A Retrospective on the Use of Export Cryptography

or,

Top 10 Ways Bill Clinton Broke TLS!

David Adrian
@davidcadrian

UNIVERSITY OF MICHIGAN

David Adrian
@davidcadrian

UNIVERSITY OF MICHIGAN

zmap

censys


Microsoft®
Windows®95


PHILIPS
Prof.
PHILIPS

Meanwhile…

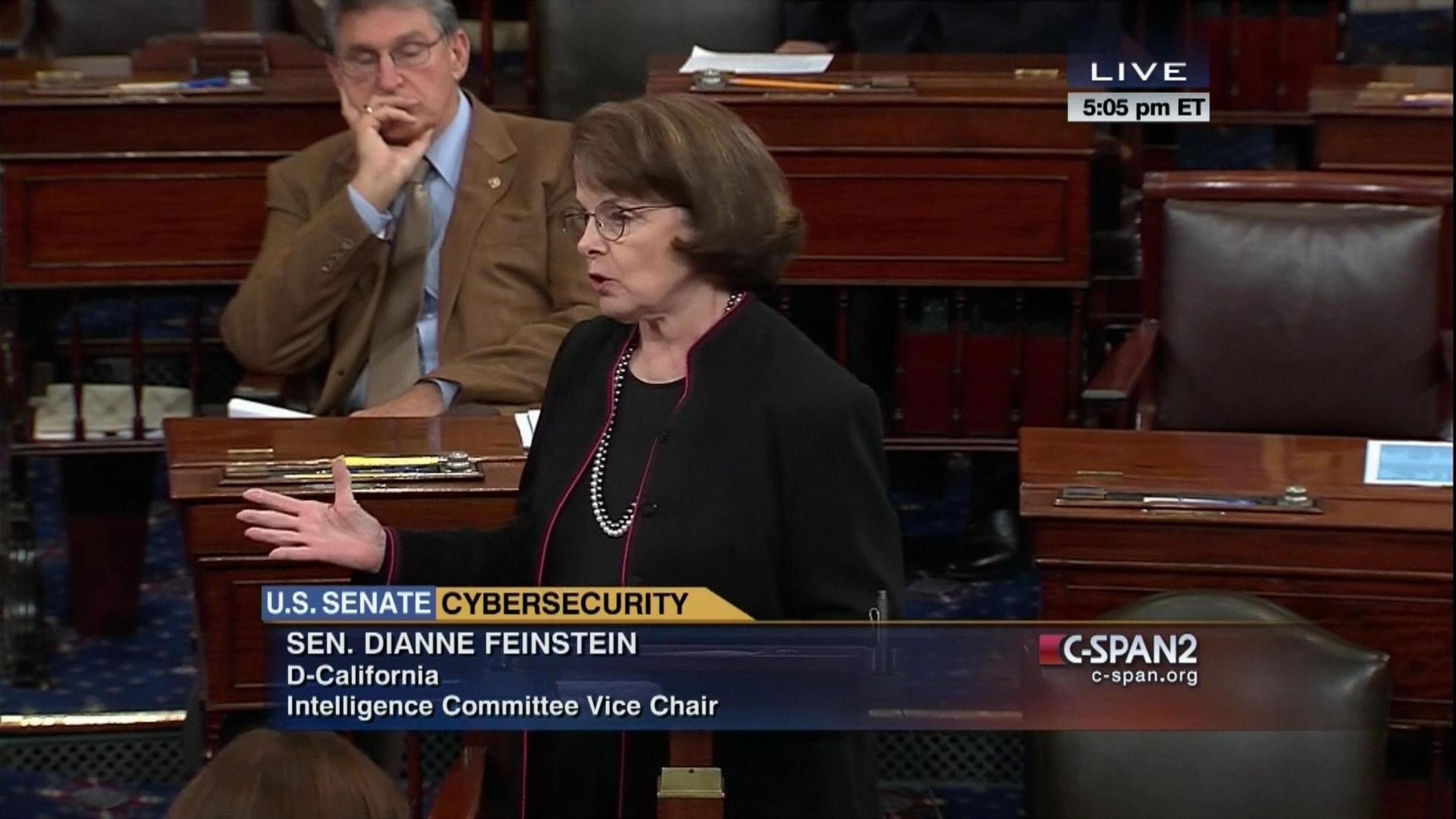
1995 — SSLv2 designed, deployed, and deprecated

1996 — SSLv3 replaces SSLv2, forms the basis for modern TLS

1999 — TLSv1.0 standardized by the IETF

Contains **export cryptography**

*Export regulations weakened protocol design to the point where they are **directly harmful** clients using modern cryptography*.


LIVE
5:05 pm ET
U.S. SENATE CYBERSECURITY
SEN. DIANNE FEINSTEIN
D-California
Intelligence Committee Vice Chair
C-SPAN2
c-span.org

Publications

A Messy State of the Union: Taming the Composite State Machines of TLS

Benjamin Beurdouche, Karthikeyan Bhargavan Antoine Delignat-Lavaud, Cedric Fournet, Markulf Kohlweiss, Alfredo Pironti, Pierre-Yves Strub, Jean-Karim Zinzindohoue

Oakland 2015

FREAK

Imperfect Forward Secrecy

***David Adrian**, Karthikeyan Bhargavan, Zakir Durumeric, Pierrick Gaudry, Matthew Green, J. Alex Halderman, Nadia Heninger, Drew Springall, and Emmanuel Thomé, Luke Valenta, Benjamin VanderSloot, Eric Wustrow, Santiago Zanella-Béguelin and Paul Zimmermann*

CCS 2015

Logjam

DROWN: Breaking TLS with SSLv2

*Nimrod Aviram, Sebastian Schinzel, Juraj Somorovsky, Nadia Heninger, Maik Dankel, Jens Steube, Luke Valenta, **David Adrian**, J. Alex Halderman, Viktor Dukhovni, Emilia Käsper, Shaanan Cohney, Susanne Engels, Christof Paar, and Yuval Shavitt*

USENIX 2016

DROWN

FREAK

Q: How do you selectively weaken a protocol based on RSA?

A: Use a shorter RSA key!

Q: How do you select which RSA key to use?

A: Convoluted protocol handshake!

Client Hello: client random, ciphers (...`RSA`...)

