2014 State of Encryption

14% of the Alexa Top Million websites supported HTTPS

- – Most didn't prefer HTTPS
- – Higher adoption than average websites

Most sites used known-weak versions of TLS

- – Only 1 of 4 popular sites supported latest TLS 1.2

4% of websites supported perfect forward secrecy (PFS)

Only 1 out of 3 emails were encrypted when sent across the Internet

In the beginning, there was nothing.

1994

Netscape Navigator

Mosiac rebrands and releases Netscape Navigator.

1995

SSLv2

February, 1995. Netscape develops SSLv2. It is released, but found to be vulnerable to length extension and MITM attacks.

1995

Internet Explorer

Microsoft release IEv1 (rebranded, licensed Mosiac) in August, 1995.. IEv2 is released three months later.

1995

Apache Web Server

The open-source Apache httpd reaches 1.0 in December. It supports virtual hosting, and largely takes over the public web server market.

1996

SSLv3

Released by Netscape. Becomes the foundation for all of modern HTTPS. In 1999 the IETF adopts SSL and renames it to TLS.

1998

Microsoft IIS 4.0

First version of Microsoft’s web server to support virtual hosting.

SSLv2

UNIVERSITY OF MICHIGAN

Client Hello: random, client-supported ciphers (…) →

Client selects cipher

← **Server Hello**: random, server ciphers (…), certificate

Client Master Key: cipher, mk_{clear}, $Enc_{PK}(mk_{secret})$ →

`write_key, read_key` = $KDF(cipher, mk_{clear} \| mk_{secret})$

← **Server Verify**: $Enc_{SWK}(random_{client})$

Client Finished: $Enc_{CWK}(random_{server})$ →

← **Server Finished**: $Enc_{SWK}(session\_id)$

Record Length (2 bytes)	Padding Length (1 byte)

Data

MAC Data (MD5 Length)	Actual Data (N)	Padding (Padding Length)

Record Length must be multiple of block size
Padding length is only if a block cipher is in use, pads to block length

MAC = MD5(secret, actual data, padding data, sequence number)
ENC-DATA = ENC(padding length, MAC, actual data, padding)

UNIVERSITY OF MICHIGAN

SSLv2 Problems

- No commitment to the handshake messages
  - MITM can force a downgrade without knowing the keys, including downgrade to export-grade ciphers
- Fixed to non-HMAC MD5 hash function
  - No collision resistance, does have preimage and second-preimage resistance
  - MAC is not an HMAC, it's just a keyed hash, so it's vulnerable to length extension
  - $HMAC(H, k, m) = H(k || H(k || m))$
- No concept of certificate chains, only leaf certificates
  - Could be a positive or a negative
- Only stream cipher is RC4
  - Known issues lowering security level below targets
- Block ciphers are all used in CBC mode
  - Padding oracles

Client Hello: random, client-supported ciphers (…)

Server Hello: random, server ciphers (…), certificate

Client Master Key: cipher, mk_{clear}, $Enc_{PK}(mk_{secret})$

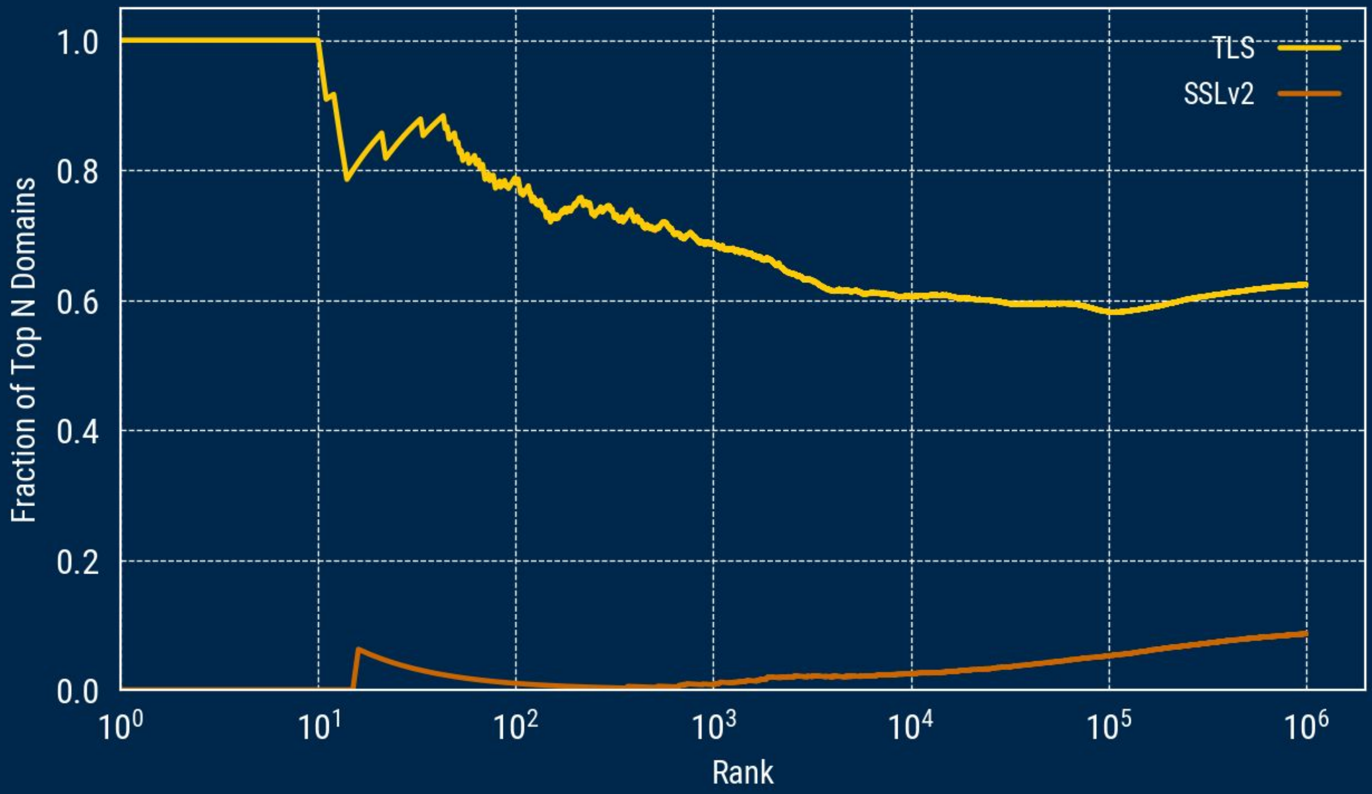
MITM can alter this and forward the rest

