

# SSL für alle

Peter Magnusson

Twitter: @Blaufish\_, Sakerhetspodcasten.se & Omegapoint.se

Joachim Strömbergson

Twitter: @Kryptoblog, secworks.se



# How to get good SSL security

# Agenda

What is HTTPS / SSL



Attacks on SSL



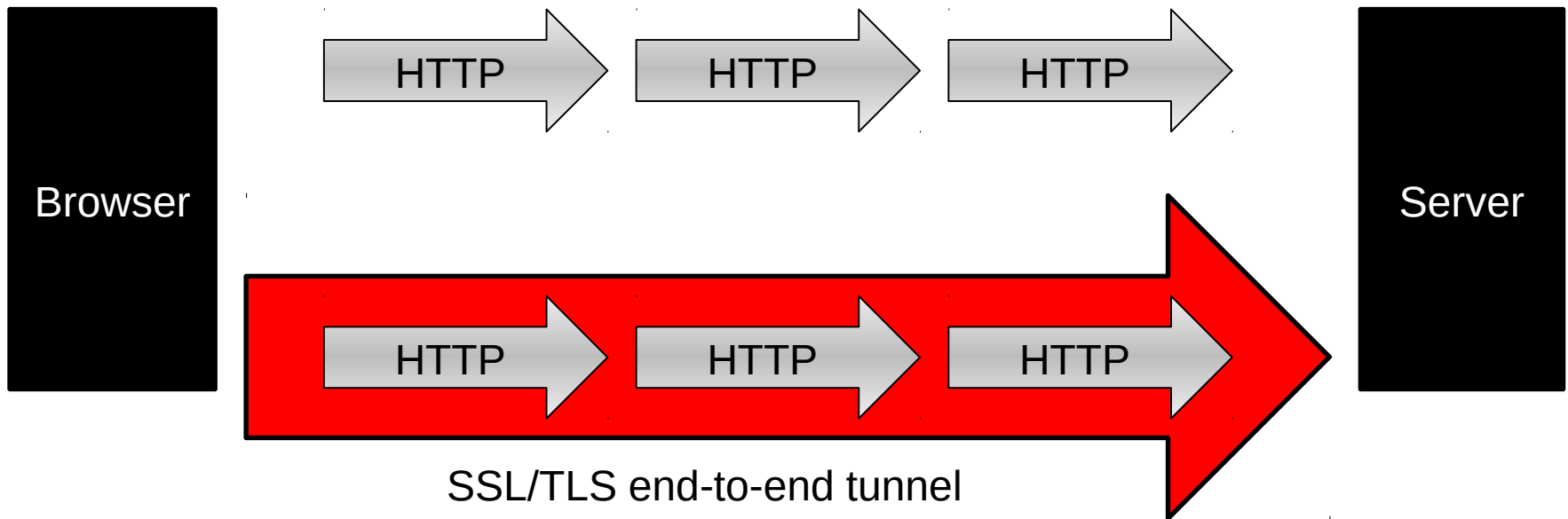
Attacks on related technology



Coding SSL/TLS

# What is HTTPS and SSL/TLS?

# What is HTTPS?



# What does SSL/TLS offer?

- Confidentiality
- Integrity / tamper resistance
- Authentication
  - Server authentication
  - Client authentication (option rarely used)
- Cryptographic agility
  - Negotiate mutually accepted cipher suits

Authenticated handshake.  
Communication is only MACed.  
DOES NOT offer non  
repudiation

# SSL/TLS versions

- SSL 3.0 (Netscape)
  - 1996, first widely used SSL. 1.0 & 2.0 broken.
- TLS 1.0
  - 1999, IETF standardization, & security fixes.
- TLS 1.1
  - 2006, CBC Cipher Block Chaining improvements
- TLS 1.2
  - 2008, Authenticated Encryption suites, AES, and many other security improvements.

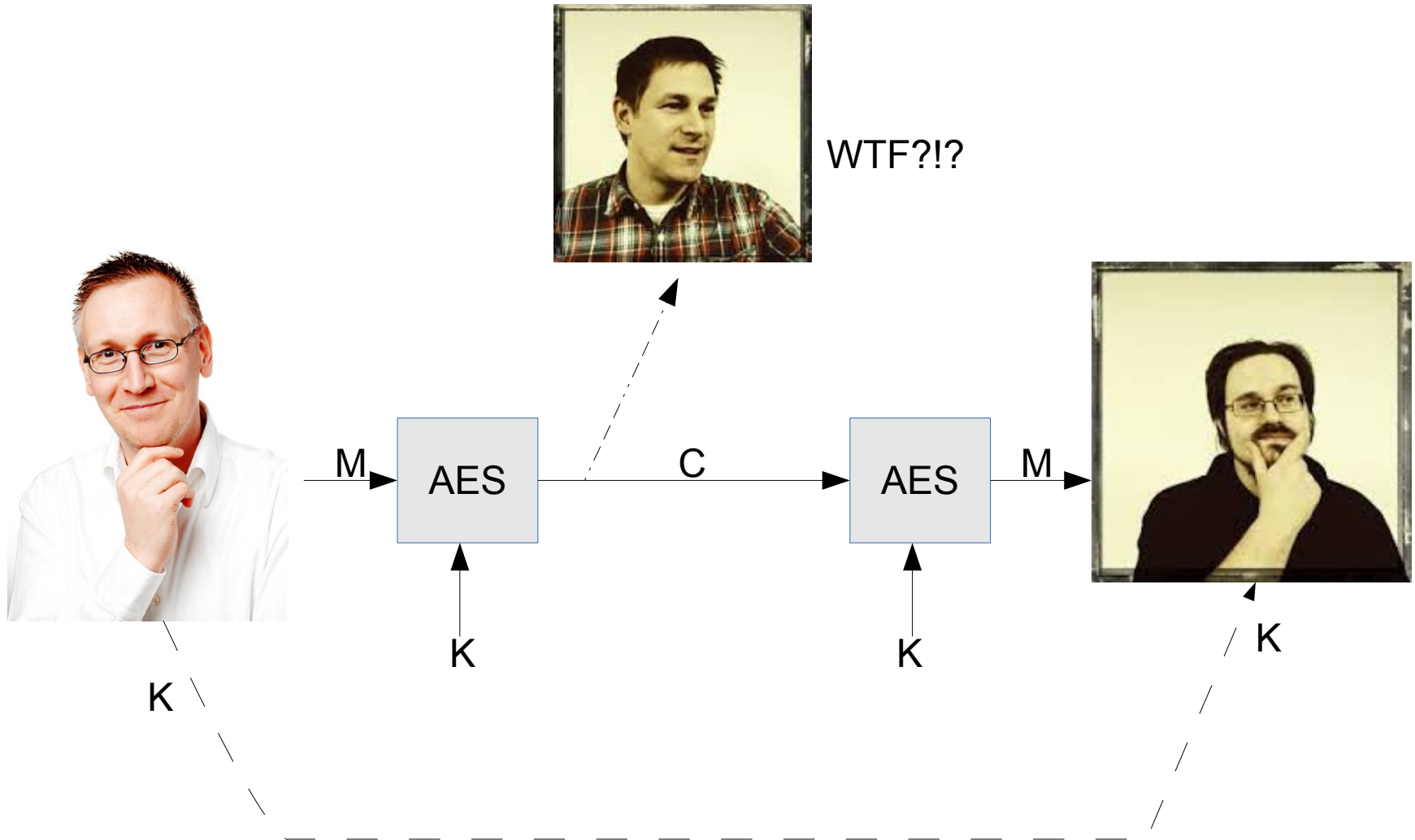
# SSL/TLS not perfect

- SSL/TLS is not particularly well designed
  - Evolution, with many hotfixes and workarounds
  - Compatibility and legacy
  - Horribly slow adoption rate of new protocol versions and more secure cipher suites
- MAC is performed on plaintext not ciphertext
  - Most suites are by design vulnerable to chosen ciphertext attacks



# Algorithms and Cipher Suites

# Symmetric ciphers



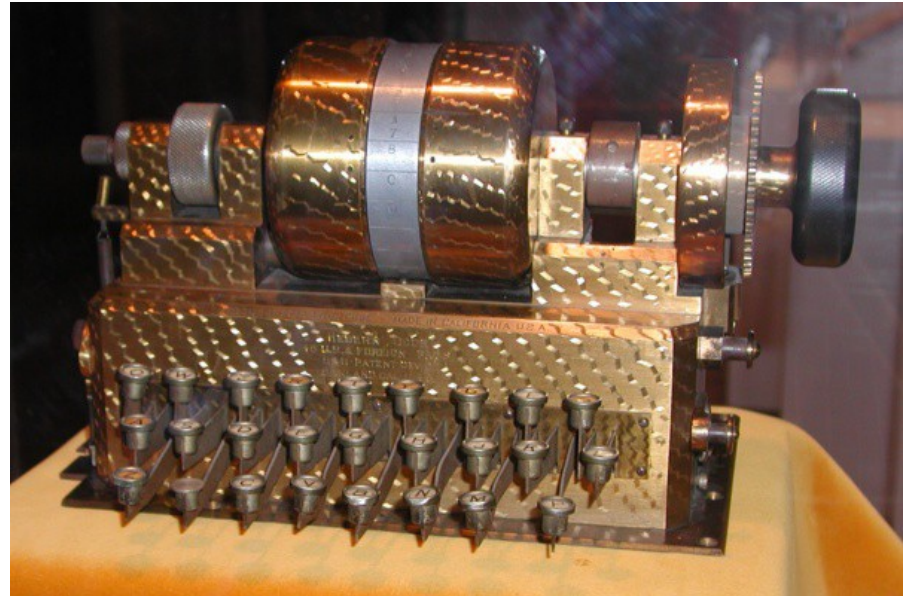
# Symmetric ciphers

- Bulk encryption
  - AES, DES/3DES, Camellia, RC2, RC4
  - Fast, efficient in SW and HW
- Provides confidentiality **NOT** integrity
  - Encrypted message can be changed
- Must transfer secret key to receiver
  - How to protect the secret key during transfer?
  - Classic cipher with classic problem

# Symmetric ciphers



Jefferson (1795)



Hebern (1918)



# Symmetric ciphers



**Hagelin C-36**

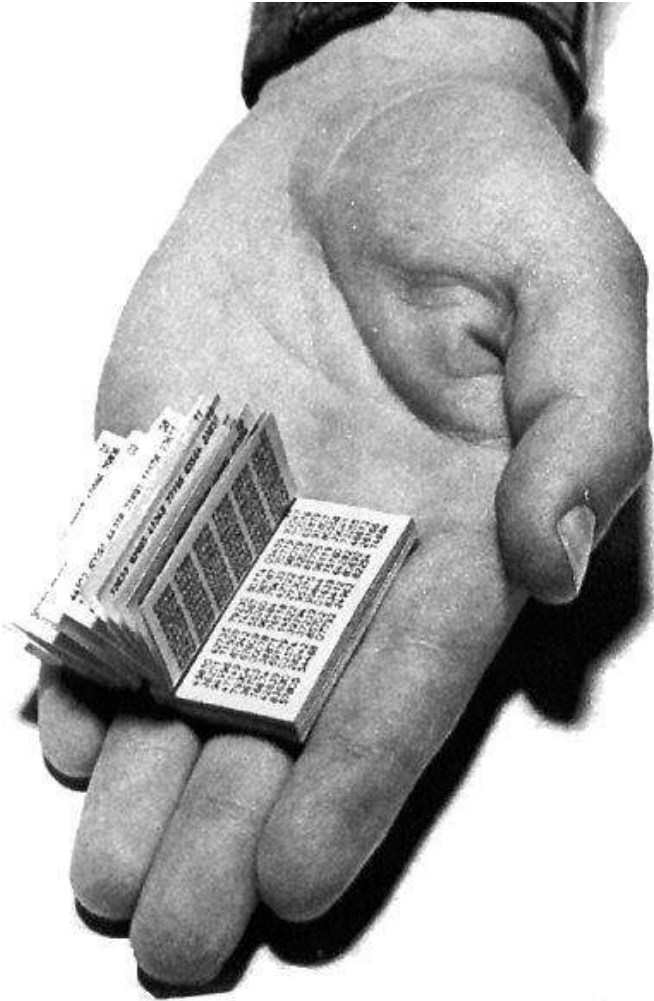


**Enigma**

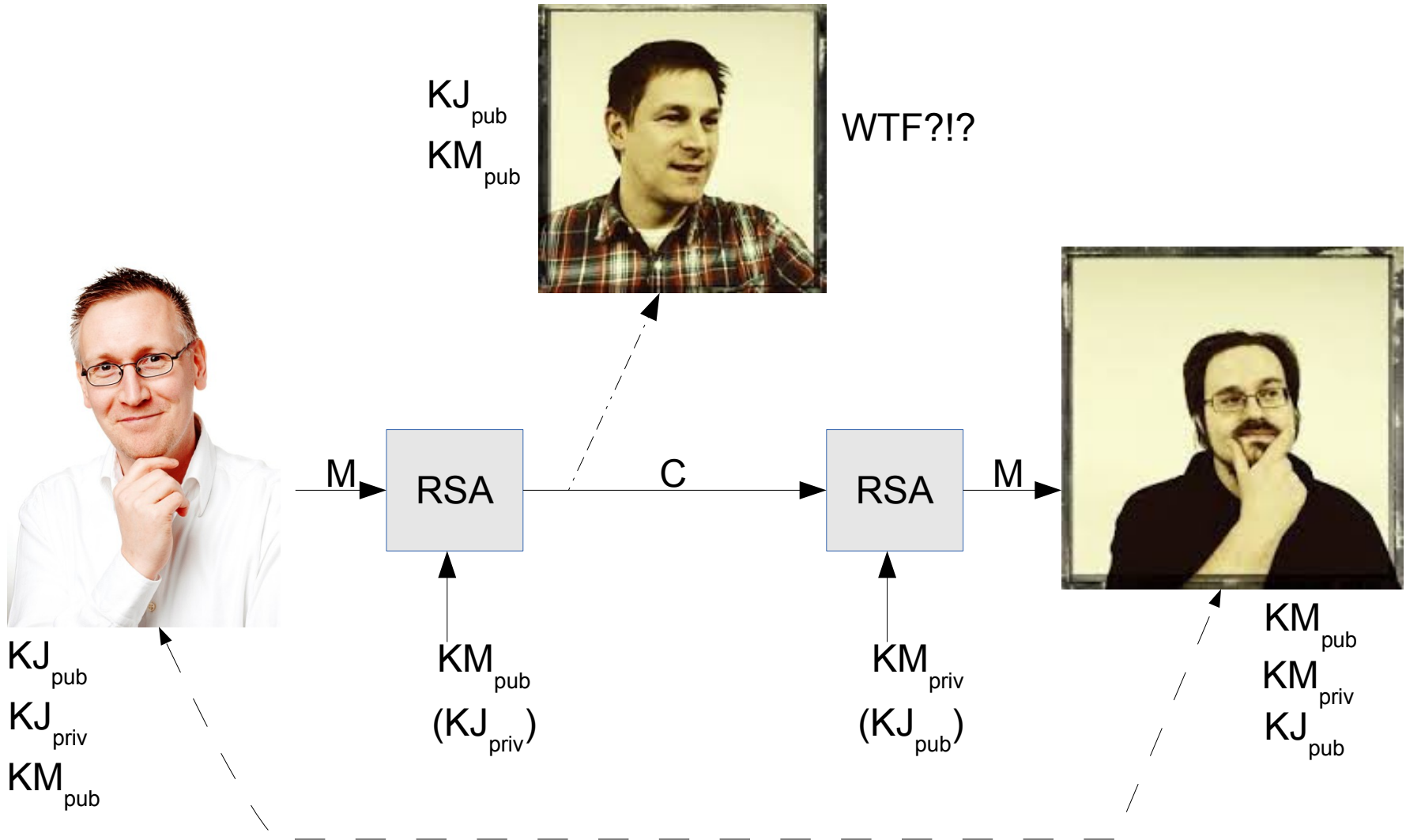


**Fialka**

# Keys for symmetric ciphers



# Asymmetric ciphers



# Asymmetric ciphers

- Public key encryption – session init
  - RSA, ECC, El Gamal
  - 1000x-10000x slower than symmetric encryption
  - Complex math – hard for embedded systems
- Provides confidentiality
  - Can provide integrity, origin authentication
  - Can provide exchange of secret keys (D-H)
- Must transfer public key to sender
  - No need to protect the key – but need to trust the key
  - CAs provides trust by proxy (assumed trust in CA)



# Hash functions

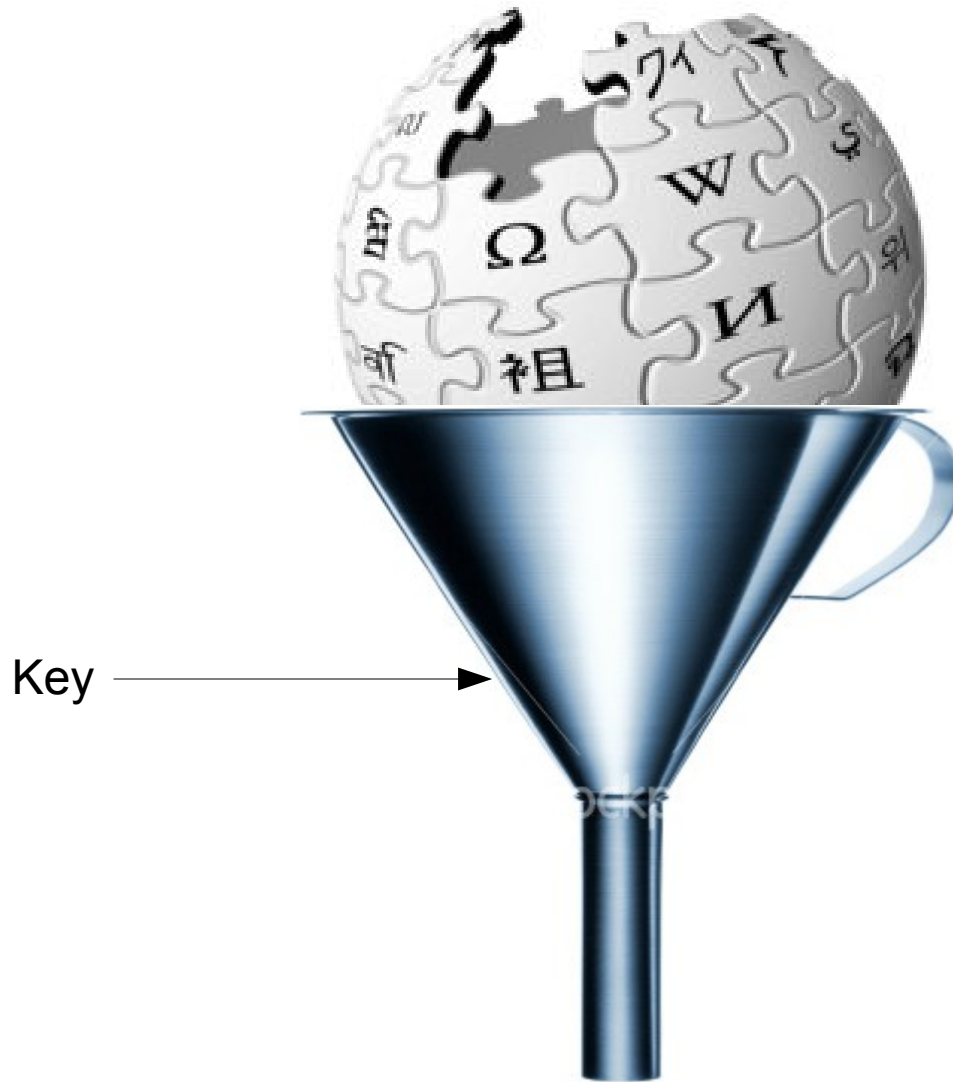


**0x557e8e7ed1534946462f4136623947ca**

# Hash functions

- Variable data size in, fixed size data out
  - Fingerprint, digest, hash related to input
- Keyless function
  - Security is based on collision resistance
- Provides data integrity
  - Detect presence of changes (errors) in the data

# MACs



**0x557e8e7ed1534946462f4136623947ca**

# MACs

## Message Authentication Codes

- Variable data size in, fixed size data out
  - HMAC, OMAC, UMAC, CBC-MAC
    - Built using hash functions, block ciphers etc.
  - Fingerprint, digest, hash related to input
- Keyed function
  - Security is based on the secrecy of the key
  - Must transfer key to recipient
- Provides data integrity and authentication
  - Detect presence of changes (errors) in the data
  - Validate that the data is from the owner of the key

