Beer-Talk bei Compass Security Schweiz AG

SSL/TLS: Angriffe und Gegenmassnahmen

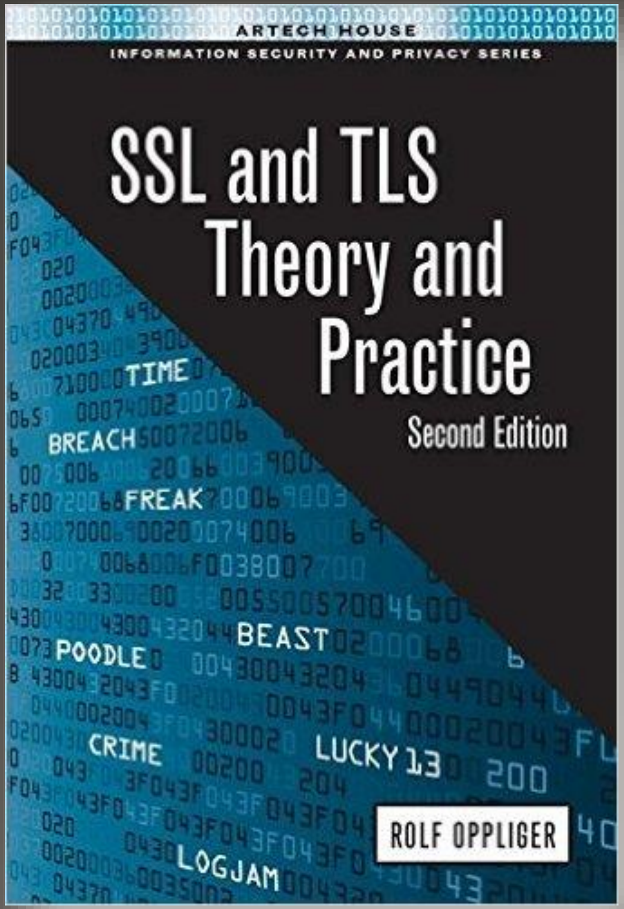
Donnerstag, 26. Mai 2016, Bern / 18:00 Uhr
Donnerstag, 09. Juni 2016, Jona / 18:00 Uhr

Guest Speaker

- eSECURITY Technologies Rolf Oppliger (founder and owner)
- Swiss Federal IT Steering Unit FITSU (scientific employee)
- University of Zurich (adjunct professor)
- Artech House (author and series editor for information security and privacy)

→ http://www.esecurity.ch/bio.html

Reference Book



© Artech House (2016)

ISBN 978-1-60807-998-8

→ http://books.esecurity.ch/ssltls2e.html

Challenge me !

Outline

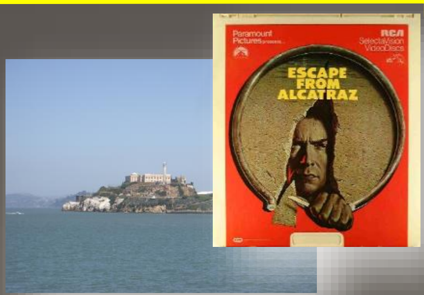
1. Introduction
2. SSL/TLS Protocols
3. Padding Oracle Attacks
4. CBC IV Attack
5. Renegotiation Attacks
6. Compression-related Attacks
7. Key Exchange Downgrade Attacks
8. Concluding Remarks

The order of the attacks is more or less chronological and not relevant

1. Introduction

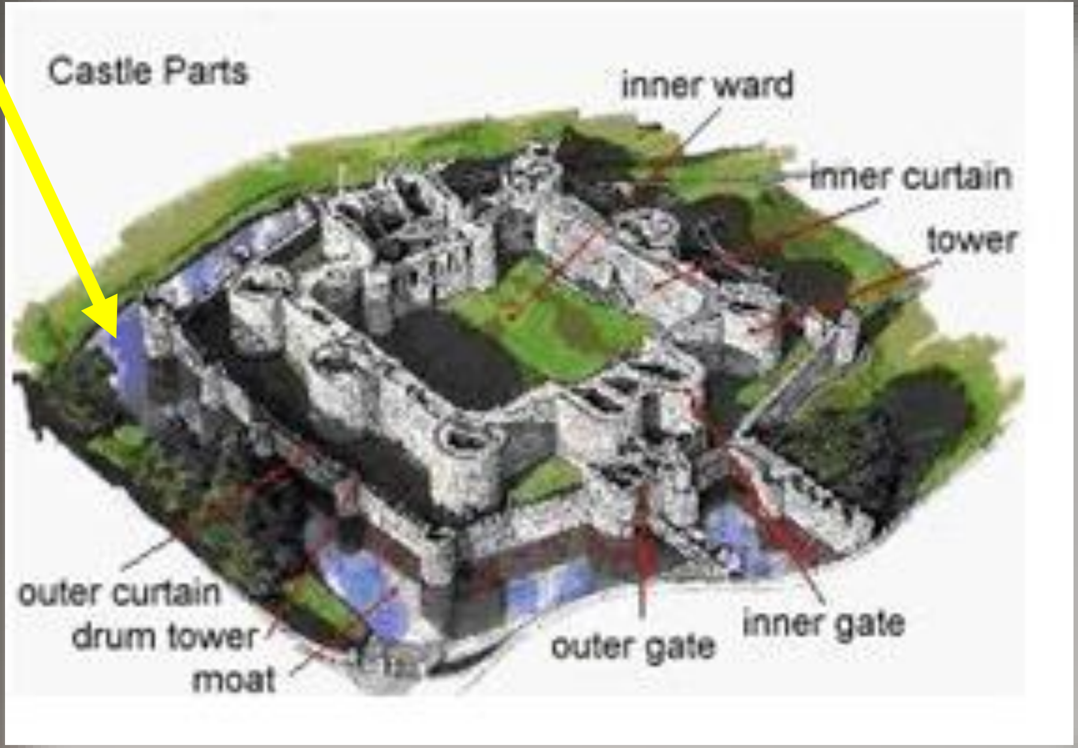
- The SSL/TLS protocols are omnipresent and widely deployed on the Internet
  - E-mail / messaging
  - Internet banking
  - Online shopping / gambling
  - Remote Internet voting
  - …
- Due to their success, the SSL/TLS protocols are under high pressure and subject to many attacks
- Why can't we solve the security problems once and for all?

- An absolute notion of security can only be achieved in theory (i.e., in a «clean» and well-defined model)
- In practice, all implementations slightly deviate from a theoretical model
- All real-world implementations have vulnerabilities and weaknesses that may be exploited
- In such a situation, it is common to play «cops and robbers»
  - «Secure» money transport
  - «Burglar-proof» safe
  - «Escape-proof» prison
  - ...

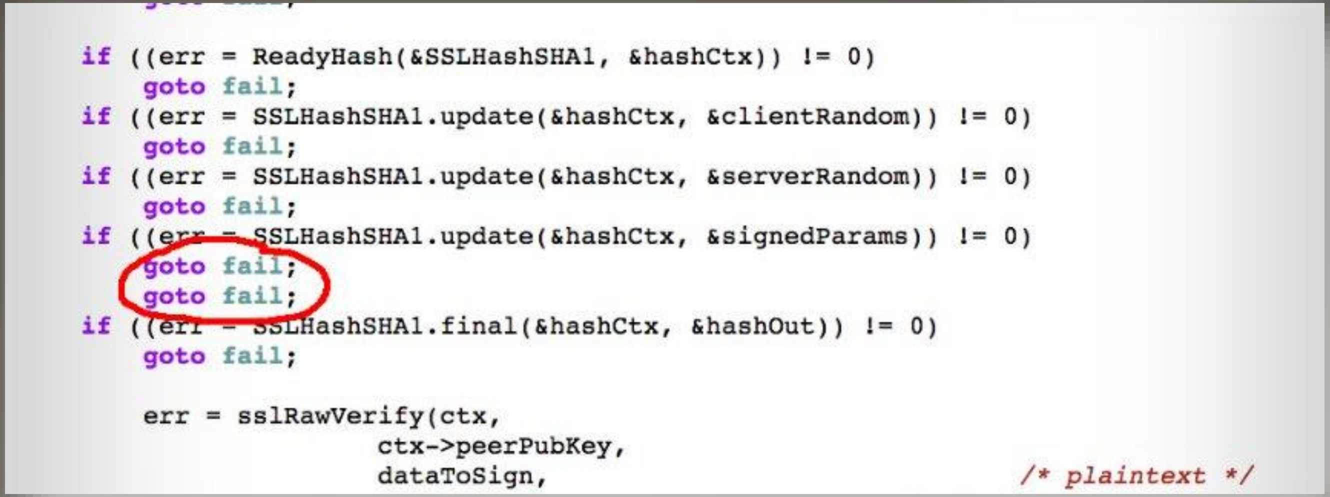


- This also applies to the SSL/TLS protocols

- A medieval castle may serve as an analogy
  - Sometimes it needs to be patched
  - Sometimes it needs to be protected with additional defenses (counter-measures)
- Both approaches are important (short-term vs. long-term)
- But they may also be subject to counterattacks
- This keeps the «cops and robbers» game alive



