



Network and Computer Security

Practical Arrangements

Prof. dr. ir. Eli De Poorter

■ Opleidingsonderdeel

- **Beveiliging van netwerken en computers**
- **6 studiepunten**

■ New teacher, new slides

- **In other words: new course**
- **Mistakes might be present**
 - ▶ Grammar, spelling, unclarities, (hopefully not) content
 - ▶ Feel free to report these to me or using the subforum on Minerva
 - ✓ Feedback is always appreciated and will never be held against you!

■ Cursus

- Slides + notepages
- English slides
 - ▶ but Dutch course and examination

■ Minerva

- Slides under /documents
- Additional materials and links

■ Reference book

- William Stallings, “**Cryptography and Network Security, principles and practices**”, 6th (international) edition, Prentice Hall, 2010;
ISBN-13: 9780137056323
 - ▶ Not mandatory
 - ▶ Website: <http://williamstallings.com/Cryptography/>
 - ✓ Useful for errata and additional resources
 - ▶ Older editions (3rd, 4th, and 5th) can still be used
 - ▶ Not mandatory

■ Class hours

- **Theory**

- ▶ Friday, from 10:30 to 12:30, room B4016

- **Labo sessions**

- ▶ Wednesday, from 09:30 to 12:30

- ▶ Normally meant for exercises

- ▶ Also used for company visit and test

■ Literature

- **Alfred J. Menezes, Paul C. van Oorschot, and Scott A. Vanstone , “Handbook of Applied Cryptography”, CRC Press, 2001, ISBN 0-8493-8523-7**
 - ▶ Rather mathematical approach of cryptography (and somewhat older)
 - ▶ Freely available in pdf format at
<http://cacr.uwaterloo.ca/hac/>

■ Literature

- Matt Bishop, “**Computer Security: Art and Science**”, Addison Wesley, Pearson Education, 2003, ISBN-13: 978-0-201-44099-7
 - ▶ A true reference about security (1136 p.)
 - ▶ Also deals with aspects of security policy

■ Software

- Software allowing you to experiment with encryption
- PGP
 - ▶ <http://www.gnupg.org>
 - ✓ The GPL version
 - ▶ <http://www.openpgp.org/>
- OpenSSL
 - ▶ <http://www.openssl.org>
 - ▶ TLS & SSL + generic toolsets
- CrypTool
 - ▶ <http://www.cryptool.org/en/>
 - ▶ Educative cryptography software

■ Samenstelling

- **theorie (semesterexamen): 50%**
- **labo: 50%**
 - ▶ alleen raadpleging handleidingen en geselecteerde bronnen
 - ▶ voorbereiding
 - ▶ permanente evaluatie (voorbereiding, activiteiten, verslag)
 - ▶ 1 test (9/12/2014)

■ Minimumvereisten

- **7/20 voor zowel theorie als labo!**

■ Theorie examen

- **Schriftelijk (niet mondeling)**
- **Tekenen voor akkoord**

■ Teacher

- Prof. dr. ir. Eli De Poorter
- E-mail: eli.depoorter@intec.ugent.be
- Information Technology Department
- Gaston Crommenlaan 8, 9050 Gent (Ledeberg)
- Room 2.09b
- Meer info: <http://www.ibcn.intec.ugent.be/route.html>

■ Labo assistants

- Pieter-Jan Maenhaut, PieterJan.Maenhaut@UGent.be
- Wim Van Den Breen, Wim.VanDenBreen@UGent.be



Network and Computer Security

Chapter 1 - Introduction

Prof. dr. ir. Eli De Poorter

- Security in the media
- Recent major incidents
 - Heartbleed (2014)
 - Sony Pictures Entertainment hack (2014)
 - Ashley Madison (2015)
 - Others
- Why do we need security?
- Scope of the course



UNIVERSITEIT
GENT



What are we talking about?



INTEC

**What comes to mind when you hear
the term “Security”?**

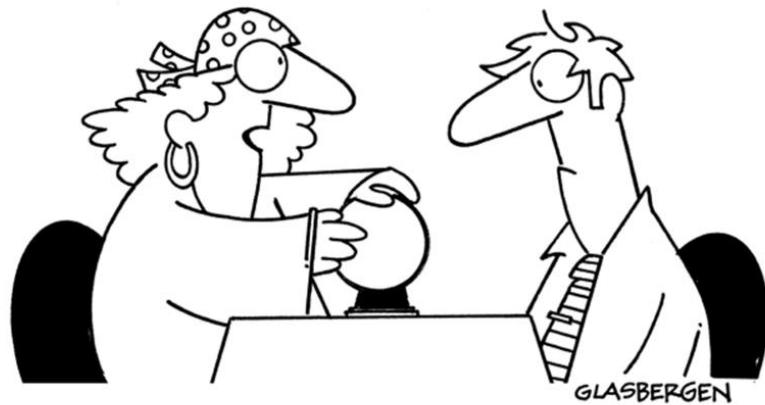


■ A few examples from the news

- “Social engineering”, Internet fraud, etc.
- Hackers
- Password security
- Privacy
- Security of confidential information
- Cybercrime, cyberterrorism, cyberwar, etc.
- Malware...
 - ▶ ...now also for MacOS and Android
- Did we mention the NSA?

- Security in the media
- Recent major incidents
- Why do we need security?
- Scope of the course

© Randy Glasbergen
glasbergen.com

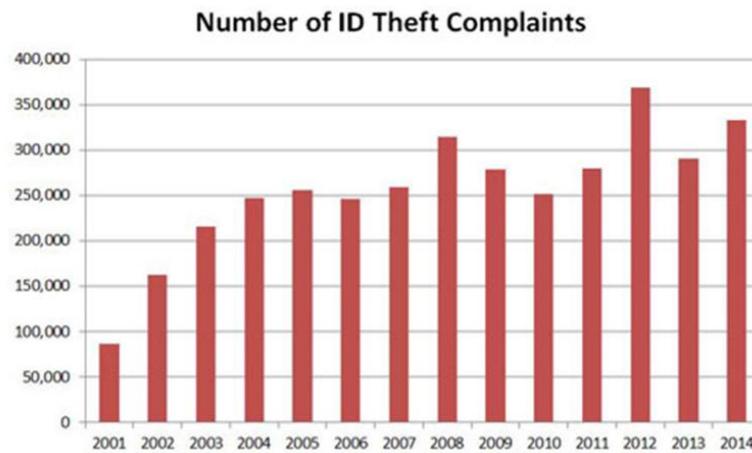


"I can't see your future, but I found your bank files, Social Security number and all of your company passwords."



Your Identity Is Worth \$5 on the Black Market
In other words, significantly less than it's worth to you....

<http://newsfeed.time.com/2013/08/26/your-identity-is-worth-5-on-the-black-market/>



http://www.idtheftawareness.com/id_theft_pages/WhatIsIdTheft.php

HACKERS RECENTLY LEAKED 153 MILLION ADOBE USER EMAILS, ENCRYPTED PASSWORDS, AND PASSWORD HINTS.

ADOBE ENCRYPTED THE PASSWORDS IMPROPERLY, MISUSING BLOCK-MODE 3DES. THE RESULT IS SOMETHING WONDERFUL:

USER PASSWORD	HINT
4e18acc1ab2262d6	WEATHER VANE SWORD
4e18acc1ab2262d6	NAME1
4e18acc1ab2262d6 a0x2876tibiaItta	DUH
8ebab6b77fe05646d	a0x2876tibiaItta
8ebab6b77fe05646d 85e7d81a2a78adc	57
4e18acc1ab2262d6	FAVORITE OF 12 APOSTLES WITH YOUR OWN HAND YOU HAVE DONE ALL THIS
1ab29ea86d6e5ca	SEXY EARLOBES
a1f912b6299e3a2b eoder1eb6b797397	BEST TOS EPISODE
a1f912b6299e3a2b 617ab0277727ad85	SUGARLAND
3f73832ad8048&f7 617ab0277727ad85	NAME + JERSEY #
1ab29ea86d6e5ca	ALPHA
977a7b894d386261	OBVIOUS
977a7b894d386261	MICHAEL JACKSON
977a7b894d386261	HE DID THE MASH, HE DID THE
38a7c2279code644 9dcald79d4dec6d5	PURLOINED
38a7c2279code644 9dcald79d4dec6d5	POV LATER 3 POKEMON
a0e57b05d2a7a7a 4dcald79d4dec6d5	

THE GREATEST CROSSWORD PUZZLE
IN THE HISTORY OF THE WORLD

<http://xkcd.com/1286/>
Adobe 2013

source: xkcd.com

Insurance giant Anthem hit by massive data

CNN Money

2015-02-05

<http://money.cnn.com/2015/02/04/technology/anthem-insurance-hack-data-security/index.html>

Personal data in database not always secure

DE
REDACTIE.BE

Persoonlijke gegevens in databases niet altijd veilig

2014-10-16

<http://deredactie.be/permalink/1.2120513>

Apple to beef up security measures after nude photo leak

CNN Money

2014-09-04

<http://money.cnn.com/2014/09/04/technology/security/apple-celebrity-photos/>

Bitcoin bank Flexcoin closes after hack

the guardian

2014-03-04

<http://www.theguardian.com/technology/2014/mar/04/bitcoin-bank-flexcoin-closes-after-hack-attack>

Clearly not the only issue
with bitcoin last year

USD per 1 XBT

5 Feb 2015 08:00 UTC
XBT/USD close 222.95394

10

Cheney's defibrillator was modified to prevent hacking



2013-10-24

<http://www.cnn.com/2013/10/20/us/dick-cheney-gupta-interview/>



2014-09-15 Stuart Carlson

washingtonpost.com

Russian Hackers Amass Over a Billion Internet Passwords

The New York Times 2014-08-05

<http://www.nytimes.com/2014/08/06/technology/russian-gang-said-to-amass-more-than-a-billion-stolen-internet-credentials>

Why I Am Skeptical About 1.2 Billion Passwords Being Stolen

Forbes

2014-08-07

Russian Hackers Amass Over a Billion Internet Passwords

The New York Times

2014-08-05

<http://www.nytimes.com/2014/08/06/technology/russian-gang-said-to-amass-more-than-a-billion-stolen-internet-credentials.html>



So you think you're safe?



NSA hacks Belgian cyberprof

NSA hackt Belgische cyberprof



2014-01-31

http://www.standaard.be/cnt/dmf20140131_049



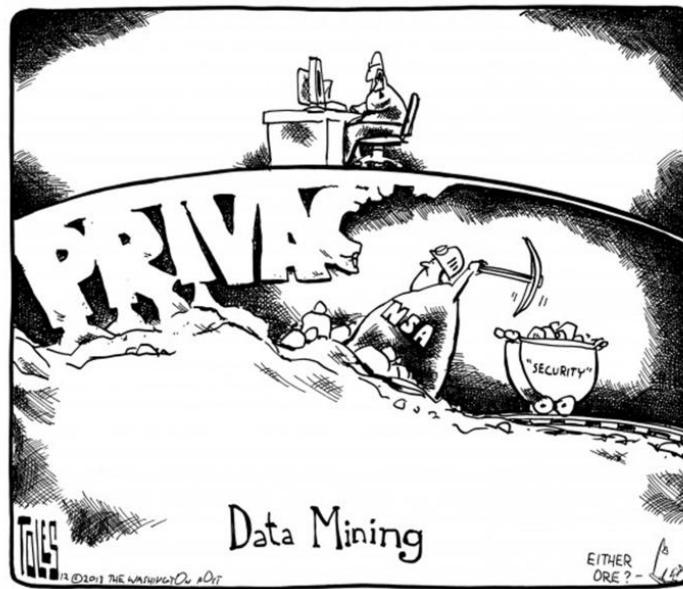
- A bit too much for a single article...
 - ...so let's have a full section in the newspaper
 - ▶ 1833 articles (2015-02-09 10:00) and counting...

NSA
The latest news and comment on the US National Security Agency
the guardian

Since 2013-06-06

<http://www.theguardian.com/us-news/nsa>

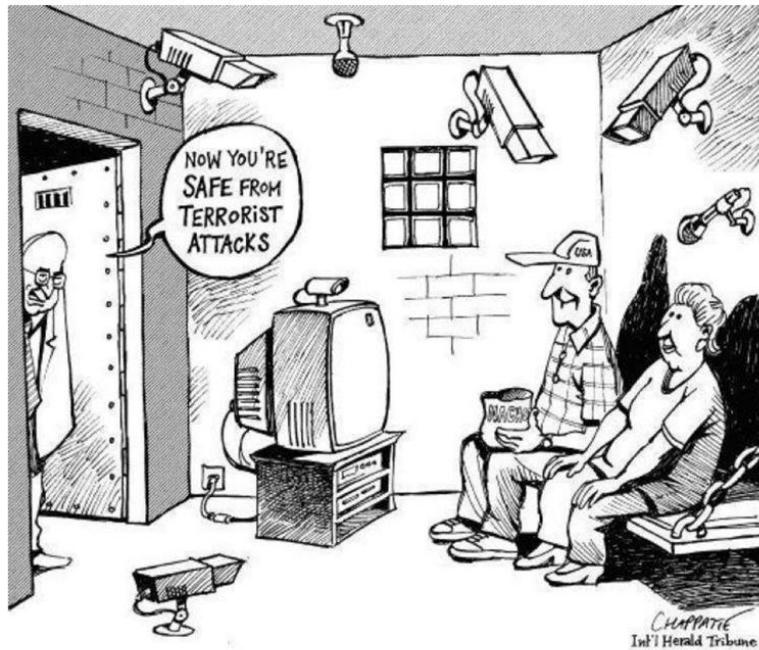
- Remain a critic, remain a sceptic
 - Journalists aren't always exactly IT experts!



washingtonpost.com

WAR IS PEACE FREEDOM IS SLAVERY IGNORANCE IS STRENGTH

George Orwell, "1984"



<http://www.prosebeforehos.com/political-ironing/01/03/national-security-america/>

19



Well, actually this is just a hoax



NSA hacks Internet forums, 'against law'

AIVD hackt internetfora, 'tegen wet in'

NRC HANDELSBLAD

2013-11-30

<http://www.nrc.nl/nieuws/2013/11/30/aivd-hackt-internetfora-tegen-wet-in/>

Revelations about the French Big Brother

Révélations sur le Big Brother français

Le Monde.fr

2013-07-04

http://www.lemonde.fr/societe/article/2013/07/04/revelations-sur-le-big-brother-francais_3441973_3224.html

British intelligence hacked Belgian telephone company

Britischer Geheimdienst hackte
belgische Telefongesellschaft

DER SPIEGEL

2013-09-20

<http://www.spiegel.de/netzwelt/web/belgacom-geheimdienst-gchq-hackte-belgische-telefongesellschaft-a-923224.html>



Consequences?



Here's what can go wrong when the government builds
a huge database about Americans

washingtonpost.com

2013-07-08

<http://www.washingtonpost.com/blogs/workblog/wp/2013/07/08/heres-what-can-go-wrong-when-the-government-builds-a-huge-database-about-americans/>

Every single IT guy, every single manager...

CROOKED TIMBER (blog)

2014-09-23

<http://crookedtimber.org/2014/09/23/every-single-it-guy-every-single-manager/>

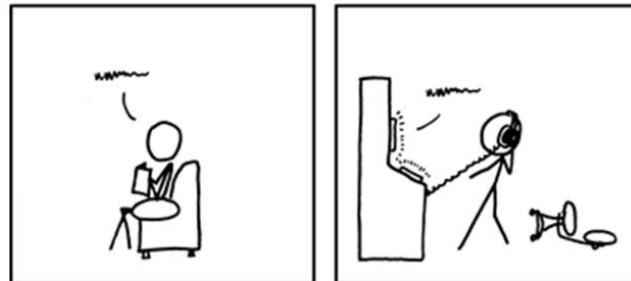
Quis custodiet ipsos custodes?
(Juvenalis, Satire 6.346–348)



2013-10-31 Ben Sargent

washingtonpost.com

NOW AND THEN, I ANNOUNCE "I KNOW
YOU'RE LISTENING" TO EMPTY ROOMS.



IF I'M WRONG, NO ONE KNOWS.
AND IF I'M RIGHT, MAYBE I JUST FREAKED
THE HELL OUT OF SOME SECRET ORGANIZATION.

<http://xkcd.com/525/>

Source: xkcd.com

Facebook signs users up to privacy policy that allows it to track you everywhere on the internet

2015-02-04

The INDEPENDENT

<http://www.independent.co.uk/life-style/gadgets-and-tech/news/facebook-signs-users-up-to-privacy-policy-that-allows-it-to-track-you-everywhere-on-the-internet-10022530.html>

Does Uber Even Deserve Our Trust?

Forbes

2014-11-25

<http://www.forbes.com/sites/chanellebessette/2014/11/25/does-uber-even-deserve-our-trust/>

Apple pushes out first-ever automatic security upgrade for Mac

CNN Money

2014-12-23

<http://money.cnn.com/2014/12/23/technology/security/apple-automatic-security-upgrade/index.html>

Number of viruses on Android smart phones increases spectacularly

Aantal virussen op Android-smartphones stijgt spectaculair DeMorgen.

2013-11-27

<http://www.demorgen.be/technologie/aantal-virussen-op-android-smartphones-stijgt-spectaculair-a1748265/>

Internet bank fraud increased by 70% in 2013

DE
REDACTIE.BE

Fraude met internetbankieren steeg met 70% in 2013

2014-02-10

<http://www.deredactie.be/permalink/1.1869606>

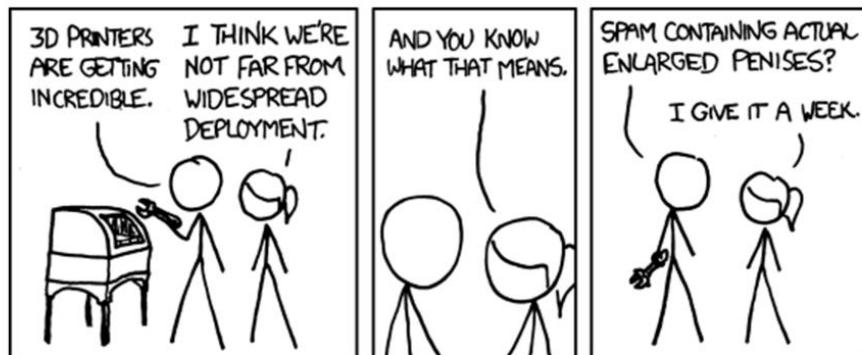
Number of Internet bank fraud cases strongly decreased

DE
REDACTIE.BE

Aantal fraudegevallen met internetbankieren daalt sterk

2015-01-30

<http://deredactie.be/permalink/1.2223667>



<http://xkcd.com/924/>

Source: xkcd.com



<http://xkcd.com/932/>

Source: xkcd.com



UNIVERSITEIT
GENT

TOM THE DANCING BUG

BY
RUBEN
BOLLING



THIRD IN A SERIES OF GOVERNMENT INFORMATION BROCHURES

YOUR government, working for YOU!

**YOU are
a computer
criminal!**



**Yes, you!
EVERYBODY
is!**



Me??



Computer criminal For example, if you have ever statutes are written so visited a website and failed to broadly, people violate follow its Terms of Service, them every day. you committed a **FEDERAL CRIME**.

And even if you didn't, there are thousands of other crimes we can charge you with.

This keeps America and its beloved corporate institutions safe.

Because if we find a Bad Guy, we don't have to figure out whether he broke this law or that; we charge him with breaking the laws **EVERYBODY** breaks!

FREQUENTLY ASKED QUESTIONS

- Q. So... am I a "Bad Guy"?
A. **YOU CAN TRUST** Federal Prosecutors to decide that.
Q. What do I do if I am charged with a computer crime?
A. **YOU CAN TRUST** Federal Prosecutors to offer a punishment that is just and fair.
Q. What about judges and juries?
A. You can take your case to them, but only by risking life-ruining, decades-long prison sentences.
- Q. How can the law allow such severe penalties for things innocent people do every day?
A. Hmm. You ask questions a bit too frequently. Are you a Bad Guy?
Q. I'm done asking questions.

IMPORTANT

If you are a *corporate executive* charged with ANY crime, identify yourself to your Prosecutor immediately, so that we may send your company a bill, and send you on your way.

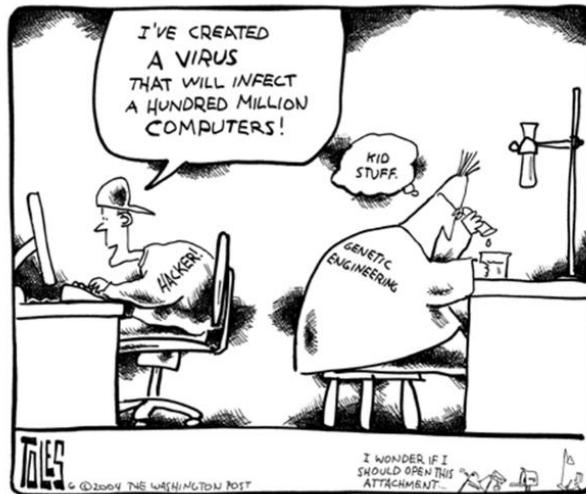


When everyone's a criminal, we're all safe from Bad Guys!

©2013 R. BOLLING - 1123 - www.tomthedancingbug.com - R.I.P. Inner Hive member Aaron Swartz

Source:
dailykos.com

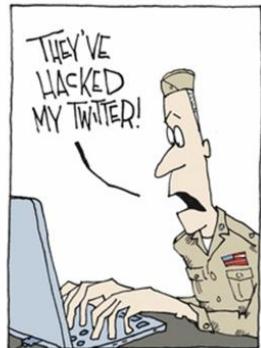
■ Cyberterror our main preoccupation?



2004-06-01 Tom Toles

washingtonpost.com

■ Cyberterror our main preoccupation?



2015-01-15 Signe Wilkinson

washingtonpost.com



There goes the 768 bits key

Exit la clé de 768 bits

LE SOIR

2010-01-09

http://archives.lesoir.be/festin-sous-marin-cryptographie-exit-la-cle-de-768-bits_t-20100109-00RQKV.html

Largest prime number ever discovered

Grootste priemgetal ooit ontdekt

DE
REDACTIE.BE

2013-02-06

<http://www.deredactie.be/permalink/1.1542554>

33

Grootste priemgetal: $2^{57}885161-1$ (17425170 decimale digits)

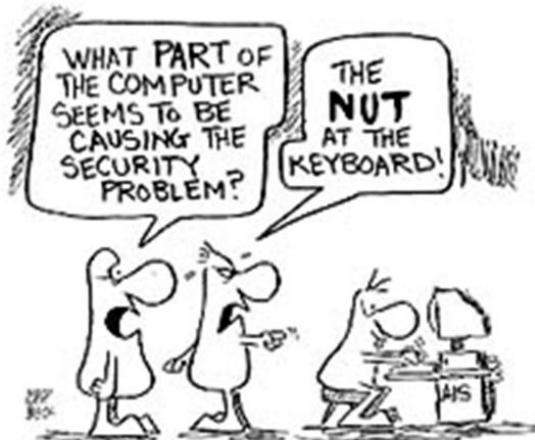
■ Secure or user-friendly?



2009-02-03 On the Fastrack

washingtonpost.com

- ...the human factor

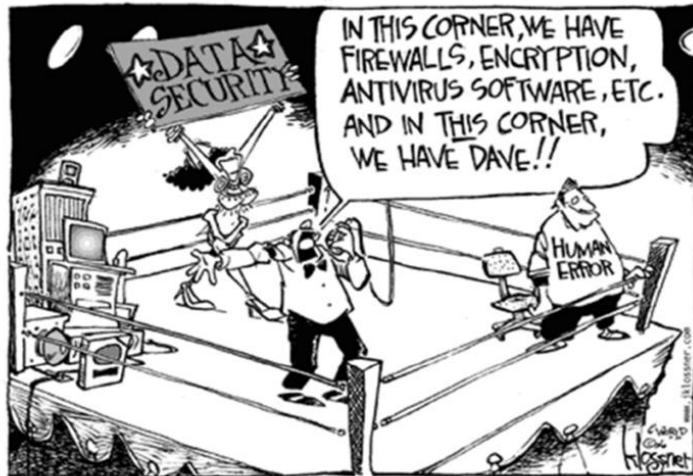


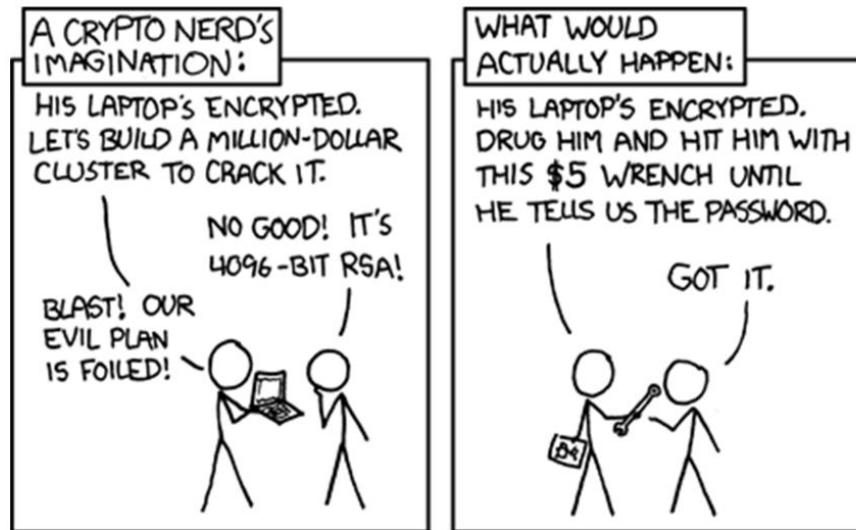
1 in 10 in a survey think HTML is an STD

2014-03-04

<http://www.latimes.com/business/technology/la-fi-tn-1-10-americans-html-std-study-finds-20140304-story.html>

- ...the human factor





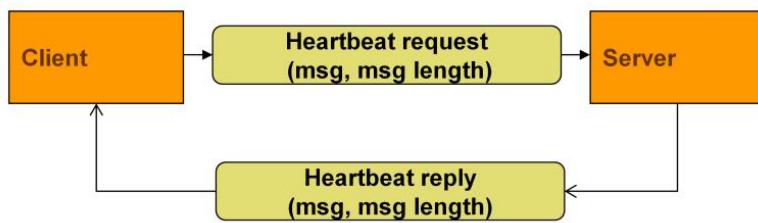
Source: xkcd.com

<http://xkcd.com/538/>

- Security in the media
- Recent major incidents
 - Heartbleed (2014)
 - Sony Pictures Entertainment hack (2014)
 - Ashley Madison (2015)
 - Others
- Why do we need security?
- Scope of the course

- **Heartbleed is a vulnerability in OpenSSL software.**
- **OpenSSL is encryption software that accesses websites through a “secure” connection**
 - E.g. **HTTPS://**.
- **Https uses SSL or TLS for encrypting sensitive data**
 - Banking, e-shopping, etc.
 - A heartbeat is send regularly to check if the other party is alive
 - ▶ For efficiency (to avoid setting up a new connection)
 - ▶ For safety (to check if we can still communicate)
- **Heartbeat**
 - **Short block of data + length of the data**

■ Heartbeat: keep alive a secure TLS connection



The Heartbeat Extension for the Transport Layer Security (TLS) and Datagram Transport Layer Security (DTLS) protocols was proposed as a standard in February 2012 by RFC 6520. It provides a way to test and keep alive secure communication links without the need to renegotiate the connection each time. The RFC 6520 Heartbeat Extension tests TLS/DTLS secure communication links by allowing a computer at one end of a connection to send a "Heartbeat Request" message, consisting of a payload, typically a text string, along with the payload's length as a 16-bit integer. The receiving computer then must send exactly the same payload back to the sender.

- Attackers create a forged heartbeat request with larger msg length
- The server does not check the actual length and replies with data from its memory

```

0700: BC 9C 2D 61 5F 32 36 30 35 26 2E 73 61 76 65 3D ...-a_2605&.save=
0710: 26 70 61 73 73 77 64 5F 72 61 77 3D 06 14 CE 6F &passwd_rawn...o
0720: A9 13 96 CA A1 35 1F 11 79 28 20 BC 2E 75 3D 63 ....S..y+ ..u=c
0730: 6A 66 6A 6D 31 68 39 6B 37 6D 36 30 26 2E 76 3D jfjm1h9k7m608.v=
0740: 30 26 2E 63 68 61 6C 6C 65 66 67 65 3D 67 7A 37 .08.challenge=gz7
0750: 6E 38 31 52 6C 52 4D 43 6A 49 47 4A 6F 71 62 33 n81R1MKcJIGJoqb3
0760: 75 69 72 61 2E 6D 6D 36 61 2E 79 70 6C 75 73 uira.mmb6a&.plusl
0770: 3D 26 2E 65 6D 61 69 6C 43 6F 64 65 3D 26 70 6B -&.emailCode=&pk
0780: 67 3D 26 73 74 65 70 69 64 3D 26 2E 65 76 3D 26 g=&stepid=&.ev=&
0790: 68 61 73 4D 73 67 72 3D 30 26 2E 63 68 68 50 3D hasMsgr=&8.chk=
07a0: 59 26 25 64 6F 6E 65 3D 68 74 74 70 25 33 41 25 Y&.done=http%3A%
07b0: 32 46 25 32 46 6D 61 69 6C 2E 79 61 68 6F 67 2E 2F%2Fmail.yahoo.
07c0: 63 6F 6D 26 2E 70 64 3D 79 60 5F 76 65 72 25 33 com&.pd=y_m_ver%3
07d0: 44 38 25 32 36 63 25 33 44 25 32 36 69 76 74 25 DB%26c%3Dx261vt%
07e0: 33 44 25 32 36 73 67 25 33 44 26 2E 77 73 30 31 3D%26sg%308.ws=1
07f0: 26 2E 63 70 3D 38 26 6E 72 3D 38 26 76 61 64 3D 8..cp=&n=8&pad=
0800: 36 26 61 61 64 3D 36 26 6C 6F 67 69 6E 3D 61 67 68aa=&81login=ag
0810: 6E 65 73 61 64 75 62 6F 61 74 65 6E 67 25 34 38 nesaduboadteng%40
0820: 79 61 68 6F 6F 2E 63 6F 6D 26 70 61 73 73 77 64 yahoo.com&passwd=
0830: 3D 30 32 34 =824 &.pe

```



The affected versions of OpenSSL allocate a memory buffer for the message to be returned based on the length field in the requesting message, without regard to the actual size of that message's payload. Because of this failure to do proper bounds checking, the message returned consists of the payload, possibly followed by whatever else happened to be in the allocated memory buffer. Heartbleed is therefore exploited by sending a malformed heartbeat request with a small payload and large length field to the vulnerable party (usually a server). The malformed block says its length is 64KB, the maximum possible. The server copies that much data from memory into the response, permitting attackers to read up to 64 kilobytes of the victim's memory that was likely to have been used previously by OpenSSL. This memory is likely to contain sensitive information, such as passwords or even the public or private key information. Since the attack is not logged in the server database, attackers can use this bug undetected. Since the contents of the memory change over time, the attack can be performed multiple times to gradually obtain more information. Moreover, the obtained information can even be used to decrypt message exchanges from the past.

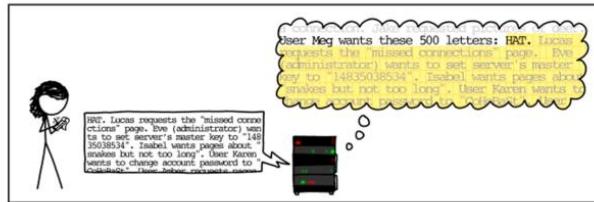
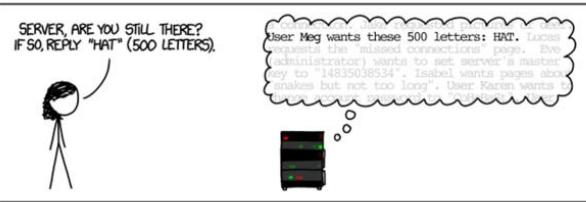
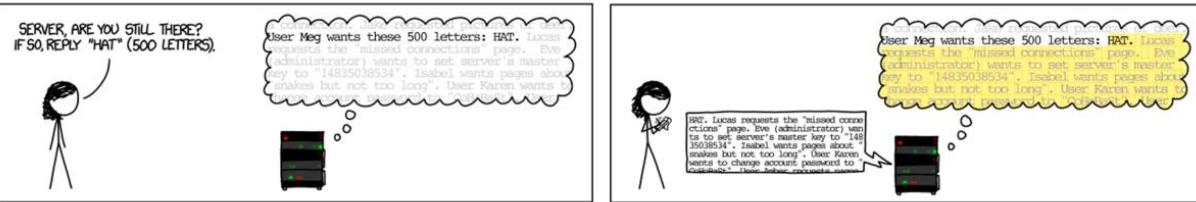
Note that heartbleed is not a flaw in TLS: it is a simple memory safety bug in OpenSSL!

More information from

<http://www.seancassidy.me/diagnosis-of-the-openssl-heartbleed-bug.html>

<http://blog.cryptographyengineering.com/2014/04/attack-of-week-openssl-heartbleed.html>

<https://vimeo.com/91425662> (video)



<https://xkcd.com/1354/>

42

Where a Heartbeat Request might ask a party to "send back the four-letter word 'bird'", resulting in a response of "bird", a "Heartbleed Request" (a malicious heartbeat request) of "send back the 500-letter word 'bird'" would cause the victim to return "bird" followed by whatever 496 characters the victim happened to have in active memory. Attackers in this way could receive sensitive data, compromising the confidentiality of the victim's communications. Although an attacker has some control over the disclosed memory block's size, it has no control over its location, and therefore cannot choose what content is revealed.

OpenSSL released

March 2012

Patch released

21 March 2014

(Some fixes had already been put in place then)

Publicly reported as vulnerable

1 April 2014

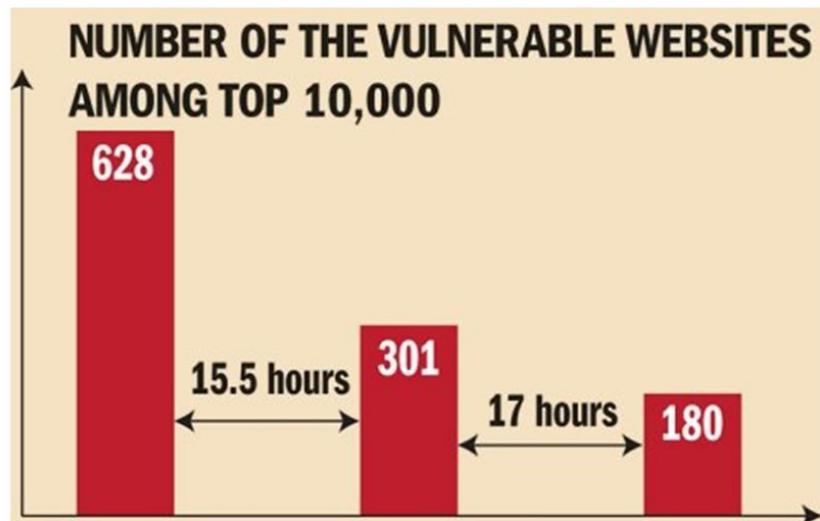
First proven attempted exploit

8 April 2014

Intentional vulnerability test

12 April 2014

How may sites are vulnerable?
(After vulnerability was reported publically)



Historical Trend of Vulnerable HTTPS Enabled Alexa Top 1 Million Websites



A list the top 1,000 most popular web domains and mail servers that remain vulnerable.
<https://zmap.io/heartbleed/>

- Security in the media
- Recent major incidents
 - Heartbleed (2014)
 - **Sony Pictures Entertainment hack (2014)**
 - Ashley Madison (2015)
 - Others
- Why do we need security?
- Scope of the course

■ Hack into Sony Pictures

- Perpetrators: The Guardians Of Peace (GOP)
- Result
 - ▶ Crippled network, theft of valuable information, theft of unreleased materials
 - ✓ Employee data, network information, security information, payment information, ...
- Vague allegations North Korea is responsible in retribution for the imminent release of an upcoming movie titled “The Interview”.
 - ▶ Alternatives include former employees, China, Russia, ...

■ Clearly political motivated

- Repeated releases, significant media attention

Sony Pictures Hacked And Blackmailed Forbes

2014-11-24

<http://www.forbes.com/sites/davelewis/2014/11/24/sony-pictures-hacked-and-blackmailed/>

**US tapped N Korean networks years ago, providing proof of Sony hack –
theguardian**

2015-01-19

<http://www.theguardian.com/film/2015/jan/19/us-tapped-n-korean-networks-years-ago-providing-proof-of-sony-hack-reports>

**FBI doubts North Korea link to Sony Pictures hack
theguardian**

2014-12-10

<http://www.theguardian.com/technology/2014/dec/10/fbi-doubts-north-korea-link-sony-pictures-hack>

- Security in the media
- Recent major incidents
 - Heartbleed (2014)
 - Sony Pictures Entertainment hack (2014)
 - **Ashley Madison (2015)**
 - Others
- Why do we need security?
- Scope of the course

■ What?

- A commercial website for enabling extramarital affairs

■ Perpetrators: "The Impact Team"

- Stolen items
 - ▶ Personal information from users, e-mails and corporate data
- Demands
 - ▶ Shut down of the site

■ Motivation

- "ethical" hacking... ?

■ Results

- Insights in falsified profiles
- Broken marriages, several suicides
- Damage up to millions?

50

Passwords on the live site were hashed using the bcrypt algorithm. A security analyst using the Hashcat password recovery tool with a dictionary based on the RockYou passwords found that among the 4,000 passwords that were the easiest to crack, "123456" and "password" were the most commonly used passwords on the live website. An analysis of old passwords used on an archived version showed that "123456" and "password" were the most common. Due to a coding error where passwords were hashed with both bcrypt and md5 11 million passwords were eventually cracked.

- Security in the media
- Recent major incidents
 - Heartbleed (2014)
 - Sony Pictures Entertainment hack (2014)
 - Ashley Madison (2015)
 - Others
- Why do we need security?
- Scope of the course

- Operation Aurora (2010)
- Australian cyberattacks (2010)
- Operation Payback (2010)
- HBGary Federal (2011)
- DigiNotar (2011)
- Operation Tunisia (2011)
- 2011 PlayStation Network outage (2011)
- Operation AntiSec (2011)
- Stratfor email leak (2012–13)
- LinkedIn hack (2012)
- South Korea cyberattack (2013)
- Snapchat hack (2013)
- Operation Tovar (2014)
- 2014 celebrity photo hack (2014)
- Heartbleed (2014)
- Shellshock (2014)
- POODLE (2014)
- Sony Pictures Entertainment hack (2014)
- Office of Personnel Management data breach (2015)
- Hacking Team (2015)
- Ashley Madison (2015)
- Stagefright (2015)
-
- To be continued....

52

All off the above incidents provide for insightful and entertaining reading material, in which both the technical aspects should be considered as well as the motivation behind the incidents. Most of them show how majorly technology and politics are entwined in current days.

- Security in the media
- Recent major incidents
- Why do we need security?
- Scope of the course

■ Why Information Security?

- Counterpart of securing material objects

- ▶ Material object have some value

- ✓ Value can often easily be determined (except for affective value)

- ▶ Can be stolen or damaged

- ✓ Causes material damage (replacement of the object, interruption of business process, etc.)

- ✓ Most damage is repairable (replacement or repair)

■ Why Information Security?

- **Counterpart of securing material objects**

- ▶ Protecting from damage

- ▶ Protecting from theft

- ✓ Locks, safes, etc.

- ▶ Cost for security/protection takes into account:

- ✓ Value of the object

- ✓ Risk of theft/damage

■ Why Information Security?

- **Value of information**

- ▶ Sometimes hard to assess
- ▶ Best estimated by damage caused
 - ✓ When information security is breached
 - ✓ But even this can be hard:
 - » what is the value of someone's privacy?

- **Threats against information**

- ▶ **Loss** of information
- ▶ **Forged** information
- ▶ **Unauthorised release** of information
- ▶ **Repudiation** of information
- ▶ etc.

■ Why Information Security?

● Consequences of security breaches

- ▶ Can't always be undone
 - ✓ Lost information
 - ✓ Unauthorised release of information

● Measures

- ▶ Information security:
 - ✓ encryption, digital signature, etc.
- ▶ Carry some cost
 - ✓ Implementation, lost ease-of-use, etc.
- ▶ Here too, dependent on...
 - ✓ ...risk of security breach
 - ✓ ...potential damage in case of breach

■ Why Information Security?

- **Value of information systems**

- ▶ Also hard to assess
- ▶ Systems are meant to enable some service
 - ✓ Damage when service is unavailable or unreliable

- **Threats against information systems**

- ▶ **Unavailability**/disruption of service
- ▶ **Unauthorised access** to service
- ▶ Threats against exchanged information
- ▶ etc.

■ Why Information Security?

- **Security measures for information systems**

- ▶ **Information security:** encryption, virus scanners, firewalls, etc.
- ▶ **Also carry some cost**
 - ✓ installation, maintenance, computation time, lost ease-of-use, etc.
- ▶ **Here too, dependent on...**
 - ✓ ...risk of security breach
 - ✓ ...potential damage in case of breach

■ Why Information Security?

- Note

- ▶ The risk of threats against information security is MUCH greater than the risk of threats against material objects
 - ✓ Much more diverse attacks because of available computation power and almost ubiquitous network connectivity

- Security in the media
- Recent major incidents
- Why do we need security?
- Scope of the course

- Chapter 1: Introduction
- Chapter 2: Basic concepts
- Chapter 3: Network and communication security
- Chapter 4: Software and systems security
- Chapter 5: Information security
- Chapter 6: Encryption algorithms
- Chapter 7: Legal aspects
- Chapter 8: Security Threads

Questions?



Besides the lecturers' own material, many third party, often copyrighted, material is reused within this lecture (e.g. in the notes) under the 'fair use' approach, for sake of educational purpose only, and very limited edition. As a consequence, the current slide set presentation usage is restricted, and is falling under usual copyrights usage.

At the end of every lecture, appropriate references to used materials are included.

- This work contains content adapted from, amongst other, the following sources (in no particular order)
 - William Stallings, “**Cryptography and Network Security, principles and practices**”, 6th (international) edition, Prentice Hall, 2010;
 - Matt Bishop, “**Computer Security: Art and Science**”, Addison Wesley, Pearson Education, 2003, ISBN-13: 978-0-201-44099-7
 - Lecture slides: “Informatiebeveiliging”, Universiteit Gent, Eric Laermans & Thom Dhaene
 - Wikipedia (additional note page descriptions)
 - Various (web) comics and news sites (references given throughout the slides)
 - <https://crypto.stackexchange.com/>



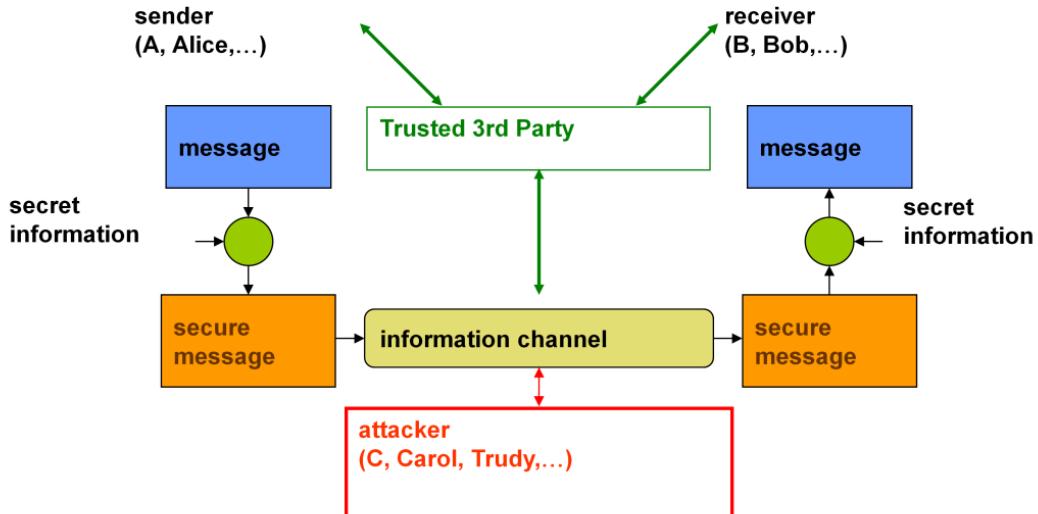
Network and Computer Security

Chapter 2 – Basic concepts

Prof. dr. ir. Eli De Poorter

- A security model
- Security goals
 - Confidentiality
 - Authentication
 - Access control / authorization
 - Data integrity
 - Non-repudiation
 - Availability
- Security threats
- Security mechanisms
 - Encryption
 - ▶ Symmetric encryption
 - ▶ Asymmetric encryption
 - Hash functions
 - Message authentication codes

- A security model
- Security goals
 - Confidentiality
 - Authentication
 - Access control / authorization
 - Data integrity
 - Non-repudiation
 - Availability
- Security threats
- Security mechanisms
 - Encryption
 - ▶ Symmetric encryption
 - ▶ Asymmetric encryption
 - Hash functions
 - Message authentication codes



4

As was already stated by Auguste Kerckhoffs in the 19th century: a cryptosystem should be secure even if everything about the system, except the key, is public knowledge (Kerckhoff's principle). This means that typically the security transformation is publicly known (and is often standardized), and only the secret information (key) used in the security transformation to convert the message into a secure message is kept secret. The full method for secure information exchanges between A(lice) and B(ob) may require a prior exchange of several messages, including both data and control messages.

Alice and Bob are two commonly used placeholder name to make security exchanges easier to follow. The names are used for convenience; for example, "Alice sends a message to Bob encrypted with his public key" is easier to follow than "Party A sends a message to Party B encrypted by Party B's public key." These names were originally used by Ron Rivest in the 1978 Communications of the ACM article presenting the RSA cryptosystem, and in A Method for Obtaining Digital Signatures and Public-Key Cryptosystems published April 4, 1977, revised September 1, 1977, as technical Memo LCS/TM82.

Alice and Bob are archetypes in cryptography, Eve is also common. Names further down the alphabet are less common. Other (less frequently used) names include:

- Carol, Chuck, Carlos or Charlie, as a third participant in communications, sometimes with malicious intent.
- Craig, the password cracker (usually encountered in situations with stored hashed/salted passwords).
- Dan or Dave, a fourth participant.
- Eve, an eavesdropper, is usually a passive attacker. While she can listen in on messages between Alice and Bob, she cannot modify them.
- Trudy, a malicious attacker (an intruder.). Unlike the passive Eve, this one is the active man-in-the-middle attacker who can modify messages, substitute his/her own messages, replay old messages, and so on. The difficulty of securing a system against Trudy is much greater than against Eve.
- Walter, a warden, may be needed to guard Alice and Bob in some respect, depending on the protocol being discussed.
- Wendy, a whistleblower, is an insider with privileged access who may be in a position to divulge the information.

- A security model
- Security goals
 - Confidentiality
 - Authentication
 - Access control / authorization
 - Data integrity
 - Non-repudiation
 - Availability
- Security threats
- Security mechanisms
 - Encryption
 - ▶ Symmetric encryption
 - ▶ Asymmetric encryption
 - Hash functions
 - Message authentication codes

■ Data confidentiality

- **Data can only be read by those who are allowed to read these data**

- **Applications:**

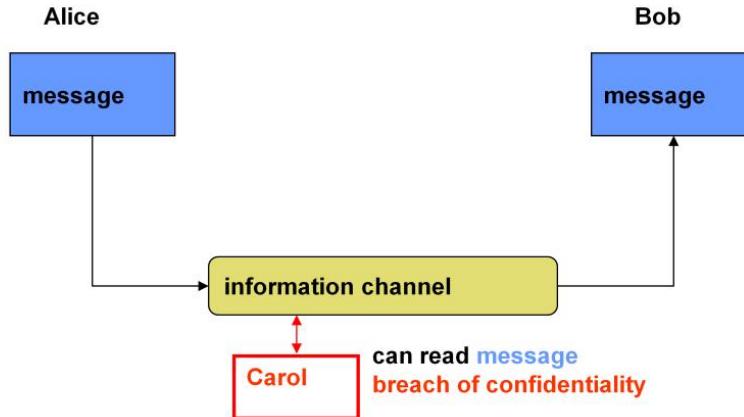
- ▶ Communicating confidential data between branches of a corporation
- ▶ Passwords
- ▶ Storage of health data
- ▶ etc.

6

Confidentiality is translated as “vertrouwelijkheid” in Dutch.

■ Data confidentiality

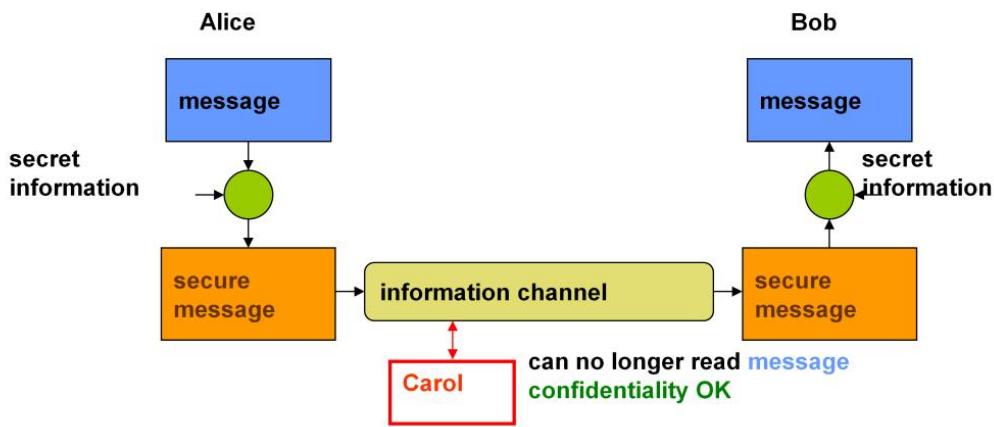
- “Non-electronic” world equivalent
 - ▶ Sensitive data for states, corporations, etc.
 - ▶ Protection of correspondence
 - ▶ Protection of phone communications
 - ▶ etc.
- Oldest security service?
 - ▶ Caesar cipher (Ancient Rome)
 - ▶ Enigma code (Germany, WW II)
 - ▶ etc.



Passive attack by Carol:
eavesdropping upon information channel

8

E.g. Alice submits a tender for a procurement by Bob. Neither party uses any special security mechanism (e.g. simple communication using e-mail, ftp, etc.). Carol can read the transmitted message. Carol can then submit her own proposal, adapted using her knowledge of Alice's offer.



9

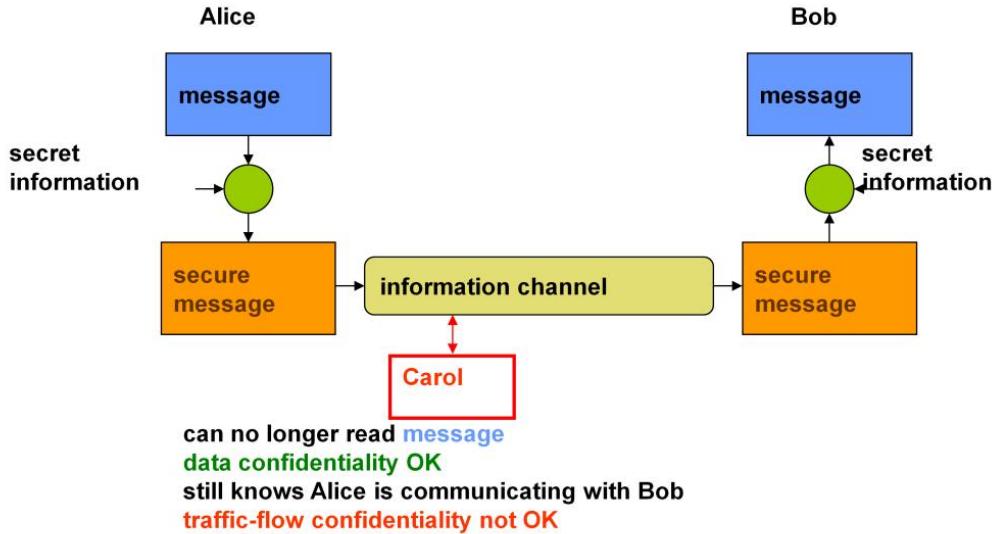
To make sure the message no longer is readable for Carol, a security transformation (encryption) is introduced, allowing only those (i.c. Alice en Bob) who have access to secret information (secret key) to recover the original message. Carol can still intercept the secure message, but she won't be able to derive the original message from this.

■ Traffic-flow confidentiality

- Keeping secret who's communicating with whom
- Applications:
 - ▶ Web surfer privacy protection
- Much harder to achieve than data confidentiality
 - ▶ So-called Privacy Enhancing Techniques (PETs)

10

As examples of existing privacy enhancing technologies communication anonymizers exist that hide the real online identity (email address, IP address, etc.) of a user and replacing it with a non-traceable identity (disposable / one-time email address, random IP address of hosts participating in an anonymising network, pseudonym, etc.). A well-known PET is the onion routing protocol used by the Tor anonymous network.



11

Passive attack by Carol: tapping the information channel between both parties A(lice) and B(ob).

With the solution for data confidentiality Carol can't read the transmitted message any longer, but she can still deduce that Alice is communicating with Bob (e.g. from IP headers, or by analyzing the communication flow outgoing from Alice and incoming at Bob: "traffic-analysis").