`write_key, read_key` = KDF(cipher, $mk_{clear} \,||\, mk_{secret}$)

Server Verify: $Enc_{SWK}(random_{client})$

Client Finished: $Enc_{CWK}(random_{server})$

Server Finished: $Enc_{SWK}(session\_id)$



SSLv2 / TLS Support Among Top 1M Domains
(2016, pre-Drown Attack)

UNIVERSITY OF MICHIGAN

Protocol	Port	All Certificates TLS	All Certificates SSLv2	Trusted Certificates TLS	Trusted Certificates 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%)

SSLv2 Support in Non-HTTPS Protocols
(2016, pre-Drown Attack)

UNIVERSITY OF
MICHIGAN

SSLv2 Good Stuff?

- Uses Key Encapsulation / Data Encapsulation (KEM/DEM)*
  - Use public keys to agree on a random number in secret (encrypt it)
  - Use random number to seed a KDF
  - Use KDF to derive a symmetric key
- Uses record layer with plaintext lengths*
  - Easy to figure out how big your buffer should be when implementing
- Doesn't try to solve key distribution (leaves it for the certificate authorities and the browser)*

*exceptions exist

TLS

UNIVERSITY OF MICHIGAN

Client Hello: client random, ciphers (…RSA…)

Ciphers define more hash types

Server Hello: server random, chosen cipher

Enables intermediate certificates

Certificate: certificate chain (public key *PK*)

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

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

Fixes cipher selection MITM

Client Finished: $E_{K_{ms}}(\text{Hash}(m1 \mid m2 \mid \ldots))$

Server Finished: $E_{K_{ms}}(\text{Hash}(m1 \mid m2 \mid \ldots))$

Client Hello

```
struct {
    ProtocolVersion client_version;
    Random random;
    SessionID session_id;
    CipherSuite cipher_suites<2..2^16-2>;
    CompressionMethod compression_methods<1..2^8-1>;
    select (extensions_present) {
        case false:
            struct {};
        case true:
            Extension extensions<0..2^16-1>;
    };
} ClientHello;
```

[RFC 5246, TLS 1.2, Rescola]

Cipher Suites

Define the key exchange, signature and hash (if needed), and symmetric encryption used for a connection.

```
TLS_RSA_WITH_AES_128_CBC_SHA
TLS_RSA_EXPORT1024_WITH_RC4_56_MD5
TLS_ECDHE_RSA_WITH_CHACHA20_POLY1305_SHA256
TLS_ECDHE_RSA_WITH_AES_128_GCM_SHA256
```

TLS certificates contain >512-bit RSA keys

- OK for authentication!
- *Literally Illegal* for key exchange in the 1990s!

BUT WAIT…

UNIVERSITY OF MICHIGAN

U.S. Export-Grade Cryptography

Until 1992, the United States severely restricted what cryptographic technology could be exported outside of the country. Loosened slightly.

Early 1990s: Two versions of Netscape Browser — US version had full strength crypto (e.g., 1024-bit RSA, 128-bit RC4) and Export version (40-bit RC2, 512-bit RSA)

1996: Bernstein v. the United States: Ninth Circuit Court of Appeals ruled that software source code was speech protected by the First Amendment and that the government's regulations preventing its publication were unconstitutional

Decision later withdrawn, but U.S. changed policy to allow, no precedent set

Export Key Length Restrictions

Regulations applied to communication with non-US entities

Public-key Cryptography: Max 512-bit public keys

- Finite Field Diffie-Hellman (key exchange)
- RSA (key exchange, encryption)

Symmetric Cryptography: Max 40-bit keys

- Block ciphers (DES)
- Stream ciphers (RC4)

Signatures and Message Authentication Codes were *unregulated*

All types of export cryptography have led to attacks against modern cryptography.

TLS Attacks

TLS 1.0 to 1.2

TLS 1.1 (2006)

- Implicit Initialization Vector (IV) is replaced with an explicit IV to protect against Cipher block chaining (CBC) attacks
- Handling of padded errors is changed to use the bad_record_mac alert rather than the decryption_failed alert

TLS 1.2 (2008)

- The MD5/SHA-1 combination in the pseudorandom function (PRF) was replaced with cipher-suite-specified PRFs.
- The MD5/SHA-1 combination in the digitally-signed element was replaced with a single has
- Addition of support for authenticated encryption with additional data modes
- Extensions!

Multiplexing on Names

If you have more than one service per host, you need to multiplexed by some identifier (usually name).

HTTP virtual hosting is powered by the `Host` header. TLS exposes this via the SNI extension (cleartext).

Any secure protocol has to answer:

- How does it multiplex?
- Is the identifier private or public?

