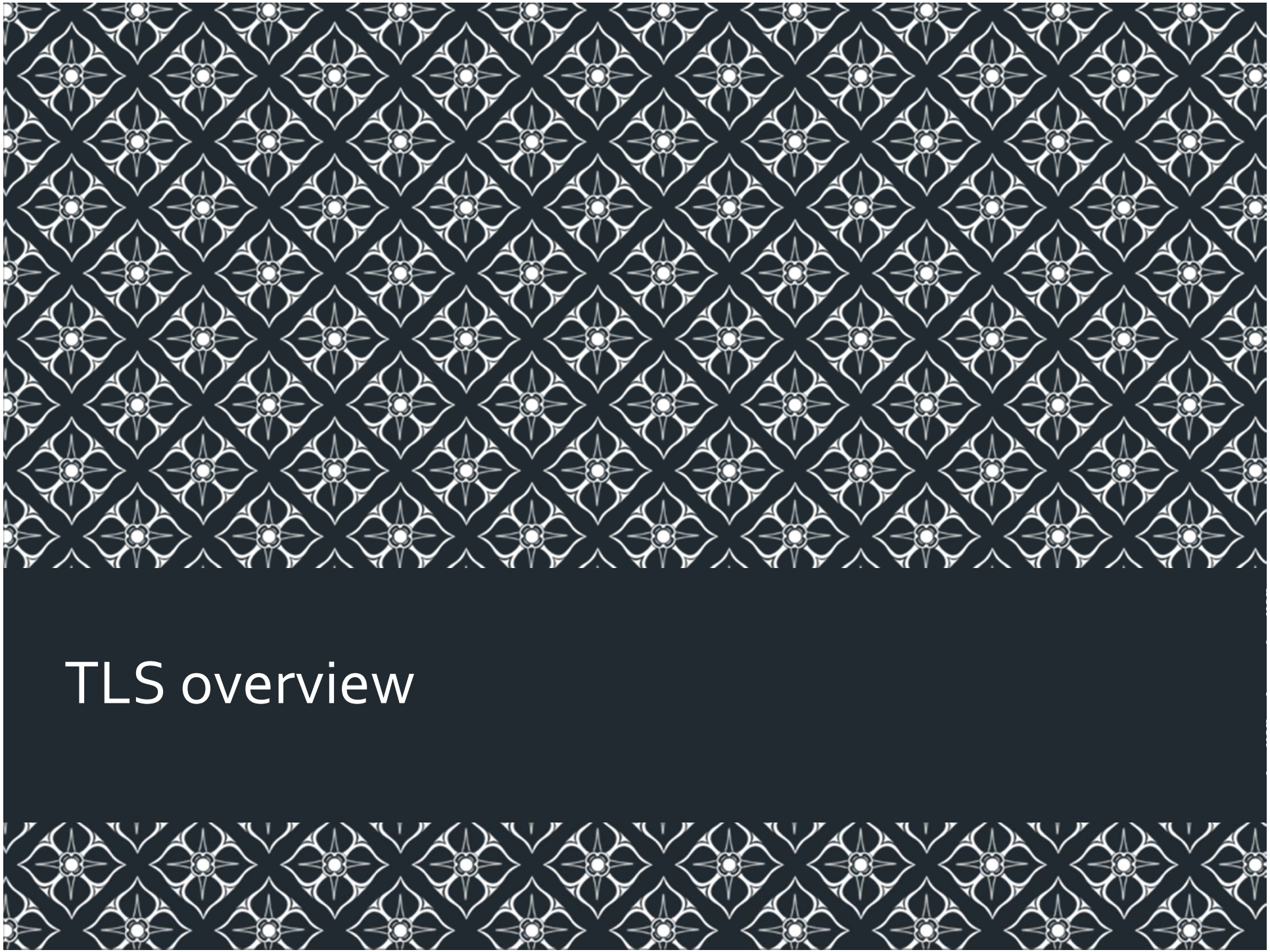
Bar-Ilan Winter School
Lecture 1-3
TLS: A Real-World Secure Channel Protocol

Kenny Paterson @kennyog

ROYAL HOLLOWAY UNIVERSITY OF LONDON

Lecture overview

- Aim to provide a reasonably detailed picture of a complex, widely-deployed protocol.
- Illustrating how the different cryptographic concepts we've been learning about are combined in a real protocol.
- Understand some of the additional protocol features that real-world requirements lead to.
- Study some high-profile attacks on TLS, and see how the protocol's deployment has evolved over time.
- Look at TLS 1.3 – the future of TLS.


TLS overview

TLS Overview

- SSL = Secure Sockets Layer.
  - Developed by Netscape in mid 1990s.
  - SSLv1 broken at birth.
  - SSLv2 flawed in several ways, now IETF-deprecated (RFC 6176).
  - SSLv3 now considered broken (POODLE + RC4 attacks), but still widely supported.

- TLS = Transport Layer Security.
  - IETF-standardised version of SSL.
  - TLS 1.0 in RFC 2246 (1999).
  - TLS 1.1 in RFC 4346 (2006).
  - TLS 1.2 in RFC 5246 (2008).
  - TLS 1.3 currently in development in IETF.

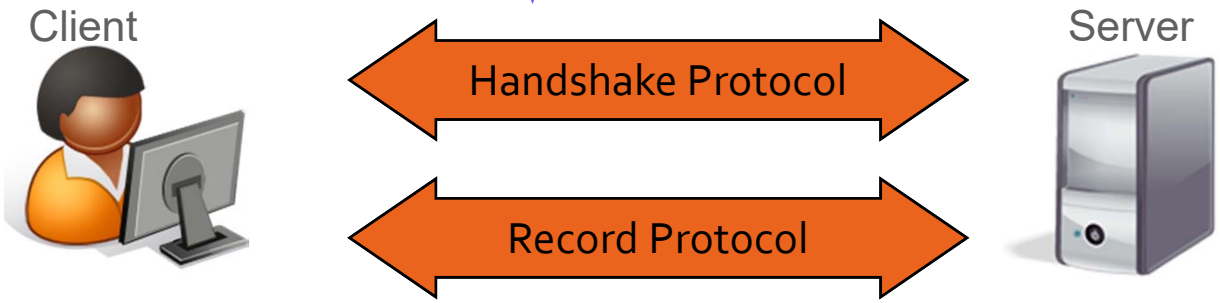
Importance of TLS

- Originally designed for secure e-commerce, now used much more widely.
  - Retail customer access to online banking facilities.
  - Access to gmail, facebook, Yahoo, etc.
  - Mobile applications, including banking apps.
  - Payment infrastructures.

- TLS has become a *de facto* secure protocol of choice.
  - Used by hundreds of millions (billions?) of people and devices every day.
  - So we should analyse it, in order to find and remove flaws.
  - Maybe we'll also find new requirements not well-reflected in the current key exchange literature?

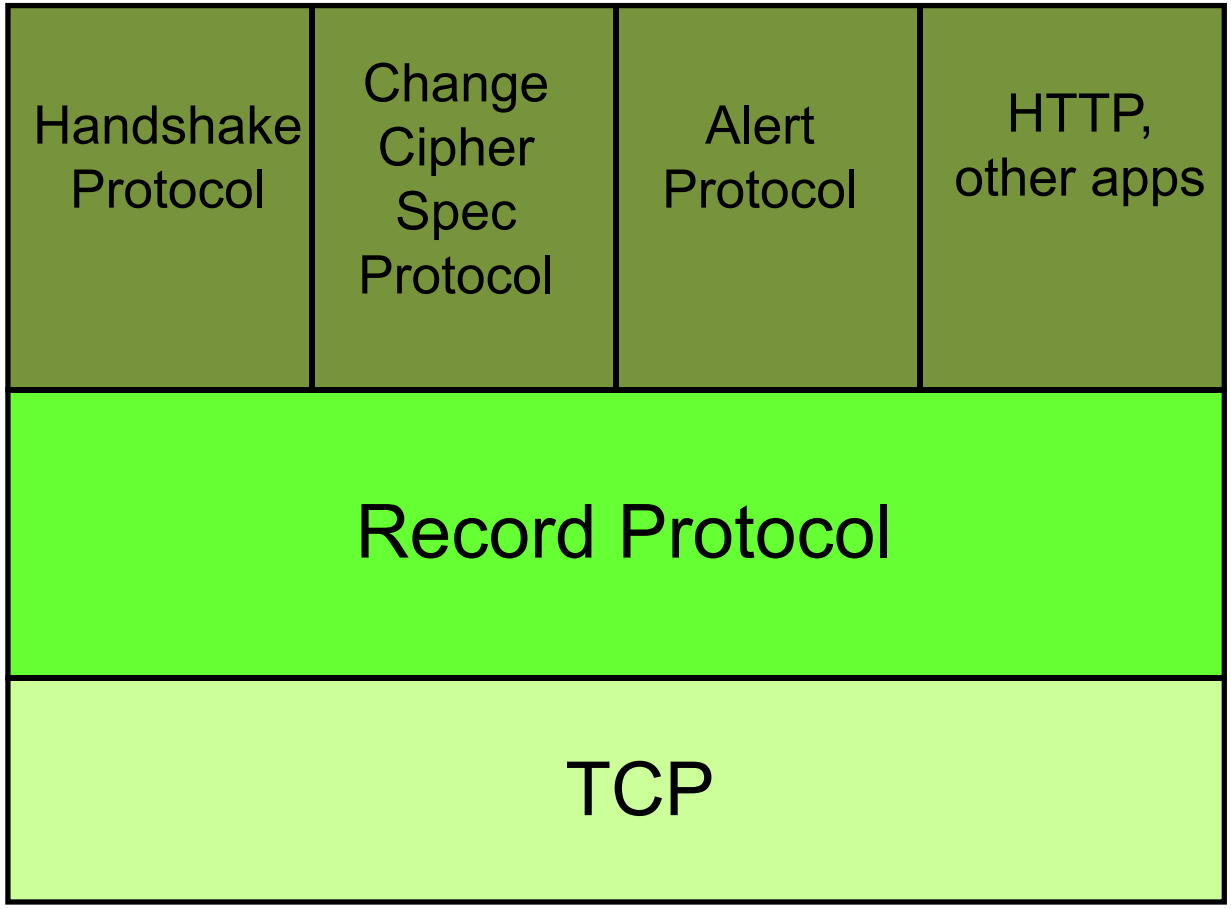
Highly Simplified View of TLS

Used by client and server to
1. Negotiate cipher suite
2. Authenticate
3. Establish keys used in the Record Protocol



Provides confidentiality and integrity/authenticity of application layer data using keys from Handshake Protocol

TLS Protocol Architecture



The TLS Ecosystem (1/3)

- Servers
  - Including managed service providers (CloudFlare, Akamai)
- Clients
  - Of all shapes and sizes
  - Web browsers to embedded devices
- Certification Authorities (CAs)
  - Of all shapes , sizes and levels of security
  - Typically 300 root CA keys in browser.
- Software vendors
  - From Google (BoringSSL) down to one-man open-source operations
  - OpenSSL somewhere in-between, once used by approx 90% of web servers.
- Hardware vendors

The TLS Ecosystem (2/3)

- TLS versions:
  - SSL 3.0, TLS 1.0, TLS 1.1, TLS 1.2, TLS 1.3 nearly complete
  - Some servers even still support SSL 2.0
- 200+ cipher suites
  - https://www.thesprawl.org/research/tls-and-ssl-cipher suites
  - Some very common, e.g.

    TLS_RSA_WITH_AES_128_CBC_SHA256
  - Some highly esoteric, e.g.

    TLS_KRB5_WITH_3DES_EDE_CBC_MD5
  - Some offering no security:

    TLS_NULL_WITH_NULL_NULL !
- TLS extensions
  - Too numerous to mention.
- DTLS

The TLS Ecosystem (3/3)

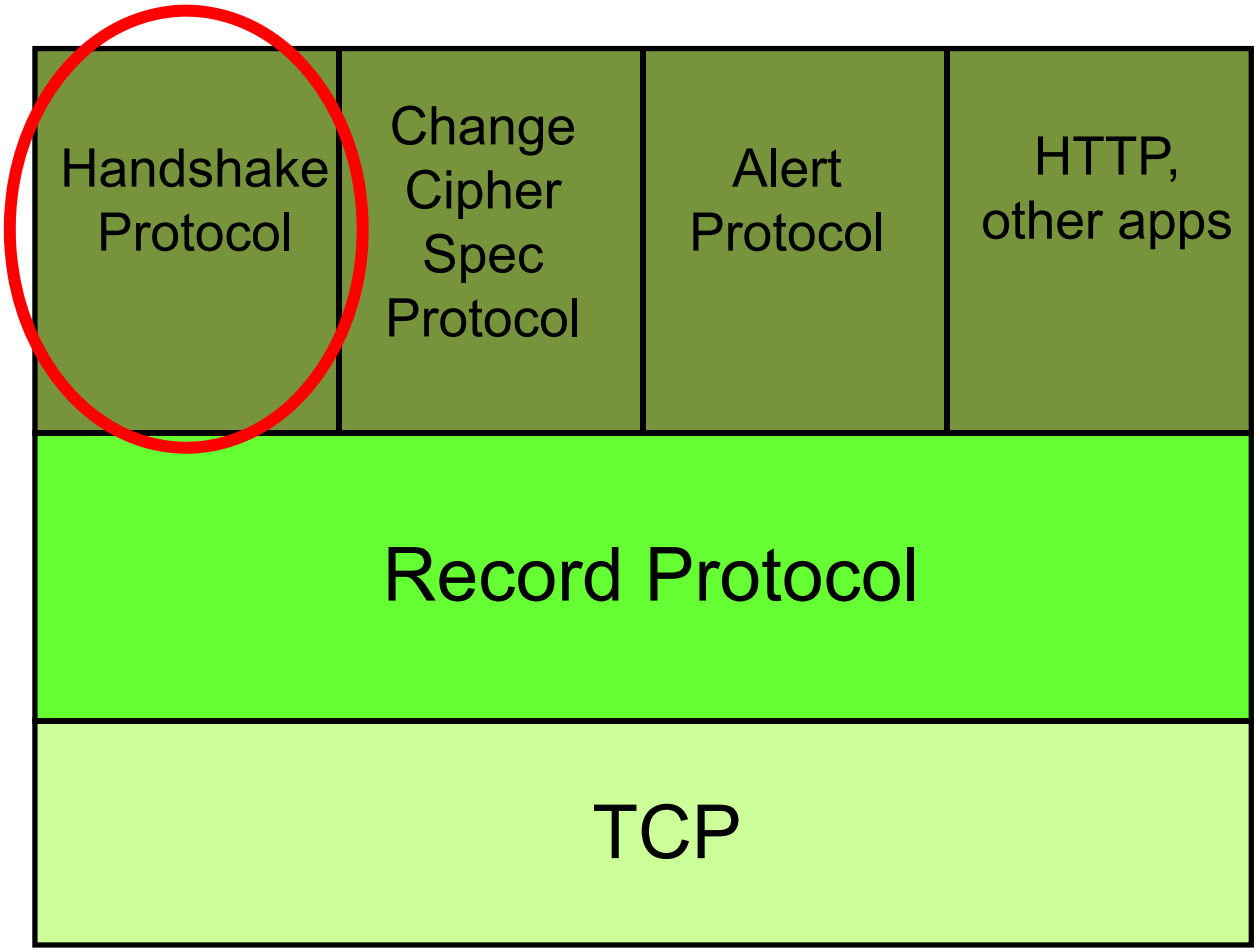
- IETF TLS Working Group
  - Also IETF UTA Working Group (UTA = Using TLS in Applications)
  - And CFRG (Crypto Forum Research Group)
- Growing community of researchers
  - Finding attacks and building security proofs
  - Analysis of TLS 1.3
  - TRON workshop and follow-ups, including CWTLS1.3 co-located with CRYPTO 2018.
- The TLS ecosystem has become very complex and vibrant.