Server Hello: server random, chosen cipher

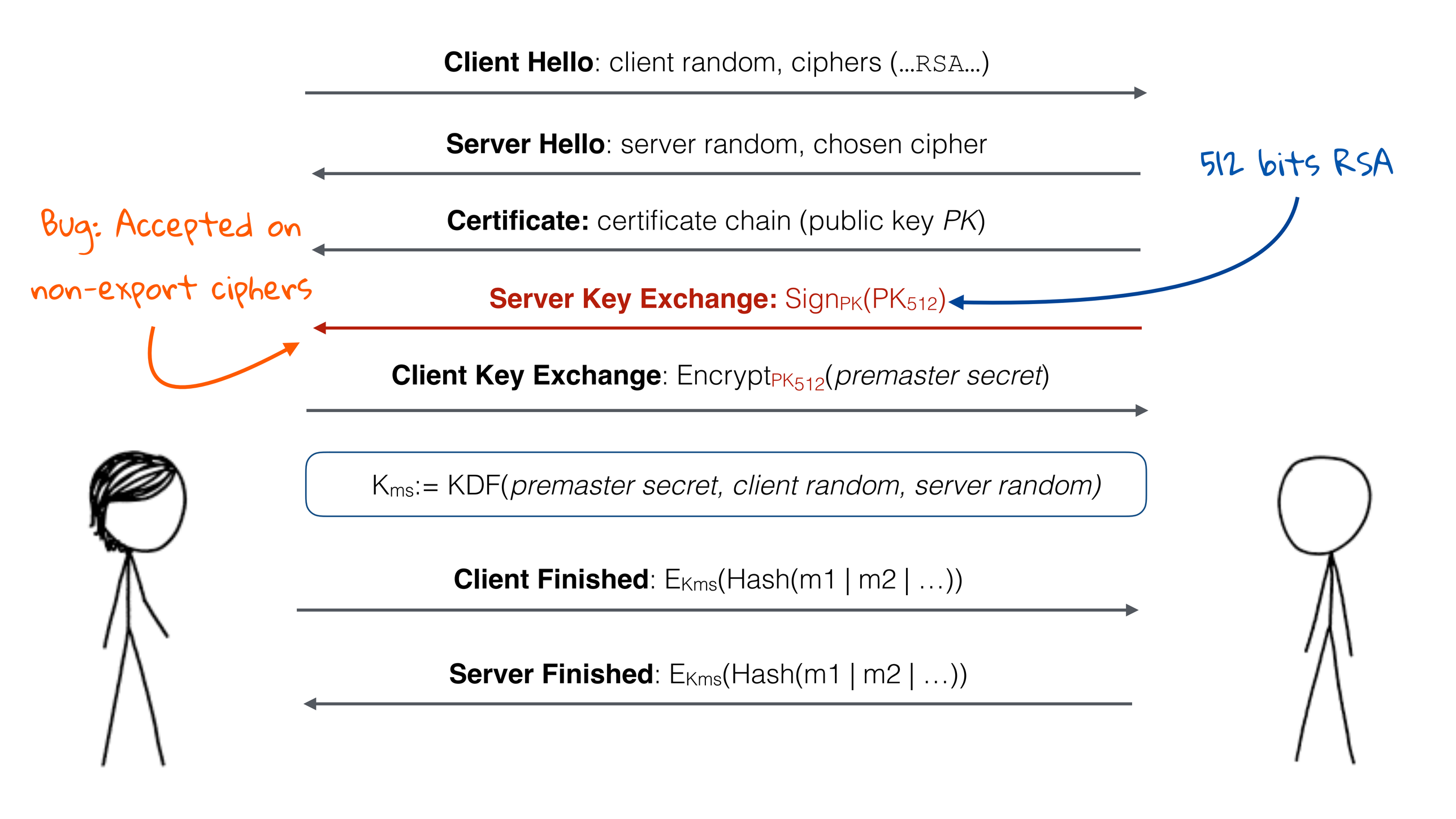
Certificate: certificate chain (public key *PK*)

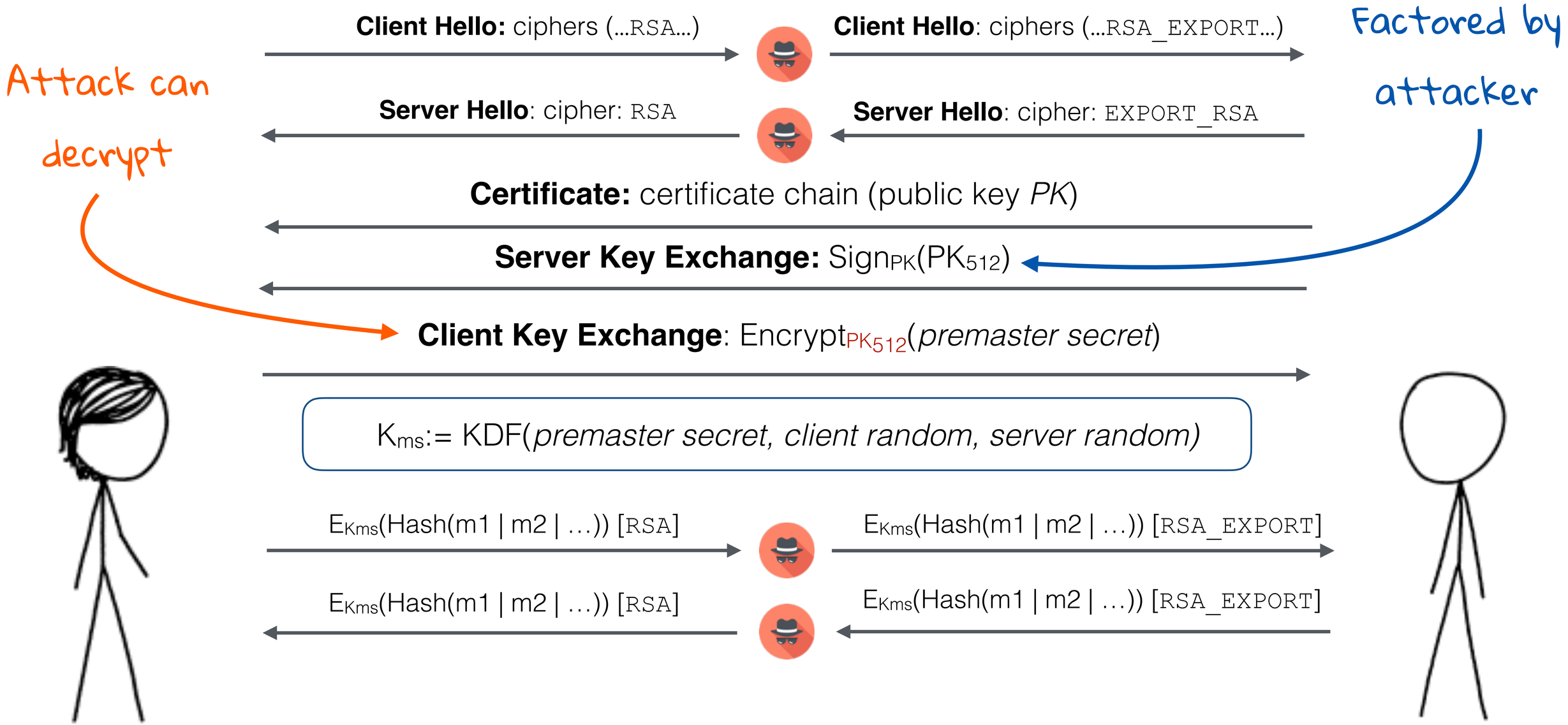
Client Key Exchange: Encrypt_{PK}(*premaster secret*)

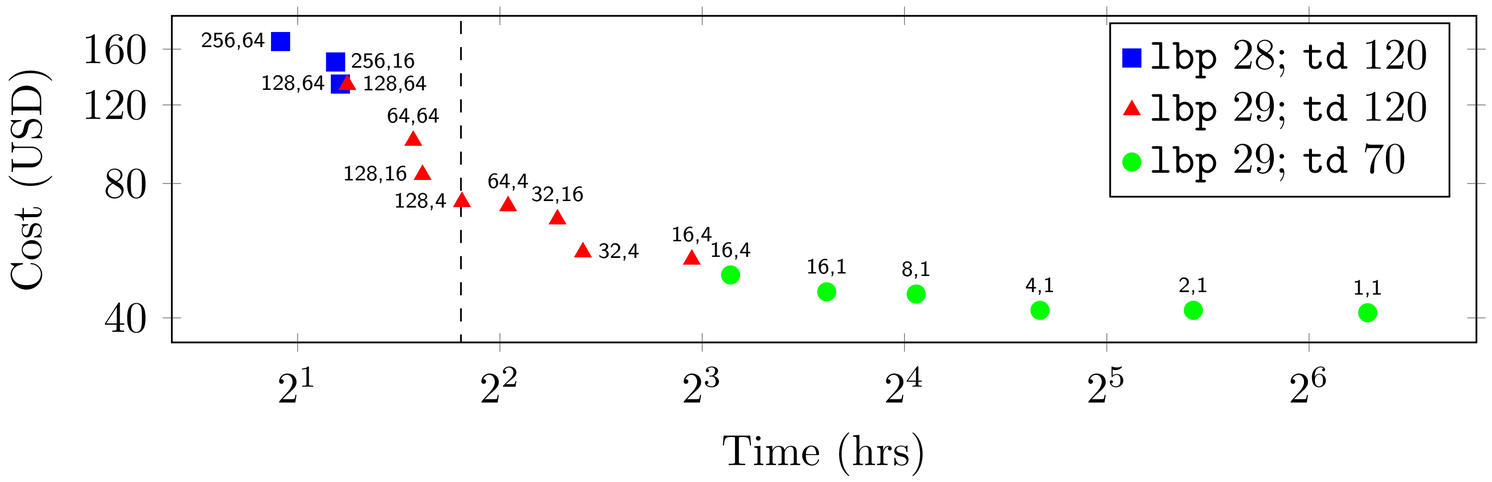
K_{ms}:= KDF(*premaster secret, client random, server random)*