- In this talk, we focus on the security of SSL/TLS at the protocol level
- There are many other topics related to SSL/TLS security that are equally important but not addressed here
  - Trapdoors (e.g., Superfish)
  - Certificate-based attacks
  - Errors and bugs (e.g., «Goto fail» bug, Heartbleed, …)
  - Cryptographic weaknesses and problems
    - Weak PRBGs
    - BERserk
    - Transcript collision attacks (SLOTH)
    - …



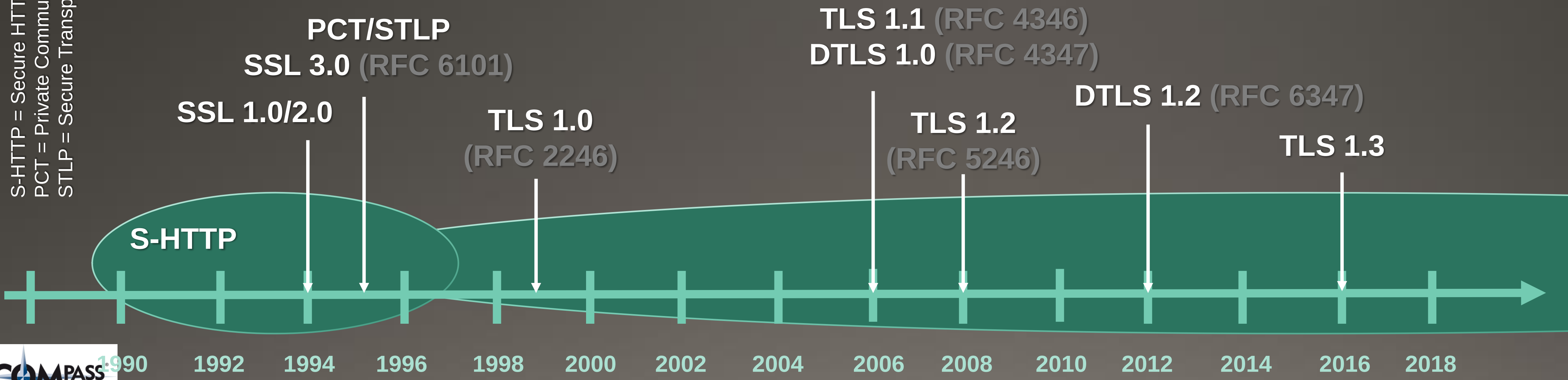
```
    if ((err = ReadyHash(&SSLHashSHA1, &hashCtx)) != 0)
        goto fail;
    if ((err = SSLHashSHA1.update(&hashCtx, &clientRandom)) != 0)
        goto fail;
    if ((err = SSLHashSHA1.update(&hashCtx, &serverRandom)) != 0)
        goto fail;
    if ((err = SSLHashSHA1.update(&hashCtx, &signedParams))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

        err = sslRawVerify(ctx,
                           ctx->peerPubKey,
                           dataToSign,                /* plaintext */
```

2. SSL/TLS Protocols

- The Secure Sockets Layer (SSL) protocol was originally developed by Netscape Communications in the 1990s
- It was later adopted (and adapted) by the IETF Transport Layer Security (TLS) WG

S-HTTP = Secure HTTP
PCT = Private Communication Technology
STLP = Secure Transport Layer Protocol



- The SSL/TLS protocols provide four basic security services
  - Peer entity authentication service
  - Data authentication service
  - Connection confiden-tiality service
  - Connection integrity service (without recovery)

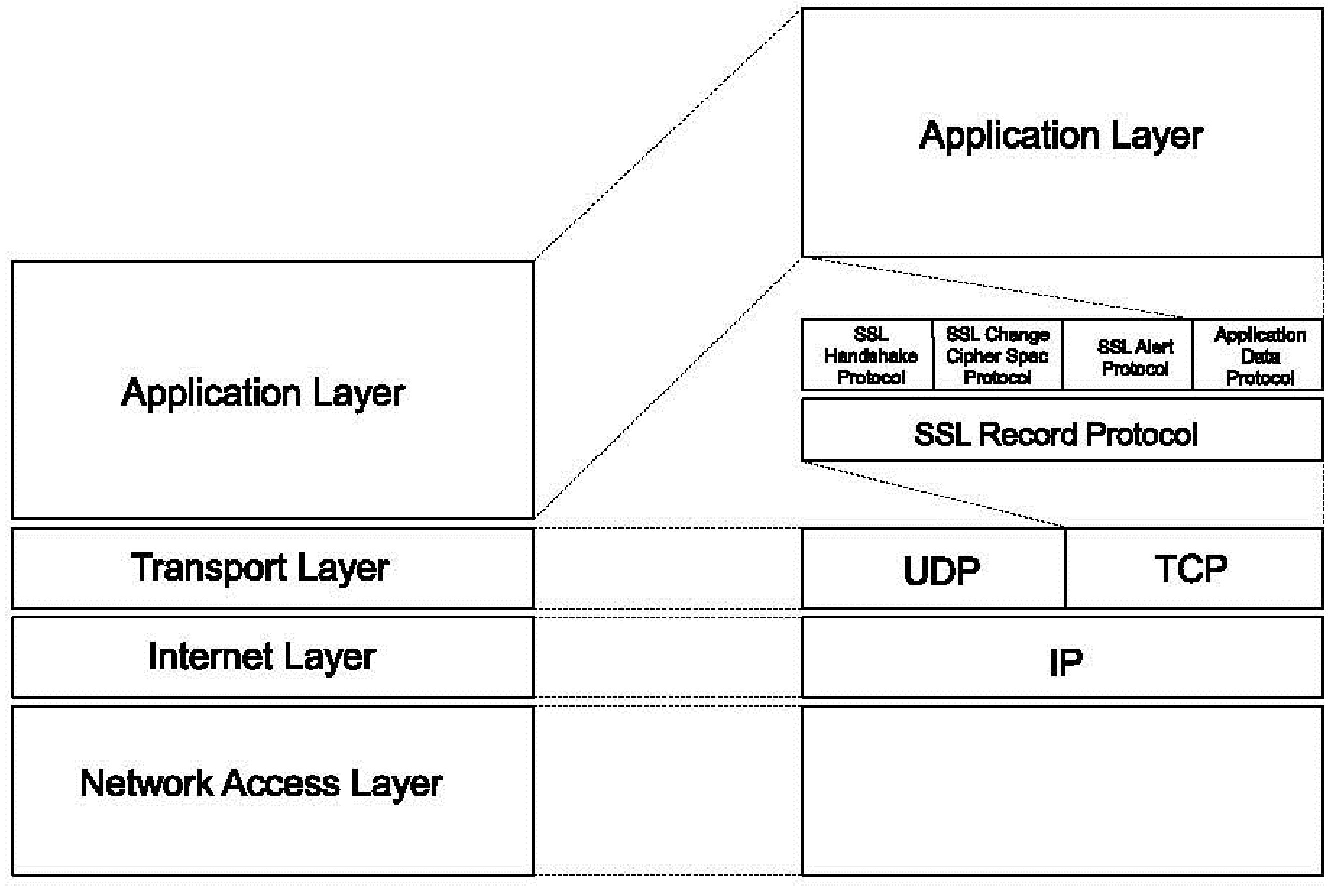


Figure 2.1 The SSL with its (sub)layers and (sub)protocols.

