Browser Security

Outline

- Crash course in HTTP and HTTPS.
- Classical security model VS browser security.
- Weaknesses and attacks on the CA infrastructure.
- Weaknesses and attacks on the SSL protocol.
- Other vulnerabilities

Most HTTPS requests start out as HTTP

- User writes “my-bank.com” in address field
- Browser re-writes to “http://my-bank.com” and sends the request to my-bank.com
- Server responds with http redirect to “https://my-bank.com”.
- Finally, the client and server negotiate the SSL connection, and the request is sent over TLS.

Crash course in HTTP

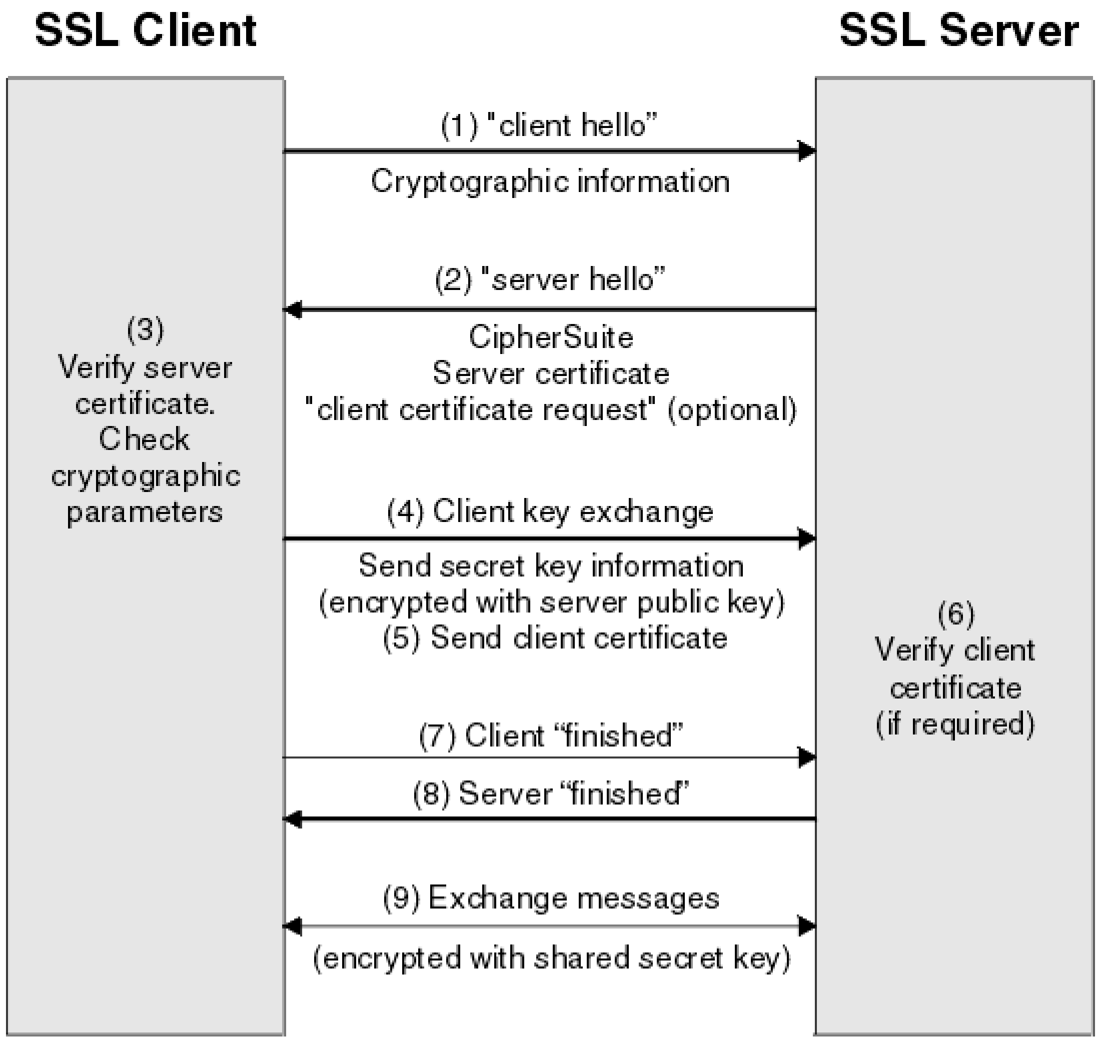
- Request sent from browser to server on port 80:
  - GET /path/index.html?arg1=1 HTTP/1.1
    Host: my-bank.com
    user-agent: opera
    Cookie: *visited = 1*

- Response from server:
  - HTTP/1.1 301 Moved Permanently
    Location: https://my-bank.com/path/index.html?arg1=1