TLS Record Protocol

Redacted


TLS Handshake Protocol

TLS Protocol Architecture



TLS Handshake Protocol – Goals

Establishes keys (and IVs) needed by the Record Protocol.

- Via establishment of the TLS master_secret and subsequent key derivation.

Provides authentication of server (usually) and client (rarely)

- Using public key cryptography supported by digital certificates.
- Or pre-shared key (less commonly).

Protects negotiation of all cryptographic parameters.

- SSL/TLS version number.
- Encryption and hash algorithms.
- Authentication and key establishment methods.
- To prevent version rollback and cipher suite downgrade attacks.

TLS Handshake Protocol – Key Establishment

TLS supports several key establishment mechanisms.

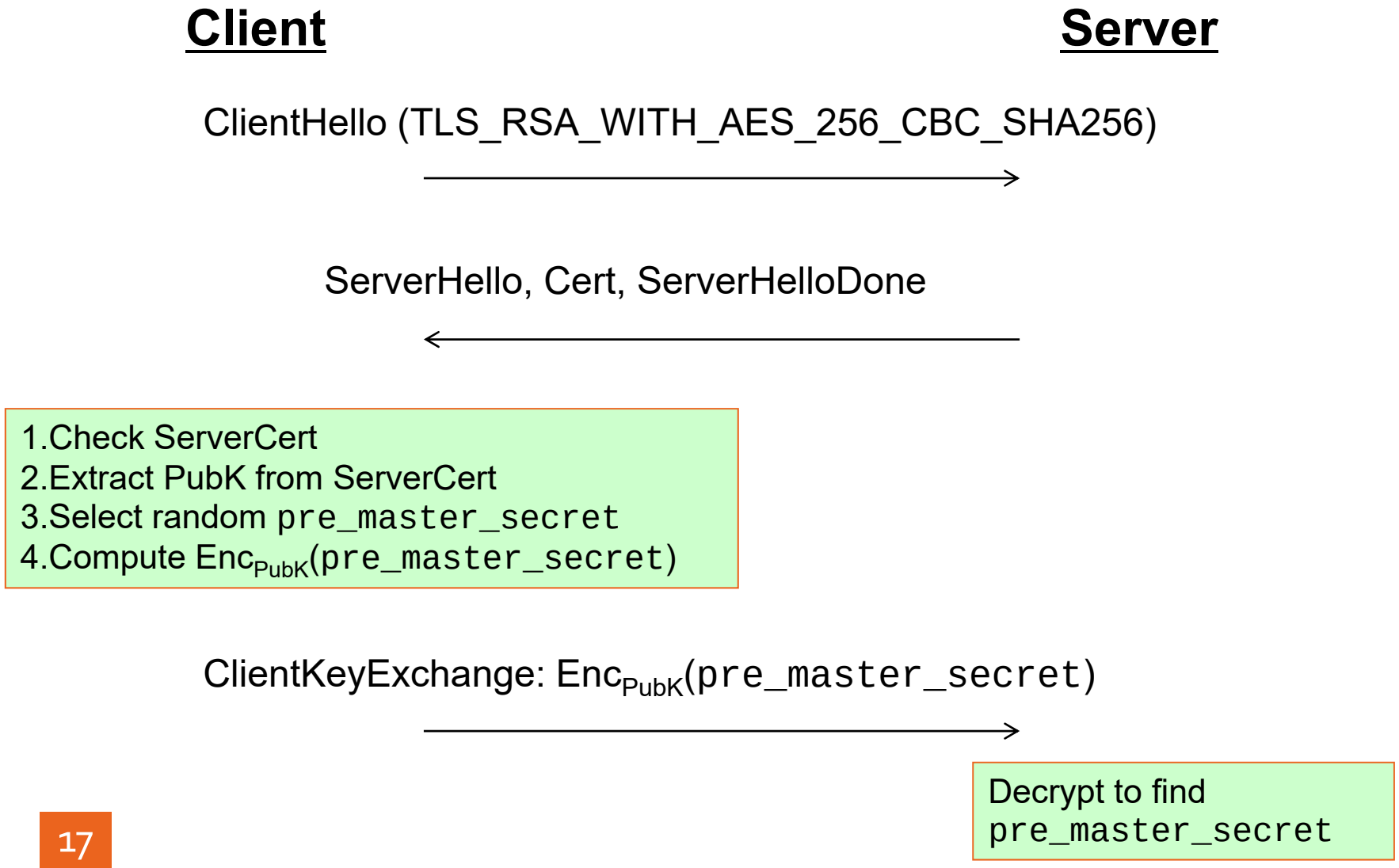
Method used is negotiated during the Handshake Protocol itself.

- Client sends list of *cipher suites* it supports in ClientHello; server selects one and tells client in ServerHello.
- e.g. TLS_RSA_WITH_AES_256_CBC_SHA256
- e.g. TLS_ECDHE_RSA_WITH_RC4_128_SHA
- cipher suites are encoded as 2-byte values.

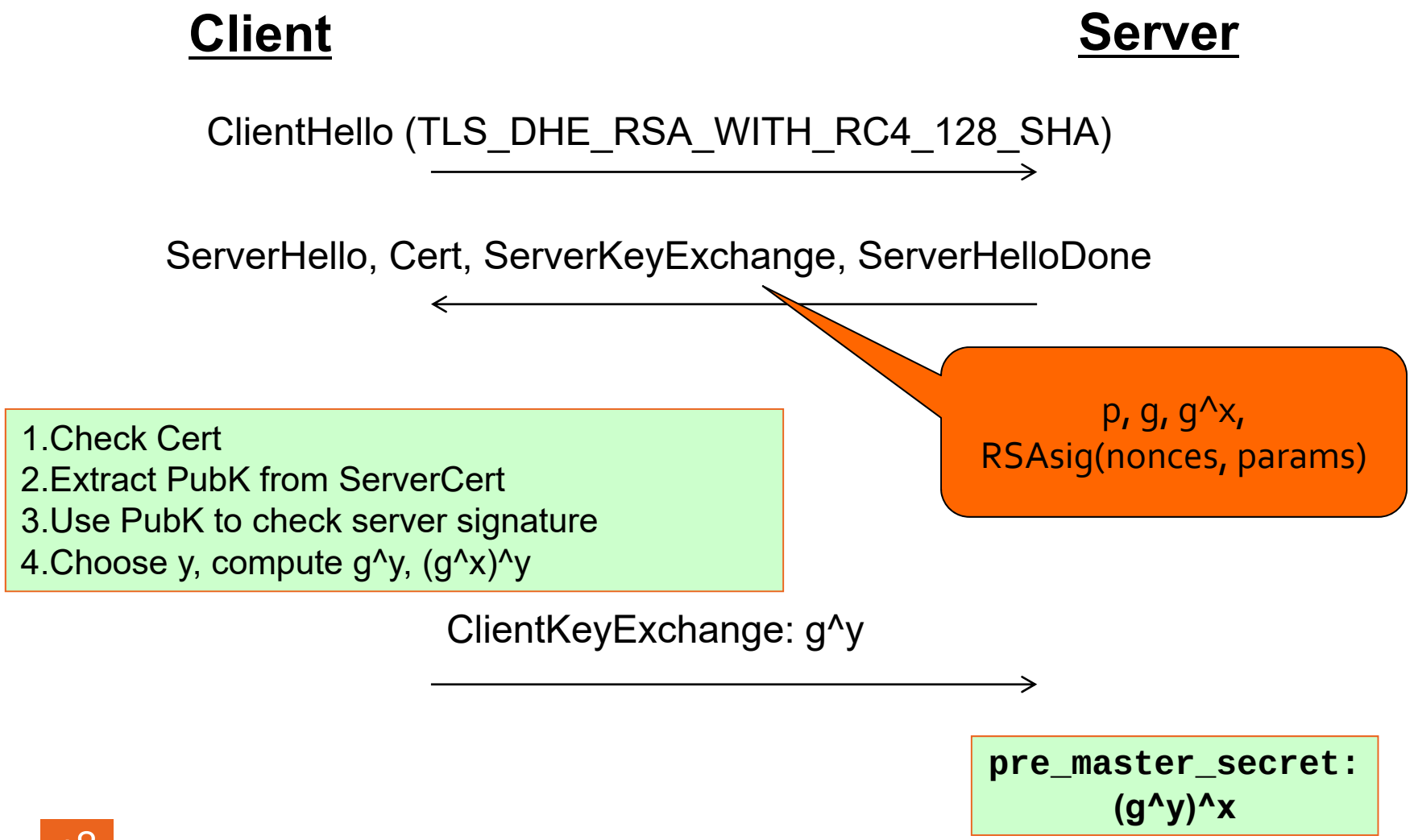
A common choice is RSA encryption.

- Server sends RSA public key and certificate chain after ServerHello.
- Client chooses pre_master_secret, encrypts using public RSA key of server, sends to server.
- RSA encryption is based on PKCS#1 v1.5 padding method.

TLS Handshake Protocol – RSA-based Key Establishment (Simplified)



TLS Handshake Protocol – Ephemeral DH-based Key Establishment (Simplified)



TLS Handshake Protocol – Other Key Establishment Options

Static Diffie-Hellman

- Server certificate contains DH parameters (group, generator g) and static DH value g^x.
- Client chooses y, computes g^y and sends to server.

$$\text{pre\_master\_secret} = g^{xy}.$$

Anonymous Diffie-Hellman

- Each side sends Diffie-Hellman values in group chosen by server, but no authentication of these values.
- Vulnerable to man-in-middle attacks.

FF-DH-based Cipher Suites for TLS

- Originally, only finite-field DH was available in TLS; ECC came later.
- Recall: server chooses and sends parameters (p,g,g^x).
- Parameters are actually under-specified: it is hard for client to verify that:
  - p is prime.
  - g has large prime order dividing $p-1$.
  - g^x is indeed a power of g, and not in some other subgroup.
- Most implementations perform only rudimentary checks.
- Issues are meliorated to some extent by use of safe-primes ($p = 2q+1$ with q prime), but also the source of some attacks, e.g. Lim-Lee small sub-group attacks.
- See Valenta *et al*. (NDSS 2017) for more details.

ECC-based Cipher Suites for TLS

- ECC-based cipher suites for TLS were first defined in RFC 4492 (Blake-Wilson *et al.*, 2006).
- Negotiated via **TLS extensions** sent in ClientHello/ServerHello messages.
- 25 different curves + 3 point formats defined in RFC 4492, along with the ability to negotiate bespoke curve.
- Many curves taken from NIST and ANSI standards, e.g. NISTp256.
- Dozens of new cipher suites (56 with "ECDH(E)", 24 with "ECDSA") including "SHOULD support" for:
  - TLS_ECDH_ECDSA_WITH_3DES_EDE_CBC_SHA
  - TLS_ECDH_ECDSA_WITH_AES_128_CBC_SHA
  - TLS_ECDHE_RSA_WITH_3DES_EDE_CBC_SHA
  - TLS_ECDHE_RSA_WITH_AES_128_CBC_SHA

TLS Handshake Protocol – Key Establishment Notes

- Typical ClientHello offers many different cipher suites, choice of which to use is made by server.
- ClientHello and ServerHello also contain 32-byte nonces (28-byte random values + 4-byte time encoding).
- These are signed by the server in DH-based cipher suites, and involved in key derivation.
- Important for security – informally, preventing session replay attacks forcing reuse of session keys.