SSL/TLS Record Protocol

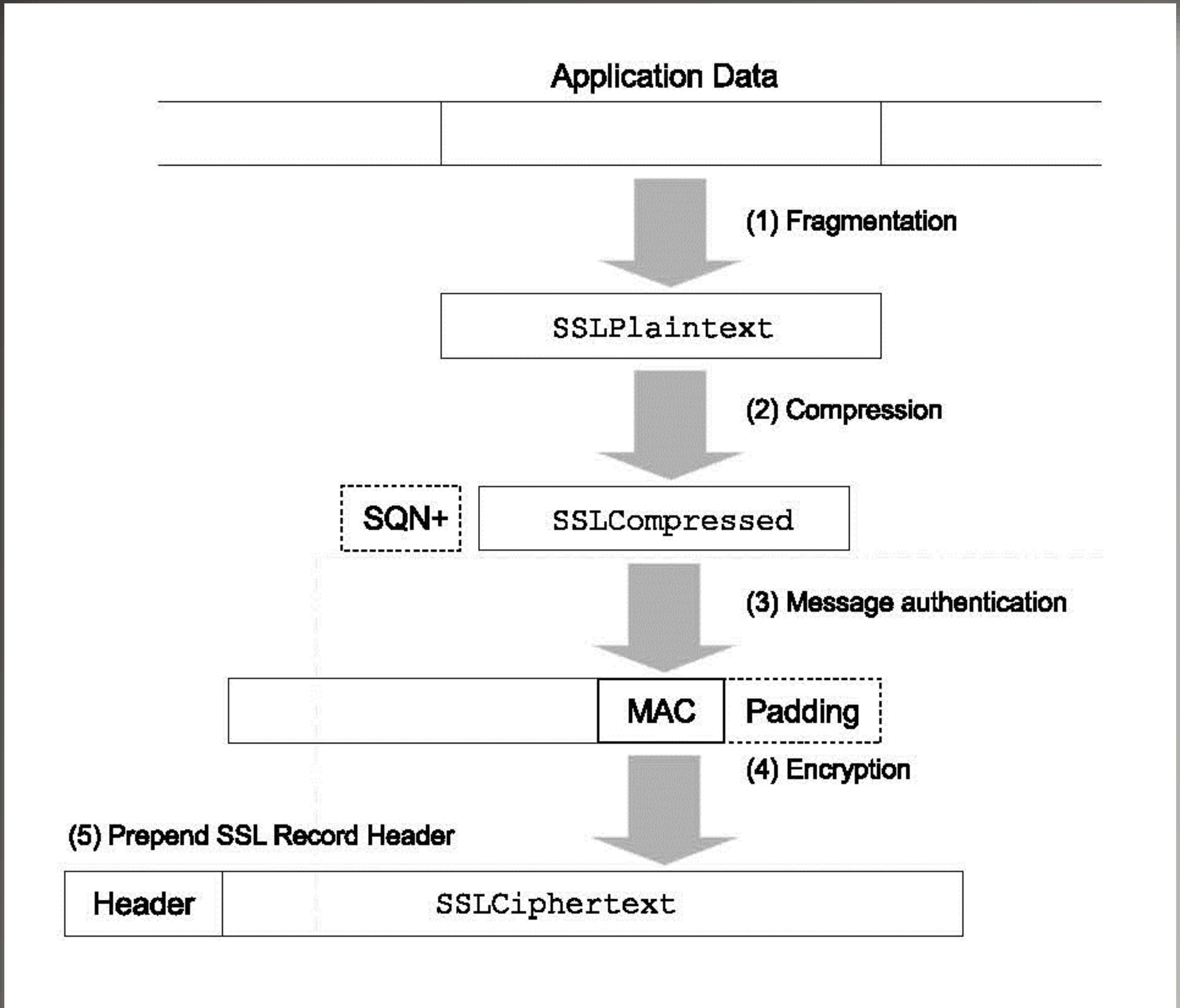


Figure 2.3 The SSL record processing (overview).

- Each SSL/TLS record consists of
  - Type (1 byte)
    - `20` = Change Cipher Spec
    - `21` = Alert
    - `22` = Handshake
    - `23` = Application Data
  - Version (2 bytes) `0x0300 = 3,0 (SSL)`
  - Length (2 bytes) $< 2^{14}-1 = 16,384$
  - Fragment (variable length)
- The SSL/TLS record protocols follow the Authenticate-then-Encrypt (AtE) approach

SSL/TLS Handshake Protocol

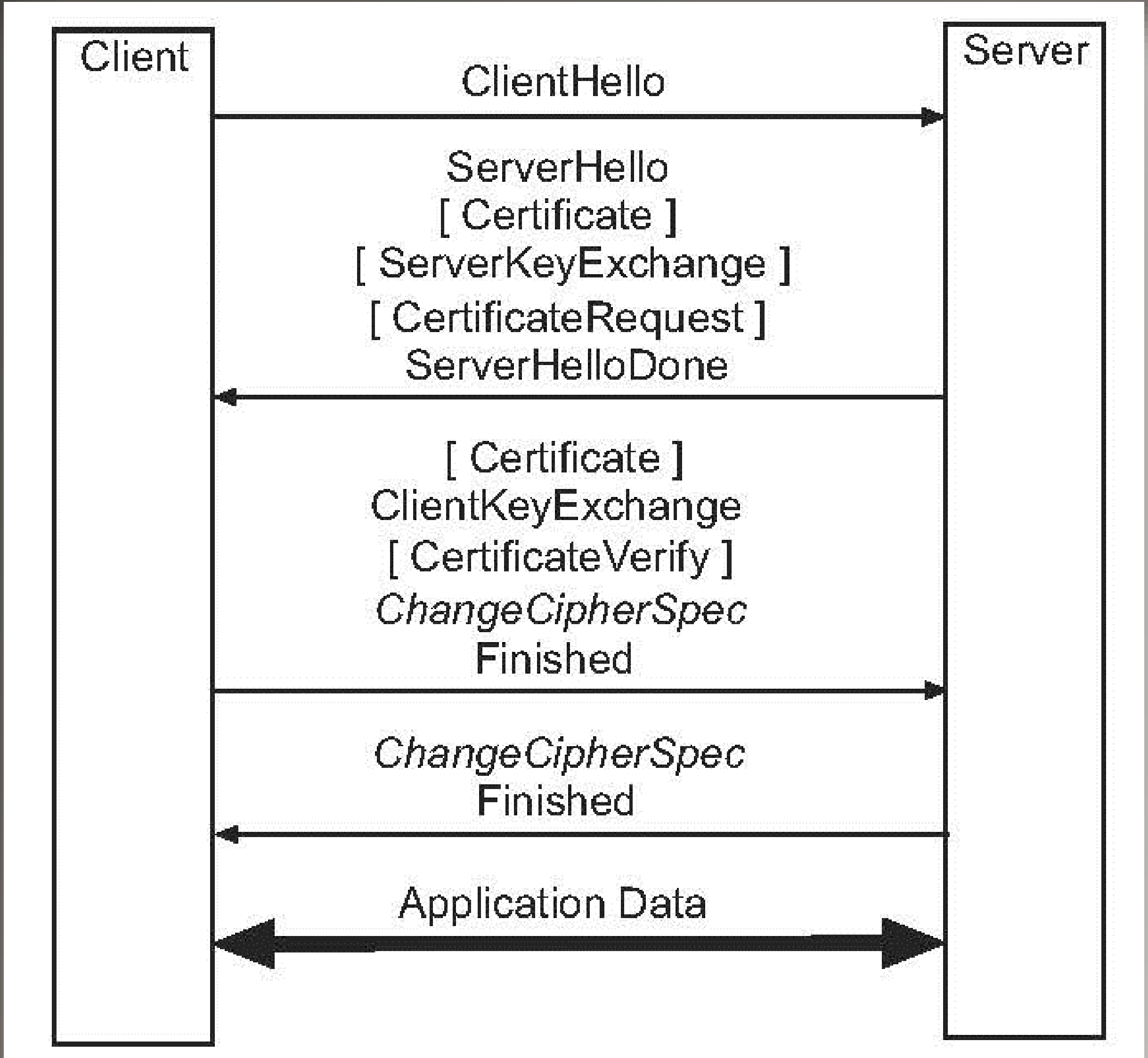


Figure 2.5 The SSL handshake protocol.

- The SSL/TLS protocols employ cryptographic techniques from the 1990s
- It has been assumed that the protocols are sufficiently secure to be used in practice
- But in the recent past, a number of sophisticated attacks (e.g., BEAST, CRIME, TIME, BREACH, POODLE, Lucky 13, FREAK, Logjam, and DROWN) have brought the SSL protocol to the end of its life cycle and the TLS protocol to several new versions
- Each TLS protocol version corrects some vulnerabilities or weaknesses of its predecessor
- With the soon-to-be-released version 1.3, the TLS protocol has reached a “new” (and very mature) level of security
- This is not the end of the story (as «attacks always get better»)

3. Padding Oracle Attacks

- In cryptography, a padding oracle attack refers to a special type of a chosen ciphertext attack (CCA) or adaptive CCA (CCA2)
- Examples
  - Bleichenbacher attack (1998)
  - Vaudenay attack (2002)