- Client re-sends the same request over port 443 tunneled in TLS

Crash course in HTTPS

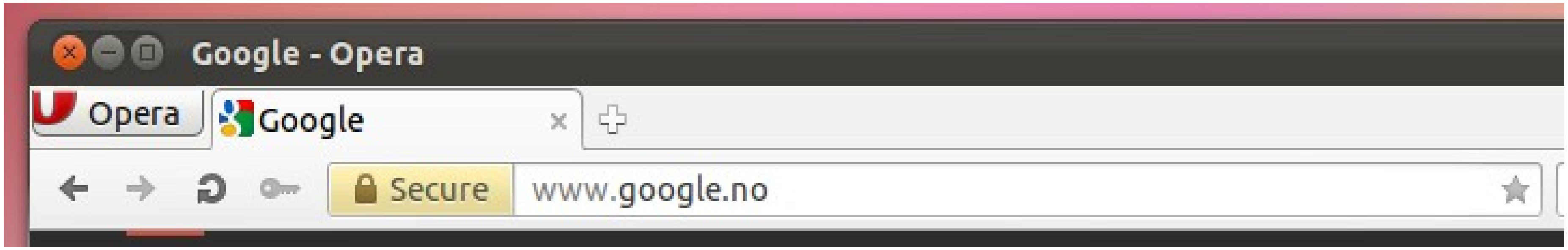
HTTPS://www.my-bank.com

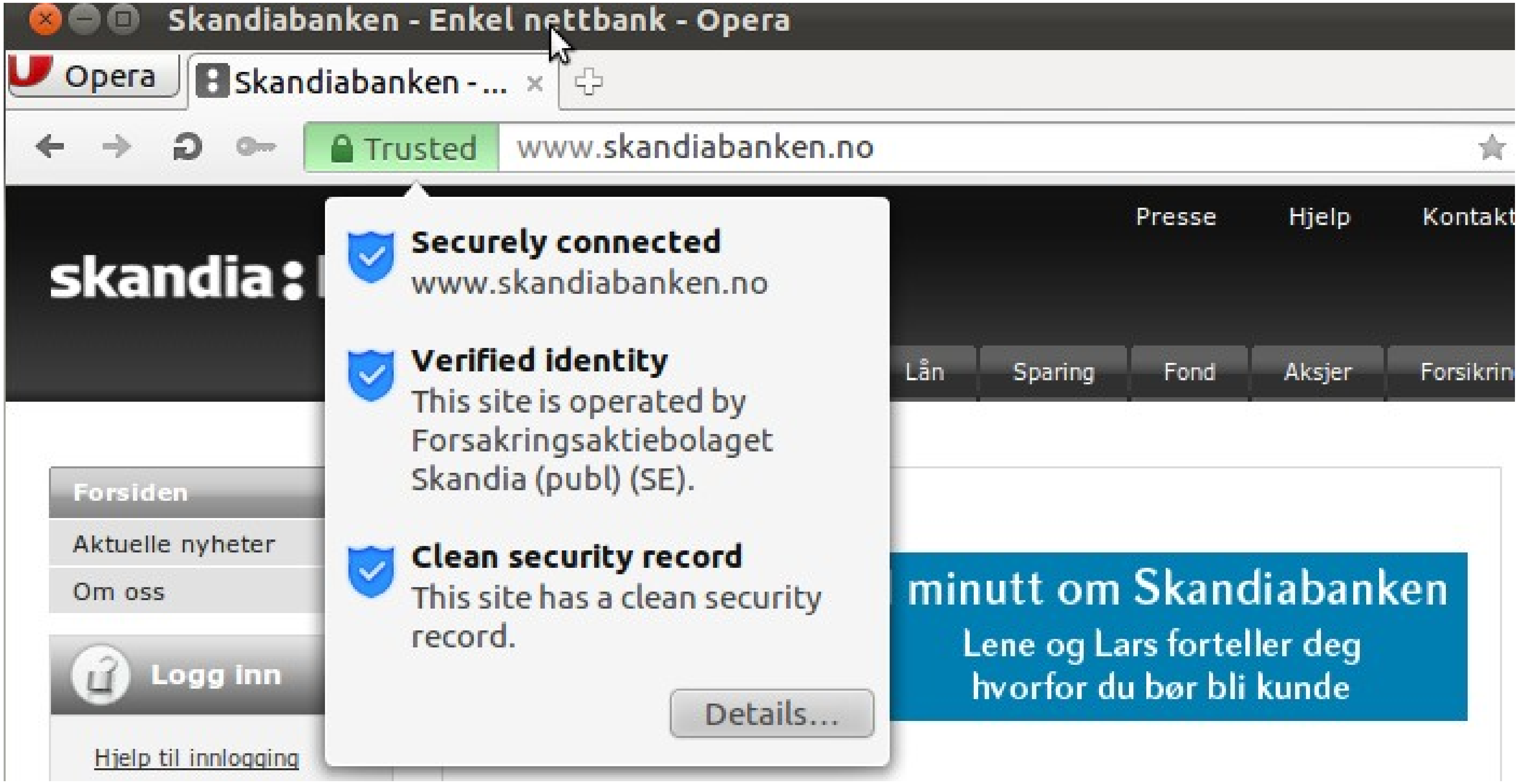

SSL Client
SSL Server
(1) "client hello"
Cryptographic information
(2) "server hello"
CipherSuite
Server certificate
"client certificate request" (optional)
(3) Verify server certificate. Check cryptographic parameters
(4) Client key exchange
Send secret key information (encrypted with server public key)
(5) Send client certificate
(6) Verify client certificate (if required)
(7) Client "finished"
(8) Server "finished"
(9) Exchange messages
(encrypted with shared secret key)

Crash course in HTTPS

- The HTTP request is tunneled through the TLS connection on port 443
  - GET /path/index.html?arg1=1 HTTP/1.1
    Host: my-bank.com
    user-agent: opera
    Cookie: *visited = 1*
- *Server responds with*
  - HTTP/1.1 200 OK
    Server: Apache

    <HTML content>



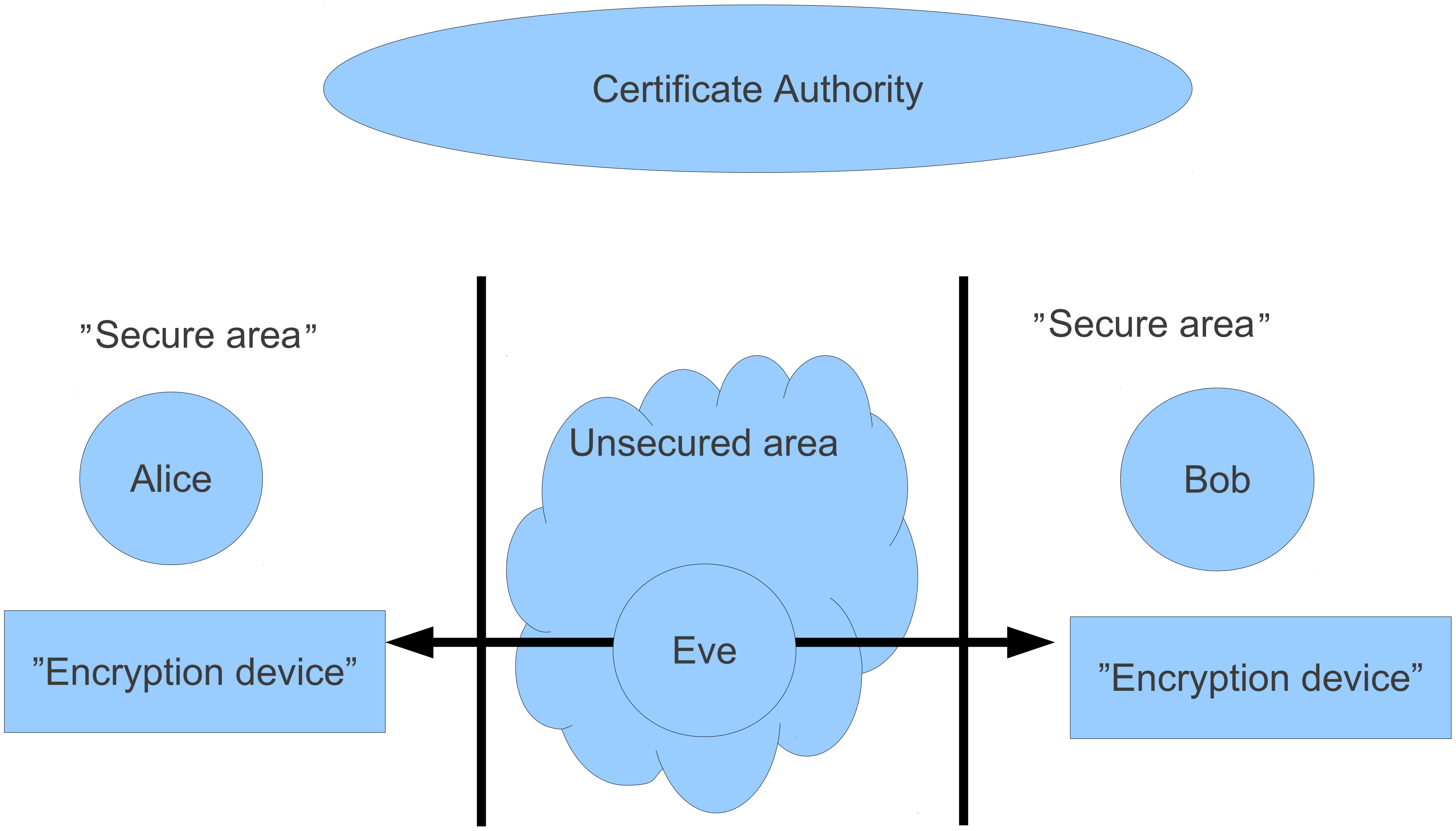


Securely connected
www.skandiabanken.no

Verified identity
This site is operated by Forsakringsaktiebolaget Skandia (publ) (SE).