Timeline of TLS Attacks

- **2012** — BEAST attack against TLS 1.0 CBC ciphers. Many folks recommend using RC4 in response
- **2012** — CRIME attack shows that TLS compression is broken
- **2013** — Lucky 13: padding oracle attack against CBC cipher suites
- **2014** — POODLE Attack: padding oracle attack against SSLv3 results in browsers removing support
- **2015** — FREAK Attack: protocol vulnerability in TLS allows attackers to trick clients into using “export-grade” cryptography if server supports Export Grade RSA
- **2015** — Logjam Attack: protocol vulnerability found that enables attackers to downgrade some connections to export grade Diffie-Hellman. Browsers remove traditional D-H support.
- **2016** — RC4 deprecation: after a string of attacks against RC4, major browsers remove support
- **2016** — DROWN attack: cross-protocol attack on export-grade AES
- **2016** — Sweet32: Birthday attacks on 64-bit block ciphers like 3DES
- **2017** — First public SHA-1 collision

Timeline of TLS Attacks

2012	BEAST attack against TLS 1.0 CBC ciphers. Many folks recommend using RC4 in response
2012	CRIME attack shows that TLS compression is broken
2013	Lucky 13: padding oracle attack against CBC cipher suites
2014	POODLE Attack: padding oracle attack against SSLv3 results in browsers removing support
2015	FREAK Attack: protocol vulnerability in TLS allows attackers to trick clients into using “export-grade” cryptography if server supports Export Grade RSA
2015	Logjam Attack: protocol vulnerability found that enables attackers to downgrade some connections to export grade Diffie-Hellman. Browsers remove traditional D-H support.
2016	RC4 deprecation: after a string of attacks against RC4, major browsers remove support
2016	DROWN attack: cross-protocol attack on export-grade AES
2016	Sweet32: Birthday attacks on 64-bit block ciphers like 3DES
2017	First public SHA-1 collision

Timeline of TLS Attacks

2012	BEAST attack against TLS 1.0 CBC ciphers. Many folks recommend using RC4 in response
2012	CRIME attack shows that TLS compression is broken
2013	Lucky 13: padding oracle attack against CBC cipher suites
2014	POODLE Attack: padding oracle attack against SSLv3 results in browsers removing support
2015	FREAK Attack: protocol vulnerability in TLS allows attackers to trick clients into using “export-grade” cryptography if server supports Export Grade RSA
2015	Logjam Attack: protocol vulnerability found that enables attackers to downgrade some connections to export grade Diffie-Hellman. Browsers remove traditional D-H support.
2016	RC4 deprecation: after a string of attacks against RC4, major browsers remove support
2016	DROWN attack: cross-protocol attack on export-grade AES
2016	Sweet32: Birthday attacks on 64-bit block ciphers like 3DES
2017	First public SHA-1 collision

Full Timeline: https://www.feistyduck.com/ssl-tls-and-pki-history/

Timeline of TLS Attacks

2012	BEAST attack against TLS 1.0 CBC ciphers. Many folks recommend using RC4 in response
2012	CRIME attack shows that TLS compression is broken
2013	Lucky 13: padding oracle attack against CBC cipher suites
2014	POODLE Attack: padding oracle attack against SSLv3 results in browsers removing support
2015	FREAK Attack: protocol vulnerability in TLS allows attackers to trick clients into using "export-grade" cryptography if server supports Export Grade RSA
2015	Logjam Attack: protocol vulnerability found that enables attackers to downgrade some connections to export grade Diffie-Hellman. Browsers remove traditional D-H support.
2016	RC4 deprecation: after a string of attacks against RC4, major browsers remove support
2016	DROWN attack: cross-protocol attack on export-grade AES
2016	Sweet32: Birthday attacks on 64-bit block ciphers like 3DES
2017	First public SHA-1 collision

Imperfect Forward Secrecy: How Diffie-Hellman Fails in Practice

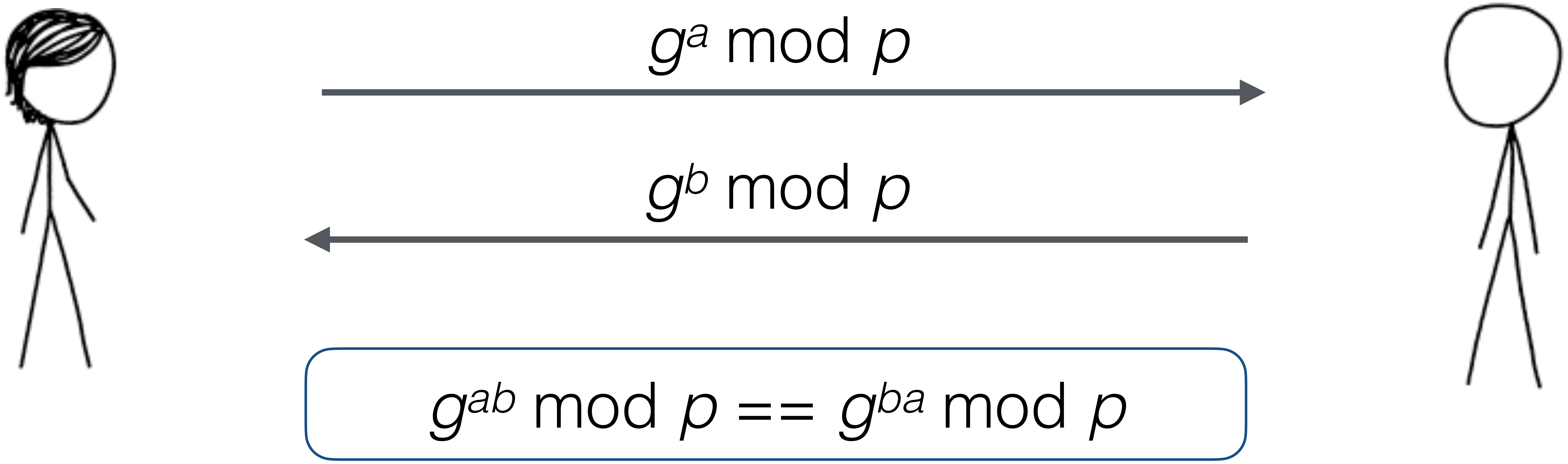
David Adrian, Karthikeyan Bhargavan, Zakir Durumeric, Pierrick Gaudry, Matthew Green, J .
Alex Halderman, Nadia Heninger, Drew Springall, Emmanuel Thomé, Luke Valenta,
Benjamin VanderSloot, Eric Wustrow, Santiago Zanella-Beguelin, and Paul Zimmermann