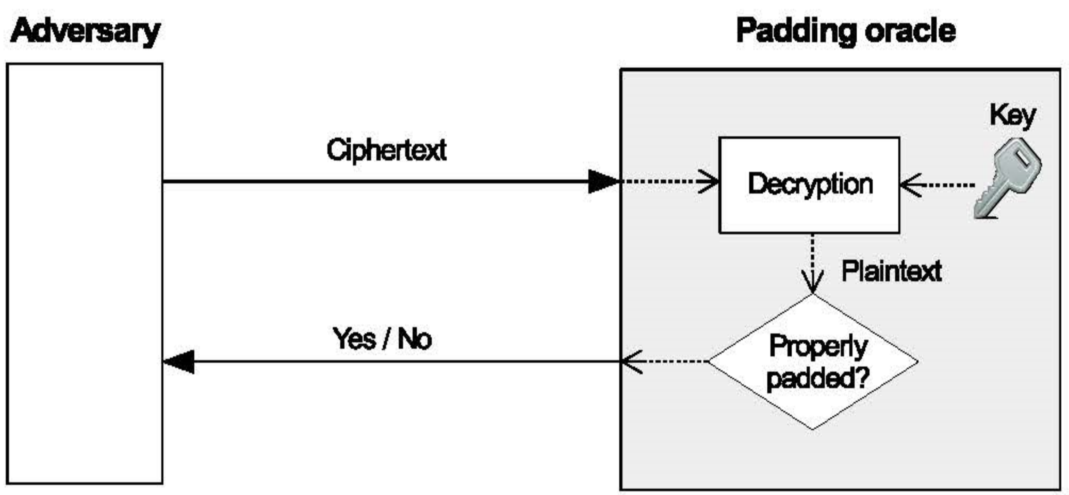


Figure B.1 A padding oracle attack.

Bleichenbacher Attack

- In 1998, Daniel Bleichenbacher proposed a padding oracle attack against PKCS #1 (block type 2) that is used in the SSL/TLS ClientKeyExchange message

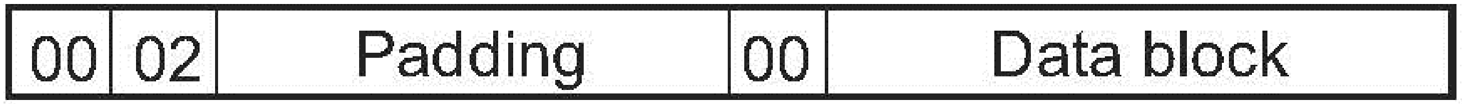


Figure B.2 PKCS #1 block format for encryption (block type 2).

- The adversary can send arbitrary ciphertexts $C_i = C \cdot r_i^e$ to the oracle, and for each C_i the oracle returns one bit of information, i.e., whether the plaintext after decryption [i.e., $P_i = C_i^d = (C \cdot r_i^e)^d = C^d \cdot r_i^{ed} = P \cdot r_i$] is properly padded and sized (this yields information about $P = P_i / r_i$)
- If invoked sufficiently many times ($\sim 2^{20} \approx 10^6$), then the adversary can decrypt an RSA-encrypted message (e.g., a ClientKeyExchange message) and extract the premaster secret (cf. pp. 247 – 251)

- Patch
  - Since TLS 1.0, it is informally required that an implementation avoids leaking information about the correctness of the padding (including, for example, error messages and timing information)
  - Since TLS 1.2, it is formally required that an implementation generates a random string anyway, and that this string is used instead of the premaster secret in case of a padding error (the protocol aborts later)

```
1. Generate a string R of 46 random bytes

2. Decrypt the message to recover the plaintext M

3. If the PKCS#1 padding is not correct, or the length of message
   M is not exactly 48 bytes:
      pre_master_secret = ClientHello.client_version || R
   else If ClientHello.client_version <= TLS 1.0, and version
   number check is explicitly disabled:
      pre_master_secret = M
   else:
      pre_master_secret = ClientHello.client_version || M[2..47]
```

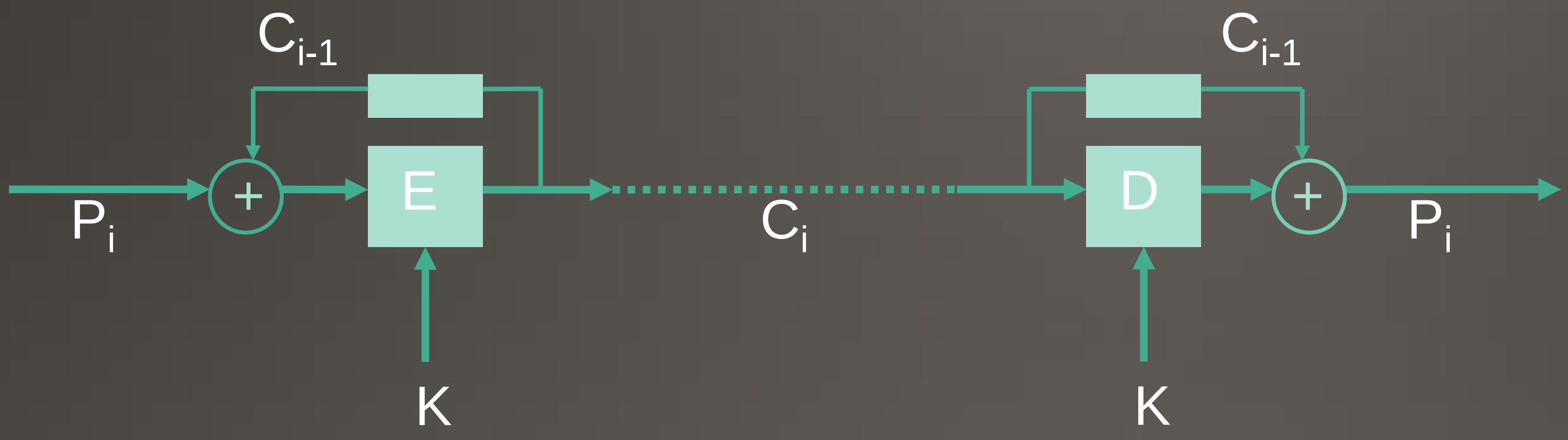
- In March 2016, a group of researchers showed that this patch is not foolproof
- They came up with a **Decrypting RSA with Obsolete and Weakened eNcryption (DROWN)** attack
- It is a cross-protocol attack that exploits the fact that many servers still support SSL 2.0 (using the same RSA key)
- The DROWN attack starts from the observation that the patch can be circumvented, if the adversary can send a ciphertext to the server multiple times and he or she can recognize whether the master secret used by the server is always the same (only in this case is the ciphertext properly padded)
- In SSL 2.0, this can be done for two reasons
  - The master secret is derived deterministically from the decrypted ciphertext (since SSL 3.0, the decrypted ciphertext refers to a premaster secret that takes into account addional randomness to derive the master secret)
  - 40-bit export ciphers are supported by default (and the respective keys can be found in an exhaustive search)

- Countermeasure
  - Update PKCS #1 to make it secure against CCA2
  - This was done in PKCS #1 version 2 by adopting a padding scheme known as Optimal Asymmetric Encryption Padding (OAEP)
  - PKCS #1 version 2 is mandatory since TLS 1.2
- Unfortunately, even a CCA2-secure encryption scheme may not defeat all possibilities to mount a Bleichenbacher attack against an implementation (due to the existence of side-channels)
- Respective attacks have been shown by James Manger, as well as Vlastimil Klima, Ondrej Pokorny, and Tomas Rosa

Vaudenay Attack

- The Bleichenbacher attack only affects asymmetric encryption (i.e., RSA encryption)
- In 2002, Serge Vaudenay proposed a similar (padding oracle) attack that affects any symmetric encryption using a block cipher in CBC mode (cf. pp. 251 – 260)
- The original attack was purely theoretical, and it was not clear how to mount it in practice
- In 2003, it was shown by Vaudenay et al. that the attack can be mounted in practice (i.e., to decrypt an IMAP4 password sent over an SSL/TLS connection)

- Cipherblock chaining (CBC) mode



Encryption: $C_0 = IV$ and $C_i = E_K(P_i \oplus C_{i-1})$ for $i>0$

Decryption: $C_0 = IV$ and $P_i = D_K(C_i) \oplus C_{i-1}$ for $i>0$

- The decryption formula also applies at the bit or byte level

- In CBC mode, it is required that the plaintext length is a multiple of the block size (typically, k = 16 bytes for AES)
- In theory, there are many padding schemes that can be used
- In practice, PKCS #7 (RFC 5652) is the most widely used padding scheme (PKCS #5 is similar but restricted to a block length of 8 bytes)