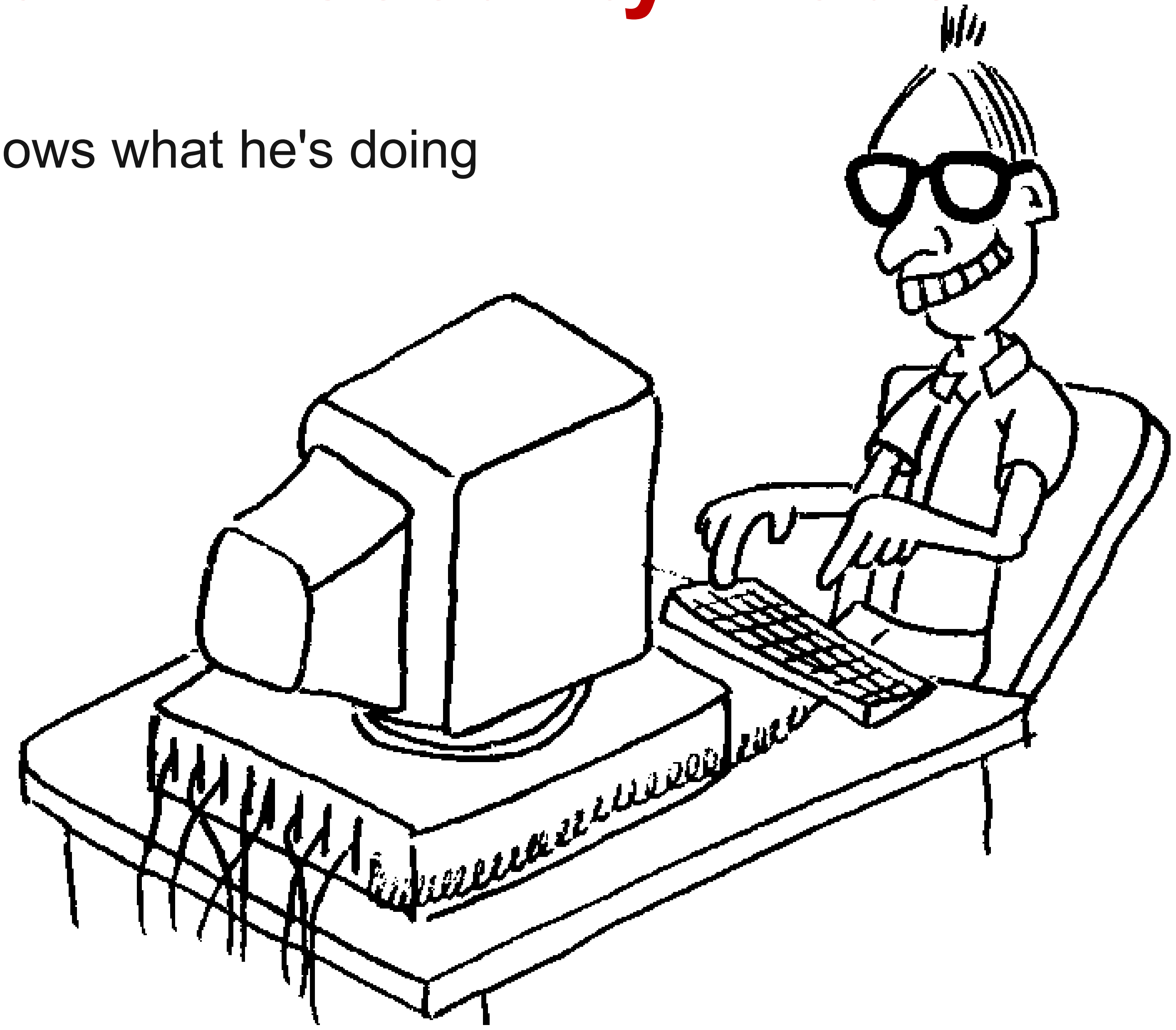
Clean security record
This site has a clean security record.