TLS Handshake Protocol – Key Establishment Notes

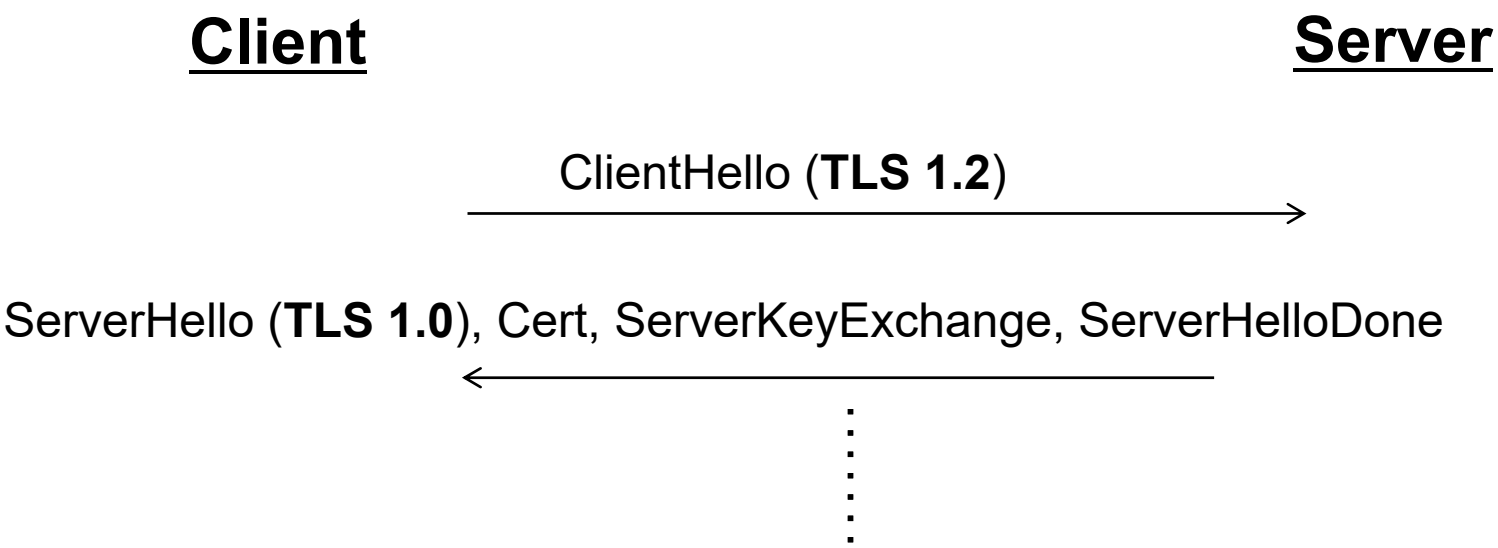
- ClientHello also offers SSL/TLS version number; server replies with its choice.

  Semantics: client: I support up to version x;

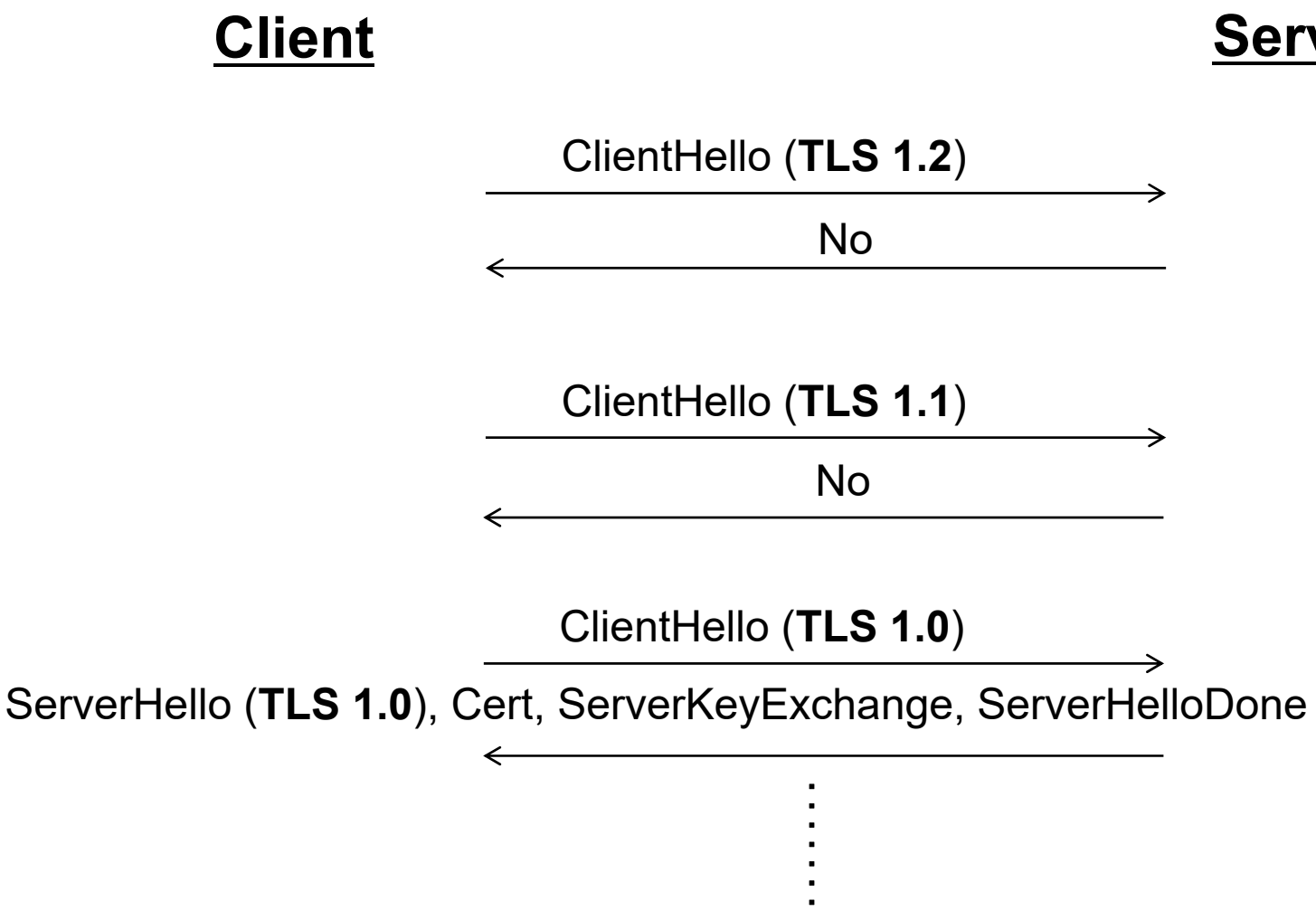
  server: I will use version y ≤ x.

  - Legacy servers do not implement this correctly, simply failing if they don't support version x.
  - Typical client behaviour: try again with lower version in a fresh handshake, with no memory of offers in previous handshakes carried over.
  - Security consequence: an active MITM can force client/server to roll back to **lowest** SSL/TLS version they are both willing to use!
  - POODLE attack exploits this to roll back to SSL3 and then perform Moeller attack on SSLv3 padding (see later).
  - The problem has been reanimated with the coming of TLS 1.3.

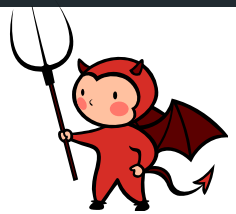
TLS Handshake Protocol Version Negotiation – Ideal World



TLS Handshake Protocol Version Negotiation – Real World (Version Intolerance)



TLS Handshake Protocol Version Negotiation – Real World Attack



Client **Server**

ClientHello (**1.2**)

No

ClientHello (**1.1**)

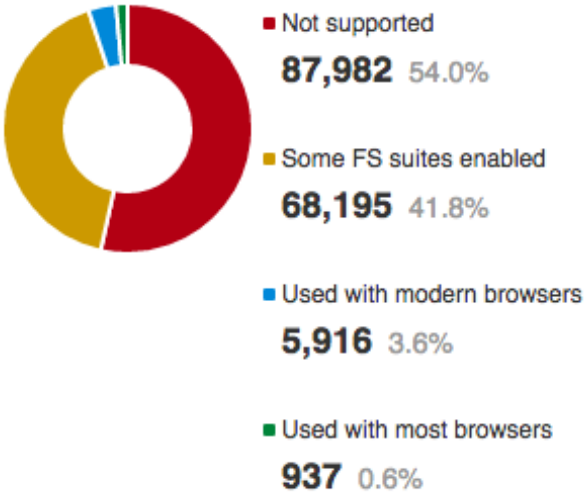
No

ClientHello (**1.0**) ClientHello (**1.0**)

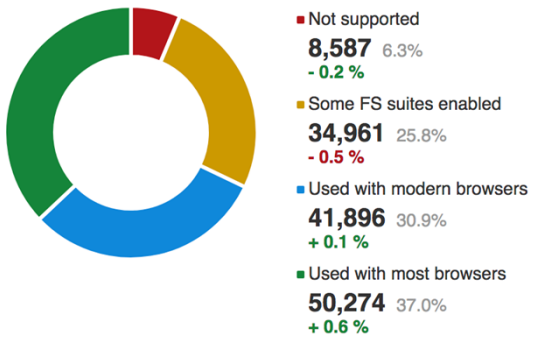
ServerHello (**1.0**), Cert, ServerKeyExchange, ServerHelloDone

TLS Handshake Protocol – Forward Security?

- An attacker who learns the RSA private key can decrypt old sessions and passively eavesdrop on all future RSA-based sessions!
- A well-known issue (lack of forward security), but given prominence by the Snowden revelations.
- This and performance benefits has driven an increased usage of forward-secure, Diffie-Hellman-based cipher suites over the last few years.



SSL pulse, Oct. 2013



SSL pulse, January 2018

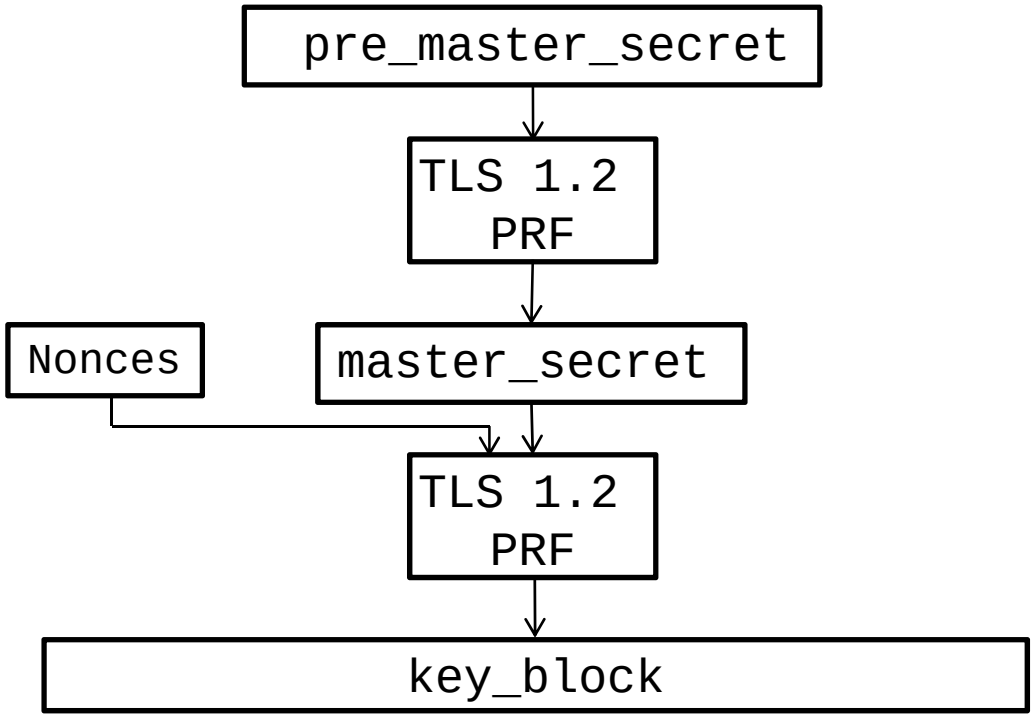
TLS Handshake Protocol – Reliance on Randomness

- An attacker who can predict a client's choice of pms or client/server DH private value can passively eavesdrop on all sessions!
  - And nonces in Hello messages may already leak information about state of client or server PRNG.
  - Hence backdoored PRNGs present a serious risk to TLS security: they may allow recovery of future PRNG output from observed output(s).
  - See Checkoway et al. (USENIX Security 2014) for extended analysis of exploitability of Dual EC PRNG in the TLS context.
- Relatedly, many server implementations default to using a "repeated ephemeral" value.
- cf. CVE-2016-0701:

  OpenSSL provides the option SSL_OP_SINGLE_DH_USE for ephemeral DH (DHE) in TLS. It is not on by default.

- Hence one-time server compromise would undermine the security of many client sessions.