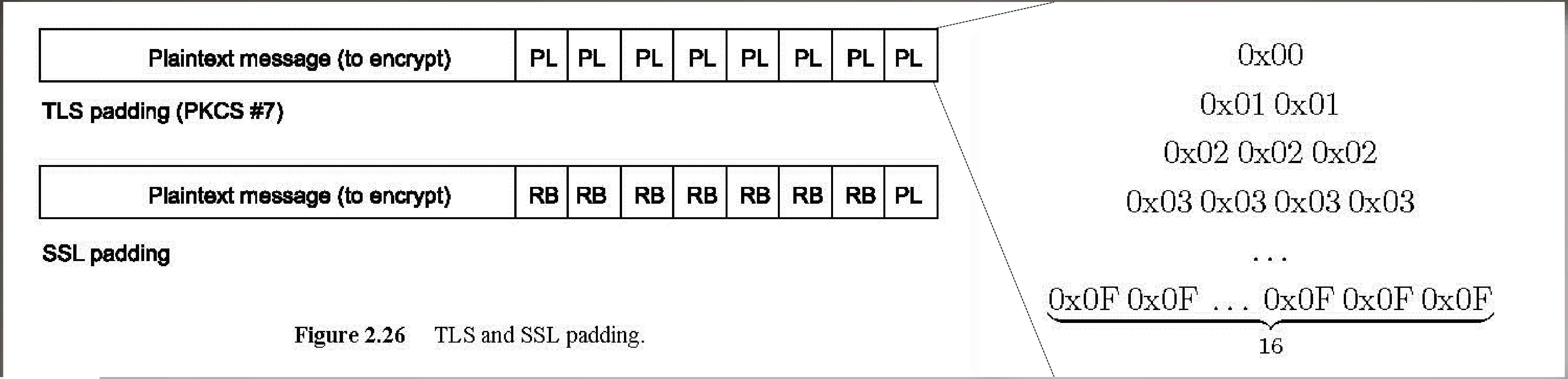


Figure 2.26 TLS and SSL padding.

PL = Padding length -1
RB = Random Byte

- In a Vaudenay attack, the adversary tries to decrypt a ciphertext block C_i that may comprise a password or bearer token
- The k bytes of $C_i = C_i[1]C_i[2]C_i[3]\ldots C_i[k]$ can be attacked individually
- The adversary knows $C_{i-1} \| C_i$ and mounts a CCA2
- In each step, the adversary replaces C_{i-1} with a specifically crafted ciphertext block C' and submits the two-block ciphertext $C' \| C_i$ as the payload of an SSL/TLS record to the (padding) oracle
- For each submission, the oracle reveals one bit of information, namely whether $D_K(C' \| C_i)$ is properly padded or not
- This information can be revealed by a particular error message (`decryption_failed` vs. `bad_record_mac`) or a timing (side-) channel
- Note that the MAC is computed iff the record is properly padded

- To attack $C_i[16]$, the adversary tries out all possible byte values for C'[16], until the padding oracle responds in the affirmative way
- In this case, the adversary knows that the decrypted block has a valid padding (most likely 0x00)
- This means that $D_K(C_i)[16] \oplus C'[16] = 0x00$ and – because $C_i = E_K(P_i \oplus C_{i-1})$ – $D_K(E_K(P_i \oplus C_{i-1}))[16] \oplus C'[16] = 0x00$
- This, in turn, means that $(P_i \oplus C_{i-1})[16] \oplus C'[16] = 0x00$, $P_i[16] \oplus C_{i-1}[16] \oplus C'[16] = 0x00$, and hence $P_i[16] = C_{i-1}[16] \oplus C'[16]$
- $P_i[16]$ can be determined, because $C_{i-1}[16]$ and C'[16] are known values

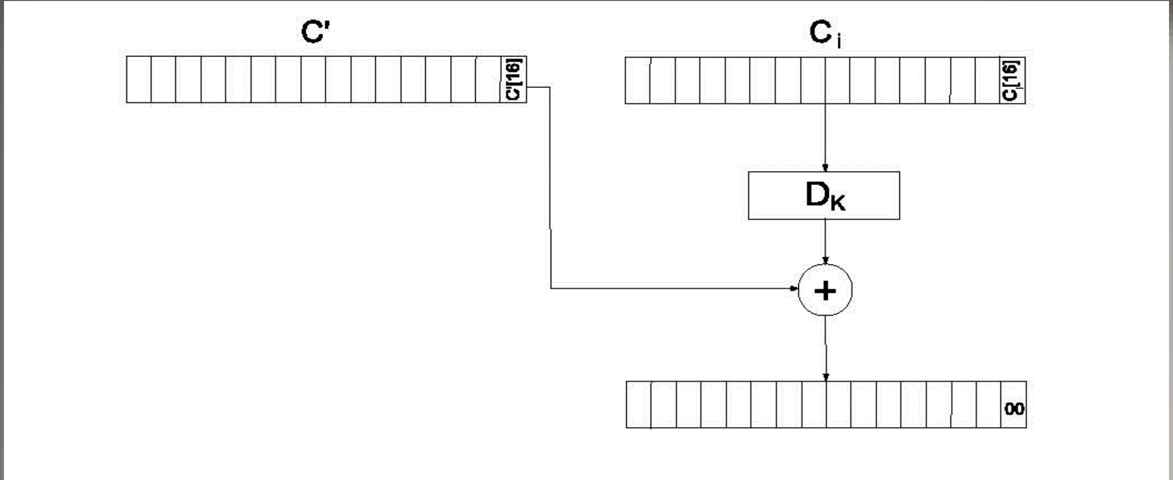
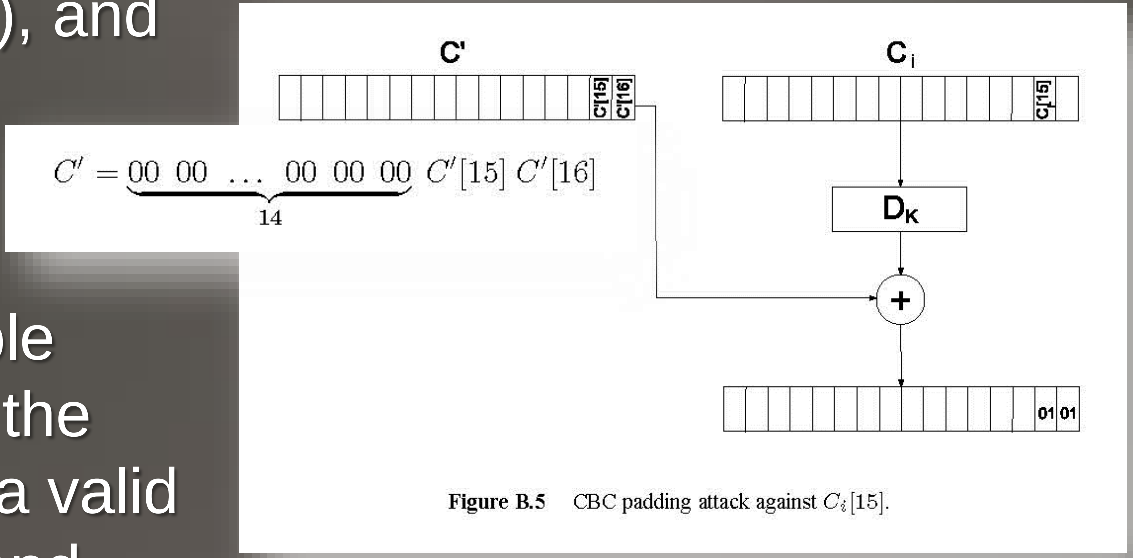


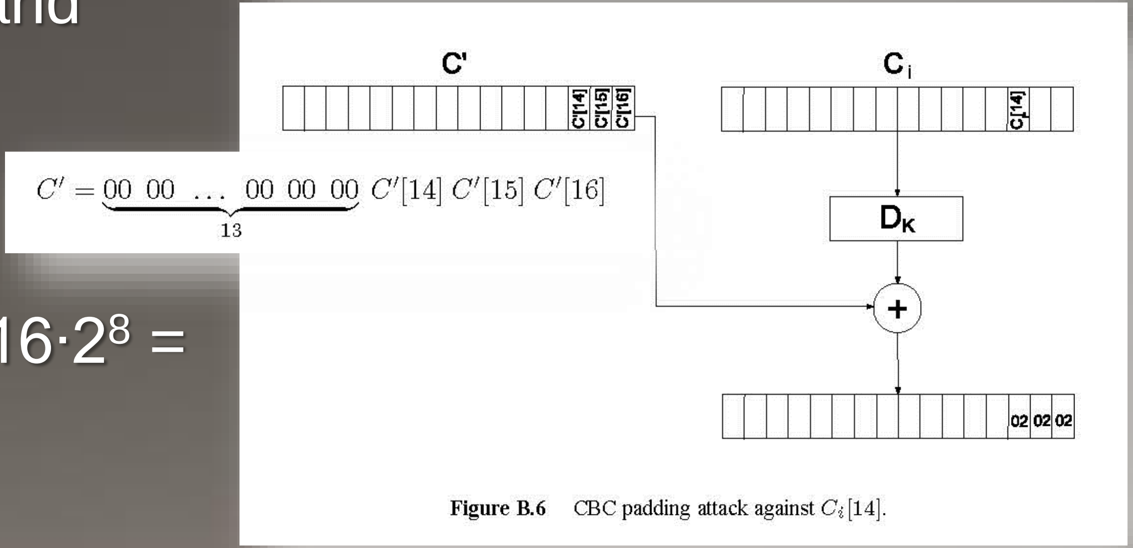
Figure B.4 CBC padding attack against $C_i[16]$.