Diffie-Hellman Key Exchange

First published key exchange algorithm

Public Parameters

- *p* (a large prime)
- *g* (generator for group *p*)



Diffie-Hellman on the Internet

Diffie-Hellman is pervasive on the Internet today

Primary Key Exchange

- SSH
- IPSEC VPNs

Ephemeral Key Exchange

- HTTPS
- SMTP, IMAP, POP3
- all other protocols that use TLS

“Sites that use perfect forward secrecy can provide better security to users in cases where the encrypted data is being monitored and recorded by a third party.”

“Ideally the DH group would match or exceed the RSA key size but 1024-bit DHE is arguably better than straight 2048-bit RSA so you can get away with that if you want to.”

“With Perfect Forward Secrecy, anyone possessing the private key and a wiretap of Internet activity can decrypt nothing.”

2015 Diffie-Hellman Support

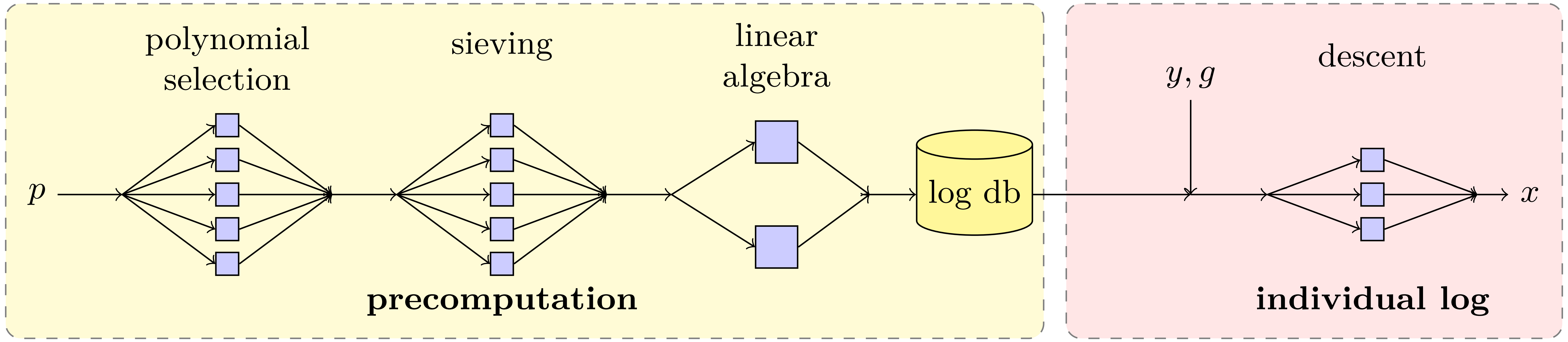
Protocol	Support
HTTPS (Top Million Websites)	68%
HTTPS (IPv4, Browser Trusted)	24%
SMTP + STARTTLS	41%
IMAPS	75%
POP3S	75%
SSH	100%
IPSec VPNs	100%