Client Finished: E_{Kms}(Hash(m1 | m2 | …))

Server Finished: E_{Kms}(Hash(m1 | m2 | …))


Client Hello: client random, ciphers (...RSA...)
Server Hello: server random, chosen cipher
512 bits RSA
Bug: Accepted on non-export ciphers
Certificate: certificate chain (public key *PK*)
Server Key Exchange: $Sign_{PK}(PK_{512})$
Client Key Exchange: $Encrypt_{PK_{512}}$(*premaster secret*)
K_{ms}:= KDF(*premaster secret, client random, server random)*
Client Finished: E_{Kms}(Hash(m1 | m2 | …))
Server Finished: E_{Kms}(Hash(m1 | m2 | …))


Attack can decrypt
Factored by attacker
Client Hello: ciphers (...RSA...)
Client Hello: ciphers (...RSA_EXPORT...)
Server Hello: cipher: RSA
Server Hello: cipher: EXPORT_RSA
Certificate: certificate chain (public key PK)
Server Key Exchange: Sign_PK(PK_512)
Client Key Exchange: Encrypt_PK512(premaster secret)
K_ms := KDF(premaster secret, client random, server random)
E_Kms(Hash(m1 | m2 | ...)) [RSA]
E_Kms(Hash(m1 | m2 | ...)) [RSA_EXPORT]
E_Kms(Hash(m1 | m2 | ...)) [RSA]
E_Kms(Hash(m1 | m2 | ...)) [RSA_EXPORT]



Factoring as a Service

Luke Valenta, Shaanan Cohney, Alex Liao, Joshua Fried, Satya Bodduluri, Nadia Heninger

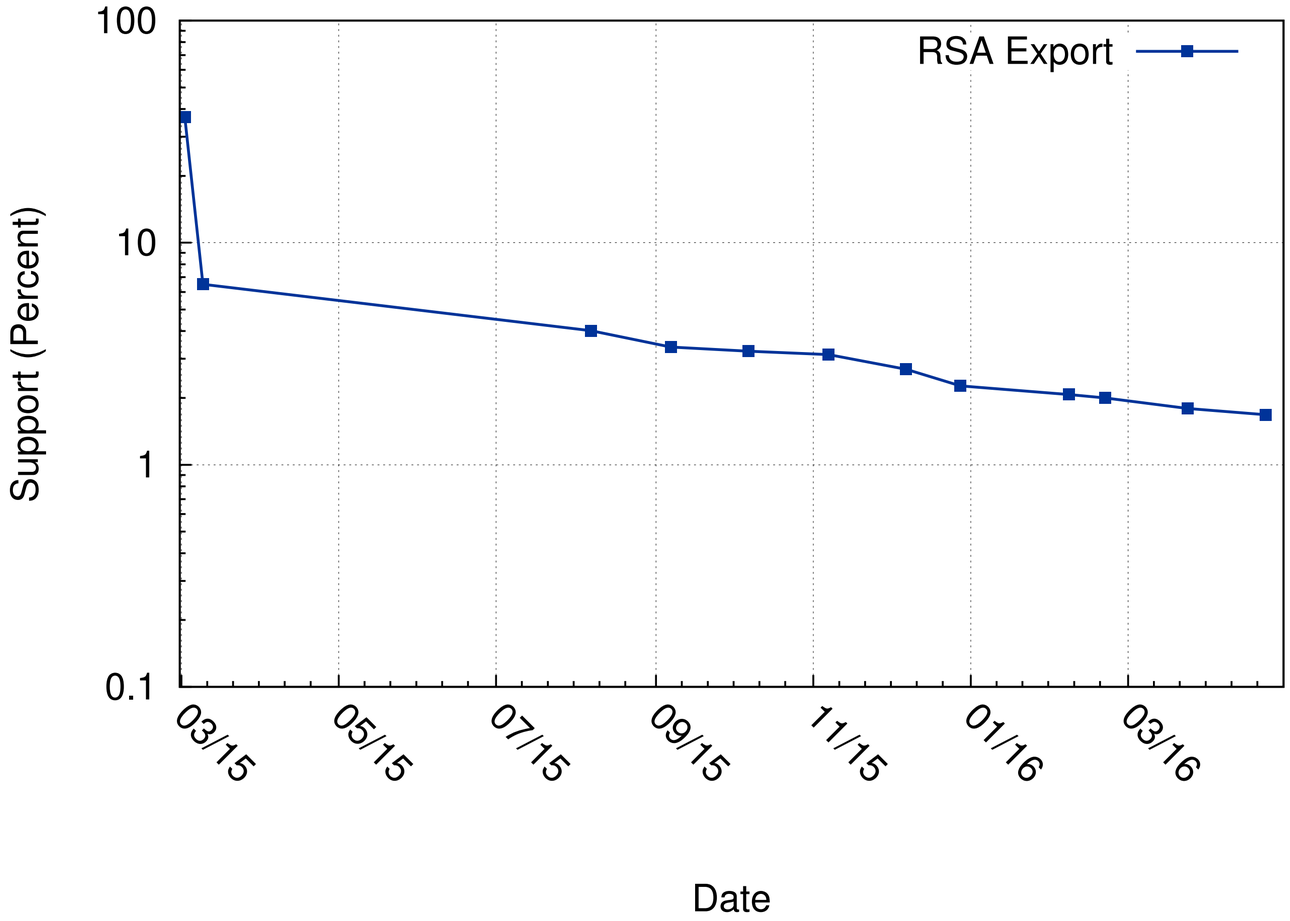
FC 2015

https://teespring.com/shop/hobby-tshirts/factoring

Anyone who can spend $100 to factor a server's RSA export key can impersonate that server!