- To attack $C_i[15]$, the adversary updates $C'[16] = 0x01 \oplus P_i[16] \oplus C_{i-1}[16]$, tries out possible byte values for $C'[15]$ until the decryption yields a valid padding (i.e., 0x0101), and then decrypts $C_i[15]$
- To attack $C_i[14]$, the adversary updates $C'[16]$ and $C'[15]$, tries out possible byte values for $C'[14]$ until the decryption of $C_i[14]$ yields a valid padding (i.e., 0x020202), and then decrypts $C_i[14]$
- This procedure is repeated for all bytes in C_i
- For k=16, the workload is $16 \cdot 2^8 = 2^4 \cdot 2^8 = 2^{12} = 4'096$



$$C' = \underbrace{00\ 00\ \ldots\ 00\ 00\ 00}_{14}\ C'[15]\ C'[16]$$

Figure B.5 CBC padding attack against $C_i[15]$.

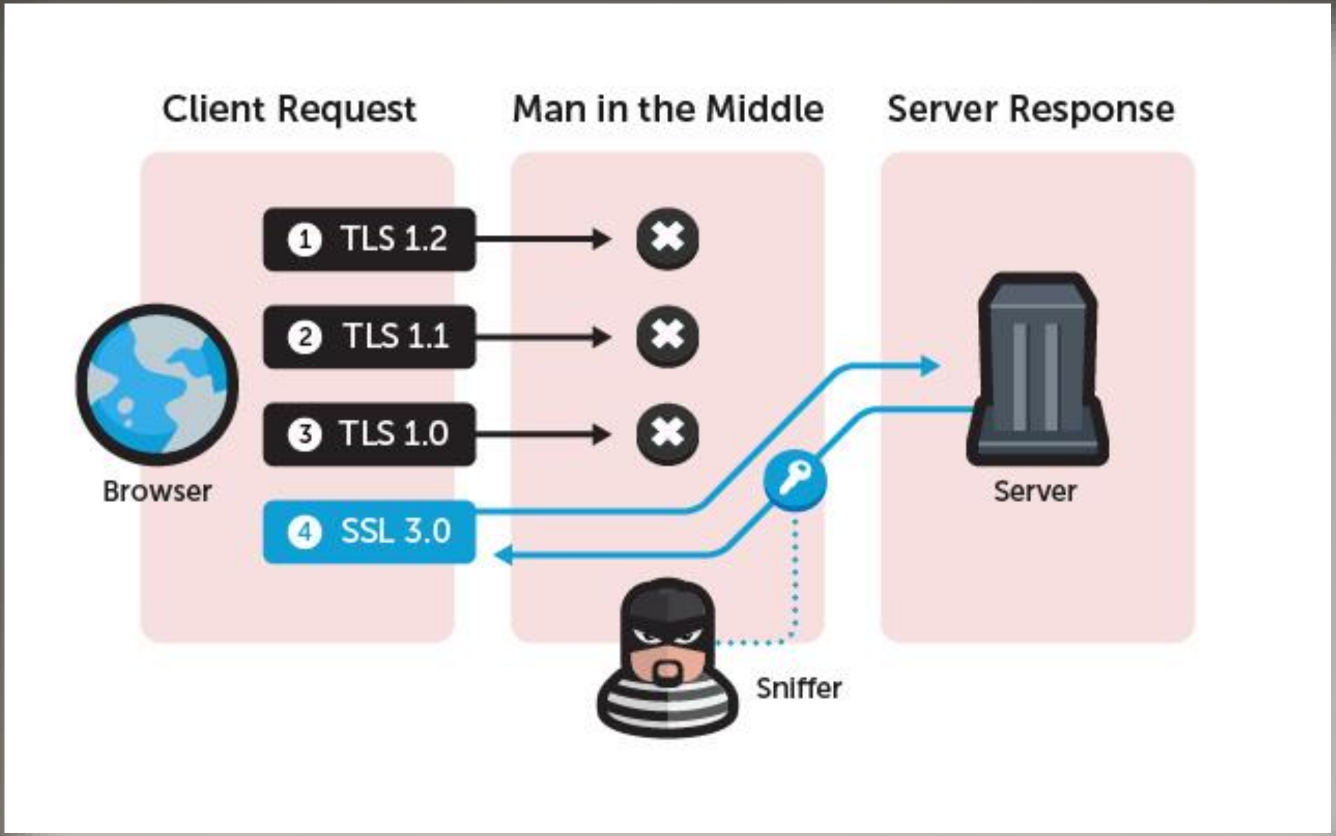


$$C' = \underbrace{00\ 00\ \ldots\ 00\ 00\ 00}_{13}\ C'[14]\ C'[15]\ C'[16]$$

Figure B.6 CBC padding attack against $C_i[14]$.

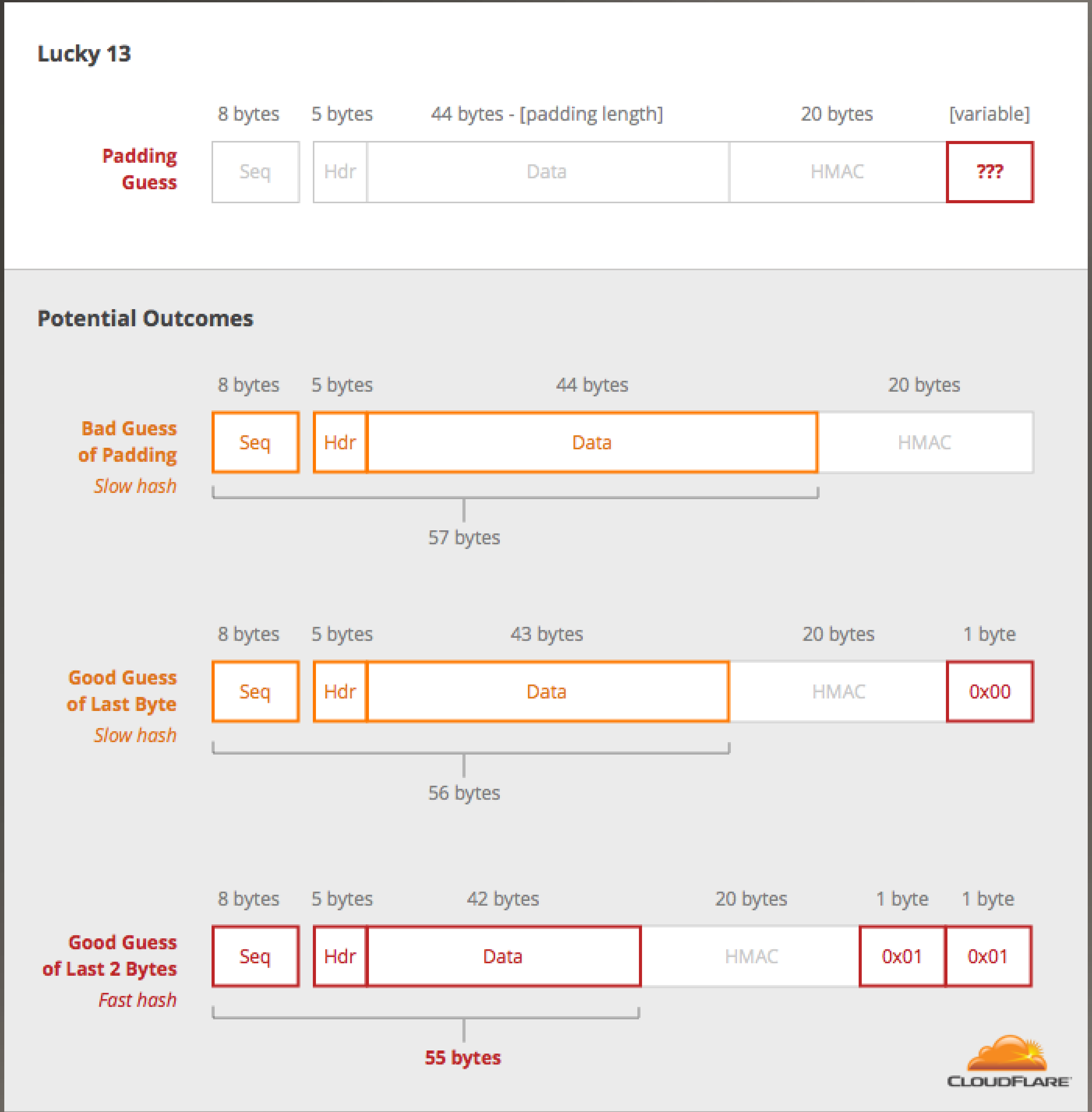
- The attack does not target the key and is independent from it
- This means that the attack can be mounted even if C_i is encryp-ted with different keys (e.g., in multiple sessions)
- This makes the attack feasible in practice
- The SSL protocol employs a simpler padding scheme that uses random padding bytes (only the last padding byte refers to the padding length)
- This simplifies the padding oracle attack considerably, because it is simple to distinguish a correct padding from an incorrect one
  - If the padding is correct, then the protocol execution continues and no error message is sent (i.e., the HAMC remains valid)
  - If the padding is incorrect, then the protocol execution is aborted and an error message is sent

