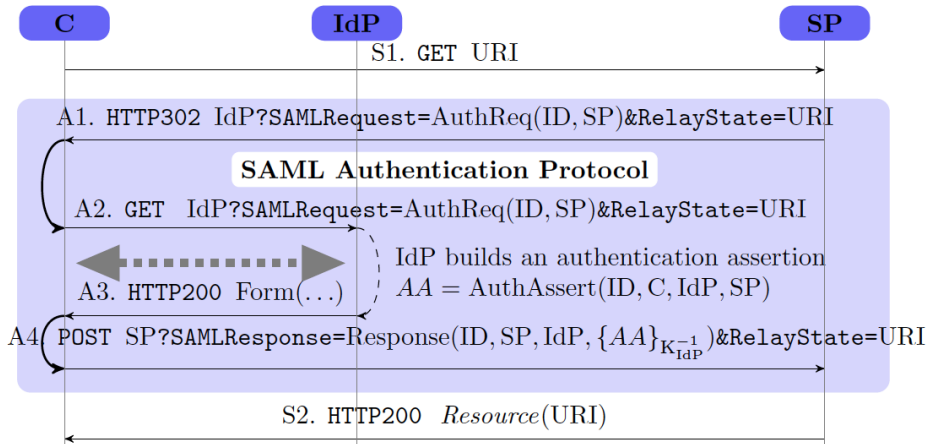
Legend:

$A \xrightarrow[\text{ch}]{M} B$: A sends M on ch confidential to B

$A \bullet \xrightarrow[\text{ch}]{M} B$: A sends M on ch authentic for A

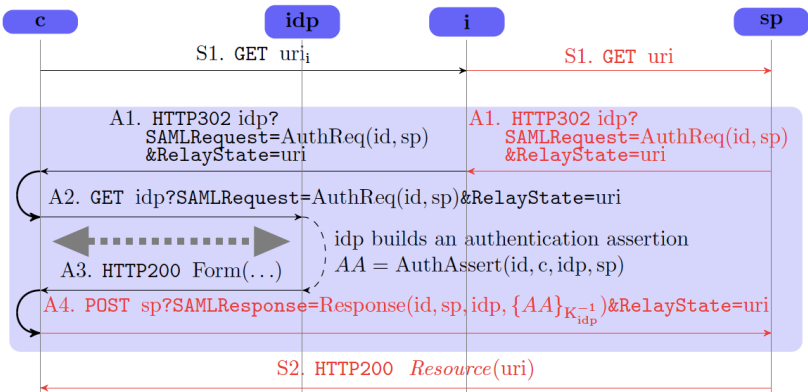
$A \bullet \xrightarrow[\text{ch}]{M} \bullet B$: M is sent on ch authentic for A and confidential to B

SAML SSO protocol (OASIS 2012)



Attack SAML SSO protocol (OASIS 2012)

[A. Armando, R. Carbone, L. Compagna, J. Cullar, G. Pellegrino, A. Sorniotti, "From Multiple Credentials to Browser-Based Single Sign-On: Are We More Secure?", Chapter in Future Challenges in Security and Privacy for Academia and Industry]



⇒ XSS attack on SAML-based SSO for Google Apps