Classical PKI security model

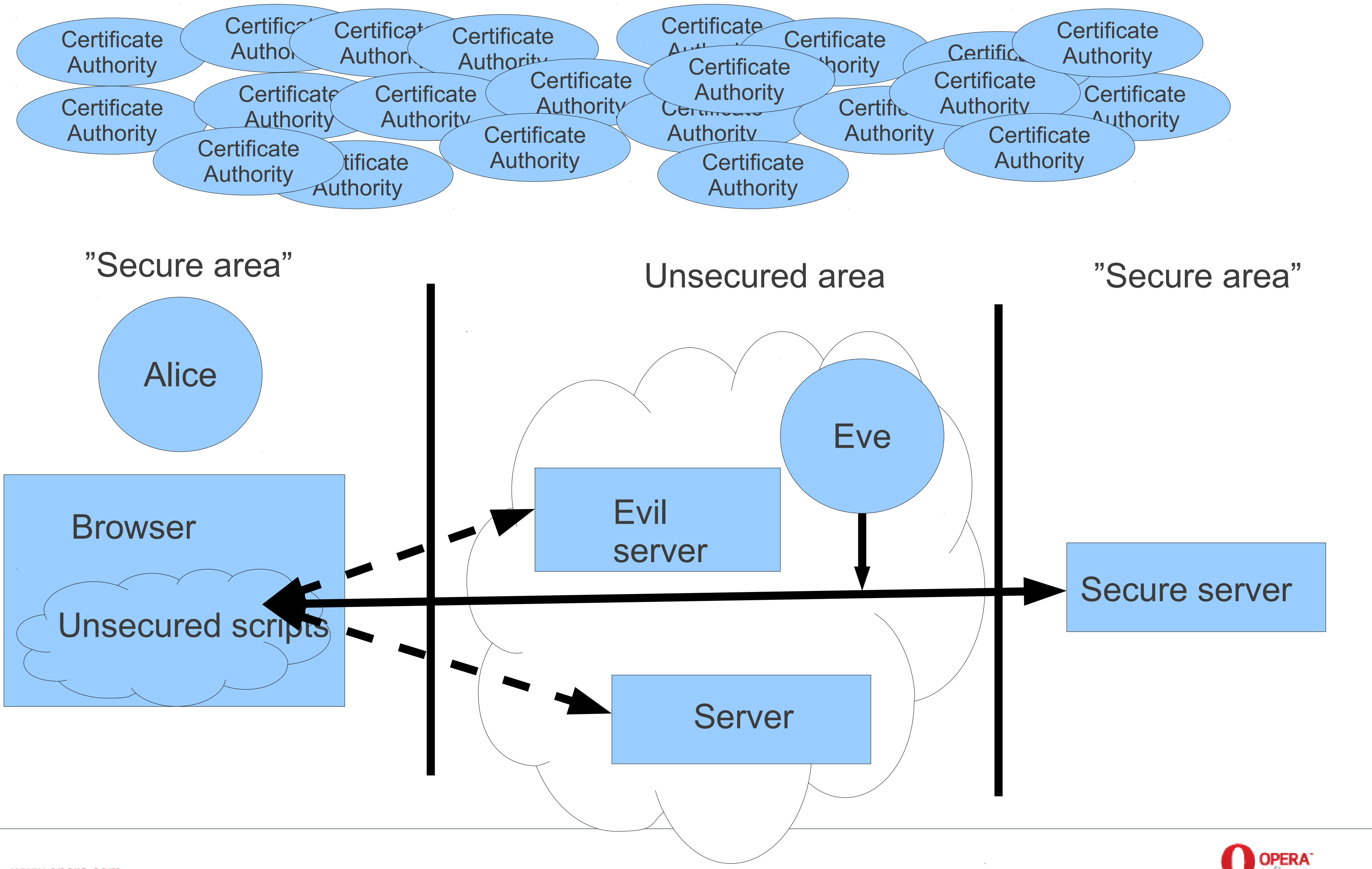


Classical PKI security model

- The user knows what he's doing



Browser PKI security model

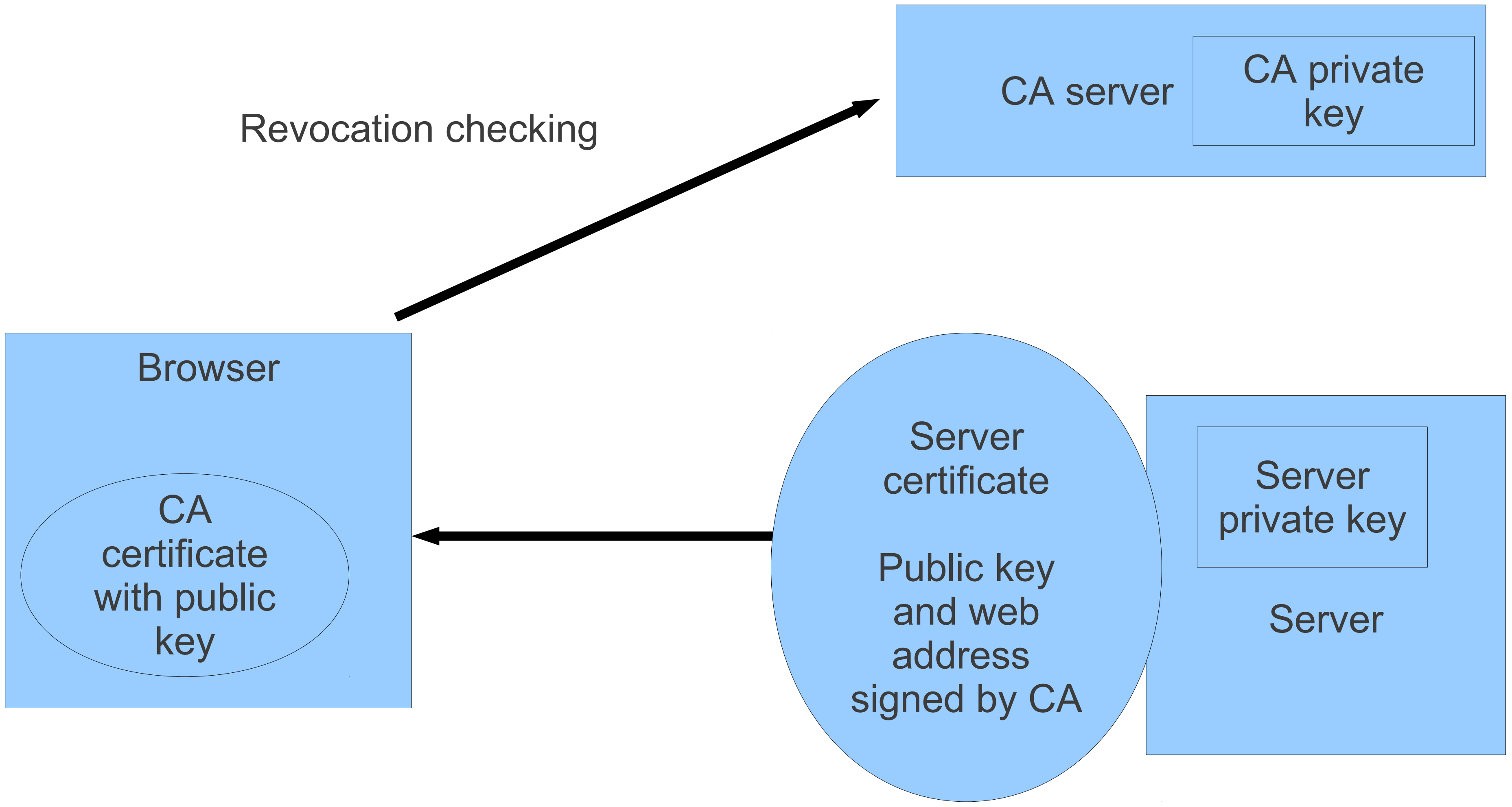


Browser PKI security model

- The user does not know what encryption is.

Attacks on the CA infrastructure

Certificate infrastructure basics



How a browser verifies a server

- Browser sends a request to server www.example.com
- Server responds with a certificate containing (among others) the public key and “www.example.com”, signed by a CA.
- The browser checks the signature using the public CA key from the CA certificate installed in the browser.
- The browser checks that the address “www.example.com” matches the address given in the server certificate.
- Using challenge-response techniques, the browser checks that the server really owns the private key of the certificate.

CA and certificate problems

- Site certificates are not issued publicly
- One CA can issue certificates for any website, even if a web page already has one.
- Inflation in numbers of Certificate Authorities
- Each CA can issue sub-CA certificates. We don't know how many exist.
- If *one single* CA or sub-CA is compromised, all websites are at risk of being compromised.

CA and certificate problems

- Any CA practically has a license to intercept any connection, by issuing false site certificates.
- An attacker than breaks into a CA, and issues site certificates can intercept any connection.
- Security is not better than the weakest CA
  - The high security of *your* CA has limited impact on your security

CA and certificate problems

- Can we trust all CAs?
- Can we trust all sub-CAs?
- Do *all* CAs and sub-CAs have sufficient security?

Organizational problems

- The SSL protocol did not take commercial interests and organizational issues into account.
  - «It's just a protocol»
- If one major browser installs a CA certificate, the others follows due to commercial interests. No browser can afford that major web sites break.
  - «Race to bottom»
- CA infrastructure specification is developed by the private organization CABForum, which is dominated by CAs and browser vendors.
  - Commercial interests, veto rights and such makes it hard to reach «bold» decisions

What does it take to become CA in Opera?

- The Certification Authority must document a satisfactory audit by a recognized practitioner.
  - An accredited auditor that can audit according to «The WebTrust program for Certification Authorities» or various ETSI or ISO programs
  - CAs that are operating as part of a national government may be audited by a government auditor
- Certificates signed by the root certificates must have a significant presence on the public Internet.
- Technical requirements of the certificate, like key lengths, expiration dates etc.