■ Privacy

- Often confused with confidentiality
 - ▶ Not every confidentiality requirement involves privacy
 - ✓ E.g. intellectual property in a business requires confidentiality, no privacy
- Related to private life
 - ▶ The right to choose what you divulge about yourself
 - ▶ Culture dependent
 - ▶ Fundamental right, legally protected since long
 - ✓ Foundation for secrecy of correspondence
 - ▶ Just like any fundamental right, this right isn't unlimited

- A security model
- Security goals
 - Confidentiality
 - **Authentication**
 - Access control / authorization
 - Data integrity
 - Non-repudiation
 - Availability
- Security threats
- Security mechanisms
 - Encryption
 - ▶ Symmetric encryption
 - ▶ Asymmetric encryption
 - Hash functions
 - Message authentication codes

■ Authentication

- Related to **identification**
- Guaranteeing the authenticity of a communication based on:
 - ▶ Entity authentication
 - ▶ Attribute authentication
 - ▶ Data-origin authentication

■ Entity authentication

- Identity

- ▶ Distinguished based on collection of data (attributes or characteristics)
 - » E.g. ID-number, mail address, ...
- ▶ Each entity has a unique identity

■ Identification

- Authentication of an entity's identity

- ▶ Often used for entity authentication...
- ▶ ...but a stronger requirement than simple entity authentication

- E.g. Logging in via biometrical means

15

According to the Merriam-Webster dictionary, an entity is “something that has separate and distinct existence and objective or conceptual reality”. An entity typically has a number of attributes (or characteristics) that, taken together, form a unique combination. Entity authentication requires a validation to check if communicating parties are who they claim to be. In practical implementations, entity authentication is often based on authentication of some attributes of these entities. An example is a persons first name, last name, date and hour of birth and place of birth that together uniquely identify a person. Alternatively, the identity number of a passport is also an attribute that uniquely identifies a person.

Identification is used when an identity needs to be uniquely represented and the identity needs to be validated. An example where entity authentication doesn't really require the entity's identification is the registration with many websites. During registration the user generates a login and password, and provides some data (name, address, email address, etc.) to the website. Typically the only important data is the e-mail address (to which possibly the password will be sent), while all other data absolutely don't need to be authentic. The next time the user logs in to this website, he'll be authenticated via the login and password (he is then deemed the authentic owner of this couple). But one can't assert his identity is authentic, since it wasn't really verified at any time.

■ Attribute authentication

- **attribute**

- ▶ Characteristic of an entity
- ▶ Possible attributes
 - ✓ Identity
 - ✓ Function
 - ✓ etc.

- **Communicating parties exhibit the characteristics they claim to have**

- ▶ An entity is often authenticated through the authentication of some of its attributes

16

An already sufficiently authenticated entity (e.g. using an electronic identity card, which adequately captures the identity of a communicating entity) may need to authenticate additional attributes (e.g. the fact that he is a doctor to access certain medical information).

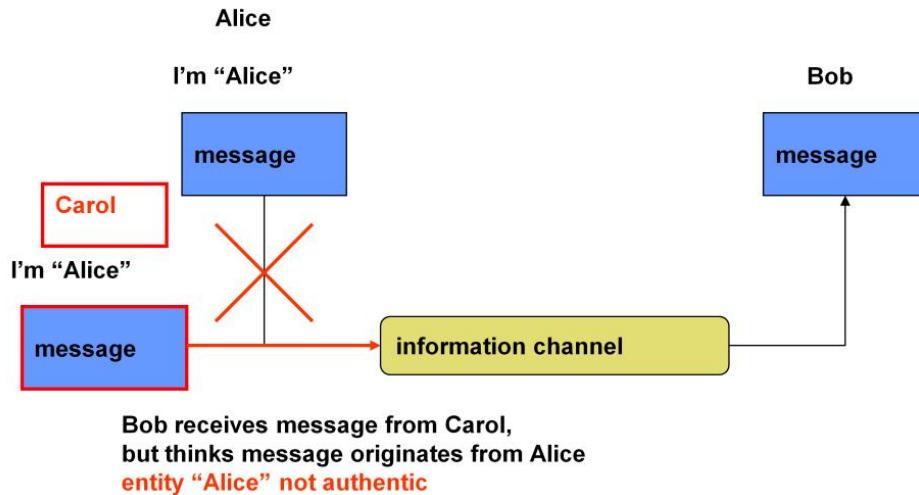
■ Data-origin authentication

- The data indeed originates from the specified source
- Important to evaluate whether data are reliable
 - ▶ Based on reliability of their origin
 - ▶ Part of **data integrity** (cf. further on)
- Difference with entity authentication
 - ▶ No interaction with data source
 - ▶ Not all solutions for entity authentication will be operative

■ Authentication

● Equivalent in “non-electronic” world

- ▶ Author of a letter is who he claims to be
- ▶ Person on the other side of the phone is who he claims to be
- ▶ Policeman ringing at the door indeed is a policeman
- ▶ Mobile phone battery is genuine
- ▶ etc.

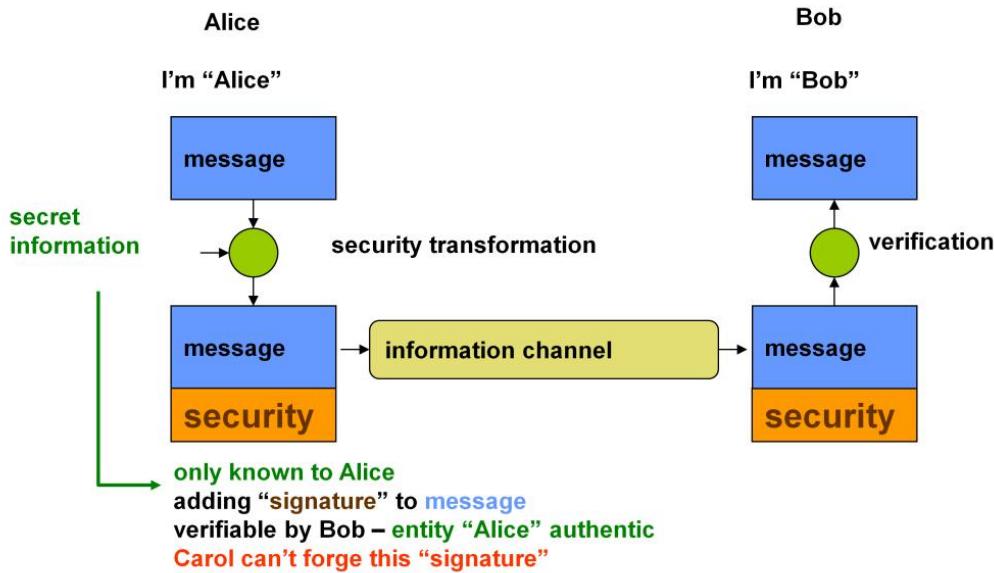


19

Active attack by Carol on the communication between Alice and Bob, e.g. by hijacking the communication, or more simply by sending a forged message.

Example 1) Alice communicates with website of Bank Bob. Bob needs to know whether it is indeed Alice who logs in, and not Carol masquerading as Alice in order to gain access to Alice's accounts at Bank Bob. (**authenticity of the entity** Alice; conversely, Alice may also want to be sure she's communicating with the website of Bank Bob and not with a fake website set up by Carol).

Example 2) Bob has received a message from software vendor Alice (e.g. by email) with a major security patch for the software in attachment. Before Bob installs the patch, he'd like to be certain it really originates from Alice... and isn't some piece of malware coming from Carol. (**authenticity of the origin** of the data received by Bob).



20

Make sure Carol can't generate a message in which she claims to be someone else anymore. Add some "signature" to Alice's message. This "signature" binds the entity Alice to the contents of the message, so it must:

- be linked to the message (another message has a different "signature")
- be based on secret information only Alice possesses
- be verifiable, even by those who don't know this secret information

The digital signature is based on this principle, and also guarantees the authenticity of the origin of the data.

Note that a potential threat still lurks in this diagram. If Carol eavesdrop upon the information channel she still can intercept a secure message and then forward it again to Bob and in this way still impersonate Alice (replay).

An alternative solution is for Alice to use an (encrypted) password as secret information, that Bob will recognise as Alice's. The danger is that Carol still can eavesdrop on the forwarded password and then use it again (sometimes even if it is encrypted). A remedy against this threat is to ensure the communication between Alice and Bob can also be confidential. A possible implementation could be: sending the password over a TLS/SSL connection (see later for TLS/SSL).

- A security model
- Security goals
 - Confidentiality
 - Authentication
 - **Access control / authorization**
 - Data integrity
 - Non-repudiation
 - Availability
- Security threats
- Security mechanisms
 - Encryption
 - ▶ Symmetric encryption
 - ▶ Asymmetric encryption
 - Hash functions
 - Message authentication codes

■ Access control and authorisation

- **Determines which user may access which resources (data, computation time, etc.)**

- **Requires authentication of the entity requesting access to these resources**
 - ▶ System determines to what extent entity may access those resources
 - ▶ Access rights may depend on **entity** itself or its **attributes**

■ Access control / authorisation

● Equivalent in “non-electronic” world

- ▶ Only people with a valid entrance ticket may attend a concert
- ▶ Special member card required for some shops
- ▶ Only attending physician has access to patient's medical information
- ▶ etc.

■ Access control and authorisation

● illustration 1: access control in OS

► Authentication through login and password

- ✓ Determines which user is at the computer

► Access control determined for this user (entity)

- ✓ Full access (read, write, delete, etc.) to own files
- ✓ Limited access (e.g. read) to some other files (e.g. some system files)
- ✓ No access to other files (e.g. files belonging to other users)

► Access rights different from user to user

■ Access control and authorisation

- **illustration 2: access control to medical database**
 - ▶ Different rights for different types of users
 - ✓ E.g. physicians, nurses, patients, etc.
 - ▶ Requires authentication based on specific **attributes**
 - ▶ Access rights depend on attributes of the user
 - ▶ Access rights different from user type to user type (**roles**)
 - ✓ Role based access control

- A security model
- Security goals
 - Confidentiality
 - Authentication
 - Access control / authorization
 - Data integrity
 - Non-repudiation
 - Availability
- Security threats
- Security mechanisms
 - Encryption
 - ▶ Symmetric encryption
 - ▶ Asymmetric encryption
 - Hash functions
 - Message authentication codes

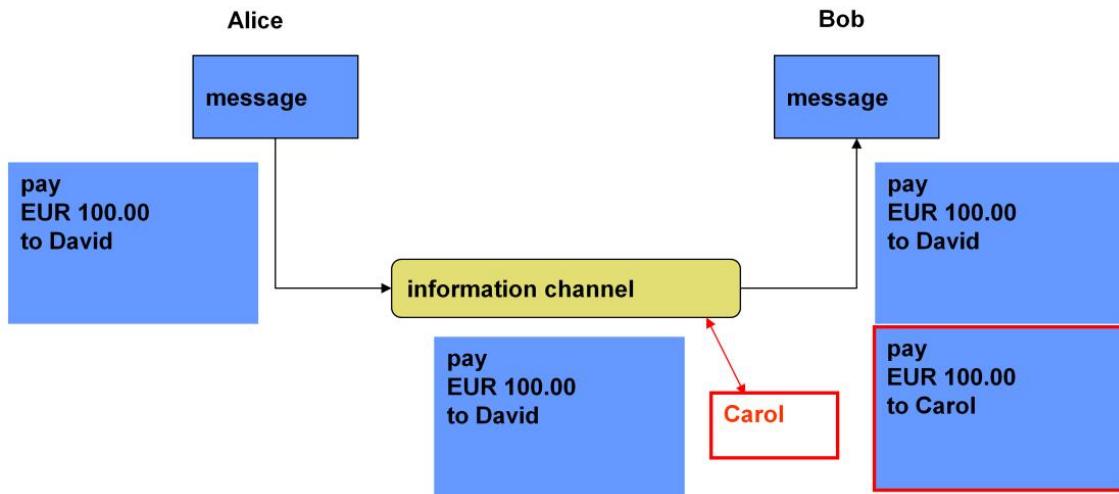
■ Data integrity

- **Guarantee that sent data and received data are identical**
 - ▶ No tampering with data en route
 - ▶ Nothing was added
 - ▶ Nothing was deleted
 - ▶ Nothing was modified
 - ▶ Nothing was replayed
- **Stronger requirement than data origin authentication**

■ Data integrity

● Equivalent in “non-electronic” world

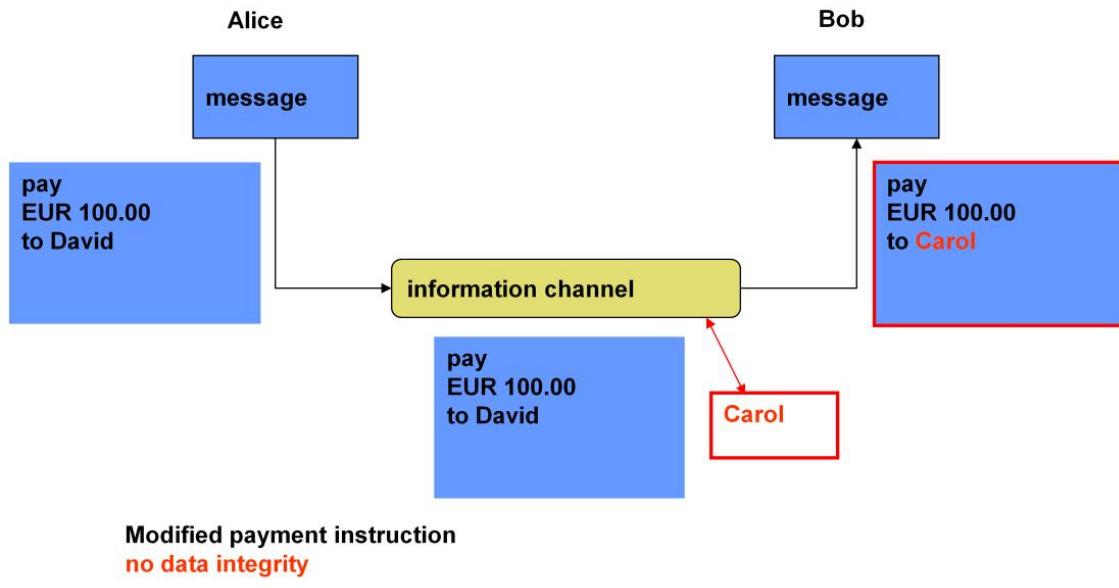
- ▶ No text was added after the signature of a contract
- ▶ The invoice date is correct
- ▶ No “creative” accounting
- ▶ No information was withheld
- ▶ Vote cast was effectively registered
- ▶ etc.



**Additional payment instruction is received
no data integrity**

29

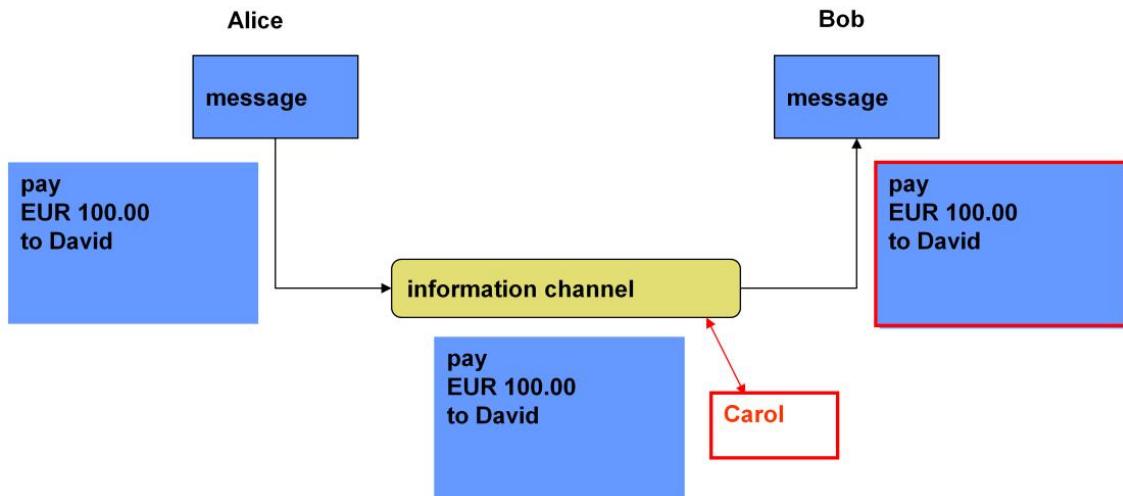
E.g. Alice sends message to bank Bob instructing to pay EUR 100.00 to David. Alice would like that this exact amount is paid to D(avid)...and that no other payments are stealthily executed.
Active attack by Carol: injection of an additional message.



30

E.g. Alice sends message to bank Bob instructing to pay EUR 100.00 to David. Alice would like that this exact amount is paid to D(avid)...and not to some other person.

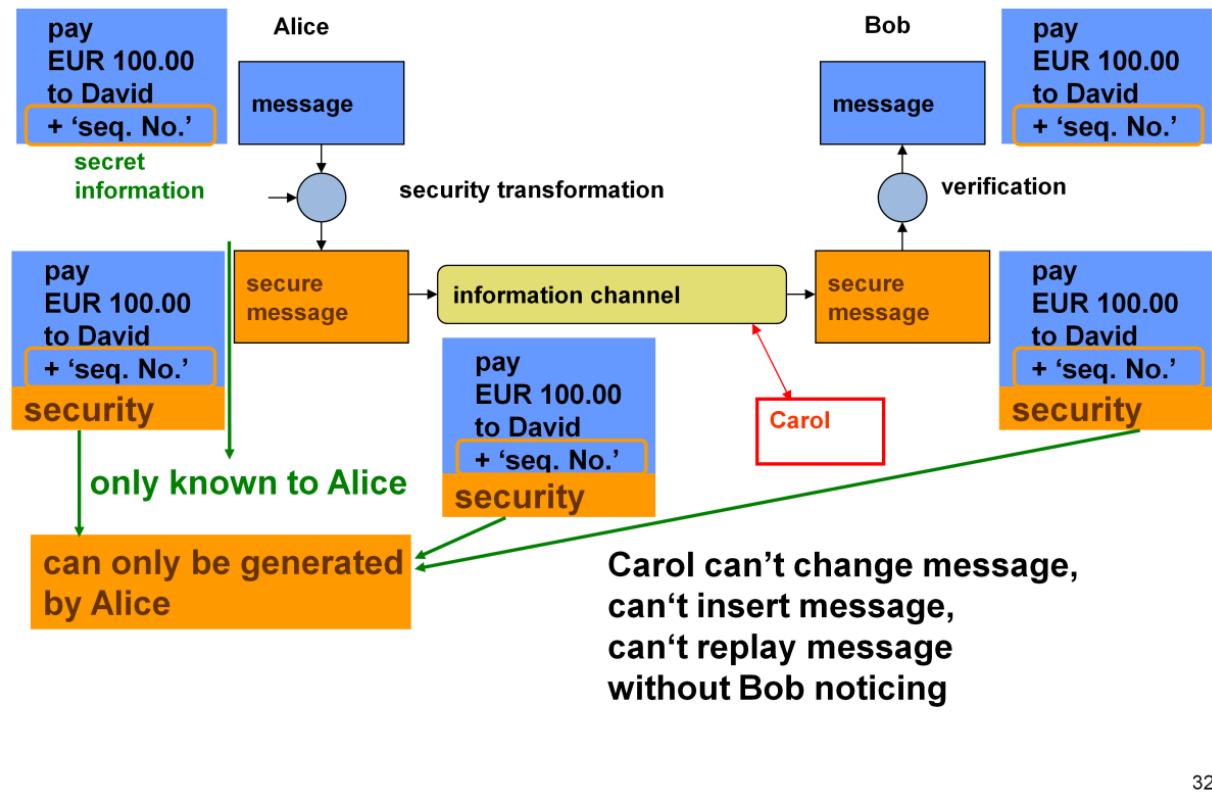
Active attack by Carol: message modification.



Payment instruction is executed more than once
no data integrity

31

E.g. Alice sends message to bank Bob instructing to pay EUR 100.00 to David. Alice would like that this exact amount is paid to D(avid)...and doesn't want a double execution of the payment.
Active attack by Carol: fraudulent reuse of data ("replay").



32

Here too a digital signature can be a solution. This signature makes it impossible for Carol surreptitiously to modify a message or to insert a message that hasn't been created by Alice.

The sequence number added to the communication aims at thwarting a “replay”. In practice, this won't be a simple serial number (e.g. 0,1,2,etc.), but it will satisfy some security requirements (to avoid the replay of an old sequence number in a later session). A possible choice may be the use of a (sufficiently large) “nonce”, which is incremented for each subsequent message. Another possibility is to use a time value instead of a regular sequence number (but beware of desynchronised clocks).

Some use modes of symmetric encryption may also thwart replay attacks (see later).

Observe that this schema doesn't guarantee full data integrity, as Carol could still cause the message **not** to reach Bob. This aspect of data integrity can only be achieved with a bi-directional communication between Alice and Bob (receipt acknowledgement).

- A security model
- Security goals
 - Confidentiality
 - Authentication
 - Access control / authorization
 - Data integrity
 - **Non-repudiation**
 - Availability
- Security threats
- Security mechanisms
 - Encryption
 - ▶ Symmetric encryption
 - ▶ Asymmetric encryption
 - Hash functions
 - Message authentication codes

■ Non-repudiation

- **For sender**

- ▶ Sender can't deny having sent the message
- ▶ Important for receiver

- **For receiver**

- ▶ Receiver can't deny having received the message
- ▶ Important for sender

■ Equivalent in “non-electronic” world

- **Being able to prove an order has been placed**
- **Being able to prove an invoice has been paid**
- **etc.**

34

Repudiation is translated in Dutch as “verwerping”, “ontkenning” or “verloochening”.

In order to achieve non-repudiation (for the sender) for a transaction, the receiver should be able to prove that the sender did send this message. This implies measures have been taken to ensure an intruder couldn't have sent the message, i.e.:

- Sender authentication
- Data integrity of the communication between sender and receiver
- The receiver will have to keep the “signature” of the message

The schema shown for data integrity would achieve these goals.

Non-repudiation for the receiver is harder to achieve. It requires a reaction from the receiver to the sender. The non-repudiation of this reaction guarantees the desired functionality.

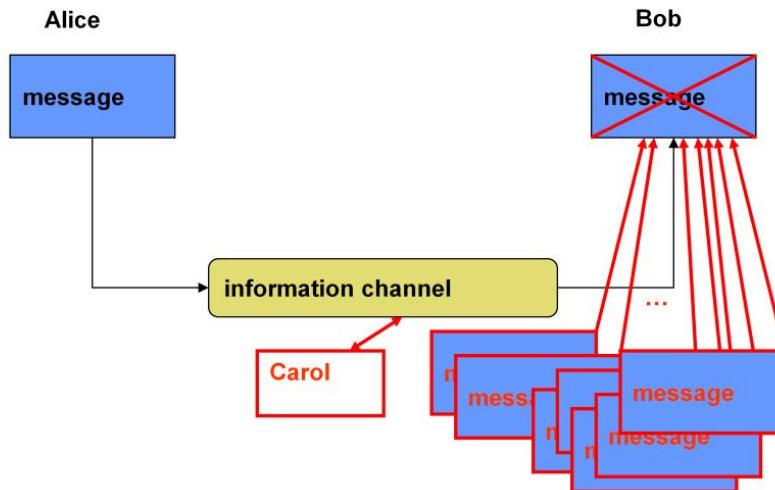
- A security model
- Security goals
 - Confidentiality
 - Authentication
 - Access control / authorization
 - Data integrity
 - Non-repudiation
 - Availability
- Security threats
- Security mechanisms
 - Encryption
 - ▶ Symmetric encryption
 - ▶ Asymmetric encryption
 - Hash functions
 - Message authentication codes

■ Meaning

- **System/service is accessible and usable for authorised users**
- **Within the limitations of the system**
 - ▶ System can be designed with a limited capacity
 - ✓ No attack against availability
 - ✓ Only poor design
 - ✓ However more vulnerable to attacks

■ Equivalent in “non-electronic” world

- **Store accessible during opening hours**
 - ▶ Could be prevented by protesters
- **Polling stations allow users to vote on election day**
 - ▶ Could be hindered by political unrest
 - ▶ NO attack: insufficient number of polling stations (e.g. Ohio 2004)
 - ✓ Just poor design
- **etc.**



Bob is swamped by the torrent of messages from Carol

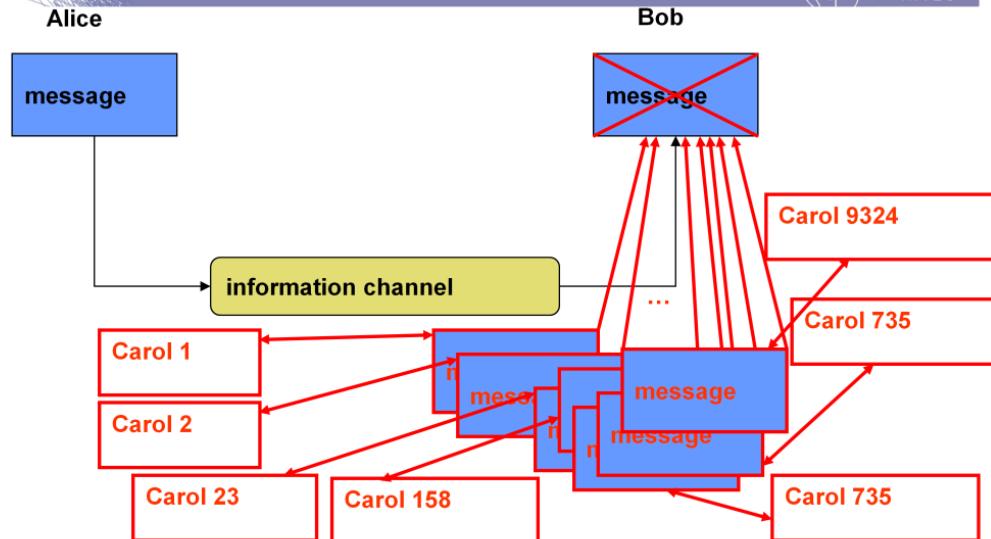
Bob's service no longer is accessible

DoS: denial-of-service

37

E.g. Alice requests information from website Bob. Under normal circumstances Bob should be able to receive, process, and answer Alice's message.

Carol executes an active attack: denial-of-service. This requires true broadband access (better than DSL or cable; university networks sometimes used to be coveted launch pads for this kind of attacks). Authentication of communicating parties makes such an attack a lot harder (although the initial phase of the authentication protocol could still be attacked).



Bob is swamped by the torrent of messages from multiple (and numerous) senders

Bob's service no longer is accessible

DDoS: distributed denial-of-service

38

E.g. Alice requests information from website Bob. Under normal circumstances Bob should be able to receive, process, and answer Alice's message.

"Carols" execute an active attack: distributed denial-of-service. A large number of attackers Carol 1, Carol 2, etc. overwhelm Bob's website under a torrent of messages. These messages may come from:

- A large group of participants
- A large group of computers contaminated by malware, where unsuspecting users are deployed for the attack (e.g. MyDoom.A worm, which performed a successful DoS attack against the SCO website)

Broadband access isn't required for this kind of attack (regular DSL or cable will be largely sufficient). This attack is much harder to fend off than regular DoS.

Practical solutions against DDoS typically involve non-cryptographic techniques (firewall, IDS, etc.) that analyse incoming communications so that anomalies can be detected and dealt with (filtering out the anomalous traffic). If needed, it may be desirable to operate a temporary graceful shutdown of the service (which is still to be preferred over a system crash, possibly involving some data loss).

- A security model
- Security goals
 - Confidentiality
 - Authentication
 - Access control / authorization
 - Data integrity
 - Non-repudiation
 - Availability
- Security threats
- Security mechanisms
 - Encryption
 - ▶ Symmetric encryption
 - ▶ Asymmetric encryption
 - Hash functions
 - Message authentication codes

■ Passive attacks

- Eavesdropping
- Traffic analysis

■ Active attacks

- Message insertion / modification
- Impersonation / masquerade
- Replay
- Denial-of-Service (DoS)
- Hijacking
 - ▶ taking over existing connection, where attacker replaces sender or receiver

40

Passive attacks offer a more limited potential to the attacker, but they are significantly harder to detect, as they don't modify the transmitted data. This is certainly true if part of the communication channel is totally open (e.g. a wireless connection). Active attacks do modify the transmitted data or the data stream, and can thus be detected more easily. However, they give the attacker a lot more options.

■ Possible attacks

- **Brute force**

- ▶ Trying out all possible keys until correct key is found

- **Cryptanalysis**

- ▶ More subtle attacks using knowledge about structure of algorithm, and possibly also additional knowledge of pairs (plaintext message, secure message), in order to recover plaintext message or key itself, or to forge secure message

- **Side-channel attacks**

- ▶ Even more subtle attacks using the physical properties (power, computation time, electromagnetic radiation, etc.) or fault injection in order to recover the plaintext or the key

■ Example side channel attacks



Here's How iPhone Thermal Cameras Can Be Used to Steal Your Pin Codes, Petapixel, August 29, 2014



Researchers crack the world's toughest encryption by listening to the tiny sounds made by your computer's CPU,
<http://www.extremetech.com>, December 18, 2013

Timing information, power consumption, electromagnetic leaks or even sound can provide an extra source of information, which can be exploited to break the system.

<http://petapixel.com/2014/08/29/heres-iphone-thermal-cameras-can-used-steal-pin-codes/>

<http://www.extremetech.com/extreme/173108-researchers-crack-the-worlds-toughest-encryption-by-listening-to-the-tiny-sounds-made-by-your-computers-cpu>

■ Categories of attacks

- “**Ciphertext only**”
 - ▶ Only secure message (ciphertext) is known to attacker
- “**Known plaintext**”
 - ▶ One or more pairs (plaintext, ciphertext) obtained with a single key are known to attacker
- “**Chosen plaintext**”
 - ▶ Requires one or more pairs (plaintext, ciphertext) obtained with a single key, where plaintext was chosen by attacker
- “**Chosen ciphertext**”
 - ▶ Requires one or more pairs (plaintext, ciphertext) obtained with a single key, where ciphertext was chosen by attacker
 - ✓ Corresponding plaintext may be “garbage”
- “**Chosen text**”
 - ▶ A combination of both “chosen plaintext” and “chosen ciphertext”

43

The term ciphertext (or secure message) typically refers to the encrypted message, whereas the term plaintext is used for the original text. The same terminology is used for text or other data structures, since any digital data source can be represented as plaintext. “Ciphertext only” attacks can be mounted using information that is readily available to the attacker (it only requires eavesdropping upon the information channel). It is however much harder to perform successful cryptanalysis with a “ciphertext only” attack. Conversely, it may be very hard to mount a “chosen text” attack as finding the desired (plaintext, ciphertext) pairs is typically hard. However, it offers more opportunities for cryptanalysis.

Some algorithms are very safe against “ciphertext only” attacks, but won’t withstand other types of attacks. This may be acceptable if the algorithm is used correctly. Other algorithms will even withstand a “choose text” attack. This is something to bear in mind when considering security mechanisms.

■ Desired degree of security?

- **Unconditionally secure**

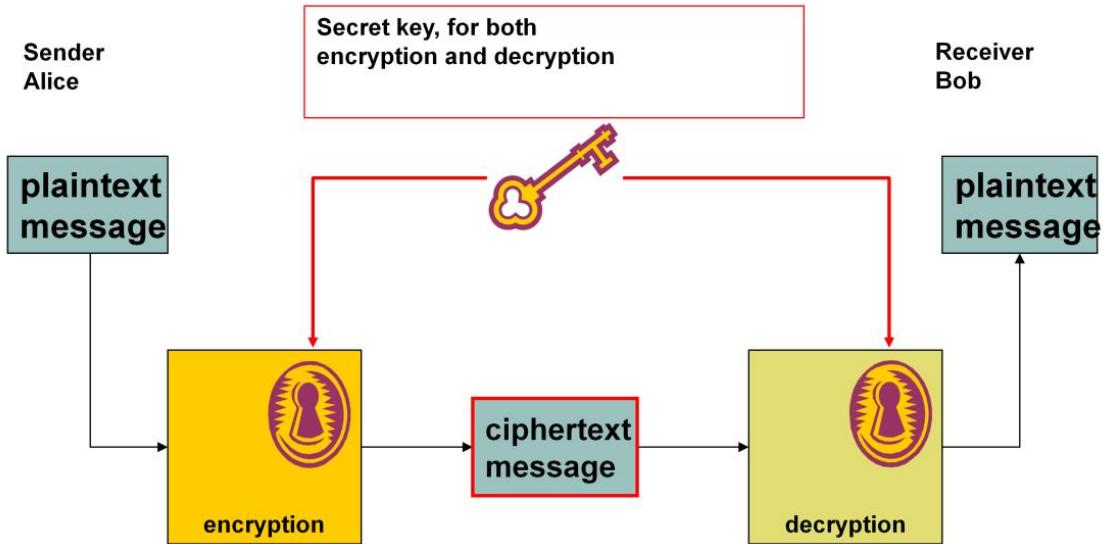
- ▶ Impossible to invert security transformation without knowing the secret information
- ▶ **No practical security mechanism achieve this**

- **Computationally secure**

- ▶ The cost of breaking the encryption is larger than the value of the information
- ▶ The time required for breaking the encryption is longer than the useful lifetime of the information
- ▶ **All practical algorithms discussed in this course are in this category**

- A security model
- Security goals
 - Confidentiality
 - Authentication
 - Access control / authorization
 - Data integrity
 - Non-repudiation
 - Availability
- Security threats
- Security mechanisms
 - Encryption
 - ▶ Symmetric encryption
 - ▶ Asymmetric encryption
 - Hash functions
 - Message authentication codes

■ Basic operation



■ Principle:

- **Transforms plaintext message into encrypted message**
 - ▶ Transformation using secret key
- **Inverse operation: decryption**
 - ▶ Transforming the encrypted message into a plaintext message
 - ▶ Using the same secret key
 - ▶ Practically infeasible to decrypt without knowing the secret key

■ Secret key size

- ▶ 40 bits: weak security
- ▶ 128 or more bits: (very) strong security

47

Symmetric encryption is the most traditional form of encryption and is also referred to as "conventional encryption".

■ Secure storage of sensitive information

- Only encrypted files are stored
 - ▶ Only owner of secret key can decipher encrypted files
 - ▶ Lost laptop doesn't imply breach of confidentiality
 - ✓ However loss of key implies loss of data
- Only secret key needs to be protected
 - ▶ E.g. Storage on separate support

■ Secure transmission of sensitive information

- Eavesdropping upon communication channel doesn't yield useful information
- Sender and receiver must share secret key
 - ▶ Sender and receiver need to trust each other
 - ▶ Requires different secret key for each pair of communicating users

■ Authentication of communicating entity A(lice)

- Only A(lice) can use this specific key in a communication with B(ob)

■ Drawbacks

- Server B(ob) needs to know the secret key of each party that wants to communicate with B(ob)
 - ▶ Unwieldy + vulnerable
- No 100% authenticity of data-origin for message
 - ▶ B(ob) too could have generated this message

■ Original

- replace each letter by letter 3 places shifted in the alphabet

- transformation:

a b c d e f g h i j k l m n o p q r s t u v w x y z
D E F G H I J K L M N O P Q R S T U V W X Y Z A B C

■ Mathematically

- Give each letter a number

a b c d e f g h i j k l m n o p q r s t u v w x y z
0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25

- Caesar cipher:

$$c = \text{ciphertext} = E(p) = (p + k) \bmod (26)$$

$$p = \text{plaintext} = D(c) = (c - k) \bmod (26)$$

■ Example:

meet me after the toga party

PHHW PH DIWHU WKH WRJD SDUWB

Substitution ciphers are encryption methods where letters of plaintext are replaced by other letters or by numbers or symbols. Whilst the early Greeks described several substitution ciphers, the first attested use in military affairs of one was by Julius Caesar, described by him in *Gallic Wars*. We call any cipher using a simple letter shift a **caesar cipher**, not just those with shift 3 as originally used.

This mathematical description uses **modulo (clock) arithmetic**. Here, when you reach Z you go back to A and start again. Mod 26 implies that when you reach 26, you use 0 instead (ie the letter after Z, or 25 + 1 goes to A or 0).

Example: howdy (7,14,22,3,24) encrypted using key f (ie a shift of 5) is MTBID

■ Disadvantages

- Only 26 possible ciphers

■ Brute force search

- Simply try each in turn
- eg. break ciphertext "GCUA VQ DTGCM"

With a Caesar cipher, there are only 26 possible keys, of which only 25 are of any use, since mapping A to A etc doesn't really obscure the message! An attacker can try each of the keys (shifts) in turn, until can recognise the original message.

Note: as mentioned before, you need to be able to **recognise** when have an original message (ie is it English or whatever). Usually easy for humans, hard for computers. Recognizing the presence of an original message is much harder when compressed data is used.

- Can be made more robust by
 - Larger key sizes
 - Introduction of permutations
 - Exploiting mathematical complexities
- Advanced methods
 - DES
 - 3 DES
 - AES
 - Twofish
 - Blowfish
 - RC4
 - IDEA
 - Serpent
 - ...
- These will be discussed in chapter 4

- A security model
- Security goals
 - Confidentiality
 - Authentication
 - Access control / authorization
 - Data integrity
 - Non-repudiation
 - Availability
- Security threats
- Security mechanisms
 - Encryption
 - ▶ Symmetric encryption
 - ▶ Asymmetric encryption
 - Hash functions
 - Message authentication codes

■ Symmetric encryption

- A few drawbacks:

- ▶ Both sender and receiver need to know the secret key
- ▶ How to exchange this key?

- Alternative technique might be useful

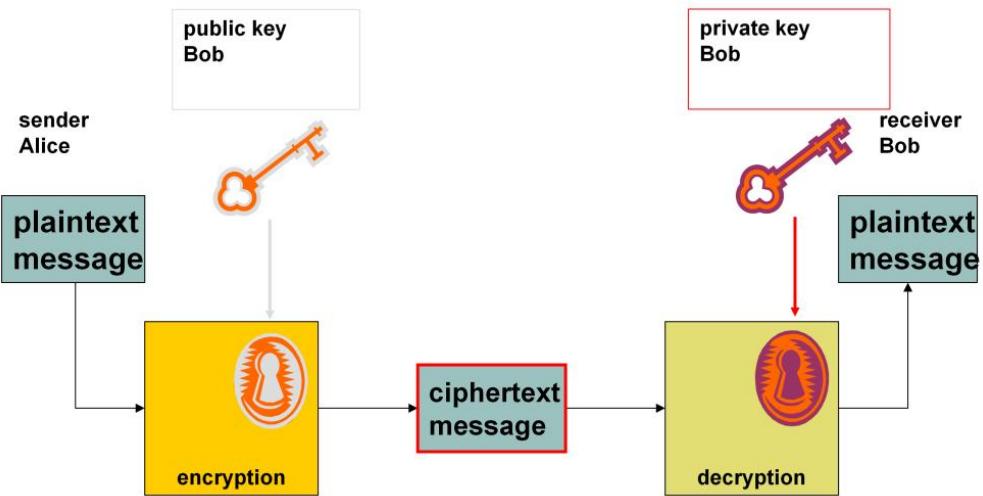
■ Asymmetric encryption or “public key encryption”

- Instead of **secret key** for each pair of users
- Key pair for each user
 - ▶ A **private key**
 - ✓ Only known to user himself
 - ▶ A **public key**
 - ✓ Public to everyone
 - ✓ Preferably as public as possible
- Principle:
 - ▶ No useful information may be derived about **private key** from knowledge of **public key**
 - ▶ Typically relies on hard mathematical problems
 - ✓ E.g. factorisation of the product of 2 large primes

55

In this course we use the expression “**private key**” for the secret information in an **asymmetric** encryption algorithm, and the expression “**secret key**” for the secret information in a **symmetric** encryption algorithm. The sole purpose of this wording is to distinguish between both types of algorithms. However, in literature about security and cryptography one may occasionally encounter the expression “**secret key**” for asymmetric encryption. The context should make clear whether one is dealing with a symmetric or with an asymmetric algorithm.

A consequence of the reliance on a hard mathematical problem is that the security of the algorithm is threatened not only by increasing computer power, but also by possible improvements in the solution techniques of the mathematical problem. This happened in the 90's with prime factorisation. The complexity of solving this problem was significantly reduced by the introduction of the GNFS and may even be further reduced in the future, hereby threatening RSA for not too long a key length (1024 bits; 768 bits should be considered broken by now).



56

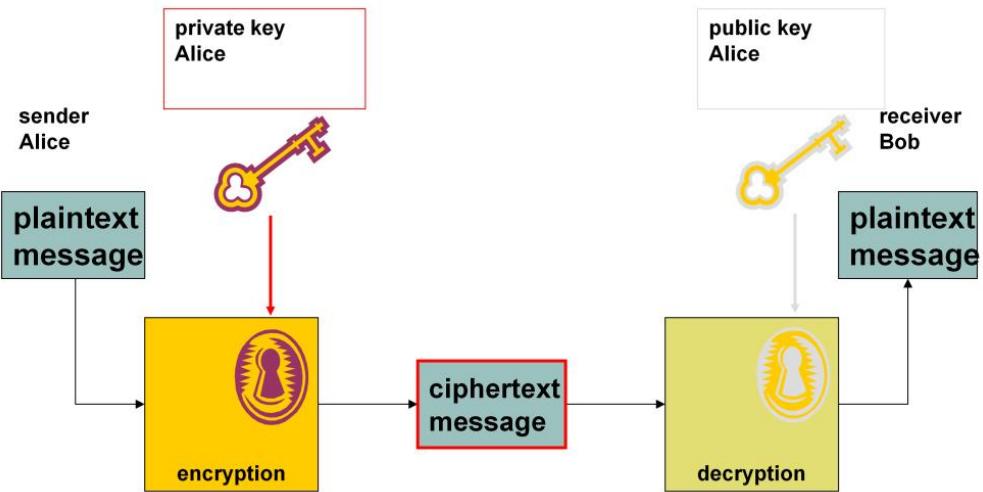
In a key pair for asymmetric encryption we write KU_A for A's public key van A, and KR_A for A's private key.

■ Concept

- **Transforms plaintext message into ciphertext message**
 - ▶ Transformation using the **public key** of the receiver of the message
- **Inverse operation: decryption**
 - ▶ Converting ciphertext message into plaintext message
 - ▶ Using receiver's **private key**
- **Only the owner of the private key can decrypt the message**

■ Comparison with symmetric encryption

- **No longer required to exchange secret key**
- **Typically much longer keys...**
- **...that aren't more secure**
 - ▶ 1024 bits RSA offers lower security than 128 bits AES
- **Much slower (by 2 to 3 orders of magnitude)**
 - ▶ No replacement, but complement:
 - ✓ E.g. encrypting secret key for symmetric encryption



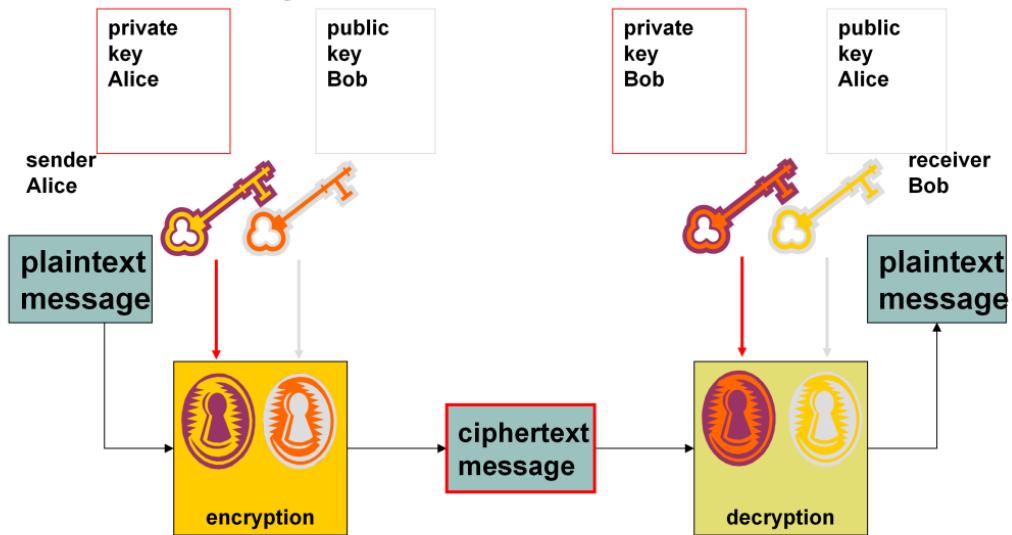
■ Concept

- **Transform plaintext message into ciphertext message**
 - ▶ Using the **private key** of the sender of the message
 - ✓ Only the owner of the **private key** can encrypt the message in this way
 - ✓ Kind of sender's "signature" for this message
- **Inverse operation: decryption**
 - ▶ Converting ciphertext message into plaintext message
 - ▶ Using sender's **public key**
 - ✓ Verification of sender's authenticity
 - ✓ Verification of message data-integrity (partially)
 - ✓ Sender must own **private key** corresponding to this public key
 - ▶ Requires knowledge of **public key ownership**

59

Data-integrity is only partially achieved. It is indeed impossible for an attacker to modify the message, but without additional measures, the attacker could still replay or suppress the message. This is sometimes called **message authentication**.

Confidentiality + authentication



■ Combining confidentiality + authentication

- Only sender A(lice) can send message encrypted using A(lice)'s **private key**
 - ▶ Authenticity of entity A(lice)
 - ▶ Data-integrity of message (partially)
 - ▶ To be verified using A(lice)'s **public key**
- Only receiver B(ob) can decrypt message that was encrypted using B(ob)'s **public key**
 - ▶ Confidentiality of communication between A(lice) and B(ob)
- Requires that each party knows the **public key** of the other party

■ Asymmetric encryption

- **Drawbacks**
 - ▶ Slow
 - ▶ Impractical to encrypt full message using asymmetric algorithm in order to achieve authentication
- **Combination with symmetric encryption for confidentiality**
 - ▶ See PGP
- **Combination with hash functions for authentication**
- **Suitable for short messages**
 - ▶ E.g. key exchange

■ **Examples:**

- **RSA (Rivest – Shamir – Adleman)**
- **EIGamal**
- **ECC (Elliptic Curve Cryptography)**
- **Diffie-Hellman**
- **XTR**

■ **These will be discussed in chapter 4**

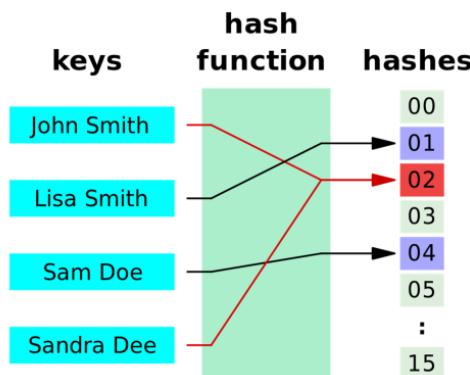
- A security model
- Security goals
 - Confidentiality
 - Authentication
 - Access control / authorization
 - Data integrity
 - Non-repudiation
 - Availability
- Security threats
- Security mechanisms
 - Encryption
 - ▶ Symmetric encryption
 - ▶ Asymmetric encryption
 - Hash functions
 - Message authentication codes

- Used in programming / databases to speed-up lookups

- Associate a short identifier to an entry

- Also used for

- Checksums, check digits, error correcting codes, ...
- And of course: security!



65

Hash functions are primarily used in hash tables, to quickly locate a data record (e.g., a dictionary definition) given its search key (the headword). Specifically, the hash function is used to map the search key to an index; the index gives the place in the hash table where the corresponding record should be stored. Typically, the domain of a hash function (the set of possible keys) is larger than its range (the number of different table indexes), and so it will map several different keys to the same index. Therefore, each slot of a hash table is associated with (implicitly or explicitly) a set of records, rather than a single record. For this reason, each slot of a hash table is often called a bucket, and hash values are also called bucket indices. Thus, the hash function only hints at the record's location — it tells where one should start looking for it. Still, in a half-full table, a good hash function will typically narrow the search down to only one or two entries. The figure demonstrates a hash function that maps names to integers from 0 to 15. There is a collision between keys "John Smith" and "Sandra Dee".

A **checksum or hash sum** is a small-size datum from a block of digital data for the purpose of detecting errors which may have been introduced during its transmission or storage. It is usually applied to an installation file after it is received from the download server. By themselves checksums are often used to verify data integrity, but should not be relied upon to also verify data authenticity.

A **check digit** is a form of redundancy check used for error detection on identification numbers (e.g. bank account numbers) which have been input manually. It is analogous to a binary parity bit used to check for errors in computer-generated data. It consists of a single digit (sometimes more than one) computed by an algorithm from the other digits (or letters) in the sequence input.

Forward error correction (FEC) or channel coding are techniques used for controlling errors in data transmission over unreliable or noisy communication channels. The central idea is the sender encodes the message in a redundant way by using an error-correcting code (ECC). The redundancy allows the receiver to detect a limited number of errors that may occur anywhere in the message, and often to correct these errors without retransmission.

- Alternative terms

- “hash code”
- “message digest”

- Notation: $h = H(M)$

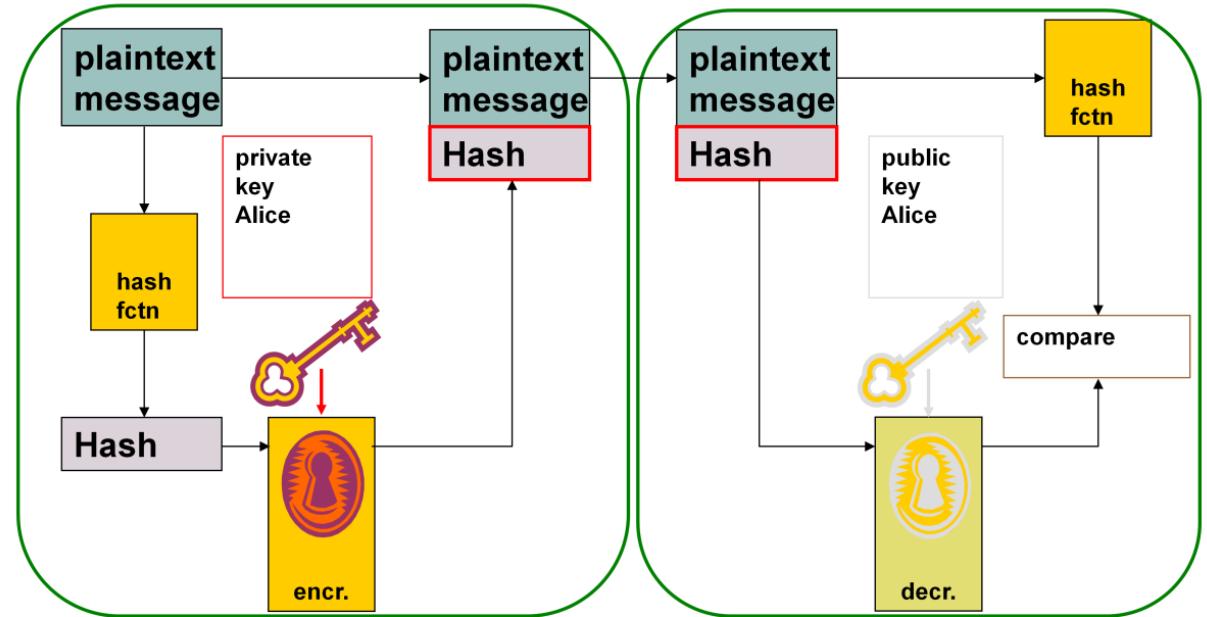
- Fixed number of bits
 - ▶ 128 for MD5, 160 for SHA-1

- Acts as a “fingerprint”

- Used for modification detection
 - ▶ Changing messages changes corresponding hash value
- By itself not secure
 - ▶ E.g. no authentication

sender Alice

receiver Bob



67

In this use of a hash function for message authentication, an asymmetric encryption algorithm is used (to encrypt the non-secret hash value). Authentication of communicating entity A(lice) is provided, since only A(lice) can use this particular key in communication with B(ob). Authentication of the sent message is provided, since a modified message would correspond to different (encrypted) hash value. To further ensure that message tampering is not possible, a counter sequence is typically added to the message and encrypted together with the hash value.

The encrypted hash value is the “digital signature” of the corresponding plaintext message. This is one of the most common schemes for digital signatures (see later), where RSA or ElGamal can be used as an encryption algorithm.

- Should be able to work for input messages of any length
- Hash values must be easy to compute
- One-way function
 - Given h , infeasible to find message x so that $h=H(x)$
- Weak collision resistance
 - Given message x , infeasible to find message y ($\neq x$) so that $H(y)=H(x)$
- Strong collision resistance
 - Infeasible to find different messages x and y so that $H(y)=H(x)$

68

The main goal of the “one-way function” property is to make it impossible to recover the original message from the hash value only. This is generally not a major issue as a hash function typically maps a huge message space on a much smaller hash value space. This is however essential when the message space is rather small (e.g. when hashing passwords).

We'll see why weak and strong collision resistance are important when dealing with the generation of digital signatures. Both forms of collision resistance requirement exclude too simple and too naive hash function implementations (e.g. using XOR operations).

Strong collision resistance isn't required for all applications.

Note that the requirement “easy to compute” is not always valid. When encrypting passwords, special hash algorithms or “key derivation functions” (such as bcrypt, scrypt or PBKDF) are used that require more time to compute. This has the advantage that attackers that somehow gain access to the hashed passwords can not too easily calculate which hashcode is derived from each password. Algorithms such as bcrypt include adaptive functionality: over time, the variables such as the iteration count can be increased to make it slower in order to keep the algorithm resistant to brute-force search attacks even with increasing computation power.