# Cipher suites in my OpenSSL

```
js@secworks82.gotanet.se: /Users/js>openssl ciphers -v 'ALL:COMPLEMENTOFALL'
```

ADH-SEED-SHA	SSLv3	Kx=DH	Au=None	Enc=SEED(128)	Mac=SHA1	
DHE-RSA-SEED-SHA	SSLv3	Kx=DH	Au=RSA	Enc=SEED(128)	Mac=SHA1	
DHE-DSS-SEED-SHA	SSLv3	Kx=DH	Au=DSS	Enc=SEED(128)	Mac=SHA1	
SEED-SHA	SSLv3	Kx=RSA	Au=RSA	Enc=SEED(128)	Mac=SHA1	
ADH-AES256-SHA	SSLv3	Kx=DH	Au=None	Enc=AES(256)	Mac=SHA1	
DHE-RSA-AES256-SHA	SSLv3	Kx=DH	Au=RSA	Enc=AES(256)	Mac=SHA1	
DHE-DSS-AES256-SHA	SSLv3	Kx=DH	Au=DSS	Enc=AES(256)	Mac=SHA1	
AES256-SHA	SSLv3	Kx=RSA	Au=RSA	Enc=AES(256)	Mac=SHA1	
ADH-AES128-SHA	SSLv3	Kx=DH	Au=None	Enc=AES(128)	Mac=SHA1	
DHE-RSA-AES128-SHA	SSLv3	Kx=DH	Au=RSA	Enc=AES(128)	Mac=SHA1	
DHE-DSS-AES128-SHA	SSLv3	Kx=DH	Au=DSS	Enc=AES(128)	Mac=SHA1	
AES128-SHA	SSLv3	Kx=RSA	Au=RSA	Enc=AES(128)	Mac=SHA1	
ADH-DES-CBC3-SHA	SSLv3	Kx=DH	Au=None	Enc=3DES(168)	Mac=SHA1	
ADH-DES-CBC-SHA	SSLv3	Kx=DH	Au=None	Enc=DES(56)	Mac=SHA1	
EXP-ADH-DES-CBC-SHA	SSLv3	Kx=DH(512)	Au=None	Enc=DES(40)	Mac=SHA1	export
ADH-RC4-MD5	SSLv3	Kx=DH	Au=None	Enc=RC4(128)	Mac=MD5	
EXP-ADH-RC4-MD5	SSLv3	Kx=DH(512)	Au=None	Enc=RC4(40)	Mac=MD5	export
EDH-RSA-DES-CBC3-SHA	SSLv3	Kx=DH	Au=RSA	Enc=3DES(168)	Mac=SHA1	
EDH-RSA-DES-CBC-SHA	SSLv3	Kx=DH	Au=RSA	Enc=DES(56)	Mac=SHA1	
EXP-EDH-RSA-DES-CBC-SHA	SSLv3	Kx=DH(512)	Au=RSA	Enc=DES(40)	Mac=SHA1	export
EDH-DSS-DES-CBC3-SHA	SSLv3	Kx=DH	Au=DSS	Enc=3DES(168)	Mac=SHA1	
EDH-DSS-DES-CBC-SHA	SSLv3	Kx=DH	Au=DSS	Enc=DES(56)	Mac=SHA1	
EXP-EDH-DSS-DES-CBC-SHA	SSLv3	Kx=DH(512)	Au=DSS	Enc=DES(40)	Mac=SHA1	export
DES-CBC3-SHA	SSLv3	Kx=RSA	Au=RSA	Enc=3DES(168)	Mac=SHA1	
DES-CBC-SHA	SSLv3	Kx=RSA	Au=RSA	Enc=DES(56)	Mac=SHA1	
EXP-DES-CBC-SHA	SSLv3	Kx=RSA(512)	Au=RSA	Enc=DES(40)	Mac=SHA1	export
EXP-RC2-CBC-MD5	SSLv3	Kx=RSA(512)	Au=RSA	Enc=RC2(40)	Mac=MD5	export
RC4-SHA	SSLv3	Kx=RSA	Au=RSA	Enc=RC4(128)	Mac=SHA1	
RC4-MD5	SSLv3	Kx=RSA	Au=RSA	Enc=RC4(128)	Mac=MD5	
EXP-RC4-MD5	SSLv3	Kx=RSA(512)	Au=RSA	Enc=RC4(40)	Mac=MD5	export
DES-CBC3-MD5	SSLv2	Kx=RSA	Au=RSA	Enc=3DES(168)	Mac=MD5	
DES-CBC-MD5	SSLv2	Kx=RSA	Au=RSA	Enc=DES(56)	Mac=MD5	
EXP-RC2-CBC-MD5	SSLv2	Kx=RSA(512)	Au=RSA	Enc=RC2(40)	Mac=MD5	export
RC2-CBC-MD5	SSLv2	Kx=RSA	Au=RSA	Enc=RC2(128)	Mac=MD5	
EXP-RC4-MD5	SSLv2	Kx=RSA(512)	Au=RSA	Enc=RC4(40)	Mac=MD5	export
RC4-MD5	SSLv2	Kx=RSA	Au=RSA	Enc=RC4(128)	Mac=MD5	
NULL-SHA	SSLv3	Kx=RSA	Au=RSA	Enc=None	Mac=SHA1	
NULL-MD5	SSLv3	Kx=RSA	Au=RSA	Enc=None	Mac=MD5	

NULL = NO  
cipher!

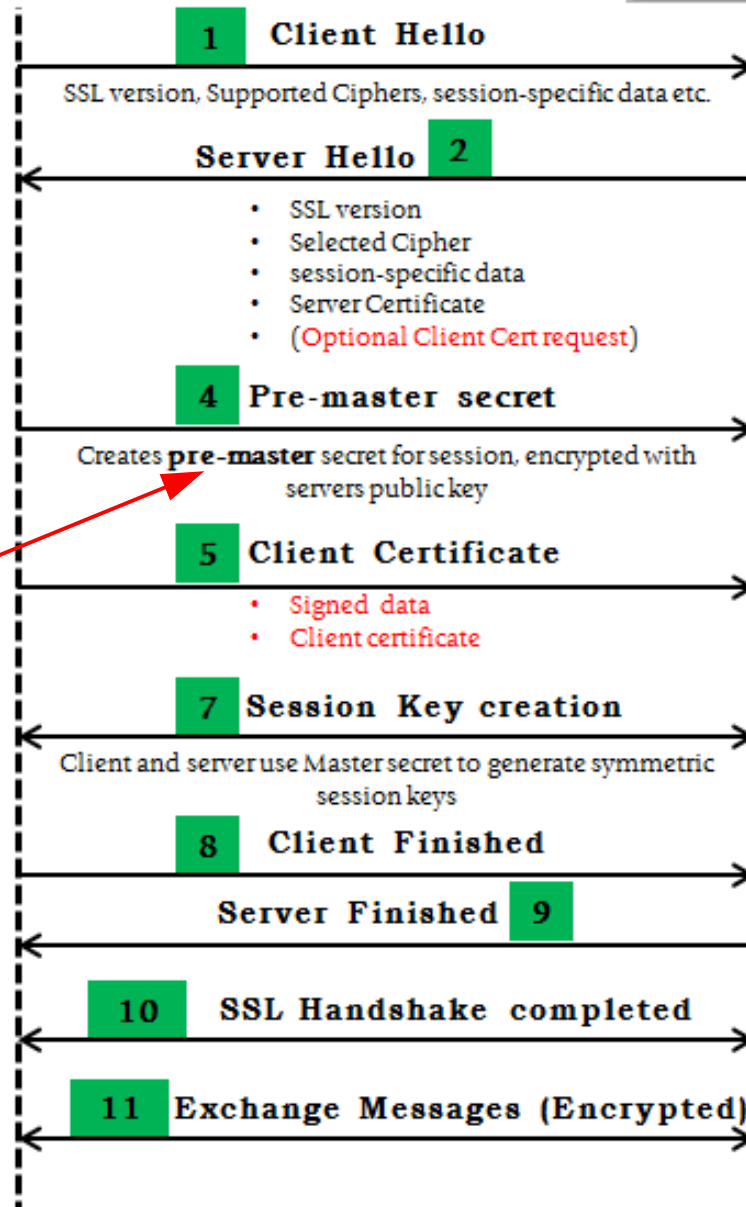


Client



# The SSL/TLS handshake

**3**  
Client authenticates server



The session  
key!

# Session keys in SSL/TLS

- Client generates session master secret
- Client send secret to server. The secret is protected **using the servers public key**
- Session keys are then derived by client and server (encryption, MAC)

**If the private key of the server is lost all previous sessions can be decrypted by extracting their master secrets**

# Perfect Forward Secrecy

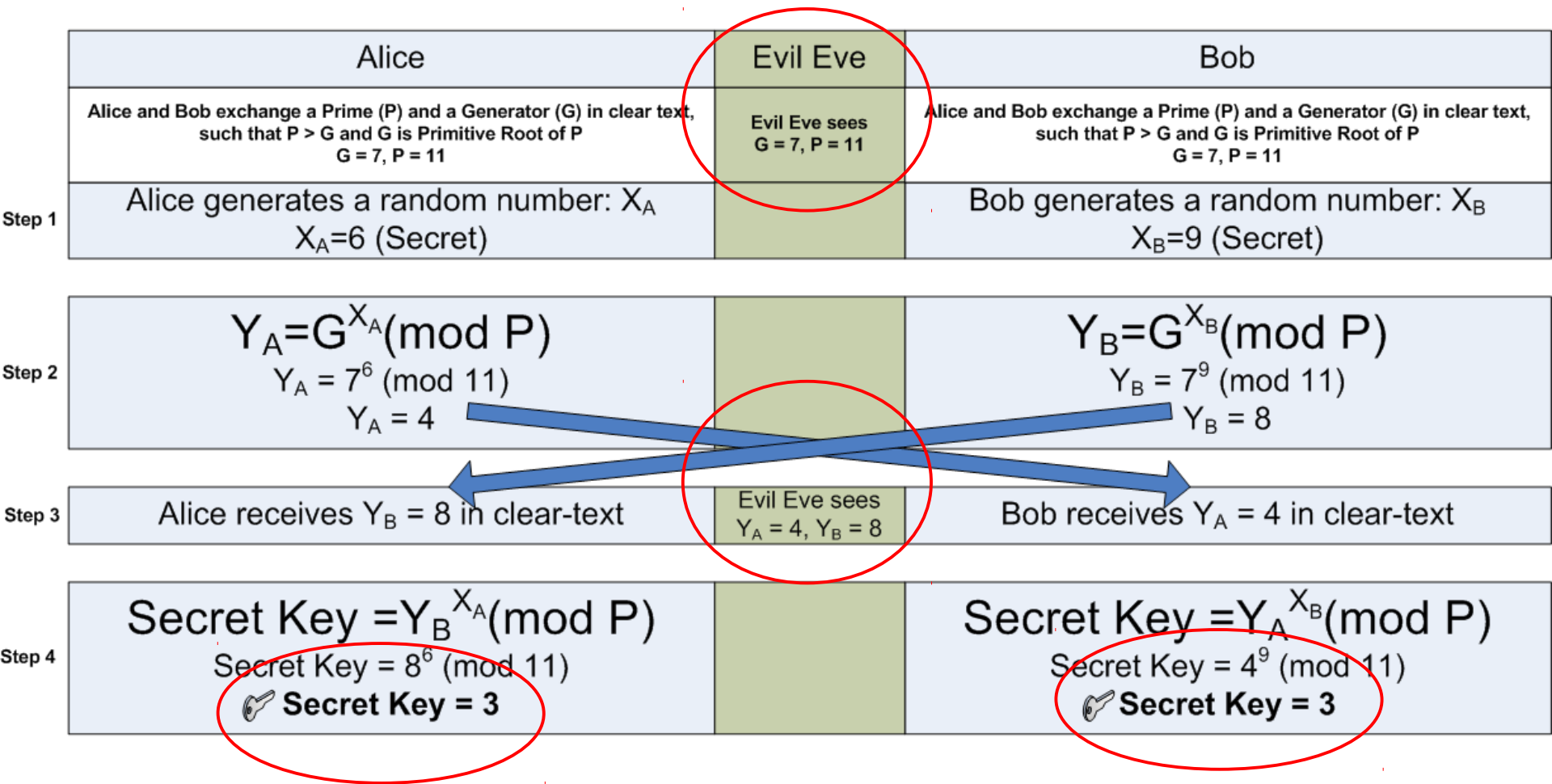
(Or simply Forward Secrecy)

- No master secret is transferred to server
  - Client and server agrees on common secret
  - Communicates using public messages
    - Diffie-Hellman key exchange
- After the session all secrets are discarded

Even if the private key of the server is lost all previous sessions are protected



# Diffie Hellman Key Exchange



# Forward Secrecy at Twitter

Friday, November 22, 2013 | By Jacob Hoffman-Andrews (@j4cob) [21:39 UTC]

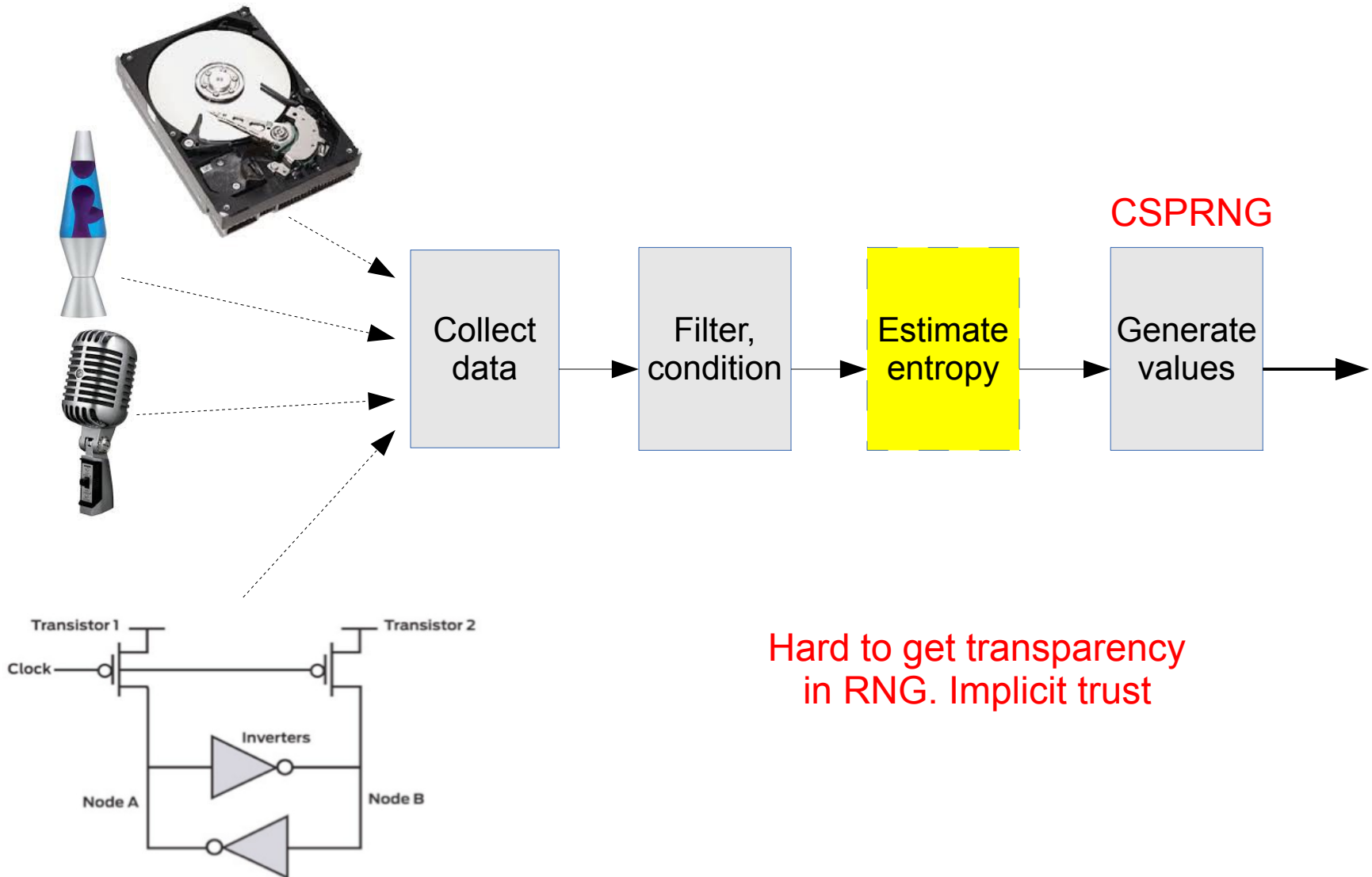
---

Tweet

As part of our continuing effort to keep our users' information as secure as possible, we're happy to announce that we recently enabled forward secrecy for traffic on [twitter.com](#), [api.twitter.com](#), and [mobile.twitter.com](#). On top of the usual confidentiality and integrity properties of HTTPS, forward secrecy adds a new property. If an adversary is currently recording all Twitter users' encrypted traffic, and they later crack or steal Twitter's private keys, they should not be able to use those keys to decrypt the recorded traffic. As [the Electronic Frontier Foundation points out](#), this type of protection is increasingly important on today's Internet.

# It is all about the keys!

# Random Number Generation



TOUR OF ACCOUNTING

OVER HERE WE HAVE OUR RANDOM NUMBER GENERATOR.

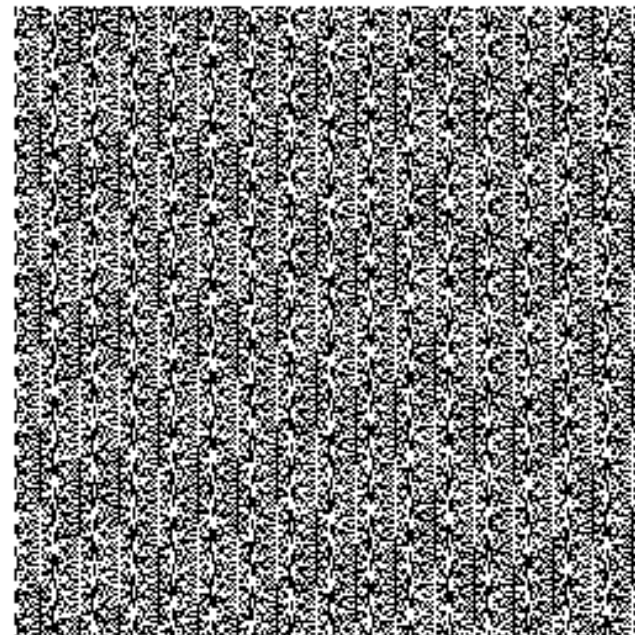
NINE NINE NINE NINE NINE NINE

ARE YOU SURE THAT'S RANDOM?

THAT'S THE PROBLEM WITH RANDOMNESS: YOU CAN NEVER BE SURE.

www.dilbert.com

© 2001 United Feature Syndicate, Inc.



## PHP rand() on Microsoft Windows

# Debian key generator

Linux Debian (Etch, Lenny, Sid) 2006-2008

```
MD_Update(&m,buf,j);  
[ .. ]  
MD_Update(&m,buf,j); /* purify complains */
```

Security audit in Debian using Valgring, Purify  
OKed by OpenSSL dev

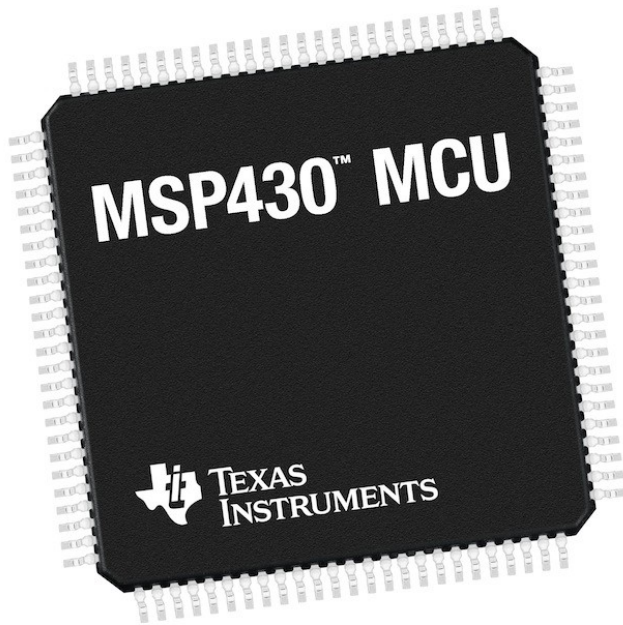


No mixing of random values  
during init

Process ID became seed  
At most ~17 bits strength

Affected SSL/TLS and a lot of  
applications





# Z-Stack

```
air% hexdump random.bin | grep -P color "\7cpe1 e8 4e f4 87"
00000000 02 01+00;60 e8 2e 7c e1 e8 4e f4 87 62 49 56 fe
00080000 01 00 60 e8 2e 7c e1 e8 f4 e f4 87 62 49 56 fe 80
00100000 00 "60 e8 2e 7c e1 e8 4e f4 87 62 49 56" fe 80 00
00180000 60 1e8 2e 7c e1 e8 4e f4 87 f62 a49 )56 fe 80 00 60
air% █ return;
```

Source: T Goodspeed

# Key length and strength

<b>Symmetric</b>	<b>RSA/DLOG</b>	<b>EC</b>
48	480	96
50	512	100
56	640	112
62	768	124
64	816	128
73	1024	146
80	1248	160
89	1536	178
103	2048	206
112	2432	224
128	3248	256
141	4096	282
160	5312	320
192	7936	384
256	15424	512



Table 7.4: Security levels (symmetric equivalent).