Serious CA incident 1

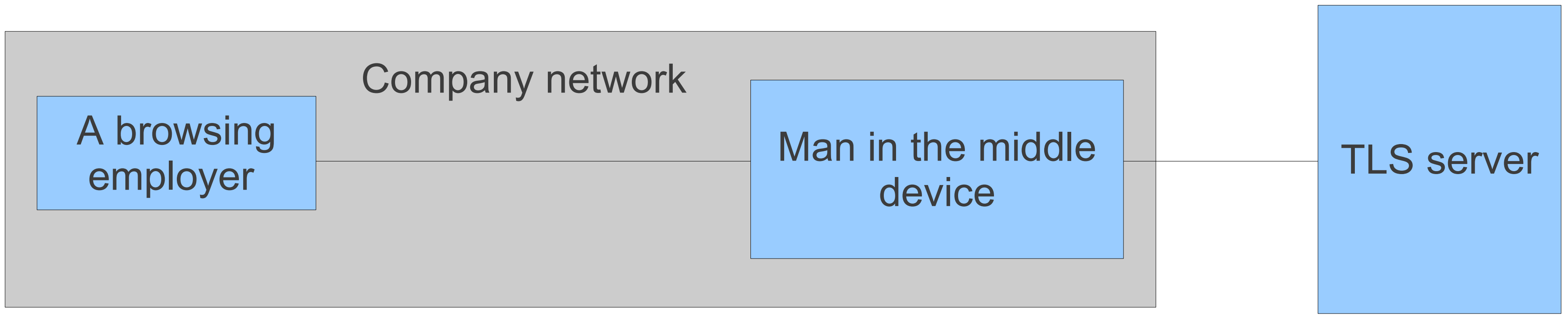
- Summer 2011, the Comodo CA incident.
  - Servers were compromized, false certificates were issued
    - Google, gmail, facebook etc...
  - Comodo reacted relatively quickly and notified relevant parties, and fixed the issues.
  - Comodo is still a browser CA.

Serious CA incident 2

- Summer 2011, Diginotar CA
  - Servers were compromized, false certificates were issued.
    - Google, gmail, facebook etc...
  - False certificates where used to attack users in Iran
    - Chrome (Googles browser) detected the attacks due to certificate pinning of their servers.
  - Diginotar tried to silently fix the problems, and did not notify relevant parties.
  - Diginotar was removed from all browsers and is now bankrupt.

Serious CA incident 3

- The Trustwave incident, spring 2012
  - Trust wave created a device containing a sub-CA which was allowed to issue fake site certificates on-the-fly.
  - This allowed companies to monitor the network traffic of the company's employers, even on secure TLS pages.
  - If lost or misused, this man-in-the-middle device could be used to intercept on *any* network connection worldwide.
  - Did not loose browser CA status - Maybe they should have?



Bad private key generation

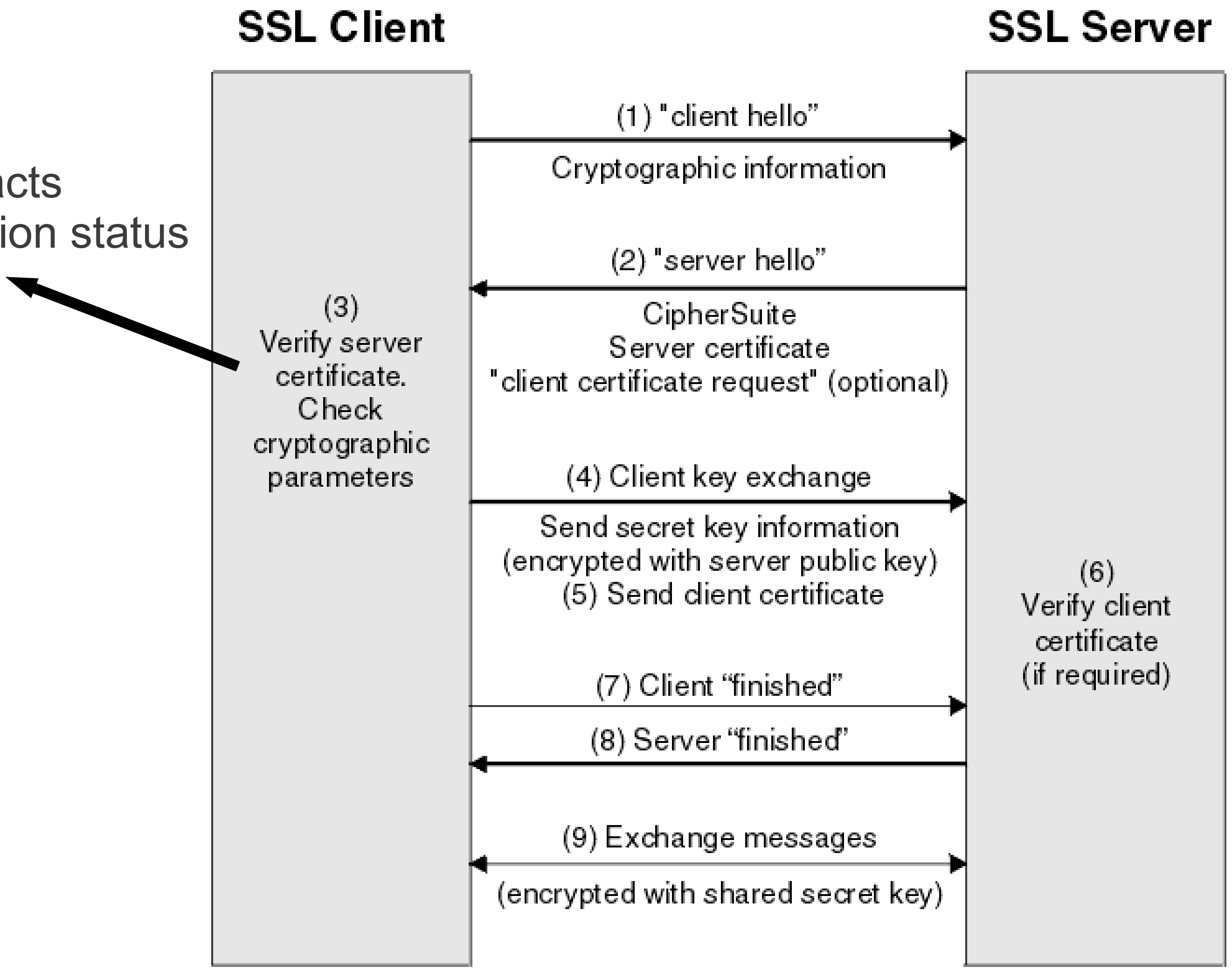
- “Ron was wrong, Whit is right”
  - Arjen K. Lenstra , James P. Hughes , Maxime Augier, Joppe W. Bos , Thorsten Kleinjung , and Christophe Wachter
- They simply ran greatest common divisor (GCD) algorithm on the public *n=pq* for millions of servers
- Many servers had used the same bad seed or random number generator for the prime factors.

- 2011 was a bad year for security
- We definitely get the impression that the CA infrastructure is failing.

Revocation of certificates

Revocation basics

At step 3, the browser contacts the CA to check the revocation status for given site certificate.