- Can we replace the plaintext from Alice?
- Weak collision resistance requirement
 - If absent, possible to replace plaintext message with different message with same hash value
 - ▶ Will indeed yield same encrypted hash value
 - ▶ No knowledge of private key required
 - ▶ No longer guaranteed message origin authentication or data-integrity

69

For a good hash function with n bits hash value, generating a forged message should be hard, typically in the order of 2^n . I.e. it should require a brute force attack in which almost all possible plaintext messages should be generated before a plaintext message is found with the same hash value.

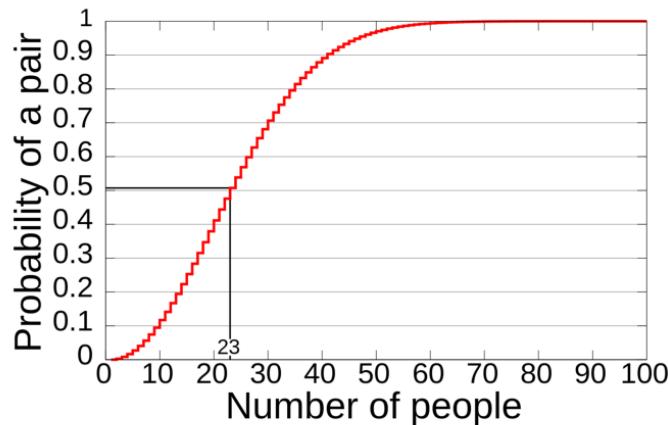
- Can we create a false plaintext / hash pair?
- Strong collision resistance requirement
 - Important for digital contracts
 - ▶ Otherwise attacker may generate 2 variants of contract/message with different contents but identical hash value
 - ✓ Having one message signed by the other party also yields a valid digital signature by the other party of the other message
 - ✓ Indeed, encrypted hash values are identical
 - ✓ No knowledge of private key required
 - ▶ Breaks non-repudiation

70

For a good hash function with n bits hash value, generating a forged message should be hard, typically in the order of $2^{n/2}$. I.e. it should require a brute force attack in which $2^{n/2}$ combinations are generated before two plaintext / hash pairs are found with the same hash values.

■ Problem with strong collision resistance

- Much harder to achieve than weak collision resistance
- Vulnerable to “birthday” attack



Computed probability of at least two people sharing a birthday amongst X people.

71

Intuitively this can be understood as follows. Although it is difficult to find a person with the same birthday as you (probability 1 in 365), the likelihood that at least two persons in a group shares a birthday is much higher. Similarly, the probability of finding a plaintext / hashvalue pair with same hash value amongst a group of plaintext messages is much higher than identifying a plaintext with the same hash as a specific previous plaintext message.

■ Problem with strong collision resistance

- Much harder to achieve than weak collision resistance
- Vulnerable to “birthday” attack
 - ▶ Derives its name from seemingly paradoxically high probability that 2 persons within a group celebrate their birthday on the same day
 - ▶ Hash values with n bits: $N = 2^n$ possible hash values
- Vulnerable to “birthday” attack
 - ▶ Probability of having 2 messages from a group of k messages with the same hash values is:
 $P(N,k)=1-N!/((N - k)! N^k)$
 - ✓ Can be approximated ($1 \ll k \ll N$) as
 $P(N,k) \approx 1 - \exp(-k^2/(2N))$
 - ✓ From which $k \approx -N^{1/2} \ln(1-P(N,k))$
 - ✓ $P(N,k) = 0,99$ implies $k \approx 4,6 \times N^{1/2} \approx 4,6 \times 2^{n/2}$
 - ▶ This is the best possible result for n bits hash value

■ Problem with strong collision resistance

● Use variant of “birthday” attack

► Generate $k \approx 2^{n/2}$ variants of original message

- ✓ E.g. by playing with the number of white spaces (using 64 optional white spaces allows to create 2^{64} message variants)
- ✓ Compute hash value for each variant in this first group

► Generate $k \approx 2^{n/2}$ variants of forged message

- ✓ Compute hash value of each variant in this second group

73

The probability of finding variants in groups 1 and 2 with identical hash values is $P(N,k) \approx 1 - \exp(-k^2/N)$. An approximation for $P(N,k) \approx 1 - {}_k C_0 \times {}_{N-k} C_k / {}_N C_k$, where ${}_N C_k = N!/(k!(N-k)!)$ = the number of combinations of k elements from a group of N .

Having the selected message from group 1 signed also yields digital signature for corresponding selected message from group 2.

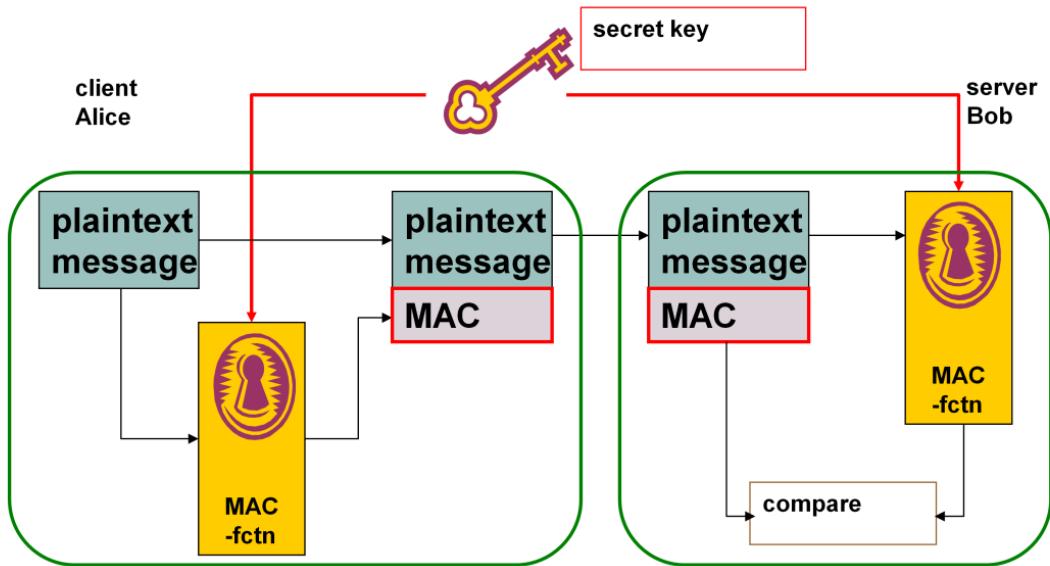
- A security model
- Security goals
 - Confidentiality
 - Authentication
 - Access control / authorization
 - Data integrity
 - Non-repudiation
 - Availability
- Security threats
- Security mechanisms
 - Encryption
 - ▶ Symmetric encryption
 - ▶ Asymmetric encryption
 - Hash functions
 - Message authentication codes

■ Message Authentication Code (MAC)

- “Cryptographic checksum”
- Uses both (plain)text and shared key as input

■ Additional functionality

- Checks whether message was modified
- Checks whether message originates from correct sender
- When counter is included in message: checks whether message order was preserved





■ MAC vs symmetric encryption

- **Similarity:** both sender and receiver share a **secret key**
 - ▶ Similar authentication mechanism
 - ▶ Drawback that receiver also needs to know **secret key**
- **Differences with symmetric encryption**
 - ▶ No confidentiality function
 - ▶ “Irreversible encryption”: decryption impossible

■ MAC vs hash function

- MAC also depends on **secret key**, while hash value only depends on message
- Notation: $MAC = C_K(M)$
 - where K : secret key ; M : message

■ Authentication of communicating entity Alice

- Only Alice can use this particular key in a communication with Bob
- Verification by Bob: recomputing MAC corresponding to received message using shared secret information

■ Authentication of transmitted message

- Modified message would yield different MAC
- Partial data-integrity

■ Adding counter sequence to message

- Impossible to tamper with message order
- Counter sequence secured by MAC
- Protection against replay

■ Benefits over symmetric encryption

- Same authentication function as symmetric encryption
- But no confidentiality function
- Allows for separation of both security functions
 - ▶ E.g. verifying authenticity on separate (trusted) server (using MAC)
 - ▶ E.g. confidentiality up to the work station (using encryption)
- Also allows authenticated storage of message
 - ▶ Storing MAC and secret key separately
 - ▶ Allows later verification
 - ✓ E.g. against tampering with program files

- Original message is available (no confidentiality)
 - “known plaintext” attacks are possible!
 - ▶ Deriving secret key from this should be hard
- Requirements
 - From the observation of M , $C_K(M)$ it must be infeasible to construct new message M' so that $C_K(M') = C_K(M)$
 - ▶ Required for (partial) data-integrity
 - Values of $C_K(M)$ should be uniformly distributed over space of possible MACs: the probability for two arbitrary messages M and M' to yield identical MACs $C_K(M') = C_K(M)$ should be equal to 2^{-n}

■ Ideal MAC

- Only vulnerable to brute force attack
- Testing all possible secret keys
 - ▶ For k bits key length: order of 2^k operations
- Search for messages yielding given MAC value
 - ▶ Comparable to weak collision resistance for hash function
 - ▶ Requires “chosen text” attack (or knowledge of secret key)
 - ✓ Attacker can't generate MACs himself without the secret key
 - ✓ So, must be generated by target
 - ▶ For n bits MAC size: order of 2^n operations

■ Example hash functions

- **MD5 (not safe anymore)**
- **SHA family**
 - ▶ SHA-0, SHA-1, SHA-224, SHA-256, SHA-384, SHA-512, SHA-3

■ Further information

- **Algorithms: Chapter 4**
- **Digital signature: Chapter 5**

- A security model
- Security goals
 - Confidentiality
 - Authentication
 - Access control / authorization
 - Data integrity
 - Non-repudiation
 - Availability
- Security threats
- Security mechanisms
 - Encryption
 - ▶ Symmetric encryption
 - ▶ Asymmetric encryption
 - Message authentication codes



Besides the lecturers' own material, many third party, often copyrighted, material is reused within this lecture (e.g. in the notes) under the 'fair use' approach, for sake of educational purpose only, and very limited edition. As a consequence, the current slide set presentation usage is restricted, and is falling under usual copyrights usage.

At the end of every lecture, appropriate references to used materials are included.

- This work contains content adapted from, amongst other, the following sources (in no particular order)
 - William Stallings, “**Cryptography and Network Security, principles and practices**”, 6th (international) edition, Prentice Hall, 2010;
 - Matt Bishop, “**Computer Security: Art and Science**”, Addison Wesley, Pearson Education, 2003, ISBN-13: 978-0-201-44099-7
 - Lecture slides: “Informatiebeveiliging”, Universiteit Gent, Eric Laermans & Thom Dhaene
 - Wikipedia (additional note page descriptions)
 - <https://crypto.stackexchange.com>



Network and Computer Security

Chapter 3 – Network and communication security

Prof. dr. ir. Eli De Poorter

■ Until now: basic concepts

- **Symmetric encryption**
- **Asymmetric encryption**
- **Hash functions**
- **Message authentication codes**

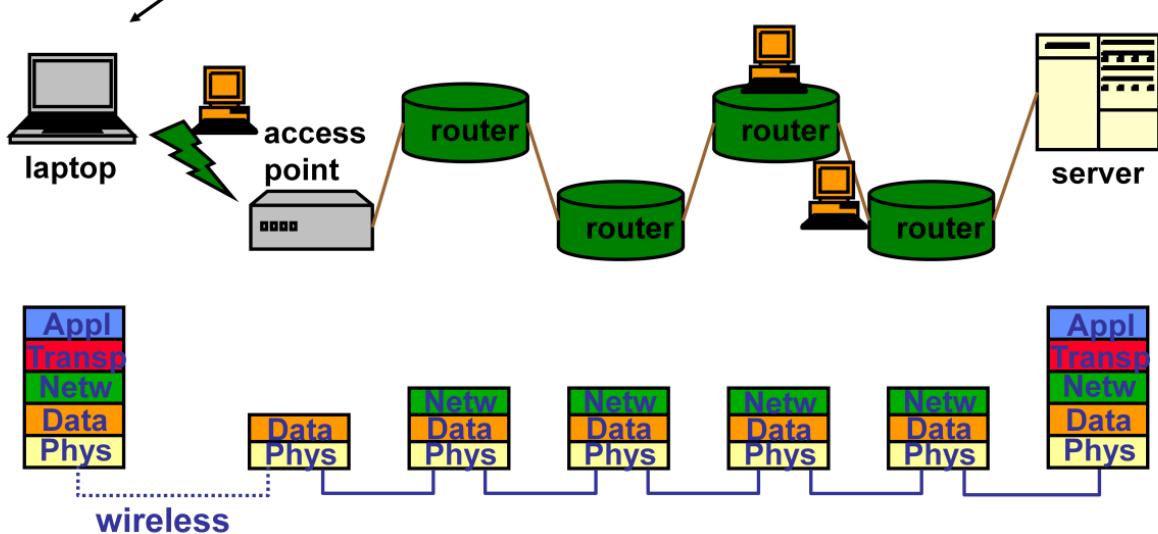
■ From now

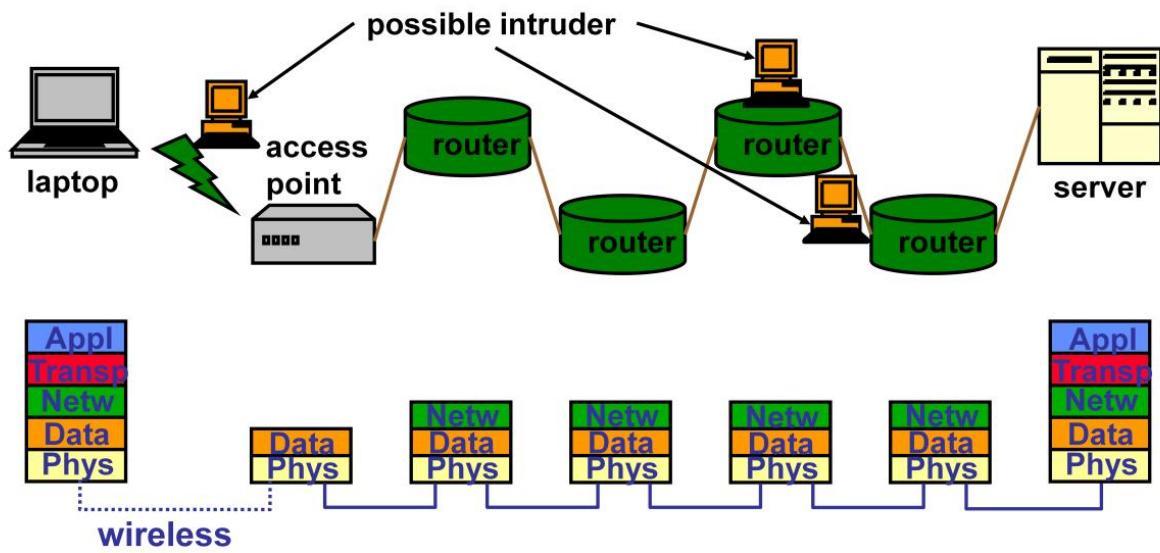
- **Applying these concepts to design security protocols for networked devices**
 - ▶ Secure configuration of devices
 - ▶ Exchanging keys
 - ▶ Secure networking protocols
 - ▶ Firewalls

- Network model
- Secure configuration of devices
 - SSH (Secure Shell)
- Exchanging keys
 - Out of band
 - Diffie-Hellman
 - Asymmetric encryption
 - Trusted third party
 - ▶ Key Distribution Centre (KDC)
 - ▶ Public Key Infrastructure (PKI)
- Secure networking protocols
 - Transport layer: TLS & SSL
 - Network layer: IPSec & VPN
 - Data link layer: WEB & WPA
- Firewalls
 - Packet filter
 - Circuit-level gateway
 - Application-level gateway (proxy)

- Network model
- Secure configuration of devices
- Exchanging keys
- Secure networking protocols
- Firewalls

possible intruder (prevention: see chapter 4)



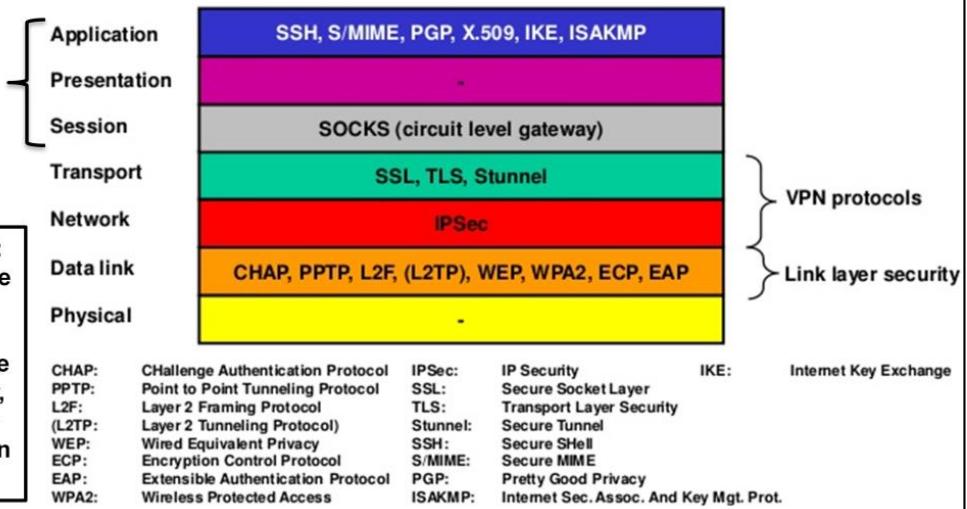


Often taken together
as single application
layer



Focus of chapter 4:
system and software
security

However, we include
SSH in this chapter,
since it is used for
secure configuration
of the network

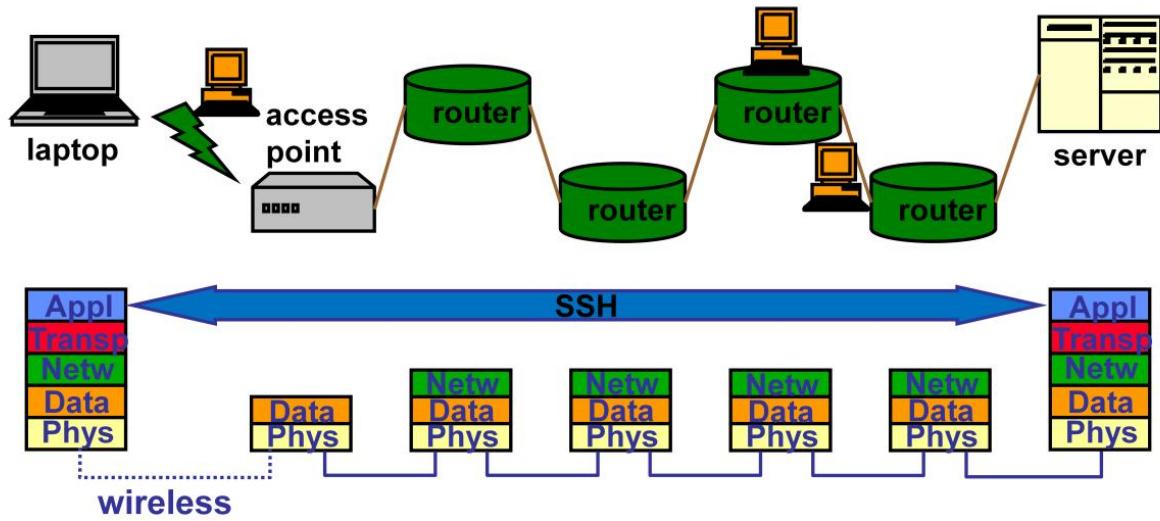


7

SSH is an application layer security protocol. However, it includes a “transport layer protocol” component which runs on top of the OSI transport layer, acting as a secure connection between other OSI application layer protocols (such as HTTP) and the actual OSI transport layer. Due to the confusingly named RFC 4253, SSH is often depicted as a transport layer protocol, which it is not.

More information can be found in:
 RFC 4253 Transport Layer Protocol
 RFC 4251 Application layer protocol

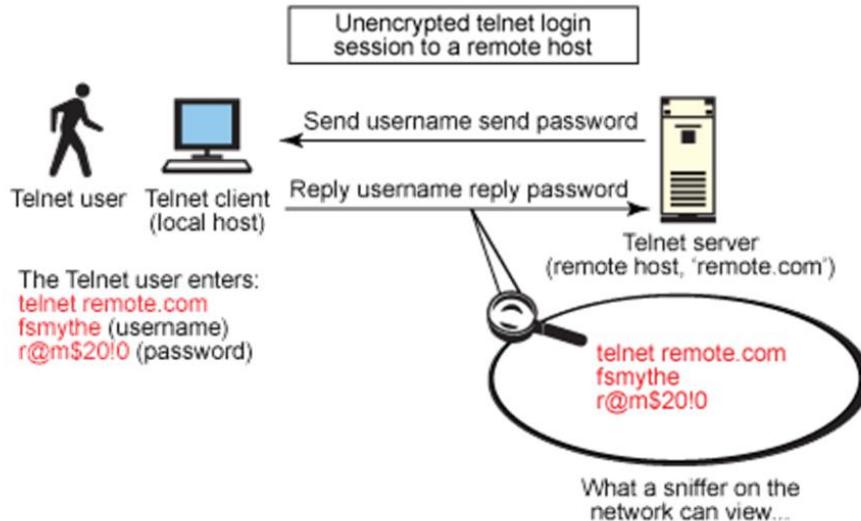
- Network model
- Secure configuration of devices
 - SSH (**Secure Shell**)
- Exchanging keys
- Secure networking protocols
- Firewalls



p. 9

Note that many routers and even switches also include SSH management functionality (e.g. they include some application layer functionality).

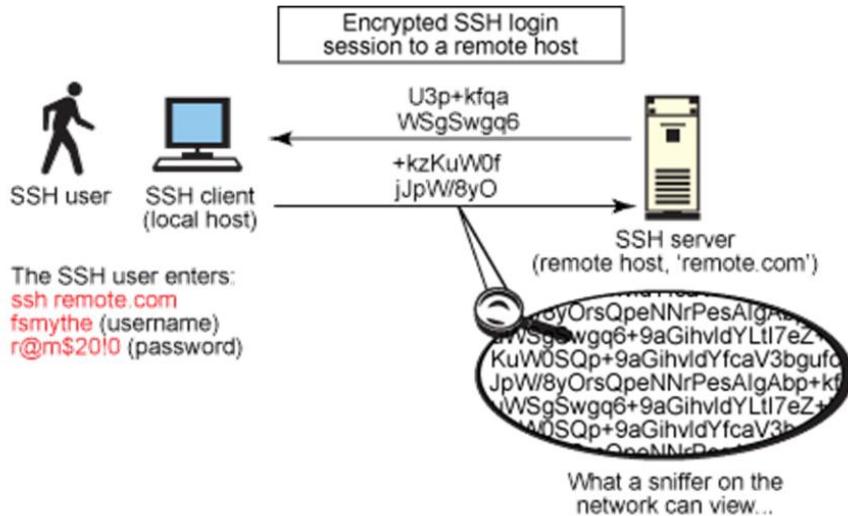
■ Telnet for server management: inherently unsafe



10

Secure Shell (SSH) was intended and designed for protection when remotely accessing another host over the network. The figures show how easily a telnet session can be casually viewed by anyone on the network using a network-sniffing application such as Wireshark. When using an unsecured, "clear text" protocol such as telnet, anyone on the network can pilfer your passwords and other sensitive information. The figure shows user fsmythe logging in to a remote host through a telnet connection. He enters his user name fsmythe and password r@m\$20!0, which are both then viewable by any other user on the same network as our hapless and unsuspecting telnet user.

■ Server management using SSH



11

Secure Shell (SSH) was intended and designed to offer protection when remotely accessing another host over the network and was designed to be relatively simple and inexpensive to implement. The initial version, SSH1, focused on providing a secure remote logon facility to replace Telnet and other remote logon schemes that provided no security. A new version, SSH2, provides a standardized definition of SSH and improves on SSH1 in numerous ways.

This slide provides an overview of a typical SSH session and shows how the encrypted protocol cannot be viewed by any other user on the same network segment. Every major Linux and UNIX distribution now comes with a version of the SSH packages installed by default—typically, the open source OpenSSH packages—so there is little need to download and compile from source. If you're not on a Linux or UNIX platform, a plethora of open source and freeware SSH-based tools are available that enjoy a large following for support and practice, such as WinSCP, Putty, FileZilla, TTSSH, and Cygwin (POSIX software installed on top the Windows operating system). These tools offer a UNIX- or Linux-like shell interface on a Windows platform. In addition, Windows 10 will include SSH support by default.

■ SSH (Secure SHell)

- **Features**

- ▶ Protocol for secure remote login
- ▶ Also supports tunneling (VPN use)
 - ✓ TCP tunneled through SSH connection
- ▶ Can also be used to transfer files using associated protocols
SCP (Secure CoPy) or **SFTP** (SSH File Transfer Protocol)
- ▶ X-session forwarding and port forwarding

- **Originally developed by SSH Communications Security Corp., Finland**

- **Two distributions available:**

- ▶ Freeware (www.openssh.org)
- ▶ Commercial version

- **IETF standard (RFCs 4250-4256)**

p. 12

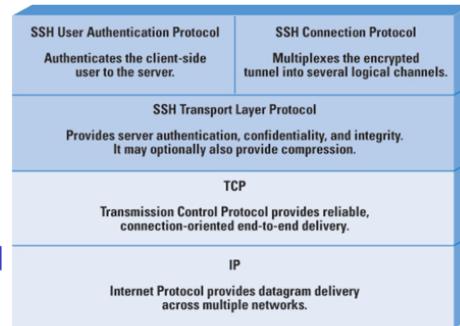
SSH provides a general client-server capability and encrypts the network exchange. In addition, it can be used to secure such network functions as file transfer (Secure Copy (SCP) and Secure File Transfer Protocol (SFTP)), X session forwarding, and port forwarding to increase the security of other insecure protocols. Various types of encryption are available, ranging from 512-bit encryption to as high as 32768 bits, inclusive of ciphers, like Blowfish, Triple DES, CAST-128, Advanced Encryption Scheme (AES), and ARCFour. Higher-bit encryption configurations come at a cost of greater network bandwidth use.

SSH2 is documented as a proposed standard in RFCs 4250 through 4256.

- **SSH transport layer protocol**
 - Server authentication, confidentiality, and integrity
 - Compression (optional)
 - On top of *reliable transport layer* (e.g. TCP)

- **SSH user authentication protocol**
 - Client authentication
 - On top of SSH transport layer

- **SSH connection protocol**
 - Multiplexes secure tunnel provided by SSH transport layer and user authentication layer into several logical channels
 - Logical channels can be used for various purposes



p. 13

IETF RFCs 4251 through 4256 define SSH as the "Secure Shell Protocol for remote login and other secure network services over an insecure network." The shell consists of three main elements.

Transport Layer Protocol: This protocol accommodates server authentication, data confidentiality, and data integrity with perfect forward privacy (that is, if a key is compromised during one session, the knowledge does not affect the security of earlier sessions). The transport layer is responsible for key exchange and server authentication. It sets up encryption, integrity verification, and (optionally) compression and exposes to the upper layer an API for sending and receiving plain text packets. This layer is typically run over a TCP/IP connection but can also be used on top of any other dependable data stream.

User Authentication Protocol: This protocol authenticates the client to the server and runs over the transport layer. Common authentication methods include password, public key, keyboard-interactive, GSSAPI, SecureID, and PAM.

Connection Protocol: This protocol multiplexes the encrypted tunnel to numerous logical channels, running over the User Authentication Protocol. The connection layer defines channels, global requests, and the channel requests through which SSH services are provided. A single SSH connection can host multiple channels concurrently, each transferring data in both directions. Channel requests relay information such as the exit code of a server-side process. The SSH client initiates a request to forward a server-side port.

The "SSH transport layer protocol" is not to be confused with the "OSI transport layer": it is part of the SSH protocol and runs on top of the layer 4 OSI network layer (i.e.: the actual transport layer of the network stack, e.g. TCP or any other reliable transport layer protocol).



UNIVERSITEIT
GENT



SSH



INTEC

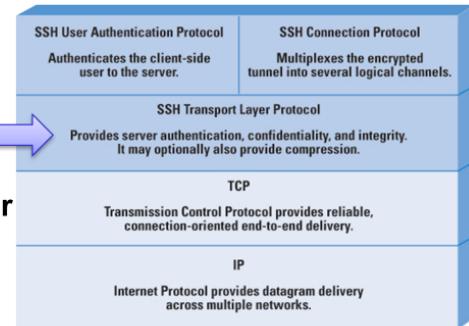
■ Security features

- **Uses well-established algorithms for encryption, data-integrity, key exchange, and public key management**
- **Encryption with at least 128 bit keys**
- **Security algorithm negotiation**
 - ▶ Basic protocol needn't be changed when switching to other, newer algorithms

p. 14

■ Features

- **Server authentication, confidentiality, and integrity**
- **Compression (optional)**
- **On top of *reliable* transport layer (e.g. TCP)**

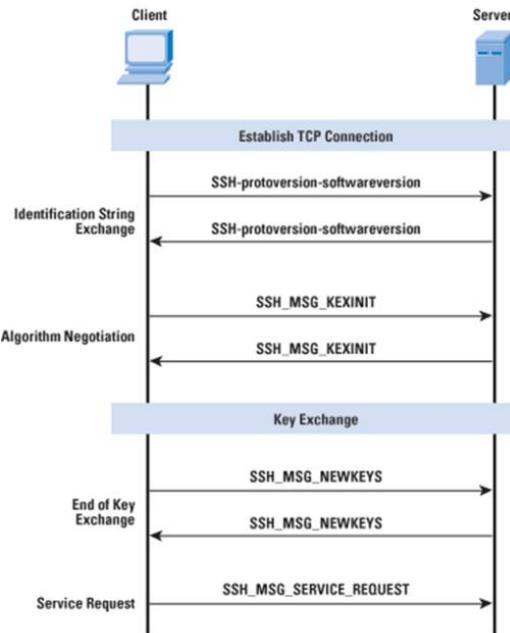


p. 15

Server authentication occurs at the transport layer protocol, based on the server possessing a public-private key pair. A server may have multiple host keys using multiple different asymmetric encryption algorithms. Multiple hosts may share the same host key. In any case, the server host key is used during key exchange to authenticate the identity of the host. For this authentication to be possible, the client must have presumptive knowledge of the server public host key. RFC 4251 dictates two alternative trust models that can be used:

- The client has a local database that associates each host name (as typed by the user) with the corresponding public host key. This method requires no centrally administered infrastructure and no third-party coordination. The downside is that the database of name-to-key associations may become burdensome to maintain.
- The host name-to-key association is certified by a trusted *Certification Authority* (CA) (see later). The client knows only the CA root key and can verify the validity of all host keys certified by accepted CAs. This alternative eases the maintenance problem, because ideally only a single CA key needs to be securely stored on the client. On the other hand, each host key must be appropriately certified by a central authority before authorization is possible.

■ Security algorithm negotiation



p. 16

SSH includes negotiation algorithms to determine suitable encryption algorithms. The SSH Transport Layer packet exchange consists of a sequence of steps, illustrated above. First, the client establishes a TCP connection to the server with the TCP protocol (which is not part of the Transport Layer Protocol). When the connection is established, the client and server exchange the following data packets (using the data field of a TCP segment).

Identification string exchange

The first step, the *identification string exchange*, begins with the client sending a packet with an identification string of the form: "SSH-*protoversion-softwareversion SP comments CR LF*", where SP, CR, and LF are space character, carriage return, and line feed, respectively. An example of a valid string is `SSH-2.0-billsSSH_3.6.3q3<CR><LF>` (SSH protocol version 2, software version billsSSH_3.6.3q3). The server responds with its own identification string.

Algorithm negotiation

Next comes *algorithm negotiation*. Each side sends an `SSH_MSG_KEXINIT` containing lists of supported algorithms in the order of preference to the sender. Each type of cryptographic algorithm has one list. The algorithms include key exchange, encryption, MAC algorithm, and compression algorithm. For each category, the algorithm chosen is the first algorithm on the client's list that is also supported by the server.

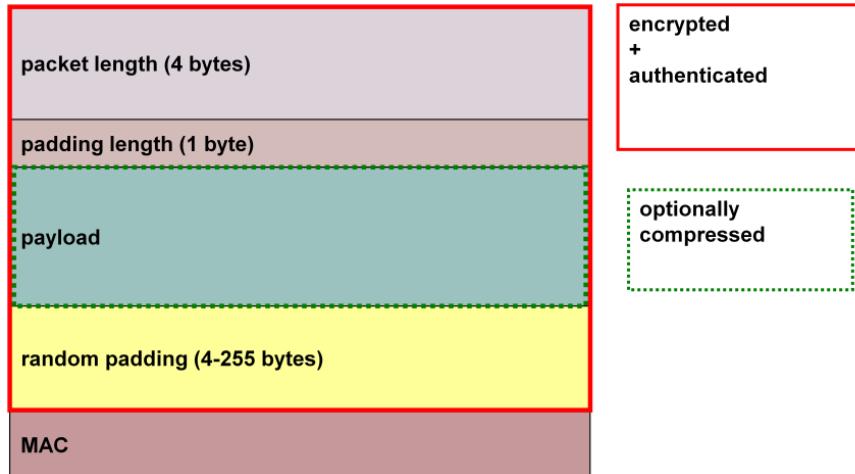
Key exchange

The next step is *key exchange*. The specification allows for alternative methods of key exchange, but at present only two versions of Diffie-Hellman (asymmetrical encryption) key exchange are specified. Both versions are defined in RFC 2409 and require only one packet in each direction. The *end of key exchange* is signaled by the exchange of `SSH_MSG_NEWKYS` packets. At this point, both sides may start using the generated keys.

Service request

The final step is *service request*. The client sends an `SSH_MSG_SERVICE_REQUEST` packet to request either the User Authentication or the Connection Protocol. Subsequent to this request, all data is exchanged as the payload of an SSH Transport Layer packet, protected by encryption and MAC.

■ Binary packet protocol



p. 17

Each data packet is in the following format:

Packet length: Packet length is the length of the packet in bytes, not including the packet length and Message Authentication Code (MAC) fields.

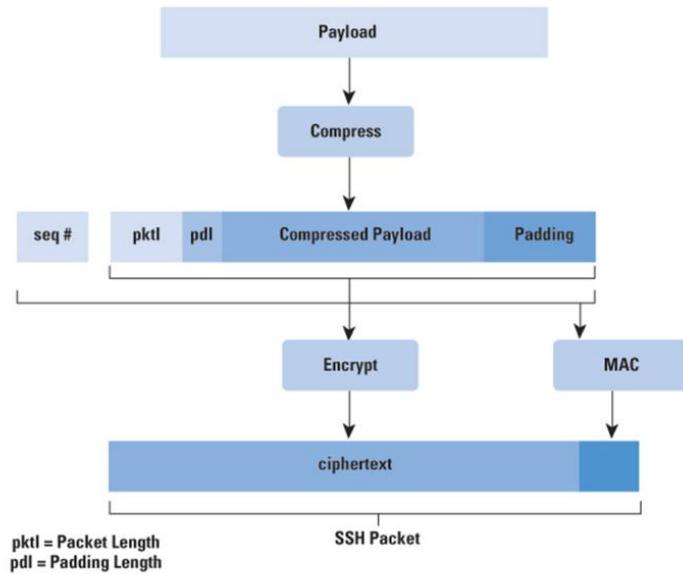
Padding length: Padding length is the length of the random padding field.

Payload: Payload constitutes the useful contents of the packet. Prior to algorithm negotiation, this field is uncompressed. If compression is negotiated, then in subsequent packets this field is compressed. Maximal (uncompressed) size is (at least) 32768 bytes (longer packets may be supported, depending on the implementation).

Random padding: After an encryption algorithm is negotiated, this field is added. It contains random bytes of padding so that that total length of the packet (excluding the MAC field) is a multiple of the cipher block size, or 8 bytes for a stream cipher. The length of the padding is arbitrary to thwart traffic analysis.

Message Authentication Code (MAC): If message authentication has been negotiated, this field contains the MAC value. The MAC value is computed over the entire packet plus a sequence number, excluding the MAC field. The sequence number is an implicit 32-bit packet sequence that is initialized to zero for the first packet and incremented for every packet. The sequence number is not included in the packet sent over the TCP connection.

■ Binary packet protocol



p. 18

Process illustrating the calculation of encrypted packets.

■ Encryption algorithm

- Negotiated during key exchange
- Supported algorithms
 - ▶ REQUIRED: 3des-cbc (“168” bit key)
 - ▶ RECOMMENDED: aes128-cbc
 - ▶ OPTIONAL: aes192-cbc, aes256-cbc, arcfour (i.e. RC4), etc.
- May be different for each direction
- Secret key established during key exchange
- All packets sent in 1 direction SHOULD be considered a single data stream

Supported encryption algorithms	
3des-cbc*	Three-key Triple Digital Encryption Standard (3DES) in Cipher-Block-Chaining (CBC) mode
blowfish-cbc	Blowfish in CBC mode
twofish256-cbc	Twofish in CBC mode with a 256-bit key
twofish256-cbc	Twofish in CBC mode with a 256-bit key
twofish192-cbc	Twofish with a 192-bit key
twofish128-cbc	Twofish with a 128-bit key
aes256-cbc	Advanced Encryption Standard (AES) in CBC mode with a 256-bit key
aes192-cbc	AES with a 192-bit key
aes128-cbc**	AES with a 128-bit key
Serpent256-cbc	Serpent in CBC mode with a 256-bit key
Serpent192-cbc	Serpent with a 192-bit key
Serpent128-cbc	Serpent with a 128-bit key
arcfour	RC4 with a 128-bit key
cast128-cbc	CAST-128 in CBC mode

p. 19

■ MAC algorithm

- Negotiated during key exchange
- Supported algorithms
 - ▶ REQUIRED: hmac-sha1
 - ▶ RECOMMENDED: hmac-sha1-96, hmac-sha2-256
 - ✓ hmac-sha1-96: first 96 bits of hmac-sha1
 - ▶ OPTIONAL: hmac-md5, hmac-md5-96, hmac-sha2-512, etc.
- May be different for each direction
- MAC = mac(key, seq. nr. | unencrypted packet)
 - ▶ Implicit seq. nr. (4 bytes), not sent with packet
 - ▶ Seq. nr. initialised to 0, incremented after each packet

Supported MAC algorithms		
hmac-sha1*	HMAC-SHA1; Digest length = Key length = 20	
hmac-sha1-96**	First 96 bits of HMAC-SHA1; Digest length = 12; Key length = 20	
hmac-md5	HMAC-SHA1; Digest length = Key length = 16	
hmac-md5-96	First 96 bits of HMAC-SHA1; Digest length = 12; Key length = 16	

p. 20

The implicit sequence number for the MAC is never reset, not even if the keys and the algorithms for the SSH-connection are renegotiated later on.

■ **New symmetric keys are generated and exchanged
for each new session**

- Different keys for encryption and for MAC algorithms
- Can even support different keys for each direction

■ **How are the symmetric keys exchanged?**

- In a secure way?

■ Key exchange: shared session key

- Server sends

- ▶ Its shared session key

- ✓ Diffie Helman (see key exchange chapter)

- ▶ Its host (public) key

- ✓ For authentication
 - ✓ Support for DSA, ECDSA, ED25519 or RSA

- ▶ A signature over (among other things)

- ✓ SSH_MSG_KEXINIT messages from both client and server (including cookies)
 - ✓ DH public keys
 - ✓ Host (public) key

p. 22

Key exchange happens in two phases: (i) first a shared key is generated using Diffie-Helman, (ii) afterwards the shared key is signed with the public key of the client to provide authentication.

For the creation and exchange of a symmetric key, Diffie Helman (DH) is used (see key exchange chapter). The way SSH uses DH is as an ephemeral algorithm: DH parameters are generated for individual sessions, and are destroyed as soon as they're no longer needed. The only thing the long-lasting key pair is used for is authentication. This gives forward secrecy: stealing the private key doesn't let you decrypt old sessions. This key agreement results in a shared session key.

In addition, after a shared key is established, the client host key is used to sign the Diffie-Hellman parameters. SSH protocol 2 supports signatures based on DSA, ECDSA, ED25519 or RSA signature algorithms. They are only used after key exchange to sign the DH messages and prove the client has the private key (one side of DH parameters being signed is enough to prevent a MITM, because the attacker can't impersonate both client and server; it's easier to make the server always have to have a keypair than make the client always have to have a keypair).

Now that the shared session key has been securely generated and authenticated, the rest of the session is encrypted using a symmetric cipher

■ Key exchange: server authentication

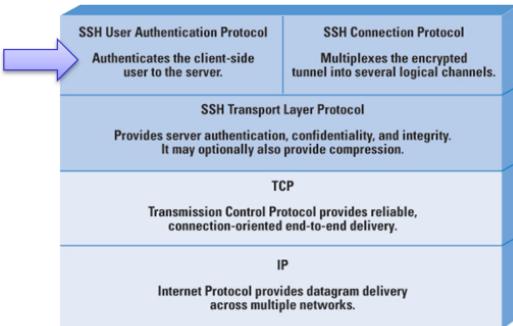
- Based on server host key
- Various possible techniques for key identification
 - ▶ Using local databases
 - ▶ Using certification authorities (CA) and certificates
 - ▶ Using offline channels
 - ✓ E.g. key “fingerprint”
 - ▶ Using “best effort” approach
 - ✓ Accept host key without check during first connection to the server
 - ✓ Save host key in local database
 - ✓ Check against saved key for later connections
- See alternatives from key exchange chapter

■ Key re-exchange

- Possible at any moment (except during key exchange)
- Can be initiated by either party
- Session ID doesn't change
- Algorithms may be changed
- Session keys are changed
- Recommended
 - ▶ After some amount of data (e.g. 1 GB)
 - ▶ After some amount of time (e.g. 1 h)

■ SSH user authentication protocol

- Client authentication
- On top of SSH transport layer



p. 25

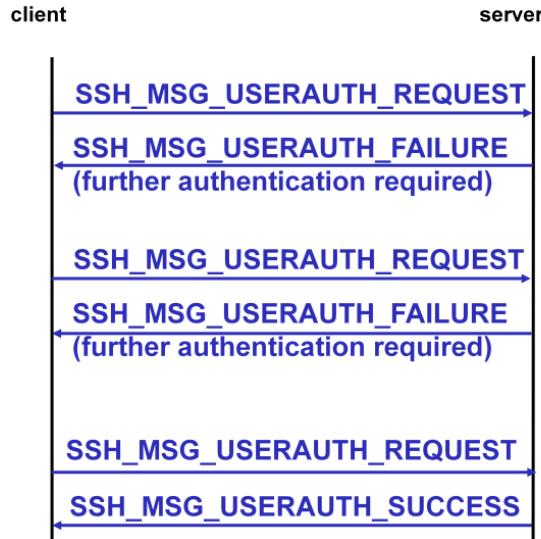
The *User Authentication Protocol* provides the means by which the client is authenticated to the server.

- Underlying SSH transport layer provides confidentiality and data-integrity
- Has access to session ID
- Server timeout for authentication
- Limited number of failed authentication attempts
- 3 authentication methods:
 - Public key
 - password
 - Host based

p. 26

The server will disconnect the client if the authentication has not been accepted within the timeout period (recommended value for the timeout: 10 minutes).

The recommended value for the maximal number of failed authentication attempts is 20 within a single session.



p. 27

The message exchange involves the following steps:

1. The client sends a `SSH_MSG_USERAUTH_REQUEST` with a requested method of none.
2. The server checks to determine if the username is valid. If not, the server returns `SSH_MSG_USERAUTH_FAILURE` with the partial success value of false. If the username is valid, the server proceeds to step 3.
3. The server returns `SSH_MSG_USERAUTH_FAILURE` with a list of one or more authentication methods to be used.
4. The client selects one of the acceptable authentication methods and sends a `SSH_MSG_USERAUTH_REQUEST` with that method name and the required method-specific fields. At this point, there may be a sequence of exchanges to perform the method.
5. If the authentication succeeds and more authentication methods are required, the server proceeds to step 3, using a partial success value of true. If the authentication fails, the server proceeds to step 3, using a partial success value of false.
6. When all required authentication methods succeed, the server sends a `SSH_MSG_USERAUTH_SUCCESS` message, and the Authentication Protocol is over.

The `SSH_MSG_USERAUTH_REQUEST` message contains the following fields: the user name, the service name, the method name, plus additional fields specific for the authentication method used. *Username* is the authorization identity the client is claiming, *service name* is the facility to which the client is requesting access (typically the SSH Connection Protocol), and *method name* is the authentication method being used in this request ("publickey", "password" or "hostbased").

If the server either rejects the authentication request or accepts the request but requires one or more additional authentication methods, the server sends a `SSH_MSG_USERAUTH_FAILURE` message. The `SSH_MSG_USERAUTH_FAILURE` message isn't necessarily an indication of a fatally unsuccessful authentication. It may be merely an indication that the desired authentication isn't yet achieved. This message contains a list of authentication methods that can continue and a partial success flag, which is set to TRUE if the previous request was successful but further authentication is needed, and is set to FALSE if the previous request failed. This implies that the number of `SSH_MSG_USERAUTH_REQUEST` and `SSH_MSG_USERAUTH_FAILURE` messages isn't necessarily the same for all authentication processes.

Eventually, when the authentication is complete, the server replies with a `SSH_MSG_USERAUTH_SUCCESS` message and starts the requested service.

■ Authentication methods

- “Public key” method

- ▶ Supported by all implementations
- ▶ Client signs using his private key
- ▶ Server verifies signature and client’s public key
- ▶ Issue:
 - ✓ Computationally expensive
 - ✓ Few clients have public/private key pairs

- “Password” method

- ▶ Supported by all implementations
- ▶ Most widely used today

- “Host based” method

- ▶ Support is optional
- ▶ Authentication based on user’s host
- ▶ Client sends signature generated using client’s private host key
- ▶ Server verifies signature en client’s public host key
- ▶ Similar to “public key”
 - ✓ Except for host authentication instead of user authentication

p. 28

The server may require one or more of the following authentication methods:

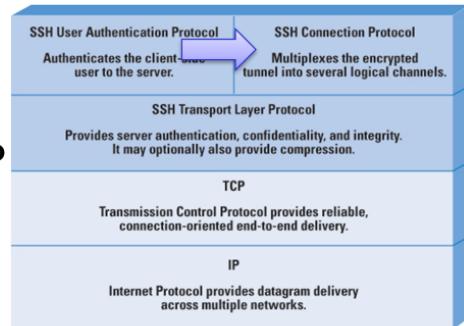
publickey: The details of this method depend on the public-key algorithm chosen. In essence, the client sends a message to the server that contains the client's public key, with the message signed by the client's private key. When the server receives this message, it checks to see whether the supplied key is acceptable for authentication and, if so, it checks to see whether the signature is correct. The SSH_MSG_USERAUTH_REQUEST may also contain a flag set to FALSE (instead of TRUE) and no signature. This allows the client to test whether this (computationally expensive) authentication method is accepted by the server. If it is, the server will respond with a SSH_MSG_USERAUTH_PK_OK message and the authentication can continue.

password: The client sends a message containing a plaintext password, which is protected by encryption by the Transport Layer Protocol. If the server answers with a SSH_MSG_USERAUTH_PASSWD_CHANGEREQ message, the client will change the password by sending a SSH_MSG_USERAUTH_REQUEST for the “password” method containing a TRUE (instead of FALSE) flag, the old password (in plaintext) and the new password (in plaintext).

Host based: Authentication is performed on the client's host rather than the client itself. Thus, a host that supports multiple clients would provide authentication for all its clients. This method works by having the client send a signature created with the private key of the client host. Thus, rather than directly verifying the user's identity, the SSH server verifies the identity of the client host—and then believes the host when it says the user has already authenticated on the client side.

■ SSH connection protocol

- Multiplexes secure tunnel provided by SSH transport layer and user authentication layer into several logical channels
- Logical channels can be used for various purposes



p. 29

The SSH Connection Protocol runs on top of the SSH Transport Layer Protocol and assumes that a secure authentication connection is in use. That secure authentication connection, referred to as a *tunnel*, is used by the Connection Protocol to multiplex a number of logical channels.

Confusingly, RFC 4254, "The Secure Shell (SSH) Connection Protocol," states that the Connection Protocol runs on top of the Transport Layer Protocol whilst the User Authentication Protocol. RFC 4251, "SSH Protocol Architecture," states that the Connection Protocol runs over the User Authentication Protocol. In fact, the Connection Protocol runs over the Transport Layer Protocol, but assumes that the User Authentication Protocol has been previously invoked.

■ Applications implemented as “channels”

- All channels are multiplexed into single encrypted tunnel (provided by SSH transport layer protocol)
- Channels identified by channel numbers at both ends of the connection
- Channel numbers may differ for same channel at client and server sides

■ Example channels

- Session (shell, file transfer, e-mail, system command, ...)
- X11-connections
- Local port forwarding (direct TCP/IP)
- Remote port forwarding (forwarded TCP/IP)

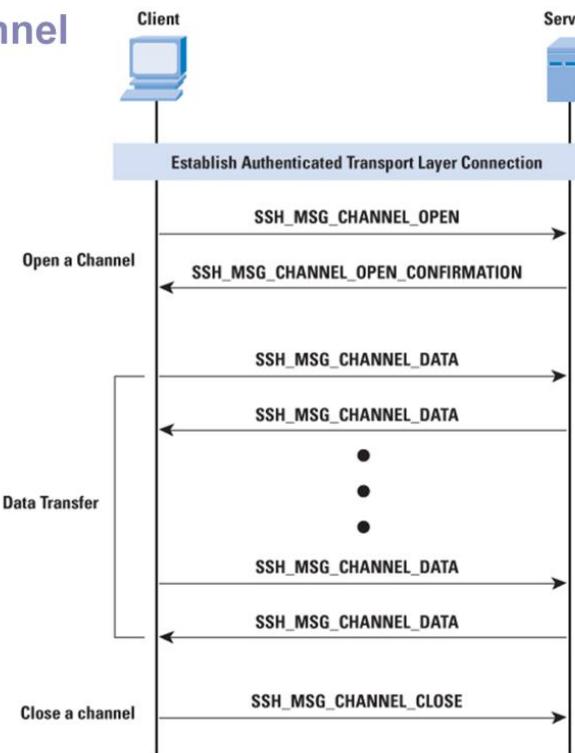
p. 30

All types of communication using SSH, such as a terminal session, are supported using separate channels. Either side may open a channel. For each channel, each side associates a unique channel number, which need not be the same on both ends. Channels are flow-controlled using a window mechanism. No data may be sent to a channel until a message is received to indicate that window space is available.

Four channel types are recognized in the SSH Connection Protocol specification:

- *session*: Session refers to the remote execution of a program. The program may be a shell, an application such as file transfer or e-mail, a system command, or some built-in subsystem. When a session channel is opened, subsequent requests are used to start the remote program.
- *x11*: This channel type refers to the X Window System, a computer software system and network protocol that provides a GUI for networked computers. X allows applications to run on a network server but be displayed on a desktop machine.
- *direct-tcpip*: This channel type is local port forwarding, as explained subsequently.
- *forwarded-tcpip*: This channel type is remote port forwarding, as explained subsequently.

■ Opening a channel



p. 31

The life of a channel progresses through three stages: opening a channel, data transfer, and closing a channel.

When either side wishes to open a new channel, it allocates a local number for the channel and then sends a message containing the channel type, sender channel, initial window size and maximum packet size. The *channel type* identifies the application for this channel, as described previously. The *sender channel* is the local channel number. The *initial window size* specifies how many bytes of channel data can be sent to the sender of this message without adjusting the window. The *maximum packet size* specifies the maximum size of an individual data packet that can be sent to the sender. For example, one might want to use smaller packets for interactive connections to get better interactive response on slow links.

If the remote side is able to open the channel, it returns a `SSH_MSG_CHANNEL_OPEN_CONFIRMATION` message, which includes the sender channel number, the recipient channel number, and window and packet size values for incoming traffic. Otherwise, the remote side returns a `SSH_MSG_CHANNEL_OPEN_FAILURE` message with a reason code indicating the reason for failure.

After a channel is open, *data transfer* is performed using a `SSH_MSG_CHANNEL_DATA` message, which includes the recipient channel number and a block of data. These messages, in both directions, may continue as long as the channel is open.

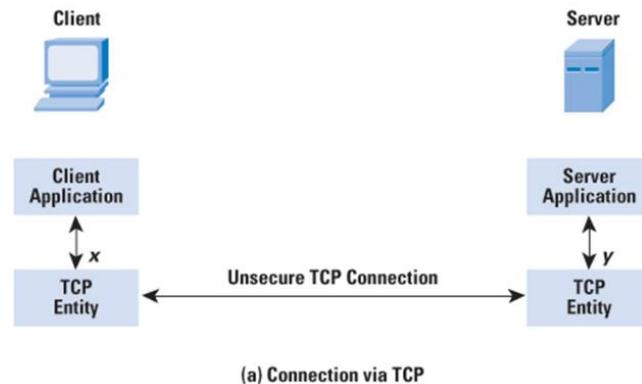
When either side wishes to close a channel, it sends a `SSH_MSG_CHANNEL_CLOSE` message, which includes the recipient channel number.