Breaking Diffie-Hellman

Computing discrete log is best known attack against DH

In other words, Given $g^x \equiv y \bmod p$, compute x

Number Field Sieve

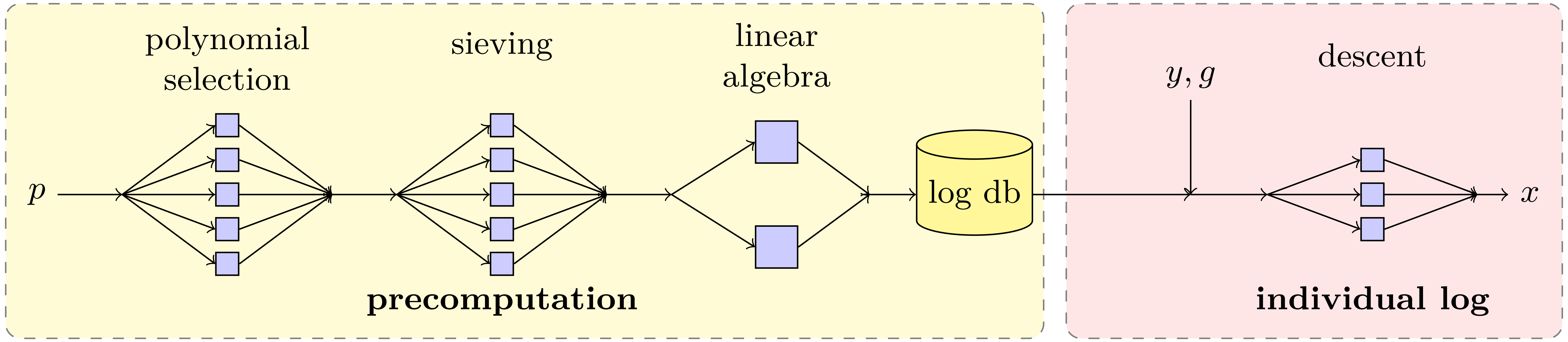


Breaking Diffie-Hellman

Computing discrete log is best known attack against DH

In other words, Given $g^x \equiv y \bmod p$, compute x

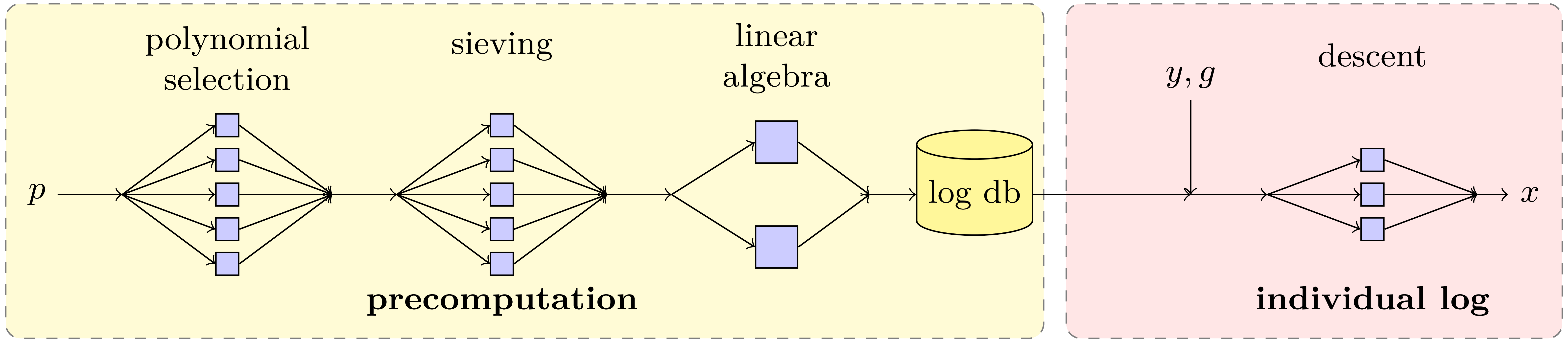
Number Field Sieve



Pre-computation is only dependent on *p*!

Breaking Diffie-Hellman

Number Field Sieve



	Sieving	Linear Algebra	Descent
DH-512	2.5 core years	7.7 core years	10 core min.

Lost in Translation

This was known within the cryptographic community

However, not within the systems community

66% of IPSec VPNs use a single 1024-bit prime

Lost in Translation

This was known within the cryptographic community

However, not within the systems community

66% of IPSec VPNs use a single 1024-bit prime

Are the groups used in practice still secure given this "new" information?