TLS Key Derivation

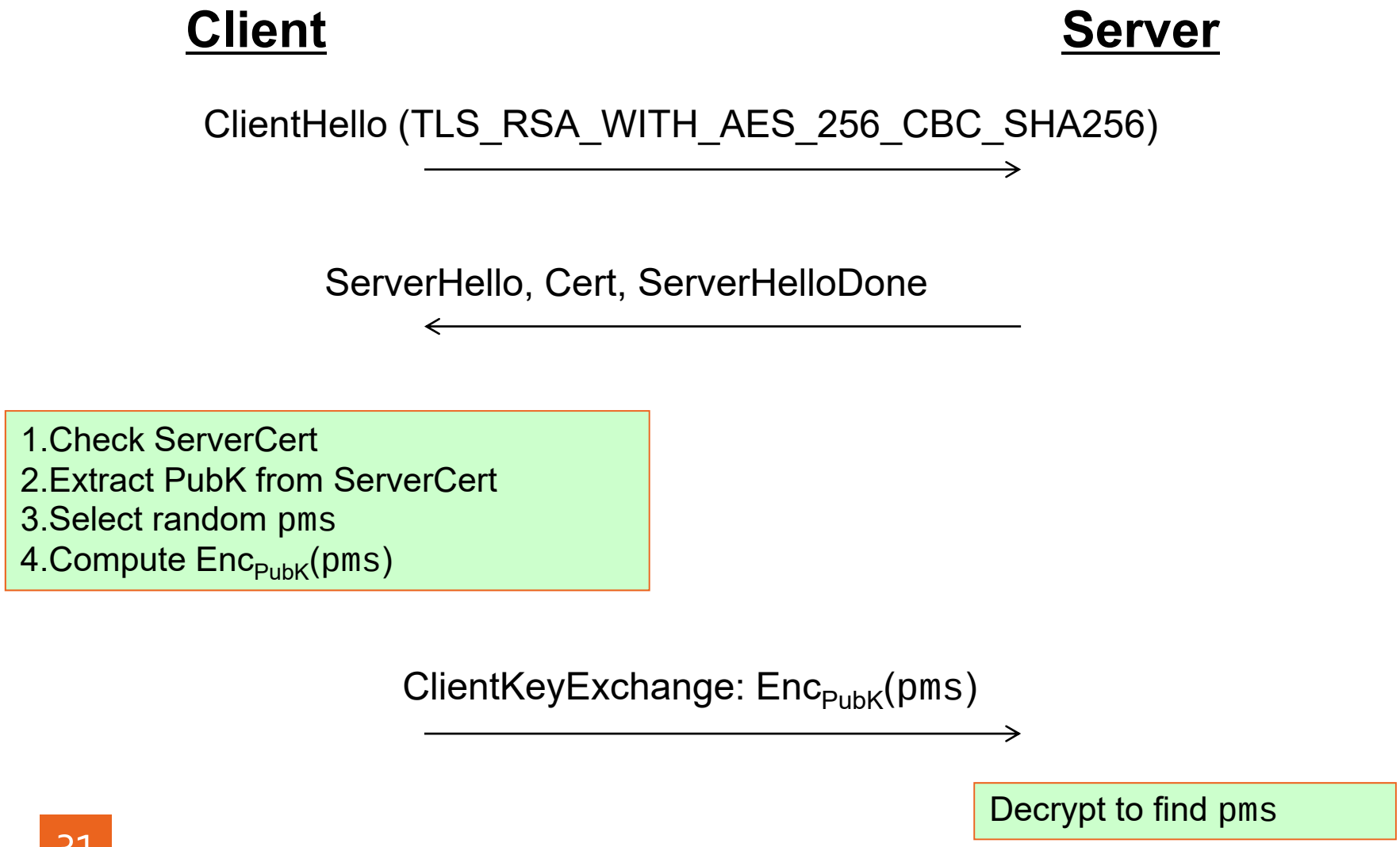


TLS Key Derivation

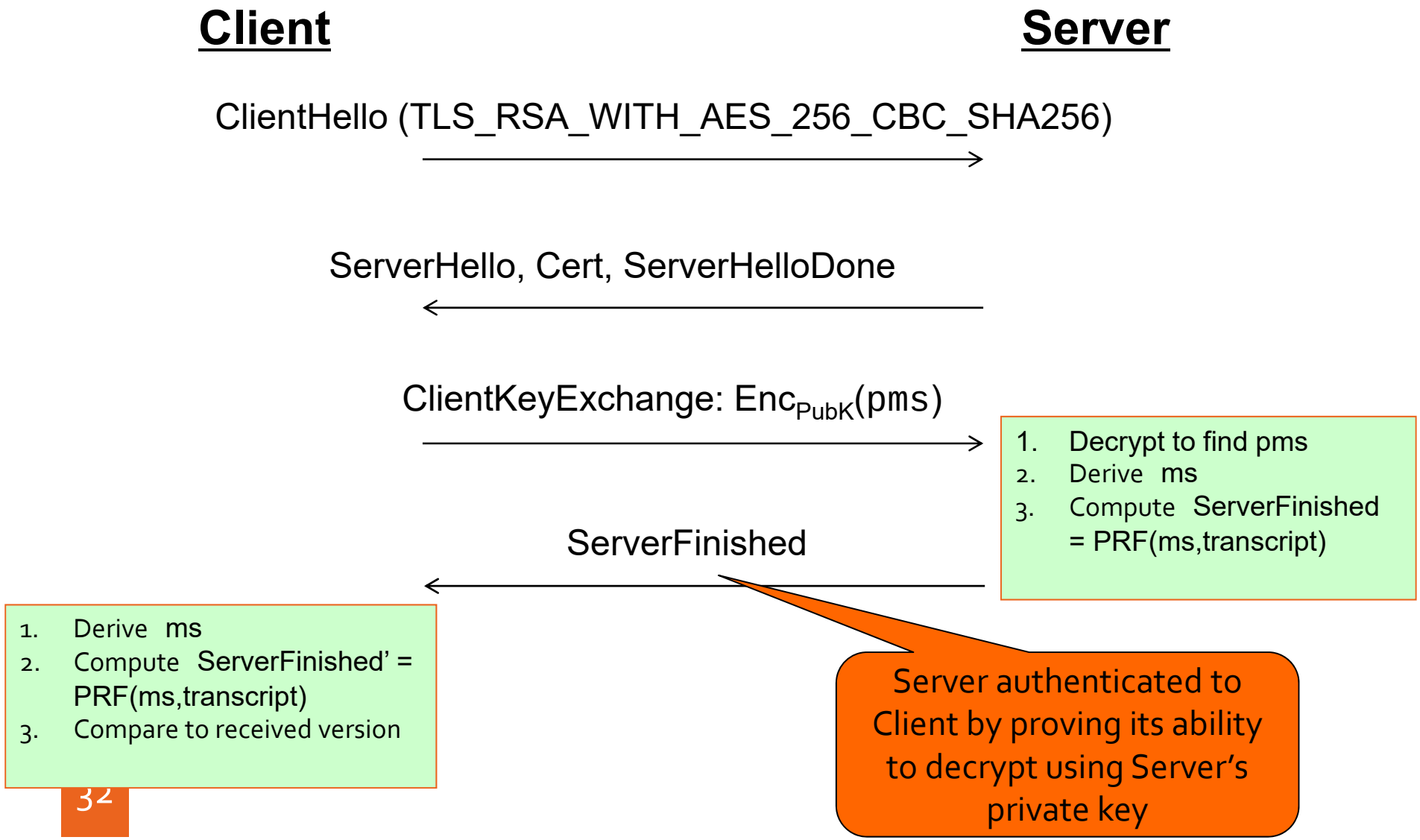
Keys used by MAC and encryption algorithms in the Record Protocol are derived from pre_master_secret (pms):

- Derive ms from pms using TLS Pseudo-Random Function (PRF).
- Default PRF for TLS1.2 is built by iterating HMAC-SHA256 in a specified way; earlier versions use *ad hoc* MD5/SHA-1 combination.
- Derive key_block from ms and client/server nonces exchanged during Handshake Protocol.
- Again using the TLS PRF in TLS1.2.
- Split up key_block into MAC keys, encryption keys and IVs for use in Record Protocol as needed.
- NB1: neither client nor server identity is involved in key derivation, nor any cipher suite context.
- NB2: splitting up of key_block into components depends on cipher suite.

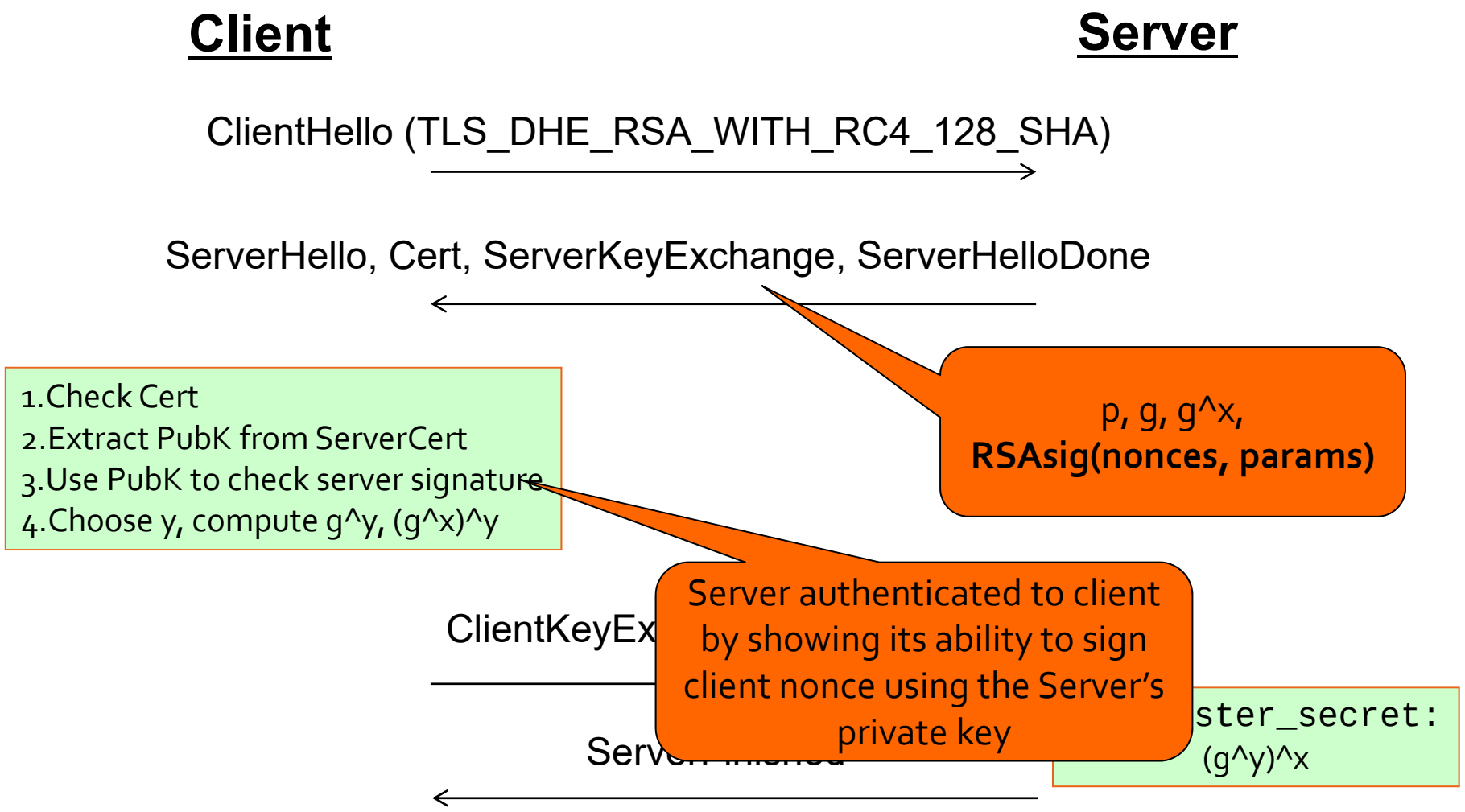
TLS Handshake Protocol – RSA-based Authentication?



TLS Handshake Protocol – RSA-based Authentication



TLS Handshake Protocol – Authentication for Ephemeral DH-based Key Establishment



TLS Handshake Protocol – Authentication

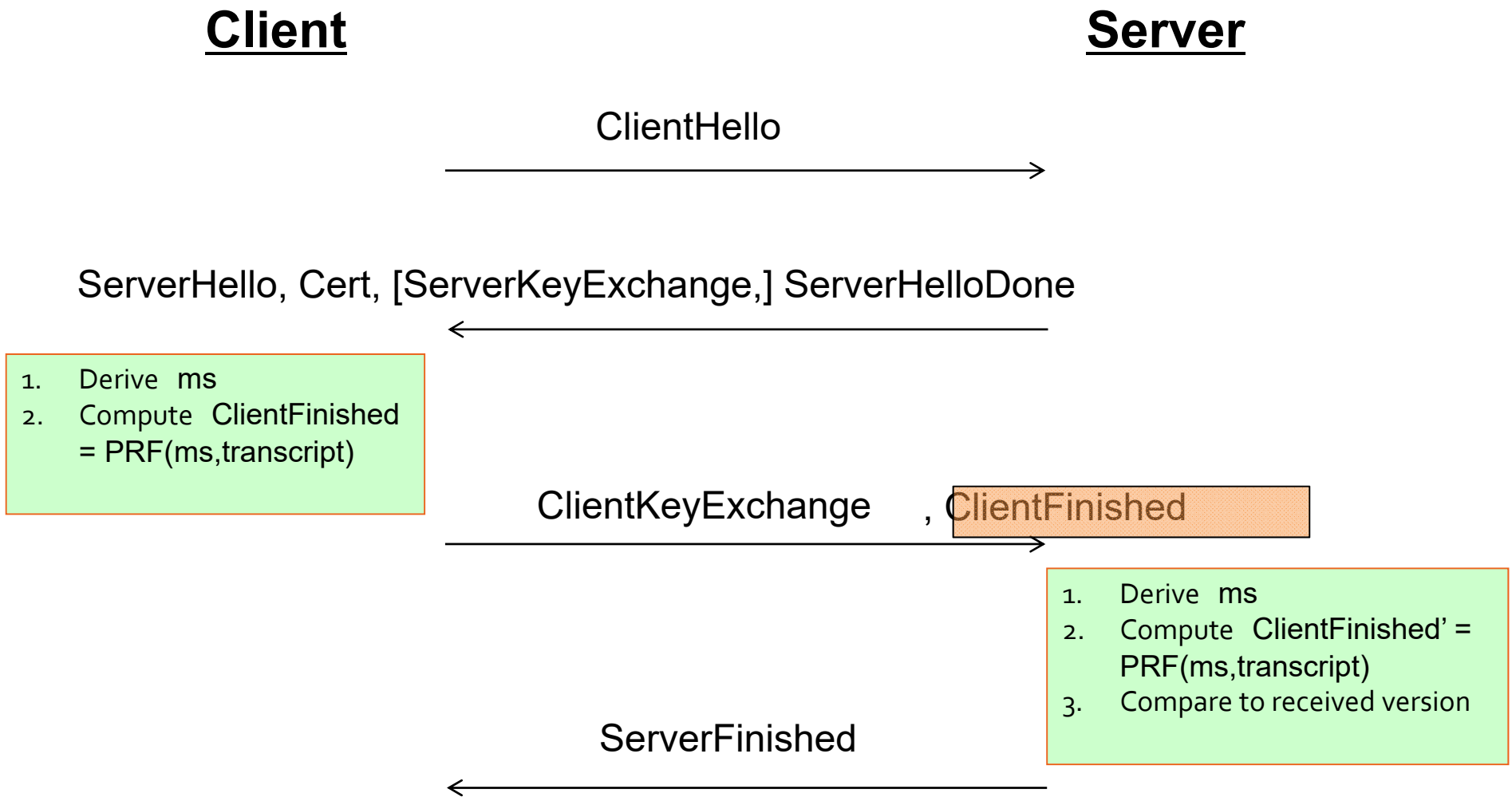
TLS supports several different entity authentication mechanisms for clients and servers.