■ SSH port forwarding

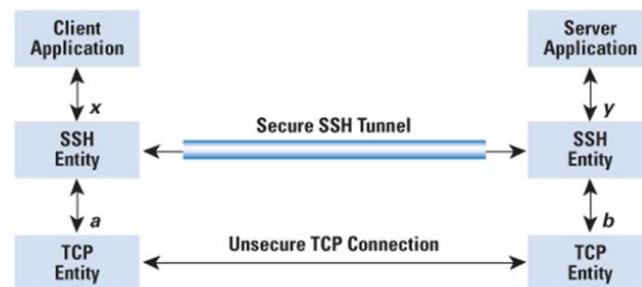
• Goals

1. Transform insecure TCP to secure SSH connections
 - ✓ “SSH tunnel”
2. Bypass network restrictions

• Local and remote port forwarding



(a) Connection via TCP



(b) Connection via SSH Tunnel

. 32

One of the most useful features of SSH is *port forwarding*. Port forwarding or port mapping is an application of network address translation (NAT) that redirects a communication request from one address and port number combination to another while the packets are traversing a network gateway. SSH port forwarding provides the ability to convert any insecure TCP connection into a secure SSH connection. It is also referred to as SSH tunneling. We need to know what a port is in this context. A port is an identifier of a user of TCP. So, any application that runs on top of TCP has a port number. Incoming TCP traffic is delivered to the appropriate application on the basis of the port number. For example, for the *Simple Mail Transfer Protocol* (SMTP), the server side generally listens on port 25, so that an incoming SMTP request uses TCP and addresses the data to destination port 25. TCP recognizes that this address is the SMTP server address and routes the data to the SMTP server application. The figure illustrates the basic concept behind port forwarding. We have a client application that is identified by port number x and a server application identified by port number y . At some point, the client application invokes the local TCP entity and requests a connection to the remote server on port y . The local TCP entity negotiates a TCP connection with the remote TCP entity, such that the connection links local port x to remote port y . To secure this connection, SSH is configured so that the SSH Transport Layer Protocol establishes a TCP connection between the SSH client and server entities with TCP port numbers a and b , respectively. A secure SSH tunnel is established over this TCP connection. Traffic from the client at port x is redirected to the local SSH entity and travels through the tunnel where the remote SSH entity delivers the data to the server application on port y . Traffic in the other direction is similarly redirected.

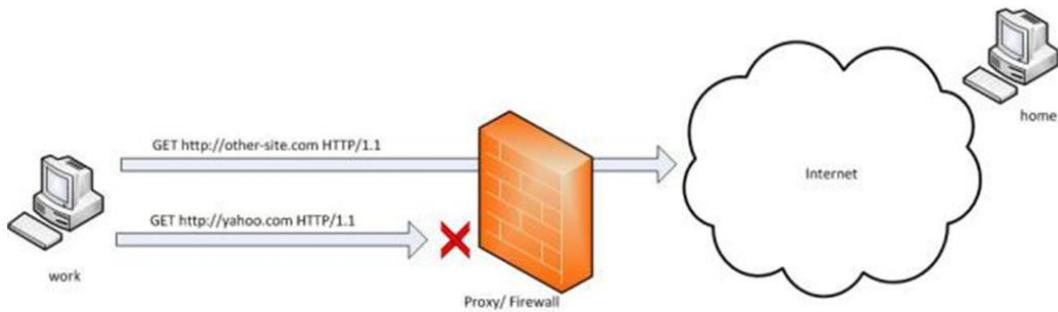
SSH supports two types of port forwarding: local forwarding and remote forwarding.

Local forwarding is used to **forward data securely from another client application running on the same computer** (using different port numbers) as the Secure Shell Client. It is frequently used to (i) transform an insecure TCP connection by a secure one, and/or (ii) to bypass network restrictions. Local port forwarding allows the client to set up a "hijacker" process. This process will intercept selected application-level traffic and redirect it from an unsecured TCP connection to a secure SSH tunnel. SSH is configured to listen on selected ports, grabs all traffic that uses the selected port and sends it through an SSH tunnel. On the other end, the SSH server sends the incoming traffic to the destination port dictated by the client application. Local port forwarding is the most common type of port forwarding.

With **remote forwarding**, the user's SSH client acts on the server's behalf. The client receives traffic with a given destination port number, places the traffic on the correct port, and sends it to the destination the user chooses. Remote port forwarding allows **other computers to access applications hosted on remote servers**. This form of port forwarding enables applications on the server side of a Secure Shell (SSH) connection to access services residing on the SSH's client side. With remote forwarding, the user's SSH client acts on the server's behalf. The client receives traffic with a given destination port number, places the traffic on the correct port, and sends it to the destination the user chooses. A typical example of remote forwarding follows: You wish to access a server at work from your home computer. Because the work server is behind a firewall, it will not accept an SSH request from your home computer. However, from work you can set up an SSH tunnel using remote forwarding.

■ Local Port Forwarding (outgoing tunnel)

- Example: create an outgoing tunnel passing a firewall

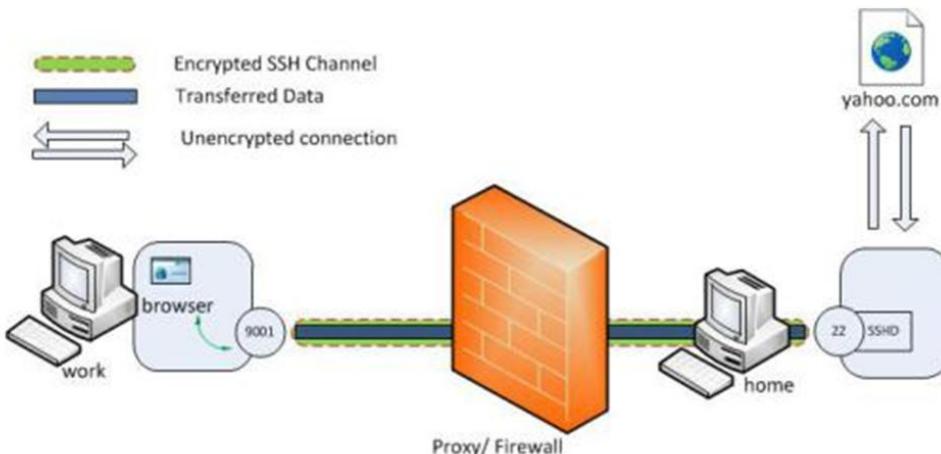


p. 33

Let's say that yahoo.com is being blocked using a proxy filter in the University. A SSH tunnel can be used to bypass this restriction.

■ Local Port Forwarding (outgoing tunnel)

- The remote host (server) acts as a gateway
 - ▶ Example: create an outgoing tunnel passing a firewall
- Command:
 - ▶ `ssh -L local_port:remote_host:remote_port login@servername`
 - ▶ `ssh -L 9001@yahoo.com:80 home`



p. 34

The syntax of the local port forwarding command is as follows (The 'L' switch indicates that a local port forward is need to be created):

- `ssh -L <local-port-to-listen>:<remote-host>:<remote-port> <gateway>`

It can be interpreted as: I want to securely connect to <remote host> on <remote port> and to this end I set up an SSH tunnel from <local port> at my current host to <gateway> (which will act as a gateway).

Let's name my machine at the university as 'work' and my home machine as 'home' ('home' needs to have a public IP for this to work). I am running an SSH server on my home machine. To create the SSH tunnel execute the following from the 'work' machine.

- `ssh -L 9001@yahoo.com:80 home`

Now the SSH client at 'work' will connect to the SSH server running at 'home' (usually running at port 22) binding port 9001 of 'work' to listen for local requests thereby creating a SSH tunnel between 'home' and 'work'. At the 'home' end it will create a connection to 'yahoo.com' at port 80. So 'work' doesn't need to know how to connect to yahoo.com. Only 'home' needs to worry about that. Now it is possible to browse to yahoo.com by visiting `http://localhost:9001` in the web browser at the 'work' computer. The 'home' computer will act as a gateway which would accept requests from 'work' machine and fetch data to tunnel it back. The connection from 'host' to 'yahoo.com' is only made when the browser makes the request (i.e. not already during the tunnel setup). The channel between 'work' and 'home' will be encrypted while the connection between 'home' and 'yahoo.com' will be unencrypted.

It is also possible to specify a port belonging to the 'home' computer itself instead of connecting to an external host. This option is typically not used if gateway functionality is required (i.e.: not for passing a firewall), but if an unsecure connection needs to be replaced by a secure connection. Typical use cases include using SSH to securely retrieve (port 110, POP) or transmit (port 25, SMTP) e-mails. For example, when setting up a VNC session between 'work' and 'home', the command line would be as follows.

- `ssh -L 5900:localhost:5900 home` (executed from 'work')

So what does localhost refer to? Is it the 'work' since the command line is executed from 'work'? Turns out that it is not: the localhost is considered relative to the gateway ('home' in this case), not the machine from where the tunnel is initiated. So this will make a connection to port 5900 of the 'home' computer where the VNC client would be listening in. As such, the command is equivalent to

- `ssh -L 5900:home:5900 home` (executed from 'work')

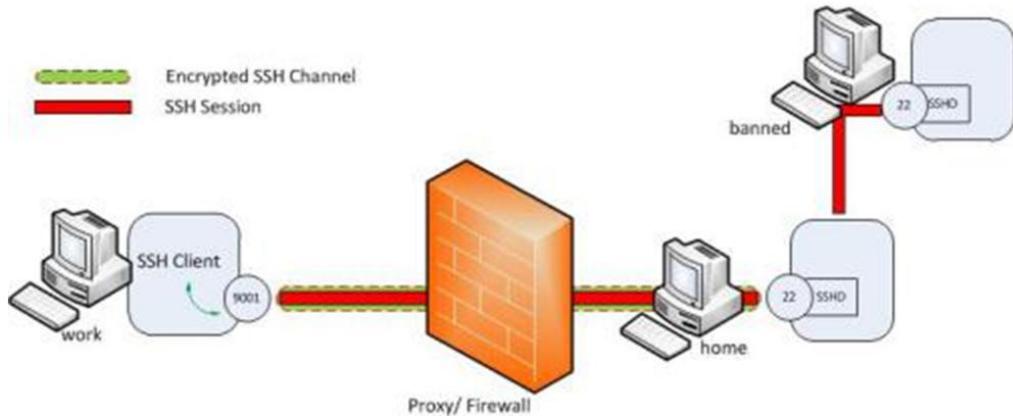
This last notation makes the fact that in this case the server ('work') does not act as a gateway more explicit.

■ Local Port Forwarding (outgoing tunnel)

• Example: tunnelling of SSH sessions

▶ `ssh -L 9001:banned:22 home`

▶ `ssh -p 9001 localhost`



p. 35

The created tunnel can be used to transfer any type of connection and is thus not limited to web browsing sessions. In addition to mail sessions, ftp sessions, etc., we can even tunnel SSH sessions. Let's assume there is another computer ('banned') to which we need to SSH from within University but the SSH access is being blocked. It is possible to tunnel a SSH session to this host using a local port forward.

As can be seen now the transferred data between 'work' and 'banned' are encrypted end to end. For this we need to create a local port forward as follows.

- `ssh -L 9001:banned:22 home` (executed from 'work')

Now 'home' acts as a gateway for tunneling SSH sessions to 'banned'.

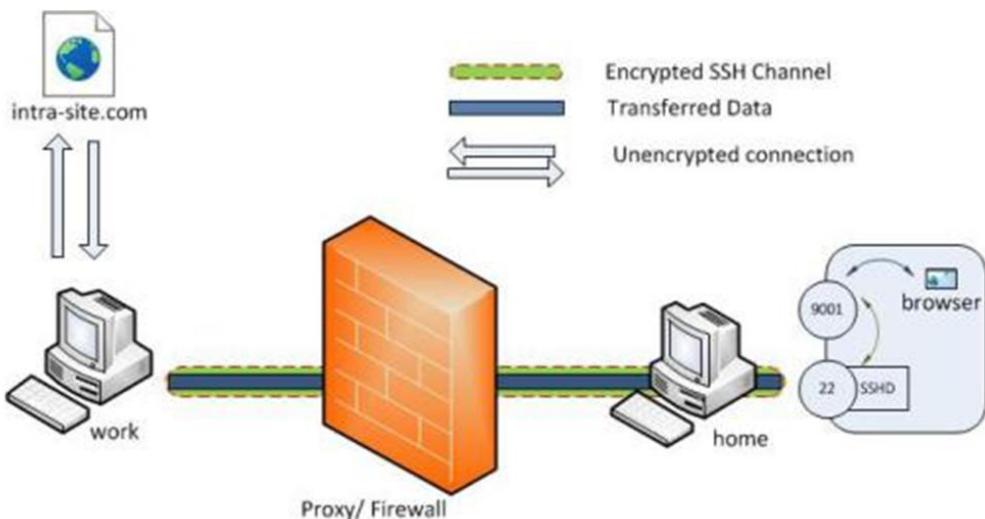
To connect over SSH with 'banned', we create an SSH session to local port 9001.

- `ssh -p 9001 localhost`

This new SSH session will automatically get tunneled to 'banned' via the 'home' gateway due to the previously configured port forwarding rules.

■ Remote Port Forwarding (incoming or reverse tunnel)

- The local host (client) acts as a gateway
 - ▶ Example: firewall blocking all traffic
- Command:
 - ▶ `ssh -R remote_port:local_host:local_port login@servername`
 - ▶ `ssh -R 9001:intra-site.com:80 home`



p. 36

The syntax of the remote port forwarding command is as follows (The 'R' switch indicates that a remote port forward is created):

- `ssh -R <server-port> :<remote-host>:<remote-port> <server host>`

The command can be interpreted as: I will act as a gateway to forward connections from <server port> on <server host> to <remote-port> on <remote-host> using a secure SSH tunnel.

Let's say we want to connect to an internal university website from home, but the university firewall is blocking all incoming traffic. How can we connect from 'home' to internal network so that we can browse the internal site? (A VPN setup is a good candidate here, but however for this example let's assume we don't have this facility). As discussed, local port forwarding creates an outgoing tunnel and as such is not useful in this case. However, we can use remote port forwarding (also referred to as SSH reverse tunneling) to create an incoming tunnel. As before, we will initiate the tunnel from the 'work' computer behind the firewall. This is possible since only incoming traffic is blocking and outgoing traffic is allowed. However instead of the earlier case the client will now be at the 'home' computer.

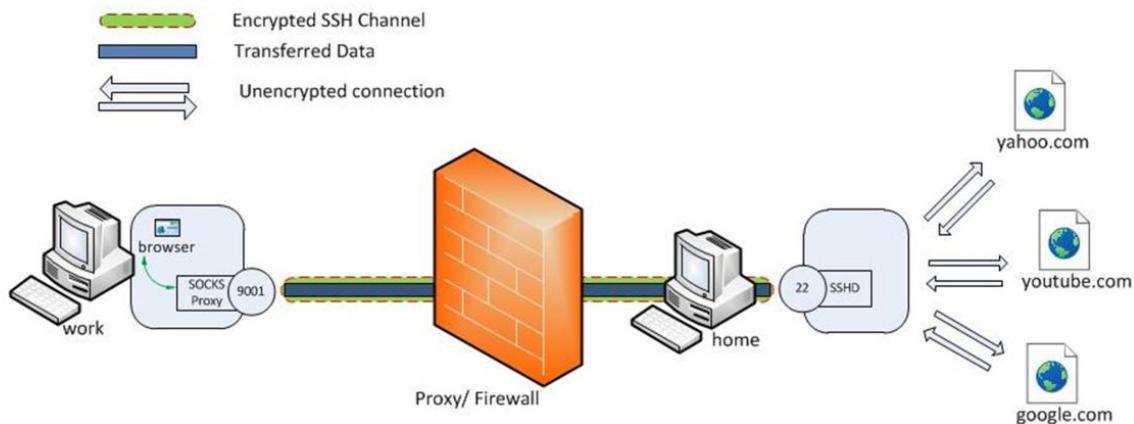
- `ssh -R 9001:intra-site.com:80 home` (executed from 'work', note the R instead of L)

Once executed the SSH client ('work') will connect to the SSH server (running at home) to create an SSH channel. Then the server will bind port 9001 on the 'home' machine to listen for incoming requests which would subsequently be routed through the created SSH channel between 'home' and 'work'. Now it's possible to browse the internal site by visiting `http://localhost:9001` in 'home' web browser. The 'work' SSH server will then create a connection to the intra-site and relay back the response to 'home' via the created SSH channel.

As such, the command is very similar to the local port forwarding command, but the SSH client (work) acts as a gateway for incoming traffic rather than the SSH server (home). The main advantage is that in this case the creation of the tunnel was initiated from the gateway, allowing it to bypass NAS or firewall restrictions.

■ Dynamic Port Forwarding

- **SOCKS proxy**
 - **Command**
- `ssh -D 9001 home`



p. 37

Although useful, local and remote port forwarding require the creation of yet another tunnel if you need to connect to another website. Dynamic port forwarding makes it possible to proxy traffic to any site using the SSH created channel.

Dynamic port forwarding allows to configure one local port for tunneling data to all remote destinations. However to utilize this the client application connecting to local port should send their traffic using the SOCKS protocol. At the client side of the tunnel a SOCKS proxy would be created and the application (eg. browser) uses the SOCKS protocol to specify where the traffic should be sent when it leaves the other end of the ssh tunnel.

- `ssh -D 9001 home` (executed from 'work')

Here SSH will create a SOCKS proxy listening in for connections at local port 9001 and upon receiving a request would route the traffic via SSH channel created between 'work' and 'home'. It is required to configure the browser to point to the SOCKS proxy at port 9001 at localhost.

■ Disable weak passwords

■ Only use SSH2

■ Restrict root access

■ Etc

■ Examples at

- <http://www.ibm.com/developerworks/aix/library/au-sshsecurity/>

38

Many good systems administrators are nervous about security. Here is a list of processes and configurations that you can use to tighten and enhance SSH security with regard to remote host access.

Restrict the root account to console access only:

```
# vi /etc/ssh/sshd_config
    PermitRootLogin no
```

Create private-public key pairs using a strong passphrase and password protection for the private key (never generate a password-less key pair or a password-less passphrase key-less login):

```
ssh-keygen -t rsa -b 4096 (Use a higher bit rate for the encryption for more security)
```

Only use SSH Protocol 2:

```
# vi /etc/ssh/sshd_config
    Protocol 2
```

Restrict the available interfaces that SSH will listen on and bind to:

```
# vi /etc/ssh/sshd_config
    ListenAddress 192.168.100.17
    ListenAddress 209.64.100.15
```

Set user policy to enforce strong passwords to protect against brute force, social engineering attempts, and dictionary attacks:

```
# < /dev/urandom tr -dc A-Za-z0-9_ | head -c8
oP0FNAUT[
```

Confine SFTP users to their own home directories by using Chroot SSHD:

```
# vi /etc/ssh/sshd_config ChrootDirectory /data01/home/%u
    X11Forwarding no
    AllowTcpForwarding no
```

Disable empty passwords:

```
# vi /etc/ssh/sshd_config
    PermitEmptyPasswords no
```

Always keep the SSH packages and required libraries up to date on patches:

```
# yum update openssh-server openssh openssh-clients -y
```

SSH supports numerous, diverse methods and techniques for authentication that you can enable or disable. Within the /etc/ssh/sshd_config file, you make these configurations changes by entering the keyword listed for the authentication method followed by yes or no. Here are some of the common configuration changes:

```
# RSAAuthentication yes
# PubkeyAuthentication yes
# RhostsRSAAuthentication no
# HostbasedAuthentication no
#     RhostsRSAAuthentication      and      HostbasedAuthentication      PasswordAuthentication      yes
ChallengeResponseAuthentication no
# KerberosAuthentication no GSSAPIAuthentication yes
```

■ SSH shortcomings

- Not designed for slow connections
 - ▶ lagging SSH console session.
- Not designed for connection drops
 - ▶ SSH state connection is lost
- SSH works over TCP
 - ▶ No IP address roaming supported.
- Not designed for large network latencies and round trip times

■ Alternative: MOSH (Mobile Shell)

- **Designed for mobile devices and unreliable connections**

- Network model
- Secure configuration of devices
 - SSH (**Secure Shell**)
- Exchanging keys
- Secure networking protocols
- Firewalls



Besides the lecturers' own material, many third party, often copyrighted, material is reused within this lecture (e.g. in the notes) under the 'fair use' approach, for sake of educational purpose only, and very limited edition. As a consequence, the current slide set presentation usage is restricted, and is falling under usual copyrights usage.

At the end of every lecture, appropriate references to used materials are included.

- This work contains content adapted from, amongst other, the following sources (in no particular order)
 - William Stallings, “**Cryptography and Network Security, principles and practices**”, 6th (international) edition, Prentice Hall, 2010;
 - Matt Bishop, “**Computer Security: Art and Science**”, Addison Wesley, Pearson Education, 2003, ISBN-13: 978-0-201-44099-7
 - Lecture slides: “Informatiebeveiliging”, Universiteit Gent, Eric Laermans & Thom Dhaene
 - Wikipedia (additional note page descriptions)
 - <https://crypto.stackexchange.com>
 - <http://www.ibm.com/developerworks/aix/library/au-sshsecurity/>
 - <https://chamibuddhika.wordpress.com/2012/03/21/ssh-tunnelling-explained/>



Network and Computer Security

Chapter 3 – Network and communication security

Prof. dr. ir. Eli De Poorter

- Network model
- Secure configuration of devices
 - SSH (Secure Shell)
- Exchanging keys
 - Out of band
 - Diffie-Hellman
 - Asymmetric encryption
 - Trusted third party
 - ▶ Key Distribution Centre (KDC)
 - ▶ Public Key Infrastructure (PKI)
- Secure networking protocols
 - Transport layer: TLS & SSL
 - Network layer: IPSec & VPN
 - Data link layer: WEB & WPA
- Firewalls
 - Packet filter
 - Circuit-level gateway
 - Application-level gateway (proxy)

- Network model
- Secure configuration of devices
- **Exchanging keys**
 - Out of band
 - Diffie-Hellman
 - Asymmetric encryption
 - Trusted third party
 - ▶ Key Distribution Centre (KDC)
 - ▶ Public Key Infrastructure (PKI)
- Secure networking protocols
- Firewalls

- Symmetric encryption is much more efficient
 - But how to distribute the shared key securely?
- Goal
 - Agree on a key that two parties can use for a symmetric encryption, in such a way that an eavesdropper cannot obtain the key
- Methods
 - Out of band
 - Using Diffie-Hellman
 - Using asymmetric encryption
 - Using trusted third party
 - ▶ Public-key infrastructure (PKI)
 - ▶ Kerberos
 - ▶ Etc.

4

Given two parties A and B, multiple **key distribution** alternatives exist

1. A can select key and physically deliver to B
2. third party can select & deliver key to A & B
3. if A & B have communicated previously can use previous key to encrypt a new key
4. if A & B have secure communications with a third party C, C can relay key between A & B

Physical delivery (1 & 2) is simplest - but only applicable when there is personal contact between recipient and key issuer. This is fine for link encryption where devices & keys occur in pairs, but does not scale as number of parties who wish to communicate grows. 3 is mostly based on 1 or 2 occurring first.

A third party, whom all parties trust, can be used as a **trusted intermediary** to mediate the establishment of secure communications between them (4). The clients must trust the intermediary not to abuse the knowledge of all session keys. As the number of parties grows, variants of 4 are the only practical solution to the huge growth in number of keys potentially needed.

- Network model
- Secure configuration of devices
- Exchanging keys
 - Out of band
 - Diffie-Hellman
 - Asymmetric encryption
 - Trusted third party
 - ▶ Key Distribution Centre (KDC)
 - ▶ Public Key Infrastructure (PKI)
- Secure networking protocols
- Firewalls

- **Exchanging public keys “out-of-band”**
 - E.g. non-electronically
 - Signing public keys of other entities
 - Accepting public keys that were digitally signed by sufficiently trusted entities
- **OK for acquaintances**
- **Less suitable for large organisations**

p. 6

This is more or less the basic principle behind the system used in PGP (Pretty Good Privacy) (see later).

- Network model
- Secure configuration of devices
- Exchanging keys
 - Out of band
 - Diffie-Hellman
 - Asymmetric encryption
 - Trusted third party
 - ▶ Key Distribution Centre (KDC)
 - ▶ Public Key Infrastructure (PKI)
- Secure networking protocols
- Firewalls

■ $g = \text{primitive root mod } p$

- **Example: the number 3 is a primitive root modulo 7**
- **$g = \text{generator of the multiplicative group of integers modulo } n$**

$$\begin{aligned}3^1 &= 3 = 3^0 \times 3 \equiv 1 \times 3 = 3 \equiv 3 \pmod{7} \\3^2 &= 9 = 3^1 \times 3 \equiv 3 \times 3 = 9 \equiv 2 \pmod{7} \\3^3 &= 27 = 3^2 \times 3 \equiv 2 \times 3 = 6 \equiv 6 \pmod{7} \\3^4 &= 81 = 3^3 \times 3 \equiv 6 \times 3 = 18 \equiv 4 \pmod{7} \\3^5 &= 243 = 3^4 \times 3 \equiv 4 \times 3 = 12 \equiv 5 \pmod{7} \\3^6 &= 729 = 3^5 \times 3 \equiv 5 \times 3 = 15 \equiv 1 \pmod{7}\end{aligned}$$

8

In modular arithmetic a number g is a primitive root modulo n if for every integer a coprime to n , there is an integer k such that $g^k \equiv a \pmod{n}$. In other words, g is a generator of the multiplicative group of integers modulo n .

- How can two parties agree on a secret value when all of their messages might be overheard by an eavesdropper?
 - The Diffie-Hellman algorithm accomplishes this, and is still widely used.
 - First practical method for establishing a shared secret over an unsecured communication channel (1976)

- Concept

- Based on the discrete logarithm problem
 - ▶ No efficient general method for computing discrete logarithms on conventional computers is known
 - ▶ Exploits the fact that $((g^b \text{ mod } p)^a \text{ mod } p) = ((g^a \text{ mod } p)^b \text{ mod } p)$
 - ✓ if p = prime number and g = primitive root mod p
- Remember: also the factoring integer problem is challenging and used for the generation of public keys

9

The Diffie-Hellman key agreement protocol (1976) was the first practical method for establishing a shared secret over an unsecured communication channel. It is based on discrete logarithms. Discrete logarithms are fundamental to a number of public-key algorithms, including Diffie-Hellman key exchange and the digital signature algorithm (DSA). Discrete logs (or indices) share the properties of normal logarithms, and are quite useful. The logarithm of a number is defined to be the power to which some positive base (except 1) must be raised in order to equal that number. If working with modulo arithmetic, and the base is a primitive root, then an integral discrete logarithm exists for any residue. However whilst exponentiation is relatively easy, finding discrete logs is not, in fact is as hard as factoring a number. This is an example of a problem that is "easy" one way (raising a number to a power), but "hard" the other (finding what power a number is raised to giving the desired answer). Problems with this type of asymmetry are very rare, but are of critical usefulness in modern cryptography.

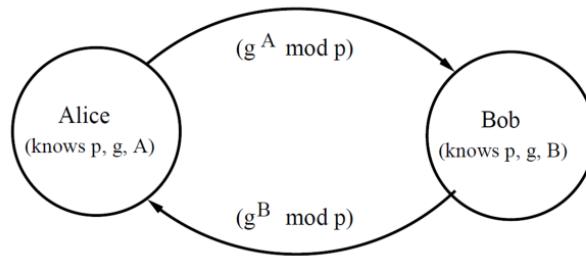
While computing discrete logarithms and factoring integers are distinct problems, they share some properties:

- both problems are difficult (no efficient algorithms are known for non-quantum computers),
- for both problems efficient algorithms on quantum computers are known,
- algorithms from one problem are often adapted to the other, and
- the difficulty of both problems has been used to construct various cryptographic systems.



- Both users agree on global parameters:
 - large prime integer or polynomial p
 - g being a primitive root mod p
- each user (eg. A) generates their key
 - chooses a secret key (number): $a < p$
 - compute their public key: $y_A = g^a \text{ mod } p$
- each user makes public that key y_A

In the Diffie-Hellman key exchange algorithm, there are two publicly known numbers: a prime number p and an integer “ a ” that is a primitive root of p . The prime p and primitive root a can be common to all using some instance of the D-H scheme. Note that the primitive root a is a number whose powers successively generate all the elements mod p . Users Alice and Bob choose random secrets x 's, and then "protect" them using exponentiation to create their public y 's. For an attacker monitoring the exchange of the y 's to recover either of the x 's, they'd need to solve the discrete logarithm problem, which is hard.

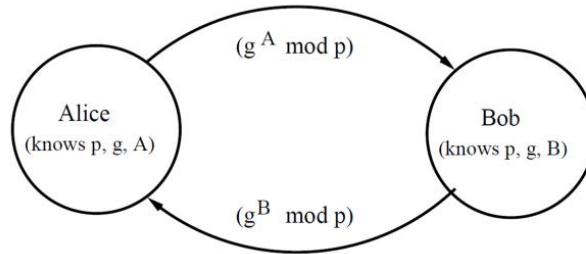


■ Steps in the algorithm:

1. **Alice and Bob agree on a prime number p and a base g**
 - ▶ These do not need to be kept secret
2. **Alice chooses a secret number a , and sends $y_A = g^a \bmod p$.**
3. **Bob chooses a secret number b , and sends $y_B = g^b \bmod p$.**
4. **Alice computes $(y_B^a \bmod p)$.**
5. **Bob computes $(y_A^b \bmod p)$.**

Both Alice and Bob can use this number as their key.

Notice that p and g need not be protected.



■ Example

- Alice and Bob agree on $p = 23$ and $g = 5$.
- Alice chooses $a = 6$
- Bob chooses $b = 15$.

What are the public keys?

What is the shared key?

Clearly, much larger values of a , b , and p are required.

An eavesdropper cannot discover this value even if she knows p and g and can obtain each of the messages.

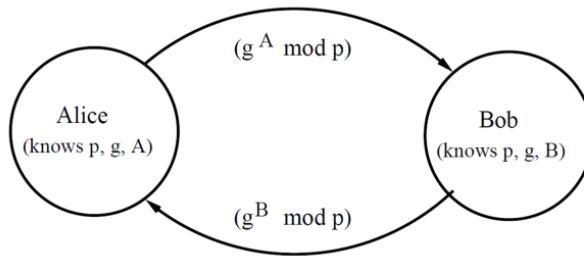
12

Alice chooses $a = 6$ and sends $5^6 \bmod 23 = 8$.

Bob chooses $b = 15$ and sends $5^{15} \bmod 23 = 19$.

Alice computes $19^6 \bmod 23 = 2$.

Bob computes $8^{15} \bmod 23 = 2$.



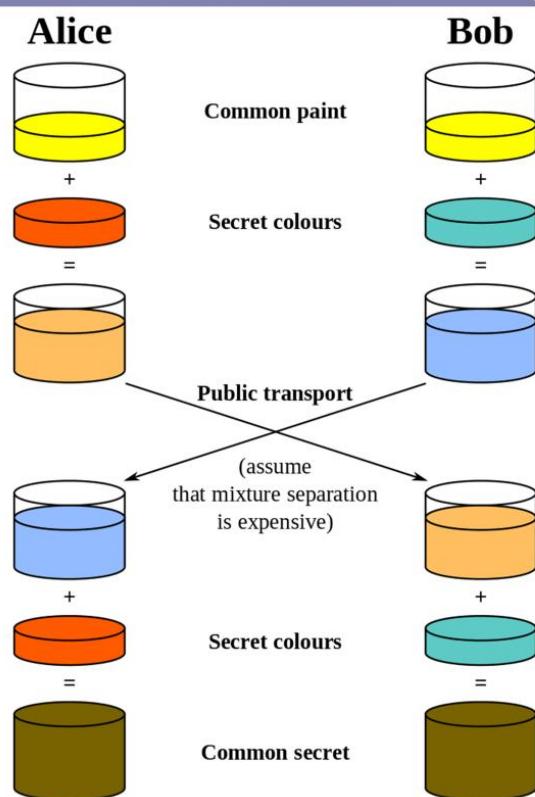
■ Complexity example

- **p is a prime of around 300 digits**
- **a and b at least 100 digits each.**

■ Discovering the shared secret given g , p , $g^a \bmod p$ and $g^b \bmod p$ would take longer than the lifetime of the universe, using the best known algorithm.

- **This is called the discrete logarithm problem**

- Discrete logarithm problem: illustration



■ Denial-of-Service attacks

● When basic scheme is used

- ▶ After receiving attacker's public key the victim will...
 - ✓ ...compute the public key and send it to the attacker
 - » Who may have given a false address to that purpose
 - ✓ ...compute the session key (in each configuration)

● Computation intensive operations

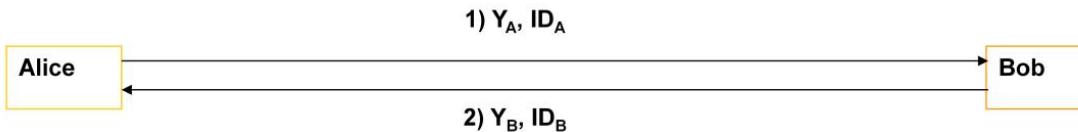
- ▶ May be (ab)used to paralyse a server
- ▶ Prior (partial) authentication needed to avoid needless heavy computations

15

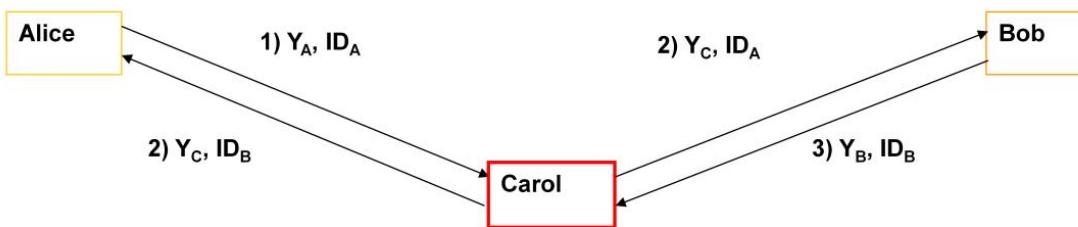
This type of attack is less severe against targets that use *fixed Diffie Helman* (see later), a scheme in which no new keys need to be computed for each session. To prevent this attack from happening, *ephemeral Diffie Helman* (see later) includes a separate prior authentication mechanism that is less computationally intensive.

■ “Man-in-the-Middle” attacks

- Normal operation



- Attack scheme



16

After the attack Alice will be communicating with Carol using a secret key derived from Y_A and Y_C , and Bob will be communicating with Carol using a secret key derived from Y_B and Y_C , while Alice and Bob falsely assume they're communicating with each other using a mutually agreed upon secret key. Carol will intercept the encrypted traffic between Alice and Bob, decrypt it, and re-encrypt it, without Alice or Bob noticing.

This attack is only possible absent any authentication of the public keys (Y_A , Y_B , and Y_C). This typically is an issue with anonymous or ephemeral DH when no additional authentication mechanism is used. When fixed DH is used it is possible to link the (fixed) key pairs to the communicating entities (see later certificates). Authenticated DH also averts this attack, unless the (fixed) shared secret key is compromised.

■ Many possible variants

- Diffie-Hellman using ECC

- ▶ Better security

- Fixed DH

- ▶ Each entity has a fixed private (X_A and X_B) and public key (Y_A and Y_B)
 - ▶ Public parameters are signed by certification authority (CA)
 - ▶ Fixed secret key for each pair of entities
 - ✓ OK if usage of secret key is limited

- Anonymous DH

- ▶ No built-in authentication mechanism
 - ✓ No certainty about who owns public key
 - ✓ Useful when 1 entity has no key pair
 - ▶ Purely for exchanging (session) key
 - ✓ Simple but vulnerable

- Ephemeral DH

- ▶ Private keys generated for each session
 - ▶ Different secret session key for each session
 - ▶ Authentication using different mechanism
 - ✓ RSA, DSA, etc.
 - ▶ Perfect Forward Secrecy

17

Elliptic curve Diffie–Hellman (ECDH) is a variant of the Diffie–Hellman protocol using elliptic curve cryptography.

Fixed Diffie–Hellman embeds the server's public parameter in the certificate, and the CA then signs the certificate. That is, the certificate contains the Diffie–Hellman public-key parameters, and those parameters never change.

Anonymous Diffie–Hellman uses Diffie–Hellman, but without authentication. Because the keys used in the exchange are not authenticated, the protocol is susceptible to Man-in-the-Middle attacks.

Ephemeral Diffie–Hellman uses temporary, public keys. Each instance or run of the protocol uses a different public key. The authenticity of the server's temporary key can be verified by checking the signature on the key. Because the public keys are temporary, a compromise of the server's long term signing key does not jeopardize the privacy of past sessions. This is known as Perfect Forward Secrecy (PFS).

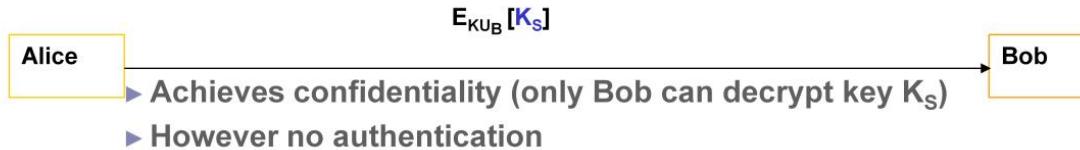
■ Challenge remains:

- To avoid man in the middle attacks, authentication is used.
- But how to do this securely?

- Network model
- Secure configuration of devices
- Exchanging keys
 - Out of band
 - Diffie-Hellman
 - **Asymmetric encryption**
 - Trusted third party
 - ▶ Key Distribution Centre (KDC)
 - ▶ Public Key Infrastructure (PKI)
- Secure networking protocols
- Firewalls

■ Using asymmetric encryption

- Basic scenario



- Improved version



p. 20

An extremely simple scheme was put forward by Merkle in 1979.

- A generates a new temporary public key pair
- A sends B the public key and their identity
- B generates a session key K sends it to A encrypted using the supplied public key
- A decrypts the session key and both use the session key

However, this approach is insecure against an adversary who can intercept messages and then either relay the intercepted message or substitute another message. The opponent can thereby impersonate both halves of protocol (i.e. a man-in-the-middle attack).



■ **Distribution of public keys can be considered using one of:**

- **public announcement**
- **publicly available directory**
- **public-key authority**
- **public-key certificates**

Several techniques have been proposed for the distribution of public keys, which can mostly be grouped into the categories shown.

Each of them has several advantages and disadvantages in terms of privacy, scalability, complexity, etc.

■ Public Announcement

- **users distribute public keys to recipients or broadcast to community at large**
 - ▶ eg. append PGP keys to email messages or post to news groups or email list
- **major weakness is forgery**
 - ▶ anyone can create a key claiming to be someone else and broadcast it
 - ▶ until forgery is discovered can masquerade as claimed user

The point of public-key encryption is that the public key is public, hence any participant can send his or her public key to any other participant, or broadcast the key to the community at large. Its major weakness is forgery, anyone can create a key claiming to be someone else and broadcast it, and until the forgery is discovered they can masquerade as the claimed user.

■ Publicly Available Directory

- **registering keys with a managed public directory**
- **directory must be trusted with properties:**
 - ▶ contains {name,public-key} entries
 - ▶ participants register securely with directory
 - ▶ participants can replace key at any time
 - ▶ directory is periodically published
 - ▶ directory can be accessed electronically
- **still vulnerable to tampering or forgery**

A greater degree of security can be achieved by maintaining a publicly available dynamic directory of public keys. Maintenance and distribution of the public directory would have to be the responsibility of some trusted entity or organization. This scheme is clearly more secure than individual public announcements but still has vulnerabilities to tampering or forgery.

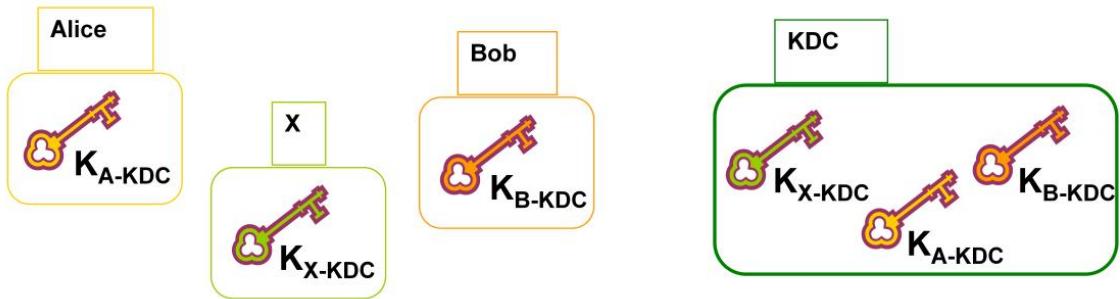
- Network model
- Secure configuration of devices
- Exchanging keys
 - Out of band
 - Diffie-Hellman
 - Asymmetric encryption
 - Trusted third party
 - ▶ Key Distribution Centre (KDC)
 - ▶ Public Key Infrastructure (PKI)
- Secure networking protocols
- Firewalls

■ Key Distribution Centre (KDC) or Public-Key Authority

- improve security by tightening control over distribution of keys from directory
- has properties of directory
- and requires users to know shared key for the directory
- then users interact with directory to obtain any desired key securely
 - ▶ does require real-time access to directory when keys are needed

Stronger security for public-key distribution can be achieved by providing tighter control over the distribution of keys from the directory.

- Each user owns secret key allowing him to communicate with key distribution centre (KDC)



■ Typically a hierarchy of keys

- **session key**

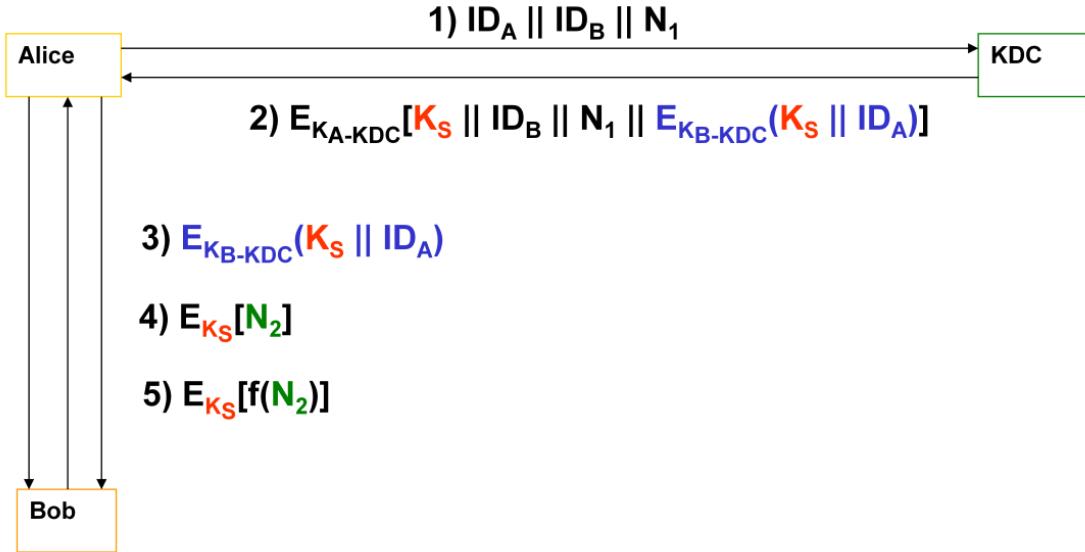
- ▶ temporary key
- ▶ used for encryption of data between users
- ▶ for one logical session then discarded

- **master key**

- ▶ used to encrypt session keys
- ▶ shared by user & key distribution center

The use of a key distribution center is based on the use of a hierarchy of keys. At a minimum, two levels of keys are used: a session key, used for the duration of a logical connection; and a master key shared by the key distribution center and an end system or user and used to encrypt the session key.

For communication among entities within the same local domain, the local KDC is responsible for key distribution. To balance security & effort, a new session key should be used for each new connection-oriented session. A new session key is used for a certain fixed period only or for a certain number of transactions. An automated key distribution approach provides the flexibility and dynamic characteristics needed to allow a number of terminal users to access a number of hosts and for the hosts to exchange data with each other, provided they trust the system to act on their behalf. The use of a key distribution center imposes the requirement that the KDC be trusted and be protected from subversion.



p. 28

N_1 : nonce, which is used only once.

ID_A , ID_B : identities of Alice, resp. Bob

- 1) Alice asks KDC to initiate a communication with Bob and sends “nonce” N_1
- 2) KDC answers with encrypted (using Alice's key, K_{A-KDC} , which authenticates KDC w.r.t. Alice and achieves confidentiality) message consisting of the session key, Bob's identity, “nonce” N_1 (excluding replay), encrypted (using Bob's key, K_{B-KDC}) message (session key and Alice's identity); only Alice can decrypt this message and thus decrypt the session key
- 3) Alice sends session key and her own identity, encrypted with Bob's key (as received from KDC), after which Bob (and nobody else) can also decrypt the session key
- 4) Bob answers with second “nonce” N_2 , encrypted with session key, authenticating himself w.r.t. Alice (by his knowledge of K_s)
- 5) Alice answers with processed (e.g. adding 1 to N_2) “nonce” N_2 , encrypted with session key, authenticating herself w.r.t. Bob (using challenge-response mechanism)

■ Needham-Schroeder protocol

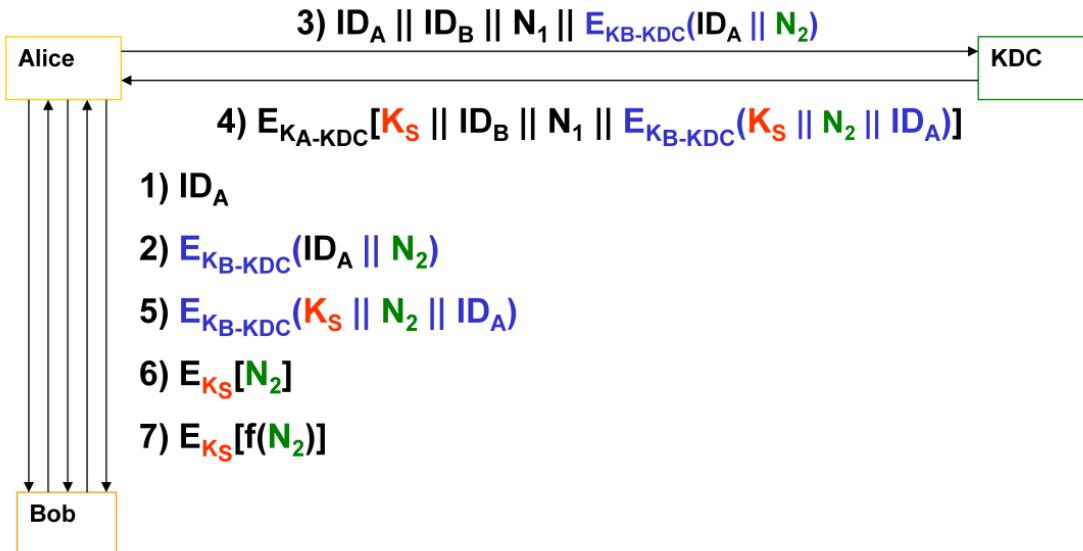
- Security depends on:

- ▶ Secret key K_{A-KDC}
- ▶ Secret key K_{B-KDC}
- ▶ Secret session key K_s

- ✓ Even after session has expired

■ Needham-Schroeder protocol

- **Attack against Bob using compromised K_S**
 - ▶ Carol replays step 3 using compromised K_S (only requires eavesdropping upon earlier communication)
 - ▶ Carol can answer Bob's challenge using compromised key K_S
 - ▶ Bob believes he's communicating with Alice, while he's in fact communicating with Carol
- **OBS. 1: this kind of attack is very unlikely to be successful (recovering K_S is very hard)**
- **OBS. 2: improved versions of this protocol exist**



p. 31

N_1, N_2 : nonces, which are used only once.

ID_A, ID_B : identities of Alice, resp. Bob

- 1) Alice informs Bob about communication
- 2) Bob answers with encrypted “nonce” (only readable to Bob and KDC)
- 3) Alice asks KDC to initiate a communication with Bob and sends “nonce” N_1 together with encrypted nonce N_2
- 4) KDC answers with encrypted (using Alice's key, K_{A-KDC} , which authenticates KDC w.r.t. Alice and achieves confidentiality) message consisting of the session key, Bob's identity, “nonce” N_1 (excluding replay), encrypted (using Bob's key, K_{B-KDC}) message (session key, “nonce” N_2 and Alice's identity); only Alice can decrypt this message and thus decrypt the session key
- 5) Alice sends session key, N_2 , and own identity, encrypted with Bob's key (as received from KDC), after which Bob (and nobody else) can also decrypt the session key (“nonce” N_2 is a guarantee for the freshness of key K_S)
- 6) Bob answers with second “nonce” N_2 , encrypted with session key, authenticating himself w.r.t. Alice (by his knowledge of K_S)
- 7) Alice answers with processed (e.g. adding 1 to N_2) “nonce” N_2 , encrypted with session key, authenticating herself w.r.t. Bob (using challenge-response mechanism)

■ Advantages

- **KDC generates a new shared key for each communication session between A and B**
 - ▶ Less risk on compromised shared keys, no reuse

A typical operation with a KDC involves a request from a user to use some service. The KDC will use cryptographic techniques to authenticate requesting users as themselves. It will also check whether an individual user has the right to access the service requested. If the authenticated user meets all prescribed conditions, the KDC can issue a ticket permitting access.

KDCs mostly operate with symmetric encryption.

In most (but not all) cases the KDC shares a key with each of all the other parties.

The KDC produces a ticket based on a server key.

The client receives the ticket and submits it to the appropriate server.

The server can verify the submitted ticket and grant access to the user submitting it.

Security systems using KDCs include Kerberos. (Actually, Kerberos partitions KDC functionality between two different agents: the AS (Authentication Server) and the TGS (Ticket Granting Service).)

■ Cerberus

- Greek methodology
- Fitting name for a KDC
 - ▶ 3-headed hellhound
 - ▶ Guardian of the underworld



33

- **User - password based authentication based on late-70's Needham -Schroeder algorithms.**
 - Kerberos Authentication Server aka KDC (Key Distribution Center) shares long-term secret (password) with each authorized user.
 - User logs in and established a short term session key with the AS which can be used to establish his identity with other entities, e.g. file system, other hosts or services each of which trusts the authority server.
- **The authorization mechanism needs to be integrated with the each function, e.g. file access, login, telnet, ftp, ...**
- **The central server is a single point of vulnerability to attack and failure.**
- **Been in use for 20 years. We are now at version 5.**

A typical operation with a KDC involves a request from a user to use some service. The KDC will use cryptographic techniques to authenticate requesting users as themselves. It will also check whether an individual user has the right to access the service requested. If the authenticated user meets all prescribed conditions, the KDC can issue a ticket permitting access.

KDCs mostly operate with symmetric encryption.

In most (but not all) cases the KDC shares a key with each of all the other parties.

The KDC produces a ticket based on a server key.

The client receives the ticket and submits it to the appropriate server.

The server can verify the submitted ticket and grant access to the user submitting it.

Security systems using KDCs include Kerberos. (Actually, Kerberos partitions KDC functionality between two different agents: the AS (Authentication Server) and the TGS (Ticket Granting Service).)

- Network model
- Secure configuration of devices
- Exchanging keys
 - Out of band
 - Diffie-Hellman
 - Asymmetric encryption
 - Trusted third party
 - ▶ Key Distribution Centre (KDC)
 - ▶ Public Key Infrastructure (PKI)
- Secure networking protocols
- Firewalls

■ Public key management

- **Issue:**

- ▶ How does one know who owns some public key?
- ▶ Building a trust relationship

- **Solution:**

- ▶ Simple system
- ▶ PKI (“Public Key Infrastructure”)

- Certificates allow key exchange without real-time access to public-key authority
- a certificate binds identity to public key
 - usually with other info such as period of validity, rights of use etc
- with all contents signed by a trusted Public-Key or Certificate Authority (CA)
- can be verified by anyone who knows the public-key authorities public-key

A further improvement is to use certificates, which can be used to exchange keys without contacting a public-key authority, in a way that is as reliable as if the keys were obtained directly from a public-key authority. A certificate binds an **identity** to **public key**, with all contents **signed** by a trusted Public-Key or Certificate Authority (CA). This can be verified by anyone who knows the public-key authorities public-key.

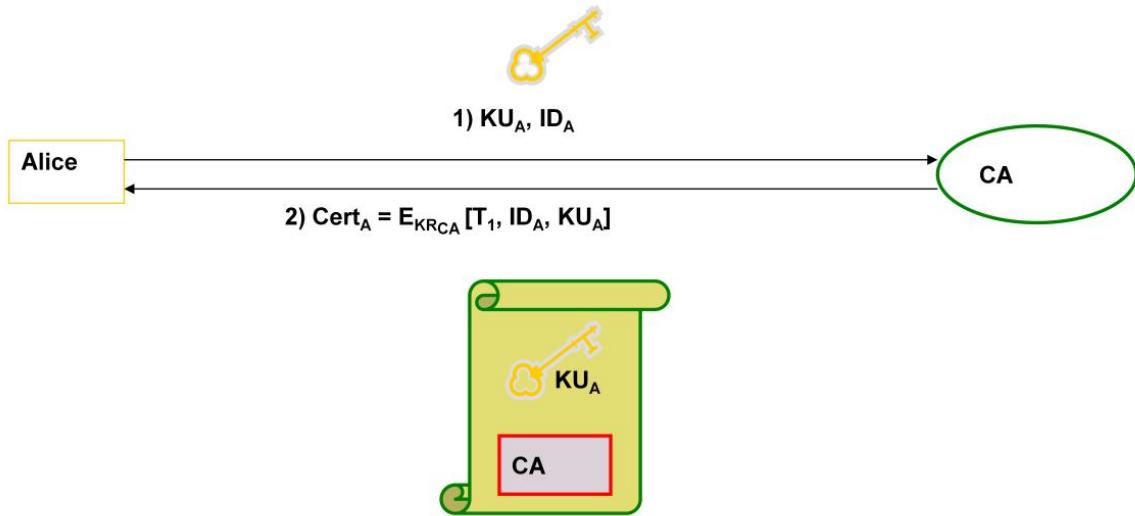
■ Using certification authority (CA)

- **Guarantees identity of the user of the public key**
 - ▶ Entity registers key at CA, where entity proves its identity
 - ▶ CA verifies identity and delivers certificate binding this identity to the public key
 - ▶ Entity can use certificate (with CA's guarantee) to prove its ownership of the public key
- **For more about certificates, see also later (X.509)**

p. 38

One scheme has become universally accepted for formatting public-key certificates: the X.509 standard. X.509 certificates are used in most network security applications, including IP security, secure sockets layer (SSL), secure electronic transactions (SET), and S/MIME.