512-bit Keys and the Logjam Attack on TLS

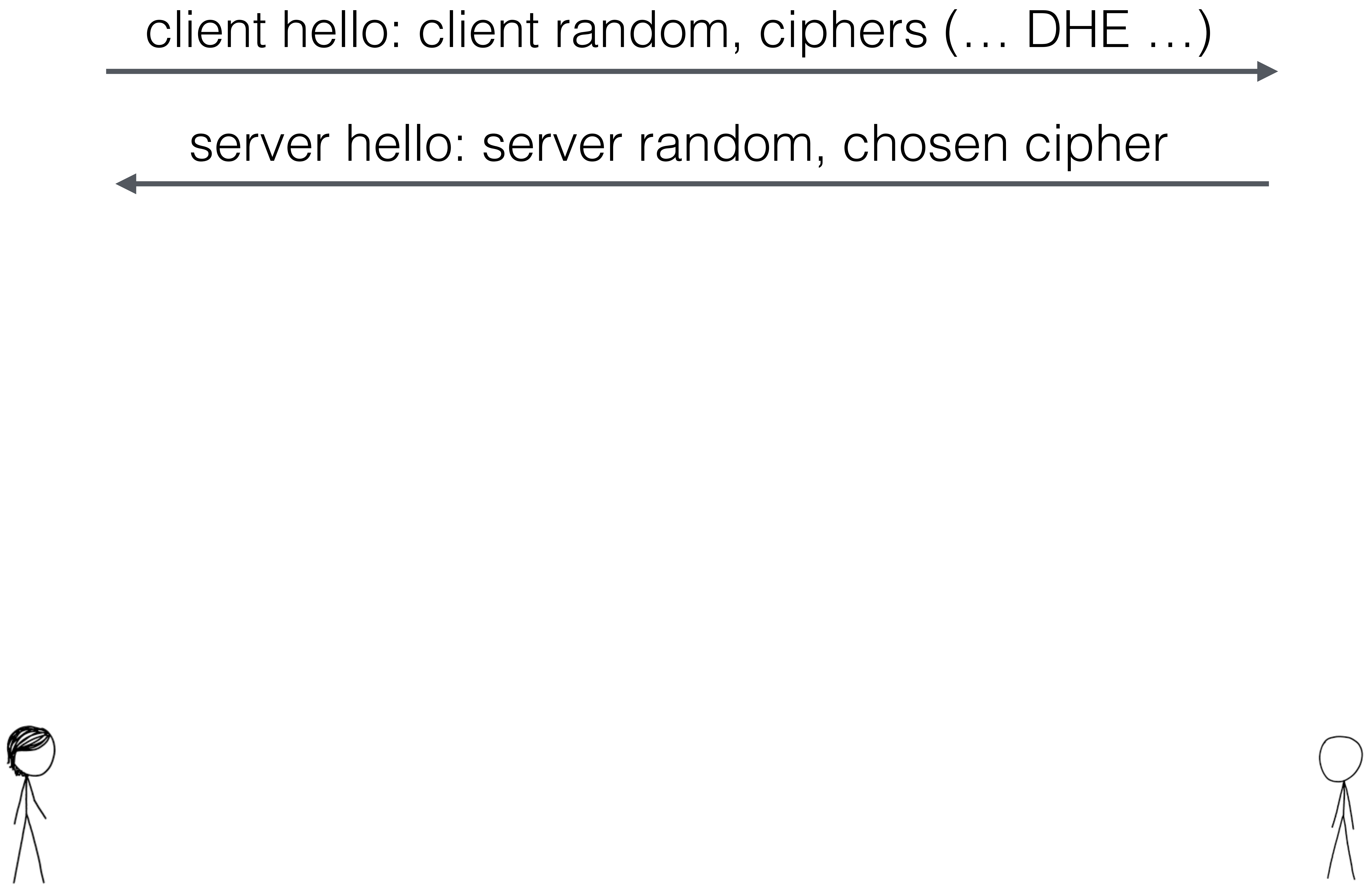
Diffie-Hellman in TLS

The majority of HTTPS websites use 1024-bit DH keys

However, nearly 8.5% of Top 1M still support *Export DHE*

Source	Popularity
Apache	82%
mod_ssl	10%
Other (463 distinct primes)	8%

Normal TLS Handshake



Normal TLS Handshake

client hello: client random, ciphers (… DHE …) →

← server hello: server random, chosen cipher

← certificate, p, g, g^a, $\text{Sign}_{\text{CertKey}}(p, g, g^a)$

g^b →

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

Normal TLS Handshake

client hello: client random, ciphers (… DHE …) →

← server hello: server random, chosen cipher

← certificate, p, g, g^a, $\text{Sign}_{\text{CertKey}}$(p, g, g^a)

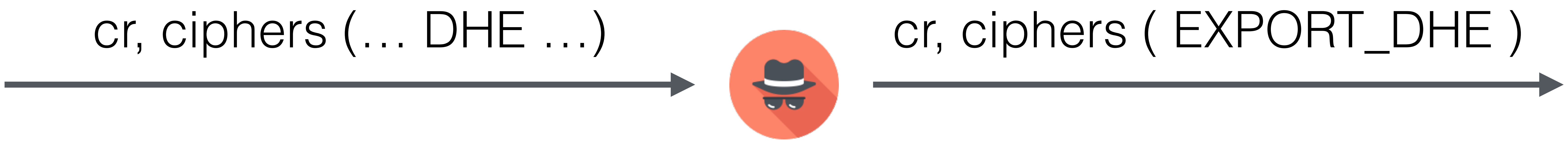
g^b →

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

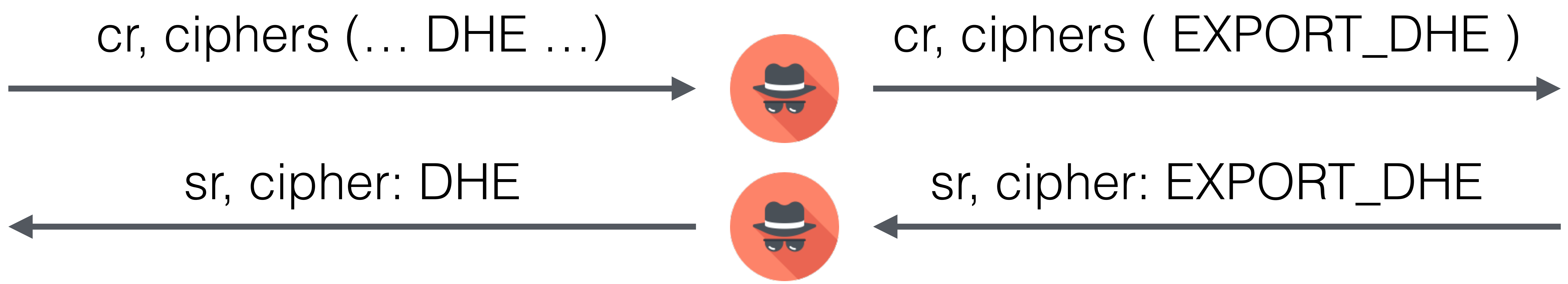
client finished: Sign_{Kms}(Hash(m1 | m2 | …)) →

← server finished: Sign_{Kms}(Hash(m1 | m2 | …))