Method used is negotiated along with key exchange method during the Handshake Protocol itself.

RSA: Ability of server to decrypt pms using its private key, derive ms from pms and then generate correct PRF value in ServerFinished message.

DHE/ECDHE: Ability of server to sign ClientNonce using its private key.

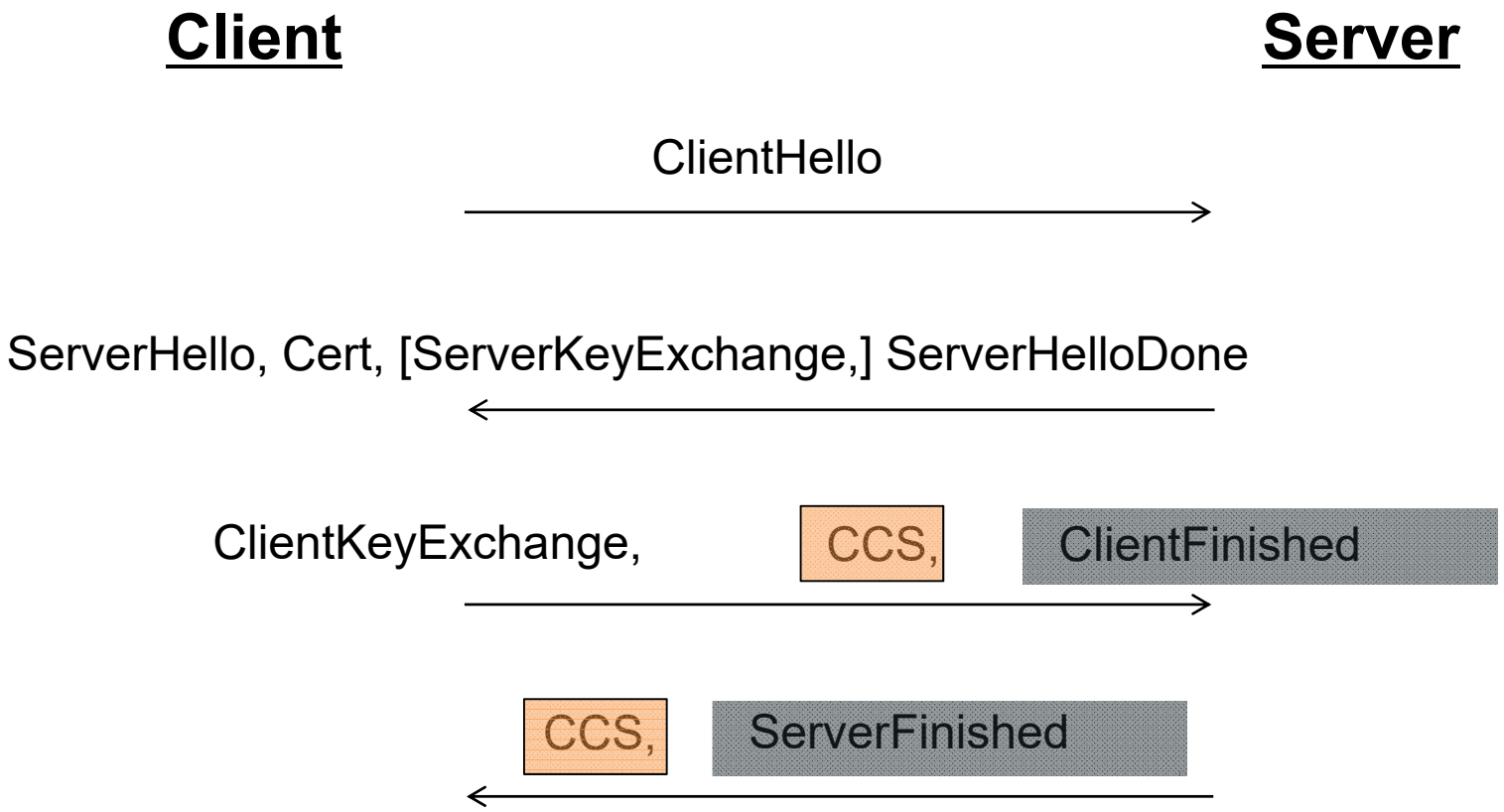
TLS Handshake Protocol – ClientFinished



TLS Handshake Protocol – Finished Messages

- TLS Finished messages enable each side to check that both views of the Handshake Protocol are the same.
- Computed as PRF(ms,transcript) where transcript = sender's view of all protocol messages sent and received up to *this* point.
- Compared by recipient to expected value; protocol aborts if mismatch is observed.
- Designed to prevent version rollback and cipher suite downgrade attacks.
  - Attacker attempts to manipulate client/server view of cipher suite(s) accpeted/offered, or of version offered/accepted.
  - Ineffective if attacker can compute ms during protocol run.

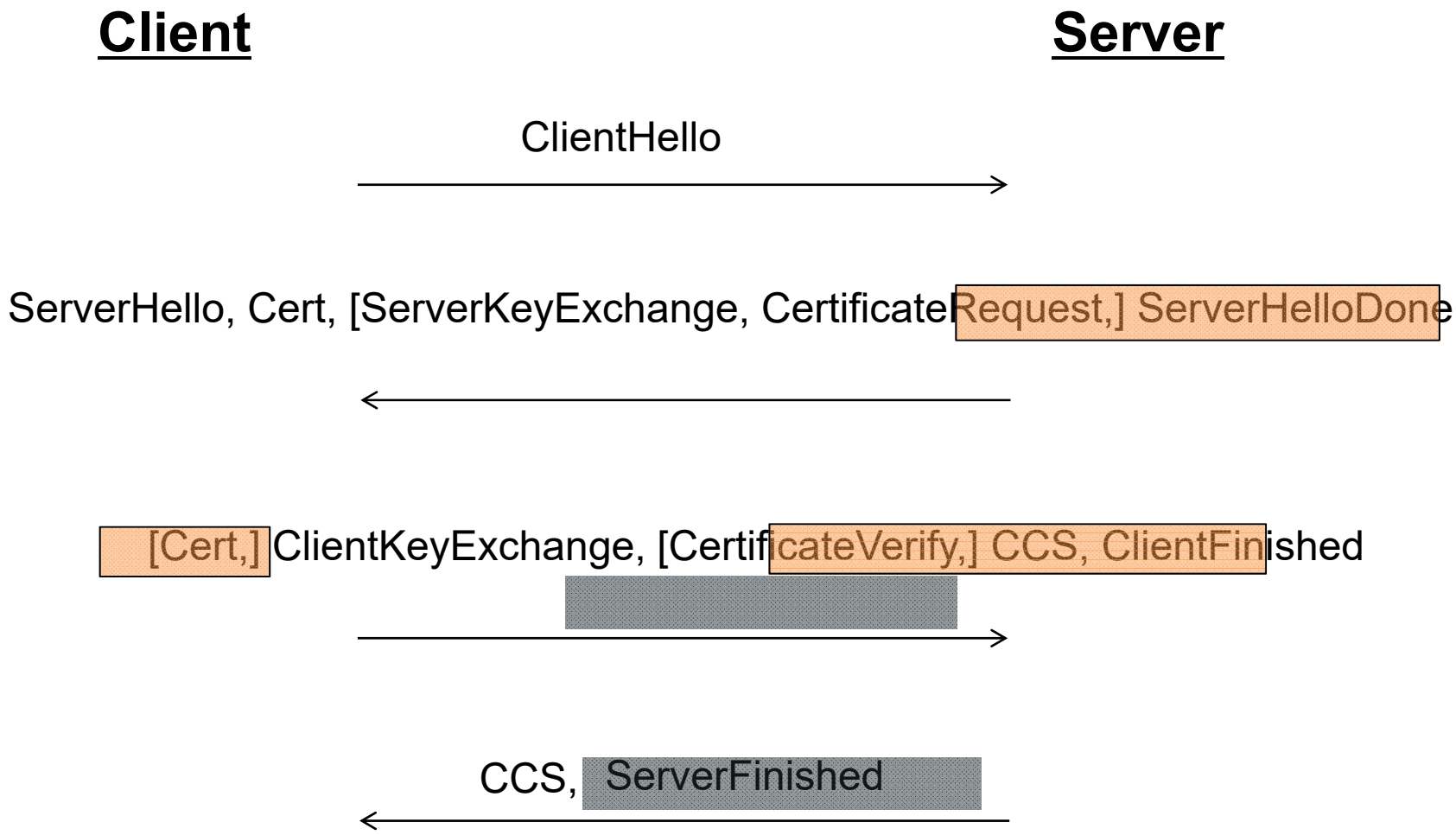
TLS Handshake Protocol – ChangeCipherSpec



TLS Handshake Protocol – CCS Messages

- ChangeCipherSpec messages enable parties to inform each other that they are switching to the recently agreed keys in the Record Protocol.
- Here, this means that all subsequent messages are protected using the agreed cipher suite (e.g. AES_256_CBC_SHA256).
- Not part of the Handshake Protocol, so not included in transcripts when computing Finished messages.

TLS Handshake Protocol – Client Authentication



TLS Handshake Protocol – Client Authentication

- Client authentication is optional and rarely used in the web setting.
- Server requests client's certificate in its Hello message.
- Client responds with:
  - Cert: client's certificate (chain).
  - CertificateVerify: signature on protocol transcript up to this point.
  - Notice the misnomers!

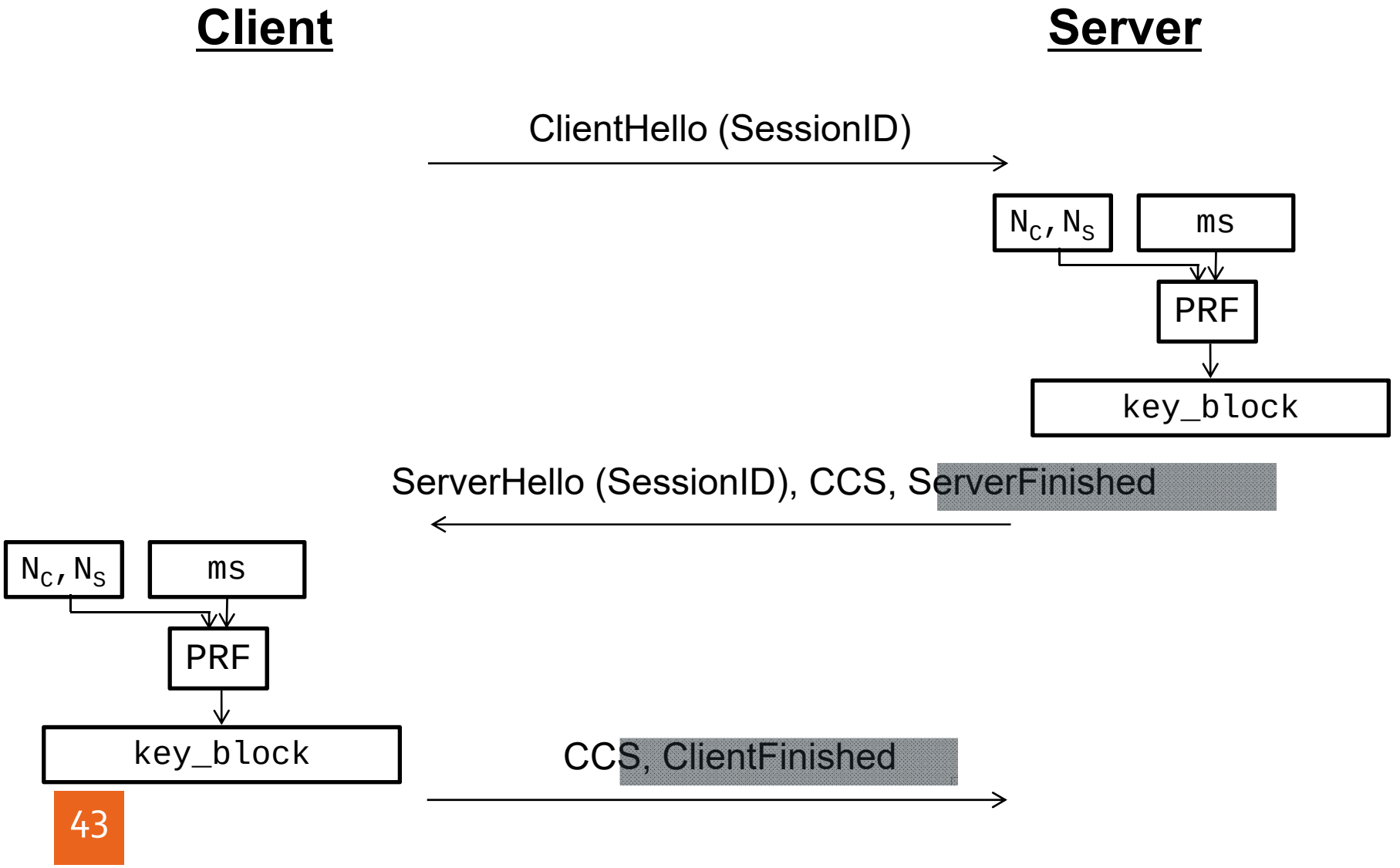
TLS Handshake Protocol – Renegotiation

- **Renegotiation** allows re-keying and change of cipher suite during a session.
  - For example, to force strong client-side authentication before access to a particular resource on the server is allowed.
  - Or to publicly negotiate a weak cipher suite and then upgrade to a stronger one over an encrypted channel.
- Initiated by client sending ClientHello or server sending ServerHelloRequest.
  - Followed by full run of Handshake Protocol.
  - Protocol is run over the existing Record Protocol, so receives its protection.

TLS Handshake Protocol – Session Resumption

- **Session resumption** allows authentication and shared secrets to be reused across multiple, parallel *connections* in a single session.
- E.g., allows fetching multiple resources from same website without re-doing full, expensive Handshake Protocol.
- Client and Server quote existing SessionID and exchange fresh nonces.
- Also enabled by use of session ticket mechanism, RFC 5077.
  - Uses a TLS extension to signal/transmit a cryptographic "blob" from server to client, carrying session state.