■ Certificates

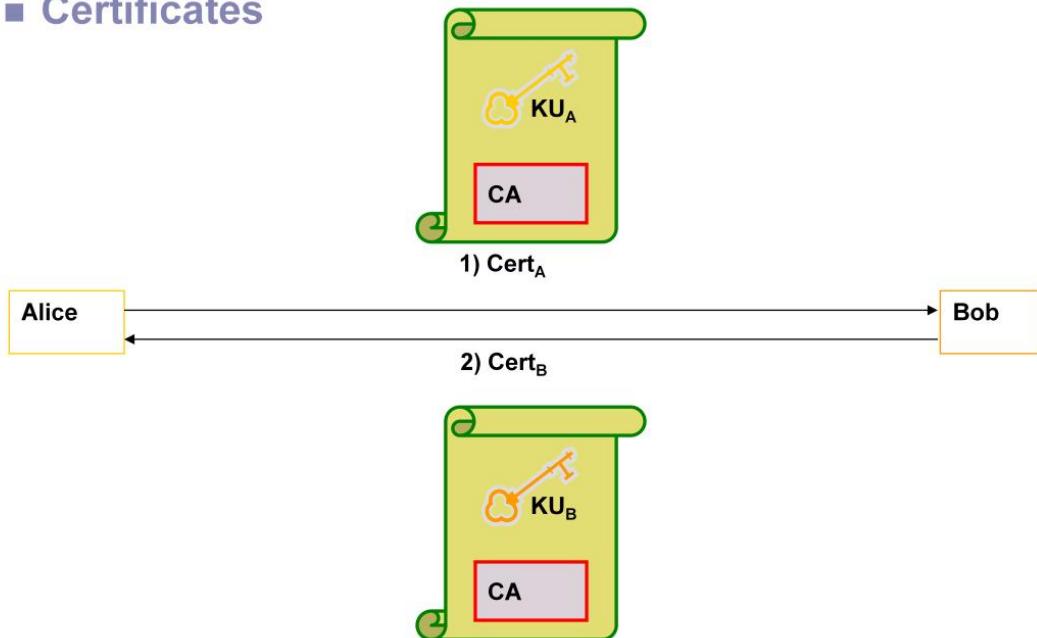


p. 39

ID_A : Alice's identity

- Alice registers public key at CA and proves her identity
- CA creates a certificate binding Alice's public to her identity, together with some time value (validity of the certificate), using a digital signature. Certificate can be verified using the public key of the CA (KU_{CA}).

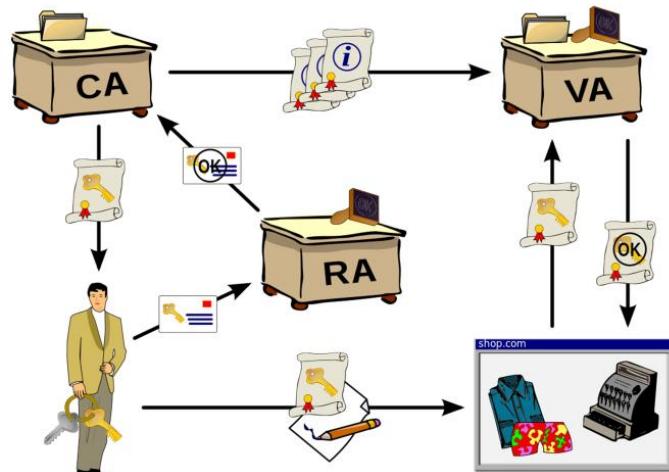
■ Certificates



p. 40

- 1) Alice sends certificate she has obtained from CA to Bob, allowing Bob (using CA's public key) to verify the public key Alice uses really belongs to Alice
- 2) Bob sends his certificate back to Alice for verification

- CA – Certification Authority
- RA – Registration Authority
- VA – third-party validation authority

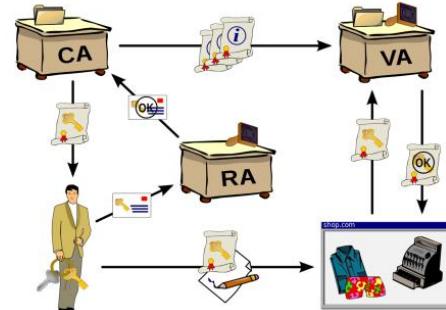


41

In cryptography, a PKI is an arrangement that binds public keys with respective user identities by means of a certificate authority (CA). The user identity must be unique within each CA domain. The third-party validation authority (VA) can provide this information on behalf of the CA. The binding is established through the registration and issuance process. Depending on the assurance level of the binding, this may be carried out by software at a CA or under human supervision. The PKI role that assures this binding is called the registration authority (RA). The RA is responsible for accepting requests for digital certificates and authenticating the person or organization making the request. In a Microsoft PKI, a registration authority is usually called a subordinate CA.

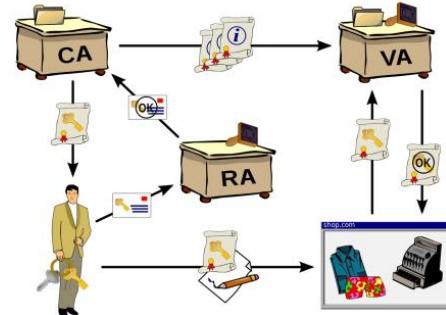
■ CA – Certification Authority

- Issuer/Signer of the certificate
 - ▶ Binds public key with identity+attributes
- Provider
 - ▶ Enterprise CA
 - ▶ Individual as CA (PGP)
 - ✓ Web of trust
 - ▶ “Global” or “Universal” CAs
 - ✓ VeriSign, Equifax, Entrust, CyberTrust, Identrus, ...
- Trust is the key word



■ RA – Registration Authority

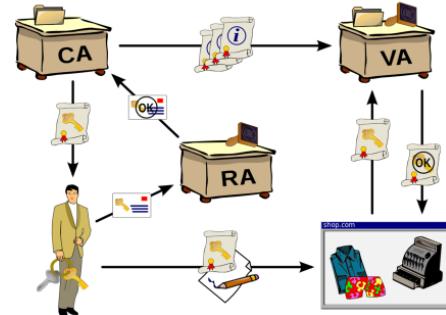
- Also called LRA – Local RA
- Goal: off-load some work of CA to LRAs
- Support all or some of:
 - ▶ Identification
 - ▶ User key generation/distribution
 - ✓ passwords/shared secrets and/or public/private keys
 - ▶ Interface to CA
 - ▶ Key/certificate management
 - ✓ Revocation initiation
 - ✓ Key recovery



■ VA – third-party validation authority

- The third-party validation authority (VA) can provide this information on behalf of the CA.

▶ Why is this useful?



■ Using certification authority (CA)

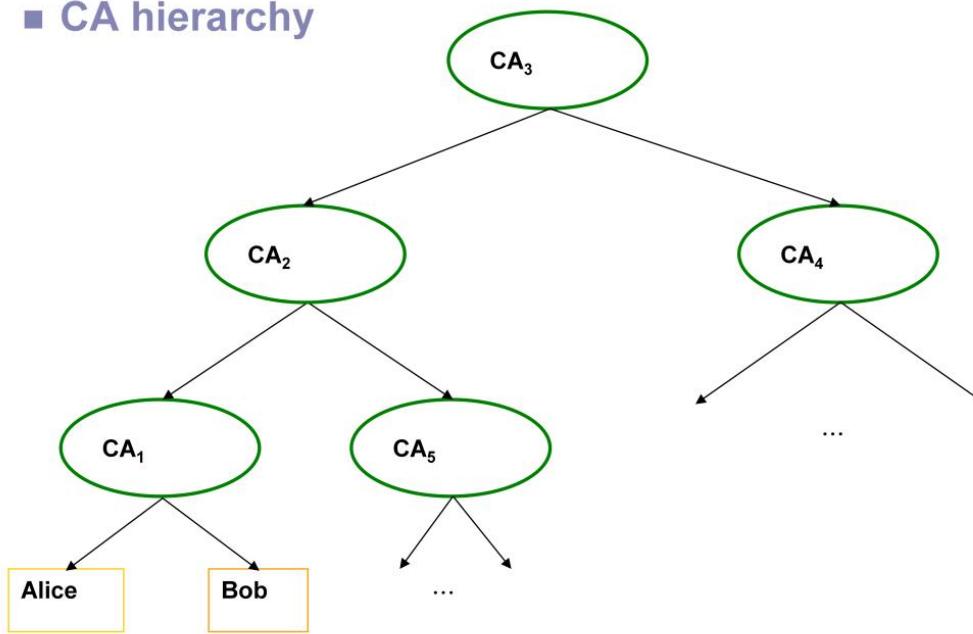
- Remaining issue: CA's public key

- ▶ 1 order of magnitude smaller, as there are far fewer CAs than user

- Who to trust?
 - Which certificates can be trusted
- Source of Trust
 - How it is established?
- Limiting/controlling trust in a given environment

- Commonly used methods
 - CA Hierarchy
 - Distributed
 - Web
 - User-centric
 - Cross-certification

■ CA hierarchy



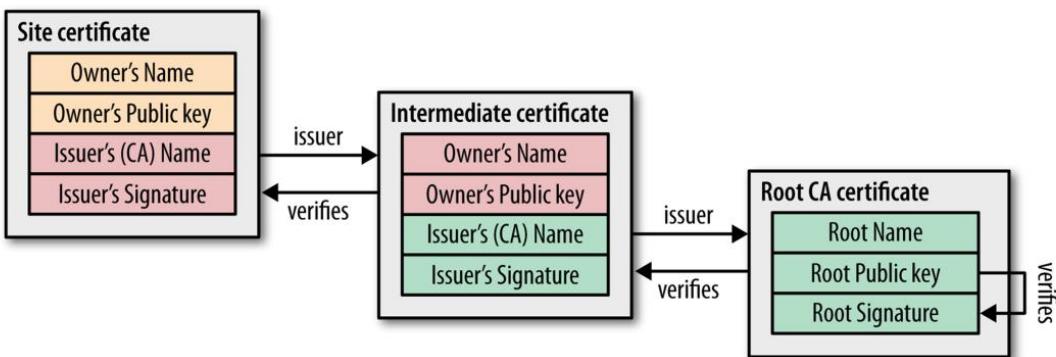
p. 47

Have CA's public key signed by more important CA, thereby building a CA hierarchy (see also later X.509)

- 1) Public keys of Alice and Bob are signed by CA₁
 - 2) Public key of CA₁ is signed by CA₂
 - 3) Public key of CA₂ is signed by CA₃
- Until number of CAs is sufficiently limited, so that a limited list of trusted public keys exists (e.g. VeriSign, Belgian government, etc.)
- 4) CA₂ can also sign public keys of other CAs (such as CA₅), which in turn can sign the keys of other users/CAs
 - 5) CA₃ at a higher level in the hierarchy can also in turn sign the public keys of other CAs (such as CA₄), etc.

■ Web model

- A number of root CAs pre-installed in browsers
- The set of root CAs can be modified
 - ▶ But will it be?
- Root CAs are unrelated (no cross-certification)
 - ▶ Except by “CA powers” of browser manufacturer
 - ▶ Browser manufacturer = (implicit) Root CA

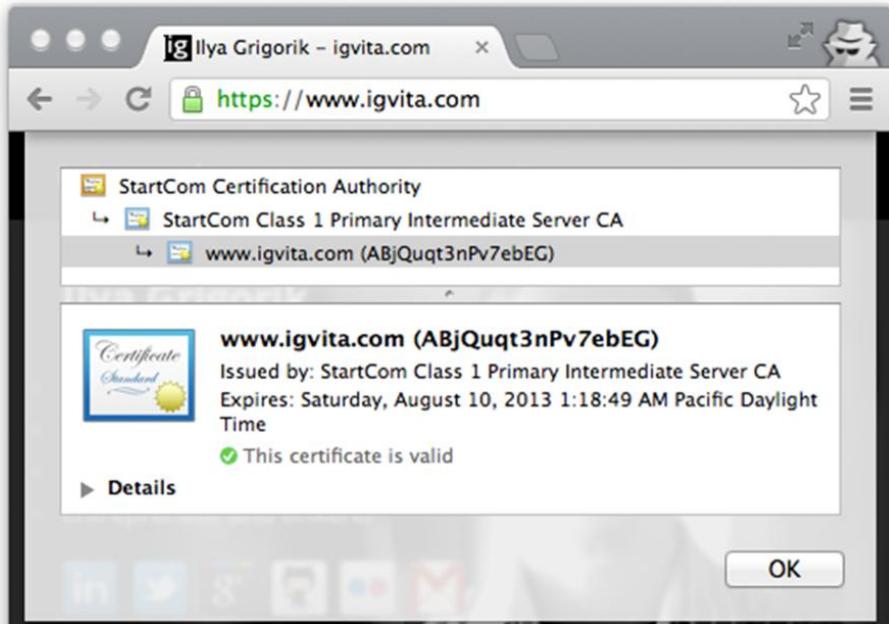


Whom does your browser trust, and whom do you trust when you use the browser? There are at least three answers to this question:

- Manually specified certificates: Every browser and operating system provides a mechanism for you to manually import any certificate you trust. How you obtain the certificate and verify its integrity is completely up to you.
- Certificate authorities: A certificate authority (CA) is a trusted third party that is trusted by both the subject (owner) of the certificate and the party relying upon the certificate.
- The browser and the operating system: Every operating system and most browsers ship with a list of well-known certificate authorities. Thus, you also trust the vendors of this software to provide and maintain a list of trusted parties.

In practice, it would be impractical to store and manually verify each and every key for every website (although you can, if you are so inclined). Hence, the most common solution is to use certificate authorities (CAs) to do this job for us: the browser specifies which CAs to trust (root CAs), and the burden is then on the CAs to verify each site they sign, and to audit and verify that these certificates are not misused or compromised. If the security of any site with the CA's certificate is breached, then it is also the responsibility of that CA to revoke the compromised certificate.

■ Web model



Every browser allows you to inspect the chain of trust of your secure connection, usually accessible by clicking on the lock icon beside the URL.

- igvita.com certificate is signed by StartCom Class 1 Primary Intermediate Server.
- StartCom Class 1 Primary Intermediate Server certificate is signed by the StartCom Certification Authority.
- StartCom Certification Authority is a recognized root certificate authority.

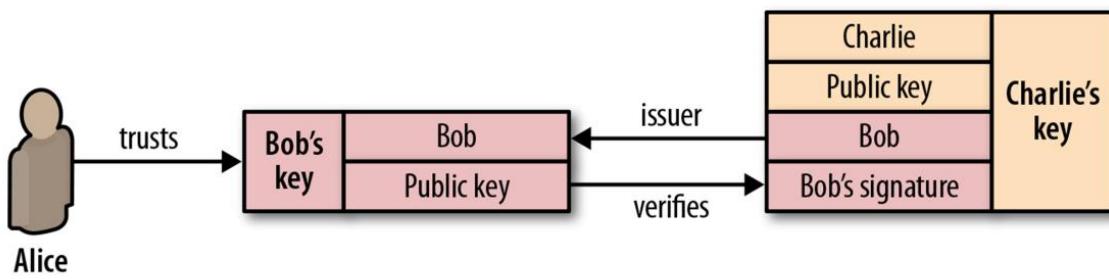
The "trust anchor" for the entire chain is the root certificate authority, which in the case just shown, is the StartCom Certification Authority. Every browser ships with a pre-initialized list of trusted certificate authorities ("roots"), and in this case, the browser trusts and is able to verify the StartCom root certificate. Hence, through a transitive chain of trust in the browser, the browser vendor, and the StartCom certificate authority, we extend the trust to our destination site.

Every operating system vendor and every browser provide a public listing of all the certificate authorities they trust by default. Use your favorite search engine to find and investigate these lists.

In practice, there are hundreds of well-known and trusted certificate authorities, which is also a common complaint against the system. The large number of CAs creates a potentially large attack surface area against the chain of trust in your browser.

■ User centric

- PGP
- User = her own Root CA
 - ▶ Webs of trust
- Good
 - ▶ User fully responsible for trust
- Bad
 - ▶ User fully responsible for trust
 - ▶ Corporate/gov/etc. like to have central control
 - ✓ User-centric not friendly to centralized trust policies



A user centric model (also referred to as web of trust (WoT) model) relies on graph superconnectivity. With a WoT everybody is a CA, and each actor is its own and unique root CA; to cope with gullible or downright malicious "CA", WoT users (i.e. Web browsers) accept a target certificate as valid only if they can verify it through many paths which go through distinct CA and all concur to posit the target server key. WoT security is very dependent on critical mass: it will not give you much until sufficiently many people collaborate.

Alice and Bob could have exchanged their public keys when they met in person, and because they know each other well, they are certain that their exchange was not compromised by an impostor—perhaps they even verified their identities through another, secret (physical) handshake they had established earlier! Next, Alice receives a message from Charlie, whom she has never met, but who claims to be a friend of Bob's. In fact, to prove that he is friends with Bob, Charlie asked Bob to sign his own public key with Bob's private key and attached this signature with his message. In this case, Alice first checks Bob's signature of Charlie's key. She knows Bob's public key and is thus able to verify that Bob did indeed sign Charlie's key. Because she trusts Bob's decision to verify Charlie, she accepts the message and performs a similar integrity check on Charlie's message to ensure that it is, indeed, from Charlie. What we have just done is established a chain of trust: Alice trusts Bob, Bob trusts Charlie, and by transitive trust, Alice decides to trust Charlie. As long as nobody in the chain gets compromised, this allows us to build and grow the list of trusted parties.

Authentication on the Web and in your browser follows the exact same process as shown.

■ Cross-Certification

- **Mechanism:**
 - ▶ Certificates for CAs (not end-entities)
- **Intra- vs. Inter- domain**
- **One or two directions**
 - ▶ CA1 certifies CA2 and/or CA2 certifies CA1
- **Control**
 - ▶ Cross-certificate limits trust
 - ✓ Name, policy, path length, etc. constraints

■ Certificate renewal

- Same keys, same cert, but new dates
- Preferably automatic
- but watch for attributes change!

■ Certificate history

- Key history
 - ▶ For owner: eg to read old encrypted msgs
- Key archive
 - ▶ “For public”: audit, old sigs, disputes, etc.

■ Certificate cancellation

- Certificate Expiration
 - ▶ Natural “peaceful” end of life
- Certificate Revocation
 - ▶ Untimely death, possibly dangerous causes
 - ▶ New keys, new certificate

■ certificate revocation list (CRL)

- **Requested by**
 - ▶ Owner, employer, arbiter, TTP, ???, ...
- **Request sent to**
 - ▶ RA/CA
- **Mechanisms for Revocation checks**
 - ▶ Certificate Revocation Lists (CRLs)
 - ▶ On-line Certificate Status Protocol (OCSP)
 - ✓ Will it live? (SCVP)
- **Revocation delay**
 - ▶ According to Certificate Policy

Occasionally the issuer of a certificate will need to revoke or invalidate the certificate due to a number of possible reasons: the private key of the certificate has been compromised, the certificate authority itself has been compromised, or due to a variety of more benign reasons such as a superseding certificate, change in affiliation, and so on.

Certificate Revocation List (CRL)

Certificate Revocation List (CRL) is defined by RFC 5280 and specifies a simple mechanism to check the status of every certificate: each certificate authority maintains and periodically publishes a list of revoked certificate serial numbers. Anyone attempting to verify a certificate is then able to download the revocation list and check the presence of the serial number within it—if it is present, then it has been revoked. The CRL file itself can be published periodically or on every update and can be delivered via HTTP, or any other file transfer protocol. The list is also signed by the CA, and is usually allowed to be cached for a specified interval. In practice, this workflow works quite well, but there are instances where CRL mechanism may be insufficient:

- The growing number of revocations means that the CRL list will only get longer, and each client must retrieve the entire list of serial numbers.
- There is no mechanism for instant notification of certificate revocation—if the CRL was cached by the client before the certificate was revoked, then the CRL will deem the revoked certificate valid until the cache expires.

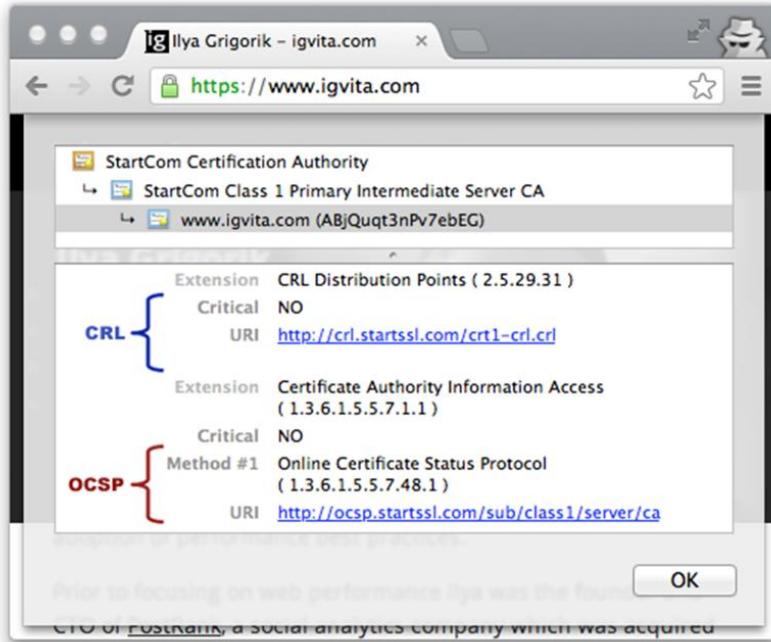
Online Certificate Status Protocol (OCSP)

To address some of the limitations of the CRL mechanism, the Online Certificate Status Protocol (OCSP) was introduced by RFC 2560, which provides a mechanism to perform a real-time check for status of the certificate. Unlike the CRL, which contains all the revoked serial numbers, OCSP allows the verifier to query the certificate database directly for just the serial number in question while validating the certificate chain. As a result, the OCSP mechanism should consume much less bandwidth and is able to provide real-time validation. However, no mechanism is perfect, and the requirement to perform real-time OCSP queries creates several problems of its own:

- The CA must be able to handle the load of the real-time queries.
- The CA must ensure that the service is up and globally available at all times.
- The client must block on OCSP requests before proceeding with the navigation.
- Real-time OCSP requests may impair the client's privacy because the CA knows which sites the client is visiting.

In practice, CRL and OCSP mechanisms are complementary, and most certificates will provide instructions and endpoints for both. The more important part is the client support and behavior: some browsers distribute their own CRL lists, others fetch and cache the CRL files from the CAs. Similarly, some browsers will perform the real-time OCSP check but will differ in their behavior if the OCSP request fails. If you are curious, check your browser and OS certificate revocation settings!

■ certificate revocation list (CRL)



To address revocations, the certificates themselves contain instructions on how to check if they have been revoked. Hence, to ensure that the chain of trust is not compromised, each peer can check the status of each certificate by following the embedded instructions, along with the signatures, as it walks up the certificate chain.

- Network model
- Secure configuration of devices
- Exchanging keys
 - Out of band
 - Diffie-Hellman
 - Asymmetric encryption
 - Trusted third party
 - ▶ Key Distribution Centre (KDC)
 - ▶ Public Key Infrastructure (PKI)
- Secure networking protocols
- Firewalls



Besides the lecturers' own material, many third party, often copyrighted, material is reused within this lecture (e.g. in the notes) under the 'fair use' approach, for sake of educational purpose only, and very limited edition. As a consequence, the current slide set presentation usage is restricted, and is falling under usual copyrights usage.

At the end of every lecture, appropriate references to used materials are included.

■ This work contains content adapted from, amongst others, the following sources (in no particular order)

● Books

- ▶ William Stallings, “**Cryptography and Network Security, principles and practices**”, 6th (international) edition, Prentice Hall, 2010;
- ▶ Matt Bishop, “**Computer Security: Art and Science**”, Addison Wesley, Pearson Education, 2003, ISBN-13: 978-0-201-44099-7

● Lecture slides:

- ▶ “Informatiebeveiliging”, Universiteit Gent, Eric Laermans & Thom Dhaene
- ▶ Stallings, 2014, Lecture slides by Lawrie Brown

● Wikipedia

- ▶ Note page descriptions

● Websites

- ▶ <https://crypto.stackexchange.com>
- ▶ <http://www.roguelynn.com/words/explain-like-im-5-kerberos/>
- ▶ http://chimera.labs.oreilly.com/books/123000000545/ch04.html#COT_CA



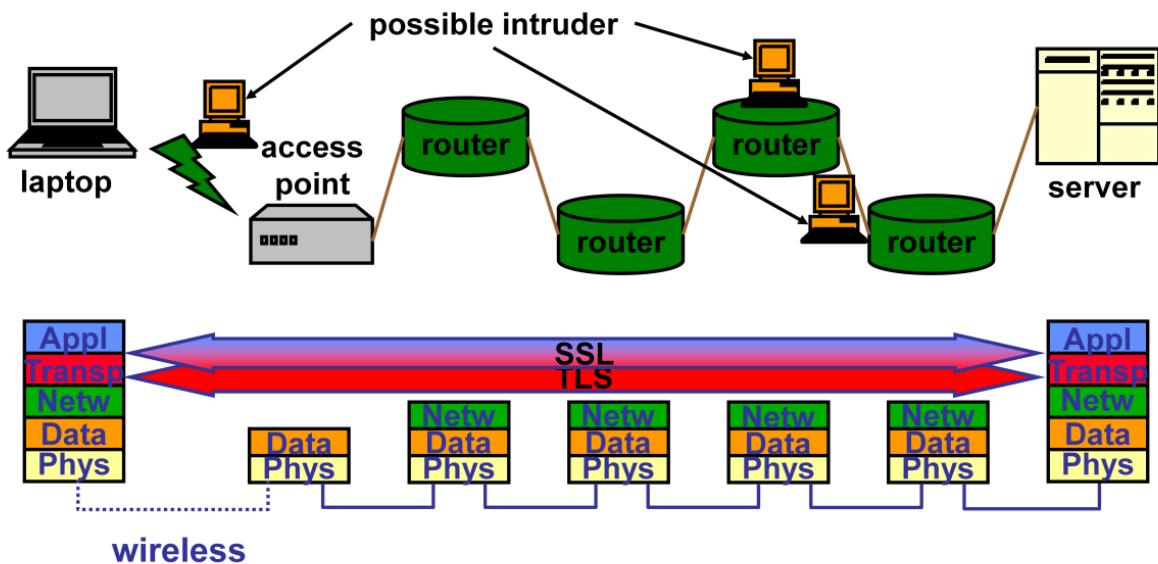
Network and Computer Security

Chapter 3 – Network and communication security

Prof. dr. ir. Eli De Poorter

- Network model
- Secure configuration of devices
 - SSH (Secure Shell)
- Exchanging keys
 - Out of band
 - Diffie-Hellman
 - Asymmetric encryption
 - Trusted third party
 - ▶ Key Distribution Centre (KDC)
 - ▶ Public Key Infrastructure (PKI)
- Secure networking protocols
 - Transport layer: TLS & SSL
 - Network layer: IPSec & VPN
 - Data link layer: WEB & WPA
- Firewalls
 - Packet filter
 - Circuit-level gateway
 - Application-level gateway (proxy)

- Network model
- Secure configuration of devices
- Exchanging keys
- Secure networking protocols
 - Transport layer: TLS & SSL
 - Network layer: IPSec & VPN
 - Data link layer: WEB & WPA
- Firewalls



■ SSL (Secure Socket Layer)

- **Created by Netscape**
- **Meanwhile version 3**
- **Designed to provide security *on top of TCP***
 - ▶ Intermediate layer between transport and application layer
- **Rarely used anymore**

■ TLS (Transport Layer Security)

- **IETF standard (RFC 5246 for version 1.2)**
 - ▶ RFC 2246 for original version 1.0
- **Derived from SSLv3**

p. 5

The SSL protocol was originally developed at Netscape to enable ecommerce transaction security on the Web, which required encryption to protect customers' personal data, as well as authentication and integrity guarantees to ensure a safe transaction. To achieve this, the SSL protocol was implemented under the application layer, directly on top of TCP, enabling protocols above it (HTTP, email, instant messaging, and many others) to operate unchanged while providing communication security when communicating across the network. When SSL is used correctly, a third-party observer can infer the connection endpoints, type of encryption, as well as the frequency and an approximate amount of data sent, but cannot read or modify any of the actual data.

When the SSL protocol was standardized by the IETF, it was renamed to Transport Layer Security (TLS). Many use the TLS and SSL names interchangeably, but technically, they are different, since each describes a different version of the protocol. TLS only differs in a few points from SSLv3. It generally concerns technical implementations of some algorithms. E.g. the session keys are derived in a different way (using a pseudorandom function PRF), and a different MAC function is used. Different "Alert Codes" are used, with a different categorisation in fatal errors and warnings. We limit ourselves to the discussion of TLS 1.2 since SSL is rarely used anymore. One may still occasionally encounter TLS 1.0 or TLS 1.1 for secure websites, although TLS 1.2 is now supported by any up-to-date version of any common browser (be it IE, Firefox, Chrome, Opera, etc.) (check the configuration parameters of your browser).

■ Similarities

- Both secure the transport layer by providing tunnels

■ Differences

- **TSL/SSL is designed for protecting generic transport layer traffic**
- **SSH includes multiplexing, user authentication, terminal management.**
TSL/SSL doesn't
- **SSL uses X.509 certificates, SSH a custom format**
- **Different optimizations**
 - ▶ SSH: shell applications, TLS: https speed ups
 - ▶ SSH2/SFTP vs FTP-TLS

SSL	SSH	
n.a.	RFC4254	Connection multiplexing
n.a.	RFC4252	User authentication
RFC5246	RFC4253	Encrypted data transport

6

SSL is a transport layer method for protecting data transported over a network, whereas SSH is a network application for logging in and sharing data with a remote computer.

TLS has goals and features similar to those of the SSH Transport and User Authentication protocols. It provides a single, full-duplex byte stream to clients, with cryptographically assured privacy and integrity, and optional authentication. It differs from SSH in the following principal ways:

- TLS server authentication is optional: the protocol supports fully anonymous operation, in which neither side is authenticated. Such connections are inherently vulnerable to man-in-the-middle attacks. In SSH-TRANS, server authentication is mandatory, which protects against such attacks. Of course, it is always possible for a client to skip the step of verifying that the public key supplied by the server actually belongs to the entity the client intended to contact (e.g. using the /etc/ssh_known_hosts file). However, SSH-TRANS at least demands going through the motions.
- TLS lacks user authentication because it doesn't need it (TLS just needs to authenticate the two connecting interfaces, which SSH can also do). When using TLS, authentication is done at higher layer such as in the webbrowser.
- In TLS both client and server authentication are done with X.509 public-key certificates whereas SSH has its own format. This makes TLS a bit more cumbersome to use than SSH in practice, since it requires a functioning public-key infrastructure (PKI) to be in place, and certificates are more complicated things to generate and manage than SSH keys. However, a PKI system provides scalable key management, which SSH currently lacks.
- TLS does not provide the range of client authentication options that SSH does; public-key is the only option.
- Since SSL is a transport layer protocol, it lacks connection multiplexing capabilities. TLS does not have the extra features provided by the SSH Connection Protocol (SSH-CONN). SSH-CONN uses the underlying SSH-TRANS connection to provide multiple logical data channels to the application, as well as support for remote program execution, terminal management, tunneled TCP connections, flow control, etc.
- The transport layer protection in SSH is similar in capability to TLS. TLS and SSH both provide the cryptographic elements to build a tunnel for confidential data transport with checked integrity. To this end, they use similar techniques, and may suffer from the same kind of attacks, so they should provide similar security (i.e. good security) assuming they are both properly implemented.

Conceptually, you could take SSH and replace the tunnel part with the one from SSL. You could also take HTTPS and replace the SSL tunnel with SSH-with-data-transport and a hook to extract the server public key from its certificate. There is no scientific impossibility and, if done properly, security would remain the same. However, there is no widespread set of conventions or existing tools to realize this. So we do not use SSL and SSH for the same things, but that's because of what tools historically came with the implementations of those protocols, not due to a security related difference. And whoever implements or uses SSL or SSH would be well advised to look at what kind of attacks were tried on both protocols.

Although SSL/TLS is most well-known for its use in setting up secure http connections (i.e.: HTTPS), other application protocols can also utilize SSL/TLS tunnels. For example, there is an enhancement to standard FTP (as defined in RFC 959), which uses the same FTP commands (and protocol) over secure sockets, i.e. over SSL/TLS. This enhancement is defined in RFC 4217 and is known under the names of FTPS, FTP-TLS, and FTP-over-SSL. This is not the same protocol as the SFTP protocol, which also provides secure file transfer (over SSH tunnels) but is not part of the FTP standard and as such not fully FTP compatible. As such, the two protocols are completely different, not related to each other and not (!) compatible with each other. SSH2/SFTP is more popular in the Unix world while FTP-TLS is more popular in the Windows World.

■ TLS connection

- **Transport between communicating entities**

- ▶ Based on peer-to-peer relation between client and server
- ▶ Associated with 1 TLS session
- ▶ State defined by number of parameters:
 - ✓ Random numbers for each connection
 - ✓ Keys for symmetric encryption
 - ✓ Initialisation vectors (IV) in CBC-mode
 - ✓ Sequence number for received/sent messages
 - ✓ Key for computation of MAC

p. 7

A connection is a communication channel between a client and a server. Connections are usually short lived and servers are usually configured to timeout a connection if it is left idle for too long.

■ TLS session

- **Association between client and server**
 - ▶ Created using “TLS Handshake Protocol”
 - ▶ State defined by number of parameters
 - ✓ Session identifier
 - ✓ X.509v3 certificate of communicating entities
 - ✓ Compression method used
 - ✓ Specification of cryptographic techniques
 - ✓ “master secret”
 - ✓ “is resumable”

p. 8

The difference between a connection and a session is that connection is a live communication channel, and a session is a way of maintaining state on the server side (including a set of negotiated cryptography parameters). You can close a connection, but can keep the session, even store it to disk, and subsequently resume it using another connection, maybe in completely different process, or even after system reboot (of course, stored session should be kept both on the client and on the server). On the other hand, you can renegotiate TLS parameters and create an entirely new session without interrupting connection.

The session identifier is an arbitrary byte sequence, chosen by the server. It is possible that no certificate is used for one or for both communicating parties.

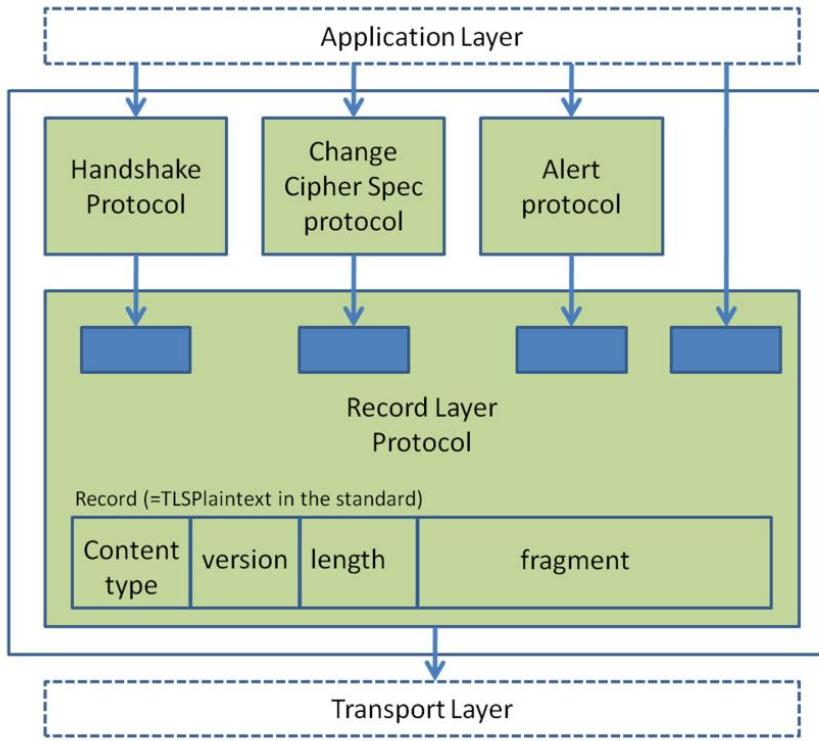
The compression method also is optional.

The “master secret” is secret information (48 bytes) shared by server and client.

The flag “is resumable” indicates whether the session can be used to start new connections.

■ TLS architecture

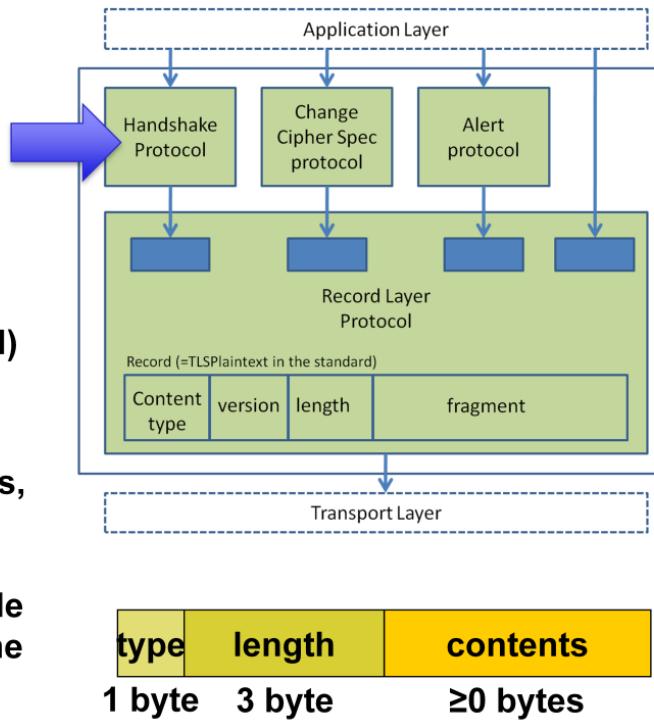
- **Base layer “TLS Record Protocol”**
 - ▶ Security services for protocols in higher layers
- **On top of it:**
 - ▶ applications (HTTP, FTP, SMTP, etc.)
 - ▶ TLS-specific protocols
 - ✓ “TLS Handshake Protocol”
 - ✓ “TLS Change Cipher Protocol”
 - ✓ “TLS Alert Protocol”



p. 10

The TLS protocol has itself a two layered architecture; the TLS Record layer protocol and the TLS Handshaking protocols. The latter has three sub-protocols; the Handshake protocol, the Alert protocol and the Change Cipher Spec protocol. On top of these protocols, the application layer resides.

- Allows (reciprocal) authentication of server and client
- Security negotiation
- Message structure:
 - 1 byte for message type (10 types defined)
 - 3 bytes for message length
 - The message contents, with the parameters required for this message type (variable length, empty for some message types)

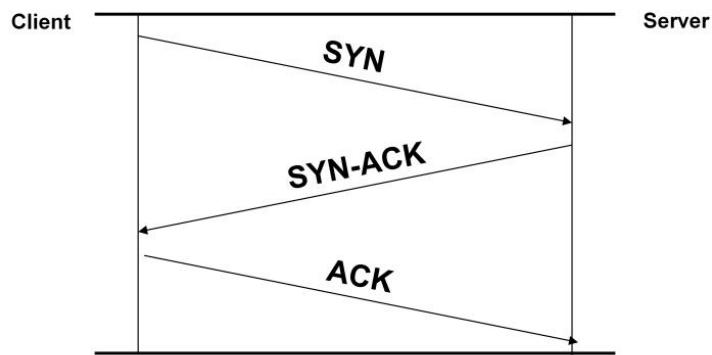


p. 11

Before the client and the server can begin exchanging application data over TLS, the encrypted tunnel must be negotiated: the client and the server must agree on the version of the TLS protocol, choose the ciphersuite, and verify certificates if necessary.

■ Phase 0

- Setting up a TCP connection



p. 12

The TCP session has to be established before SSL/TLS protocol can start. For setting up a TCP connection, we first complete the TCP three-way handshake.



■ Phase 0

- Setting up a TCP connection

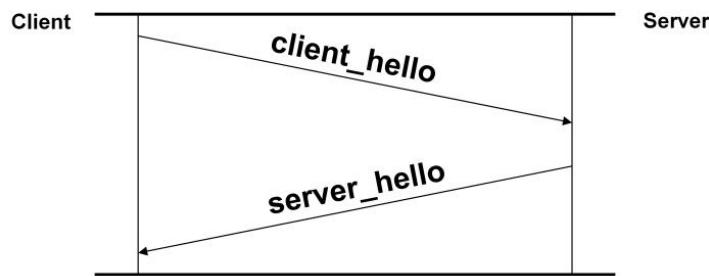
	Time	Source	Destination	Protocol	Info
1	0.000000	130.235.201.241	130.235.203.249	TCP	inst1_boots > https [SYN] Seq=0 Win=16384 Len=0 Ms
2	0.000452	130.235.203.249	130.235.201.241	TCP	https > inst1_boots [SYN, ACK] Seq=0 Ack=1 win=584
3	0.000494	130.235.201.241	130.235.203.249	TCP	inst1_boots > https [ACK] Seq=1 Ack=1 Win=17520 Le
4	0.001074	130.235.201.241	130.235.203.249	SSL	Client Hello
5	0.001341	130.235.203.249	130.235.201.241	TCP	https > inst1_boots [ACK] Seq=1 Ack=141 win=6432 L
6	0.005269	130.235.203.249	130.235.201.241	TLSV1	Server Hello,
7	0.005838	130.235.203.249	130.235.201.241	TLSV1	Certificate, Server Hello Done
8	0.006480	130.235.201.241	130.235.203.249	TCP	inst1_boots > https [ACK] seq=141 Ack=1895 win=175
9	0.012905	130.235.201.241	130.235.203.249	TLSV1	Alert (Level: Fatal, Description: Unknown CA)
10	0.013244	130.235.201.241	130.235.203.249	TCP	inst1_boots > https [RST, ACK] Seq=148 Ack=1895 wi
11	0.072262	130.235.201.241	130.235.203.249	TCP	inst1 bootc > https [SYN] Seq=0 Win=16384 Len=0 Ms

p. 13

First three packets: setting up a TCP connection

■ Phase 1

• Defining security capabilities



p. 14

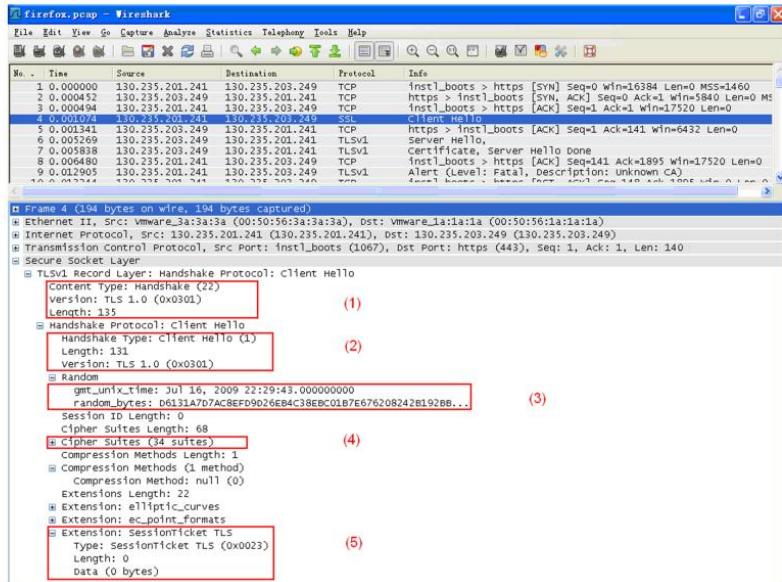
The client sends a message to the server of type "client_hello" with following parameters:

- Version: the highest TLS version supported by the client
- Random: a time stamp (32 bits) and 28 bytes generated by a secure random number generator; to be used as a nonce
- SessionID: non-zero value if the client wants to update the parameters of an existing connection or to create a new connection within an existing session.
- CipherSuite: a list of combinations of cryptographic algorithms supported by the client (combinations in order of preference); such a combination describes the key exchange method (RSA, DH (fixed, ephemeral, or anonymous), etc.), the encryption specification (algorithm, hash function, keys, etc.), the MAC function, and (starting with version 1.2) the PRF (pseudorandom function) used.
- Compression Method: a list of compression techniques supported by the client.

The server responds with a "server_hello" message, with a similar structure as the "client_hello" message. The chosen version of TLS is the lowest of 1) the version requested by the client 2) the highest version supported by the server. The "CipherSuite" contains the combination proposed by the client, which was chosen by the server (similar for the "Compression Method"). Because of the numbering used (cf. Major and Minor version fields in the TLS record header) TLS 1.0 is a higher version than SSLv3.

■ Phase 1

● Defining security capabilities



The screenshot shows a Wireshark capture of a TLS handshake. Frame 4, which is the ClientHello message, is highlighted. The details pane shows the following structure:

- Record Layer:**
 - Content Type:** Handshake (22)
 - Version:** TLS 1.0 (0x0301)
 - Length:** 135
- Handshake Protocol:** Client Hello (1)
 - Type:** Client Hello (1)
 - Version:** TLS 1.0 (0x0301)
- Random:**
 - gmt_unix_time:** Jul 16, 2009 22:29:43.000000000
 - random_bytes:** D633A7D7ACBEFD9026EB4C38EBC01B7E676208242B192B...
- Cipher Suites:** (34 suites)
 - Compression Methods:** Length: 1
 - Compression Methods:** null (0)
 - Extensions:** Length: 22
 - Extension:** elliptic_curves
 - Extension:** ec_point_formats
 - Extension:** SessionTicket TLS
 - Type:** SessionTicket TLS (0x0023)
 - Length:** 0
 - Data:** (0 bytes)

p. 15

4th packet: ClientHello (client to server)

(1) Record Layer Protocol Type and Version

Here the value 22 represents the Handshake protocol contained in this Record Layer and the client uses TLS version1.0.

(2) Handshake Message type

Here the code 1 identifies the ClientHello message

(3) client side Random

The Random number generated by Client here is a parameter for the key calculation between client and server.

(4) Cipher suites

There are 34 suites supported. The details of this field are shown in the next slide.

(5) SessionTicket

SessionTicket is a TLS extension included in the ClientHello message, which defines a way to resume a TLS session without requiring session-specific states at the TLS server. Here the extension is empty since the client connect to the server for the first time.

■ Phase 1

● Defining security capabilities

firefox.pcap - Wireshark

Length: 131
Version: TLS 1.0 (0x0301)
Random
Session ID Length: 0
Cipher Suites Length: 68

Cipher suites (34 suites)

- Cipher Suite: TLS_ECDHE_ECDSA_WITH_AES_256_CBC_SHA (0xc00a)
- Cipher Suite: TLS_ECDHE_RSA_WITH_AES_256_CBC_SHA (0xc014)
- Cipher Suite: TLS_DHE_RSA_WITH_CAMELLIA_256_CBC_SHA (0x0088)
- Cipher Suite: TLS_DHE_DSS_WITH_CAMELLIA_256_CBC_SHA (0x0087)
- Cipher Suite: TLS_DHE_RSA_WITH_AES_128_CBC_SHA (0x0039)
- Cipher Suite: TLS_DHE_DSS_WITH_AES_128_CBC_SHA (0x0038)
- Cipher Suite: TLS_ECDH_RSA_WITH_AES_256_CBC_SHA (0xc00f)
- Cipher Suite: TLS_ECDH_ECDSA_WITH_AES_256_CBC_SHA (0xc005)
- Cipher Suite: TLS_RSA_WITH_CAMELLIA_256_CBC_SHA (0x0084)
- Cipher Suite: TLS_RSA_WITH_AES_256_CBC_SHA (0x0005)
- Cipher Suite: TLS_ECDH_ECDSA_WITH_RC4_128_SHA (0xc007)
- Cipher Suite: TLS_ECDH_ECDSA_WITH_AES_128_CBC_SHA (0xc009)
- Cipher Suite: TLS_ECDH_RSA_WITH_RC4_128_SHA (0xc011)
- Cipher Suite: TLS_DHE_RSA_WITH_AES_128_CBC_SHA (0x0013)
- Cipher Suite: TLS_DHE_RSA_WITH_CAMELLIA_128_CBC_SHA (0x0045)
- Cipher Suite: TLS_DHE_DSS_WITH_CAMELLIA_128_CBC_SHA (0x0044)
- Cipher Suite: TLS_DHE_RSA_WITH_AES_128_CBC_SHA (0x0033)
- Cipher Suite: TLS_DHE_DSS_WITH_AES_128_CBC_SHA (0x0032)
- Cipher Suite: TLS_ECDH_RSA_WITH_RC4_128_SHA (0xc00c)
- Cipher Suite: TLS_ECDH_RSA_WITH_AES_128_CBC_SHA (0xc00e)
- Cipher Suite: TLS_ECDH_RSA_WITH_CAMELLIA_128_CBC_SHA (0xc002)
- Cipher Suite: TLS_ECDH_ECDSA_WITH_AES_128_CBC_SHA (0xc004)
- Cipher Suite: TLS_RSA_WITH_CAMELLIA_128_CBC_SHA (0x0041)
- Cipher Suite: TLS_RSA_WITH_RC4_128_M05 (0x0004)
- Cipher Suite: TLS_RSA_WITH_RC4_128_SHA (0x0005)
- Cipher Suite: TLS_RSA_WITH_AES_128_CBC_SHA (0x002f)
- Cipher Suite: TLS_ECDH_ECDSA_WITH_3DES_EDE_CBC_SHA (0xc008)
- Cipher Suite: TLS_ECDHE_RSA_WITH_3DES_EDE_CBC_SHA (0xc012)
- Cipher Suite: TLS_DHE_RSA_WITH_3DES_EDE_CBC_SHA (0x0016)
- Cipher Suite: TLS_DHE_DSS_WITH_3DES_EDE_CBC_SHA (0x0013)
- Cipher Suite: TLS_ECDH_RSA_WITH_3DES_EDE_CBC_SHA (0xc00d)
- Cipher Suite: TLS_ECDH_ECDSA_WITH_3DES_EDE_CBC_SHA (0xc003)
- Cipher Suite: SSL_RSA_FIPS_WITH_3DES_EDE_CBC_SHA (0xfeff)
- Cipher Suite: TLS_RSA_WITH_3DES_EDE_CBC_SHA (0x000a)

Compression Methods Length: 1
Compression Methods (1 method)

p. 16

■ Phase 1

● Defining security capabilities

firefox.pcap - Wireshark

No.	Time	Source	Destination	Protocol	Info
1	0.000000	130.235.201.241	130.235.203.249	TCP	https > instl_boots > https [SYN] Seq=0 Win=46384 Len=0 MSS=1460
2	0.000452	130.235.203.249	130.235.201.241	TCP	https > instl_boots [SYN, ACK] Seq=0 Ack=1 Win=5840 Len=0 Ms
3	0.000494	130.235.201.241	130.235.203.249	TCP	instl_boots > https [ACK] Seq=1 Ack=1 Win=17520 Len=0
4	0.001074	130.235.201.241	130.235.203.249	SSL	Client Hello
5	0.001341	130.235.203.249	130.235.201.241	TCP	https > instl_boots [ACK] Seq=1 Ack=141 Win=6432 Len=0
6	0.001341	130.235.201.241	130.235.203.249	TCP	Server Hello
7	0.001341	130.235.201.241	130.235.203.249	TLSV1	Client Hello done
8	0.006480	130.235.201.241	130.235.203.249	TCP	instl_boots > https [ACK] Seq=141 Ack=1895 Win=17520 Len=0
9	0.012905	130.235.201.241	130.235.203.249	TLSV1	Alert (Level: Fatal, Description: Unknown CA)
10	0.013244	130.235.201.241	130.235.203.249	TCP	instl_boots > https [RST, ACK] Seq=141 Ack=1895 Win=0 Len=0
11	0.013244	130.235.201.241	130.235.203.249	TCP	instl_boots > https [SYN] Seq=0 Win=16384 Len=0 MSS=1460
12	0.072706	130.235.203.249	130.235.201.241	TCP	https > instl_boots [SYN] Seq=0 Ack=1 Win=5840 Len=0 Ms
13	0.072751	130.235.201.241	130.235.203.249	TCP	instl_boots > https [ACK] Seq=1 Ack=1 Win=17520 Len=0

Frame 6: 1514 bytes on wire (1181 bits), 1514 bytes captured (1181 bits) on interface eth0
 u Ethernet II, Src: vmware_3a:3a:3a (00:50:56:3a:3a:3a), Dst: vmware_3a:3a:3a (00:50:56:3a:3a:3a)
 I Internet Protocol Version 4, Src: 130.235.203.249 (130.235.203.249), Dst: 130.235.201.241 (130.235.201.241)
 T Transmission Control Protocol, Src Port: https (443), Dst Port: instl_boots (1007), Seq: 1, Ack: 141, Len: 1460
 S Secure Socket Layer
 TLSV1 Record Layer: Handshake Protocol: Server Hello
 Content Type: Handshake (22) (1)
 Version: TLS 1.0 (0x0301)
 Length: 48
 Handshake Type: Server Hello (2) (2)
 Length: 44
 Version: TLS 1.0 (0x0301)
 Random
 GMT UNIX time: Jul 16, 2009 23:33:08.000000000
 Random bytes: 75487E0389FE42CE1A0A604E06974F5919A502D2C96BA3B...
 Session ID Length: 0
 Cipher Suite: TLS_RSA_WITH_AES_256_CBC_SHA (0x0035) (4)
 Compression Method: null (0)
 Extensions Length: 4
 Extension: SessionTicket TLS
 Type: SessionTicket TLS (0x0023)
 Length: 0
 Data (0 bytes) (5)

p. 17

6th packet: ServerHello (server to client)

(1) Record Layer Protocol Type and Version

The server indicate its use of TLS version 1.0.