RSA Export Support

Date	Support (Trusted HTTPS)
March 3, 2015	36.7%
March 10, 2015	6.5%
March 25, 2016	1.8%



Client Vulnerability

Gathered data about clients visiting freakattack.com

- *Implemented TLS server that sent RSA key exchange on non-export ciphers*
- *Attempted to load subdomain using Javascript*

1.2M page loads, 223K (18%) vulnerable

- *Data is biased and not complete (from 2 days post disclosure)*
- *Users are not deduplicated*
- *Data is from several days after disclosure, browsers were in the process of patching*

Vulnerable Firefox user agents

- *Of the 223K vulnerable clients, 15.6K (7.0%) identified as Firefox*
- *Mozilla NSS was not vulnerable, this is likely due to client-side MITM proxies*
- *Experimentally confirmed behavior with packet traces of Avast Anti-Virus*

Mitigations

Disable RSA export ciphers

- *Bugs in state machine are less impactful if bad crypto is disabled*

Update OpenSSL/SecureTransport/SChannel

- *All libraries were patched in 2015*

Details on https://freakattack.com

- *Instructions on how to patch various server software.*

FREAK Origins

SSLv3 drafted in the middle of DJB v. US

- *FREAK would not have existed if the regulations had been lifted*

FREAK is a protocol bug in SSLv3, implementation bug in TLS 1.0

- *Need to have clear specifications with well defined edges*
- *Standard should not be OpenSSL*

FREAK is caused by interaction between export and non-export

- *Individually, ciphers were implemented correctly*
- *Composing state machines is difficult*
- *Reasoning at both the protocol and implementation level is hard*

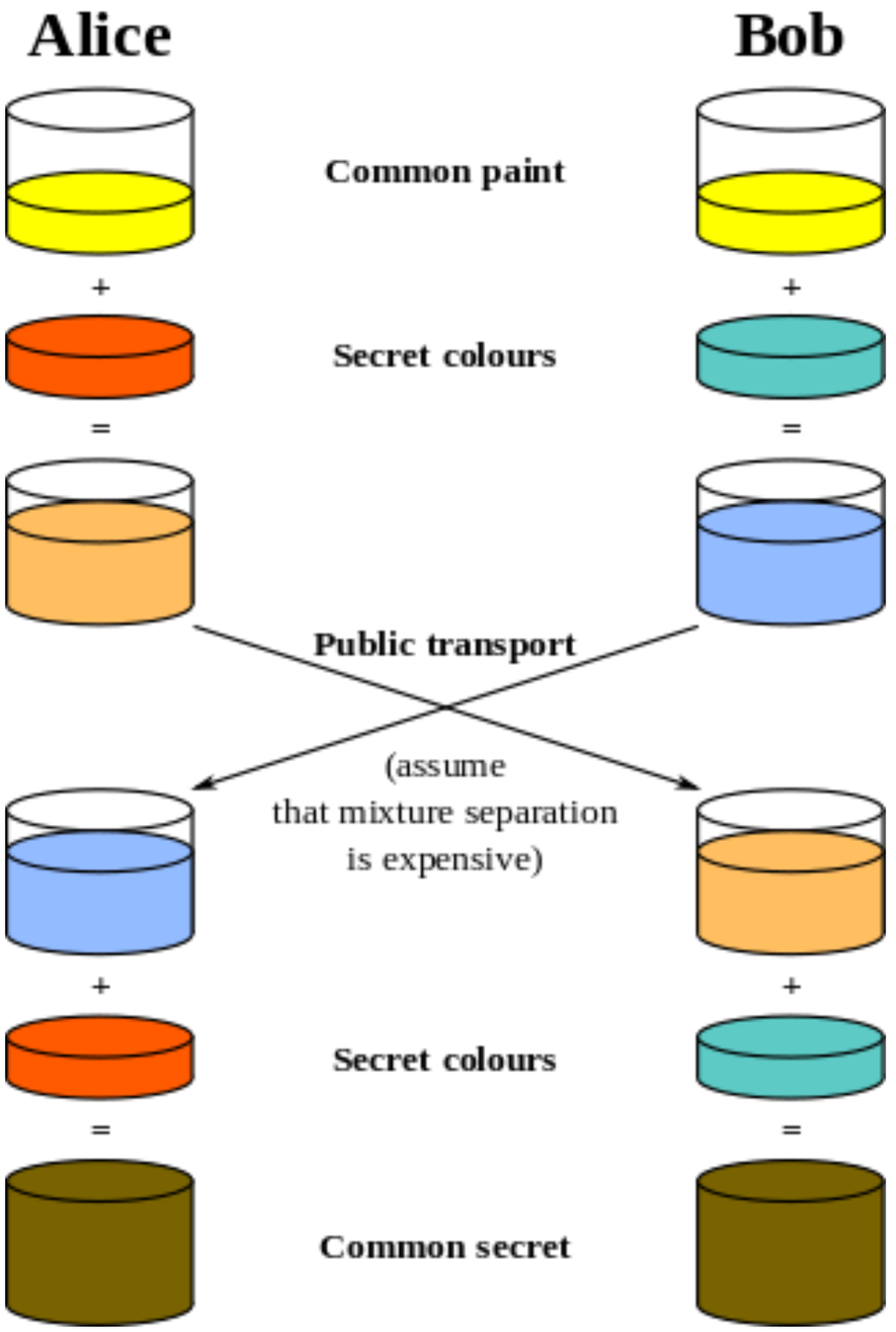
Logjam

Q: How do you selectively weaken a protocol based on Diffie-Hellman?

A: Use a shorter prime!

Q: How do you select which prime to use?

A: Convoluted protocol handshake!


Alice
Bob
Common paint
+
Secret colours
=
Public transport
(assume
that mixture separation
is expensive)
+
Secret colours
=
Common secret

Logjam

Downgrade attack against TLS

- *Identical attack flow to FREAK*
- *Server must support export Diffie-Hellman ciphers*

Protocol vulnerability, not implementation bug

- *Impossible to distinguish export Diffie-Hellman exchange from non-export*
- *Client can only partially mitigate*

Compute 512-bit discrete log instead of factoring 512-bit key

- *Most work is in 1 week precomputation per prime*
- *Calculate individual discrete logs in less than one minute*

Client Hello: client random, ciphers (...`DHE`...)

Server Hello: server random, chosen cipher

Certificate: certificate chain (public key)