SSL Client
SSL Server
(1) "client hello"
Cryptographic information
(2) "server hello"
CipherSuite
Server certificate
"client certificate request" (optional)
(3) Verify server certificate. Check cryptographic parameters
(4) Client key exchange
Send secret key information (encrypted with server public key)
(5) Send client certificate
(6) Verify client certificate (if required)
(7) Client "finished"
(8) Server "finished"
(9) Exchange messages
(encrypted with shared secret key)

The certificate revocation problem

- Two parallel systems for handling revocation of site certificates:
  - Online Certificate Status Protocol (OCSP)
  - Certificate Verification lists (CRL)
- Both systems have the same problems
  - Connection, delay and performance problems
  - Implementation problems both on servers and browser clients
  - Privacy issues
- Google chrome will simply stop using OCSP and CRL

In the Comodo and Diginotar incidents, the revocation system failed.

All browser vendors ended up creating blacklists installed in the browsers and removing the CA.

Attempted fix: Browser CRL

- Instead of contacting the each CA, all revocation is handled by the browser vendor.
- Can be implemented by the browser vendor independently from CAs
  - No backward compatibility issues
- Certificates can now be revoked by the browser vendors independently from the CAs
- Revocation happens independently from browsing
- Harder for an attacker to block the revocation requests

Attempted fix: Googles CA transparency

- Publicly trusted certificates should be public knowledge
  - All issued certificates will be published in public logs, and it's not possible to
- Certificates are registered in number of append-only logs, not necessarily by the CA
  - The whole history of site certificates will be kept.
- Servers include a proof of registration which clients check and can asynchronously validate

Attempted fix: Convergence

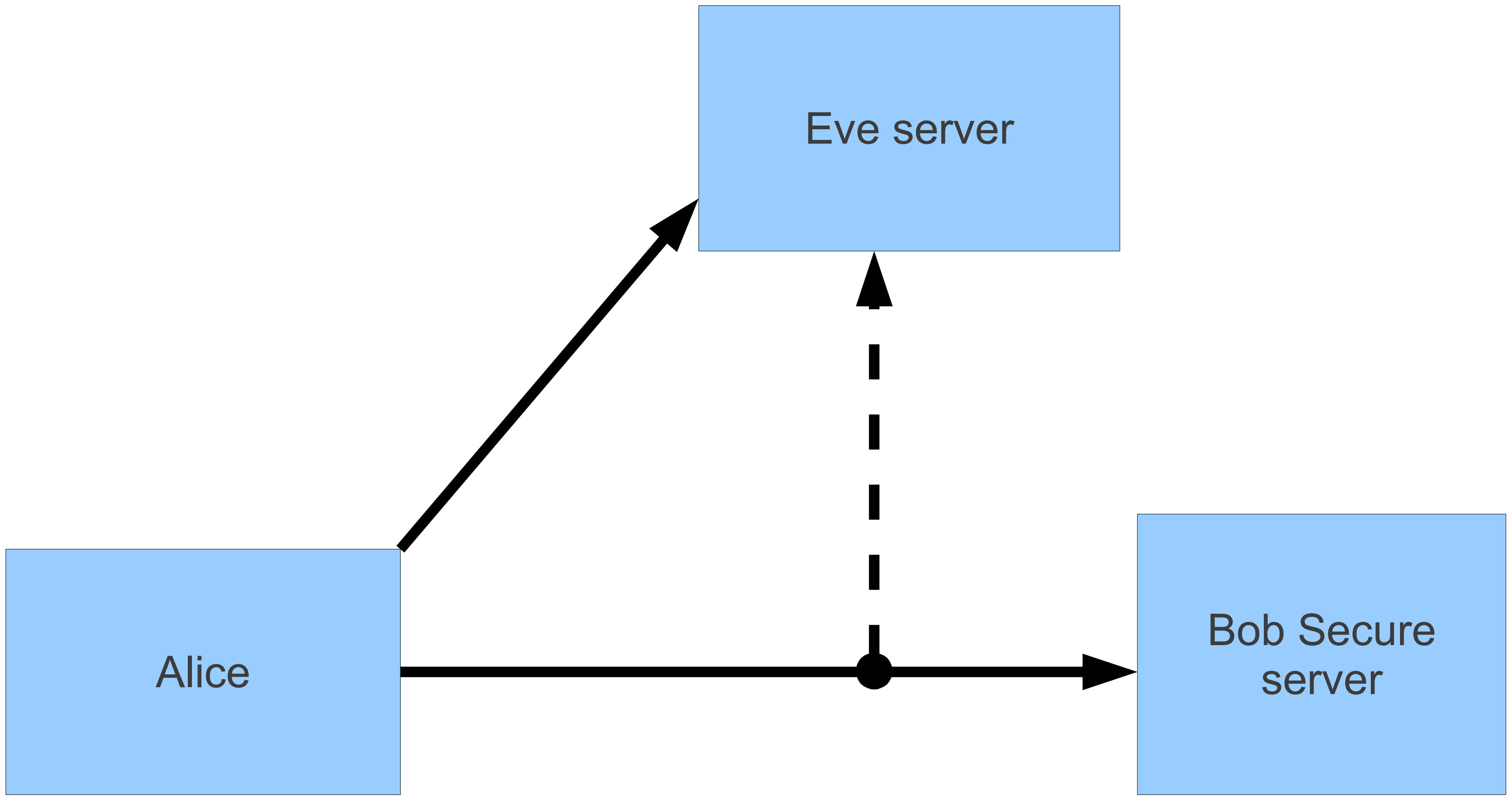
- Would remove the need of CAs
- The servers can issue their own site certificates
- The browser certificate checks with external «notaries» which compares the certificate the browser see with what it sees.
- The users choose which notaries to trust
- Will probably never be implemented
  - Demands too much of the user

Attacks on the SSL protocol

BEAST

- Juliano Rizzo and Thai Duong 2011
- One of few attacks on a TLS encryption algorithms.
- It shows the weaknesses of using classical security model on browsers.

Chosen plaintext attack



Preconditions for successful attack

- The attacker must be able to eavesdrop on network connections made from the victim's browser.
- The attack must be able to insert JavaScript into the victim's browser.
- The attacker must be able to send HTTPS requests at will.
- After listening in on the request, the attack must be able to append more data to the very same request.