(2) Handshake Message type

The ServerHello is the respons to the client request message "ClientHello".

(3) server side Random

The random number generated by the server is used in the key calculation together with the one from the client side.

(4) selected Cipher Suite

The server checks its support Cipher Suites and select one matched to the Cipher Suites brought up by client in the ClientHello message. Here the TLS_RSA_WITH_AES_256_CBC_SHA is selected, ie: the key exchange uses RSA and the encryption and MAC algorithms are AES and SHA respectively.

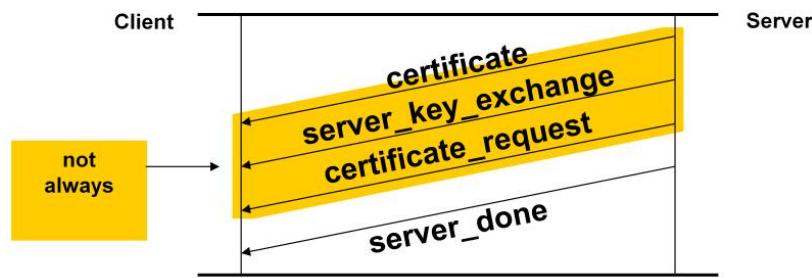
(5) SessionTicket

The server here sends an empty SessionTicket extension to indicate that it will send a new session ticket later in a NewSessionTicket handshake message before the ChangeCipherSpec message finishes.

For the wireshark traces of the remaining phases, we refer to <http://ipseclab.eit.lth.se/tiki-index.php?page=3.+SSL+and+TLS>

■ Phase 2

• Server authentication and key exchange



p. 18

If authentication is required (which is true for most key exchange procedures) the server sends a “certificate” message containing a X.509 certificate (or a chain of such certificates).

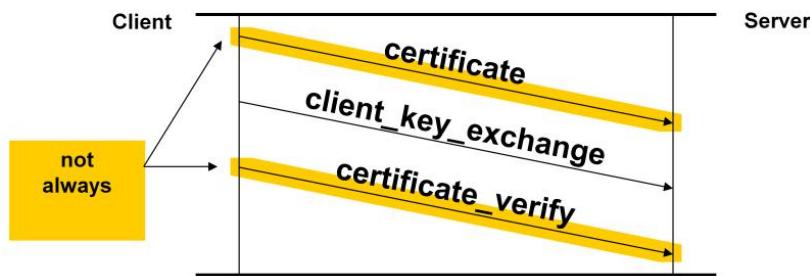
After this message may follow a “server_key_exchange” message if necessary (e.g. not for fixed DH, or if RSA is used for the key exchange, where the server key may be used for encryption). This message will generally contain a digital signature over the parameters of the server and over the nonces that were used in phase 1 (to avoid replay attacks).

If client authentication is also required, the server may request a client certificate using a “certificate_request” message, having as parameters the certificate type (for the asymmetric algorithm and the desired key usage) and a list of acceptable certification authorities (impossible for anonymous DH).

Finally, this phase is closed with a “server_done” message, without contents.

■ Phase 3

• Client authentication and key exchange



p. 19

The client verifies the validity of the certificate he has received from the server and he verifies whether the parameters he received from the server are acceptable to him.

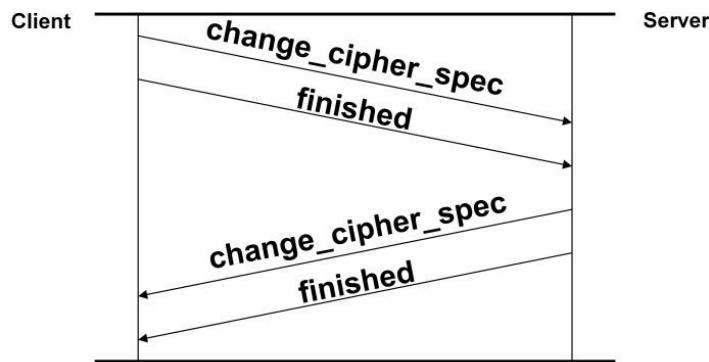
If the server has requested a certificate, the client will send a "certificate" message to the server, containing a valid certificate. If he can't do this, he sends back an empty certificate list.

The client then sends a "client_key_exchange" message, with the information needed to generate a session key (48 bytes "pre-master secret" for RSA; public parameters for ephemeral or anonymous DH; nothing for fixed DH).

Finally, the client may send a "certificate_verify" message (if a certificate had been sent before and if he has the capability to sign a message). This message is the digital signature of the hash value of information from the previous messages of the handshake, including the "Master Secret", from which the session key will later be derived). This avoids the abuse of someone else's certificate by a malicious client, who could send a "certificate" message, but not this "certificate_verify" message.

■ Phase 4

• Finishing handshake



p. 20

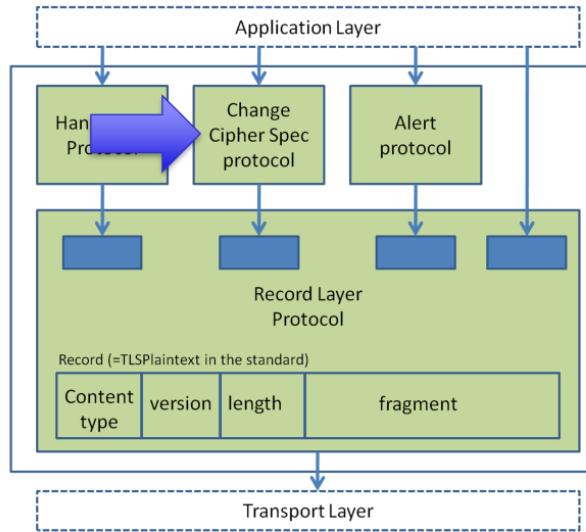
The client now sends a “change_cipher_spec” message (using the Change Cipher Spec protocol) to the server and copies the temporary “CipherSpec” (as has been agreed upon in this handshake) to the present value of “CipherSpec” (which will be used in this session).

He then sends a “finished” message (using the new security parameters), signifying that the handshake is over for him. This verifies that the key exchange and the authentication have been successful. It contains hash values (using MD5 and SHA-1 in TLS1.0 and 1.1, using SHA-256 (or the hash function agreed upon in the ciphersuite) from TLS 1.2) of data with among other things the “Master Secret” and all the previous messages from the handshake.

The server then responds with his “change_cipher_spec” message (using the Change Cipher Spec protocol) and his “finished” message.

After this client and server are able to communicate with each other and to exchange data using the SSL Record Protocol.

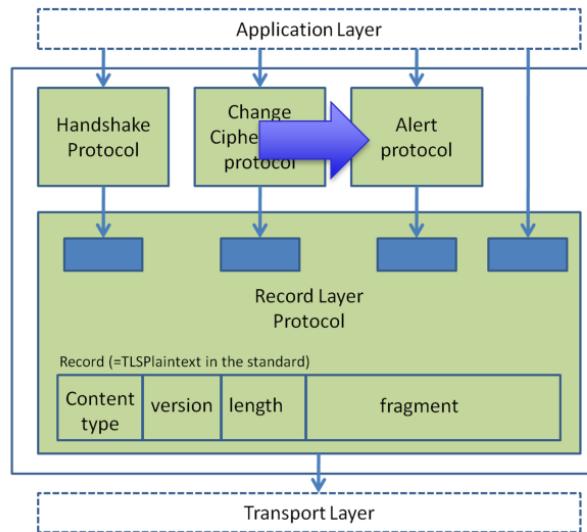
- Only 1 possible message: 1 byte with value 1
 - Causes pending state to be copied to present state
 - Needed when updating encryption techniques of present connection



p. 21

■ Messages for errors and warnings

- **1 byte for the level: “fatal” or “warning”**
 - ▶ In case of “fatal” alert:
 - ✓ Connection terminated
 - ✓ No new connections for this session
- **1 byte for problem description**



p. 22

Possible warnings:

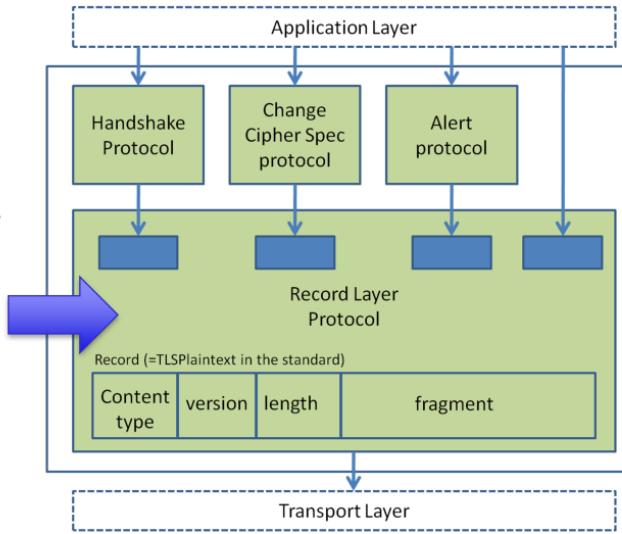
- problems with certificates
- connection termination by 1 of both communicating entities

Possible problems which can occur and are fatal:

- reception of an invalid message
- reception of an invalid MAC
- problems when decompressing
- error during handshake procedure (no acceptable security parameters were found)
- illegal parameters in a field of the handshake procedure

■ Base layer of TLS

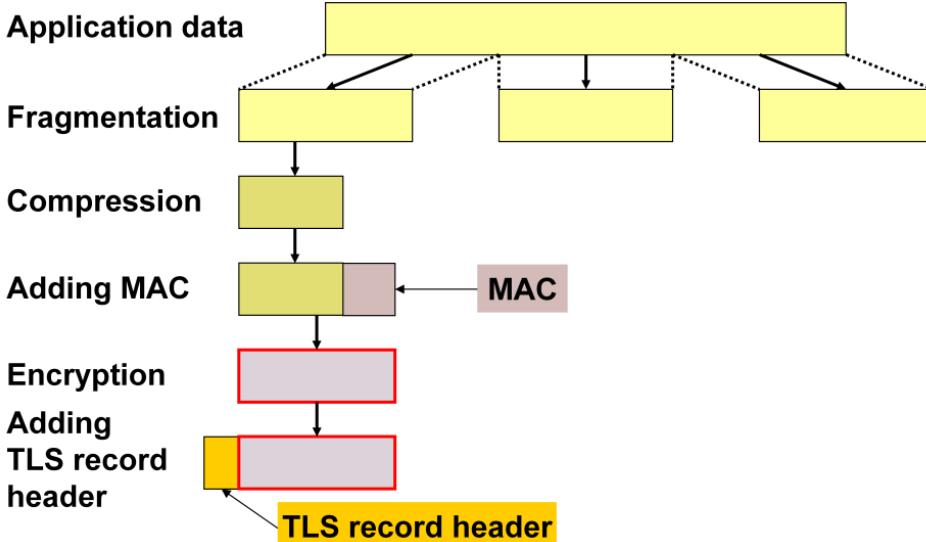
- Fragments data
- Optionally compresses data
- Provides confidentiality and authentication for TLS connections
 - ▶ Uses 2 secret keys
 - ✓ 1 for confidentiality
 - ✓ 1 for authentication using MAC
 - ▶ Both created by TLS Handshake Protocol



p. 23

The TLS Record Protocol is a layered protocol which processes the data to be transmitted, fragments the data into manageable blocks of at most 2^{14} bytes, optionally compresses the data, applies a MAC for integrity protection, encrypts, and transmits the result. Received data is handled in the reversed order: its is decrypted, verified, decompressed, reassembled, and then delivered to higher-level clients.

Note that transport-level TLS compression is not content aware and will end up attempting to recompress already compressed data (images, video, etc.). In practice, most browsers disable support for TLS compression. For these browsers, instead of relying on TLS compression, make sure your server is configured to compress all text-based assets and that you are using an optimal compression format for all other media types, such as images, video, and audio.



p. 24

A typical workflow for delivering application data is as follows:

- The application data is divided into fragments of maximum 2^{14} (=16384) bytes or 16 KB.
- Application data is optionally compressed. The compression is optional (default is no compression), must be reversible and mustn't increase the data length by more than 1024 bytes.
- Message authentication code (MAC) or HMAC is added for data integrity. The MAC is negotiated during the TLS handshake (see later). The MAC is calculated over the (possibly compressed) data fragment. The MAC is calculated over the following fields.
 - $\text{MAC}(K_{\text{MAC}}, \text{Seq}\# || \text{Type} || \text{Length} || \text{Data})$
 - MAC : MAC function used (generally HMAC)
 - K_{MAC} : shared secret key for MAC computation
 - $\text{Seq}\#$: sequence number for message (against replay attacks)
 - Type : protocol for fragment processing (cf. header)
 - Length : length of (uncompressed) data fragment
 - Data : the data fragment
- Data is encrypted using the symmetric encryption algorithm agreed upon (IDEA, DES, 3DES, RC4-40, RC4-128, AES-128, AES-256, etc.). If block encryption is used, it may be necessary to use the required padding after adding the MAC before encrypting the data.
- Finally, a TLS record header is added (see next slide)

Compared to SSH, both provide authenticated encryption, but in two different ways.

- SSH uses the so-called Encrypt-and-MAC, that is the ciphered message is juxtaposed to a message authentication code (MAC) of the clear message to add integrity. This is not proven to be always fully secure (even if in practical cases it should be enough).
- SSL uses MAC-then-Encrypt: a MAC is juxtaposed to the clear text, then they are both encrypted. This is not the best either, as with some block cipher modes parts of the MAC can be guessable and reveal something on the cipher. This led to vulnerabilities in TLS 1.0 (BEAST attack).

■ TLS Record Header

- **Content Type**
 - ▶ 8 bits, determines application layer protocol for data processing
- **Major and Minor version**
 - ▶ 2 times 8 bits (for TLS 1.2: values 3 and 3)
- **Compressed Length**
 - ▶ Length of the data

Content Type	Major Version	Minor Version	Compressed Length
--------------	---------------	---------------	-------------------

p. 25

The TLS Record protocol is also responsible for identifying different types of messages (handshake, alert, or data) via the "Content Type" field that is part of the TLS record header that is added to each outgoing packet. Only 4 categories are taken into account for the field "Content Type": the 3 TLS specific protocols (Change Cipher Spec, Alert, and Handshake Protocols) and all other application protocols (e.g. HTTP, FTP, SMTP, etc.). The values of Major and Minor version indicate that TLS can be seen as an extension of SSL (SSLv3 had values 3 and 0, TLS 1.0 had values 3 and 1, etc.).

- **Derived from**
“`pre_master_secret`” using
PRF (pseudorandom
function)
 - Generated by client and sent
to server using RSA
encryption
 - Or exchanged using DH key
exchange
 - 384 bits
- **Dependent on values of**
“`nonces`” from first phase
of handshake
 - **Reuse of keys impossible**



Roger Buffle Jr. supplies his father with
yet another computer password.

The importance of a good
random generator

■ Master Secret

● Pseudorandom function PRF

► $\text{PRF}(\text{secret}, \text{label}, \text{seed}) = \text{P_hash}(\text{secret}, \text{label} \parallel \text{seed})$

✓ $\text{P_hash}(\text{secret}, \text{seed}) = \text{HMAC_hash}(\text{secret}, \text{A}(1) \parallel \text{seed}) \parallel \text{HMAC_hash}(\text{secret}, \text{A}(2) \parallel \text{seed}) \parallel \text{HMAC_hash}(\text{secret}, \text{A}(3) \parallel \text{seed}) \parallel \dots$

» $\text{A}(0) = \text{seed}$

» $\text{A}(i) = \text{HMAC_hash}(\text{secret}, \text{A}(i-1))$

✓ hash: chosen hash function

» MD5 or SHA-1 (for TLS 1.0 and 1.1)

» SHA-256 by default (TLS 1.2)

p. 27

Provided only for your information: the method for calculating the shared key. In TLS 1.2 the PRF may depend on the cipher suite. The version presented here is the default version (and in practice more or less the only PRF used). From this session dependent shared secret value the required keys and other parameters are derived: client write MAC key, server write MAC key, client write key, server write key, client write IV, server write IV

■ Heartbeat protocol

- **New protocol (2012) on top of TLS Record Protocol**

- ▶ Can be sent at each moment

- ✓ Except during handshake

- **Main goal**

- ▶ Liveliness check

- ✓ Especially for DTLS (no session management)

- ✓ But also for TLS

- » Useful when no data is sent for a long period of time

■ Heartbeat protocol

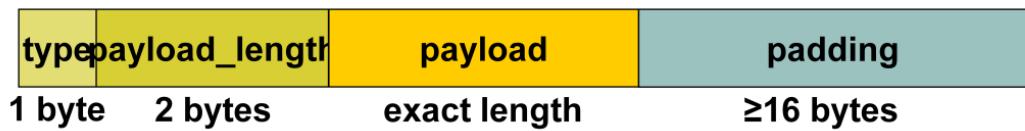
- Two types of messages

- ▶ HeartbeatRequest

- ✓ Payload length must correspond to given length
 - » Otherwise: discard message

- ▶ HeartbeatResponse

- ✓ Payload must be exact copy of payload from HeartbeatRequest
 - » Otherwise: tacitly discard message



■ Heartbleed bug

- **Nickname of CVE-2014-0160**
- **Nothing wrong with protocol**
- **But something was wrong with its implementation in OpenSSL versions 1.0.1 up to 1.0.1f**
 - ▶ Incorrectly dealing with wrong HeartbeatRequests
 - ▶ Reveals contents of remaining server memory (or causes server to crash)
 - ✓ Could be server private keys
 - ✓ Could be cached passwords

■ A few links

- **Heartbeat protocol:**
 - ▶ <https://tools.ietf.org/html/rfc6520>
- **Heartbleed bug:**
 - ▶ <http://heartbleed.com/>
 - ▶ <http://www.klocwork.com/blog/software-security/saving-you-from-heartbleed/> (with the faulty code)
 - ▶ <https://www.schneier.com/blog/archives/2014/04/heartbleed.html>
 - ▶ https://blog.wireshark.org/2014/04/heartbleed-traffic/?utm_source=rss&utm_medium=rss&utm_campaign=heartbleed-traffic
 - ▶ <http://www.riverbed.com/blogs/Retroactively-detecting-a-prior-Heartbleed-exploitation-from-stored-packets-using-a-BPF-expression.html>

■ Performance comparison with SSH

- **TLS/SSL handshake**
 - ▶ Fewer message exchanges
 - ✓ More efficient
 - ▶ Entirely authenticated
 - ✓ Only partial authentication with SSH
 - ▶ More straightforward cipher suite selection
- **No encryption of packet length in TLS/SSL**
 - ▶ Making decryption easier

■ Overhead?

- **Gmail switched to use only HTTPS**

- ▶ *On our production frontend machines, SSL/TLS accounts for less than 1% of the CPU load, less than 10 KB of memory per connection and less than 2% of network overhead. - Adam Langley (Google), 2010*

- **Facebook uses commodity hardware**

- ▶ *We have deployed TLS at a large scale using both hardware and software load balancers. We have found that modern software-based TLS implementations running on commodity CPUs are fast enough to handle heavy HTTPS traffic load without needing to resort to dedicated cryptographic hardware - Doug Beaver (Facebook)*

- **Twitter uses best encryption whenever possible**

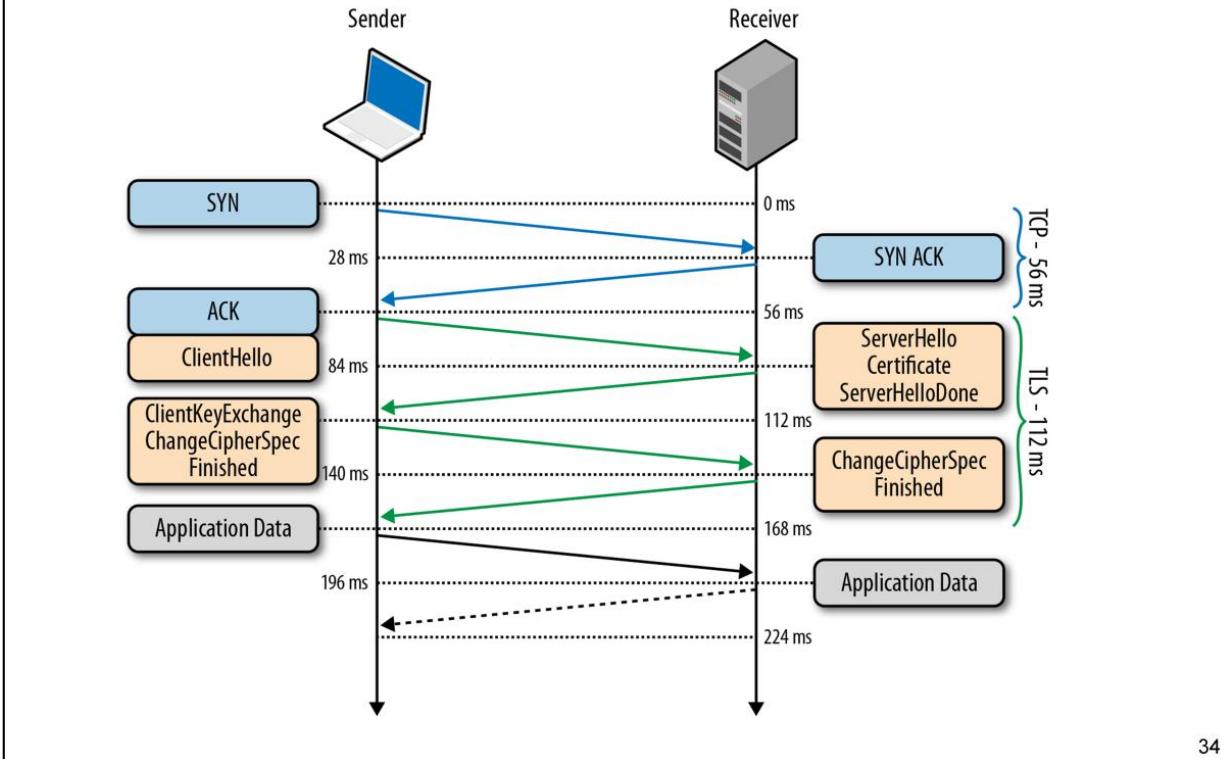
- ▶ *Elliptic Curve Diffie-Hellman (ECDHE) is only a little more expensive than RSA for an equivalent security level... In practical deployment, we found that enabling and prioritizing ECDHE cipher suites actually caused negligible increase in CPU usage. HTTP keepalives and session resumption mean that most requests do not require a full handshake, so handshake operations do not dominate our CPU usage. We find 75% of Twitter's client requests are sent over connections established using ECDHE. The remaining 25% consists mostly of older clients that don't yet support the ECDHE cipher suites. - Jacob Hoffman-Andrews (Twitter)*

33

Due to the layered architecture of the network protocols, running an application over TLS is no different from communicating directly over TCP. As such, there are no, or at most minimal, application modifications that you will need to make to deliver it over TLS. However, establishing and maintaining an encrypted channel introduces additional computational costs for both peers. Specifically, first there is the asymmetric (public key) encryption used during the TLS handshake. Then, once a shared secret is established, it is used as a symmetric key to encrypt all TLS records.

As we noted earlier, public key cryptography is more computationally expensive when compared with symmetric key cryptography, and in the early days of the Web often required additional hardware to perform "SSL offloading." The good news is this is no longer the case. Modern hardware has made great improvements to help minimize these costs, and what once required additional hardware can now be done directly on the CPU. Large organizations such as Facebook, Twitter, and Google, which offer TLS to hundreds of millions of users, perform all the necessary TLS negotiation and computation in software and on commodity hardware.

Nevertheless, techniques such as "TLS Session Resumption" (see next slide) are still important optimizations, which will help you decrease the computational costs and latency of public key cryptography performed during the TLS handshake. There is no reason to spend CPU cycles on work that you don't need to do.



Compared to traditional TCP traffic (e.g. HTTP requests), the use of TLS introduces overhead. Before the client and the server can begin exchanging application data over TLS, the encrypted tunnel must be negotiated: the client and the server must agree on the version of the TLS protocol, choose the ciphersuite, and verify certificates if necessary. Unfortunately, each of these steps requires new packet roundtrips between the client and the server, which adds startup latency to all TLS connections.

0 ms – 56 ms: TLS runs over a reliable transport (TCP), which means that we must first complete the TCP three-way handshake, which takes one full roundtrip.

56 ms – 84 ms: With the TCP connection in place, the client sends a number of specifications in plain text, such as the version of the TLS protocol it is running, the list of supported ciphersuites, and other TLS options it may want to use.

84 ms – 112 ms: The server picks the TLS protocol version for further communication, decides on a ciphersuite from the list provided by the client, attaches its certificate, and sends the response back to the client. Optionally, the server can also send a request for the client's certificate and parameters for other TLS extensions.

112 ms – 140 ms: Assuming both sides are able to negotiate a common version and cipher, and the client is happy with the certificate provided by the server, the client initiates either the RSA or the Diffie-Hellman key exchange, which is used to establish the symmetric key for the ensuing session.

140 ms – 168 ms: The server processes the key exchange parameters sent by the client, checks message integrity by verifying the MAC, and returns an encrypted "Finished" message back to the client.

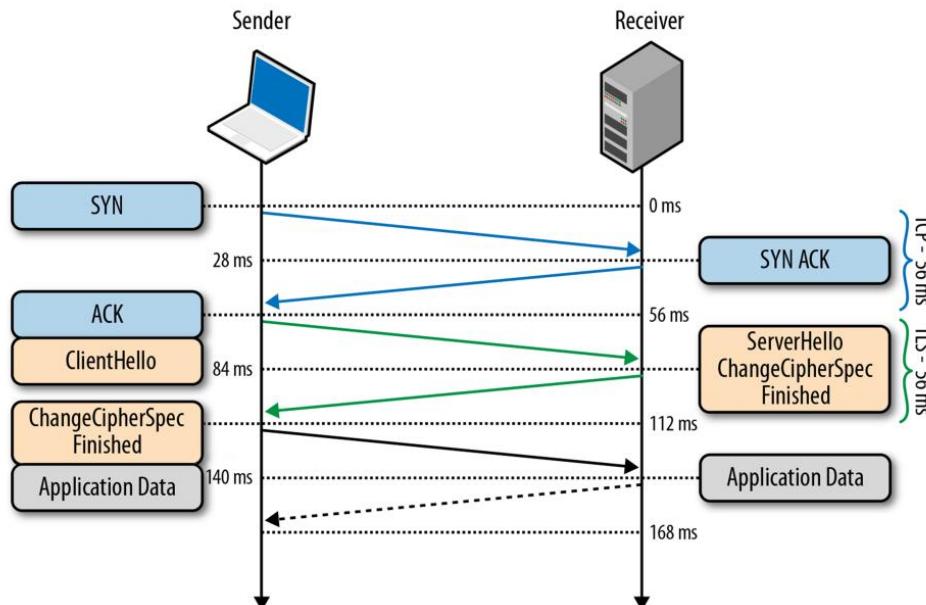
168 ms – 224 ms: The client decrypts the message with the negotiated symmetric key, verifies the MAC, and if all is well, then the tunnel is established and application data can now be sent.

(These results assume a 28 millisecond one-way "light in fiber" delay between New York and London)

It is important to realize that every TLS connection will require up to two extra roundtrips on top of the TCP handshake—that's a long time to wait before any application data can be exchanged! If not managed carefully, delivering application data over TLS can add hundreds, if not thousands of milliseconds of network latency.

■ Abbreviated handshake (session resumption)

- TLS provides an ability to resume or share the same negotiated secret key data between multiple connections.



35

New TLS connections require two roundtrips for a "full handshake" and CPU resources to verify and compute the parameters for the ensuing session. However, the good news is that we don't have to repeat the "full handshake" in every case.

Abbreviated handshake (using a session ID)

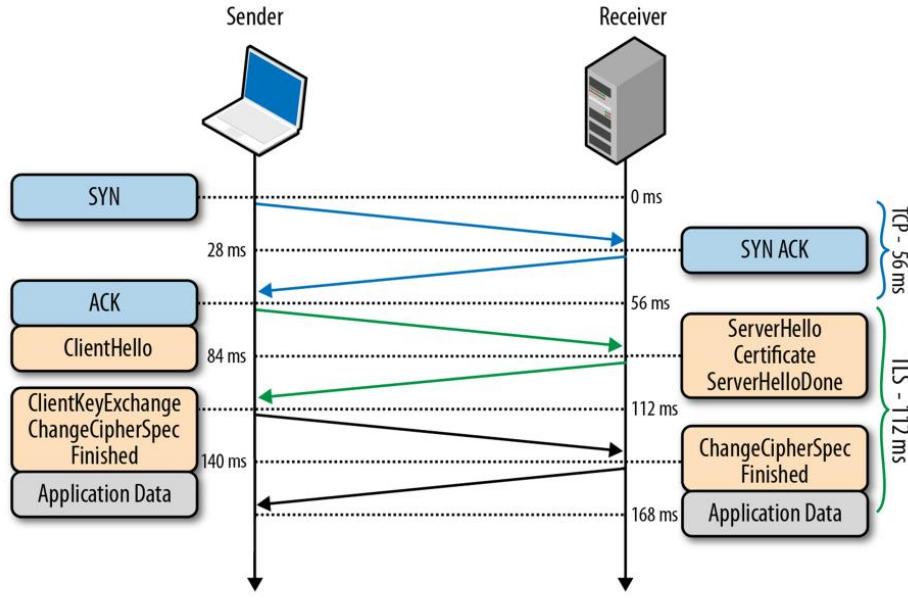
If the client has previously communicated with the server, an "abbreviated handshake" can be used, which requires one roundtrip and allows the client and server to reduce the CPU overhead by reusing the previously negotiated parameters for the secure session. To this end, session identifiers are used. The first Session Identifiers (RFC 5246) resumption mechanism was introduced in SSL 2.0, which allowed the server to create and send a 32-byte session identifier as part of its "ServerHello" message during the full TLS negotiation from the previous slide. Internally, the server could then maintain a cache of session IDs and the negotiated session parameters for each peer. In turn, the client could then also store the session ID information and include the ID in the "ClientHello" message for a subsequent session, which serves as an indication to the server that the client still remembers the negotiated cipher suite and keys from previous handshake and is able to reuse them. Assuming both the client and the server are able to find the shared session ID parameters in their respective caches, then an abbreviated handshake can take place. Otherwise, a full new session negotiation is required, which will generate a new session ID. Leveraging session identifiers allows us to remove a full roundtrip, as well as the overhead of public key cryptography, which is used to negotiate the shared secret key. This allows a secure connection to be established quickly and with no loss of security, since we are reusing the previously negotiated session data. In practice, most web applications attempt to establish multiple connections to the same host to fetch resources in parallel, which makes session resumption a must-have optimization to reduce latency and computational costs for both sides. Most modern browsers intentionally wait for the first TLS connection to complete before opening new connections to the same server: subsequent TLS connections can reuse the SSL session parameters to avoid the costly handshake. However, one of the practical limitations of the Session Identifiers mechanism is the requirement for the server to create and maintain a session cache for every client. This results in several problems on the server, which may see tens of thousands or even millions of unique connections every day: consumed memory for every open TLS connection, a requirement for session ID cache and eviction policies, and nontrivial deployment challenges for popular sites with many servers, which should, ideally, use a shared TLS session cache for best performance. None of the preceding problems are impossible to solve, and many high-traffic sites are using session identifiers successfully today. But for any multiserver deployment, session identifiers will require some careful thinking and systems architecture to ensure a well operating session cache.

To address this concern for server-side deployment of TLS session caches, the "Session Ticket" (RFC 5077) replacement mechanism was introduced, which removes the requirement for the server to keep per-client session state. Instead, if the client indicated that it supports Session Tickets, in the last exchange of the full TLS handshake, the server can include a New Session Ticket record, which includes all of the session data encrypted with a secret key known only by the server. This session ticket is then stored by the client and can be included in the SessionTicket extension within the ClientHello message of a subsequent session. Thus, all session data is stored only on the client, but the ticket is still safe because it is encrypted with a key known only by the server.

The session identifiers and session ticket mechanisms are respectively commonly referred to as *session caching* and *stateless resumption* mechanisms. The main improvement of stateless resumption is the removal of the server-side session cache, which simplifies deployment by requiring that the client provide the session ticket on every new connection to the server—that is, until the ticket has expired. In practice, deploying session tickets across a set of load-balanced servers also requires some careful thinking and systems architecture: all servers must be initialized with the same session key, and an additional mechanism may be needed to periodically rotate the shared key across all servers.

■ False start

- Don't wait for handshake



36

Session resumption does not help in cases where the visitor is communicating with the server for the first time, or if the previous session has expired, to this end the false start optimization is defined. False Start is an optional TLS protocol extension that allows the client and server to start transmitting encrypted application data when the handshake is only partially complete—i.e. once ChangeCipherSpec and Finished messages are sent, but without waiting for the other side to do the same. This optimization reduces new handshake overhead to one roundtrip. False Start does not modify the TLS handshake protocol, rather it only affects the protocol timing of when the application data can be sent. Intuitively, once the client has sent the ClientKeyExchange record, it already knows the encryption key and can begin transmitting application data—the rest of the handshake is spent confirming that nobody has tampered with the handshake records, and can be done in parallel. As a result, False Start allows us to keep the TLS handshake at one roundtrip regardless of whether we are performing a full or abbreviated handshake.

Because False Start is only modifying the timing of the handshake protocol, it does not require any updates to the TLS protocol itself and can be implemented unilaterally—i.e. the client can simply begin transmitting encrypted application data sooner. Well, that's the theory. In practice, even though TLS False Start should be backwards compatible with all existing TLS clients and servers, enabling it by default for all TLS connections proved to be problematic due to some poorly implemented servers. As a result, all modern browsers are capable of using TLS False Start, but will only do so when certain conditions are met by the server:

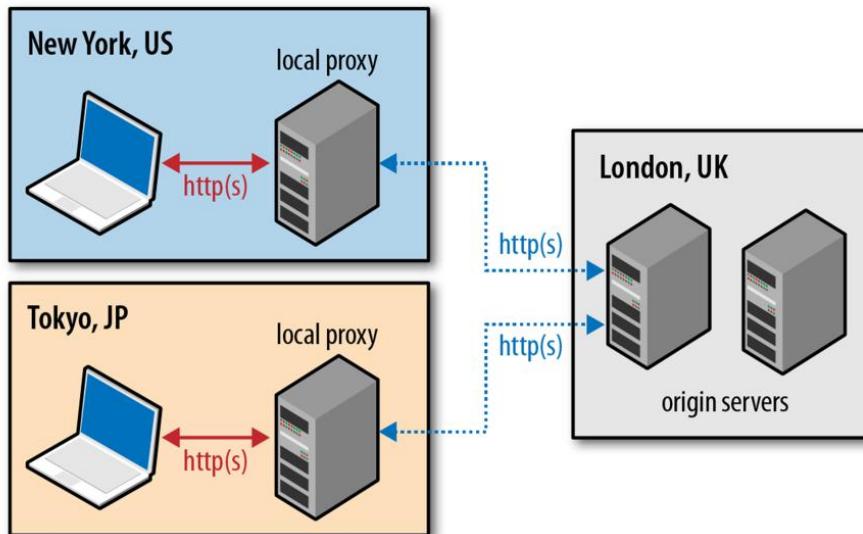
- Chrome and Firefox require an NPN/ALPN protocol advertisement to be present in the server handshake, and that the cipher suite chosen by the server enables forward secrecy.
- Safari requires that the cipher suite chosen by the server enables forward secrecy.
- Internet Explorer uses a combination of a blacklist of known sites that break when TLS False Start is enabled, and a timeout to repeat the handshake if the TLS False Start handshake failed.

To enable TLS False Start across all browsers the server should advertise a list of supported protocols via the NPN/ALPN extension—e.g. "http/1, http/1.1"—and be configured to support and prefer cipher suites that enable forward secrecy.

For best results, both optimizations should be used together to provide a single roundtrip handshake for new and returning visitors, plus computational savings for sessions that can be resumed based on previously negotiated session parameters.

■ Early termination (proxies)

- Place servers closer to the client
- Minimize latency costs



37

The extensibility and the success of HTTP created a vibrant ecosystem of various proxies and intermediaries on the Web: cache servers, security gateways, web accelerators, content filters, and many others. In some cases we are aware of their presence (explicit proxies), and in others they are completely transparent to the end user. Unfortunately, the very success and the presence of these servers has created a small problem for anyone who tries to deviate from the HTTP protocol in any way: some proxy servers may simply relay HTTP extensions or alternative wire formats they cannot interpret, others may continue to blindly apply their logic even when they shouldn't, and some, such as security appliances, may infer malicious traffic where there is none.

In other words, in practice, deviating from the well-defined semantics of HTTP on port 80 will often lead to unreliable deployments: some clients will have no problems, while others may fail with unpredictable behaviors—e.g., the same client may see different connectivity behaviors as it migrates between different networks. Due to these behaviors, new protocols and extensions to HTTP, such as WebSocket, SPDY, and others, often rely on establishing an HTTPS tunnel to bypass the intermediate proxies and provide a reliable deployment model: the encrypted tunnel obfuscates the data from all intermediaries. This solves the immediate problem, but it does have a real downside of not being able to leverage the intermediaries, many of which provide useful services: authentication, caching, security scanning, and so on.

It is worth noting that the use of proxies also has disadvantages. With HTTPS, once the packet is received by a device in the destination network (Web Server, Border Router, load balancer, etc...) it is un-encrypted and spends the rest of its journey in plain text. Many would argue that this is not a big deal since the traffic is internal at this time, but if the payload contains sensitive data, it is being stored un-encrypted in the log files of every network device it passes through until it gets to its final destination (in contrast, with SSH, typically, the destination device is specified and the transmission is encrypted until it reaches this device). There are ways of re-encrypting the HTTPS data but these are extra steps that most forget to take when implementing an HTTPS solution in their environment.

Early termination is a simple technique of placing your servers closer to the user to minimize the latency cost of each roundtrip between the client and the server. A nearby server can also terminate the TLS session, which means that the TCP and TLS handshake roundtrips are much quicker and the total connection setup latency is greatly reduced. In turn, the same nearby server can then establish a pool of long-lived, secure connections to the origin servers and proxy all incoming requests and responses to and from the origin servers. In a nutshell, moving the server closer to the client accelerates TCP and TLS handshakes.

■ Record size

```

▽ [8 Reassembled TCP Segments (11221 bytes): #169(1460), #170(1460), #172(1460), #174(1460),
   #175(1460), #177(1460), #179(1460), #180(1001)]
  [Frame: 169, payload: 0-1459 (1460 bytes)]
  [Frame: 170, payload: 1460-2919 (1460 bytes)]
  [Frame: 172, payload: 2920-4379 (1460 bytes)]
  [Frame: 174, payload: 4380-5839 (1460 bytes)]
  [Frame: 175, payload: 5840-7299 (1460 bytes)]
  [Frame: 177, payload: 7300-8759 (1460 bytes)]
  [Frame: 179, payload: 8760-10219 (1460 bytes)]
  [Frame: 180, payload: 10220-11220 (1001 bytes)]
  [Segment count: 8]
  [Reassembled TCP length: 11221]
▽ Secure Sockets Layer
  ▽ TLSv1 Record Layer: Application Data Protocol: http
    Content Type: Application Data (23)
    Version: TLS 1.0 (0x0301)
    Length: 11216
    Encrypted Application Data: 07ed92e420530da2e2755a5b5372ef32b53e0d4e7c20c3d8...

```

WireShark capture of 11,211-byte TLS record split over 8 TCP segments

38

All application data delivered via TLS is transported within a record protocol. The maximum size of each record is 16 KB, and depending on the chosen cipher, each record will add anywhere from 20 to 40 bytes of overhead for the header, MAC, and optional padding. If the record then fits into a single TCP packet, then we also have to add the IP and TCP overhead: 20-byte header for IP, and 20-byte header for TCP with no options. As a result, there is potential for 60 to 100 bytes of overhead for each record. For a typical maximum transmission unit (MTU) size of 1,500 bytes on the wire, this packet structure translates to a minimum of 6% of framing overhead. The smaller the record, the higher the framing overhead. However, simply increasing the size of the record to its maximum size (16 KB) is not necessarily a good idea! If the record spans multiple TCP packets, then the TLS layer must wait for all the TCP packets to arrive before it can decrypt the data. If any of those TCP packets get lost, reordered, or throttled due to congestion control, then the individual fragments of the TLS record will have to be buffered before they can be decoded, resulting in additional latency. In practice, these delays can create significant bottlenecks for the browser, which prefers to consume data byte by byte and as soon as possible.

Small records incur overhead, large records incur latency, and there is no one value for the "optimal" record size. Instead, for web applications, which are consumed by the browser, the best strategy is to dynamically adjust the record size based on the state of the TCP connection:

- When the connection is new and TCP congestion window is low, or when the connection has been idle for some time (e.g. slow-start restart), each TCP packet should carry exactly one TLS record, and the TLS record should occupy the full maximum segment size (MSS) allocated by TCP.
- When the connection congestion window is large and if we are transferring a large stream (e.g. streaming video), the size of the TLS record can be increased to span multiple TCP packets (up to 16KB) to reduce framing and CPU overhead on the client and server.

Using a dynamic strategy delivers the best performance for interactive traffic: small record size eliminates unnecessary buffering latency and improves the *time-to-first-{HTML byte, ..., video frame}*, and a larger record size optimizes throughput by minimizing the overhead of TLS for long-lived streams.

To determine the optimal record size for each state let's start with the initial case of a new or idle TCP connection where we want to avoid TLS records from spanning multiple TCP packets:

- Allocate 20 bytes for IPv4 framing overhead and 40 bytes for IPv6.
- Allocate 20 bytes for TCP framing overhead.
- Allocate 40 bytes for TCP options overhead (timestamps, SACKs).

Assuming a common 1,500-byte starting MTU, this leaves 1,420 bytes for a TLS record delivered over IPv4, and 1,400 bytes for IPv6. To be future-proof, use the IPv6 size, which leaves us with 1,400 bytes for each TLS record payload—adjust as needed if your MTU is lower.

Next, the decision as to when the record size should be increased and reset if the connection has been idle, can be set based on pre-configured thresholds: increase record size to up to 16 KB after X KB of data have been transferred, and reset the record size after Y milliseconds of idle time. Typically, configuring the TLS record size is not something we can control at the application layer. Instead, this is a setting and perhaps even a compile-time constant or flag on the TLS server.

As of early 2014, Google's servers use small TLS records that fit into a single TCP segment for the first ~1 MB of data, increase record size to 16 KB after that to optimize throughput, and then reset record size back to a single segment after ~1 second of inactivity—lather, rinse, repeat. Similarly, if your servers are handling a large number of TLS connections, then minimizing memory usage per connection can be a vital optimization. By default, popular libraries such as OpenSSL will allocate up to 50 KB of memory per connection, but as with the record size, it may be worth checking the documentation or the source code for how to adjust this value. Google's servers reduce their OpenSSL buffers down to about 5 KB.

■ Certificate chain

```
▷ [4 Reassembled TCP Segments (5341 bytes): #98(1402), #99(1460), #101(1176), #102(1303)]
  ▷ Secure Sockets Layer
    ▷ TLSv1.1 Record Layer: Handshake Protocol: Certificate
      Content Type: Handshake (22)
      Version: TLS 1.1 (0x0302)
      Length: 5327
    ▷ Handshake Protocol: Certificate
      Handshake Type: Certificate (11)
      Length: 5323
      Certificates Length: 5320
    ▷ Certificates (5320 bytes)
```

WireShark capture of a 5,323-byte TLS certificate chain

39

Another speed-up optimization that is targeted mainly towards HTTPS traffic is the inclusion of the certificate chains. Verifying the chain of trust requires that the browser traverse the chain, starting from the site certificate, and recursively verifying the certificate of the parent until it reaches a trusted root. To find the parent certificates, the child certificate will usually contain the URL for the parent.

Hence, the first optimization you should make is to verify that the server does not forget to include all the intermediate certificates when the handshake is performed. If you forget, many browsers will still work, but they will instead be forced to pause the verification and fetch the intermediate certificate on their own, verify it, and then continue. This will most likely require a new DNS lookup, TCP connection, and an HTTP GET request, adding hundreds of milliseconds to your handshake. Conversely, it is important to make sure you do not include unnecessary certificates in your chain. Or, more generally, you should aim to minimize the size of your certificate chain. Recall that server certificates are sent during the TLS handshake, which is likely running over a new TCP connection that is in the early stages of its slow-start algorithm. If the certificate chain exceeds TCP's initial congestion window, then we will inadvertently add yet another roundtrip to the handshake: certificate length will overflow the congestion window and cause the server to stop and wait for a client ACK before proceeding.

The certificate chain shown in the slide above is over 5 KB in size, which will overflow the initial congestion window size of older servers and force another roundtrip of delay into the handshake. One possible solution is to increase the initial congestion window. In addition, you should investigate if it is possible to reduce the size of the sent certificates:

- Minimize the number of intermediate CAs. Ideally, your sent certificate chain should contain exactly two certificates: your site and the CA's intermediary certificate; use this as a criteria in the selection of your CA. The third certificate, which is the CA root, should already be in the browser's trusted root and hence should not be sent.
- It is not uncommon for many sites to include the root certificate of their CA in the chain, which is entirely unnecessary: if your browser does not already have the certificate in its trust store, then it won't be trusted, and including the root certificate won't change that.
- A carefully managed certificate chain can be as low as 2 or 3 KB in size, while providing all the necessary information to the browser to avoid unnecessary roundtrips or out-of-band requests for the certificates themselves. Optimizing your TLS handshake mitigates a critical performance bottleneck, since every new TLS connection is subject to its overhead.

■ Performance checklist

- **Minimize certificate chain**
- **Upgrade your libraries**
- **Enable false start**
- **Use proxies**
- ...

40

As an application developer, you are shielded from virtually all the complexity of TLS. Short of ensuring that you do not mix HTTP and HTTPS content on your pages, your application will run transparently on both. However, the performance of your entire application will be affected by the underlying configuration of your server. The good news is it is never too late to make these optimizations, and once in place, they will pay high dividends for every new connection to your servers.

A short list of optimizations to keep in mind (from <http://chimera.labs.oreilly.com/books/1230000000545/ch04.html>)

- Upgrade TLS libraries to latest release, and (re)build servers against them.
- Enable and configure session caching and stateless resumption.
- Monitor your session caching hit rates and adjust configuration accordingly.
- Configure forward secrecy ciphers to enable TLS False Start.
- Terminate TLS sessions closer to the user to minimize roundtrip latencies.
- Use dynamic TLS record sizing to optimize latency and throughput.
- Ensure that your certificate chain does not overflow the initial congestion window.
- Remove unnecessary certificates from your chain; minimize the depth.
- Configure OCSP stapling on your server.
- Disable TLS compression on your server.
- Configure SNI support on your server.
- Append HTTP Strict Transport Security header.
- Get best performance from TCP; see “Optimizing for TCP”.

■ Experimenting

• Openssl

```
$> openssl s_client -state -CAfile startssl.ca.crt -connect igvita.com:443
CONNECTED(00000003)
SSL_connect:before/connect initialization
SSL_connect:SSLv2/v3 write client hello A
SSL_connect:SSLv3 read server hello A
depth=2 /C=IL/O=StartCom Ltd./OU=Secure Digital Certificate Signing
/CN=StartCom Certification Authority
verify return:1
depth=1 /C=IL/O=StartCom Ltd./OU=Secure Digital Certificate Signing
/CN=StartCom Class 1 Primary Intermediate Server CA
verify return:1
depth=0 /description=ABjQuqt3nPv7ebEG/C-US
/CN=www.igvita.com/emailAddress=ilya@igvita.com
verify return:1
SSL_connect:SSLv3 read server certificate A
SSL_connect:SSLv3 read server done A ①
SSL_connect:SSLv3 write client key exchange A
SSL_connect:SSLv3 write change cipher spec A
SSL_connect:SSLv3 write finished A
SSL_connect:SSLv3 flush data
SSL_connect:SSLv3 read finished A
-----
Certificate chain ②
0 s:/description=ABjQuqt3nPv7ebEG/C=US
/CN=www.igvita.com/emailAddress=ilya@igvita.com
i:/C=IL/O=StartCom Ltd./OU=Secure Digital Certificate Signing
/CN=StartCom Class 1 Primary Intermediate Server CA
1 s:/C=IL/O=StartCom Ltd./OU=Secure Digital Certificate Signing
/CN=StartCom Class 1 Primary Intermediate Server CA
i:/C=IL/O=StartCom Ltd./OU=Secure Digital Certificate Signing
/CN=StartCom Certification Authority
-----
Server certificate
-----BEGIN CERTIFICATE-----
... snip ...
...
No client certificate CA names sent
SSL handshake has read 3571 bytes and written 444 bytes ③
...
New, TLSv1/SSLv2, Cipher is RC4-SHA
Server public key is 2048 bit
Secure Renegotiation IS supported
Compression: NONE
Expansion: NONE
SSL-Session:
Protocol : TLSv1
Cipher   : RC4-SHA
Session-ID: 269349C84A47B2EFA7 ... ④
Session-ID-ctx:
Master-Key: 1F5F5F33D50BE6228A ...
Key-Aggr.: None
Start Time: 1354837095
Timeout   : 300 (sec)
Verify return code: 0 (ok)
```

41

To verify and test your configuration, you should familiarize yourself with the openssl command-line interface, which will help you inspect the entire handshake and configuration of your server locally.

You can observe the following 4 steps when using the openssl command.

- Client completed verification of received certificate chain.
- Received certificate chain (two certificates).
- Size of received certificate chain.
- Issued session identifier for stateful TLS resume.

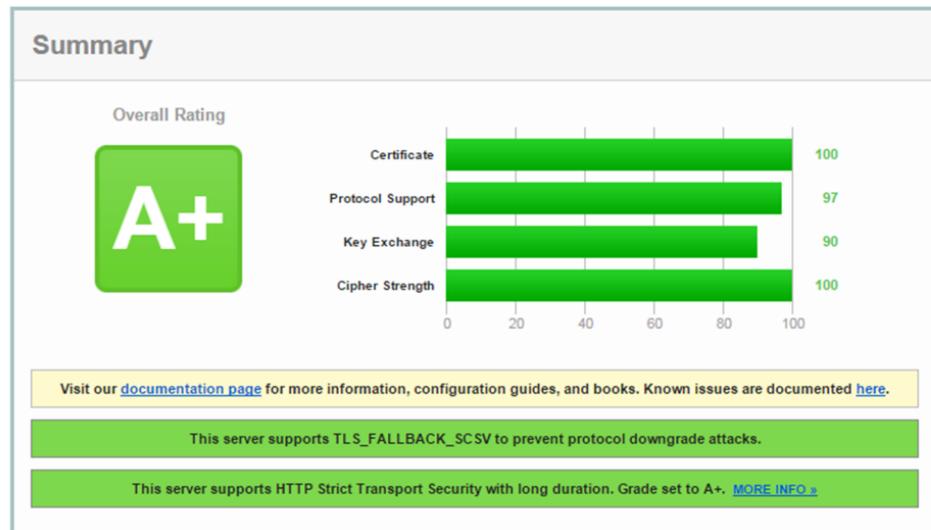
In the preceding example, we connect to igvita.com on the default TLS port (443), and perform the TLS handshake. Because the s_client makes no assumptions about known root certificates, we manually specify the path to the root certificate of StartSSL Certificate Authority—this is important. Your browser already has StartSSL’s root certificate and is thus able to verify the chain, but s_client makes no such assumptions. Try omitting the root certificate, and you will see a verification error in the log.

Inspecting the certificate chain shows that the server sent two certificates, which added up to 3,571 bytes, which is very close to the three- to four-segment initial TCP congestion window size. We should be careful not to overflow it or raise the cwnd size on the server. Finally, we can inspect the negotiated SSL session variables—chosen protocol, cipher, key—and we can also see that the server issued a session identifier for the current session, which may be resumed in the future.

■ Experimenting

● Free online scans

► Example: <https://www.ssllabs.com/ssltest/>



42

Additionally, you can use an online service, such as the <https://www.ssllabs.com/ssltest/> to scan your public server for common configuration and security flaws. It is well worth going to this site to evaluate the performance of a number of TLS connections.