TLS Handshake Protocol – Session Resumption



TLS Sessions and Connections

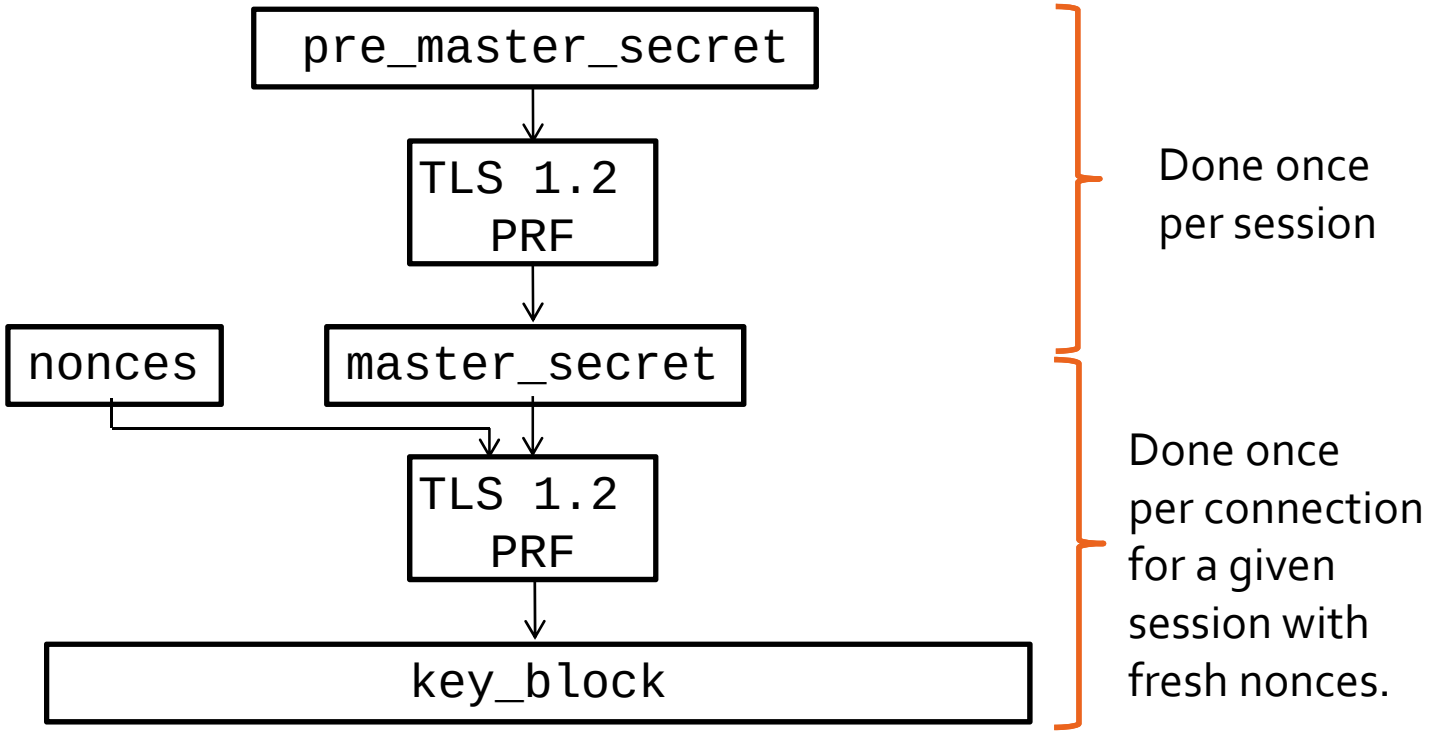
Session concept:

- Sessions are created by the Handshake Protocol.
- Session state defined by session ID and set of cryptographic parameters (encryption and hash algorithm, master secret, certificates) negotiated in Handshake Protocol.
- Each session can carry multiple **parallel** *connections*.

Connection concept:

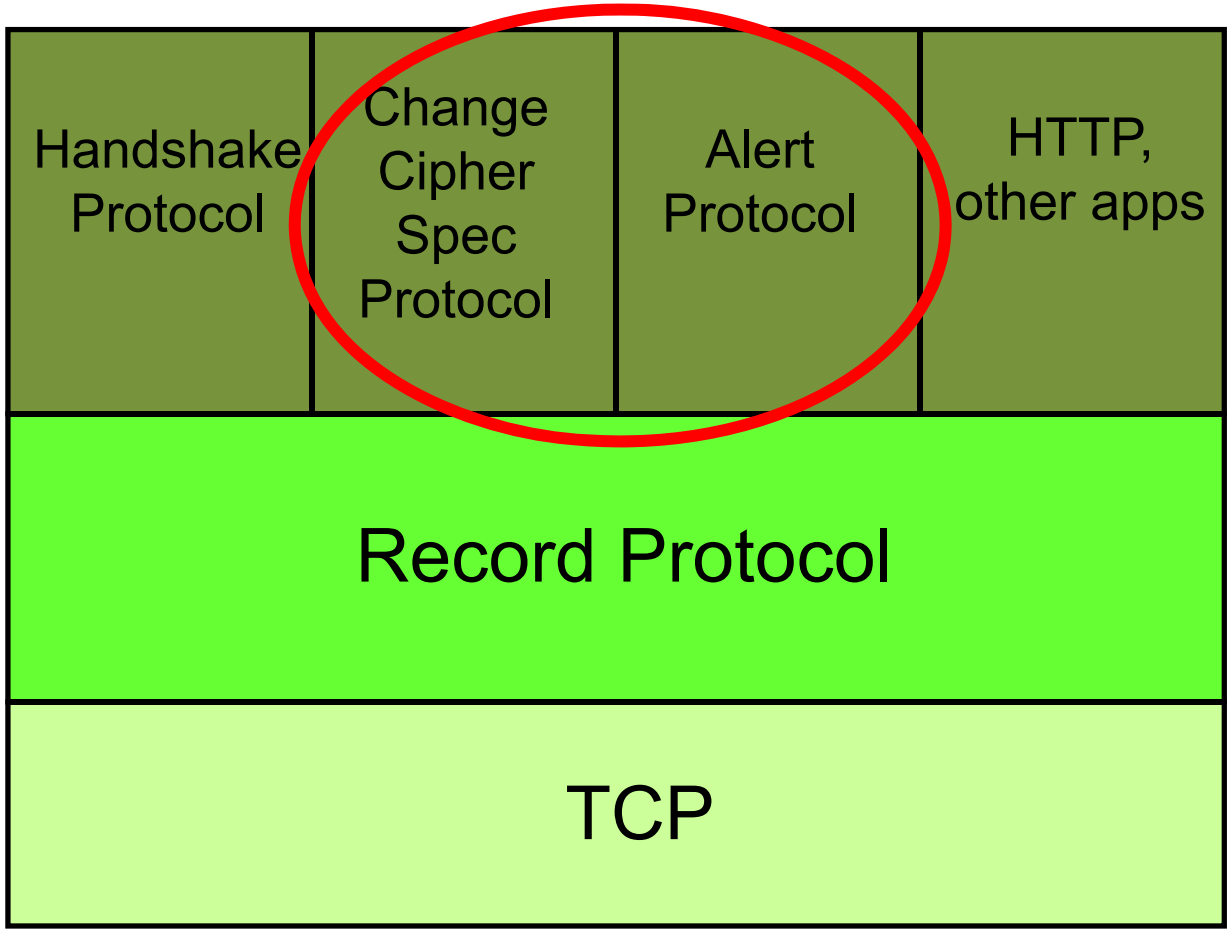
- Keys for multiple connections are derived from a single ms created during one run of the full Handshake Protocol.
- Session resumption Handshake Protocol runs exchange new nonces.
- These nonces are combined with existing ms to derive keys for each new connection.
- Avoids repeated use of expensive Handshake Protocol.
- Each TLS connection corresponds to a different TCP connection.

TLS Key Derivation and Sessions/Connections




Other TLS Protocols

TLS Protocol Architecture



Other TLS Protocols

Alert protocol.

- Management of SSL/TLS connections and sessions, error messages.
- Fatal errors and warnings.
- Defined actions to ensure clean session termination by both client and server.

Change cipher spec protocol.

- Technically not part of Handshake Protocol.
- Used to indicate that entity is changing to recently agreed cipher suite.

Both protocols run over Record Protocol (so are peers of Handshake Protocol).

TLS Extensions

Many *extensions* to TLS exist.

Allows extended capabilities and security features.

Examples:

- Renegotiation Indicator Extension (RIE), RFC 5746.
- Application layer protocol negotiation (ALPN), RFC 7301.
- Authorization Extension, RFC 5878.
- Server Name Indication, Maximum Fragment Length Negotiation, Truncated HMAC, etc, RFC 6066.

TLS Complexity

- Recall simplistic view of TLS:
  - Handshake Protocol followed by Record Protocol.
- Reality is much more complex:
  - Initial Handshake Protocol over Record Protocol with no keys.
  - Change Cipher Spec. Protocol message, switch on new keys.
  - Completion of Handshake via exchange of `Finished` messages, now running over keyed Record Protocol.
  - Followed by arbitrary sequences of Session Resumption and Renegotiation runs.
  - Most of this activity is hidden from applications.
- This complexity has turned out to have negative security consequences.


TLS Handshake Protocol Security Issues

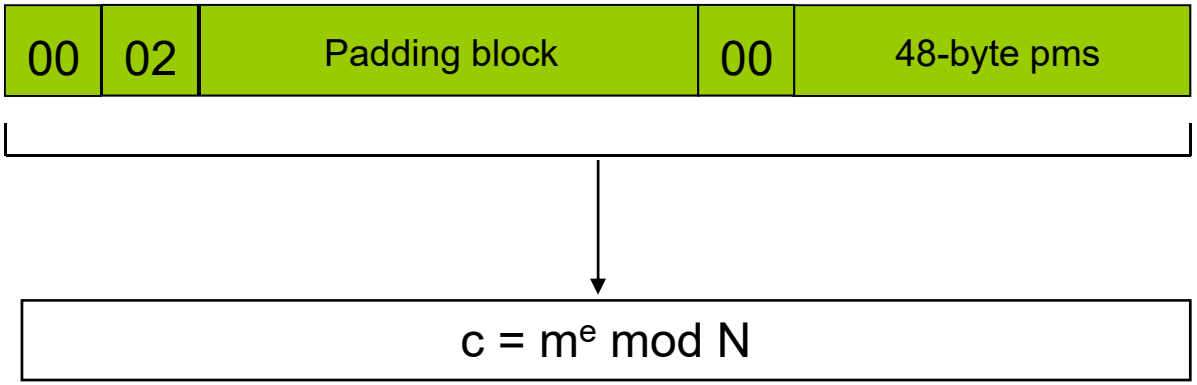
Some TLS Handshake Protocol Security Issues

- Bleichenbacher attack (1998) on PKCS#1 v1.5 padding used for RSA encryption in Handshake protocol.
  - Patched by making it hard to distinguish error messages, but attack rebooted in various ways over the years.
  - Including DROWN attack in 2016, exploiting public key reuse between SSLv2 and other versions of SSL/TLS, and extensive legacy support for SSLv2 in servers.
- Attacks exploiting continued support for weak "export-grade" cipher suites: FREAK and LOGJAM (2015).
- Attacks exploiting renegotiation and resumption: renegotiation attack (2009), triple handshake attack (2014).
- Implementation flaws of various kinds.

Bleichenbacher's Attack

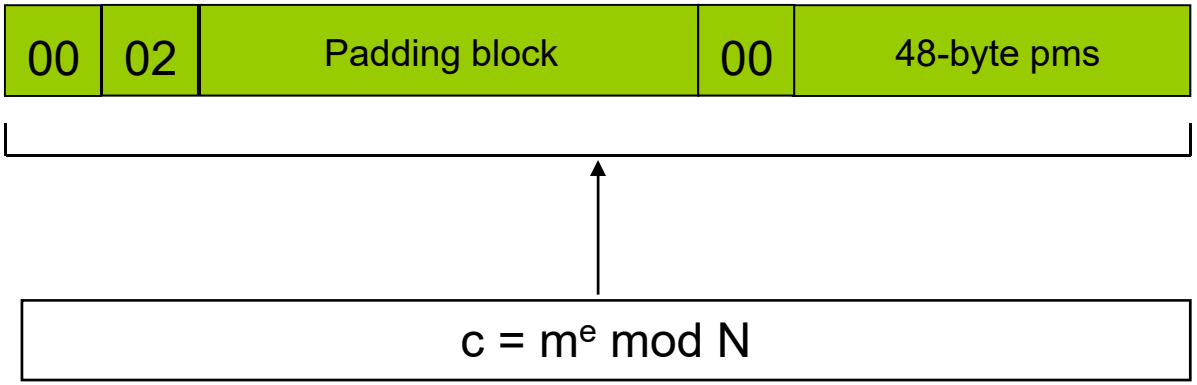
- We begin with Bleichenbacher's attack on RSA encryption used in TLS (C'98).
- This attack exploits the fact that RSA encryption scheme used in TLS (PKCS#1 v1.5) is **not** CCA secure.
- It recovers the TLS pre_master_secret (pms) for a target session using roughly 2^{20} interactions with server.

PKCS#1 v1.5, block type 2



- Plaintext must begin with "00 02" bytes.
- Padding block consists of *at least* 8 non-zero bytes.
- Should be terminated by "00" byte.
- Last 48 bytes are used as pms.
  - Additional complication: most significant two bytes are set to client TLS version.

PKCS#1 v1.5, block type 2



Think about sanity checking *m* after applying RSA decryption operation:

- Check for "00 02"?
- Check for at least 8 non-zero padding bytes or just *some* non-zero bytes?
- Check for a 00-byte? Or just extract last 48 bytes?
- Demand 00-byte to be in exactly the right position?
- Check for TLS version number?

Bleichenbacher's Attack

- Exact decryption processing for RSA is not specified in the RFCs.
- Different implementations exhibit different behaviours.
- To simplify matters: suppose that we have an oracle that on input *c* outputs whether $x := c^d \bmod N$ begins with byte pattern "00 02".
- If oracle output is "yes", then we have an inequality:

$$2B \leq x \bmod N < 3B$$

where $B = 2^{8(k-2)}$ and k is the number of bytes in modulus N.