- The respective attack is very effective
- It is called **Padding Oracle dOwngradeD Legacy Encryption (POODLE)** attack attack (cf. pp. 82 – 87)
- To enforce the use of the SSL protocol, the POODLE attack typically comes along with a protocol version downgrade attack
- SSL 3.0 was deprecated by the IETF in June 2015



- Patch (TLS)
  - Make it impossible for an adversary to distinguish between a padding error and a MAC error
    - Since TLS 1.1, there is a single alert message (i.e., `bad_record_mac`) to signal both a padding error and a MAC error
    - Also, to avoid a timing (side-) channel, the TLS 1.1 specification requires that «*implementations must ensure that record processing time is essentially the same whether or not the padding is correct. In general, the best way to do this is to compute the MAC even if the padding is incorrect, and only then reject the packet. For instance, if the pad appears to be incorrect, the implementation might assume a zero-length pad and then compute the MAC. 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.*»

- With this patch in place, it was commonly believed that an implementation would be secure against Vaudenay attacks
- This changed in 2013, when Nadhem J. Al-Fardan and Kenny Paterson demonstrated an attack known as **Lucky 13**
- The attack exploits the fact that the running time of the hash functions that are currently deployed (in particular, SHA-1) depends on the length of the input messages
- This tiny piece of information can be turned into a Vaudenay attack (cf. pp. 162 – 168)



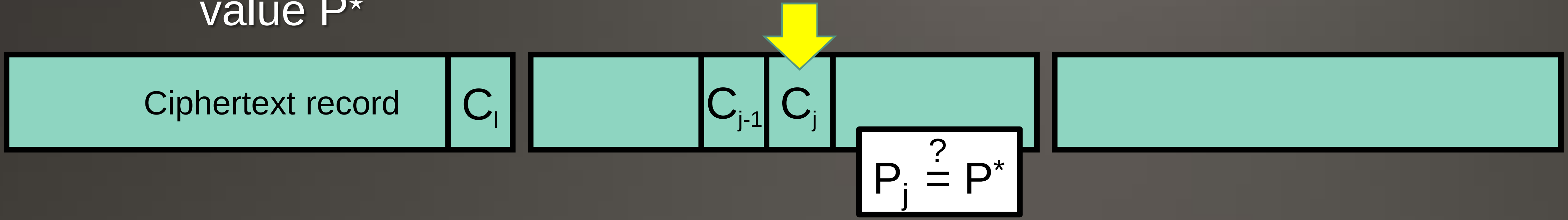
- The workload of the attack is $2^{16}+14 \cdot 2^8 \approx 70'000$
- Compare this to $\approx$ 4'000 of the «normal» Vaudenay attack

- Countermeasures
  - To defeat «only» the POODLE attack: Disable SSL 3.0 or at least employ the signaling cipher suite TLS_FALLBACK_SCSV
  - Avoid any timing channel (even for the MAC generation)
  - Avoid any block cipher in CBC mode (in SSL 3.0, TLS 1.0, and TLS 1.1, this means that RC4 must be used – but RC4 has security problems of its own)
  - Use Encrypt-then-Authenticate (EtA) instead of AtE (there is a TLS extension for this purpose)
  - Since TLS 1.2, the use of algorithms for authenticated encryption with additional data (AEAD) is recommended (cf. RFC 5116), such as counter mode and CBC-MAC (CCM), Galois/counter mode (GCM) or ChaCha20-Poly1305 (RFC 7539)
  - TLS 1.3 will mandate the use of an AEAD cipher
  - A CCA2 like the Vaudenay attack can be detected on the server side

4. CBC IV Attack

- In 2004, Gregory Bard published another attack against CBC encryption and – more specifically – the way initialization vectors (IVs) are used
- In theory, CBC encryption requires a fresh and unpredictable IV for every message (record) that is encrypted
- In practice, however, SSL 3.0 and TLS 1.0 employ an explicit IV only for the first record (in a connection) and all subsequent IVs are then implicit and taken from the final block of the preceding record
- An adversary can therefore predict the IV that is used to encrypt the next record, and this can be turned into a **blockwise chosen plain-text attack (CPA)** that allows an adversary to determine a low-entropy string, such as a password or bearer token (cf. pp. 109 – 113)

- Assume an adversary who has observed a ciphertext $C = C_1, C_2, \ldots$ that may comprise multiple records and who wants to verify a guess as to whether a particular plaintext block P_j ($j > 1$) has a particular value P^*



- The adversary can mount a blockwise CPA and repeatedly have the oracle encrypt any message P′ of his or her choice
- More specifically, the adversary constructs a message whose initial block P'_1 is equal to $C_{j-1} \oplus C_l \oplus P^*$, where
  - C_{j-1} refers to the ciphertext block that immediately precedes the block under attack
  - C_l refers to the last ciphertext block of the record that immediately precedes the record in which C_j is transmitted

- When P'_1 is encrypted, the oracle computes $C'_1 = E_K(P'_1 \oplus C_l) = E_K(C_{j-1} \oplus C_l \oplus P^\star \oplus C_l) = E_K(C_{j-1} \oplus P^\star) = E_K(P^\star \oplus C_{j-1})$
- A comparison of $C'_1 = E_K(P^\star \oplus C_{j-1})$ with the «normal» CBC encryption formula for P_j [i.e., $C_j = E_K(P_j \oplus C_{j-1})$] reveals the fact that $C'_1 = C_j$ iff $P_j = P^\star$
- This means that the adversary can verify the guess $P^\star$ by comparing C'_1 with C_j
  - If $C'_1 = C_j$, then $P^\star$ is the correct guess for P_j
  - Otherwise, i.e., if $C'_1 \neq C_j$, then the procedure can be repeated with another value for $P^\star$
- This is repeated until the correct value for P_j is found
- For n possible values, the adversary must try n/2 values on the average

- The publications of Bard were taken into account when people specified TLS 1.1
- But except from that, the vulnerability went largely unnoticed in the public
- This changed immediately when Thai Duong and Juliano Rizzo presented a tool named **Browser Exploit Against SSL/TLS (BEAST)** at the Ekoparty security conference in 2011
- The tool consisted of JavaScript code that could mount the attack inside a browser (to decrypt a Paypal token without any server interaction)
- Due to its effectiveness and efficiency, the attack tool attracted a lot of media attention
- The acronym still makes people nervous about the security of SSL/TLS

- Patch
  - A simple patch is to split every record into two records, with the first record containing less than one block of plaintext
  - In the simplest and most widely deployed case, the first byte is sent in a first record (as a «dummy record»), and the remaining n−1 bytes are then sent in a second record (this is known as «1/n−1 record splitting»)
  - The dummy record randomizes the IV that is used to encrypt the main record
- Countermeasure
  - Due to Bard's publications, the designers of the TLS protocol modified the protocol and replaced the implicit IV with an explicit one
  - Since TLS 1.1, there is an appropriately sized IV that is randomly chosen and sent along with each TLS record
  - This defeats the CBC IV attack (no counterattack is known)

5. Renegotiation Attack

- There are several reasons why a TLS session may need to be re-negotiated
- In 2009, Marsh Ray and Steve Dispensa proposed a way to exploit the TLS session renegotiation mechanism to mount a special MITM attack (cf. pp. 152 – 158)

```
GET /orderProduct?deliverTo=Address-1
X-Ignore-This: GET /orderProduct?deliverTo=Address-2
Cookie: 7892AB9854
```