Security Level	Security (bits)	Protection	Comment
1.	32	Attacks in “real-time” by individuals	Only acceptable for auth. tag size
2.	64	Very short-term protection against small organizations	Should not be used for confidentiality in new systems
3.	72	Short-term protection against medium organizations, medium-term protection against small organizations	
4.	80	Very short-term protection against agencies, long-term prot. against small organizations	Smallest general-purpose level, $\leq 4$ years protection (E.g. use of 2-key 3DES, $< 2^{40}$ plaintext/ciphertexts)
5.	96	Legacy standard level	2-key 3DES restricted to $\sim 10^6$ plaintext/ciphertexts, $\approx 10$ years protection
6.	112	Medium-term protection	$\approx 20$ years protection (E.g. 3-key 3DES)
7.	128	Long-term protection	Good, generic application-indep. recommendation, $\approx 30$ years
8.	256	“Foreseeable future”	Good protection against quantum computers unless Shor’s algorithm applies.

**RSA 2048** 

Table 7.1: Minimum symmetric key-size in bits for various attackers.

Attacker	Budget	Hardware	Min security
“Hacker”	0	PC	58
	< \$400	PC(s)/FPGA	63
	0	“Malware”	77
Small organization	\$10k	PC(s)/FPGA	69
Medium organization	\$300k	FPGA/ASIC	69
Large organization	\$10M	FPGA/ASIC	78
Intelligence agency	\$300M	ASIC	84

# Side note: Export rules

- Waasenaar agreement
- Limits the usage of strong encryption
  - Symmetric keys: 56 bits
  - RSA keys: 512 bits
  - Elliptic Curve keys: 112 bits
- Anything above these limits requires registration or permit
  - EU, USA, Japan etc – registration
  - North Korea, Iran, China – permit (or NO)

Also depends on usage

End users vs components, equipment, market etc

# Trust Stores

- Where does your app/OS find the CA cert?
- Trust Stores – the root of cert validation
  - DB of CA certs in system, browser, libs
  - Mozilla
  - Apple – OSX, iOS
  - Microsoft
  - Google – Chrome, Oracle/Java
- Can you trust it?
  - Not very transparent, easy to check
    - Source code, blobs, Excel files
  - No info when the stores are updated and why
    - NSA root?

We try to observe  
the trust stores:

<https://github.com/kirei/catt>

# Trust Store examples

## Local CAs

<a href="#">CertiSign</a>	Brazil	Certisign - Autoridade Certificadora - AC2	1024	MD5	Tuesday, June 26, 2018 5:00:00 PM
---------------------------	--------	--	------	-----	-----------------------------------

## Government CAs

<a href="#">China Internet Network Information Center (CNNIC)</a>	China	CNNIC Root	2048	SHA1	4/15/2027
<a href="#">China Internet Network Information Center (CNNIC)</a>	China	China Internet Network Information Center EV	2048	SHA1	Saturday, August 31, 2030 12:11:25 AM

## Big boys being bad

<a href="#">VeriSign</a>	USA	VeriSign Class 1 Public Primary Certification Authority	1024	SHA1	Wednesday, August 02, 2028 4:59:59 PM
--------------------------	-----	---	------	------	---------------------------------------

# Agenda

What is HTTPS / SSL



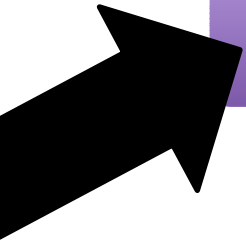
Attacks on SSL



Attacks on related technology

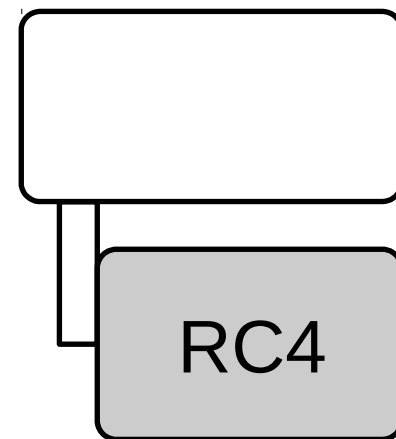
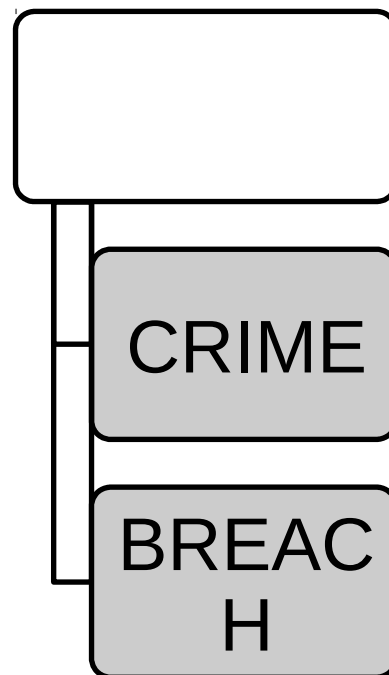
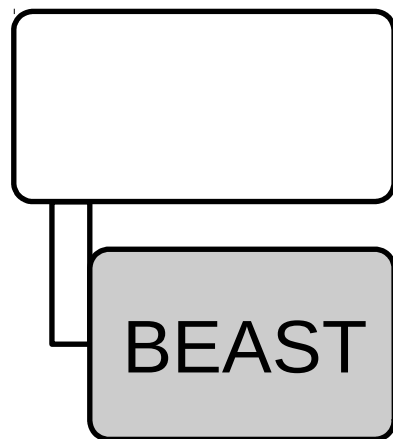
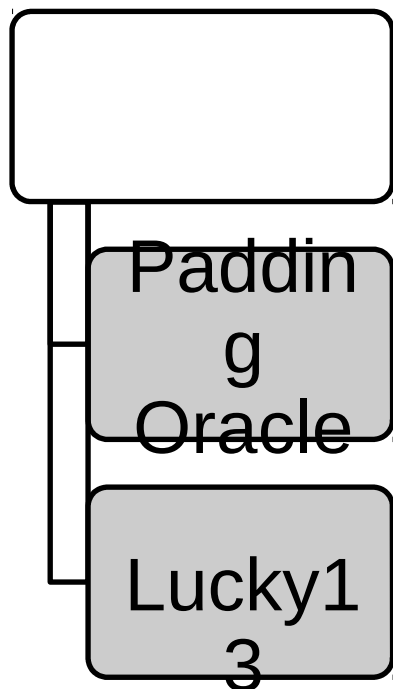


Coding SSL/TLS



# Cryptanalytic attacks against SSL

A. Shamir: "Cryptography is  
typically bypassed, not  
penetrated"





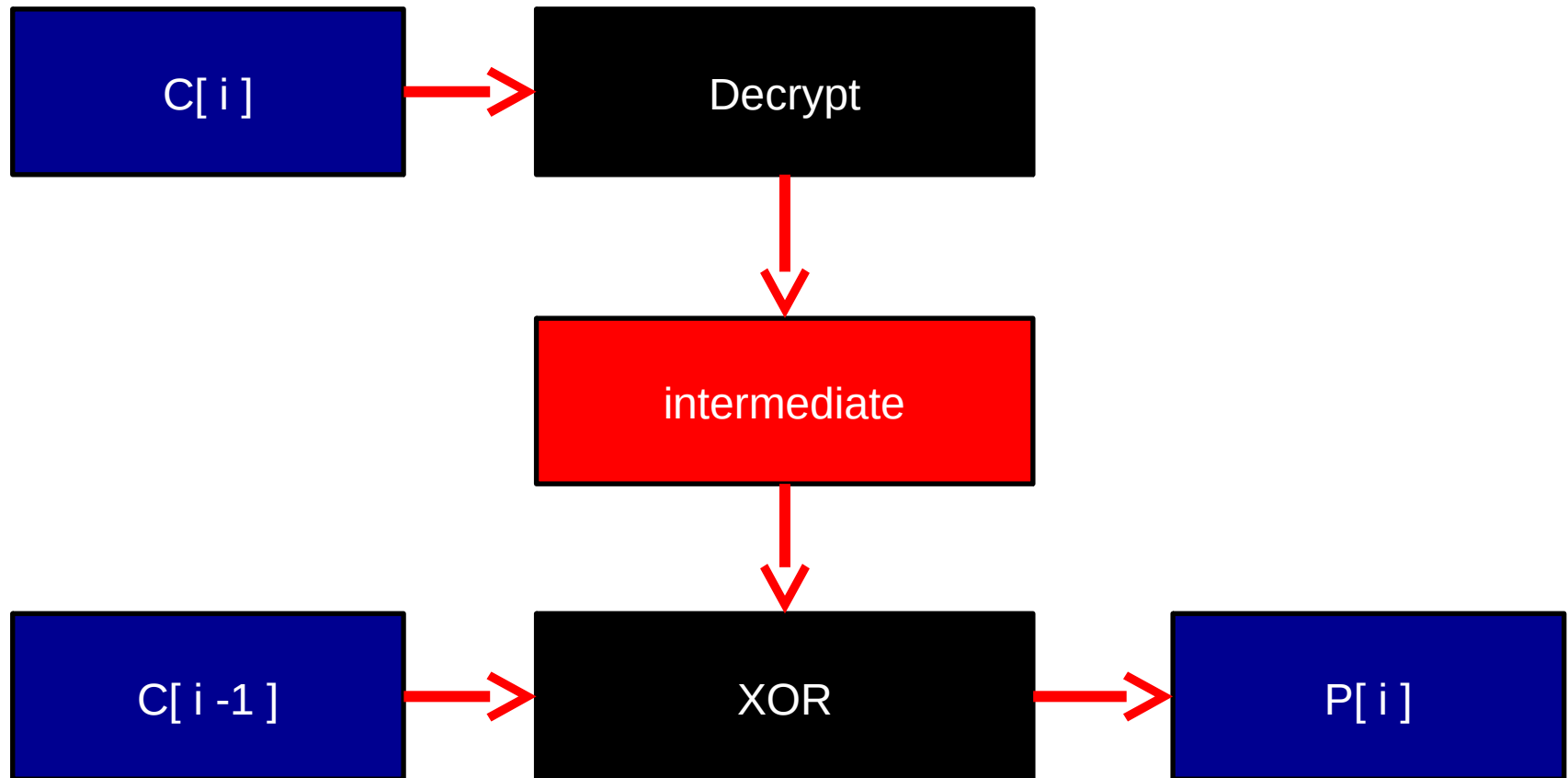
# Padding Oracle & Lucky13

Cryptanalytic attacks against SSL

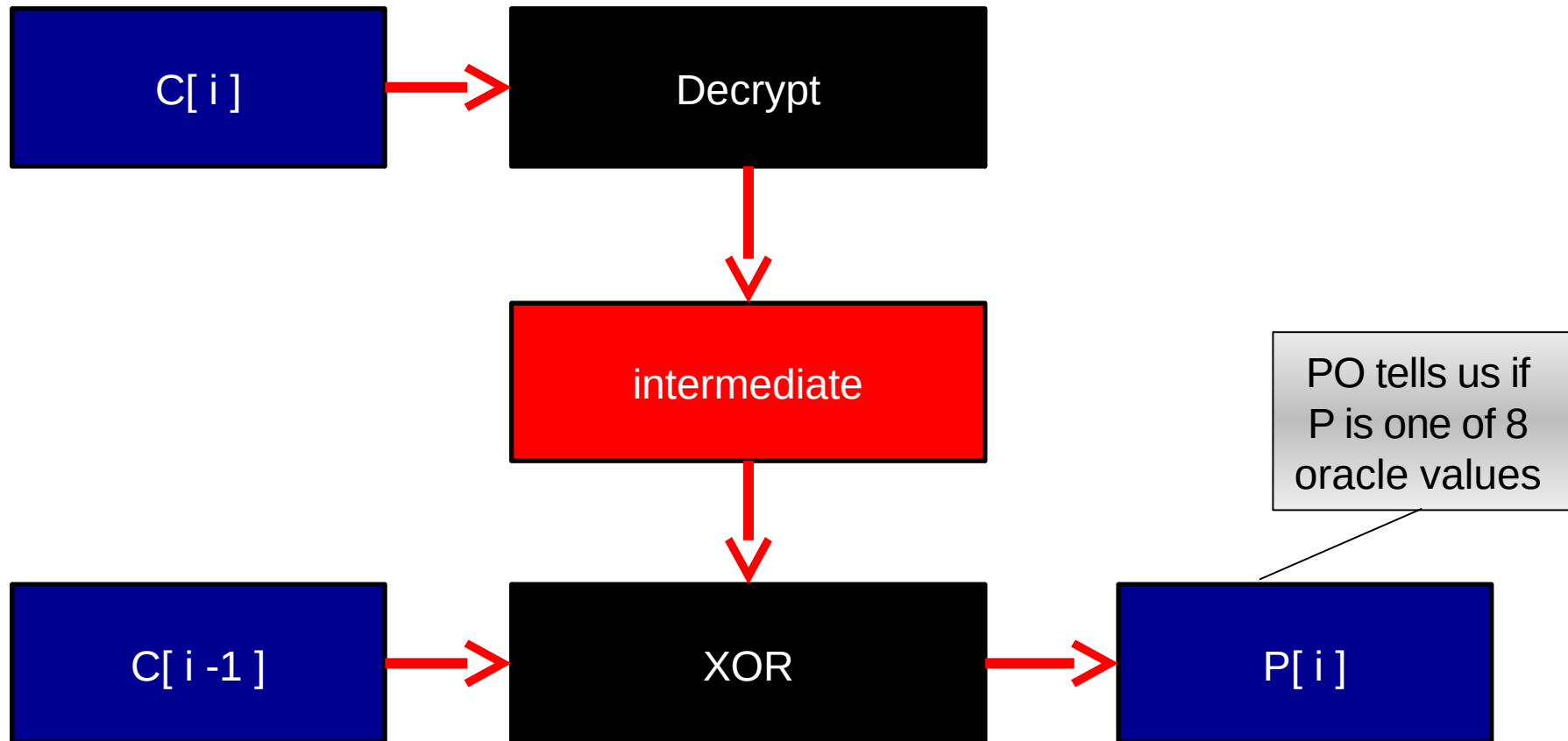
# Padding Oracle (2002)

- Padding: terminating last plaintext block
  - 1337 => 1337060606060606 PKCS5Padding
  - 1337 => 1337000000000000 ZeroPadding
- PKCS5 Padding Oracle
  - "OK" if message ends with either of:
    - 0808080808080808 **xx**07070707070707
    - xxxx**060606060606 **xxxxxx**0505050505
    - xxxxxxxx**04040404 **xxxxxxxxxx**030303
    - xxxxxxxxxxxxxxxx**0202 **xxxxxxxxxxxxxxxxxx**01