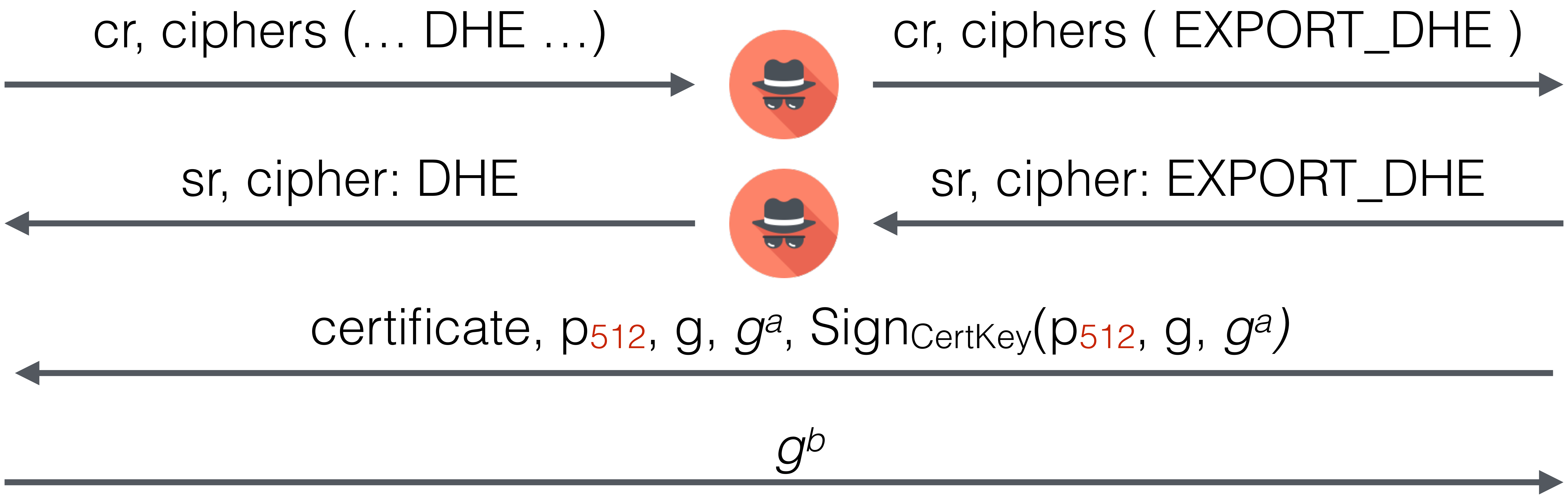
Logjam Attack



Logjam Attack

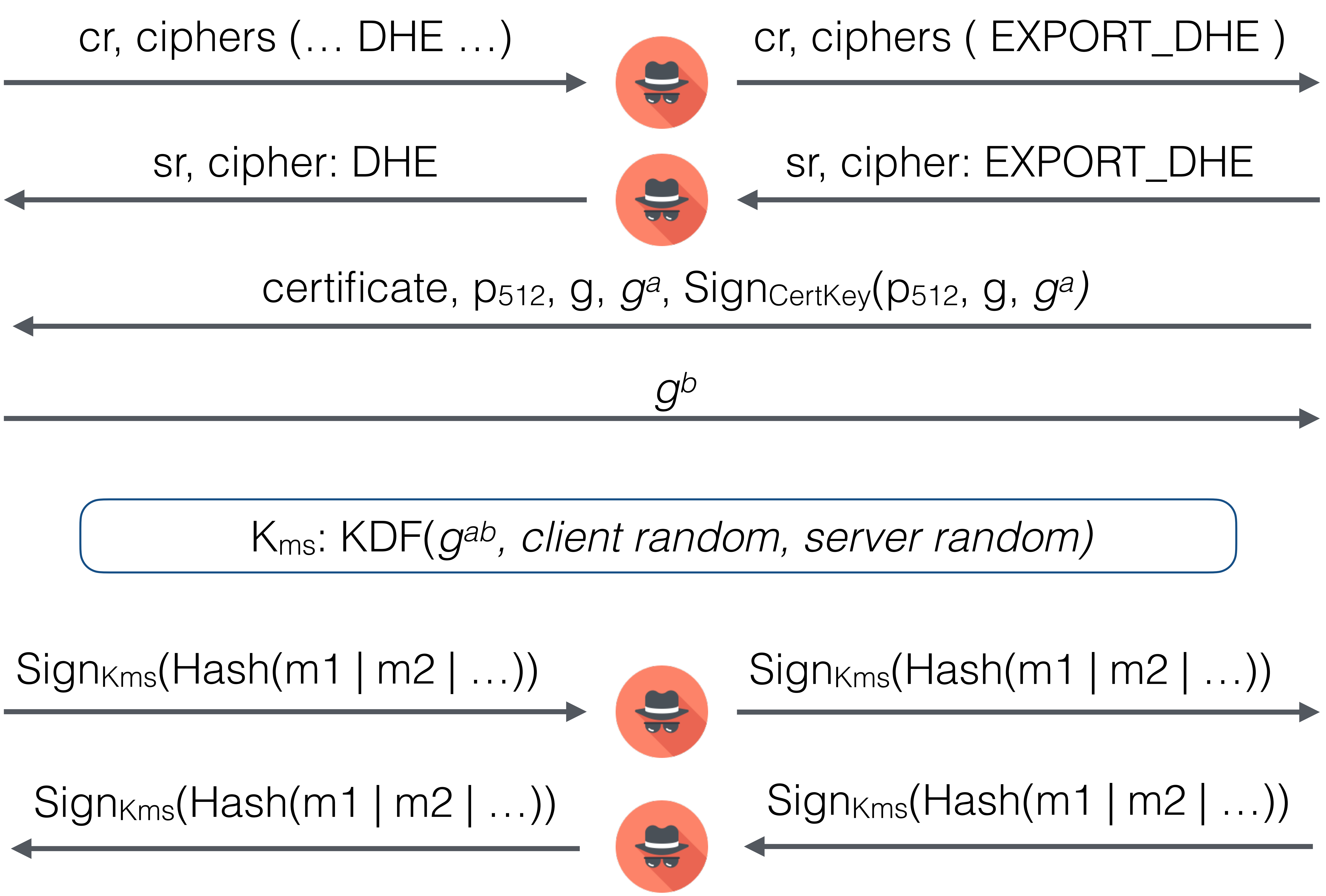


Logjam Attack



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

Logjam Attack



Computing 512-bit Discrete Logs

We modified CADO-NFS to compute two common primes

1 week pre-computation, individual log ~70 seconds

	polysel	sieving	linalg	descent
	2000-3000 cores		288 cores	36 cores
DH-512	3 hours	15 hours	120 hours	70 seconds

Logjam Mitigation

Browsers

- have raised minimum size to 768-bits
- ~~plan to move to 1024-bit in the future~~
- plan to drop all support for DHE

Server Operators

- Disable export ciphers!!
- ~~Use a 2048-bit or larger DHE key~~
- ~~If stuck using 1024-bit, generate a unique prime~~
- Moving to ECDHE

768- and 1024-bit Keys

Breaking One 1024-bit DH Key

Estimation process is convoluted due to the number of parameters that can be tuned.