Server Key Exchange: p, g, g^a, $\text{Sign}_{\text{CertKey}}(p, g, g^a)$

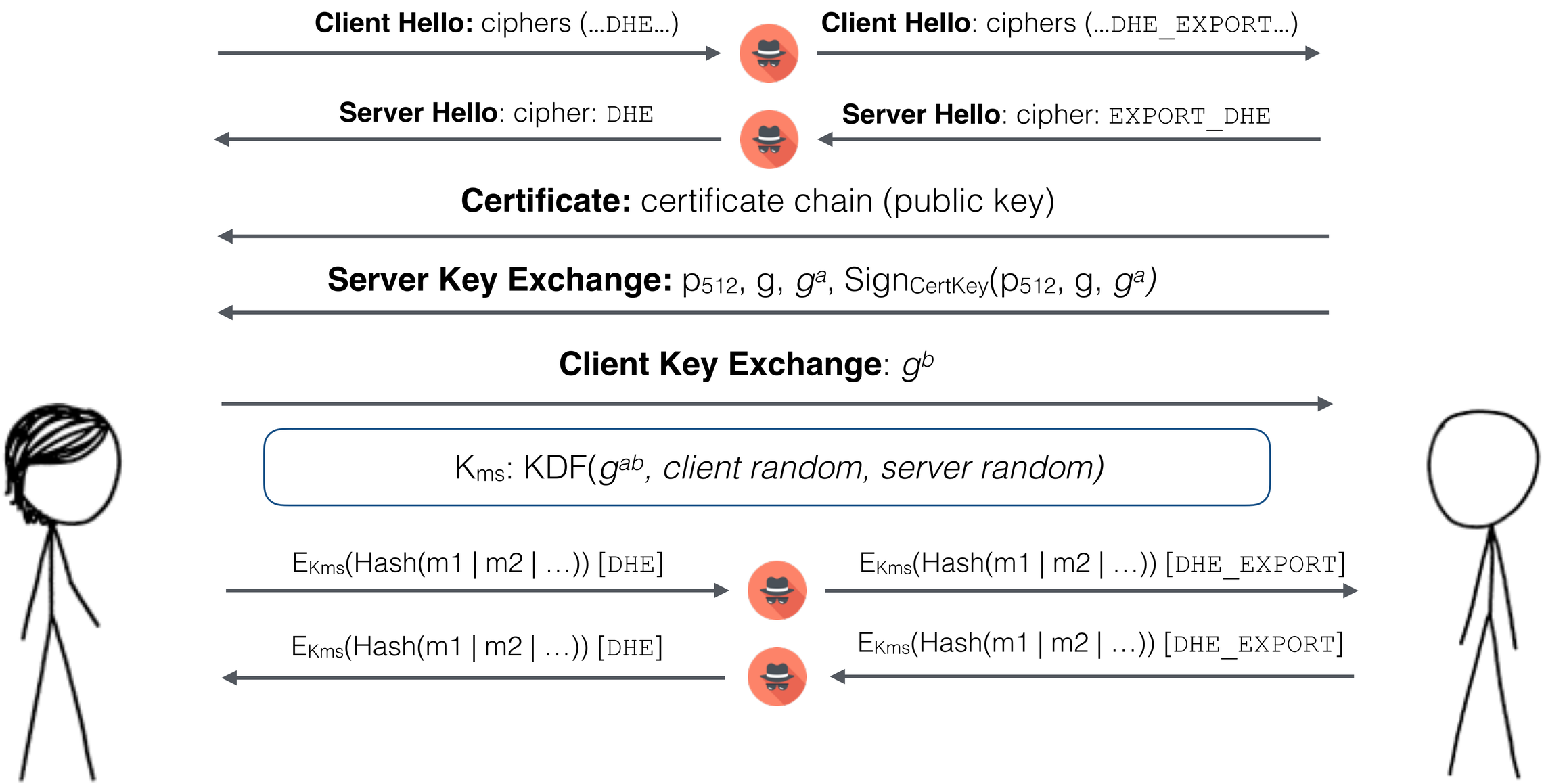
512 bit prime for export DHE ciphers

Client Key Exchange: g^b

K_{ms}: KDF(g^{ab}, *client random, server random)*

Client Finished: E_{Kms}(Hash(m1 | m2 | …))

Server Finished: E_{Kms}(Hash(m1 | m2 | …))


Client Hello: ciphers (…DHE…)
Client Hello: ciphers (…DHE_EXPORT…)
Server Hello: cipher: DHE
Server Hello: cipher: EXPORT_DHE
Certificate: certificate chain (public key)
Server Key Exchange: p_{512}, g, g^a, $Sign_{CertKey}(p_{512}, g, g^a)$
Client Key Exchange: g^b
K_{ms}: KDF(g^{ab}, *client random, server random)*
E_{Kms}(Hash(m1 | m2 | …)) [DHE]
E_{Kms}(Hash(m1 | m2 | …)) [DHE_EXPORT]
E_{Kms}(Hash(m1 | m2 | …)) [DHE]
E_{Kms}(Hash(m1 | m2 | …)) [DHE_EXPORT]

Feasibility

Do real-world servers support export Diffie-Hellman?

- *How many trusted HTTPS hosts support export DHE? Alexa Top 1M?*
- *Did people disable export DHE when disabling export RSA?*

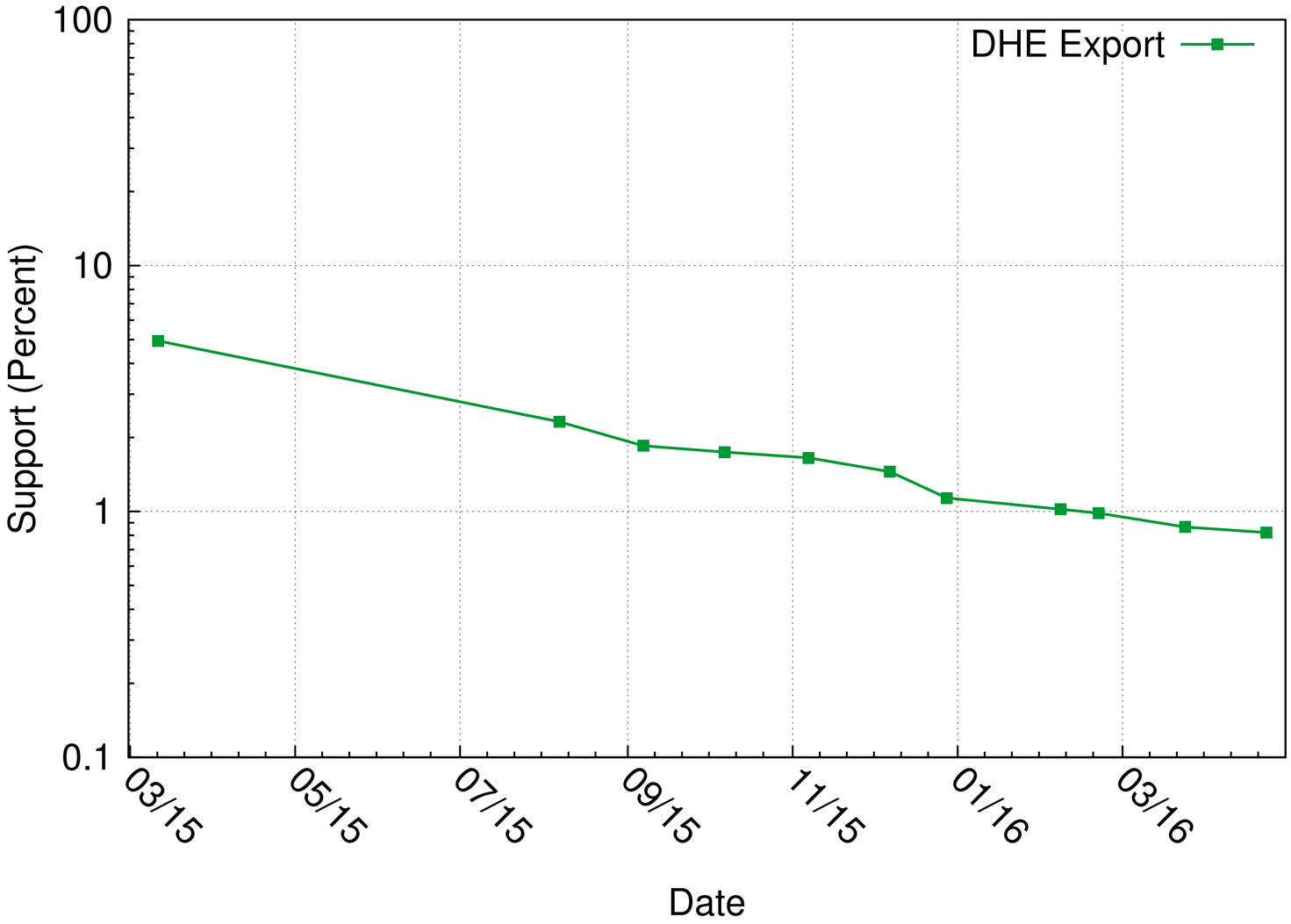
Precomputation takes ~1 week. Not feasible for many unique *p*

- *How many unique 512-bit primes are used by trusted servers?*
- *Do implementations regenerate primes?*