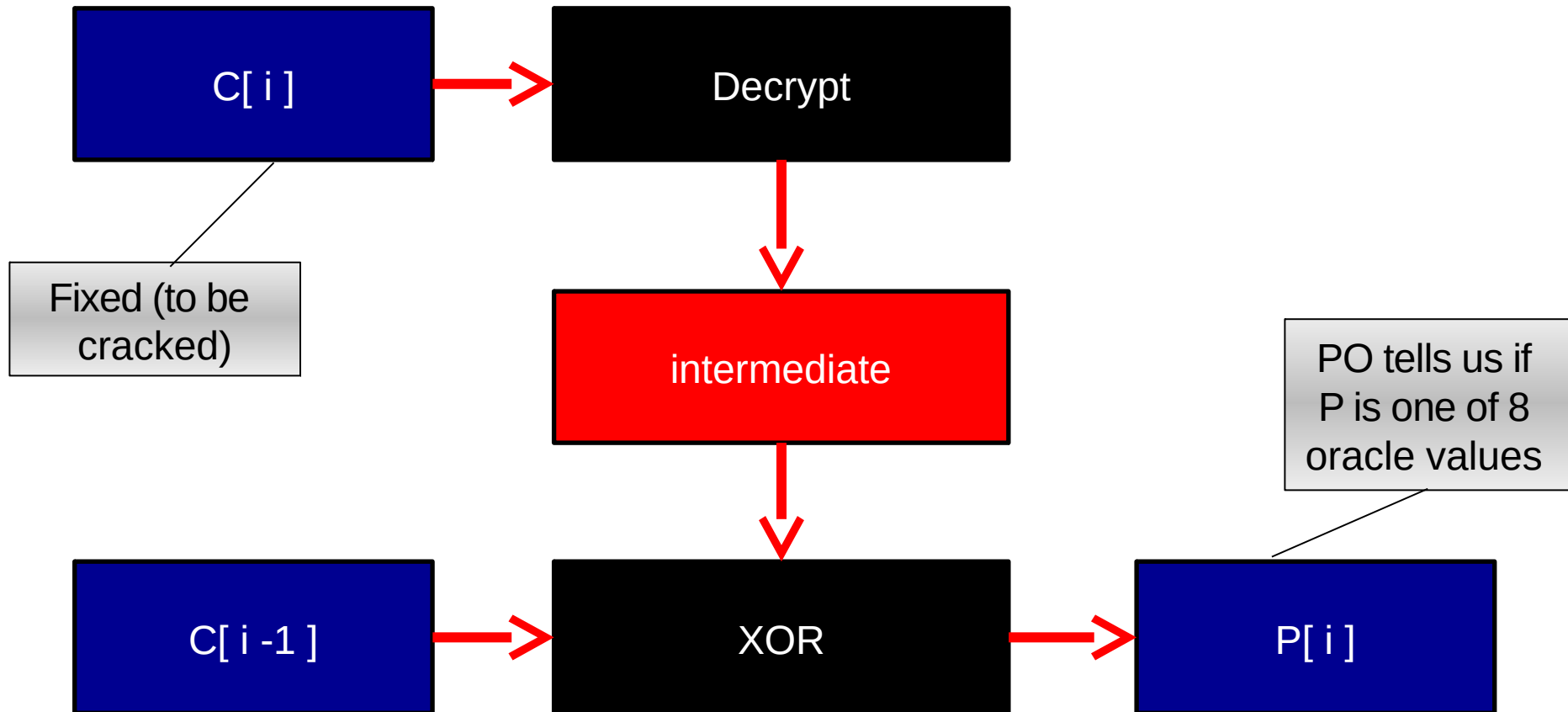
# CBC & Padding Oracle



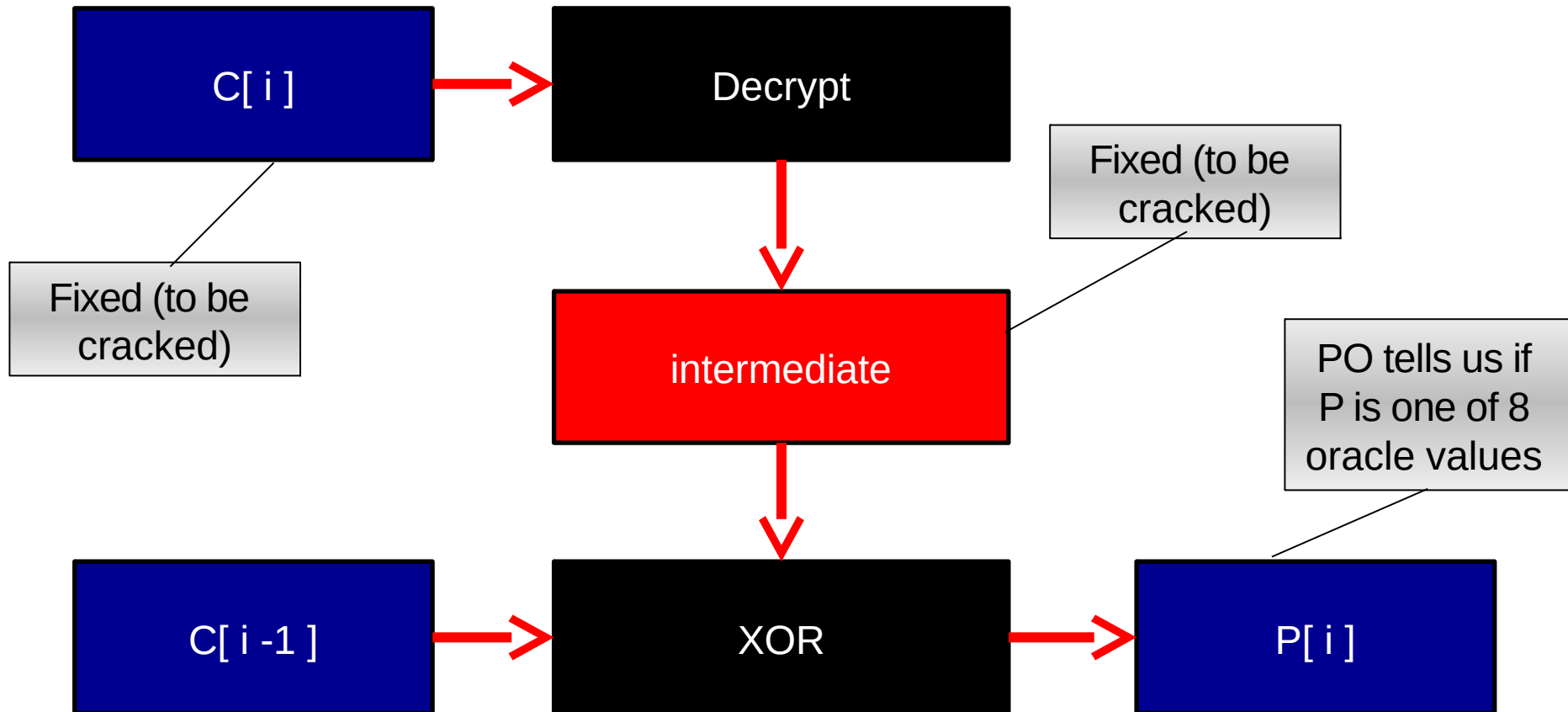
# CBC & Padding Oracle



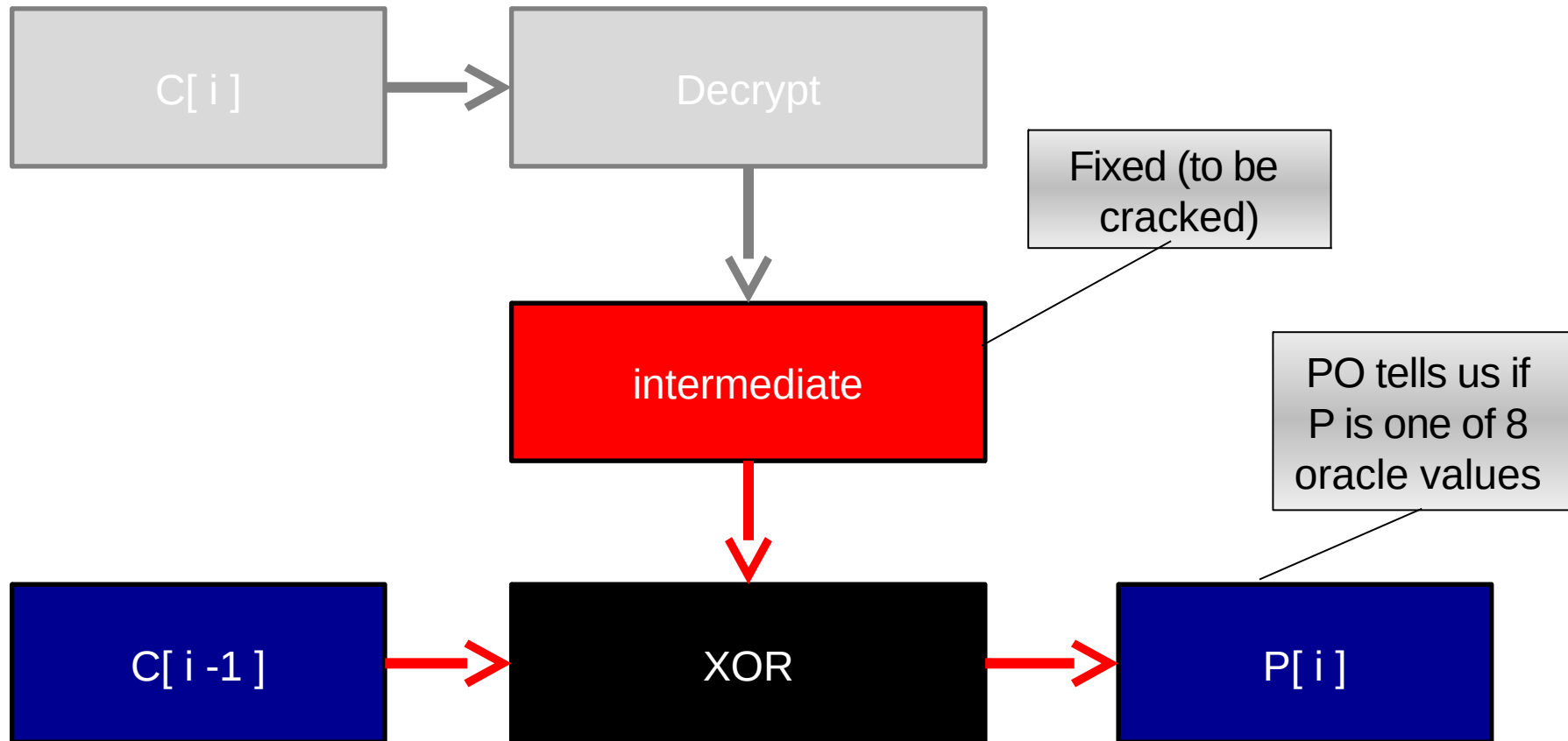
# CBC & Padding Oracle



# CBC & Padding Oracle

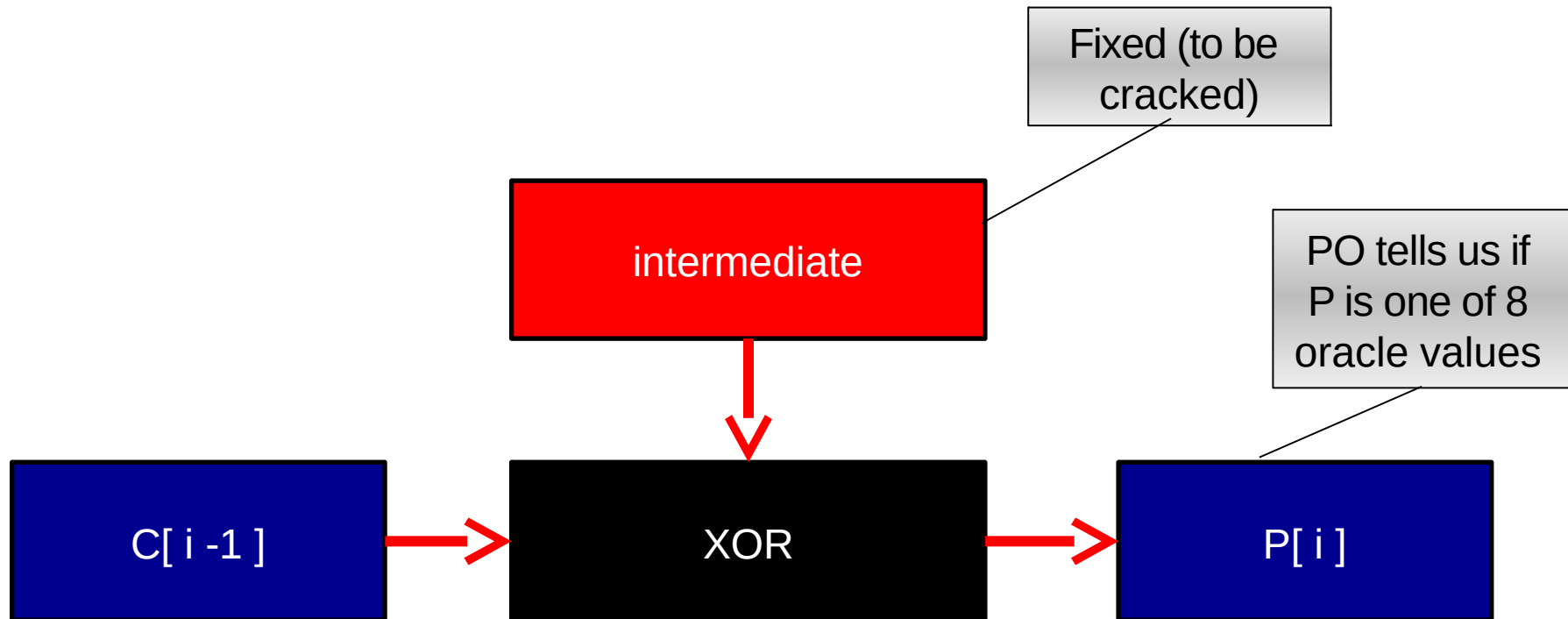


# CBC & Padding Oracle

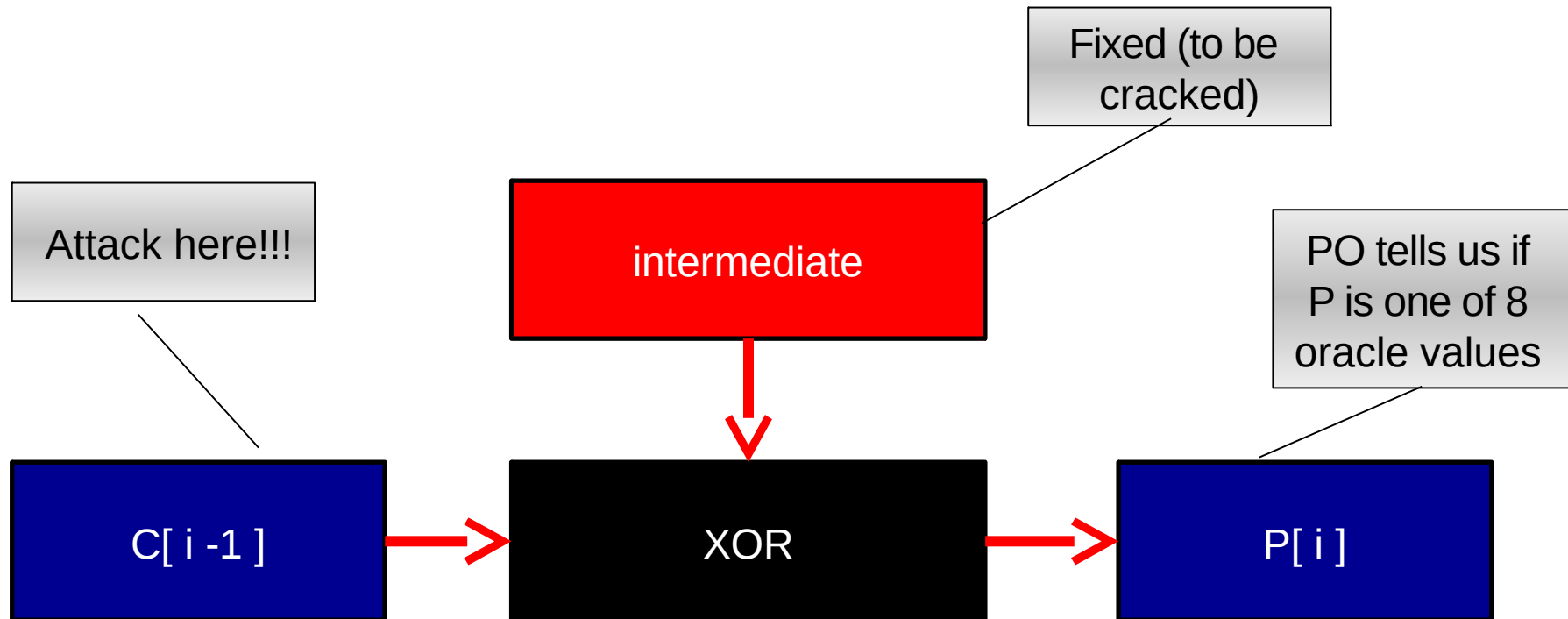




# CBC & Padding Oracle



# CBC & Padding Oracle



# Padding Oracle (2002)

- CBC + PKCS5 Oracle Attack
  - Attacker replace  $C[i-1]$  with  $R$ , can choose any  $R$
  - Attacker replace  $P[i]$  with unknown  $P$
  - intermediate =  $R \text{ XOR } P$  when  $P=0808080808080808$
  - Padding Oracle:  $P$  is correct upon this  $R$
  - Padding Oracle leaks clues about intermediate

# Padding Oracle (2002)

- CBC + PKCS5 Oracle Attack
  - Start with  $R = 0$  (or a random number), change LSB until Padding Oracle says OK!
  - ~128 messages find  $R$ :  $P = \text{xx}01 \Rightarrow \text{OK!}$
  - ~256 messages find  $R$ :  $P = \text{xx}0202 \Rightarrow \text{OK!}$
  - ~1024 messages find  $R$ :  $P = 0808080808080808 \Rightarrow \text{OK!}$
  - $\text{intermediate} = R \text{ xor } 0808080808080808$
  - $P[i] = C[i] \text{ xor } \text{intermediate}$

# Padding Oracle (2002)

- Why did padding Oracle work?
- SSL/TLS CBC cipher suits are broken
  - Authenticates AFTER decryption
- Real long term solution: new cipher suits
  - Authenticated Encryption (AEAD)
  - Authenticate BEFORE decryption

# Padding Oracle (2002)

- How was Padding Oracle fixed?
- Workaround: obfuscate the padding oracle
  - Decode as if Zero Padded
  - Failed padding => slightly longer plaintext
  - Calculate MAC
  - Fail with nearly the same execution time
  - pretends MAC failed, not padding

# Padding Oracle (2002)

TLS RFC;

"This leaves **a small timing channel**, since MAC performance depends to some extent on the size of the data fragment, but it **is not believed** to be large enough to **be exploitable**, due to the large block size of existing MACs and the small size of the timing signal. "



# Lucky 13 (2013)

- Padding Oracle returns!
  - SSL header size (13 bytes) is nearly optimal for attacking the ZeroPad + HMAC timing channel
- The small time difference can be detected
  - Many attempts & Statistical models
  - Low latency between attacker and victim required
- Second Padding Oracle / Lucky13



workaround:

Open Web Application  
Security Project

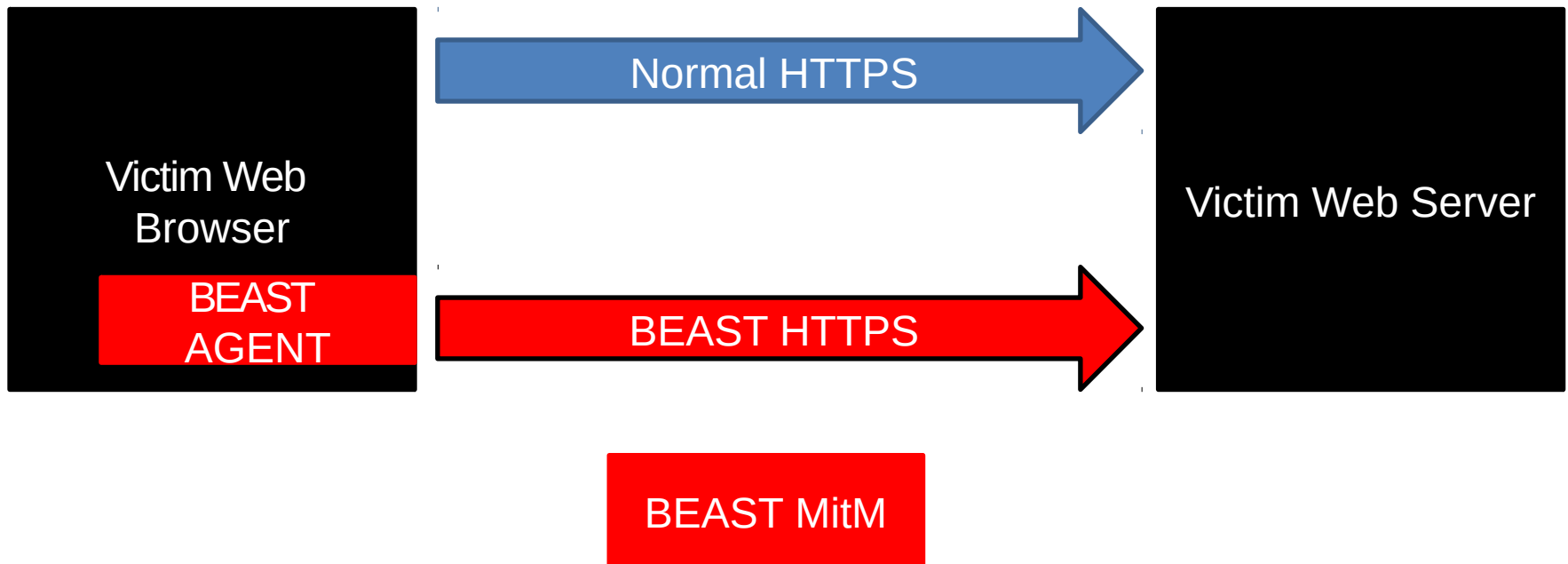
Peter Magnusson, Twitter: @Blaufish\_  
Sakerhetspodcasten.se & Omegapoint.se

–Complex coding solution to ensure constant

# BEAST

Cryptanalytic attacks against SSL

# BEAST (2011)



# BEAST (2011)

- Browser Assisted Exploitation Against SSL/TLS
  - Decrypts a message, usually a HTTP Cookie.
  - Fools the browser to attack its own encryption
- Clever SOP rule bypass
  - BEAST agent injected into HTTP://victim
  - BEAST agent attacks HTTPS://victim
  - BEAST agent doesn't know cookie itself, but it is included in HTTPS sent from BEAST agent

# BEAST (2011)

- Browser Assisted Exploitation Against SSL/TLS
  - aka "Here come the XOR Ninjas".
- CBC Cipher Block Chaining mode
- Similar XOR-attack as seen in padding oracle
  - Send  $P^* = C^* \text{ XOR IV XOR } (R||i)$
  - If one byte of  $C^* == C$ , we know 1 byte of real

# CRIME & BREACH

Cryptanalytic attacks against SSL

# CRIME (2012)

GET /abcdefgh  
Cookie: JSESSIONID=5eb63bbbe01eed



Compressed

GET /5eb63bbb  
Cookie: JSESSIONID=5eb63bbbe01eed



Compressed

# CRIME (2012)

- Compression Ratio Info-leak Made Easy
- Attack HTTPS SSL/TLS compression
  - GET /abc Cookie:  
JSESSIONID=5eb63bbbe01eed
  - GET /5eb Cookie:  
JSESSIONID=5eb63bbbe01eed
  - Shorter compressed message == guess is better
- SOP bypass not needed to launch CRIME





# CRIME (2012)

- Various SSL changes to mitigate CRIME
  - Many browsers refusing SSL Compression
  - Some web servers refusing SSL Compression
- Jay! CRIME fixed, all done
  - But... SSL Compression is not the most common form of compression

# BREACH (2013)

```
<H1>Your name is: abcdefgh</H1>  
<input hidden name="anti_csrf_token"  
value=5eb63bbbe01eeed">
```

Compressed

```
<H1>Your name is: 5eb63bbbe</H1>  
<input hidden name="anti_csrf_token"  
value=5eb63bbbe01eeed">
```

Compressed

# BREACH (2013)

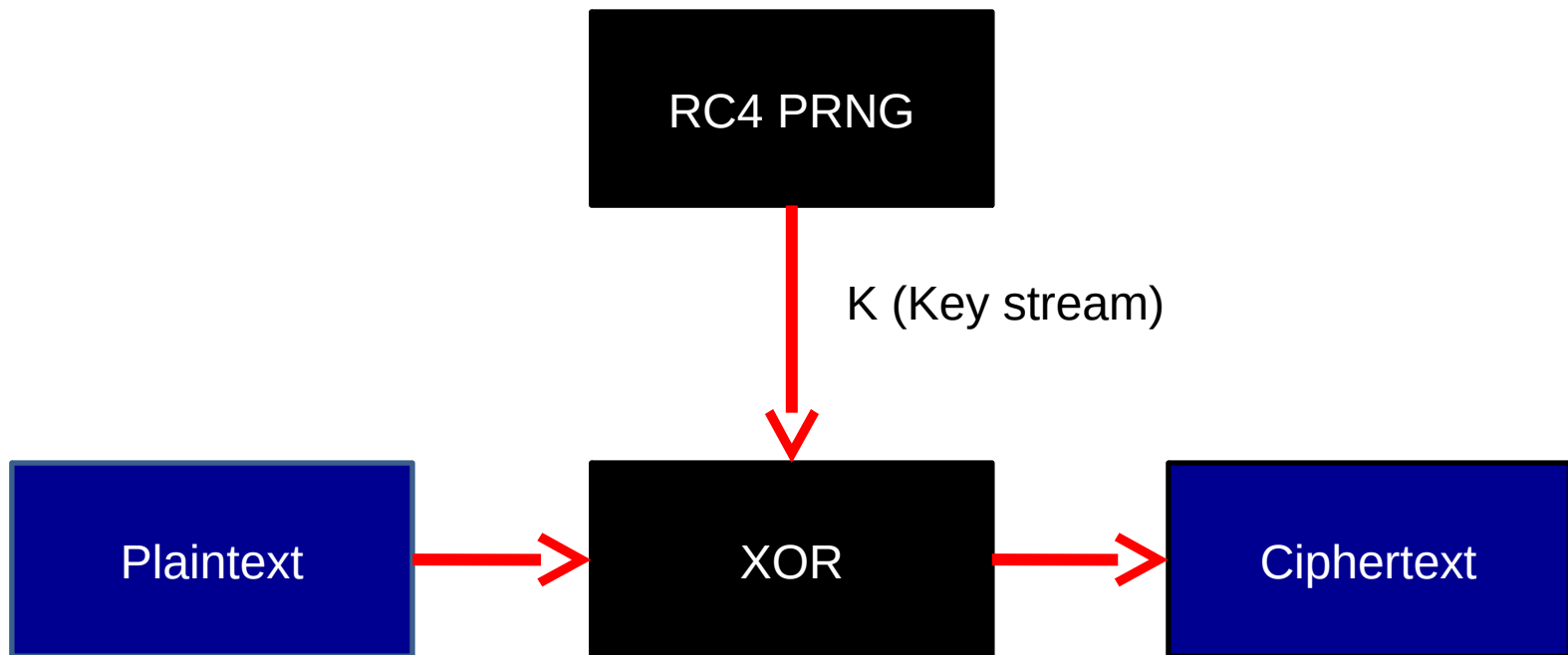
- BREACH attacks HTTP response body:
  - HTTP Compression must be enabled
  - Response contains a secret
  - Response reflects attacker input
- CRIME vs BREACH:
  - Both use compression oracles to decrypt HTTPS
  - CRIME exploit SSL/TLS compression of request
  - BREACH exploit HTTP compression of

# Weak Algorithms

Cryptanalytic attacks against SSL

# RC4 (2013)

- Designed 1987, very fast & useful



# RC4 (2013)

- Known to be weak since 2001
  - Bias (not uniform random) key stream
  - Key stream leaks some of internal state
  - WEP completely broken due to RC4 & bad design
- ...but RC4 flaws did not affect SSL/TLS (?)