Bleichenbacher's Attack

- Suppose attacker records c*, the RSA ciphertext encrypting the unknown pms for a target session.
- Attacker calls the "00 02" oracle on many, carefully selected inputs of the form $s^e c^* \bmod N$.
- Each "yes" output gives an inequality of the form:

$$2B \le sx \bmod N < 3B$$

  where s is known and x encodes pms.

- By analysing many responses from the oracle, the attacker can eventually reconstruct x and thence pms.
- Roughly 2^{20} oracle queries are needed.

Bleichenbacher's Attack

In the TLS context:

The required "00 02" oracle was obtained using error messages arising from server processing of attacker-generated ClientKeyExchange messages.

Countermeasures?

- Switch to using CCA-secure variant of RSA encryption, e.g. RSA-OAEP (cannot create "related" ciphertexts that are valid).
- Add protocol-specific countermeasures.

Bleichenbacher and TLS1.0 (1999)

TLS 1.0 was published in RFC 2246, Jan 1999, shortly after adoption of RSA-OAEP into PKCS#1v2.0.

TLS 1.0 still uses PKCS#1v1.5, despite Bleichenbacher's attack:

> *The best way to avoid vulnerability to this attack is to treat incorrectly formatted messages in a manner indistinguishable from correctly formatted RSA blocks.* ***Thus, when it receives an incorrectly formatted RSA block, a server should generate a random 48-byte value and proceed using it as the premaster secret****. Thus, the server will act identically whether the received RSA block is correctly encoded or not.*

TLS 1.1, RFC 4346 (2006)

[PKCS1B] defines a newer version of PKCS#1 encoding that is more secure against the Bleichenbacher attack. ***However, for maximal compatibility with TLS 1.0, TLS 1.1 retains the original encoding****. No variants of the Bleichenbacher attack are known to exist provided that the above recommendations are followed.*

Over-optimistic: several implementations still get it wrong, and there's now a long literature of Bleichenbacher-style attacks against RSA implementations (not just in TLS):

- Bardou et al. (Crypto 2012), Jager et al. (Esorics 2012), DROWN (Aviram et al., USENIX 2016), ROBOT (Böck *et al.*, 2017).

DROWN Attack (Aviram et al., USENIX'16)

Attack scenario:

- Server supports SSLv2 and uses the same RSA key for SSLv2 and later versions of SSL/TLS
  - Surprisingly large number of servers: circa 8% of Alexa top 150k servers in July 2016 (SSL pulse)
  - Most servers don't provide a facility to provide different key for different SSL/TLS versions anyway.
- Client has absolutely no intention to use SSLv2.

DROWN Attack (Aviram et al., USENIX'16)

ClientHello(TLS 1.2, TLS_RSA…)

ServerHello, Cert, ServerHelloDone

ClientKeyExchange, CCS, ClientFinished

RSA ciphertext c*

ClientHello(**SSL2_RC4_128_EXPORT40_WITH_MD5**)

ServerHello, Cert, ServerHelloDone

ClientKeyExchange: $\mathbf{s^e c^*}$

ServerFinished

Encrypted under 40-bit key!

DROWN Attack (Aviram et al., USENIX'16)

- Standard Bleichenbacher countermeasure: if RSA decryption of c* fails, then choose a random master secret K and carry on with the protocol.
- Send $s^e c^*$ twice in two consecutive SSLv2 handshakes:
  - If $s^e c^*$ is invalid, we get two ServerVerify messages encrypted under two different keys.
  - If $s^e c^*$ is valid, then we get two ServerVerify messages encrypted under the same key.
- But the encryption key is "only" 40-bits in size, and the plaintext is partly known.
- Perform two 40-bit key searches and compare keys to find out if $s^e c^*$ was valid or invalid.
- This provides an expensive oracle for carrying out Bleichenbacher's attack.

DROWN Attack (Aviram et al., USENIX'16)

- Roughly 10,000 SSLv2 handshakes are needed for the attack, and the attack works for (roughly) 1 in every 1000 TLS handshakes.
- Support for legacy 40-bit algorithms in SSLv2 + bugs in OpenSSL implementation make it feasible to extract plaintext underlying c*.
- Cost is 2^{50} trial decryptions (without OpenSSL bugs), but under a minute for "special DROWN" (with OpenSSL bugs).

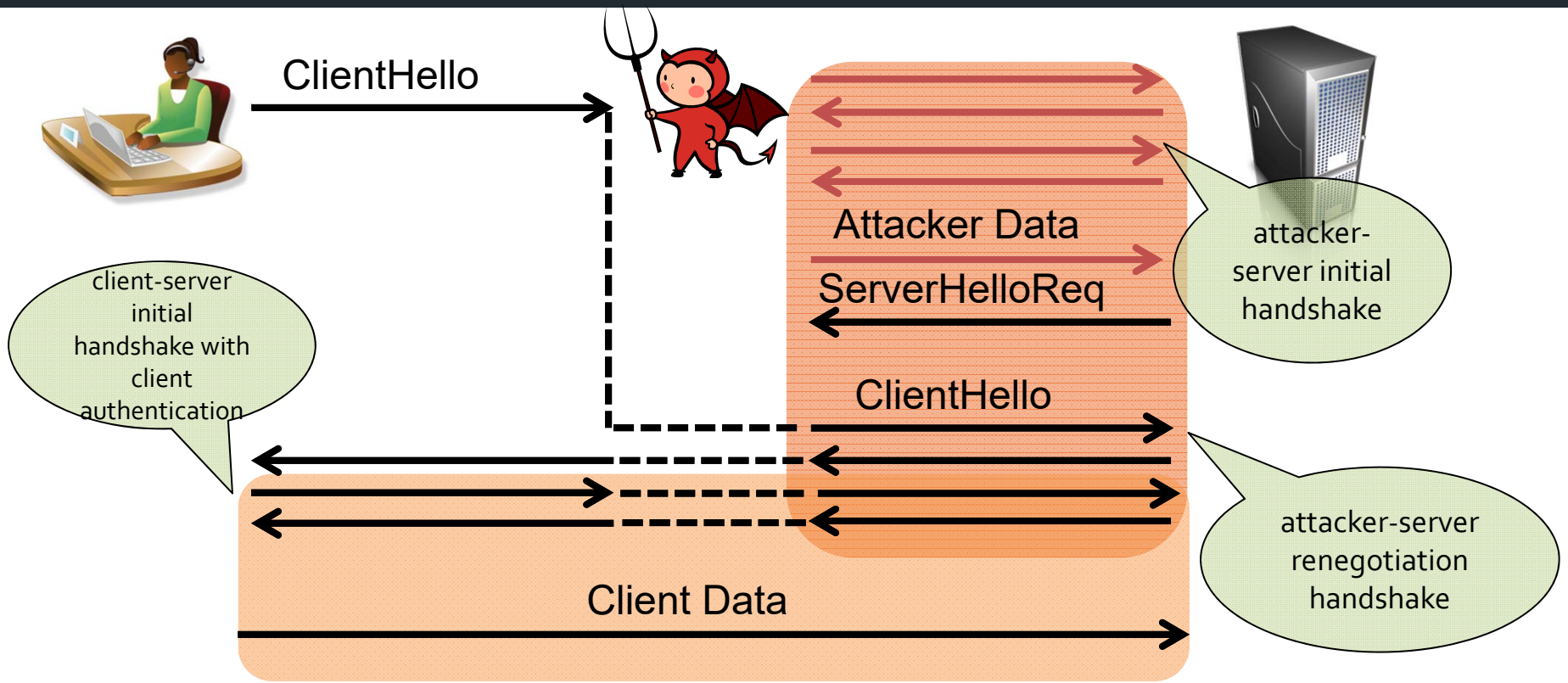
- **This is a cross-version (or cross-cipher suite) attack, made possible because of support for old versions/algorithms and key re-use across versions.**

More Recent TLS Handshake Protocol Attacks

Up until 2009, the TLS Handshake Protocol survived relatively unscathed.

Notable exception: Bleichenbacher's attack on RSA encryption used in TLS as discussed above.

Renegotiation Attack (Ray and Dispensa, Rex, 2009)



Client view: single handshake, sends ClientData.

Server view: two handshakes, receives AttackerData||ClientData from authenticated client.

Overall effect: attacker injects AttackerData as if from trusted source.

Renegotiation

- Renegotiation attack due to Ray and Dispensa, also Rex (2009).
- Server treats data as coming from either side of client authentication as being a single unit from an authenticated source.
- TLS specification does not really say how to handle this situation.
  - Flush buffer of received fragments upon renegotiation?
  - Signal to application that authentication status has changed?
  - Highlights lack of API specification for TLS.
- Attack addressed via Renegotiation Indication Extension (RIE), RFC 5746.
  - Include and verify information about previous handshakes in any renegotiation.
  - Could also disable renegotiation on server.

Triple Handshake Attack (Bhargavan et al, IEEE S&P'14)

- Triple Handshake attack: renegotiation attack rebooted.
- Complex attack leveraging lack of identities in key derivation + resumption + renegotiation.
    - Even first step in the attack (UKS attack) breaks certain authentication protocols relying on TLS.
- Attack highlights that RIE fix for renegotiation attack is not robust in the context of the full TLS Handshake Protocol.
    - Renegotiation status gets lost across resumptions.

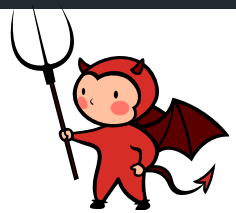
Cross-cipher Suite Attacks

- Recall server signature format in ServerKeyExchange:
  sig(nonces, params)
- Format of params depends on type of key exchange: mod *p* DH parameters or ECDH parameters.
- But *type* of parameters is not itself signed.
- Instead, it's inferred by client from the cipher suite, for which agreement is only verified later, via Finished messages.
- Leads to a theoretical attack due to Mavrogiannopoulos *et al.* (CCS'12).
  - Attacker switches cipher suite – ECDH for FFDH, or vice-versa.

Cross Cipher Suite Attack (Mavrogiannopoulos *et al.*, CCS'12)

ClientHello

TLS_DHE_RSA…



ClientHello'

TLS_ECDHE_RSA…

Server accepts TLS_ECDHE_RSA...

ServerHello, Cert, ServerKeyExchange

Changed by MITM back to TLS_DHE_RSA...

ServerHello', Cert, ServerKeyExchange

Contains server's ECDHE parameters and RSA signature on nonces and parameters*

ECDHE params are interpreted as being DHE params; RSA signature is valid!

With probability 2^{-40}, DHE params are parsed as having g = 0, 1 or -1!

*server needs to support "explicit prime curve" option, RFC 4492.

Cross Cipher Suite Attack (Mavrogiannopoulos et al, CCS 2012)

- Attack requires server to support "explicit prime curve" option (RFC 4492).
- Attack requires client to accept weak DH parameters (g = 0, 1 or -1).
    - Enabling MITM to compute pms and correct ServerFinished message to complete the handshake.
- Success rate can be boosted by repeatedly sending ClientHello message within TLS timeout on client (tens of seconds).
- Attack possible because server signature does not cover type of cipher suite, nor TLS extensions specifying use of ECC.

FREAK and LOGJAM Attacks

EXPORT cipher suites:

0x000003	TLS_RSA_EXPORT_WITH_RC4_40_MD5
0x000006	TLS_RSA_EXPORT_WITH_RC2_CBC_40_MD5
0x000008	TLS_RSA_EXPORT_WITH_DES40_CBC_SHA
0x00000B	TLS_DH_DSS_EXPORT_WITH_DES40_CBC_SHA
0x00000E	TLS_DH_RSA_EXPORT_WITH_DES40_CBC_SHA
0x000011	TLS_DHE_DSS_EXPORT_WITH_DES40_CBC_SHA
0x000014	TLS_DHE_RSA_EXPORT_WITH_DES40_CBC_SHA

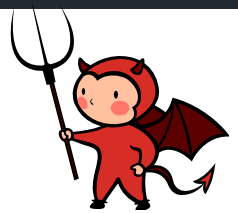
(and more)

- Introduced in the 90s in the era of export control.
- Maximum 512-bit RSA keys and 512-bit primes for DH/DHE.
- Repurpose ServerKeyExchange message to transport "ephemeral" RSA/DH/DHE keys.
- Until recently, still supported by around 25% of servers…

FREAK Attack (Beurdouche *et al.*, IEEE S&P'15)

ClientHello

TLS_RSA…



ClientHello'

TLS_RSA_EXPORT…

Server accepts TLS_RSA_EXPORT...

ServerHello, Cert, ServerKeyExchange

Changed by MITM back to TLS_RSA ...

ServerHello', Cert, ServerKeyExchange

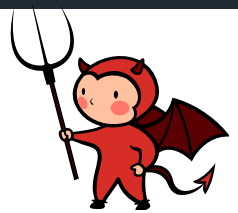
Contains server's 512-bit RSA public key and RSA signature on nonces and parameters

Buggy client processes this and accepts 512-bit RSA key for pms transport

FREAK Attack (Beurdouche *et al.*, IEEE S&P'15)

ClientHello

TLS_RSA…



ClientHello'

TLS_RSA_EXPORT…

ServerHello, Cert,
ServerKeyExchange

ServerHello', Cert,
ServerKeyExchange

Attacker pre-factors 512-bit RSA key, and can now decrypt to get pms.

ClientKeyExchange,
CCS, ClientFinished

CCS,
ServerFinished

Attacker succeeds in impersonating server.

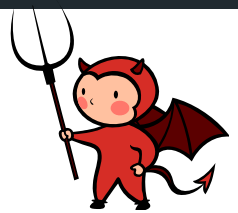
FREAK Attack (Beurdouche *et al*., IEEE S&P'15)

- Attack relies on buggy clients accepting ServerKeyExchange containing 512-bit RSA key when no such message was expected.
  - Many clients were vulnerable (https://www.smacktls.com/).
- Export RSA keys are meant to be ephemeral, but hard to generate RSA moduli in practice, so they were made long-lived.
- Cost of factoring 512-bit modulus: $50 on Amazon EC2.
- Attack arises because of common code paths in implementations, coupled with state machine failures.
  - Explored in-depth in Berdouche *et al*. paper.