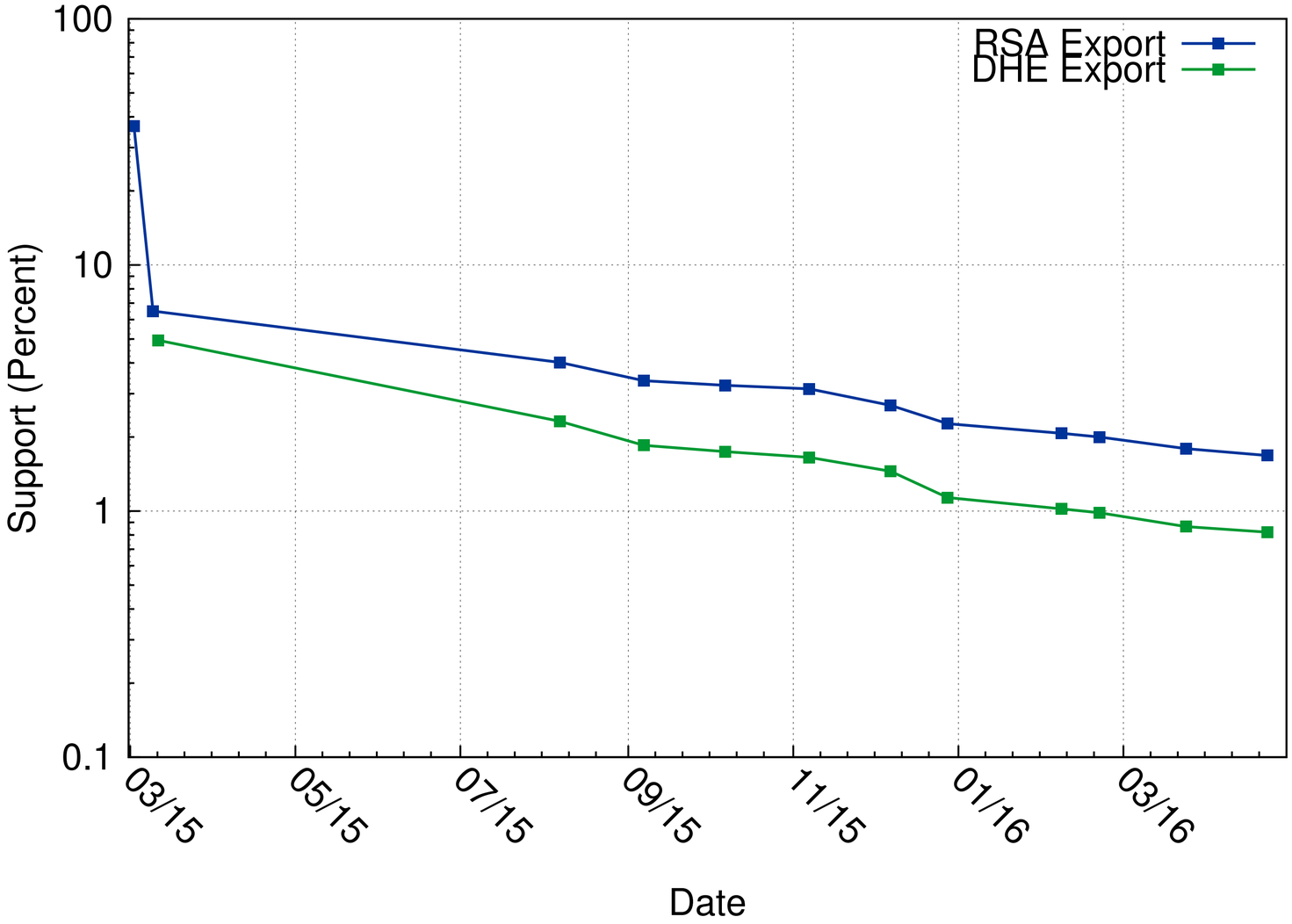
Use ZMap and ZGrab

- *Implement support for export Diffie-Hellman*
- *Parse out selected Diffie-Hellman parameters*

IPv4 Support



IPv4 Support



Top 1M Support

8.5% of the Alexa Top 1M supported `DHE_EXPORT`

3.4% of the trusted IPv4 supported `DHE_EXPORT`

Prime	Popularity among Top 1M domains
Apache mod_ssl	82%
nginx	10%
Other (463 primes)	8%

Implications for Standards

Standardized groups are Diffie-Hellman best practice

- *Many attacks on invalid groups, safer to standardize ahead of time*
- *Need to choose strong enough groups for full lifetime of protocol*

Don’t want to standardize weak groups

- *TLS would need groups strong enough to last longer than two decades*
- *Why standardize export groups when the regulations were being overturned?*

Standardized groups encourage monoculture

- *Could make impact of a 1024-bit break worse*
- *Want to move to ECDHE instead*

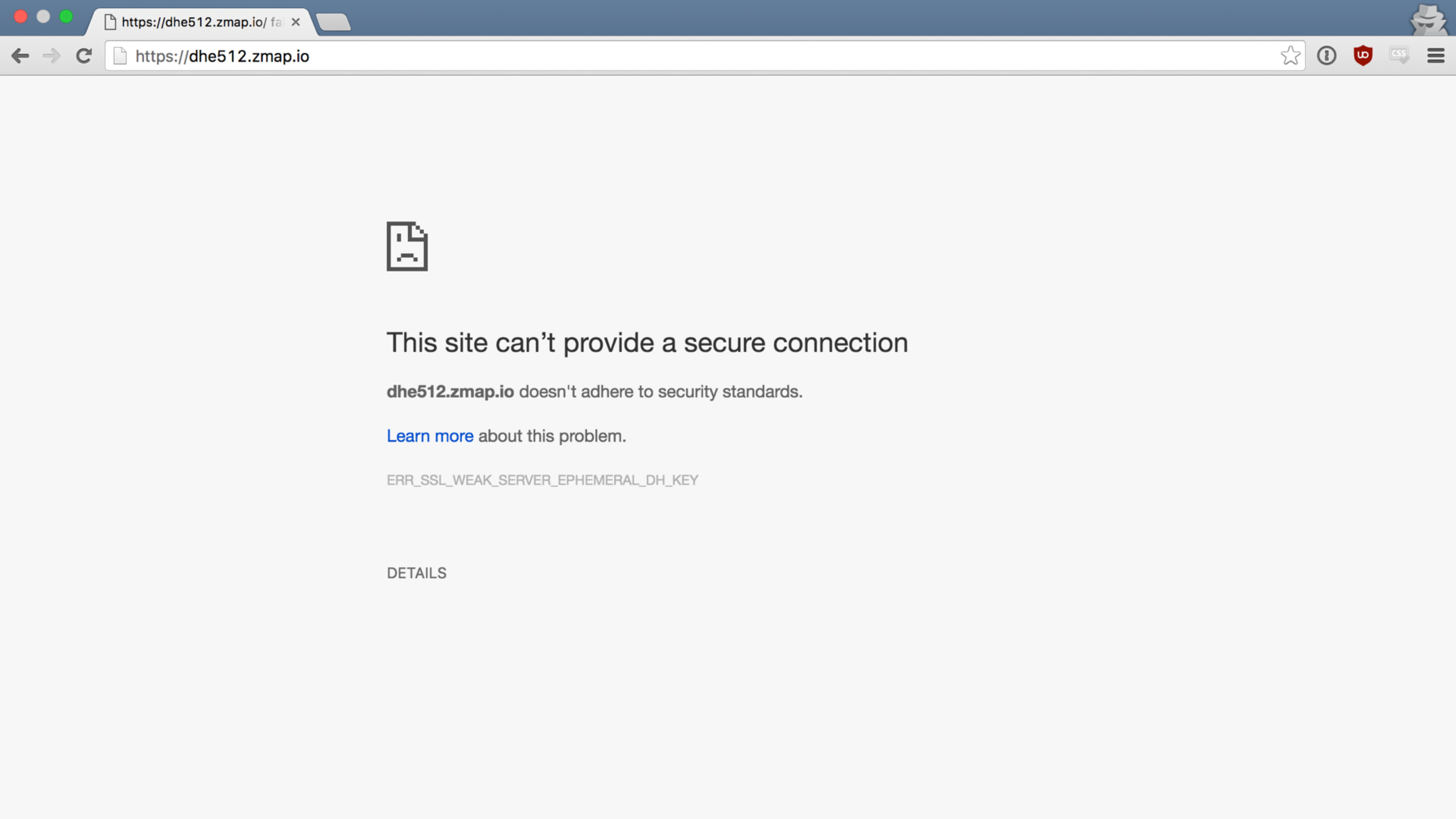
Mitigations

Browsers

- No longer support 512-bit
- Will be sunsetting 768-bit and 1024-bit
- Chrome canary has fully disabled DHE
- ERR_SSL_WEAK_SERVER_EPHEMERAL_DH_KEY

Server Operators

- Disable DHE_EXPORT
- Move to 2048-bit or elliptic curve variant (ECDHE)


https://dhe512.zmap.io/ fa
https://dhe512.zmap.io
This site can’t provide a secure connection
dhe512.zmap.io doesn't adhere to security standards.
Learn more about this problem.
ERR_SSL_WEAK_SERVER_EPHEMERAL_DH_KEY
DETAILS

DROWN

Q: How do you selectively weaken a protocol that uses symmetric ciphers?