# RC4 (2013)

- Borderline practical attacks emerge
- RC4 for SSL/TLS can be broken
  - Assuming a lot of messages
  - Assuming a lot of time
  - ANY improvement on these attacks => practical
- There is no dirty workaround for RC4
  - Not a SSL/TLS protocol issue
  - Fundamental algorithm flaw, it must be removed

# Agenda

What is HTTPS / SSL



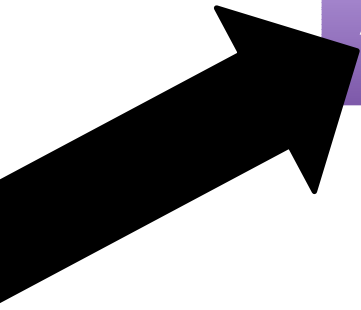
Attacks on SSL



Attacks on related technology

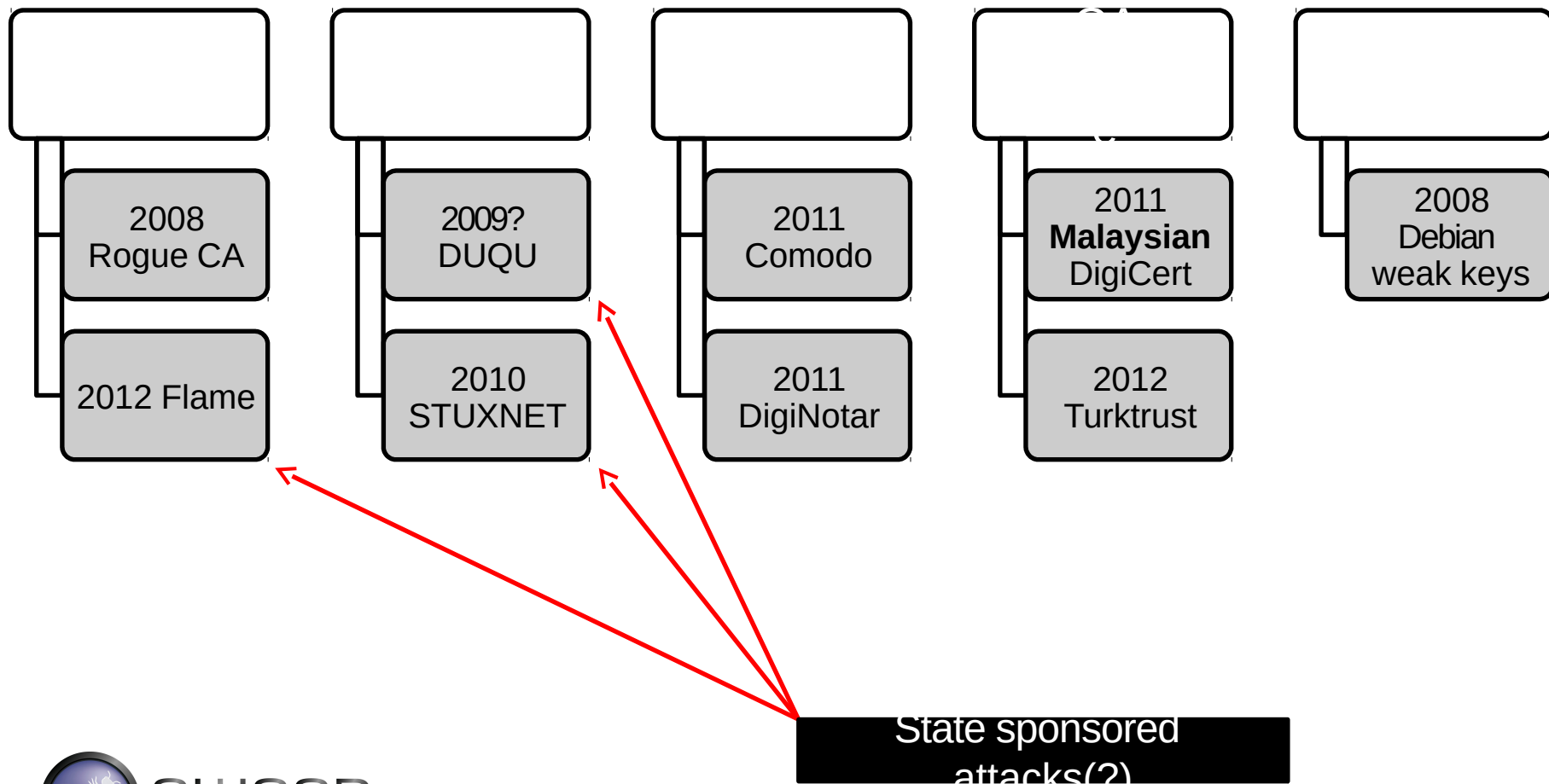


Coding SSL/TLS



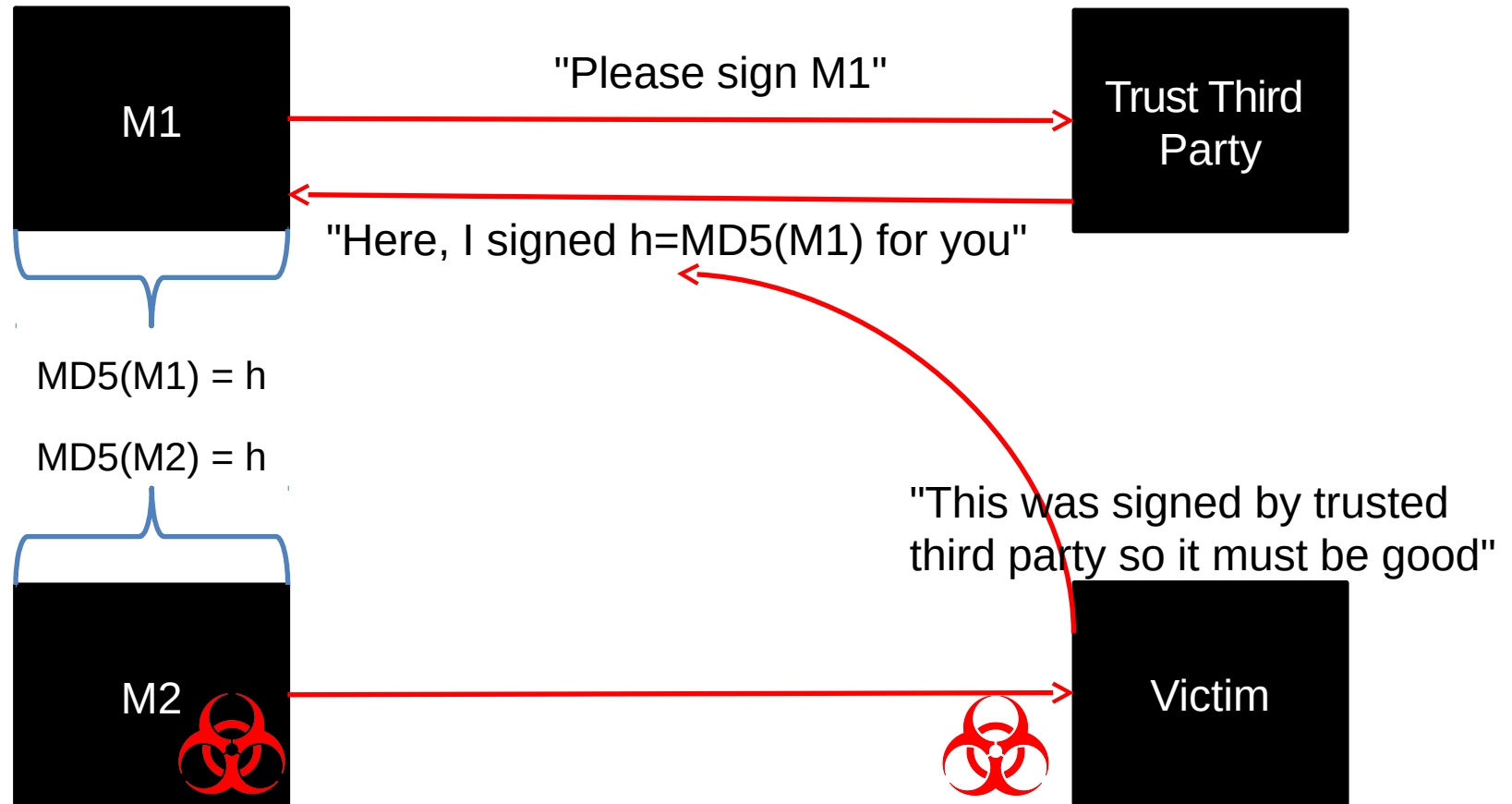


# Related attacks



# Related attacks: MD5 Pre-image

# MD5 Pre-image



# Rogue CA certificate (2008)

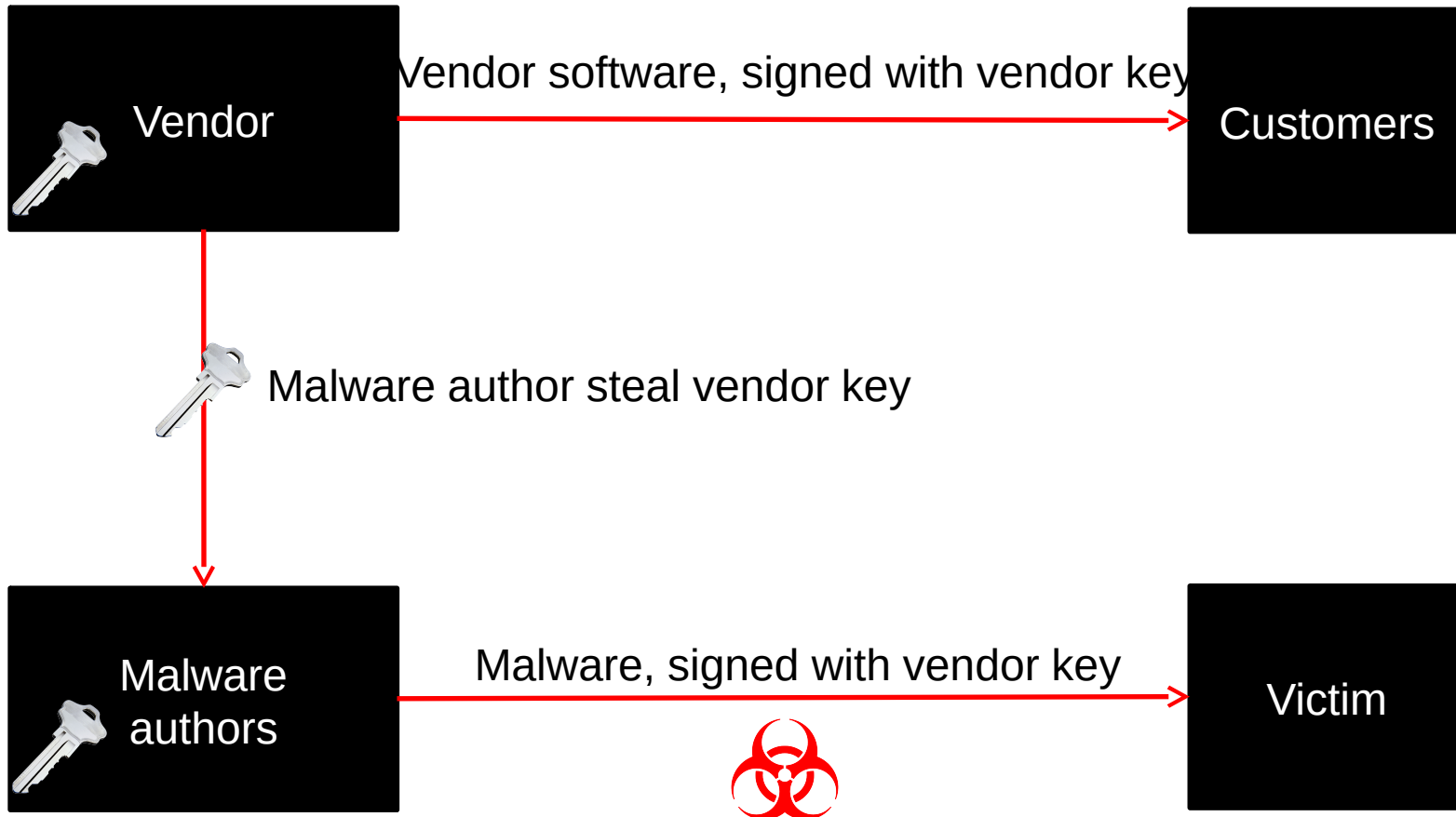
- Academic team of security researchers
- Supercomputer: 200+ PlayStation 3
- MD5 second-preimage attack
  - $M1 \neq M2, H(M1) == H(M2)$
  - RapidSSL, MD5
  - $M1$  = basic constraint  $CA=FALSE$
  - $M2$  = basic constraint  $CA=TRUE$

# Flame (2012)

- State sponsored malware?
- MD5 second-preimage attack
  - $M1 \neq M2, H(M1) == H(M2)$
  - $M1$  = Microsoft Terminal Services license
  - $M2$  = Code signing cert valid in Window Update
- Similar but NOT SAME as Rouge CA attack!
- Cryptanalyst team required to perform

# Related attacks: Stolen keys

# Stolen keys





# DUQU – stolen crypto keys

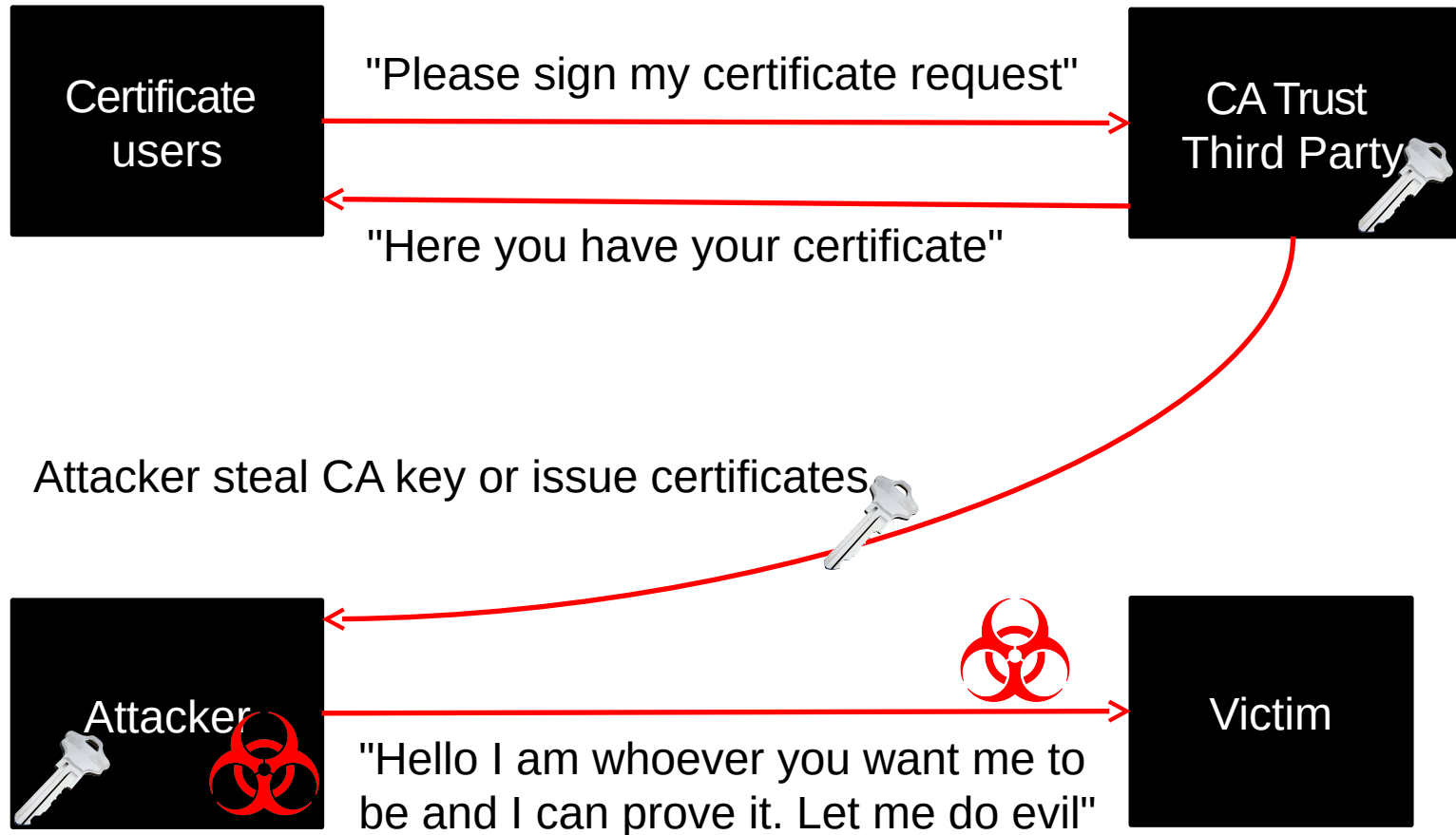
- State sponsored malware?
- Signed with stolen code sign certificate
  - C-Media Electronics, Inc. (certificate issued 2009, signature not time stamped)
- Purpose
  - Key logger & attacks targeting small CA's according to McAfee. Espionage?

# STUXNET – stolen crypto keys

- State sponsored malware?
- Signed with stolen code signing certificate
  - Realtek Semiconductor - Jan 25, 2010
  - JMicron Technology Corp - July 14, 2010
- Purpose
  - Disrupt Iranian nuclear enrichment

# Related attacks: CA Breach

# CA Breach



# Comodo (2011)

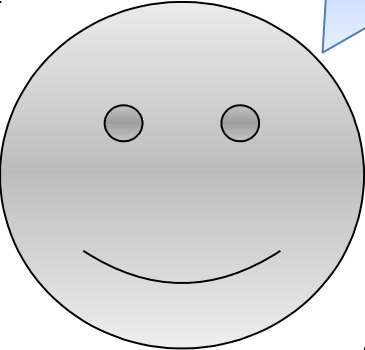
- State sponsored attack, or only hackers?
- An affiliate Registration Authority (RA) hacked
- 9 certificates fraudulently issued
  - Revoked

# DigiNotar (2011)

- State sponsored attack or only hackers?
- Iranian Man in the Middle attack
  - MitM detected by Google Chrome browsers
  - DigiNotar hacked by Iranian hackers
- 531 or more fake certificates signed
- Trust revoked, DigiNotar files for

# Related attacks: CA Malpractice

# CA Malpractice



Hello CA! Do you do  
your important job  
well?

DERP DERP DERP





# Digicert **Malaysia** (2011)

- Not an attack; severe malpractice
  - 512 bit RSA keys (crackable by ordinary criminals)
  - Certs issued w/o extensions (effectively CA, code signing, Server auth, etc... "do everything")
- Trust revoked for Digicert **Malaysia**
  - Small CA name squatting on Digicert. Why would browsers etc allow different CAs with



same name?