LOGJAM Attack (Adrian *et al.*, CCS'15)

ClientHello

TLS_DHE_RSA…



ClientHello'

TLS_DHE_RSA_EXPOR

ServerHello, Cert,
ServerKeyExchange

Attacker uses x and g^y to compute pms

ServerHello', Cert,
ServerKeyExchange

ClientKeyExchange (g^y),
CCS, ClientFinished

CCS,
ServerFinished

Contains server's 512-bit **DHE parameters** and RSA signature on nonces and parameters

Attacker solves DLP for g, g^x to compute server's private value x .

Attacker succeeds in impersonating server.

LOGJAM Attack (Adrian *et al.*, CCS'15)

- LOGJAM = Cross-cipher suite + FREAK.
  - Active attacker changes TLS_DHE_RSA… to TLS_DHE_RSA_EXPORT…
  - Server responds with weak DH parameters signed under its RSA key.
  - Client accepts these (signature does not include cipher suite details).
  - Attacker solves 512-bit DLP before client times out.
  - Attacker can then create correct ServerFinished message to impersonate server.
- Difficult to perform in practice, but not impossible for three-letter agency.
  - Servers use small numer of common primes *p.*
  - Precomputation allows each 512-bit DLP to be solved in around 90s.


Implementation Vulnerabilities

Heartbleed

- Buffer over-read vulnerability in OpenSSL implementation of DTLS Heartbeat protocol.
- High severity: remote recovery of chunks of server memory, including server private keys, private user data, etc.
- 85%+ of SSL/TLS servers rely on OpenSSL.
- Practical demonstrations of threat (e.g. Mumsnet).
- Messy disclosure in early April 2014.
- A good logo!

Certificate Processing Bugs

Many problems have been discovered in code for certificate processing.

- Fahl et al. (CCS 2012)
- Georgiev et al. (CCS 2012)
- GnuTLS bug (CVE-2014-0092)
- Apple goto fail (CVE-2014-1266)
  - Affecting Apple iOS 6.x before 6.1.6 and 7.x before 7.0.6, Apple TV 6.x before 6.0.2, and Apple OS X 10.9.x before 10.9.2.

Apple goto fail

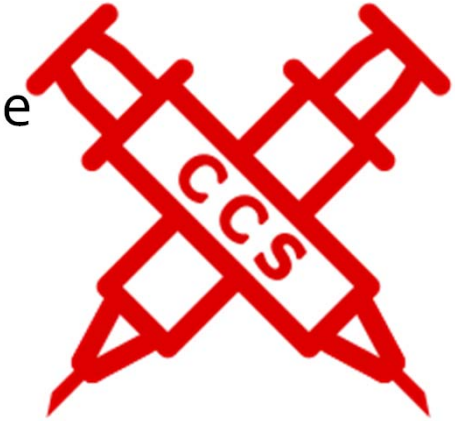
```
SSLVerifySignedServerKeyExchange(SSLContext *ctx, bool isRsa, SSLBuffer signedParams,
                    uint8_t *signature, UInt16 signatureLen)
{
        OSStatus        err;
        ...

        if ((err = SSLHashSHA1.update(&hashCtx, &serverRandom)) != 0)
                goto fail;
        if ((err = SSLHashSHA1.update(&hashCtx, &signedParams)) != 0)
                goto fail;
                goto fail;
        if ((err = SSLHashSHA1.final(&hashCtx, &hashOut)) != 0)
                goto fail;
        ...

fail:
        SSLFreeBuffer(&signedHashes);
        SSLFreeBuffer(&hashCtx);
        return err;
}
```

Causes all server signature processing on client to be bypassed!

Meaning that MITM attacker can trivially impersonate ***any*** TLS server!

CCS Mishandling Bug (CVE 2014-0224)

- OpenSSL implementation of TLS will accept ChangeCipherSpec message at any point in the TLS Handshake.
- So MITM attacker can inject it at point of his choosing.
- Result is that TLS key derivation is carried out with a zero-length master secret.
- Leading to predictable session keys.



Invalid Curve Attacks

- Implementations fail to check that received EC point is actually on specified curve.
- Leads to **invalid curve attack** on implementations reusing ephemeral values and/or ECDH cipher suites.
- Attacker (client) sends as its DH values points $P_i = (x_i, y_i)$ in EC groups of small, co-prime orders q_i.
- Server responds by computing sP, where s is long-term secret: relies on x-coordinate-only computation depending only on b in Weierstrass form.
- premastersecret is then just sP, one of q_i possible values.
- Attacker can learn s mod q_i by guessing value for ClientFinished and testing if server accepts.
- Attacker can reconstruct s using CRT.
- Original ideas in Biehl *et al.* (CRYPTO 2000) and Antipa *et al.* (PKC 2003).
- Shown to work in practice for TLS implementations by Jager *et al.* (ESORICS 2015).

Side Channel Attacks

- Use of public key primitives opens up many opportunities for side-channel attacks on implementations.
- Timing attacks on naïve (and not so naïve!) implementations of RSA, EC-DSA, DH, ECDH.
- A current focus is on **"Flush+Reload" Lowest Level Cache (LLC)** timing attacks.
- OpenSSL is by now quite well protected, but new attacks are still being discovered.
- Recent example: Pereida García and Brumley, USENIX 2017:
  - LLC attack on modular inversion code used in OpenSSL ECDSA, finding MSBs of ECDSA nonces, and thence ECDSA signing key via lattice attack.
- Other libraries are (probably) less-well protected.


TLS 1.3

TLS 1.3

- The TLS 1.3 specification is being developed in TLS Working Group of IETF.
- Major redesign compared to previous versions.
- Specification is now nearing completion, currently at draft 23.
- Several implementations underway, tracking changes to specification, working on inter-op.

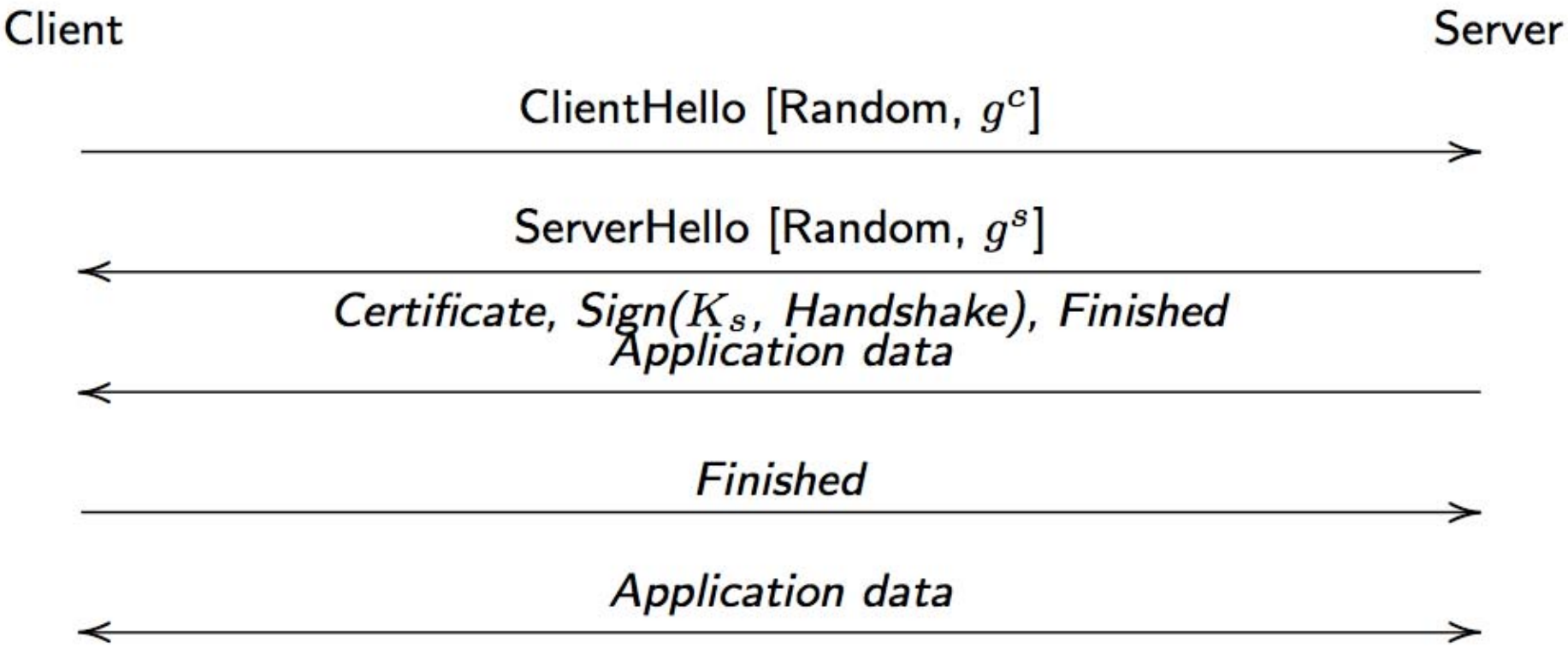
TLS 1.3

- Main objectives for TLS 1.3:
  - Reduce latency of initial secure data communication (1-RTT and 0-RTT for resumed sessions).
  - Improve security and privacy.
  - Protocol simplification (reducing options and removing broken cipher suites).
  - **No** compression, RC4, MAC-then-Encrypt, **RSA key transport, custom DH and ECDH groups, renegotiation**.
  - Unifying session resumption and PSK mechanisms.
  - But continuity for most important use cases (e.g. post handshake client authentication).

TLS 1.3

- Significant involvement of academic community during the design process.
  - Security analysis of early drafts of the protocol by several teams, using provable security and symbolic analysis.
  - Some significant errors uncovered during development.
  - Analysis on-going: draft spec keeps changing!

TLS 1.3 Handshake – 1-RTT



- Server can send secure data in its first message.
- Client can send secure data in its second message.

TLS 1.3 Handshake – 1-RTT

- Client includes DH share(s) in its first message, along with ClientHello, anticipating group that server will prefer.
- Server responds with single DH share in its ServerHello response.
- If this works, a forward-secure key is established after 1 round trip (1-RTT).
- Clients can cache groups preferred by popular servers.
- If server does not like DH groups used by client, it sends a HelloRetryRequest and a group back to client.
- In this instance, the handshake requires two round trips (2-RTT).

TLS 1.3 Handshake – DH and ECDH groups

- Limited set of DH and ECDH groups will be supported in TLS 1.3.
- Reduces likelihood of fall-back to 2-RTT.
- Removes problem of client not being able to validate groups that was inherent in TLS 1.2 and earlier.
- Removes complexity from implementations.

TLS 1.3 Handshake – DH and ECDH groups

- DH groups:
  - Specified in RFC 7919
  - |p| = 2048, 3072, 4096,6144, 8192.
  - All p are such that q = (p-1)/2 is prime.
  - Removes several avenues of attack: backdoored primes, small subgroup attacks, etc (cf. recent work by Fried *et al.*, Valenta *et al.*)
- ECDH groups:
  - Some existing curves from RFC 4492 and 2 new curves in RFC 7748.
  - NIST P256, P384, P521; Curve25519, Curve448.

TLS 1.3 Handshake – 0-RTT

- Prior versions of TLS: session resumption feature.
  - Lightweight handshake protocol, exchange of nonces and new key derivation.
  - No public key crypto, but still 1-RTT.
- Under pressure from QUIC design, TLS WG decided to add a **0-RTT** option to TLS 1.3.
  - Enables client to send encrypted data in its first message.
  - Not fully forward secure, since it uses either an old key or a new DH value from client but old DH value from server.
  - After a lot of analysis, it was realised that providing anti-replay for such messages was hard-to-impossible in distributed server environments.

TLS 1.3 Handshake – 0-RTT

- **Elegant theoretical solution**: achieve forward secure 0-RTT using HIBE + puncturable encryption techniques (e.g., Günther *et al.*, EC'17).
- **Actual solution**: forget about protecting against replay attacks and use the feature only for certain types of data where replay is not an issue.
- Now 0-RTT handshakes are bootstrapped using keys from previous protocol runs.
- **Problem**: how to explain security of 0-RRT data to developers?
- **Solution**: maintain a separate API for 0-RTT data.
- **Residual problem**: performance gain is too tempting for developers to heed warnings about its dangers.

- It was also realised that 0-RTT and PSK flows could be unified.
- PSK is an important use case for, e.g. IoT applications.

TLS 1.3 – Other features

- **Post-handshake client authentication**: previously done using renegotiation, now done with special handshake messages.
  - Server sends CertificateRequest message; client responds with Certificate, CertificateVerify, Finished.
- **Key update mechanism**: based on data limits for AES-GCM and ChaCha20Poly1305, derived from security proofs.
- **Record Protocol:** features AEAD only, traffic padding, single plaintext type field and encrypted type, use of masked nonces.
- **Key schedule:** derivations using HKDF and labels; much more complex key schedule; hash for HKDF negotiated in handshake; proper key separation of all keys allowing easier analysis.

Concluding Remarks

Concluding Remarks

- The TLS Handshake Protocol uses mostly "boring" cryptography yet is extraordinarily complex.
  - Much more so than typical key exchange protocols appearing in the scientific literature.
  - Making the protocol resistant to analysis efforts.
- Some protocol design errors were made, but not too many.
- Legacy support for EXPORT cipher suites and long tail of old versions has opened up serious vulnerabilities.
- Lack of formal state-machine description, lack of API specification, and sheer complexity of specifications have led to many serious implementation errors.
- Some, but not all of this, is being fixed in TLS 1.3.

Concluding Remarks

- Public key cryptography has evolved significantly in TLS.
- The largest shift has been from RSA key transport to elliptic curve Diffie-Hellman.
- A second shift now underway is to move to using newer elliptic curves like Curve25519, allowing greater speed and better implementation security.
- A third shift is the move away from SHA-1 in certs.
- A future shift may (will?) be needed to incorporate post-quantum algorithms.
- But implementation vulnerabilities are bound to continue to be discovered.

Fin