Beast details

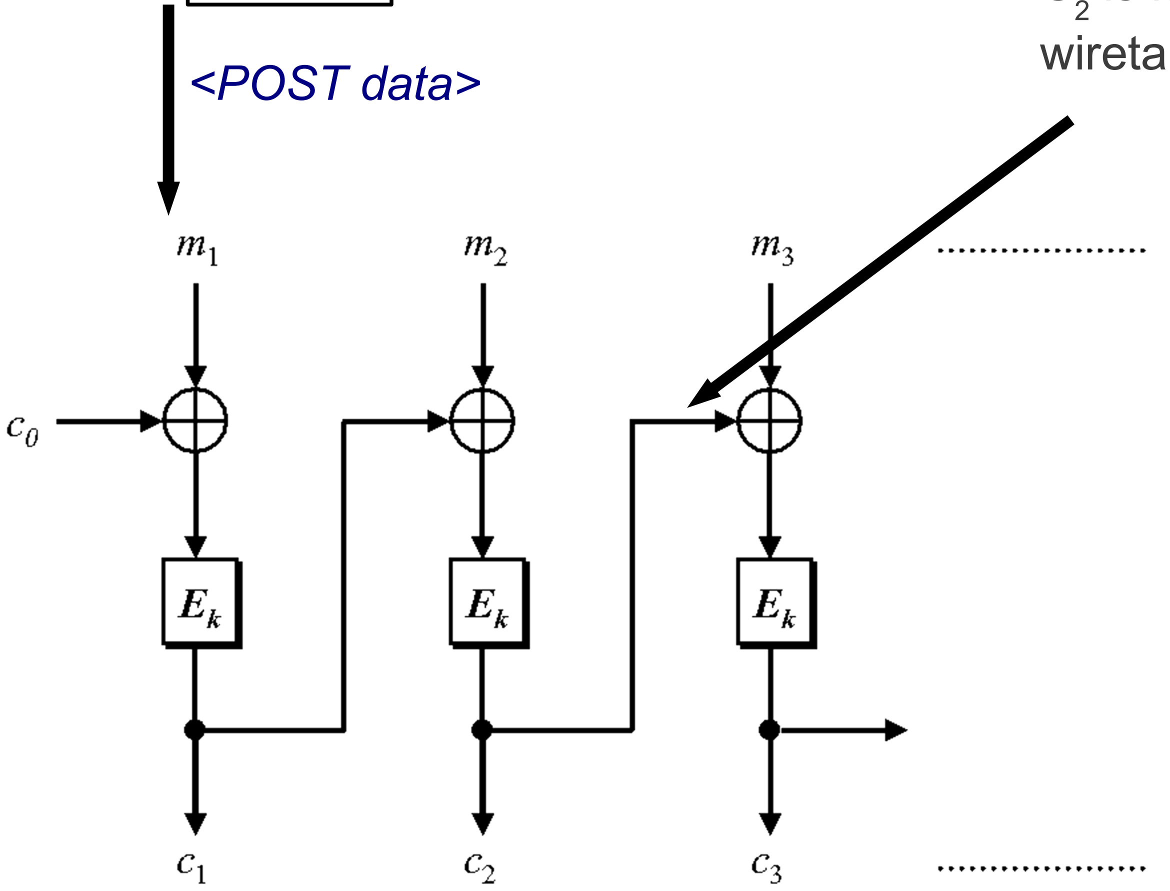
- Attacks CBC mode, where the IV for next encryption is the result of the previous encryption.
- The victim must be logged in to a secure web page, authenticated with a cookie.
- The victim must simultaneously load an 'evil' page.
- The 'evil' page issues requests to the secure web page
  - Chosen plaintext attack
- The goal of the attack is to reveal the secret session cookie.

CBC mode attack

POST /a...aaaa/index.html HTTP/1.1
Host: min-bank.no
Cookie: *12ehasaf3rfa*

<POST data>

C_2 is known due to wiretapping



Missing the point...

- Taher Elgamal, One of the designers of SSL and inventor of the ElGamal discrete log cryptosystem:

”

Now, from a practical standpoint, the real problem is you have to have malware on the machine. Honestly, if I can put malware on your machine, I'm not going to be bothering with your SSL because I can see all the data before it gets encrypted.

Defense against the Beast

- Upgrade to TLS 1.1/1.2
  - TLS 1.1/1.2 uses random IV for each encrypted record.
  - Not yet realistic. Few servers support TLS 1.1/1.2.
- Fix SSL 3.0/TLS 1.0 client side
  - Simulate TLS 1.1/1.2 by splitting each n-byte SSL record i into records of length 1 and n-1 bytes.
  - The splitting 'ruins' the known C_0.
  - This will result in a few but acceptable compatibility problems

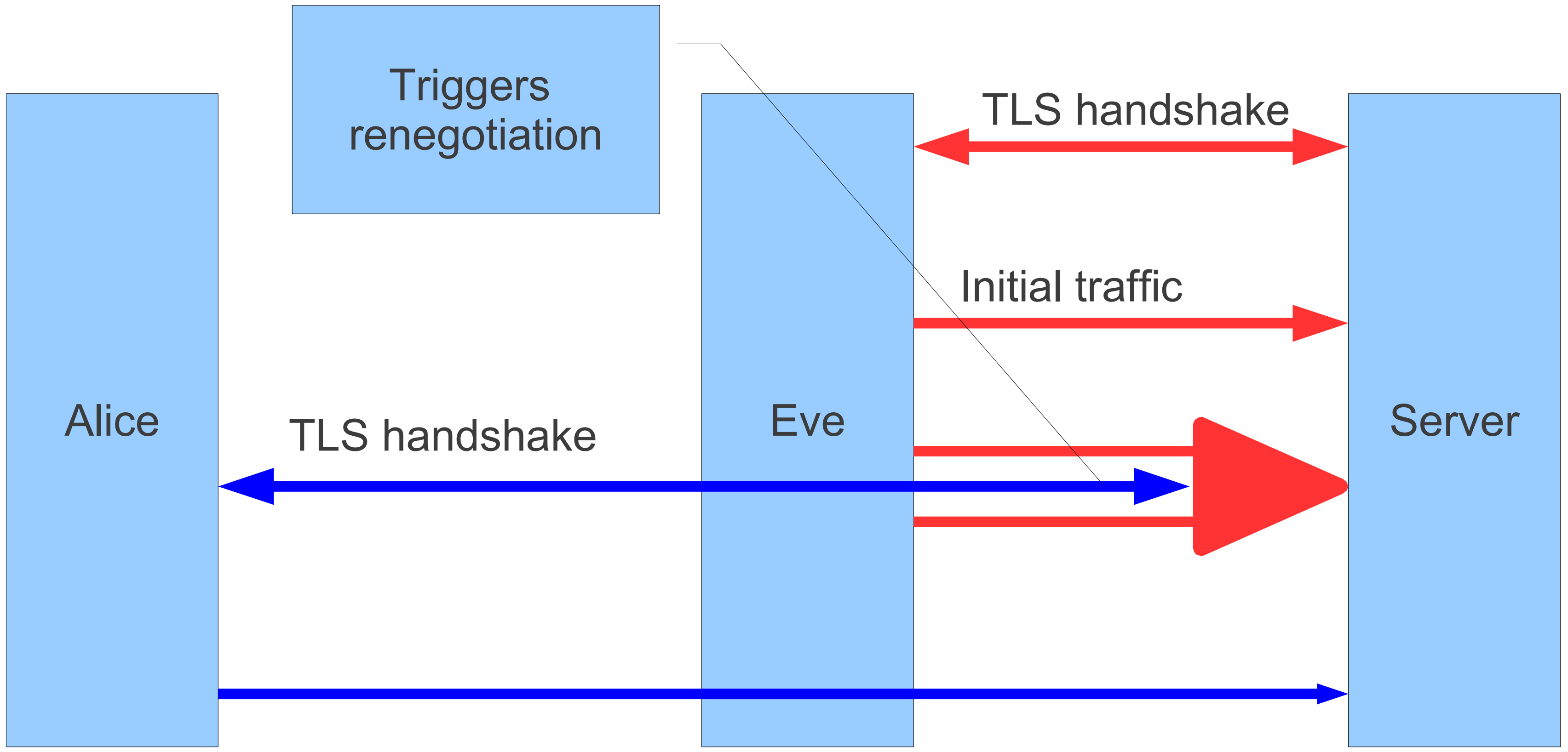
Is Beast practical?