■ Experimenting

- Free online scans
- Example:
<https://www.ssllabs.com/ssltest/>

Authentication

Server Key and Certificate #1	
Common names	deploy.innovativecodes.com
Alternative names	deploy.innovativecodes.com innovativecodes.com
Prefix handling	Not valid for "www.innovatecodes.com" - CONFLUENT
Valid from	Wed, 07 Oct 2016 09:07:27 UTC
Valid until	Thu, 06 Oct 2016 09:49:27 UTC (expires in 11 months and 29 days)
Key	RSA 2048 bits (e 65537)
Weak key (Dohlan)	No
Issuer	StarCom Class 1 Primary Intermediate Server CA
Signature algorithm	DH256withRSA
Extended Validation	No
Certificate Transparency	No
Revocation information	CRL, OCSP
Revocation status	Good (not revoked)
Trusted	Yes

Additional Certificates (if supplied)	
Certificates provided	2 (1134 by 96)
Chain issues	None

#2	
Subject	StarCom Class 1 Primary Intermediate Server CA
Fingerprint	0a3f9303ab05d00b05c05727a0019e7f9506
Valid until	Fri, 14 Oct 2022 20:54:17 UTC (expires in 7 years)
Key	RSA 2048 bits (e 65537)
Issuer	StarCom Certification Authority
Signature algorithm	DH256withRSA

Certification Paths

Path #1: Trusted	
1	Sent by server
	deploy.innovativecodes.com
	Fingerprint: e41fb0ca00b05d00b05c05727a0019e7f9506195af
	RSA 2048 bits (e 65537) / DH256withRSA
	StarCom Class 1 Primary Intermediate Server CA
	Fingerprint: 0a3f9303ab05d00b05c05727a0019e7f9506
	RSA 2048 bits (e 65537) / DH256withRSA
2	Sent by server
	StarCom Certification Authority - Self-signed
	Fingerprint: 30c0f22110905f0e6e0d949503c005476a0
	RSA 4096 bits (e 65537) / DH4096withRSA
	Weak or insecure signature, but no impact on root certificate

Path #2: Trusted	
1	Sent by server
	deploy.innovativecodes.com
	Fingerprint: e41fb0ca00b05d00b05c05727a0019e7f9506195af
	RSA 2048 bits (e 65537) / DH256withRSA
	StarCom Class 1 Primary Intermediate Server CA
	Fingerprint: 0a3f9303ab05d00b05c05727a0019e7f9506
	RSA 2048 bits (e 65537) / DH256withRSA
2	Sent by server
	StarCom Certification Authority - Self-signed
	Fingerprint: a3f3336c420b6c5e1e40d942940d0810d1a0
	RSA 4096 bits (e 65537) / DH4096withRSA

43

■ Experimenting

- **Free online scans**
- **Example:**
<https://www.ssllabs.com/ssltest/>

Configuration



Protocols

TLS 1.2	Yes
TLS 1.1	Yes
TLS 1.0	No
SSL 3	No
SSL 2	No



Cipher Suites (SSL 3+ suites in server-preferred order; deprecated and SSL 2 suites at the end)

TLS_ECDHE_RSA_WITH_AES_256_GCM_SHA384 (0xc030)	ECDH 256 bits (eq. 3072 bits RSA)	FS	256
TLS_ECDHE_RSA_WITH_AES_256_CBC_SHA (0xc028)	ECDH 256 bits (eq. 3072 bits RSA)	FS	256
TLS_ECDHE_RSA_WITH_AES_256_CBC_SHA (0xc014)	ECDH 256 bits (eq. 3072 bits RSA)	FS	256
TLS_DHE_RSA_WITH_AES_256_GCM_SHA384 (0x9f)	DH 2048 bits (p: 256, g: 1, Ys: 256)	FS	256
TLS_DHE_RSA_WITH_AES_256_CBC_SHA256 (0x6b)	DH 2048 bits (p: 256, g: 1, Ys: 256)	FS	256
TLS_DHE_RSA_WITH_AES_256_CBC_SHA (0x39)	DH 2048 bits (p: 256, g: 1, Ys: 256)	FS	256



Handshake Simulation

Android 2.3.7 <small>No SNI¹</small>	Protocol or cipher suite mismatch	<small>Fai²</small>
Android 4.0.4	Protocol or cipher suite mismatch	<small>Fai³</small>
Android 4.1.1	Protocol or cipher suite mismatch	<small>Fai³</small>
Android 4.2.2	Protocol or cipher suite mismatch	<small>Fai³</small>
Android 4.3	Protocol or cipher suite mismatch	<small>Fai³</small>
Android 4.4.2	TLS 1.2 TLS_ECDHE_RSA_WITH_AES_256_GCM_SHA384 (0xc030) FS	256
Android 5.0.0	TLS 1.2 TLS_ECDHE_RSA_WITH_AES_256_CBC_SHA (0xc014) FS	256
Baidu Jan 2015	Protocol or cipher suite mismatch	<small>Fai³</small>
BingPreview Jan 2015	TLS 1.2 TLS_ECDHE_RSA_WITH_AES_256_GCM_SHA384 (0xc030) FS	256
Chrome 43 / OS X R	TLS 1.2 TLS_ECDHE_RSA_WITH_AES_256_CBC_SHA (0xc014) FS	256
Firefox 31.0 ESR / Win 7	TLS 1.2 TLS_ECDHE_RSA_WITH_AES_256_CBC_SHA (0xc014) FS	256
Firefox 39 / OS X R	TLS 1.2 TLS_ECDHE_RSA_WITH_AES_256_CBC_SHA (0xc014) FS	256
Googlebot Feb 2015	TLS 1.2 TLS_ECDHE_RSA_WITH_AES_256_CBC_SHA (0xc014) FS	256
IE 6 / XP No FS¹ No SNI²	Protocol or cipher suite mismatch	<small>Fai³</small>

■ Experimenting

- Free online scans
- Example:
<https://www.ssllabs.com/ssltest/>

Protocol Details	
Secure Renegotiation	Supported
Secure Client-Initiated Renegotiation	No
Insecure Client-Initiated Renegotiation	No
BEAST attack	Mitigated server-side (more info)
POODLE (SSLv3)	No, SSL 3 not supported (more info)
POODLE (TLS)	No (more info)
Downgrade attack prevention	Yes, TLS_FALLBACK_SCSV supported (more info)
SSL/TLS compression	No
RC4	No
Heartbeat (extension)	Yes
Heartbleed (vulnerability)	No (more info)
OpenSSL CCS vuln. (CVE-2014-0224)	No (more info)
Forward Secrecy	Yes (with most browsers) ROBUST (more info)
Next Protocol Negotiation (NPN)	No
Session resumption (caching)	Yes
Session resumption (tickets)	Yes
OCSP stapling	No
Strict Transport Security (HSTS)	Yes max-age=63072000; includeSubDomains
Public Key Pinning (HPKP)	No
Long handshake intolerance	No
TLS extension intolerance	No
TLS version intolerance	No
Incorrect SNI alerts	No
Uses common DH primes	No
DH public server param (Ys) reuse	No
SSL 2 handshake compatibility	Yes

Miscellaneous	
Test date	Wed, 07 Oct 2015 07:13:56 UTC
Test duration	81.956 seconds
HTTP status code	200
HTTP server signature	Apache
Server hostname	ec2-54-186-234-141.us-west-2.compute.amazonaws.com

- Network model
- Secure configuration of devices
- Exchanging keys
- Secure networking protocols
 - Transport layer: TLS & SSL
 - Network layer: IPSec & VPN
 - Data link layer: WEB & WPA
- Firewalls

■ This work contains content adapted from, amongst others, the following sources (in no particular order)

● Books

- ▶ William Stallings, “**Cryptography and Network Security, principles and practices**”, 6th (international) edition, Prentice Hall, 2010;
- ▶ Matt Bishop, “**Computer Security: Art and Science**”, Addison Wesley, Pearson Education, 2003, ISBN-13: 978-0-201-44099-7

● Lecture slides:

- ▶ “Informatiebeveiliging”, Universiteit Gent, Eric Laermans & Thom Dhaene
- ▶ Stallings, 2014, Lecture slides by Lawrie Brown

● Wikipedia

- ▶ Note page descriptions

● Websites

- ▶ <http://chimera.labs.oreilly.com/books/1230000000545/ch04.html>
- ▶ <http://ipseclab.eit.lth.se/tiki-index.php?page=3.+SSL+and+TLS>
- ▶ <http://www.ciscopress.com/articles/index.asp?st=42081>

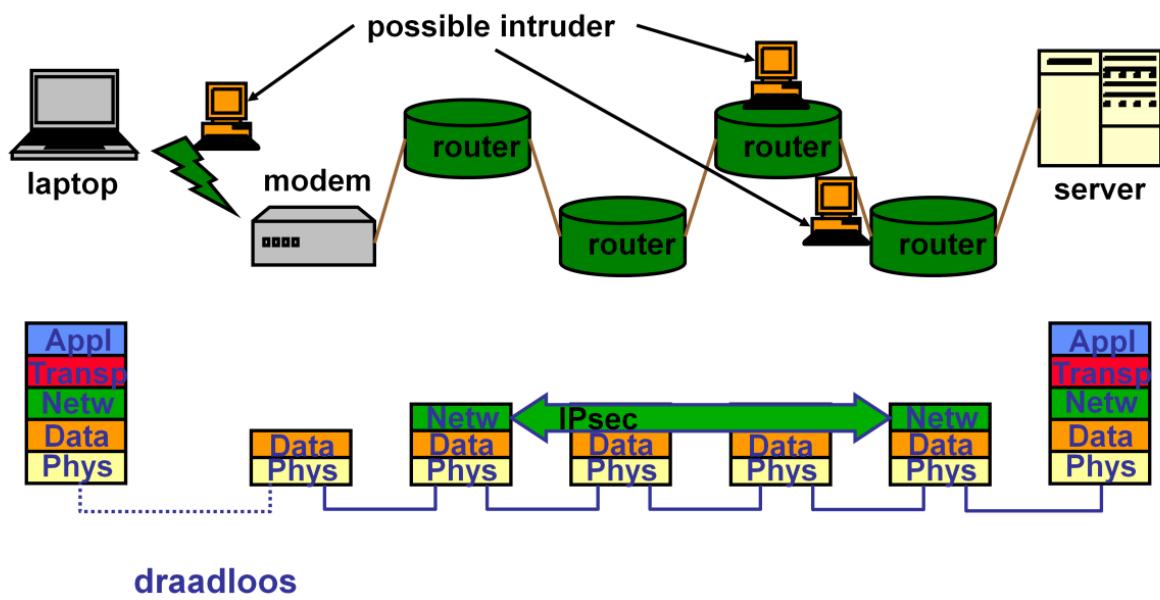


Network and Computer Security

Chapter 3 – Network and communication security

Prof. dr. ir. Eli De Poorter

- Network model
- Secure configuration of devices
- Exchanging keys
- Secure networking protocols
 - Transport layer: TLS & SSL
 - Network layer: IPSec & VPN
 - Data link layer: WEB & WPA
- Firewalls



■ IP security?

- source spoofing
- replay packets
- no data integrity or confidentiality

- 
- DOS attacks
 - Replay attacks
 - Spying
 - and more...

■ IPsec

- Secure IP connections
- Application independent!

■ Routing applications

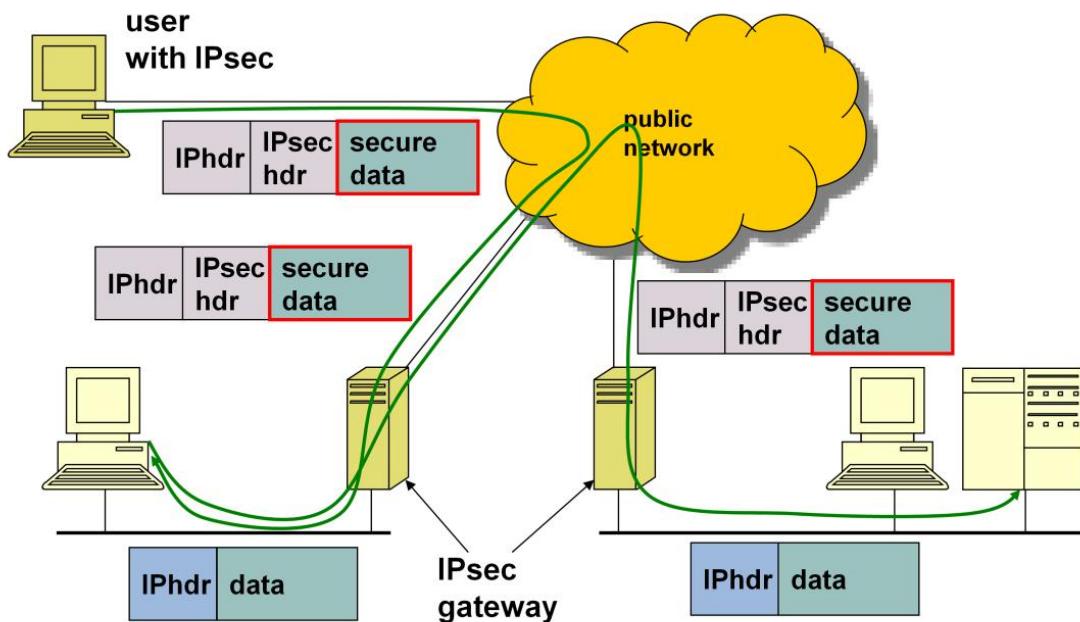
- **Authentication of routing messages**
 - ▶ Advertisements
 - ▶ Updates
 - ▶ etc.
- **Without IPsec forged routing information could be sent**

■ LAN-to-LAN

- **VPN (virtual private network) for a company:**
 - ▶ Enabling secure communication over public networks between geographically disseminated company locations
- **Gateway security**
 - ▶ Behaves as if subnetworks were connected using an ordinary LAN (inaccessible to outer world) instead of a public network

■ Client-to-LAN

- **Secure LAN access over the Internet**
- **Remote access to a trusted source from an untrusted network**
- **Often also called a VPN**



■ Advantages

- **Application independent**
 - ▶ Transparent for application layer
 - ▶ Provides security to applications that don't provide security themselves
- **Security mechanisms limited to a few access points**
 - ▶ When implemented on firewall or router
 - ▶ Offers strong security for all traffic crossing perimeter, without burdening the internal traffic
- **Possibly transparent to end users**
 - ▶ End users (almost) needn't worry about security services
- **Individual user security is possible**
 - ▶ OK for teleworkers

■ Drawbacks

- **No message security beyond secure gateway**
 - ▶ E.g. at mail server
 - ▶ No secure storage
- **Use of system resources**
 - ▶ Computation time required for cryptographic functions
- **Complexity of specification**

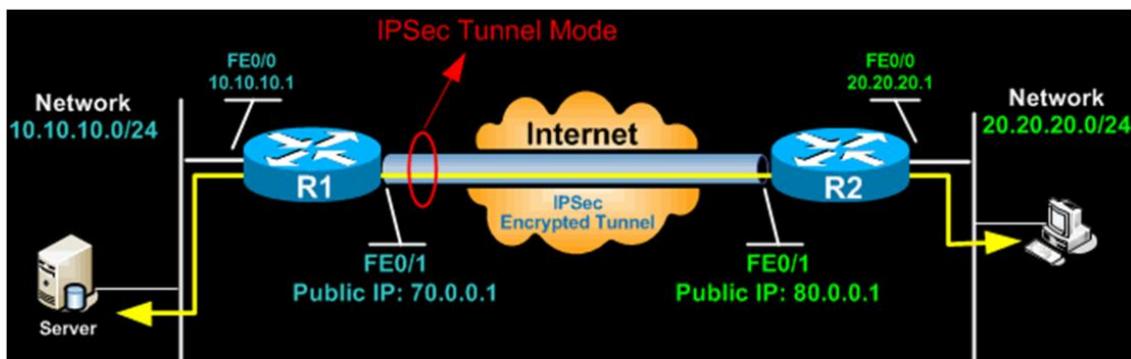
■ Achievable security services

- **Access control**
- **Connectionless integrity**
- **Data origin authentication**
- **Averting replay attacks**
- **Confidentiality**
- **Limited traffic flow confidentiality**

■ IPsec can operate in two modes of operation

• (1) Layer 2 tunnel mode (default mode)

- ▶ Typically used to tunnel IP traffic between two security gateways
- ▶ IPsec protects the full IP datagram (including IP headers)
 - ✓ New IP header is created
- ▶ Automatic NAT traversal



10

IPsec can be implemented in a host-to-host transport mode, as well as in a network tunneling mode.

Tunnel mode

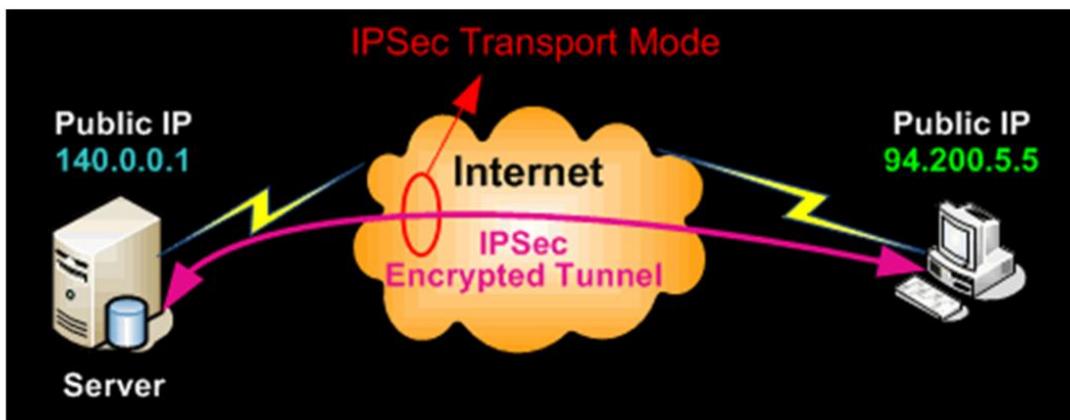
IPSec tunnel mode is the **default mode**. The tunnel mode protects any internal routing info by encrypting the IP header of the ENTIRE packet. This means IPsec wraps the original packet, encrypts it, adds a new IP header and sends it to the other side of the VPN tunnel (IPsec peer). As such, the entire IP packet is encrypted and/or authenticated. Since additional headers are added to the packet there is less space available for payload.

Tunnel mode is used to create virtual private networks for network-to-network communications (e.g. between routers to link sites), host-to-network communications (e.g. remote user access) and host-to-host communications (e.g. private chat). NAT traversal is supported with the tunnel mode. In the example, the client connects to the IPsec Gateway. Traffic from the client is encrypted, encapsulated inside a new IP packet and sent to the other end. Once decrypted by the firewall appliance, the client's original IP packet is sent to the local network.

■ IPsec can operate in two modes of operation

• (2) Transport mode

- ▶ Only the payload is encrypted / encapsulated
 - ✓ IPsec transport mode offers limited protection to IP headers
- ▶ Mainly used to provide security services for upper layer protocols



11

Transport mode

In transport mode, only the payload of the IP packet (and the ESP trailer) is usually encrypted and/or authenticated. The IP header of the original packet is neither modified nor encrypted, thus leaving the routing intact. The transport and application layers are always secured by hash, so they cannot be modified in any way (for example by translating the port numbers). Transport mode is implemented for client-to-site VPN scenarios.

IPSec Transport mode is most commonly used for end-to-end communications between devices with public IP addresses, for example for communication between a client and a server or between a workstation and a gateway (if the gateway is being treated as a host and is thus the actual destination). A good example would be an encrypted Telnet or Remote Desktop session from a workstation to a server. By default, NAT traversal **IS NOT** supported with the transport mode, but a means to encapsulate IPsec messages for NAT traversal has been defined by RFC documents describing the NAT-T mechanism.

IPSec transport mode is also used when another tunneling protocol (like GRE) is used to first encapsulate the IP data packet, then IPsec is used to protect the GRE tunnel packets. IPsec protects the GRE tunnel traffic in transport mode.



- **IPSec consists of two main protocols:**
 - **Authentication Header (AH)**
 - **Encapsulating Security Payload (ESP)**

- **Tunnel or transport mode can be implemented using the AH, ESP protocol or a combination of both**
 - **Both modes can provide data-integrity, authentication and/or confidentiality, depending on the choice of the protocols**

■ Authentication Header (AH) [RFC2402]

- **Supports data integrity and authentication of IP packets**
 - ▶ Enables router/workstation to authenticate user or application
 - ▶ Prevents IP spoofing
 - ▶ Contains anti-replay mechanism
- **MAC based authentication**
 - ▶ Requires shared secret key
 - ▶ No over mutable fields (e.g. TTL)

13

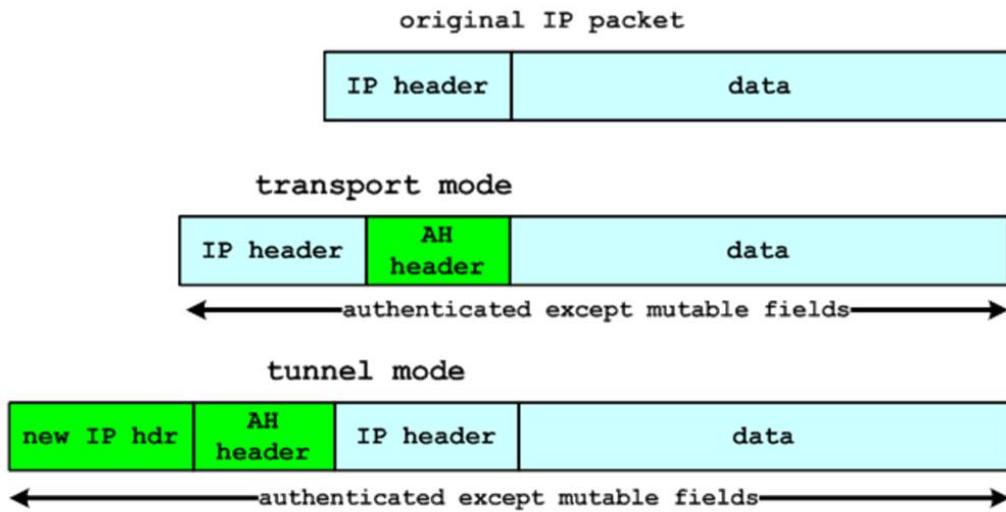
The *Authentication Header* (AH) provides authentication and integrity to the datagrams passed between two systems. This is achieved by applying a keyed one-way hash function to the datagram to create a message digest. If any part of the datagram is changed during transit, this will be detected by the receiver when it performs the same one-way hash function on the datagram and compares the value of the message digest that the sender has supplied. The fact that the one-way hash also involves the use of a secret shared between the two systems means that authenticity can be guaranteed.

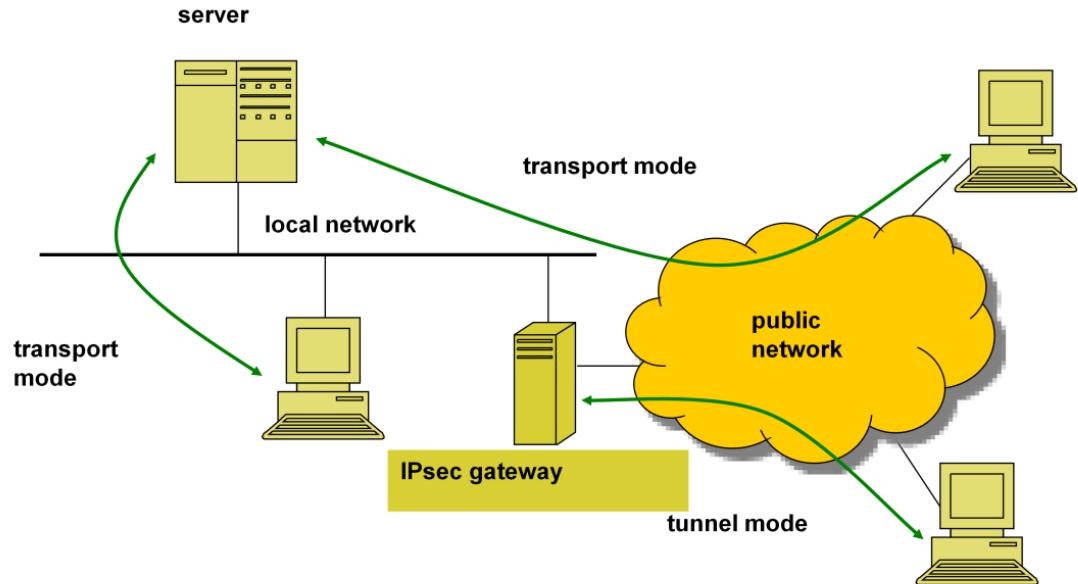
The AH function is applied to the entire datagram except for any mutable IP header fields that change in transit, such as Time To Live (TTL) fields that are modified by the routers along the transmission path. AH works as follows:

- The IP header and data payload is hashed.
- The hash is used to build a new AH header, which is appended to the original packet.
- The new packet is transmitted to the IPSec peer router.
- The peer router hashes the IP header and data payload, extracts the transmitted hash from the AH header, and compares the two hashes. The hashes must match exactly. If even one bit is changed in the transmitted packet, the hash output on the received packet will change and the AH header will not match.

■ Authentication header

- Used for both transport and tunnel modes





■ **ESP (“Encapsulating Security Payload”) [RFC2406]**

- **Achieves confidentiality function**

- ▶ And to some extent traffic flow confidentiality
- ▶ Uses traditional encryption algorithms
 - ✓ AES-128-CBC (MUST), AES-GCM , DES (obsolete)

- **Achieves (optional) authentication**

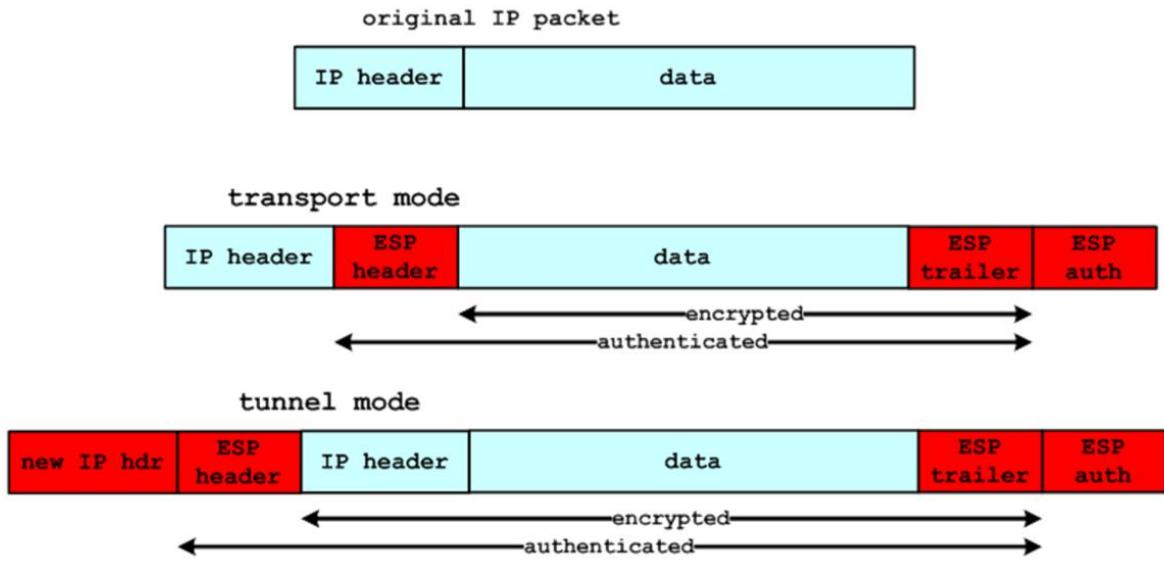
- ▶ Same principle as for AH
 - ✓ HMAC-SHA-1-96, AES-GMAC
 - ✓ “Optional” support for other MACs

16

Encapsulating Security Payload (ESP) is a security protocol used to provide confidentiality (encryption), data origin authentication, integrity, optional anti-replay service, and limited traffic-flow confidentiality by defeating traffic-flow analysis. The data payload is encrypted with ESP.

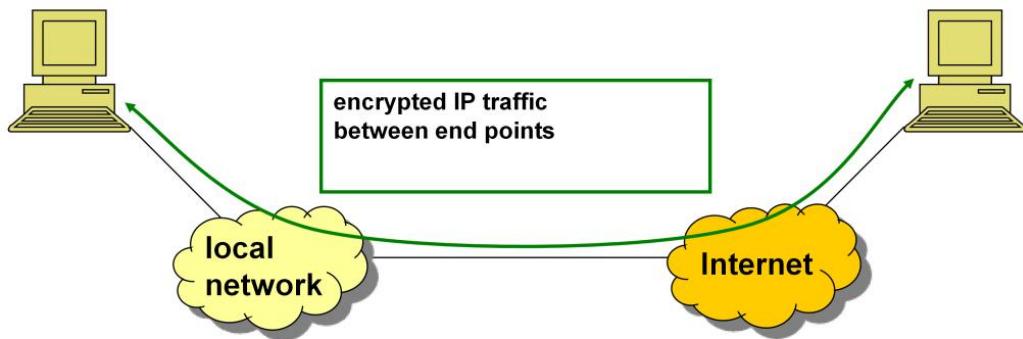
■ ESP header

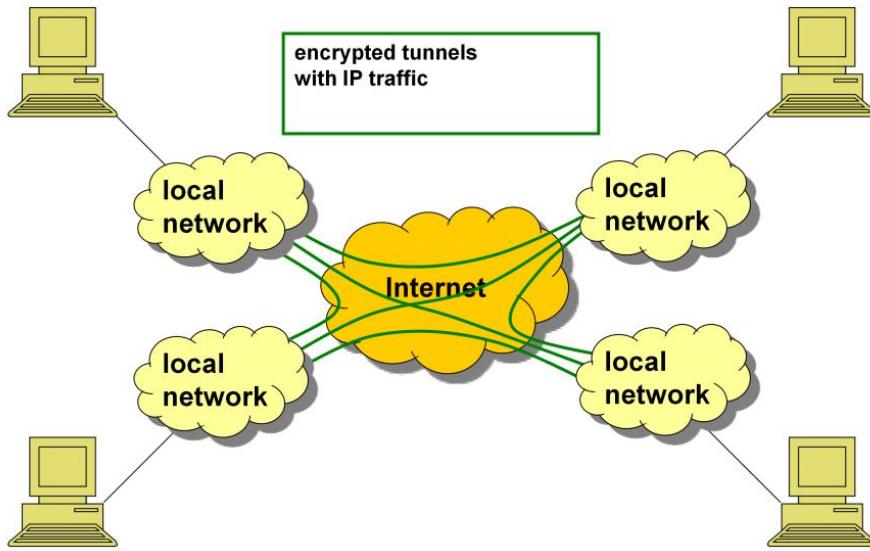
- Used for both transport and tunnel modes



■ ESP: transport mode

- From end point to end point:
 - ▶ Encryption (and possibly authentication)
 - ▶ No traffic flow confidentiality
 - ▶ E.g. for teleworking





p. 19

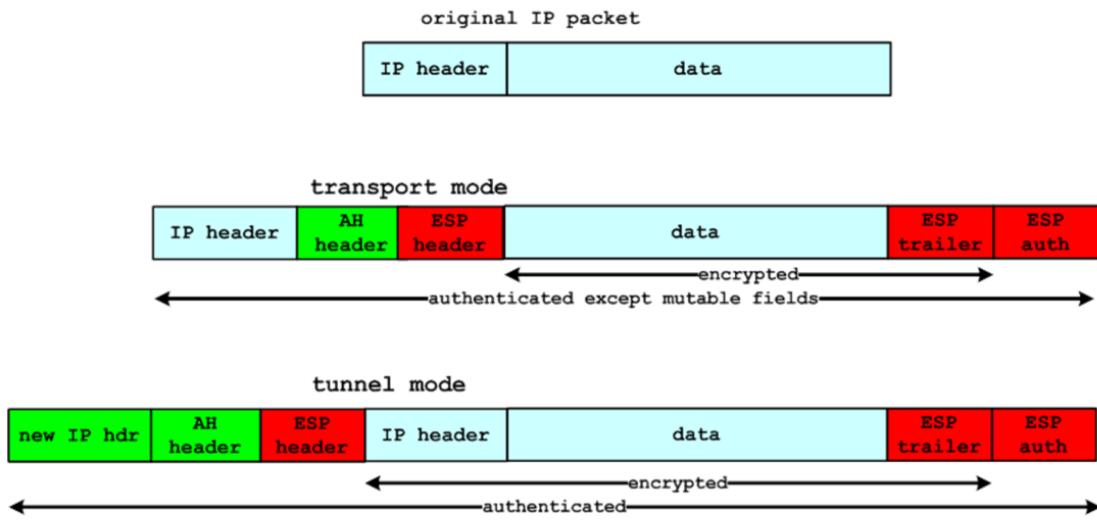
The tunnel mode can prove useful if we want to create a VPN between different locations of a company. Not every single connected device requires IPsec capabilities. Only a security gateway, which connects the (reputedly safe) local network to the Internet, requires IPsec capabilities. In this case, the local traffic will not use IPsec, while outgoing traffic (to some other company location) will be encrypted by the security gateway. In this way, possibly confidential data will be transmitted securely over the Internet from one location to another.

This configuration allows some traffic flow confidentiality, as an attacker outside one of the local networks can only observe between which local networks the traffic is flowing, but not between which workstations precisely.

In practice, the security gateway will also contain a firewall to achieve other security services.

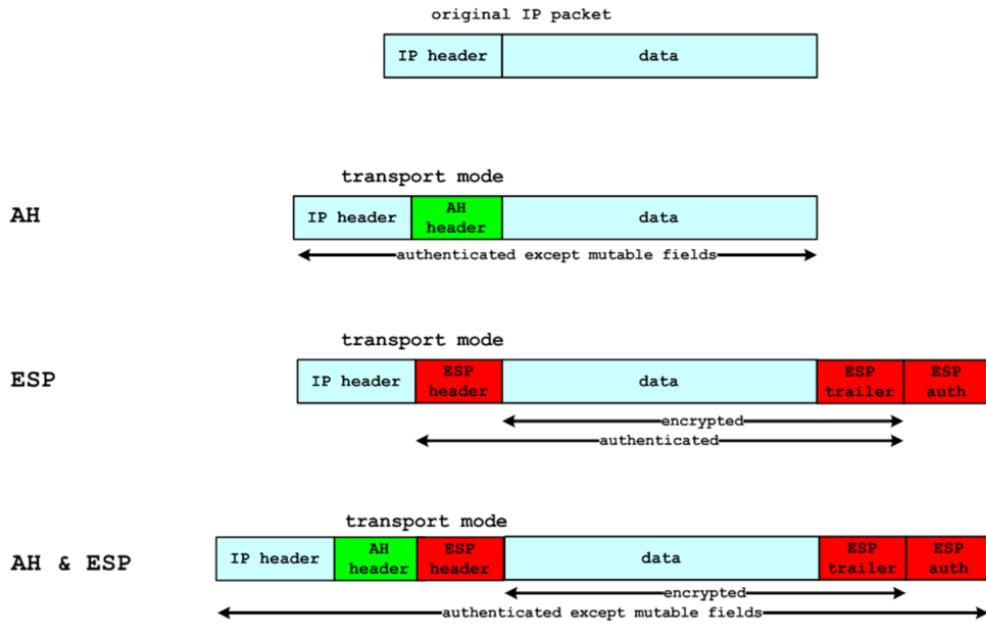
■ ESP + AH header

- Used for both transport and tunnel modes



■ IPsec transport mode summary

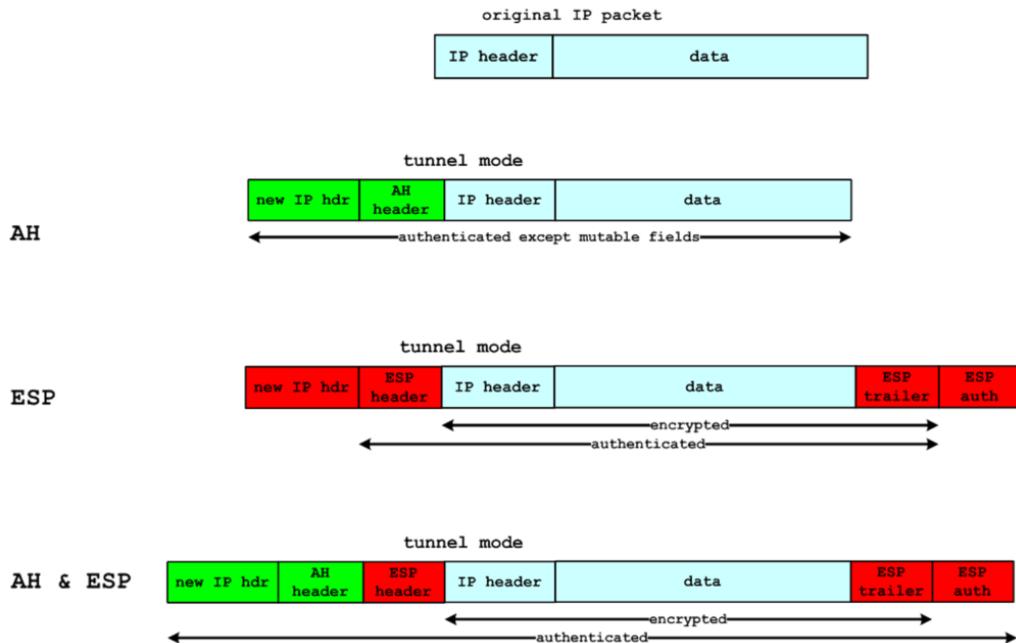
- Different AH, ESP or AH+ESP combinations



21

■ IPsec tunnel mode summary

- Different AH, ESP or AH+ESP combinations



22

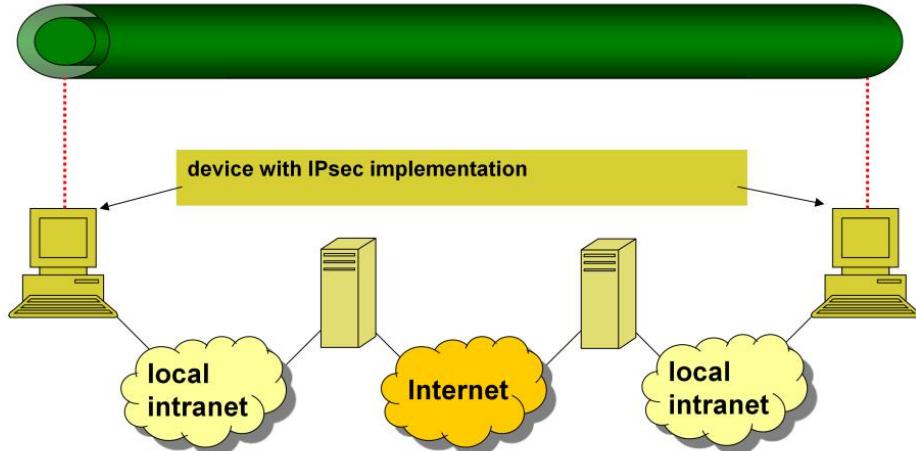
■ Security associations (SA)

- **One-way relationship between sender and receiver**
 - ▶ Provides security services to traffic carried
 - ▶ Two associations needed for bidirectional traffic
- **Identified by**
 - ▶ SPI ("Security Parameters Index")
 - ▶ IP destination address (end user, firewall, router, etc.)
 - ▶ Security protocol identification (AH or ESP)
 - ▶ Obvious bundle: AH + ESP for single IP stream

■ Combining tunnels

- **Not necessarily identical end points for each tunnel**
- **Several levels possible**
- **"Iterated tunneling"**

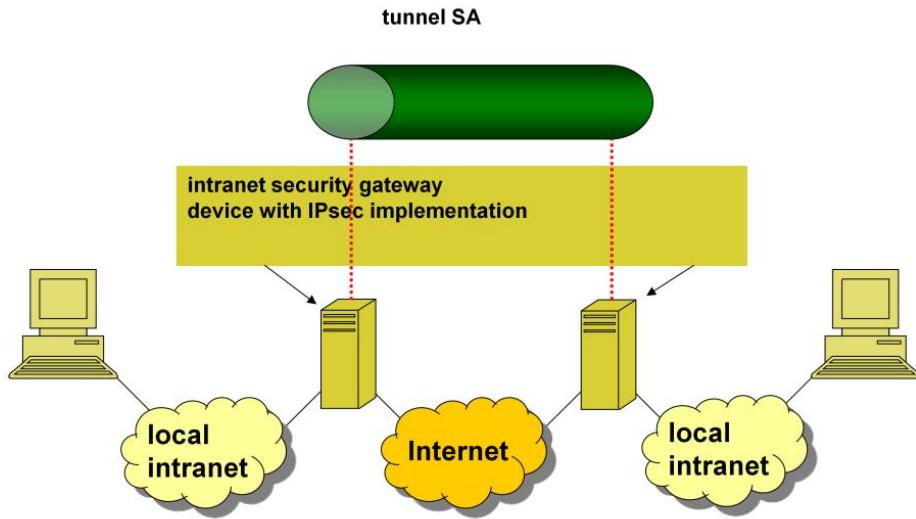
1 or more SAs



p. 24

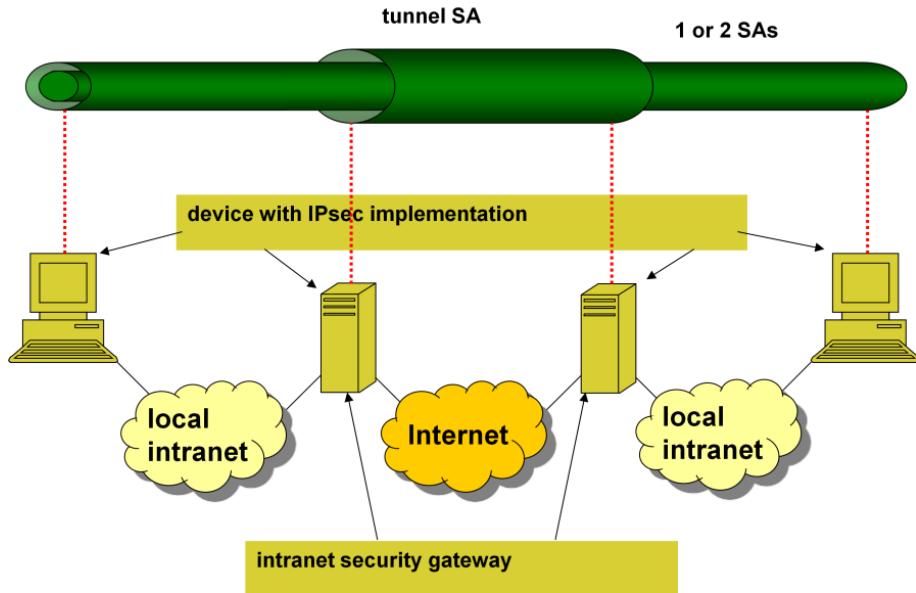
Security between two end points. Possible combinations:

- AH in transport mode
- ESP in transport mode
- AH, followed by ESP in transport mode (an ESP SA within an AH SA)
- one of the previous within an AH or ESP in tunnel mode



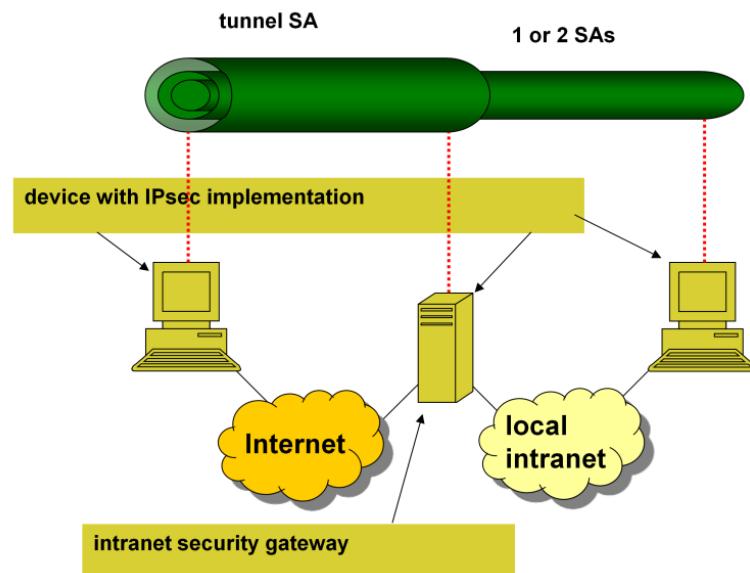
p. 25

Traffic secured between gateways (routers, firewall, etc.). No IPsec implementation at end users. Basic support for VPN. IPsec specifies that a single tunnel (with AH, ESP, or ESP with authentication) is sufficient in this case. Nested tunnels aren't required as the IPsec service applies to the entire original packet.



p. 26

Extension of case 2, offering security between the end points too. The tunnel offers authentication and/or confidentiality for traffic between intranets. Additional IPsec security may be added by the end users for the traffic within the local intranets.



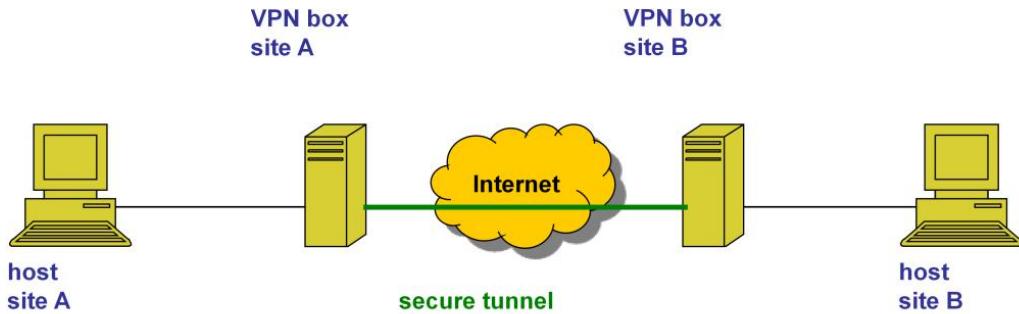
p. 27

This is the scenario for an external user (e.g. teleworker) using the Internet to access the security gateway of the company (secured by the tunnel SA) and further on a server or workstation within the company (internal communication secured by the internal SAs).

- Network model
- Secure configuration of devices
- Exchanging keys
- Secure networking protocols
 - Transport layer: TLS & SSL
 - Network layer: IPSec & VPN
 - Data link layer: WEB & WPA
- Firewalls

■ Basic principle of VPN

- **VPN = secure tunnel from site to site**
 - ▶ Not just secure remote login



■ Options

- **IPsec**
- **PPTP**
 - ▶ Point-to-Point Tunneling Protocol (Microsoft specific)
- **L2TP**
 - ▶ Layer 2 Tunneling Protocol (Microsoft + CISCO)
- **SSL/TLS**
- **SSH**
- **SSTP**
 - ▶ Secure Socket Tunneling Protocol
 - ▶ Windows only
-

■ Advantages

- Designed for this purpose
- Part of several other solutions

■ Disadvantages

- Complexity of IPsec
 - ▶ Especially key exchange (using IKE)
- Interoperability issues between different (partial) implementations of the standard
- Issues with NAT and (non-IPsec) firewalls
 - ▶ Mainly for AH



■ Point-to-Point Tunneling Protocol

■ Advantages

- Client built-in to just about all platforms
- Very easy to set up
- Fast

■ Disadvantages

- Not at all secure (the vulnerable MS CHAPv2 authentication is still the most common in use)
- Definitely compromised by the NSA

- Build on the OpenSSL library using the SSL/TLS protocols
- Advantages
 - Highly configurable
 - Very secure (probably even against the NSA)
 - Can bypass firewalls
 - Can use a wide range of encryption algorithms
 - Open source (and can therefore be readily vetted for back doors and other NSA style tampering)
- Disadvantages
 - Needs third party software
 - Can be fiddly to set up
 - Support on mobile devices is improving, but is not as good as on the desktop
 - Security on top of transport layer
 - ▶ Instead of network layer security

■ Layer 2 Tunnel Protocol (L2TP)

- On its own no encryption or confidentiality
- Therefore combines with IPsec

■ Advantages

- Usually considered very secure
- Easy to set up
- Available on all modern platforms

■ Disadvantages

- May be compromised by the NSA
- Likely deliberately weakened by the NSA
- Slower than OpenVPN
- Can struggle with restrictive firewalls

- Network model
- Secure configuration of devices
- Exchanging keys
- Secure networking protocols
 - Transport layer: TLS & SSL
 - Network layer: IPSec & VPN
 - Data link layer: WEB & WPA
- Firewalls

■ This work contains content adapted from, amongst others, the following sources (in no particular order)

● Books

- ▶ William Stallings, “**Cryptography and Network Security, principles and practices**”, 6th (international) edition, Prentice Hall, 2010;
- ▶ Matt Bishop, “**Computer Security: Art and Science**”, Addison Wesley, Pearson Education, 2003, ISBN-13: 978-0-201-44099-7

● Lecture slides:

- ▶ “Informatiebeveiliging”, Universiteit Gent, Eric Laermans & Thom Dhaene
- ▶ Stallings, 2014, Lecture slides by Lawrie Brown

● Wikipedia

- ▶ Note page descriptions

● Websites

- ▶ <http://www.ciscopress.com/articles/index.asp?st=42081>
- ▶ <http://www.deepsh.it/networking/IPSec.html>
- ▶ <http://www.firewall.cx/networking-topics/protocols/870-ipsec-modes.html>



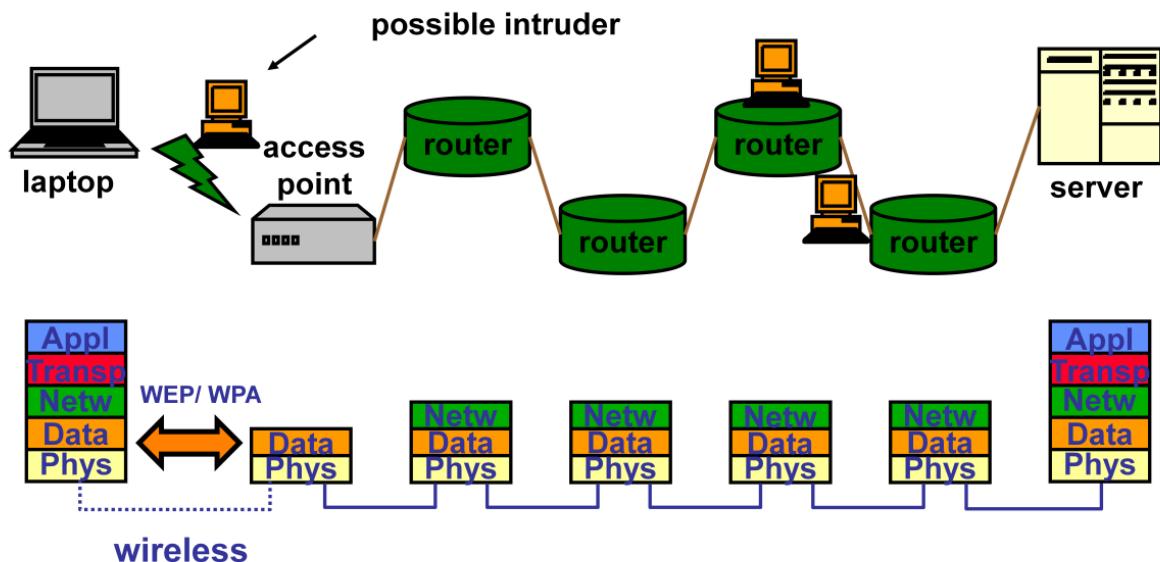
Network and Computer Security

Chapter 3 – Network and communication security

Prof. dr. ir. Eli De Poorter



- Network model
- Secure configuration of devices
- Exchanging keys
- Secure networking protocols
 - Transport layer: TLS & SSL
 - Network layer: IPSec & VPN
 - Data link layer: WEP & WPA
- Firewalls





■ Common attacks

- **Content Address Memory (CAM) table exhaustion attack**
 - ▶ Through flooding, fill the CAM table that stores which device should be contacted through each switch port
 - ▶ Device defaults to broadcasting
 - ▶ Turns a switch into a hub
- **Address Routing Protocol (ARP) spoofing**
 - ▶ Data link layer is responsible for mapping logical (IP) to physical (MAC) addresses
 - ▶ Broadcast spoofed IP packets
- **Dynamic Host Configuration Protocol (DHCP) starvation**
 - ▶ Continue sending DHCP requests

4

The network interface layer, commonly referred to as the data link layer, is the physical interface between the host system and the network hardware. It defines how data packets are to be formatted for transmission and routings. Some common link layer protocols include IEEE 802.2 and X.25. The data link layer and its associated protocols govern the physical interface between the host computer and the network hardware. The goal of this layer is to provide reliable communications between hosts connected on a network.

■ Example data layer security protocols

- **CHAP**
- **PPTP**
- **L2F**
- **ECP**
- **EAP**

■ Additional attacks in wireless networks

- **Packet overhearing**
 - ▶ Shared medium
- **Deauth (deauthentication) attack**
 - ▶ Send spoofed deauthentication messages
- **Hidden node attacks**
 - ▶ See next slide

6

Packet overhearing

Due to the shared nature of the wireless medium, all packets can be overheard

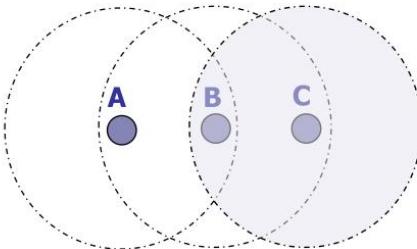
Hidden node attacks

In a wireless network many hosts or nodes are sharing a common medium. If nodes A and B are both wireless laptop computers communicating in an office environment their physical separation may require that they communicate through a wireless access point. But only one device can transmit at a time in order to avoid packet collisions. Prior to transmitting, Node A sends out a Ready to Send (RTS) signal. If it is not receiving any other traffic the access point will broadcast a Clear to Send (CTS) signal over the network. Node A will then begin transmitting while Node B knows to hold off transmitting its data for the time being. Even though it cannot directly communicate with Node A, i.e. Node A is hidden, it knows to wait based on its communication with the access point. An attacker can exploit this functionality by flooding the network with CTS messages. Then every node assumes there is a hidden node trying to transmit and will hold its own transmissions, resulting in a denial of service. Preventing hidden node attacks requires a network tool. Such a tool monitors access point traffic and develops a baseline level of traffic. Any spikes in CTS/RTS signals are assumed to be the result of a hidden node attack and are subsequently blocked.