Crude estimations based on asymptotic complexity:

	Sieving core-years	Linear Algebra core-years	Descent core-time
RSA-512	0.5	0.33	
DH-512	2.5	7.7	10 mins
RSA-768	800	100	
DH-768	8,000	28,500	2 days
RSA-1024	1,000,000	120,000	
DH-1024	10,000,000	35,000,000	30 days

Custom Hardware

If you went down this route, you would build ASICs

Prior work from Geiselmann and Steinwandt (2007) estimates ~80x speed up from custom hardware.

≈$100Ms of HW precomputes one 1024-bit prime/year

Custom Hardware

If you went down this route, you would build ASICs

Prior work from Geiselmann and Steinwandt (2007) estimates ~80x speed up from custom hardware.

≈$100Ms of HW precomputes one 1024-bit prime/year

For context… annual budgets for the U.S.

- Consolidated Cryptographic Program: 10.5B
- Cryptanalyic IT Services: 247M
- Cryptanalytic and exploitation services: 360M

TLS 1.3

TLS 1.3 What's New?

Removed:

- Problematic features from the past like compression, renegotiation
- Known broken ciphers like MD-5, SHA-1, RC4, 3DES, CBC mode, traditional finite-field Diffie-Hellman, export ciphers, user defined groups
- Non-PFS (perfect forward secret) handshakes, non-AEAD ciphers

Added:

+ Simplified handshake with one fewer round trip
+ Protection against downgrade attacks (e.g., signature over entire exchange)
+ Support for newer elliptic curves (e.g., x25519 and 448)
+ Zero RTT Session Resumption (performance win)

TLS 1.3 Design

TLS 1.3 was finalized in 2018! Process took ~5 years.

One of first major protocols to involve academic community during design. Uncovered multiple attacks, including a downgrade, cross-protocol, and key-sharing attack

Empirical tests helped design a handshake that minimizes interference with broken middle boxes

TLS 1.3 Client Hello

Problem: Needs to look like a TLS 1.2 Client Hello for compatibility reasons, but work in new ways with 1.3 servers.

Solution: TLS 1.3 only cipher suites, move version negotiation and key share to an extension, deprecate old fields. The protocol can diverge from old versions after the Client Hello.

```
struct {
  ProtocolVersion legacy_version = 0x0303;    /* TLS v1.2 */
  Random random;
  opaque legacy_session_id<0..32>;
  CipherSuite cipher_suites<2..2^16-2>;
  opaque legacy_compression_methods<1..2^8-1>;
  Extension extensions<8..2^16-1>;
} ClientHello;
```

TLS 1.3 0-RTT Mode

By agreeing on a PSK to use with future connections, it is possible for a client to being future connections before waiting for a server response [RFC 8446, Rescola, 2018]

These messages are replayable. This could lead to security flaws. The spec says not to handle requests that modify data until after the replay window is up (after the server finishes the handshake).

Functionality primarily used by “Big Tech”

This is the sketchiest part of all of TLS 1.3. Someone should measure this. Maybe ICSI has?

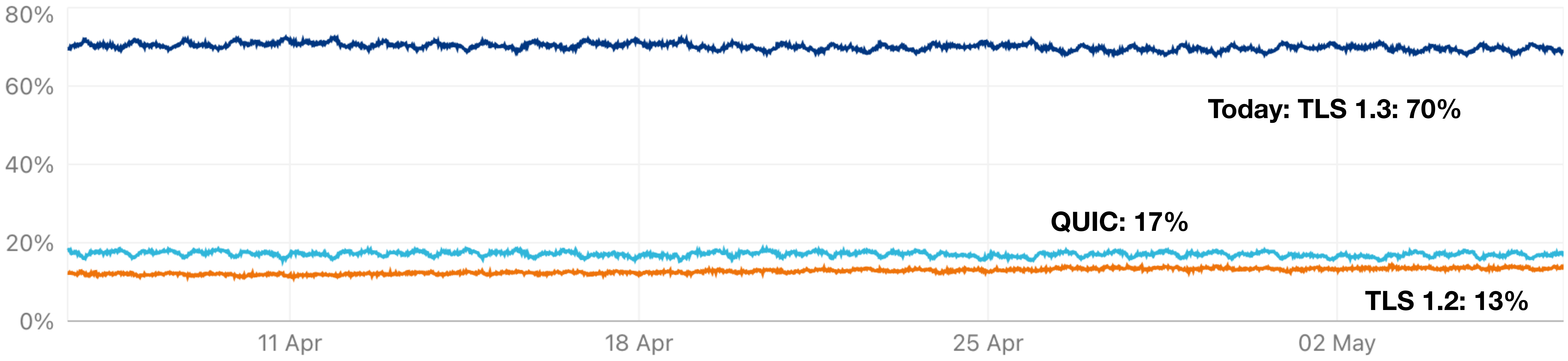
A Look Back on SSLv2: The Good Parts

TLS 1.3 fully drops the RSA KEM/DEM design inherited from SSLv2.
Not a knock on all KEM/DEM, but we have better ways of doing key agreement (DH).

Switching to AEADs makes it even easier to have authenticated plaintext data associated with a encrypted payload
Plaintext header data continues to make protocol implementation easier

For better or for worse, we still use X.509 certificates as the primary Web PKI.
Stare not into the abyss, lest you become recognized as an abyss domain expert, and they keep expecting you to stare into the damn thing.

TLS 1.3 Adoption



CLOUDFLARE

Data shown from Apr 6, 2021 1:00 PM (UTC) to May 6, 2021 1:00 PM (UTC)
Source: https://radar.cloudflare.com

Noise Protocol Framework

Noise is a set of guidelines for describing protocols for authenticated secure channels using Diffie-Hellman as the only asymmetric primitive, combined with an AEAD.

There are no signatures!

The two parties are an **initiator** and a **responder.**