OWASP

Open Web Application  
Security Project

– Some zealots removed all Digicert, breaking the

Peter Magnusson, Twitter: @Blaufish\_

Sakerhetspodcasten.se & Omegaint.se

# Turktrust (2012 XMAS!)

- Not an attack; severe malpractice
  - Accidentally issued intermediate CA to Turkish gov in 2011. Didn't clean up when becoming aware.
- Accidental Man-in-the-Middle Attack
  - Christmas December 24 2012
  - Turkish gov accidentally install intermediate CA from Turktrust into SSL inspecting firewall
  - Google Chrome users alert Google to MitM

# Trust Post Snowden

# Snowden revelations

- Open algorithms are in general good (hard for NSA)
  - AES, Diffie-Hellman, Curve25519, Blake
  - Communication security in general works
- NSA targets implementations
  - Looks for weaknesses
  - Influence/strong arm of implementations
    - SW, HW and systems
- NSA targets std development
  - NIST, ISO, ETSI/SAGE, IEEE
  - IETF (SSL/TLS)
- NSA targets data at rest/in use
  - Cloud services – Google, Apple, MS etc

**Math works!**

**Goal:**  
**Find or create easier  
methods to gain access**

**Track users and  
communication**

# What is suspect ?

- Random generators

- Intel True RNG (Bull Mountain) in Ivy Bridge, Haswell
- NIST specified Dual\_EC\_DRBG

**Suspect backdoor now confirmed**

- Algorithms

- NIST DES/3DES
- NIST SHA-1
- NIST, IEEE Elliptic Curves
- SAGE 3G, 4G algorithms
- China SMS4, ZUC
- Russia GOST

**NSA huge patent owner for ECC. Big influence in std.**

- Protocols

- SSL/TLS

- Implementations

- HSMs
- Closed Source libs, applications, systems
- MS, Cisco, IBM etc

**Can we trust security that is not open and has no transparent background?**

# What is the risk?

- Random generators
  - Weak keys, no real random numbers
- Algorithms
  - Weak algorithms – not expected strength
- Protocols
  - Key leakage, session hijacking
- Implementations
  - Backdoors, unauthorized access

**Is NSA really  
the main adversary?**

**Weaknesses are blind  
can be used by other  
adversaries**

# What is being done?

- Random generators
  - Push to open entropy source for Bull Mountain
  - Bull Mountain not replacing CSPRNG in Linux
  - Create open, transparent HSM
- Algorithms
  - Push to define new EC curves
  - Replace RC4 in SSL/TLS with modern, open stream cipher
  - Move away from dependency of NIST
- Protocols
  - Reevaluation of SSL/TLS and other IETF sec standards
- Implementations
  - Several efforts to audit, evaluate, validate sec implementations
  - MS scramble to regain trust

**NIST has lost a huge  
amount of trust**

**SHA-3**

**Fundraiser to audit TrueCrypt**  
**<https://www.fundfill.com/fund/TrueCryptAudited>**  
**16k USD raised to date**

# What can you do?

- Random generators and keys
  - **Test you generator and generated keys**
  - Use longer keys (though it has costs)
- Algorithms
  - Move away from RC4, DES/3DES – AES
  - Move away from MD5, SHA-1 to SHA-2 (256, 384, 512)
  - Be wary of ECC.
- Protocols
  - Move towards TLS 1.x
- Implementations
  - Use open implementations and libs
  - **Test your applications and systems**
  - Consider where you store (systems, services) stuff

**Perfect Forward Secrecy  
uses ECDSA**

**Normal security strategy**

**Need to be done anyway**



# Agenda

What is HTTPS / SSL



Attacks on SSL



Attacks on related technology

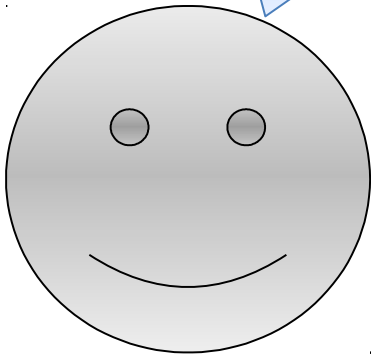


Coding SSL/TLS

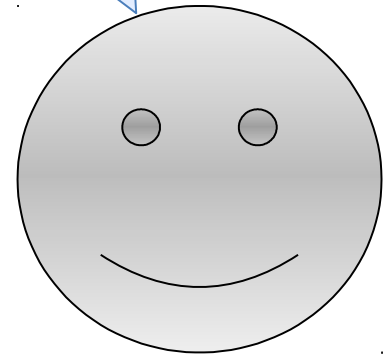
# Developing SSL/TLS code is hard

# Developing SSL/TLS code is hard

Please make changes to this huge security stack which is poorly documented, you barely understood, and throws strange error messages upon failures



Sure!  
How hard can it be?



# Why is developing SSL/TLS code hard?

- There are so many security aspects to authentication and SSL/TLS, that very few fully understand all of it.
  - Authentication: There ~ 14 rules to be checked, probably more!  
[http://en.wikipedia.org/wiki/Certification\\_path\\_validation\\_algorithm](http://en.wikipedia.org/wiki/Certification_path_validation_algorithm)
  - Failure: everything works great (only insecure)
- Your defaults are HOPEFULLY pretty

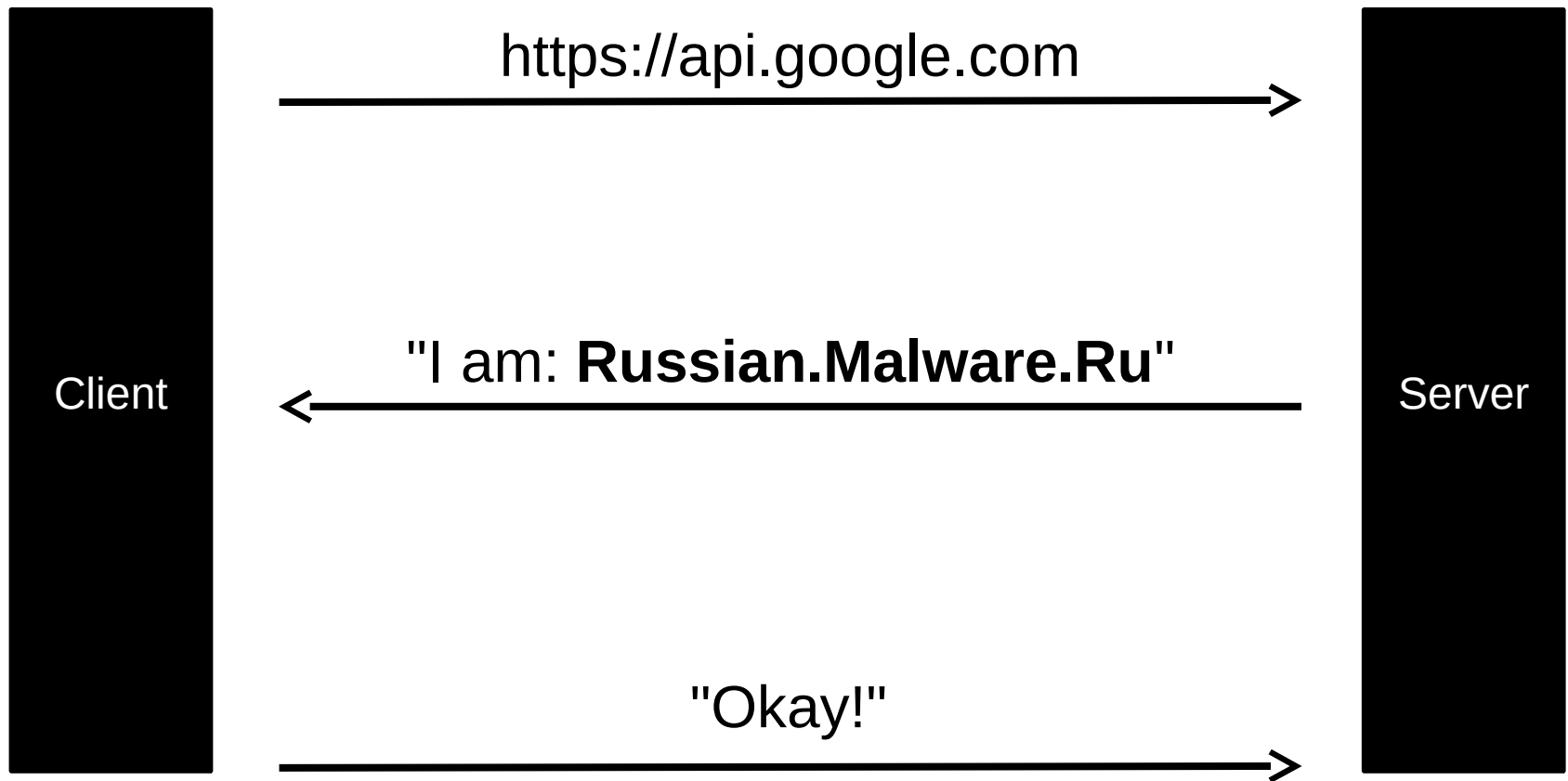
# Why is developing SSL/TLS code hard?

- BE PARANOID
  - Test test test
  - Think through many times what the code does
- Understand incomprehensive errors
  - don't rush code changes to deal with SSL errors
  - SSL error are usually logical, technical and security oriented. Unhelpful but CAN be understood.

# Classic WTF Horror

BEWARE this section contain DO  
NOTs you should NOT copy into  
your own code

# Classic WTF Horror



# Classic WTF Horror

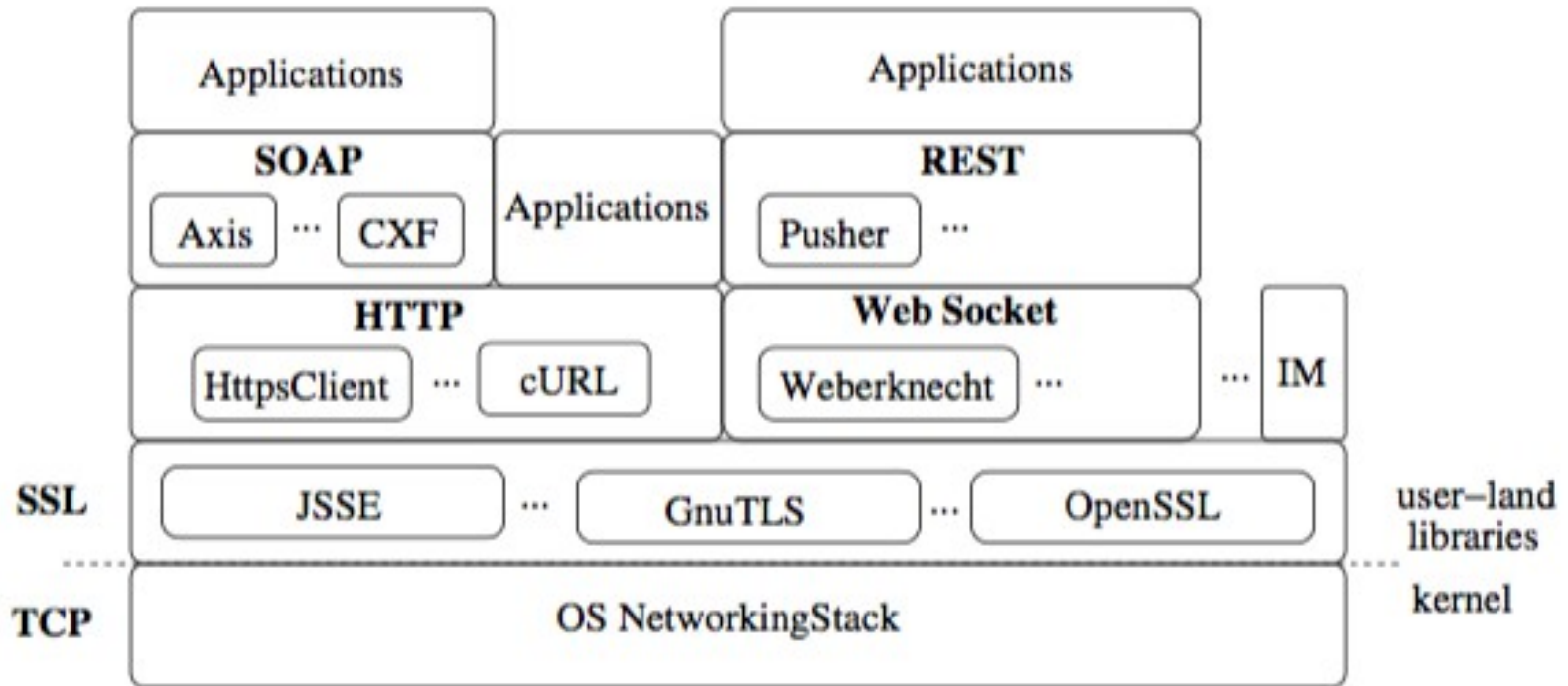
```
HttpsURLConnection.setDefaultHostnameVer  
ifier(new HostnameVerifier() {  
    public boolean verify(String s,  
        SSLSession sslSession) {  
        return true;  
    }  
});
```

- Any valid certificate for ANY site may spoof the site
- Sadly, this is "best answer" in various



# Stacks and libraries

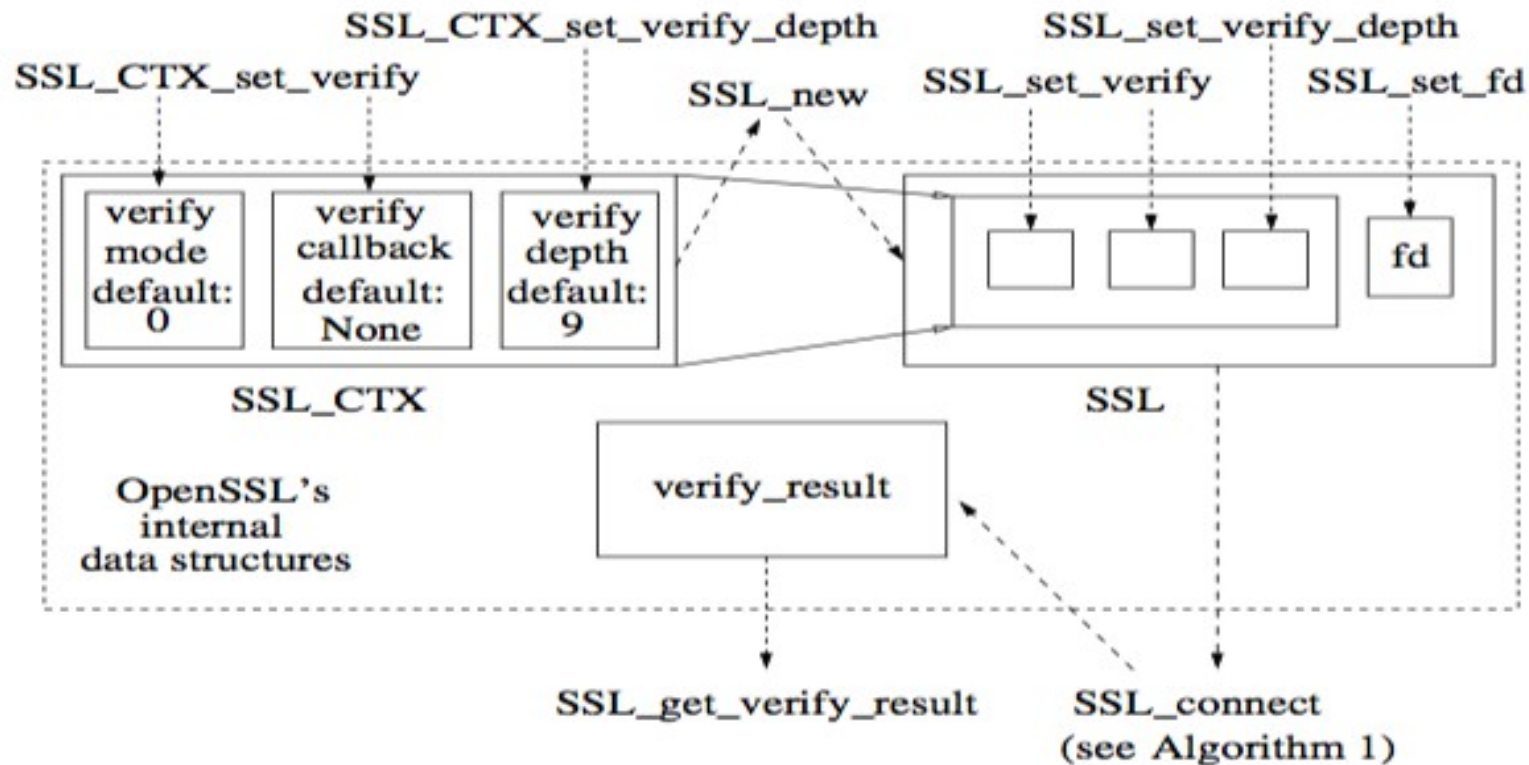
# SSL/TLS used insecurely



**Figure 2: Protocol stack.**

SSL used through layers with different APIs

# SSL/TLS used insecurely



**Figure 3: OpenSSL API for setting up SSL connections with the default chain-of-trust verification.**

Complex APIs makes for easy mistakes

# SSL/TLS used insecurely

This interface is almost perversely bad. The `VERIFYPEER` parameter is a boolean, while a similar-looking `VERIFYHOST` parameter is an integer. The following quote from the `cURL` manual explains the meaning of `CURLOPT_SSL_VERIFYHOST`:

1 to check the existence of a common name in the SSL peer certificate. 2 to check the existence of a common name and also verify that it matches the hostname provided. In production environments the value of this option should be kept at 2 (default value).

Well-intentioned developers not only routinely misunderstand these parameters, but often set `CURLOPT_SSL_VERIFYHOST` to `TRUE`, thereby changing it to 1 and thus accidentally disabling hostname verification with disastrous consequences (see Section 7.1).

```
curl_setopt($curlHandle, CURLOPT_SSL_VERIFYPEER, true);  
curl_setopt($curlHandle, CURLOPT_SSL_VERIFYHOST, true);  
...
```

Validation  
control in  
cURL

Example code  
from Amazon  
FPS (PHP)

# SSL/TLS used insecurely

## 7.3 PayPal IPN in ZenCart

ZenCart's functionality for PayPal IPN shows a profound misunderstanding of cURL's parameters. It disables certificate validation entirely, yet attempts to enable hostname verification—even though the latter has no effect if certificate validation is disabled.

```
$curlOpts=array( ...  
    CURLOPT_SSL_VERIFYPEER => FALSE,  
    CURLOPT_SSL_VERIFYHOST => 2  
    ... );
```

Even worse example code  
provided by PayPal

# SSL/TLS used insecurely

```
def _verify_hostname(self, hostname, cert):  
    # Verify hostname against peer cert  
    # Check both commonName and entries in subjectAltName,  
    # using a rudimentary glob to dns regex check  
    # to find matches  
  
    common_name = self._get_common_name(cert)  
    alt_names = self._get_subject_alt_names(cert)  
  
    # replace * with alphanumeric and dash  
    # replace . with literal .  
    valid_patterns = [re.compile(pattern.replace(r".", r"  
        \.")).replace(r"*", r"[0-9A-Za-z]+"))  
    for pattern  
        in (set(common_name) | set(alt_names))  
    ]  
  
    return any(  
        pattern.search(hostname)  
        for pattern in valid_patterns  
    )
```

Hostname  
validation  
in  
Apache  
Libcloud

google.com = \*oogle.com  
moogle.com, scroogle.com are **BAD** domains

# Handling SSL/TLS dev challenges

# Handling SSL/TLS dev challenges

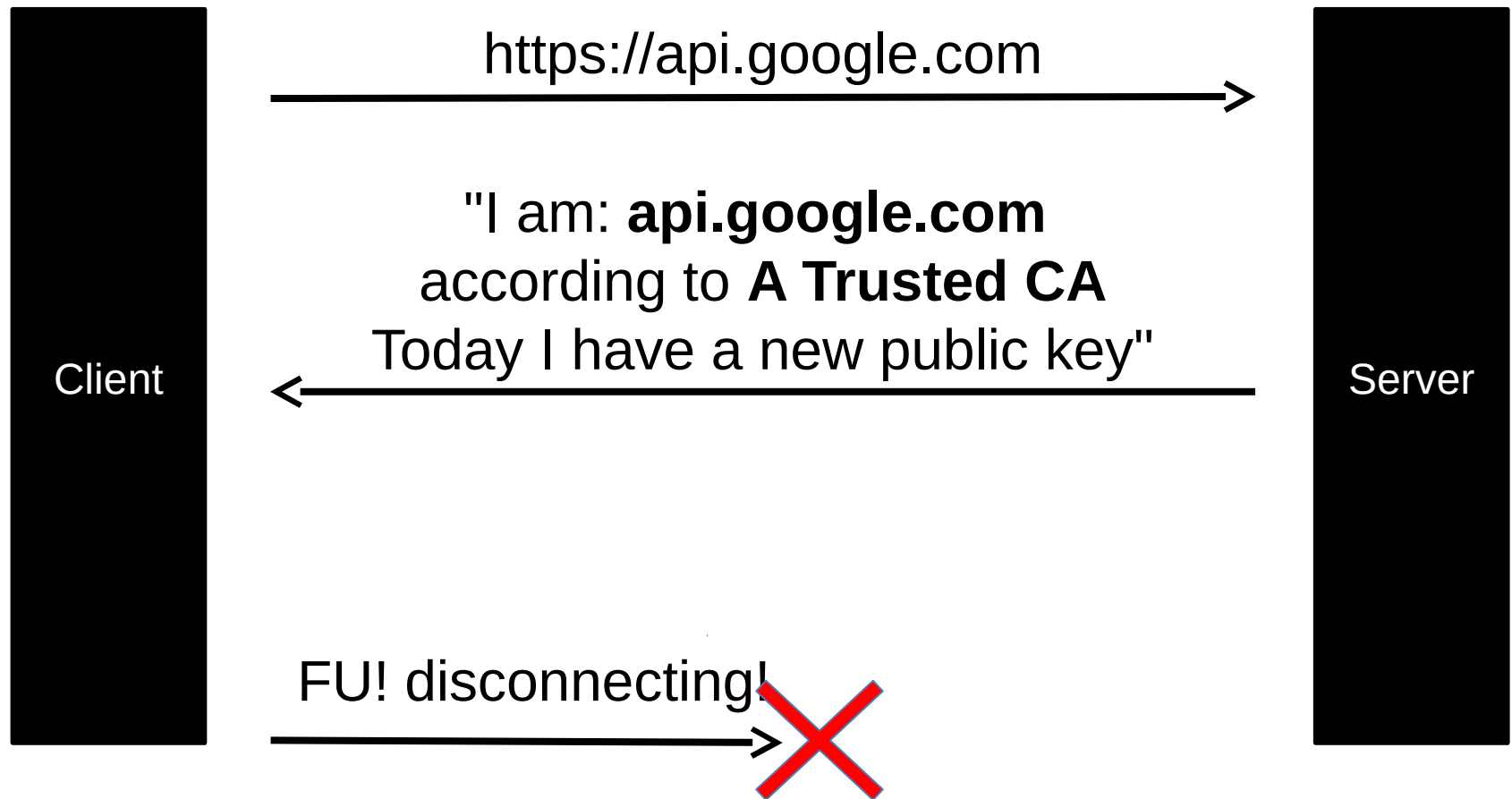
- Test environments
  - Don't create code to "handle SSL errors for dev/test"
  - Start dev environments with a Test trust store which trust the test environments
  - Run with secure standard code in dev/test
- Special cases
  - If you absolutely have to mess around, mess with specific connections instead of changing