Deauth (deauthentication) attack

Any client entering a wireless network must first authenticate with an access point (AP) and is thereafter associated with that access point. When the client leaves it sends a deauthentication, or deauth, message to disassociate itself with the access point. An attacker can send deauth messages to an access point tied to client IP addresses thereby knocking the users off-line and requiring continued re-authentication, giving the attacker valuable insight into the reauthentication handshaking that occurs. To mitigate this attack, the access point can be set up to delay the effects of deauthentication or disassociation requests (e.g., by queuing such requests for 5–10 seconds) thereby giving the access point an opportunity to observe subsequent packets from the client. If a data packet arrives after a deauthentication or disassociation request is queued, that request is discarded since a legitimate client would never generate packets in that order

■ Hidden node problem



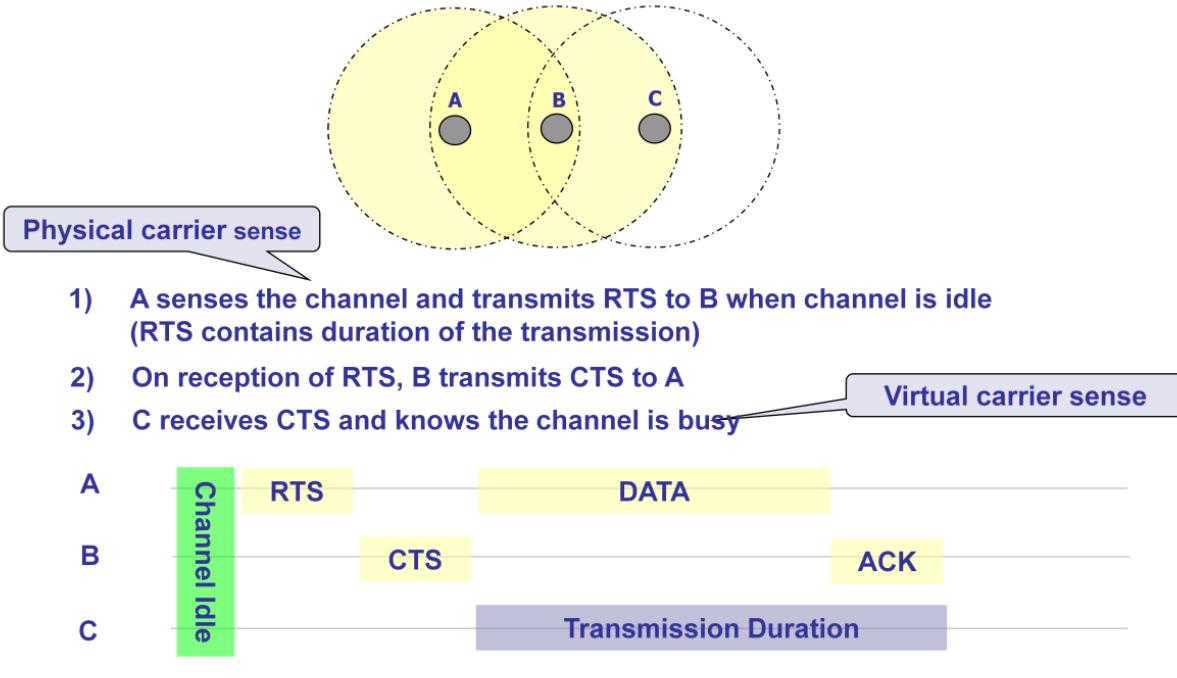
- 1) A senses the channel (CS) and transmits to B when channel is idle
- 2) C cannot detect the transmission of A and thinks the channel is idle (CS fails)
- 3) C transmits and a collision occurs at B
(CD fails at A → A is hidden for C)

→ Reduced throughput

7

Consider the scenario with three wireless devices as shown in the figure. The transmission range of A reaches B, but not C (the detection range does not reach C either). The transmission range of C reaches B, but not A. Finally, the transmission range of B reaches A and C, i.e., A cannot detect C and vice versa. A starts sending to B, C does not receive this transmission. C also wants to send something to B and senses the medium. The medium appears to be free, the carrier sense fails. C also starts sending causing a collision at B. But A cannot detect this collision at B and continues with its transmission. A is hidden for C and vice versa.

■ RTS/CTS (Request To Send, Clear To Send)



8

This slide shows the same scenario as previously shown in the slide on **hidden terminals**. Remember, A and C both want to send to B. A has already started the transmission, but is hidden for C, C also starts with its transmission, thereby causing a collision at B.

A frequently used solution is to ensure A does not start its transmission at once, but sends a **request to send (RTS)** first. B receives the RTS that contains the name of sender and receiver, as well as the length of the future transmission. This RTS is not heard by C, but triggers an acknowledgement from B, called **clear to send (CTS)**. The CTS again contains the names of sender (A) and receiver (B) of the user data, and the length of the future transmission. This CTS is now heard by C and the medium for future use by A is now reserved for the duration of the transmission. After receiving a CTS, C is not allowed to send anything for the duration indicated in the CTS toward B. A collision cannot occur at B during data transmission, and the hidden terminal problem is solved – provided that the transmission conditions remain the same. (Another station could move into the transmission range of B after the transmission of CTS.) Still, **collisions can occur** during the sending of an RTS. Both A and C could send an **RTS that collides** at B. RTS is very small compared to the data transmission, so the probability of a collision is much lower. B resolves this contention and acknowledges only one station in the CTS (if it was able to recover the RTS at all). No transmission is allowed without an appropriate CTS.



- Need for additional security mechanisms
- WLAN security solutions
 - **Wired Equivalent Privacy (WEP):**
 - ▶ Part of the original 802.11 standard. No key management, also several other weaknesses.
 - **WiFi Protected Access (WPA):**
 - ▶ Interim solution offers key management using the 802.1X authentication framework, plus improved encryption and integrity checking.
 - **IEEE 802.11i (WPA2):**
 - ▶ Same as WPA, except improved encryption (AES).

- IEEE 802.11 specifies **as an option** the use of WEP which can take care of the following security mechanisms:
 - Authentication ("shared key" user authentication)
 - Confidentiality (RC4 stream cipher encryption)
 - Integrity checking (CRC-32 integrity mechanism)

- It lacks
 - key management
 - protection against replay attacks

■ No key management in WEP



No key management in WEP \Leftrightarrow every wireless station and AP has the same "preshared" key that is used during authentication and encryption. This key is distributed manually (\Rightarrow insufficient for enterprise applications).



■ Problems with preshared keys

Manual key management is not very flexible

Same key for everybody:

In a large network, users may wish to have independent secure connections. Just a single non-honest WLAN user can break the security.

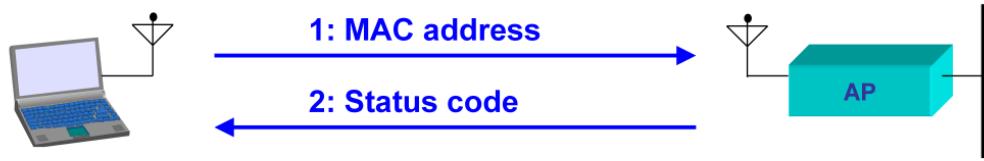
Static key:

Since it is relatively easy to crack WEP encryption in a reasonably short time, the keys should be changed often, but the preshared key concept does not support this.

■ WLAN authentication methods

- **Open system authentication (specified in WEP)**
 - ▶ actually no authentication at all
- **Shared key authentication (specified in WEP)**
 - ▶ weak due to non-existing key management
- **Authentication using SSID of AP**
- **MAC address filtering**
- **IEEE 802.1X authentication (specified in WPA)**
- **SIM/AuC authentication (in operator-based network)**

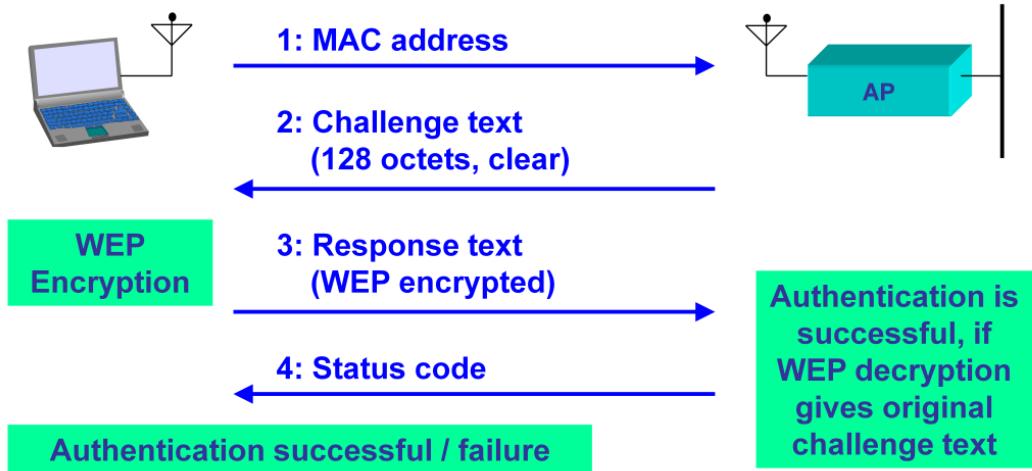
■ Open system authentication



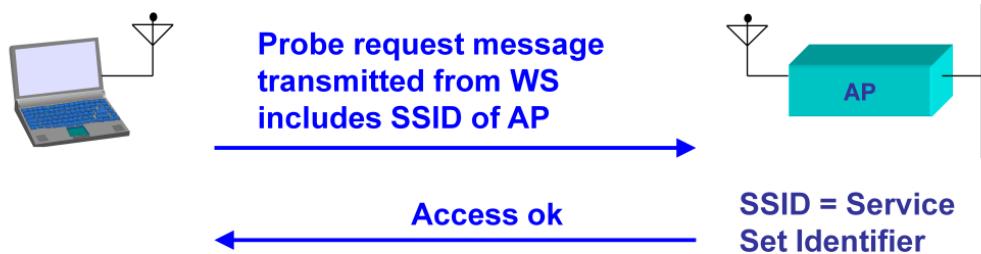
Status codes are defined in IEEE 802.11

Status code	Meaning
0	Successful
1	Unspecified failure
:	:
15	Authentication rejected (cause x)
:	:

■ Shared-key authentication

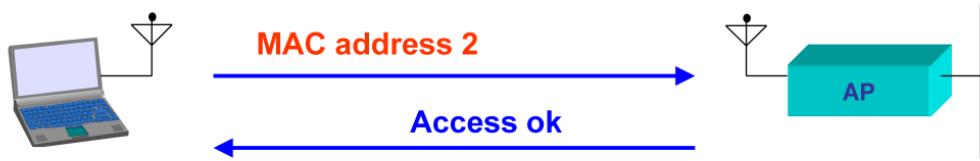


■ Authentication using SSID of AP



Not very secure: SSID is transmitted unencrypted over the wireless network and can be easily captured by an attacker.

■ MAC address filtering



Accepted MAC addresses:

MAC address 1
MAC address 2
MAC address 3
MAC address 4
:

Not very secure: Attacker can read MAC address of a wireless station attached to the WLAN and replace own MAC address with this stolen MAC address.

■ Small and static keys :

- Actual keyspace is 40 bits
- No easy way to exchange and distribute keys.
- Key change involves manually changing the key on each AP and Client.

■ Use of small, plaintext initialization vector (IV)

- IV is sent out in clear text usually at the starting of the packet.
- Dictionary of IVs and keystreams
 - ▶ Only 2^{24} possibilities
 - ▶ Can be stored in 24GB disk space
 - ▶ Can be intercepted in a very short period of time on high traffic wireless networks.

■ Weak encryption algorithms

■ Small and static keys :

- Actual keyspace is 40 bits
- No easy way to exchange and distribute keys.
- Key change involves manually changing the key on each AP and Client.

■ Use of small, plaintext initialization vector (IV)

- IV is sent out in clear text usually at the starting of the packet.
- Dictionary of IVs and keystreams
 - ▶ Only 2^{24} possibilities
 - ▶ Can be stored in 24GB disk space
 - ▶ Can be intercepted in a very short period of time on high traffic wireless networks.

■ Weak encryption algorithms

■ Cracking the WEP key

- Attacker sets NIC drivers to Monitor Mode
- Begins capturing packets with Airsnort
- Airsnort quickly determines the SSID
- Sessions can be saved in Airsnort, and continued at a later date so you don't have to stay in one place for hours
- A few 1.5 hour sessions yield the encryption key
- Once the WEP key is cracked and his NIC is configured appropriately, the attacker is assigned an IP, and can access the WLAN

■ WLAN security using WPA

- Precursor of IEEE 802.11i

■ Features

- Key management (using the 802.1X framework, it is also possible to use preshared keys)
- Authentication (using the 802.1X framework)
- Confidentiality (TKIP encryption)
- Integrity checking
- Protection against replay attacks.

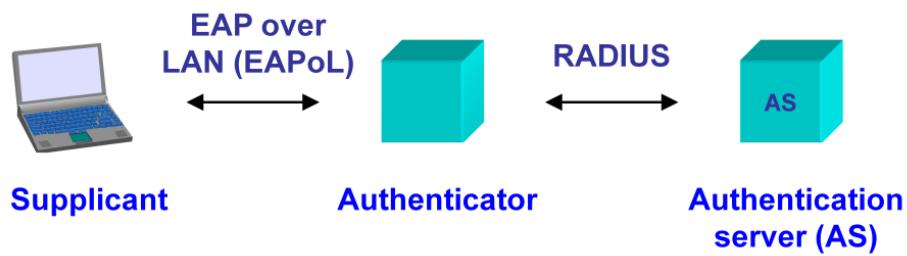
■ RC4 stream cipher, just like WEP encryption, with the following differences:

- The length of the initialization vector is **48 bit** (instead of 24 bit in WEP)
- TKIP uses **104-bit per-packet keys**, derived from a master secret and different for each packet (instead of a 40-bit or 104-bit static preshared key in WEP).

- The 802.1X authentication framework protects wired and wireless networks from unauthorised use in open environments
 - 802.1X uses **EAP (Extensible Authentication Protocol)** to handle authentication requests. As the name implies, EAP is extensible and therefore should be future proof.
 - 802.1X also uses **RADIUS (Remote Authentication Dial-in User Service)** for handling secure signalling between AP and authentication server.
 - ▶ Use of a key distribution centre

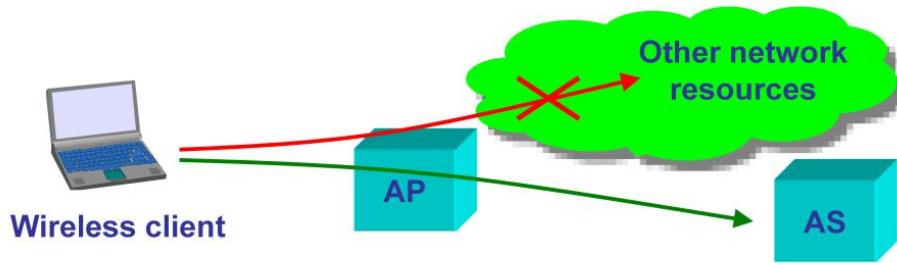
802.1X architecture

802.1X defines three network entities: **Suplicant** (the wireless client in the wireless station), **authenticator** (in a WLAN usually the AP) and **authentication server** (containing user-related authentication information).



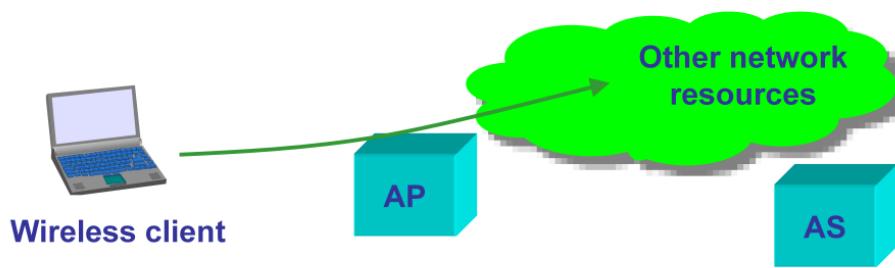
802.1X authentication procedure (1)

With 802.1X, authentication occurs after association. However, prior to successful authentication, a wireless client is only allowed access to the AS. All other traffic is blocked at the AP.



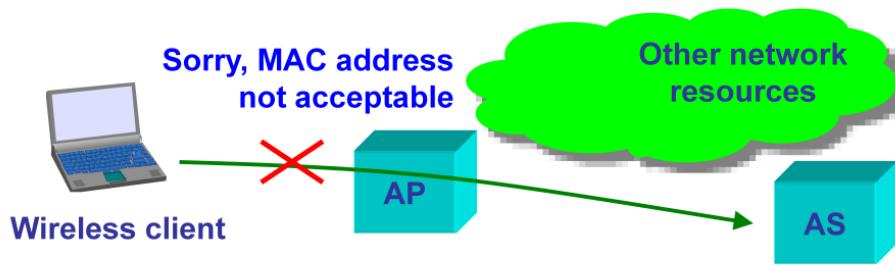
802.1X authentication procedure (2)

After successful authentication, the wireless client is granted access to other network resources by the AP.



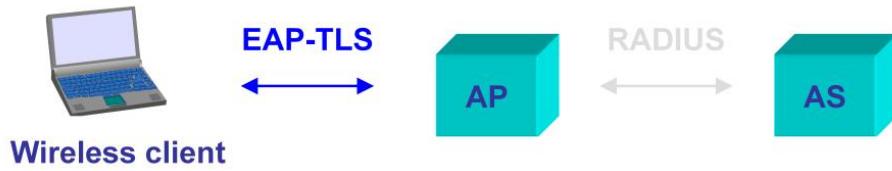
802.1X authentication procedure (3)

The authenticator (AP) can also perform authentication based on MAC address filtering (for preventing denial-of-service = DoS attacks) before starting the 802.1X authentication.

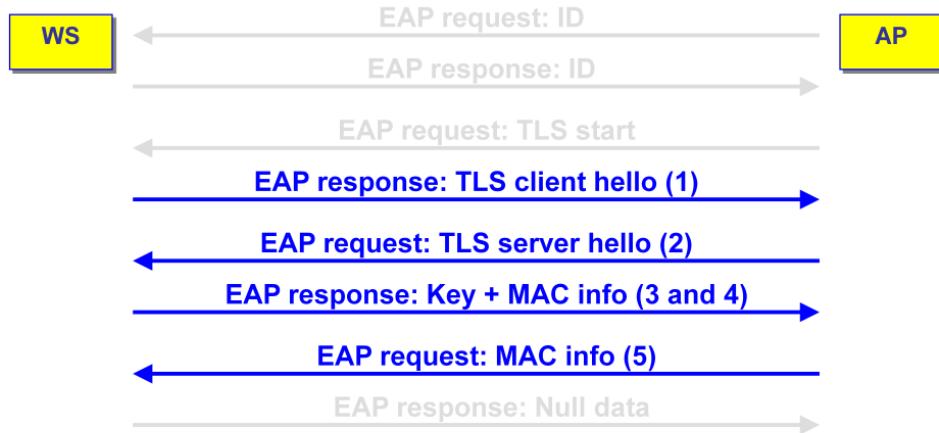


Example: EAP-TLS

As an example, SSL/TLS is one of the various options defined to be used over EAP. The next two slides show how the SSL/TLS handshake sequence is embedded into a corresponding EAP sequence.



EAP-TLS signalling sequence



Authentication in operator-based network

802.11 networks offer new possibilities when the wireless station includes a SIM (Subscriber Identity Module) that is provided by a certain **network operator / service provider**.

Through the SIM, operators can offer WLAN users added value applications such as **secure authentication**, nation-wide or worldwide **roaming**, and user-tailored **charging solutions**.



- Network model
- Secure configuration of devices
- Exchanging keys
- Secure networking protocols
 - Transport layer: TLS & SSL
 - Network layer: IPSec & VPN
 - Data link layer: WEB & WPA
- Firewalls



Besides the lecturers' own material, many third party, often copyrighted, material is reused within this lecture (e.g. in the notes) under the 'fair use' approach, for sake of educational purpose only, and very limited edition. As a consequence, the current slide set presentation usage is restricted, and is falling under usual copyrights usage.

At the end of every lecture, appropriate references to used materials are included.

■ This work contains content adapted from, amongst others, the following sources (in no particular order)

● Books

- ▶ William Stallings, “**Cryptography and Network Security, principles and practices**”, 6th (international) edition, Prentice Hall, 2010;
- ▶ Matt Bishop, “**Computer Security: Art and Science**”, Addison Wesley, Pearson Education, 2003, ISBN-13: 978-0-201-44099-7

● Lecture slides:

- ▶ “Informatiebeveiliging”, Universiteit Gent, Eric Laermans & Thom Dhaene
- ▶ Stallings, 2014, Lecture slides by Lawrie Brown
- ▶ Slides Teknillinen Korkeakoulu, S-72.3240 Wireless Personal, Local, Metropolitan and Wide Area Networks

● Wikipedia

- ▶ Note page descriptions

● Websites



Network and Computer Security

Chapter 3 – Network and communication security

Prof. dr. ir. Eli De Poorter

- Network model
- Secure configuration of devices
- Exchanging keys
- Secure networking protocols
- Firewalls
 - Packet filter
 - Circuit-level gateway
 - Application-level gateway (aka proxy)

■ Firewall

- **Located between local network and Internet**
 - ▶ Intended as protection wall against attacks from **outside**
 - ▶ Controls all incoming and outgoing traffic
 - ✓ Exclude all other access to local network
 - ✓ Possibility to generate alerts for detected anomalies (IDS functionality)
 - ▶ Enforces restrictions on network traffic
 - ✓ Only traffic authorised in security policy
 - ▶ Must be itself immune from unauthorised access
 - ✓ Requires reliable system / secure OS

■ Firewall controls

- **Control of services/direction**

- ▶ Which Internet services (e-mail, WWW, etc.) are accessible
 - ✓ Internal services from outside
 - ✓ External services from within

- **Control of users**

- ▶ Access control to services, dependent on user, which requires authentication technique

- **Behaviour control**

- ▶ Control how services are used (e.g. spam filter, filter for some websites, protection against DoS, etc.)

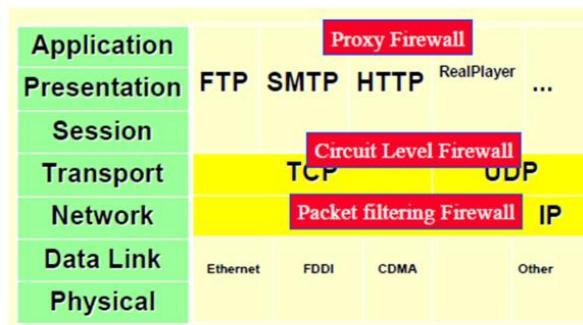


■ Limitations

- **Secures the *perimeter* of a company**
 - ▶ No protection against internal attacks or complicity from within
- **No protection against attacks that circumvent the firewall**
 - ▶ Other access points to LAN may be vulnerable (wireless, mobile devices, etc.)
- **Limited protection against malware**
 - ▶ Scanning all incoming traffic sometimes integrated in firewall, at significant computational cost

■ A few types

- **Packet filter**
- **Circuit-level gateway**
- **Application-level gateway (aka proxy)**

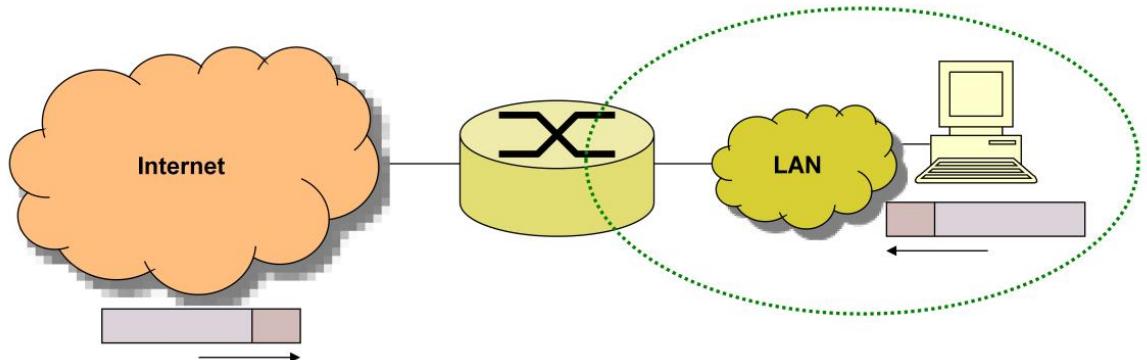


- Network model
- Secure configuration of devices
- Exchanging keys
- Secure networking protocols
- **Firewalls**
 - **Packet filter**
 - Circuit-level gateway
 - Application-level gateway (aka proxy)

■ Packet filter

• Simplest component

- ▶ Router analyses each incoming and outgoing IP packet
- ▶ Decides, based on filter rules, which packets are accepted, and which ones are blocked



p. 8

- **Filter rules based on information in IP packet**
 - **Source IP address**
 - **Destination IP address**
 - **(Source/destination) address at transport level**
 - ▶ TCP or UDP port (defines applications such as HTTP or SMTP)
 - **Transport protocol used**
 - **Router interface where packet arrives or is sent to**

■ Possible default policy for filter rules (“forward policy”)

- “Block”/“discard”

- ▶ What isn't explicitly authorised, is forbidden
 - ✓ More prudent...
 - ✓ ...but less user friendly
 - ✓ Progressively setting up list of authorised services

- “Allow”/“forward”

- ▶ What isn't explicitly forbidden, is authorised

■ Advantages

- **Basic security**
- **Simple, fast, and relatively cheap**
- **(Reasonably) transparent to applications and users**
- **Non-cryptographic (and thus no key management)**

■ Drawbacks

- **No protection against attacks at the application level**
 - ▶ All instructions for some application are authorised if the application is authorised
- **No (strong) user authentication**
- **Configuration errors are quite common**

p. 12

User authentication is possible if the packet filter is combined with an IPsec gateway (AH mechanism). This increases the efficiency of the firewall at the cost of higher computation time and the use of IPsec (much less simple than a plain packet filter).

Creating filter rules for a packet filter has been compared to programming in assembler: you have unlimited freedom but at great implementation costs.

■ Issues

- **Risk of IP spoofing**

- ▶ Cause firewall to believe packet originates from authorised network
- ▶ Can be averted by blocking incoming packets with an IP address from local network
 - ✓ Only possible if external networks can be denied access
- ▶ Can be prevented using IPsec

- **Fragmentation attack**

- ▶ Split IP packet in very small fragments, causing TCP header information to be contained in next IP packet
- ▶ To be prevented by blocking such tiny packets

■ Improvement: “stateful inspection”

- **Tracks connections**

- ▶ Dynamic table of active connections
 - ✓ addresses, ports, sequence number, etc.
- ▶ Automatic time-outs

- **Automatically allows future packets of the same flow**

■ Advantages

- **No need to manually specify new rules**

- ▶ If outbound connection is permitted, inbound traffic corresponding to the same flow is automatically allowed
 - ✓ Even on different port numbers
- ▶ Only outbound rules need to be defined

- **Computationally less expensive**

- ▶ No deep packet inspection required

■ Disadvantages

- **Not so easy for non-TCP traffic**

- ▶ Pings, UDP, ...

p. 14

Pure packet filters do not keep track of traffic and have no memory of previous packets which makes them vulnerable to spoofing attacks. Such a firewall has no way of knowing if any given packet is part of an existing connection, is trying to establish a new connection, or is just a rogue packet. In contrast, a stateful firewall (any firewall that performs stateful packet inspection (SPI) or stateful inspection) is a firewall that keeps track of the state of network connections (such as TCP streams, UDP communication) traveling across it. The firewall is programmed to distinguish legitimate packets for different types of connections. Only packets matching a known active connection will be allowed by the firewall; others will be rejected.

The classic example of a network operation that may fail with a stateless firewall is the File Transfer Protocol (FTP). By design, such protocols need to be able to open connections to arbitrary high ports to function properly. Since a stateless firewall has no way of knowing that the packet destined to the protected network (to some host's destination port 4970, for example) is part of a legitimate FTP session, it will drop the packet. Stateful firewalls with application inspection solve this problem by maintaining a table of open connections, inspecting the payload of some packets and intelligently associating new connection requests with existing legitimate connections. A stateful firewall keeps track of the state of network connections (such as TCP streams or UDP communication) and is able to hold significant attributes of each connection in memory. These attributes are collectively known as the state of the connection, and may include such details as the IP addresses and ports involved in the connection and the sequence numbers of the packets traversing the connection. Stateful inspection monitors incoming and outgoing packets over time, as well as the state of the connection, and stores the data in dynamic state tables. This cumulative data is evaluated, so that filtering decisions would not only be based on administrator-defined rules, but also on context that has been built by previous connections as well as previous packets belonging to the same connection.

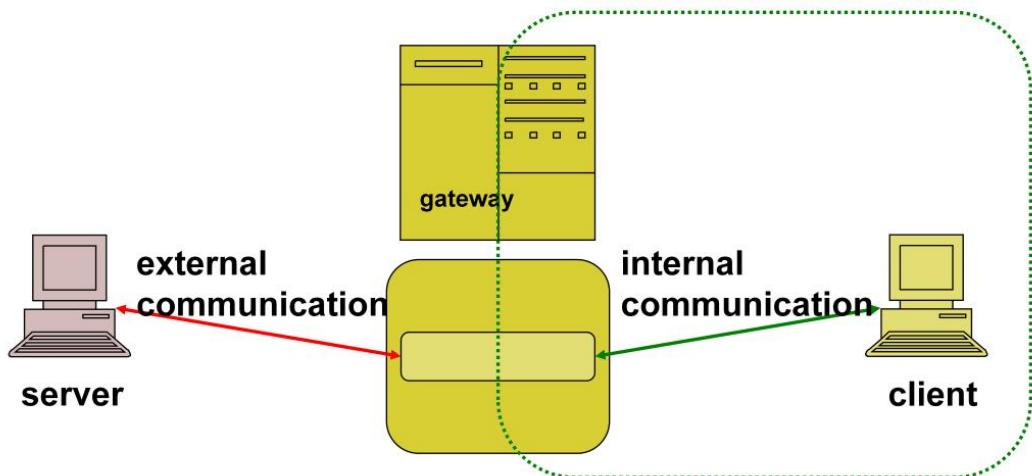
Depending on the connection protocol, maintaining a connection's state is more or less complex for the firewall. For example, TCP is inherently a stateful protocol as connections are established with a three-way handshake ("SYN, SYN-ACK, ACK") and ended with a "FIN, ACK" exchange. This means that all packets with "SYN" in their header received by the firewall are interpreted to open new connections. If the service requested by the client is available on the server, it will respond with a "SYN-ACK" packet which the firewall will also track. Once the firewall then receives the client's "ACK" response, it transfers the connection to the "ESTABLISHED" state as the connection has been authenticated bidirectionally. This allows tracking of future packets through the established connection. Simultaneously, the firewall drops all packets which are not associated with an existing connection recorded in its state table (or "SYN" packets), preventing unsolicited connections with the protected machine. Other connection protocols, namely UDP and ICMP, are not based on bidirectional connections like TCP, making a stateful firewall somewhat less secure. In order to track a connection state in these cases, a firewall must transfer sessions to the ESTABLISHED state after seeing the first valid packet. It can then only track the connection through addresses and ports of the following packets' source and destination. Unlike TCP connections, which can be closed by a "FIN, ACK" exchange, these connectionless protocols allow a session to end only by time-out.

By keeping track of the connection state, stateful firewalls provide added efficiency in terms of packet inspection. This is because for existing connections the firewall need only check the state table, instead of checking the packet against the firewall's rule set, which can be extensive. Additionally, in the case of a match with the state table, the firewall does not need to perform deep packet inspection.

- Network model
- Secure configuration of devices
- Exchanging keys
- Secure networking protocols
- Firewalls
 - Packet filter
 - **Circuit-level gateway**
 - Application-level gateway (aka proxy)

■ Circuit level gateway

- Operates as a relay at TCP level
 - ▶ Or occasionally also at UDP level



p. 16

A circuit level gateway can also be considered more or less as a proxy-server for TCP.

- 1) The gateway receives a request from the client to set up a TCP connection.
 - 2) The gateway takes care of the client authentication and required authorisation
 - 3) The gateway sets up a TCP connection with the server in name of the client
- After this TCP connection setup the gateway will transmit the data from the internal TCP connection to the external TCP connection (and vice versa).

■ Advantages

- Gateway doesn't need to know the application
- Generic as concerns the applications used
- Suitable to allow authenticated traffic through a firewall
- Can be combined with proxy servers (see later)
 - ▶ Especially for applications for which there is no proxy server

■ Drawbacks

- Unable to intercept application specific threats
 - ▶ E.g. Java applets, ActiveX, SQL injection, etc.

■ SOCKS

- **Best known example of circuit level gateway**
- **Requires a few modifications to client software or TCP/IP stack**
 - ▶ In order to support SOCKS specific calls
 - ▶ Adding SOCKS-library to client software
 - ▶ Commonly used Web browsers are SOCKS compliant
- **Used for dynamic SSH port forwarding**

- Network model
- Secure configuration of devices
- Exchanging keys
- Secure networking protocols
- Firewalls
 - Packet filter
 - Circuit-level gateway
 - **Application-level gateway (aka proxy)**

■ Application level gateway aka proxy

- **Application specific**
 - ▶ E.g. HTTP, SMTP, FTP, etc.
- **Traffic for this application only possible via proxy**

p. 20

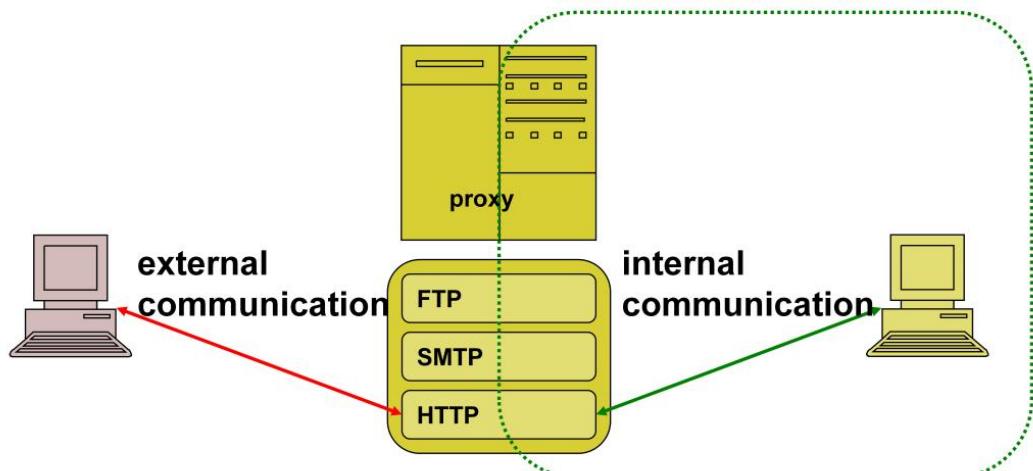
The client authentication by the proxy can also be outsourced to an authentication server (e.g. RADIUS-server). This is even a desirable solution, as critical authentication information will no longer be found at the firewall itself.

■ Application level gateway aka proxy

● Proxy has access to complete protocol

- ▶ Internal user requests service from proxy
- ▶ Proxy verifies and validates request
 - ✓ May also take care of (weak or strong) client authentication
 - ✓ Includes possibility not to support certain parts of the service
- ▶ Forwards request outward and returns result to user
- ▶ In contrast to circuit level gateway, a proxy can analyse the complete (application) protocol

- Application level gateway aka proxy



p. 22

■ Advantages

- **More secure than packet filters or circuit level gateways**
 - ▶ Not restricted to information within IP packets
 - ✓ Access to the complete application protocol
 - ▶ Limited number of (authorised) applications to be investigated
 - ✓ Other applications can be blocked by default
 - ▶ Much easier to implement auditing/logging

p. 23

A simple example a possible application of a proxy server, for which a circuit level gateway or a packet filter would come short, is the following. Suppose that for an (anonymous) FTP-server (within the local network) one wants to allow both internal and external users to upload files on the server, but that only internal users may read these files. In this case a proxy-server may allow the PUT-instruction to pass the proxy, but will block the GET-instruction for all traffic that crosses the firewall perimeter.

Another situation where a proxy may be advantageous is in blocking distributed software (e.g. web services), which is typically not blocked by a packet filter (TCP port 80 of HTTP).

■ Drawbacks

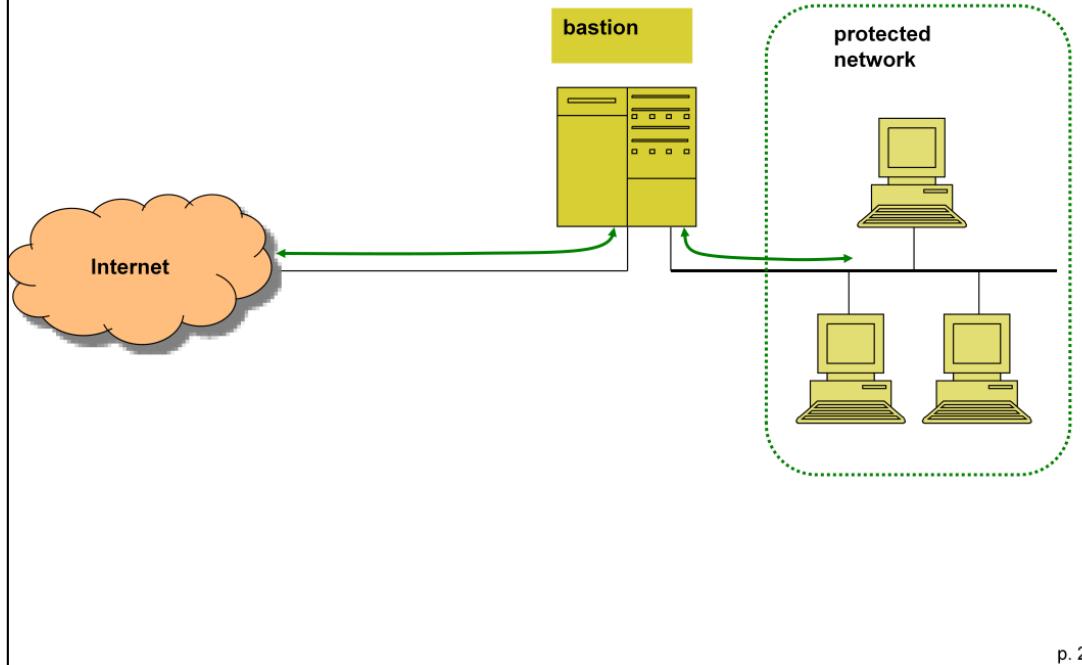
- **Requires modifications of user procedures:**
 - ▶ First login at proxy server, only thereafter at final destination
 - ▶ Or application modifications (rare)
- **Specific for 1 application (FTP, HTTP, etc.)**
- **Much more processing per connection**
 - ▶ Double connection
- **Not all services support proxies easily**
 - ▶ Impossible when protocol specification is unknown, which may happen with proprietary software

■ Bastion

- **Critical for security within network perimeter**
 - ▶ Exposed to external attacks
 - ▶ Typically used as gateway to the system
- **Desirable properties**
 - ▶ Secure OS version
 - ▶ Only essential services are installed
 - ✓ Proxy applications (DNS, FTP, SMTP, user authentication, etc.)
 - ✓ Only minimally necessary set of instructions of applications is implemented
 - ▶ Possibility to use strong authentication for access to proxy services
- **For sufficiently large systems**
 - ▶ Firewall is more than 1 single device (packet filter or gateway)
 - ▶ Combination of several elements
 - ▶ Some basic configurations in the following slides

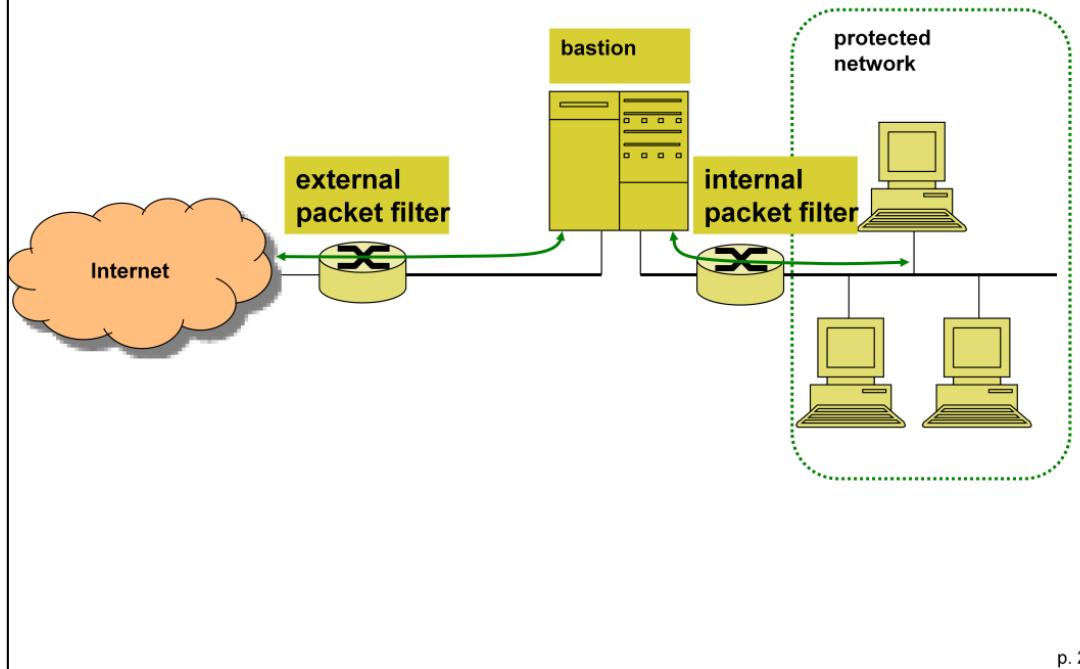
p. 25

A Bastion host is a special purpose computer on a network specifically designed and configured to withstand attacks. The computer generally hosts a single application, for example a proxy server, and all other services are removed or limited to reduce the threat to the computer. It is hardened in this manner primarily due to its location and purpose, which is either on the outside of the firewall or in the DMZ and usually involves access from untrusted networks or computers.



p. 26

A multi-homed host has more than 1 network interface, allowing to separate network parts, in this case between the external and the internal network.



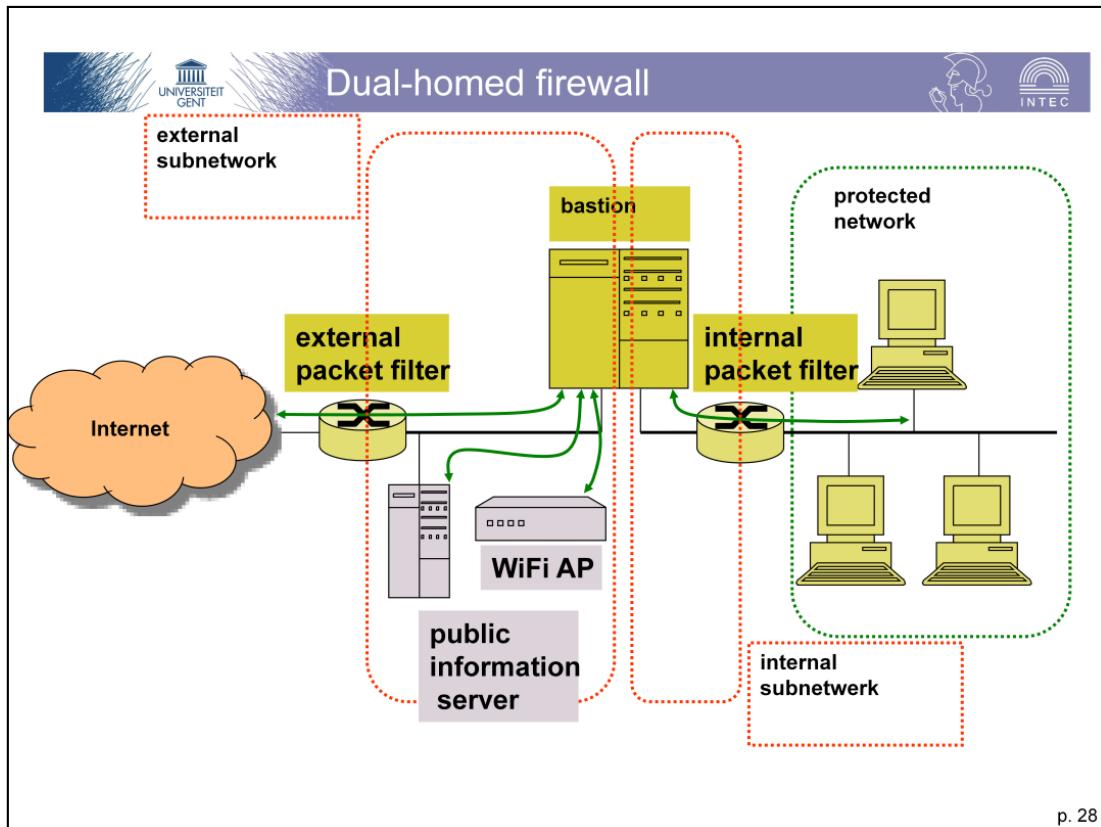
p. 27

This is a more realistic approach of a dual-homed firewall. Three elements in this firewall system: external and internal packet filter, and bastion.

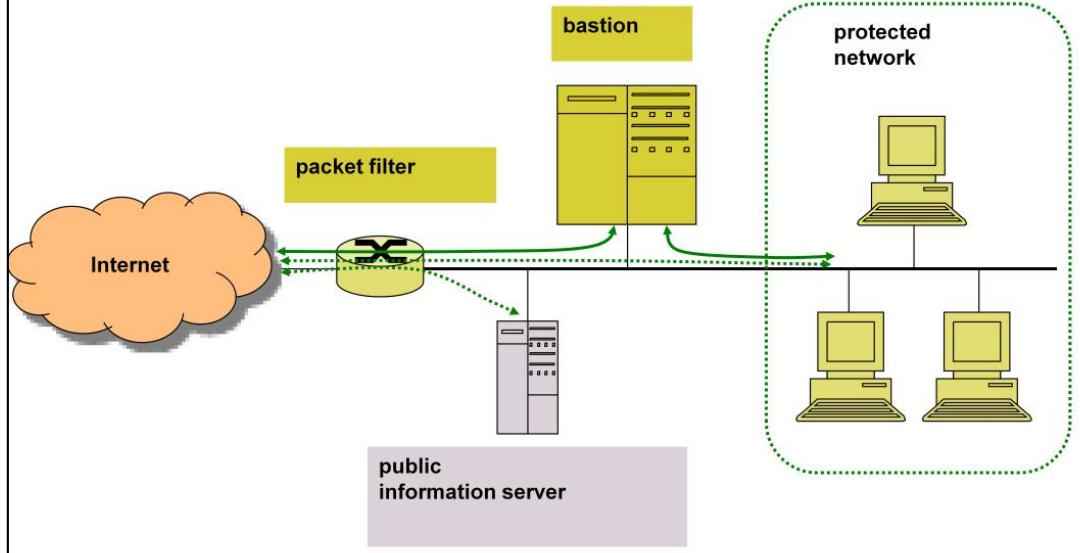
There is a (external) packet filter between the Internet and the bastion, and also a (internal) packet filter between the bastion and the protected network. The external packet filter only allows traffic between the Internet and the bastion (and no direct communication between the protected network and the Internet). The internal packet filter causes the protected network to be able to access the outside only through the bastion.

This configuration is simple and very secure, but not very flexible:

- 1) The bastion can be a limiting network performance factor as it has to handle all incoming and outgoing traffic.
- 2) If the bastion fails, connection to the outside is lost ("single point of failure").
- 3) Application protocols without proxy support are not allowed.



In this situation an external subnetwork and an internal subnetwork are created. The external subnetwork is typically adequate to host some non-critical services, such as a public webserver or access servers for other networks (e.g. wireless access points).



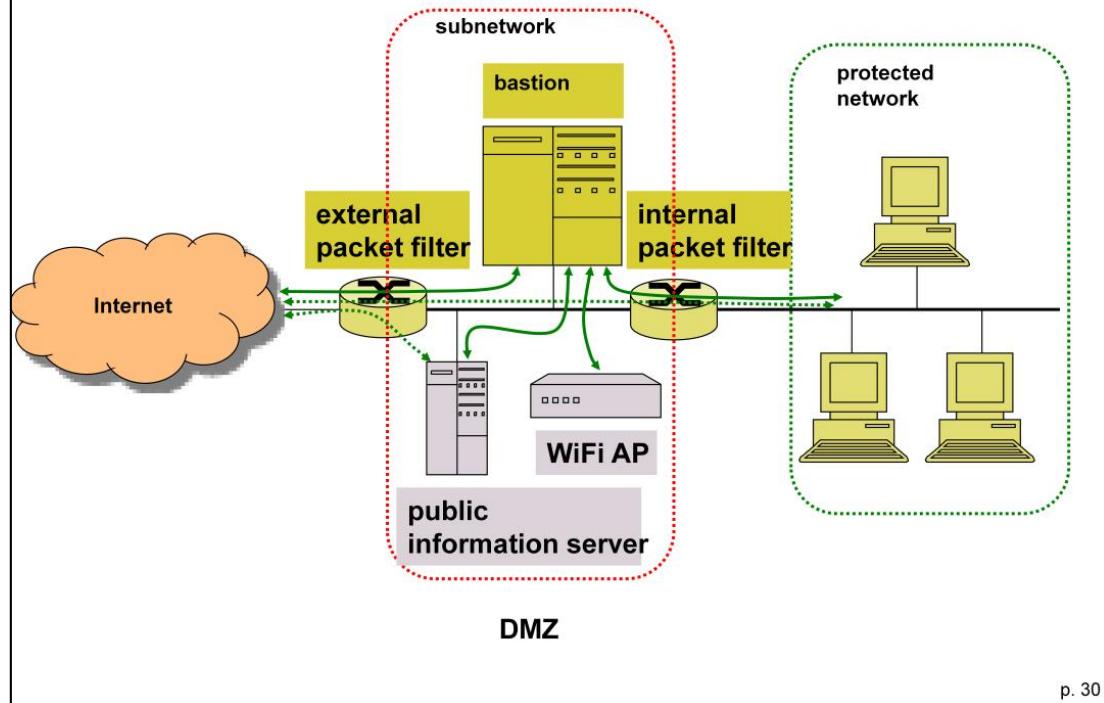
p. 29

Two elements in firewall system: packet filter and bastion.

The packet filter is configured to allow only IP packets from the Internet to the bastion, and for packets originating from the protected network, only IP packets from the bastion to the outside.

The function of the bastion is to provide authentication and the proxy-functions.

For some (presumed secure) applications (e.g. the access to a public web server, but also IPsec traffic) it may be allowed to bypass the bastion and to authorise direct communication with some part of the protected network. The flexibility of this configuration is therefore much larger, but the degree of security is potentially lower (as it is possible to bypass the bastion for some part of the traffic).



p. 30

Three elements in firewall system: external and internal packet filter, and bastion.

There is a (external) packet filter between the Internet and the bastion, and also a (internal) packet filter between the bastion and the protected network. An isolated subnetwork is thus created (which is screened by the bastion), which may also contain public information servers (this is also the zone where wireless access points are best located). Traffic from the Internet to the subnetwork and from the protected network to the subnetwork may be enabled, but any direct traffic between the Internet and the protected network will be blocked by the packet filters. The presence of the internal packet filter improves the protection compared to the situation in a "screened host firewall".

This is a configuration offering both a decent degree of protection and sufficient flexibility. Here too, it is possible to authorise for certain "secure" applications a direct communication between the protected network and the internet, without having to use the bastion).

The subnetwork is sometimes also called the "demilitarised zone" (in short DMZ).

■ Personal firewalls

• Software on PC

- ▶ Minimal perimeter (PC instead of complete network)
- ▶ More limited functionality
- ▶ Limited reliability
 - ✓ Running on regular OS
- ▶ Still useful as basic protection against external trouble

■ A few concluding remarks

- **Firewalls don't solve *all* security issues**
 - ▶ Only effective if all incoming and outgoing traffic passes through the firewall
 - ✓ Threat of wireless access points
 - ▶ No protection against internal threats
 - ✓ Including imported malware
- **Protection of a local network as a fortified citadel is a somewhat medieval concept**
- **The best protection won't withstand the creativity of internal users**

- Network model
- Secure configuration of devices
- Exchanging keys
- Secure networking protocols
- Firewalls

■ This work contains content adapted from, amongst others, the following sources (in no particular order)

● Books

- ▶ William Stallings, “**Cryptography and Network Security, principles and practices**”, 6th (international) edition, Prentice Hall, 2010;
- ▶ Matt Bishop, “**Computer Security: Art and Science**”, Addison Wesley, Pearson Education, 2003, ISBN-13: 978-0-201-44099-7

● Lecture slides:

- ▶ “Informatiebeveiliging”, Universiteit Gent, Eric Laermans & Thom Dhaene
- ▶ Stallings, 2014, Lecture slides by Lawrie Brown

● Wikipedia

- ▶ Note page descriptions

● Websites