- By default Opera was not vulnerable
  - Same origin policies, and vulnerable websocket protocol is off by default.
  - It's hard add more known plaintext data on same connection
- Juliano Rizzo and Thai Duong demonstrated the attack using java plugin.
  - Java is fixed and now has working same origin policy.

TLS renegotiation attack

- Plaintext injection attack.
- Uses a vulnerability in the TLS renegotiation logic.

TLS renegotiation attack



The vulnerability is caused by a missing association between initial connection and renegotiated connection.

How the attack works

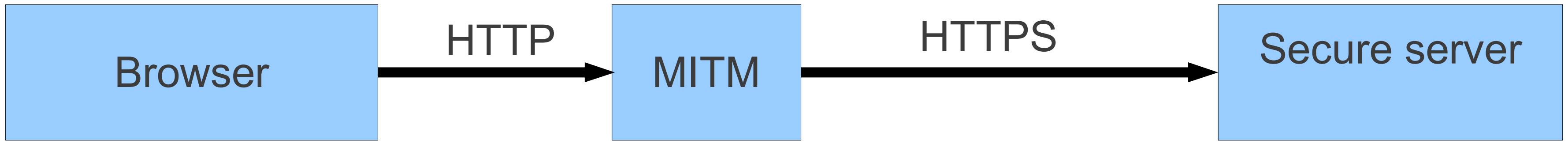
- Eve sends
  - GET /pizza?address=attackersaddress HTTP/1.1
    X-Ignore:
- Alice sends
  - GET /pizza?address=victimssaddress HTTP/1.1
  - Cookie: victimscookie
- Eve grabs the encrypted data from victim and sends it to the server through the existing connection.
- The server decrypt the Eves request, perform renegotiation with Alice (who does not notice) and decrypts Alice's request:
  - GET /pizza?address=attackersaddress HTTP/1.1
    X-Ignore: GET /pizza?address=victimssaddress HTTP/1.1
    Cookie: victimscookie

Defense against renegotiation attack

- The server can turn off renegotiation support
- Server and client may support an SSL extension with random data. The renegotiation connection must know this secret.
- No browser enforces this extension yet.
  - Too many servers will fail.

HTTPS stripping attack

- Attack on the initial HTTP request.
  - A man in the middle can simply grab the HTTP request, and make sure the connection is never upgraded to HTTPS.
  - The attacker then opens up HTTPS connection to the secure server, and servers it as HTTP to the browser client.



- The attack is crude, and the user will not see the padlock icon.
  - However, since no warning is showed, this kind of attack is surprisingly successful.

HTTP Strict Transport Security (HSTS)

- Measure to prevent HTTPS stripping attack
- On first HTTPS request, a HSTS server will respond with a HSTS header.
- From that point on, the browser will never connect to the server using HTTP
  - All HTTP requests will be rewritten to HTST internally in the browser
- The browser is still vulnerable on the first connect to a new server.

Why is not TLS 1.1 and 1.2 used?

- Chicken and egg problem
- Browsers support TLS 1.1/1.2, but have turned it off by default due to bad server TLS implementations.
    - Old/bad TLS clients tends to accept any TLS version, thus negotiating a version they do not support.
- Servers do not support TLS 1.1/1.2 since browsers have turned it off.

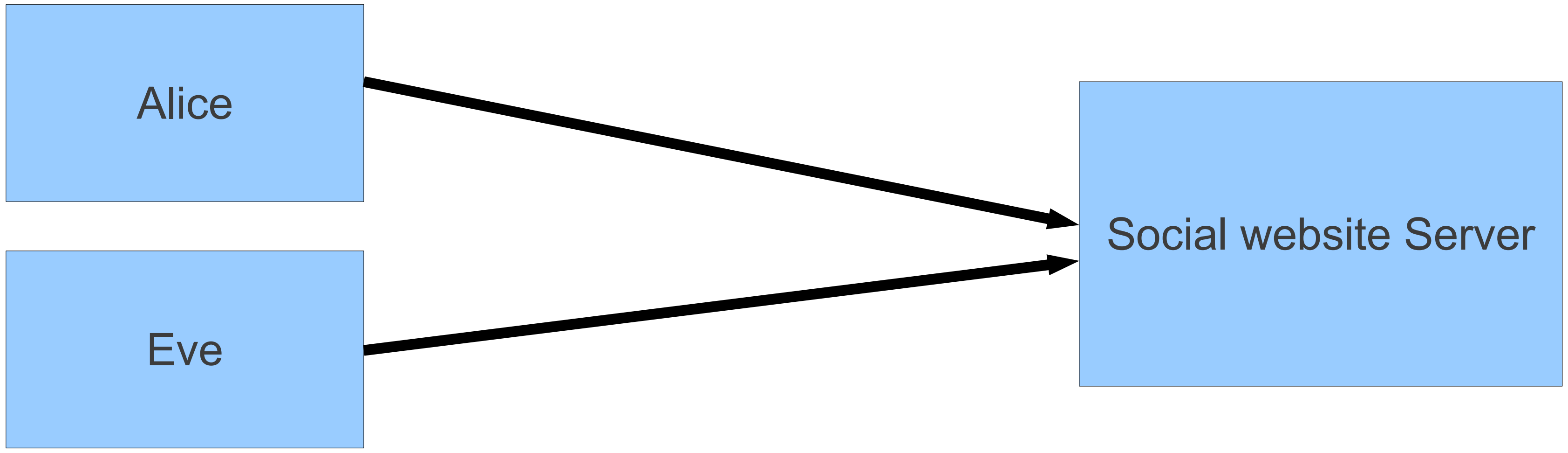
Other security issues

Cross Site Request Forgery (XSRF)

- Assume a user Alice has two open tabs:
  - Bob's good page https://www.bob.com
  - Eve's evil page www.evil-evel.com
- Alice is logged into Bob's server and authentication mechanism is sessions cookies.
- Eve's web page can now issue https requests from Alices browser to Bob's server
  - The browser will automatically attach the session cookie to the forged request
- Example: http://www.bob.com/delete_all
- Defense: add secret URL argument in addition to session cookie.
  - https://www.bob.com/?secret_arg=adfsfbrgseg

Cross site scripting (XSS)

- Alice is connected to social website, and views content posted by others.
- Eve sends <script> “evil script” </script> to the website.
- Server does not properly sanitize the post from Eve
- Alice views Eve's post and loads the script
- The script now has access to content on Alice's web page



Fingerprinting

- A browser reveals quite a bit information about the user's system
  - Screen size,
  - timezone,
  - which plugins does the browser support,
  - which fonts are installed
  - Which features are on
  - Browser vendor and version
- https://panopticlick.eff.org/
  - My browser on this computer revealed 21.5 bits of identifying information

Double click attack

- Users often double click on web page links, since that is what they are used to do in the operating system
- An web page can create a download or install link, and carefully make the web page such that the “INSTALL” button for installing appears at the same spot as the link
- This is fixed by delaying the appearance of the install button.

Buffer overflows

- Can give full access to install malware on user's computer.

Attack on plugins

- Java
- Flash