# Certificate Pinning

[https://www.owasp.org/index.php/  
Certificate\\_and\\_Public\\_Key\\_Pinni  
ng#Android](https://www.owasp.org/index.php/Certificate_and_Public_Key_Pinning#Android)

# Certificate Pinning



# Certificate Pinning

- Force your system to only trust a specific certificate
  - If you do not trust certificate authorities
  - CA breaches
  - CA malpractice
  - Compelled certificates by a malicious government
- But only add security, don't remove checks
  - [https://www.owasp.org/index.php/Certificate\\_and\\_Public\\_Key\\_Pinning](https://www.owasp.org/index.php/Certificate_and_Public_Key_Pinning)#Android



# Certificate Pinning (1/2)

```
private static String PUB_KEY =
"30820122300d06092a864886f70d0101 ... ..";

public void checkServerTrusted(X509Certificate[] chain, String authType)
throws CertificateException {
...
    // Perform customary SSL/TLS checks
    try {
        TrustManagerFactory tmf = TrustManagerFactory.getInstance("X509");
        tmf.init((KeyStore) null);
        for (TrustManager trustManager : tmf.getTrustManagers()) {
            ((X509TrustManager) trustManager).checkServerTrusted(chain,
authType);
        }
    }
...
}
```



# Certificate Pinning (2/2)

// Hack ahead: BigInteger and toString(). We know a DER encoded Public Key begins

// with 0x30 (ASN.1 SEQUENCE and CONSTRUCTED), so there is no leading 0x00 to drop.

```
RSAPublicKey pubkey = (RSAPublicKey)
chain[0].getPublicKey();
```

```
String encoded = new BigInteger(1 /*
positive */,
pubkey.getEncoded()).toString(16);
```



// Pin it!  
OWASP

Open Web Application  
Security Project

final boolean expected =

Peter Magnusson, Twitter: @Blaufish\_

Sakerhetspodcasten.se & Omegapoint.se

# Force crazy TLS Security?

# Force crazy TLS Security?

SecureSocketFactory extends **SSLSocketFactory**

```
{  
public Socket createSocket(Socket s, ..... ) throws  
IOException {
```

```
    s.setEnabledProtocols(new String[] { "TLSv1.2" });
```

```
    s.setEnabledCipherSuites(new String[]  
{ "TLS_DHE_RSA_WITH_AES_256_GCM_SHA384",  
"TLS_ECDH_RSA_WITH_AES_256_GCM_SHA384" });
```

AES GCM and other AEAD cipher  
suites not officially supported in  
java7

# SSL Best Practice

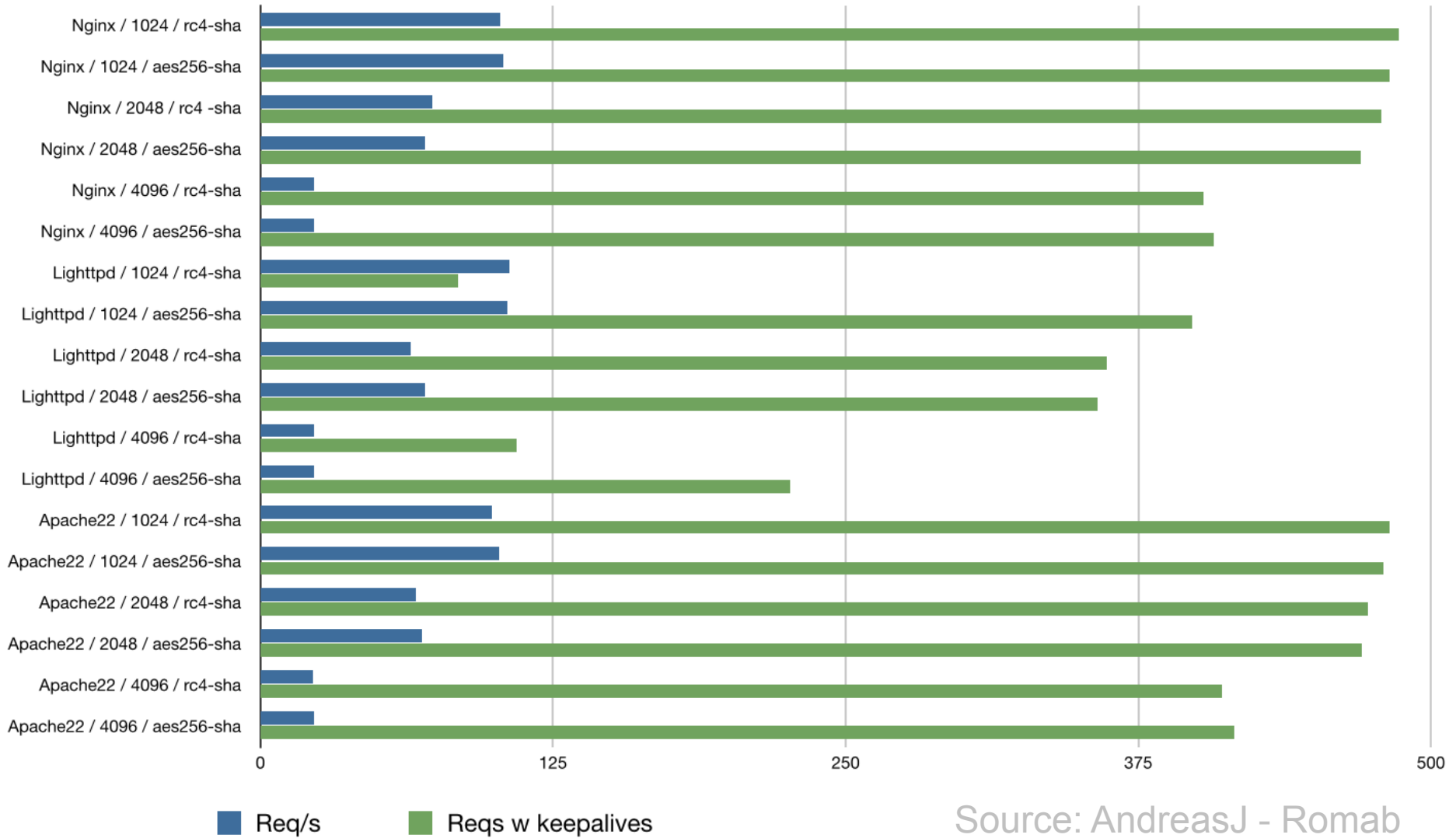


# USE HTTPS!

# Cost of longer keys

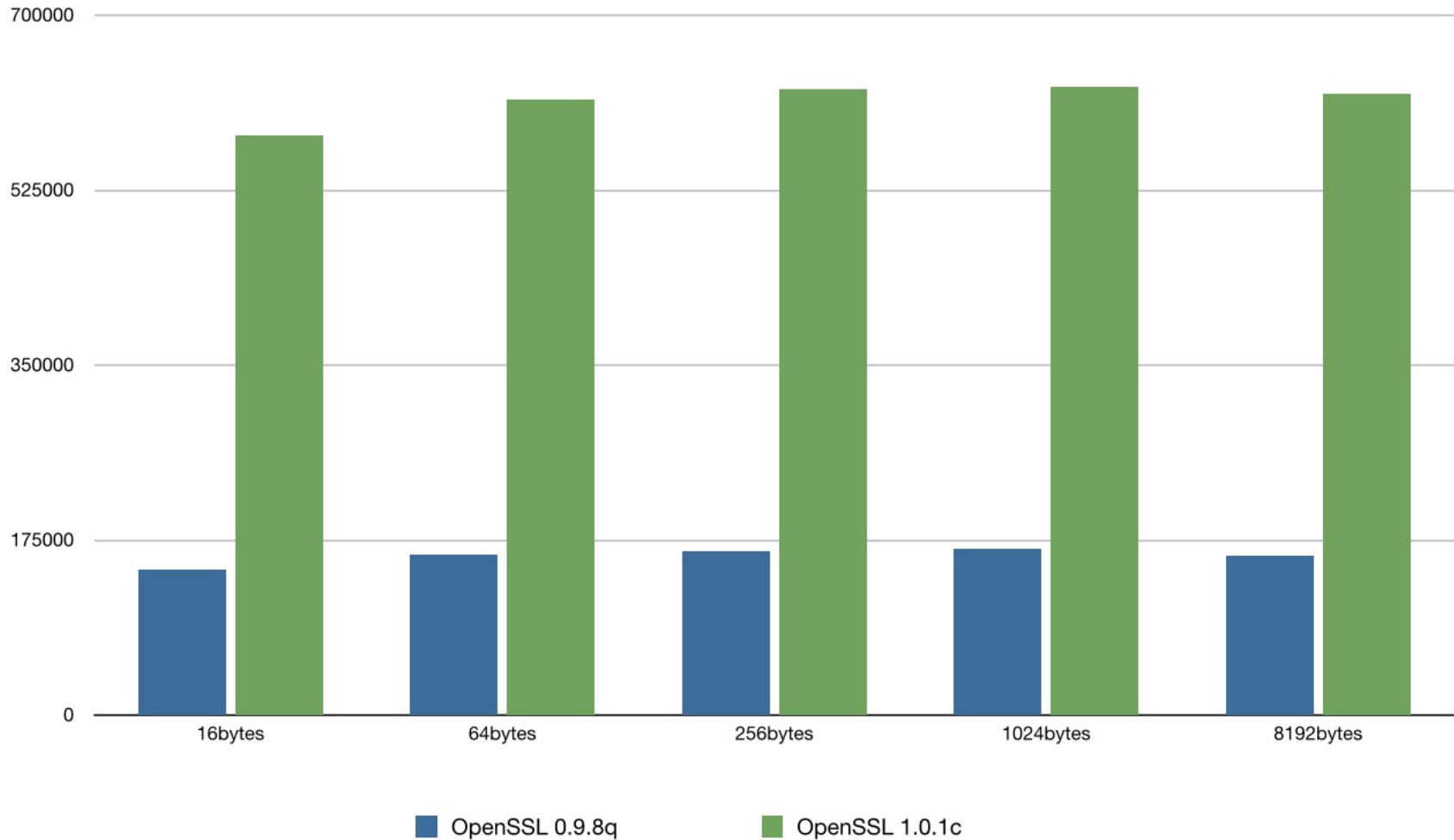
- Rapidly decreased performance
  - But Keepalives is more important
  - Upgrade of openssl can improve performance
- Interoperability
  - OSX, iOS supports max 4096 bits by default
  - Chrome, Android max ~2300 bits until recently

# server/rsa key/cipher



# OpenSSL

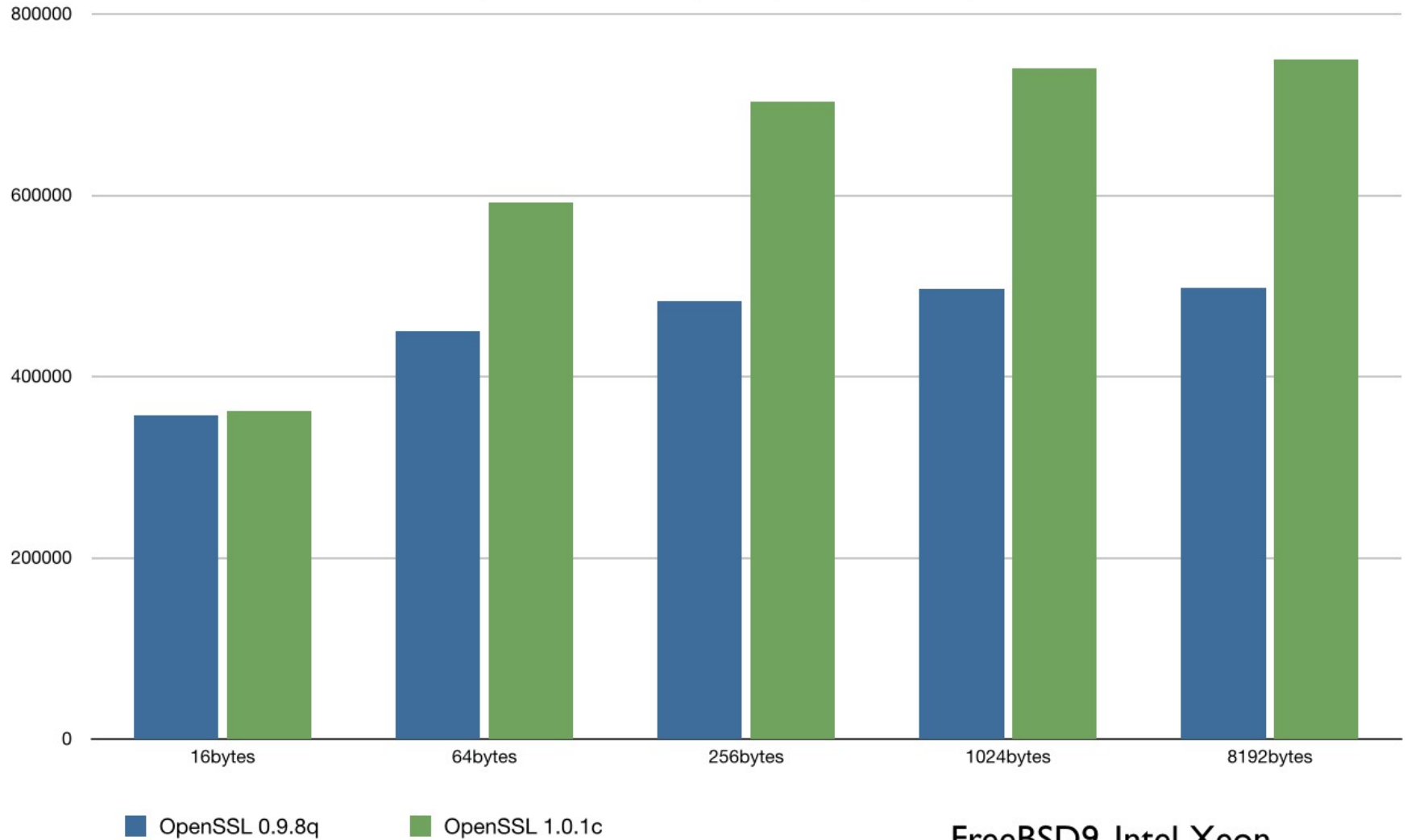
OpenSSL Performance, 1000bytes/sec processed, AES128-CBC



FreeBSD9, Intel Xeon  
CPU E31230 @ 3.20GHz

# OpenSSL

OpenSSL Performance, 1000bytes/sec processed, RC4



# CA, Trust Store requirements

## Microsoft

### Microsoft Security Advisory (2880823)

#### Deprecation of SHA-1 Hashing Algorithm for Microsoft Root Certificate Program

Published: Tuesday, November 12, 2013

Version: 1.0

#### General Information

##### Executive Summary

Microsoft is announcing a policy change to the Microsoft Root Certificate Program. The new policy will no longer allow root certificate authorities to issue X.509 certificates using the SHA-1 hashing algorithm for the purposes of SSL and code signing after January 1, 2016. Using the SHA-1 hashing algorithm in digital certificates could allow an attacker to spoof content, perform phishing attacks, or perform man-in-the-middle attacks.

**Recommendation:** Microsoft recommends that certificate authorities no longer sign newly generated certificates using the SHA-1 hashing algorithm and begin migrating to SHA-2. Microsoft also recommends that customers replace their SHA-1 certificates with SHA-2 certificates at the earliest opportunity. Please see the **Suggested Actions** section of this advisory for more information.

No root certs issued with SHA-1 after 2016

Use certs with SHA-2 (there are problems though)

# CA, Trust Store requirements

## Mozilla

- **June 30, 2011** – Mozilla will stop accepting MD5 as a hash algorithm for intermediate and end-entity certificates. After this date software published by Mozilla will return an error when a certificate with an MD5-based signature is used.
  - [bug 650355](#) - "Stop accepting MD5 as a hash algorithm in signatures (toggle security.enable\_md5\_signatures to false)" -- Fixed in Mozilla 16 (Firefox 16).
  - [bug 590364](#) - "By default, stop accepting MD5 as a hash algorithm in certificate signatures" - Until this bug is fixed, non-Gecko software that uses NSS will still accept MD5 signatures. (Gecko is the layout engine developed by the Mozilla Project, originally called NGLayout.) -- In NSS 3.14
- **December 31, 2013** – Soon after this date, Mozilla will disable the SSL and Code Signing trust bits for root certificates with RSA key sizes smaller than 2048 bits. If those root certificates are no longer needed for S/MIME, then Mozilla will remove them from NSS.
  - TEST: You can test the behavior of changing the root certificate trust bit settings, as described here: [https://wiki.mozilla.org/CA:UserCertDB#Changing\\_Root\\_Certificate\\_Trust\\_Bit\\_Settings](https://wiki.mozilla.org/CA:UserCertDB#Changing_Root_Certificate_Trust_Bit_Settings)

No root certs with 1024 bit keys in January 2014

Check your root

# Deploying Forward Secrecy

- Upgrade server and libs
  - Apache 2.4
  - Nginx 1.0.6, 1.1.0
  - OpenSSL 1.0.1c
  - GnuTLS 3.2.7
- Performance cost
  - Est 15% CPU and comm during init



# TEST HTTPS!

You are here: [Home](#) > [Projects](#) > [SSL Server Test](#) > romab.com

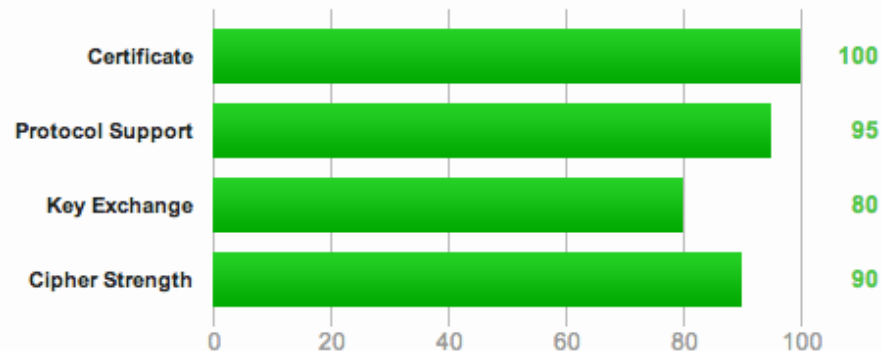
## SSL Report: romab.com (192.195.142.8)

Assessed on: Thu Nov 21 13:11:35 UTC 2013 | [Clear cache](#)

[Scan Another »](#)

### Summary

#### Overall Rating



Documentation: [SSL/TLS Deployment Best Practices](#), [SSL Server Rating Guide](#), and [OpenSSL Cookbook](#).

This server provides robust [Forward Secrecy](#) support.