A: Send N - 5 bytes of the key in cleartext!

State of SSLv2

SSLv2 is already known to be broken

- *Does not authenticate handshake*
- *Only used for one year (1995), officially deprecated in 2011*

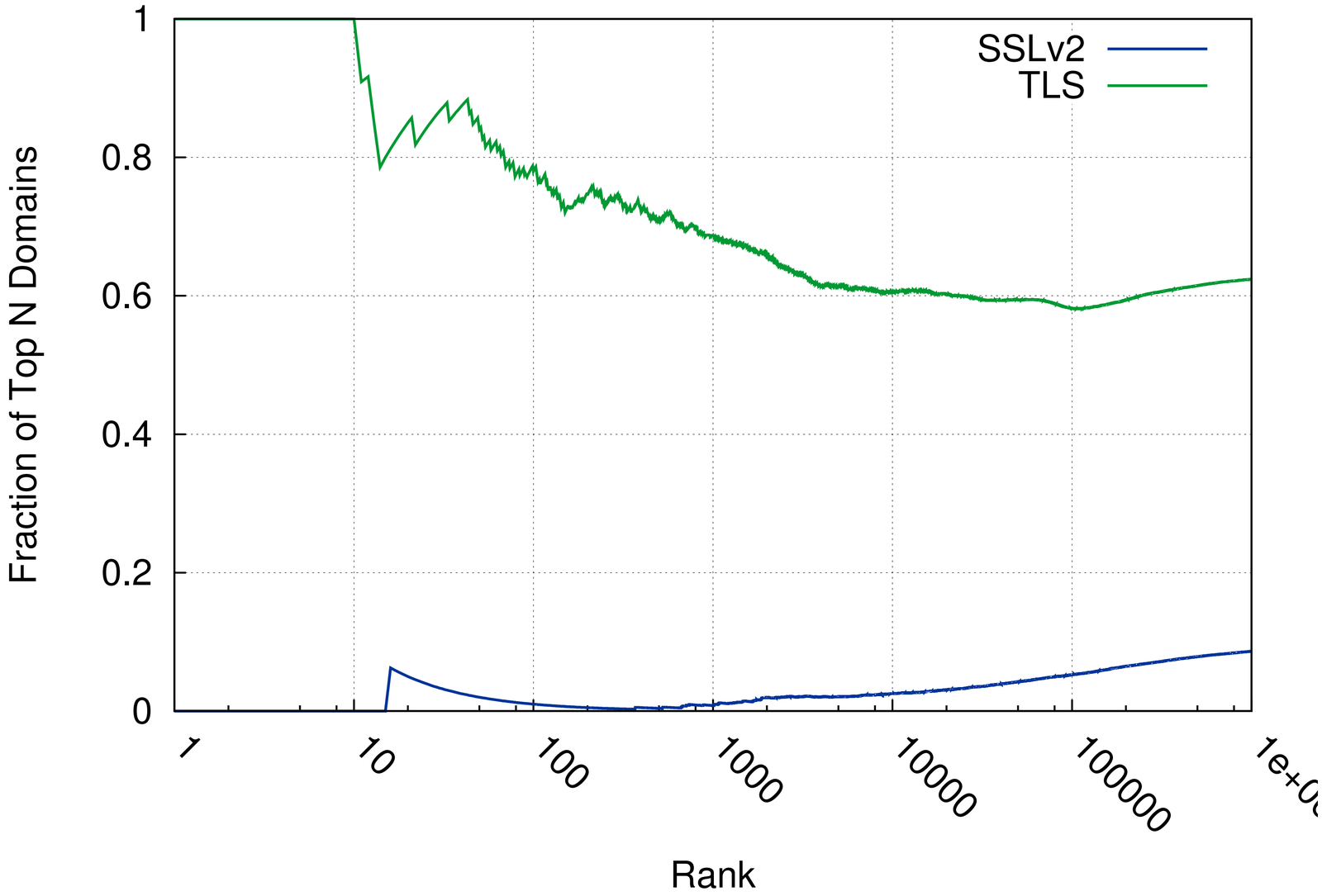
FREAK and Logjam show harms of supporting obsolete cryptography

- *Conventional wisdom for servers was to support all ciphers for compatibility*
- *Recent work has shown this advice to be actively harmful*

Is SSLv2 a harmless vestige, or can it be used to attack modern TLS?

- *SSLv2 has export ciphers, how does this affect modern TLS?*
- *Do servers still support SSLv2 for compatibility? Are people actually using SSLv2?*

Top 1M SSLv2 Support



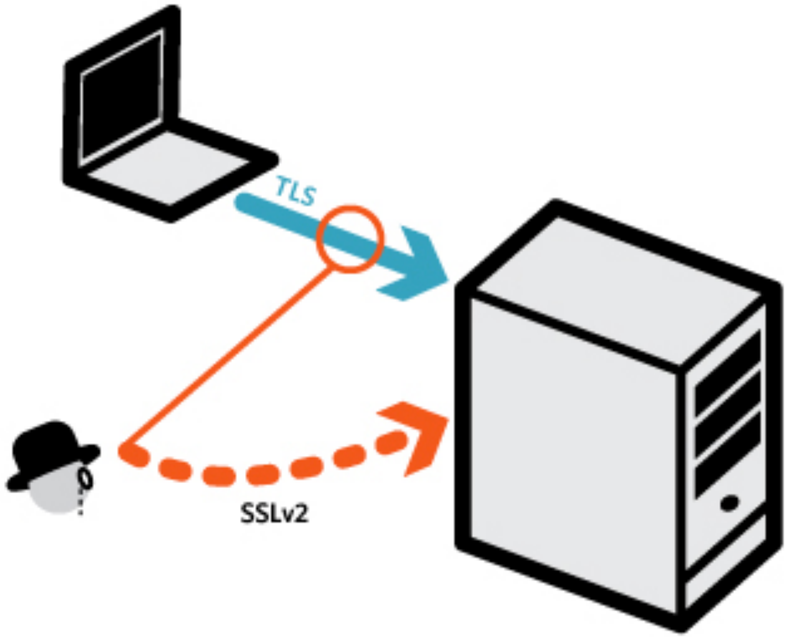
Non-HTTPS SSLv2

Protocol	Port	All Certificates		Trusted Certificates	
		TLS	SSLv2	TLS	SSLv2
SMTP	**25**	**3,357 K**	**936 K (28%)**	**1,083 K**	**190 K (18%)**
POP3	**110**	4,193 K	404 K (10%)	1,787 K	230 K (13%)
IMAP	**143**	4,202 K	473 K (11%)	1,781 K	223 K (13%)
HTTPS	**443**	34,727 K	5,975 K (17%)	17,490 K	1,749 K (10%)
SMTPS	**465**	3,596 K	291 K (8%)	1,641 K	40 K (2%)
SMTP	**587**	3,507 K	423 K (12%)	1,657 K	133 K (8%)
IMAPS	**993**	4,315 K	853 K (20%)	1,909 K	260 K (14%)
POP3S	**995**	4,322 K	884 K (20%)	1,974 K	304 K (15%)