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

You are here: [Home](#) > [Projects](#) > [SSL Server Test](#) > molndal.se

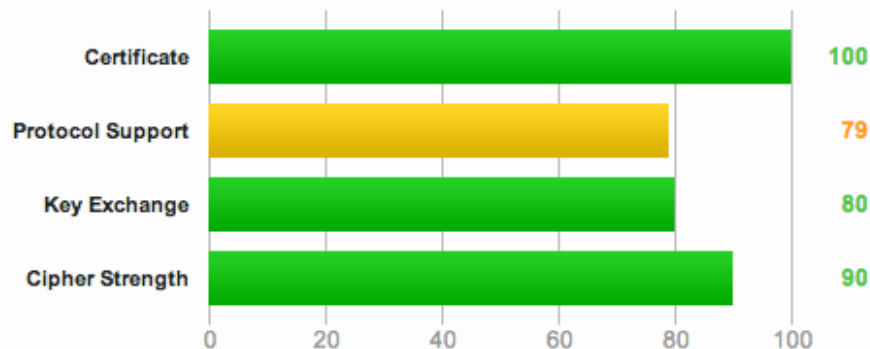
## SSL Report: molndal.se (31.193.200.10)

Assessed on: Thu Nov 21 14:22:19 UTC 2013 | [Clear cache](#)

[Scan Another »](#)

### Summary

#### Overall Rating



Documentation: [SSL/TLS Deployment Best Practices](#), [SSL Server Rating Guide](#), and [OpenSSL Cookbook](#).

This server does not mitigate the [CRIME attack](#). Grade capped to B.

This site supports only older protocol versions, but not the most recent and more secure TLS 1.2.

You are here: [Home](#) > [Projects](#) > [SSL Server Test](#) > abanan.se

## SSL Report: abanan.se (62.101.37.2)

Assessed on: Thu Nov 21 14:19:16 UTC 2013 | [Clear cache](#)

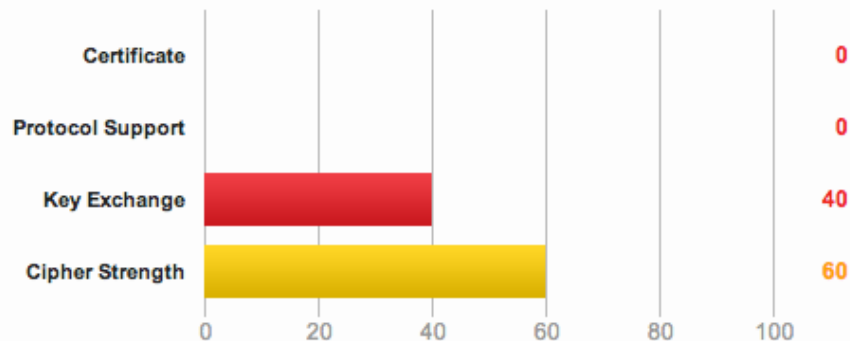
[Scan Another »](#)

### Summary

#### Overall Rating



If trust issues are ignored: F



Documentation: [SSL/TLS Deployment Best Practices](#), [SSL Server Rating Guide](#), and [OpenSSL Cookbook](#).

This server supports SSL 2, which is obsolete and insecure. Grade set to F.

This site supports only older protocol versions, but not the most recent and more secure TLS 1.2.

[Home](#)[Qualys.com](#)[Projects](#)[Contact](#)

You are here: [Home](#) > [Projects](#) > [SSL Server Test](#) > omegapoint.se

## SSL Report: omegapoint.se (194.103.93.60)

Assessed on: Thu Nov 21 14:04:12 UTC 2013 | [Clear cache](#)

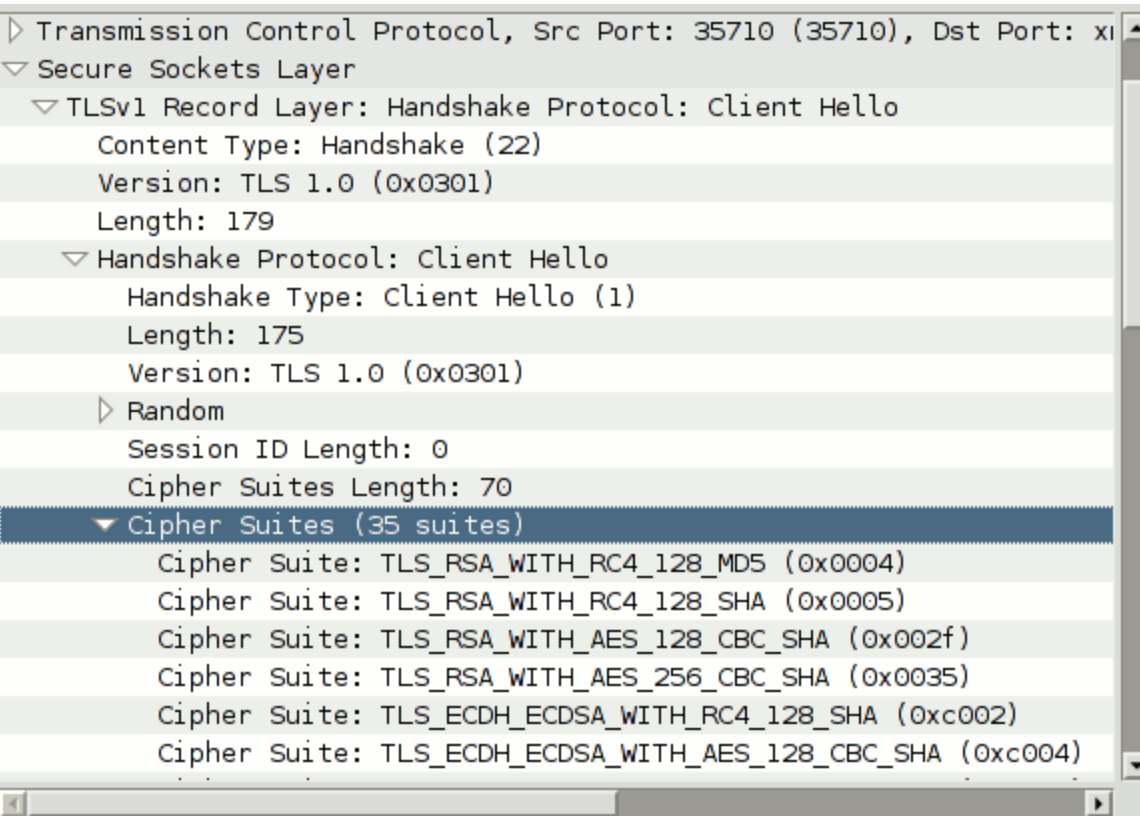
[Scan Another »](#)

Assessment failed: No secure protocols supported

# Check & change in-app suites

Android uses the  
cipher suite order  
in Java

AES-256 and SHA1 to  
RC4 and MD5 in 2010



Android 2.2.1	Android 2.3.4, 2.3.7	Android 4.2.2, 4.3
DHE-RSA-AES256-SHA	RC4-MD5	RC4-MD5
DHE-DSS-AES256-SHA	RC4-SHA	RC4-SHA
AES256-SHA	AES128-SHA	AES128-SHA
EDH-RSA-DES-CBC3-SHA	DHE-RSA-AES128-SHA	AES256-SHA

Change the order  
in your app

# Client validation

## SSL/TLS Capabilities of Your Browser (Experimental)

User Agent: Mozilla/5.0 (Macintosh; Intel Mac OS X 10\_9\_0) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/31.0.1650.26 Safari/537.36

### Details



#### Protocols\*

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

(\*) This test reliably detects only the highest supported protocol.



#### Cipher Suites (in order of preference)

TLS_ECDHE_ECDSA_WITH_AES_128_GCM_SHA256 (0xc02b) <i>Forward Secrecy</i>	128
TLS_ECDHE_RSA_WITH_AES_128_GCM_SHA256 (0xc02f) <i>Forward Secrecy</i>	128
TLS_DHE_RSA_WITH_AES_128_GCM_SHA256 (0x9e) <i>Forward Secrecy</i>	128
TLS_RSA_WITH_AES_128_GCM_SHA256 (0x9c)	128
TLS_ECDHE_ECDSA_WITH_AES_256_CBC_SHA (0xc00a) <i>Forward Secrecy</i>	256
TLS_ECDHE_RSA_WITH_AES_256_CBC_SHA (0xc014) <i>Forward Secrecy</i>	256

<https://www.ssllabs.com/ssltest/viewMyClient.html>

# Client Stress Testing

- TLSpretense
  - Check that your client **does not fail open**
  - Accepts certs for wrong domain
  - Accepts broken chains
  - Null byte host name

<https://github.com/iSECPartners/tlspretense/>



# Sslyze

- Open source SSL/TLS testing tool
  - Test public AND private servers
- IIS sponsored development of new functions
  - OCSP, CRL, Multiple Trust Stores, SNI etc
  - HSTS, HTTP vs HTTPS (content)
- Upstream merge of some parts
  - New generation with API breakage during project

# Sslyze

```
e>./sslyze.py --regular --crl --ocsp --sni=auto --hsts --starttls=auto www.kirei.se
```

## \* Certificate :

```
CRL verification:      Certificate not revoked in CRL
SNI:                   SNI enabled with virtual domain www.kirei.se
Trusted or NOT Trusted: Trusted
OCSP verification:     Certificate not revoked
Validated by Trust Store: apple-20130529
Validated by Trust Store: microsoft-20130905
Validated by Trust Store: copied_ca
Validated by Trust Store: stripped_ca
Validated by Trust Store: java7v25-20130810
Validated by Trust Store: mozilla-20130812
X509 Policy in certificate: True
Hostname Validation:   OK - SNI CN www.kirei.se Matches
SHA1 Fingerprint:      FB35A7AAEC8A32FCC7E4016EAF5734459DCF5809

Common Name:           www.kirei.se
Issuer:                 /C=GB/ST=Greater Manchester/L=Salford/O=Comodo CA Limited/CN=PositiveSSL CA
Serial Number:         BC95078646835C5C090AAB839F1DDBFE
Not Before:            Aug 25 00:00:00 2011 GMT
Not After:             Aug 24 23:59:59 2014 GMT
Signature Algorithm:   sha1WithRSAEncryption
Key Size:              2048
X509v3 Subject Alternative Name: DNS:www.kirei.se, DNS:kirei.se
```

## \* HSTS :

Supported:

max-age=864000

## \* SSLV3 Cipher Suites :

```
./sslyze.py --regular --sni=auto --starttls=auto --crl --ocsp --hsts ab.se
```

\* Certificate :

CRL verification:	No CRL URI in certificate
SNI:	SNI enabled with virtual domain ab.se
Trusted or NOT Trusted:	NOT Trusted
OCSP verification:	No OCSP Responder in certificate
Not validated by Trust Store:	apple-20130529 - self signed certificate
Not validated by Trust Store:	microsoft-20130905 - self signed certificate
Not validated by Trust Store:	copied_ca - self signed certificate
Not validated by Trust Store:	stripped_ca - self signed certificate
Not validated by Trust Store:	java7v25-20130810 - self signed certificate
Not validated by Trust Store:	mozilla-20130812 - self signed certificate
X509 Policy in certificate:	False
Hostname Validation:	MISMATCH
SHA1 Fingerprint:	28953D02EE7763D865242C2BE08E2F8DDDB610EE
Common Name:	intranet.ab.se
Issuer:	/CN=intranet.ab.se
Serial Number:	B365F5B94A020F50
Not Before:	Jul 23 17:30:56 2010 GMT
Not After:	Jul 20 17:30:56 2020 GMT
Signature Algorithm:	sha1WithRSAEncryption
Key Size:	1024

# SSL Status

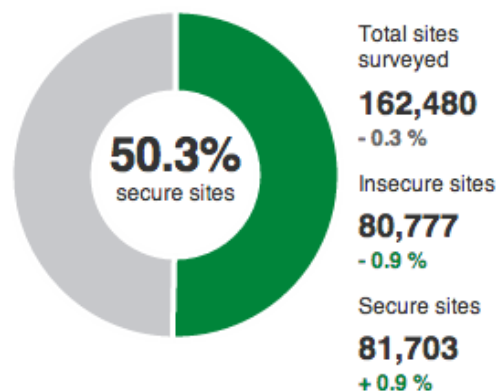
November 02, 2013

### Summary

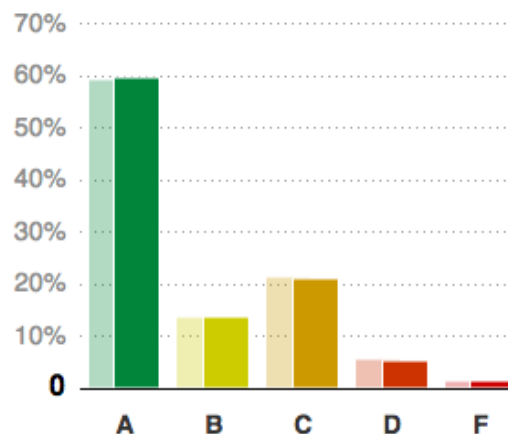
Published Date: **November 02, 2013**  
Comparisons are made against the previous month's data.

[< Previous](#)[Next >](#)

#### SSL Security Summary



#### SSL Labs Grade Distribution

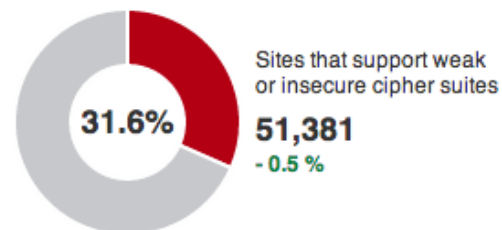


### Key Findings

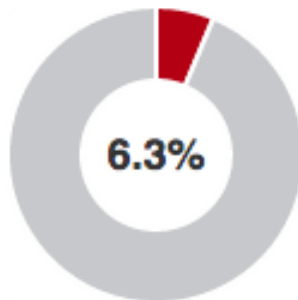
#### Certificate Chain



#### Cipher Strength



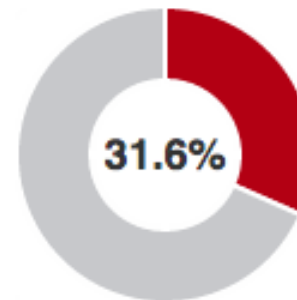
## Certificate Chain



Sites with incomplete  
certificate chain

**10,249**  
- 0.1 %

## Cipher Strength



Sites that support weak  
or insecure cipher suites

**51,381**  
- 0.5 %

## Key Strength



**0**

Sites with keys  
below 1024 bits

+ 0 since previous month

## Strict Transport Security

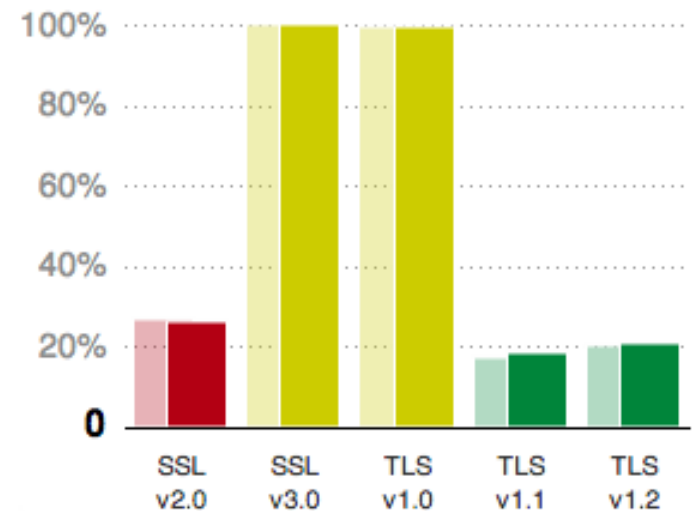


**977**

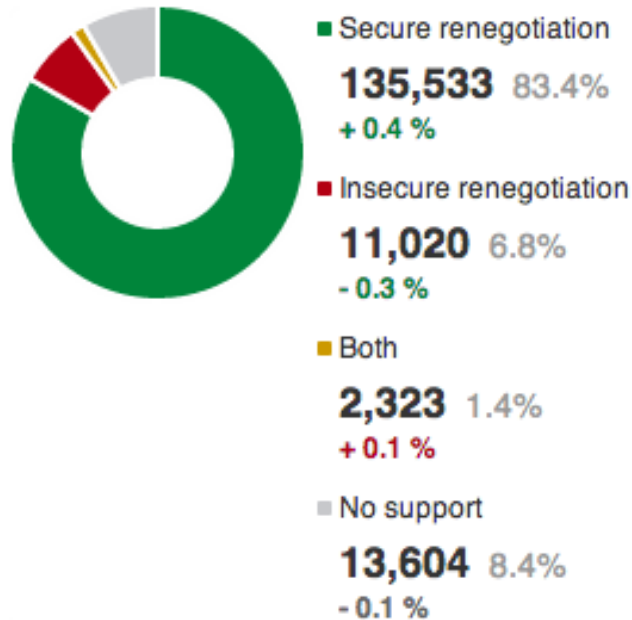
Sites that support HTTP  
Strict Transport Security

+ 52 since previous month

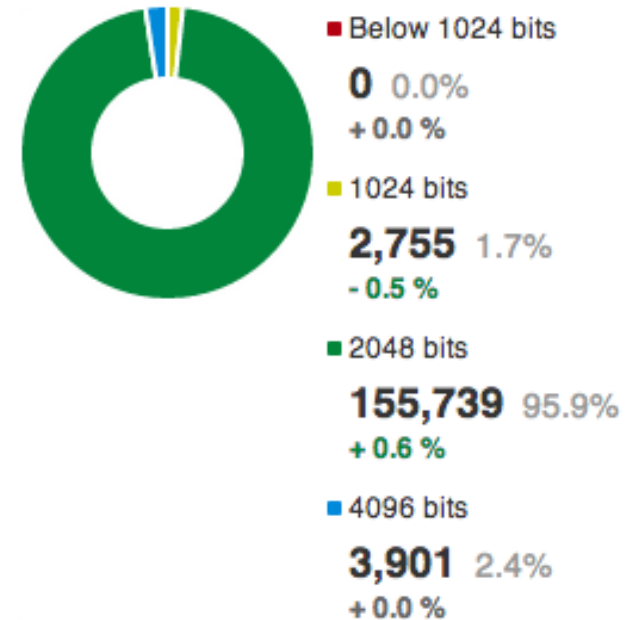
## Protocol Support



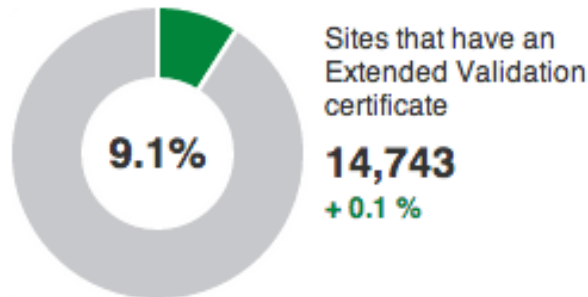
## Renegotiation Support



## Key Strength Distribution



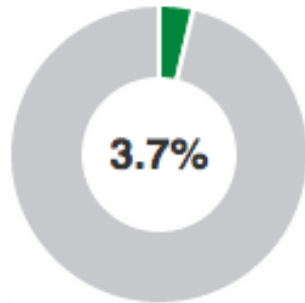
## Extended Validation Certificates



## BEAST Attack

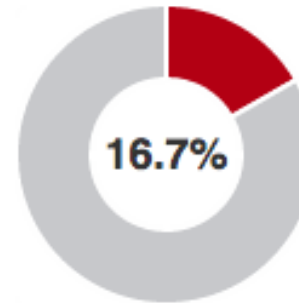


## SPDY



Sites that support  
the SPDY protocol  
**6,023**  
+ 0.7 %

## TLS Compression / CRIME



Sites that support  
TLS compression  
**27,137**  
- 0.8 %

## Forward Secrecy



■ Not supported  
**87,565** 53.9%  
- 0.1 %

■ Some FS suites enabled  
**67,847** 41.8%  
+ 0.0 %

■ Used with modern browsers  
**6,016** 3.7%  
+ 0.1 %

■ Used with most browsers  
**1,052** 0.6%  
+ 0.0 %

## RC4



■ Not Supported  
**11,857** 7.3%  
+ 0.1 %

■ Some RC4 suites enabled  
**91,358** 56.2%  
- 0.1 %

■ Used with modern browsers  
**59,265** 36.5%  
+ 0.0 %



Open Web Application  
Security Project

Source: <https://www.worthyinternet.org/ssl-pulse/>

Sakernetspodcasten.se & Omegapoint.se



# Questions?

These slides will be available at:  
[www.slideshare.net/blaufish](http://www.slideshare.net/blaufish)  
[www.owasp.org/index.php/Göteborg](http://www.owasp.org/index.php/Göteborg)

## References

- OpenSSL Cookbook
  - <https://www.feistyduck.com/books/openssl-cookbook/>
- SSL/TLS Deployment Best Practice
  - <https://www.ssllabs.com/projects/best-practices/index.html>
- Sslyze
  - IIS version: <https://github.com/kirei/sslyze>
  - Upstream: <https://github.com/iSECPartners/sslyze>
-