DROWN

A server is vulnerable to DROWN if:

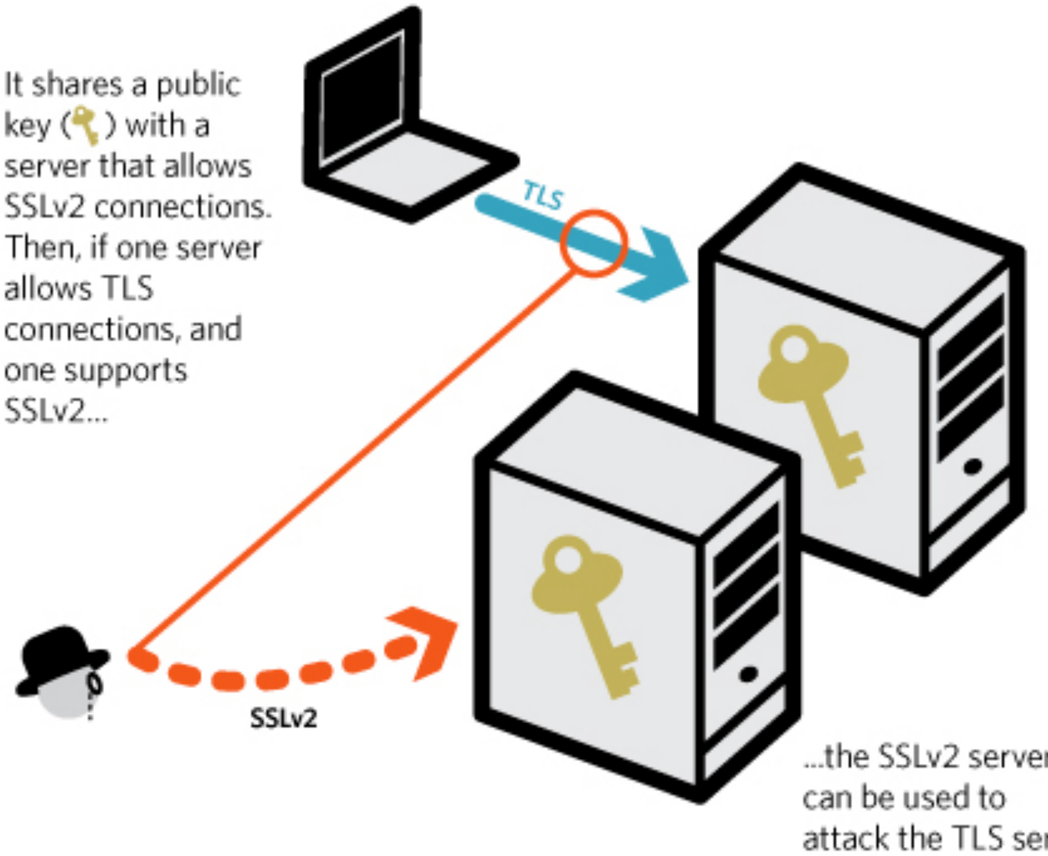
It allows both TLS and SSLv2 connections



17% 83%

17% of HTTPS servers still allow SSLv2 connections

It shares a public key (🔑) with a server that allows SSLv2 connections. Then, if one server allows TLS connections, and one supports SSLv2...



...the SSLv2 server can be used to attack the TLS server

17% 16% 67%

When taking key reuse into account, an additional 16% of HTTPS servers are vulnerable, putting 33% of HTTPS servers at risk

Impact of Key Reuse

		All Certificates			Trusted Certificates		
Protocol	Port	TLS	SSLv2	Vulnerable Key	TLS	SSLv2	Vulnerable Key
SMTP	25	3,357 K	936 K (28%)	1,666 K (50%)	1,083 K	190 K (18%)	686 K (63%)
POP3	110	4,193 K	404 K (10%)	1,764 K (42%)	1,787 K	230 K (13%)	1,031 K (58%)
IMAP	143	4,202 K	473 K (11%)	1,759 K (59%)	1,781 K	223 K (13%)	1,022 K (58%)
HTTPS	443	34,727 K	5,975 K (17%)	**11,444 K (33%)**	17,490 K	1,749 K (10%)	**3,931 K (22%)**
SMTPS	465	3,596 K	291 K (8%)	1,439 K (40%)	1,641 K	40 K (2%)	949 K (58%)
SMTP	587	3,507 K	423 K (12%)	1,464 K (40%)	1,657 K	133 K (8%)	986 K (59%)
IMAPS	993	4,315 K	853 K (20%)	1,835 K (43%)	1,909 K	260 K (14%)	1,119 K (59%)
POP3S	995	4,322 K	884 K (20%)	1,919 K (44%)	1,974 K	304 K (15%)	1,191 K (60%)

Early DROWN Patching

	Disclosure (March 1)	Still Vulnerable (March 26)
Trusted HTTPS Top 1M	25%	15%
Trusted HTTPS	22%	16%
All HTTPS	33%	28%

Special DROWN

Leave no Bleichen-unbachered!

*An **implementation** bug that allows for attackers to man-in-the-middle secure connections.*

	Special DROWN Vulnerable Key	Special DROWN Vulnerable Name
Trusted HTTPS Top 1M	9%	19%
Trusted HTTPS	26%	38%
All HTTPS	26%	—

Mitigations and Lessons

Fully disable SSLv2

- *Don't only disable export ciphers*
- *If only ciphers are disabled, make sure they're actually disabled (CVE-2015-3197)*

Have single-use keys

- *Usually discussed in the context of signatures vs. encryption*
- *Prudent to use different keys across different protocol versions*

Authenticate the client before sending secret-derived data

- *DROWN is possible because of the early* `ServerVerify` *message*
- *Design protocols to check the client has knowledge of the secret first*

Lessons and Implications

Technology Implications

Obsolete cryptography considered harmful

- *Maintaining support for old services is not harmless backward compatibility*
- *Not just harmful as bloat in modern protocols—existence is also harmful*

Limit complexity

- *Cryptographic APIs and state machines are often overly complicated*
- *Design protocols to limit implementation mistakes*
- *Design APIs to limit usage mistakes*

Weakened cryptography considered harmful

- *All forms of export cryptography are now broken*
- *Export RSA (FREAK attack), Export DHE (Logjam), Export symmetric (DROWN)*

Policy Implications

Cryptography regulations have lasting effects

- *Maintaining support for old services is not harmless backward compatibility*
- *Not just harmful as bloat in modern protocols—existence is also harmful*

Technological evidence opposes backdooring cryptography

- *Weakened/export cryptography is not the same as a backdoor*
- *Weakened crypto is arguably less intrusive than backdoors, but still devastating*
- *Current state of technology suggests cryptography is fragile enough*

Cannot assign cryptography based on nationality

- *Internet is global, traffic flows everywhere, CDNs amplify this effect*
- *Can't technologically say a non-US citizen uses different cryptography*

A Retrospective on the Use of Export Cryptography

David Adrian
@davidcadrian

UNIVERSITY OF MICHIGAN

Attacks

https://freakattack.com

https://weakdh.org

https://drownattack.com

Contact

https://davidadrian.org