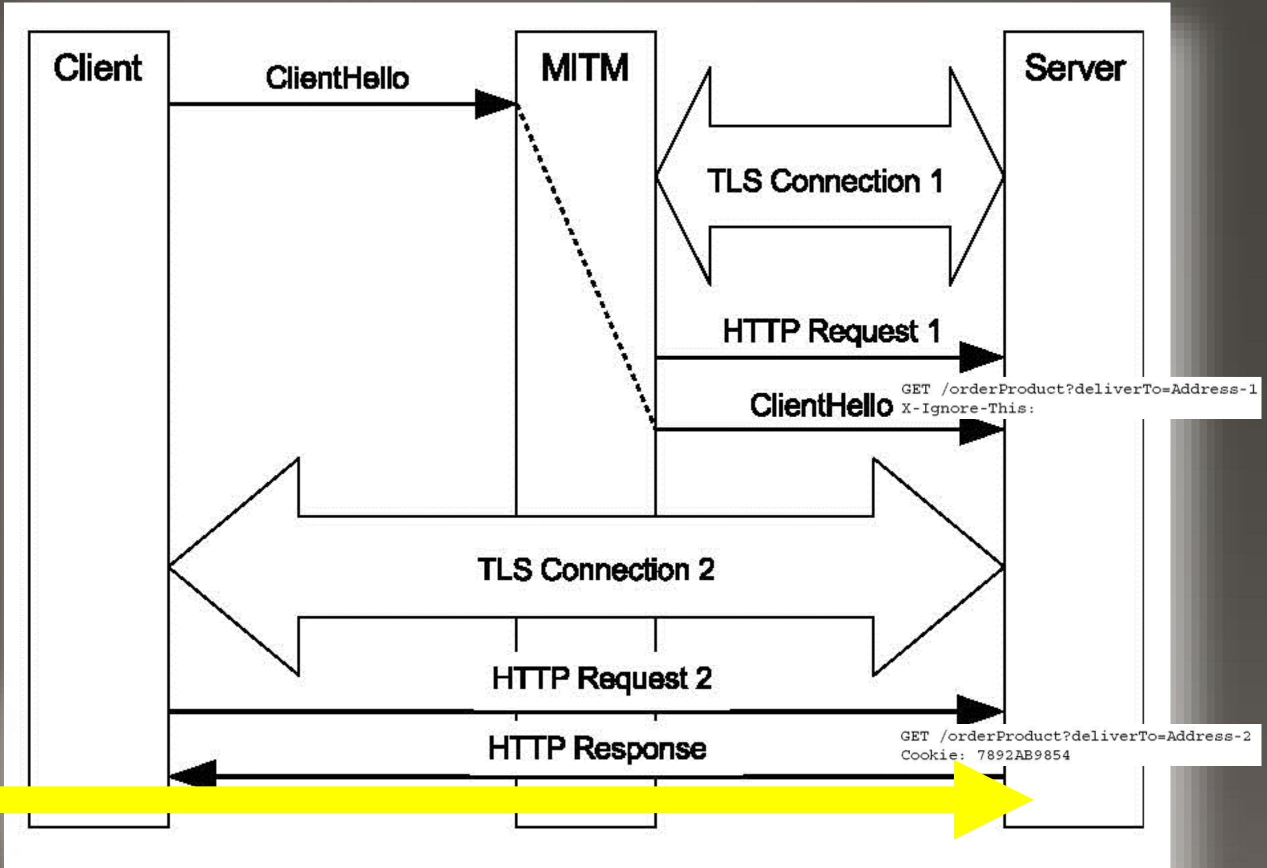


Figure 3.10 The TLS renegotiation attack (overview).

- Patches
  - Renegotiation attacks are conceptually similar to cross-site request forgery (CSRF) attacks, so any protection mechanism against CSRF may also help protecting against renegotiation attacks
  - Also, because the renegotiation feature is optional, a simple and straightforward protection mechanism is to disable the feature and not support renegotiation in the first place
  - More specifically, it is sufficient to only disable client-initiated renegotiation
  - This is not as simple as it looks like, mainly because renegotiation is still needed in many application settings

- Countermeasure
  - A proper countermeasure provides handshake recognition, meaning that it must ensure that both parties have the same view of the previous handshake (i.e., the handshake that is renegotiated)
  - This can be achieved by the TLS `renegotiation_info` extension (value 65,281) specified in RFC 5740
  - The `renegotiation_info` extension data field comprises the `verify_data` field(s) from the Finished handshake message(s) of the session that is renegotiated
  - The first message (i.e., the ClientHello message) must include an empty `renegotiation_info` extension
  - Since some implementations have problems with empty extensions, the client may alternatively include a special signaling cipher suite, i.e., TLS_EMPTY_ RENEGOTIATION_INFO_SCSV (0x00FF) in the list of supported ciphers (the secure renegotiation remains the same)

- Unfortunately, the protection the TLS `renegotiation_info` extension provides is not foolproof
- In 2014, it was shown by Karthikeyan Bhargavan, Antoine Delignat-Lavaud et al. that a **Triple Handshake Attack** is still feasible
- In this attack, the MITM can exploit two facts (or weaknesses)
  - The RSA key exchange is susceptible to an unknown key-share attack
  - If a session resumption is performed after an unknown key-share attack, then the two connections share the same `verify_data` field in the respective Finished messages

- The attack works in three steps (handshakes)
  1. The MITM mounts an unknown key-share attack to establish two TLS connections that share the same master key and session ID
  2. The MITM waits until the client initiates a session resumption (that only requires the master key and the session ID) and then simply relays handshake messages forth and back → there are now two synchronized connections that share the same `verify_data` field value
  3. Finally, the MITM can mount a "normal" renegotiation attack (i.e., he or she can send HTTP request 1 to the server and trigger a renegotiation that requires client-side authentication, e.g. using a client-side certificate)
- Note that the MITM can initiate a renegotiation in step 3 only because he or she knows the proper `verify_data` field value

- The bottom line is that renegotiation attacks remain feasible (even if the TLS `renegotiation_info` extension is used)
- The IETF TLS WG is looking for a renegotiation mechanism that is inherently more secure
  - The most secure possibility is to refuse any change of certificates during renegotiation
  - Another possibility is to make sure that an unknown key-share cannot take place (e.g., if all TLS connections use a distinct und hopefully unique master secret)

```
master_secret =
   PRF(pre_master_secret,"master secret",
       client_random + server_random)
```

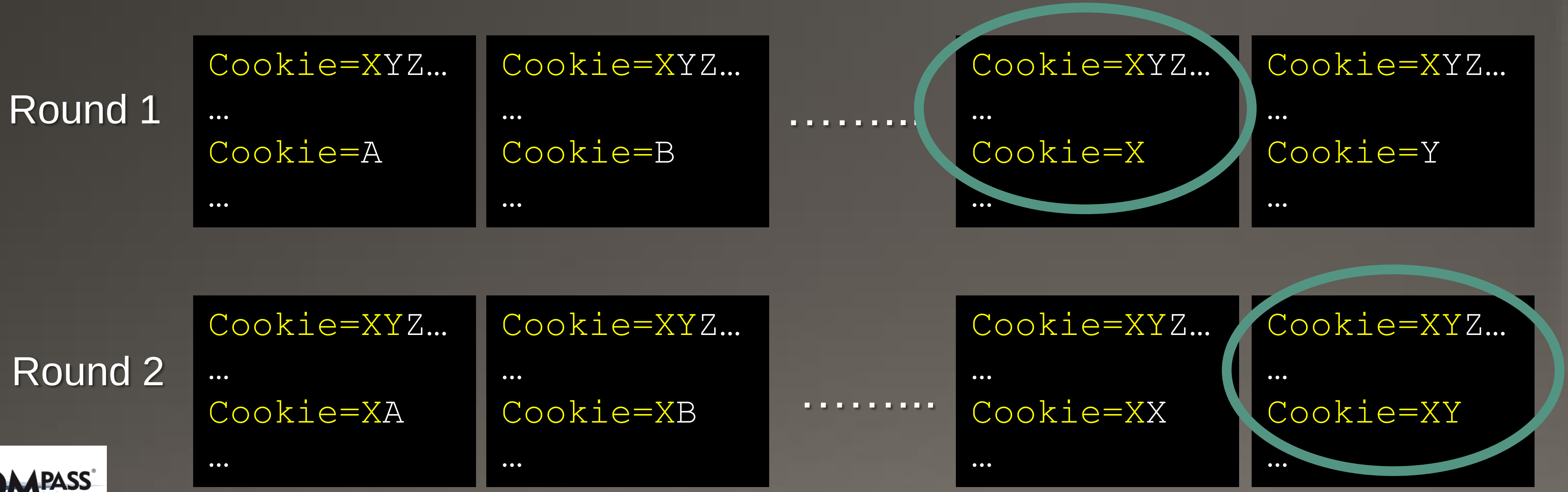
→

```
master_secret =
   PRF(pre_master_secret,"extended master secret",
       session_hash)
```

6. Compression-related Attacks

- In 2002, John Kelsey published a research paper in which he claimed that combining compression and encryption may be dange-rous (in contrast to «normal» intuition)
- Again, this went unnoticed until Rizzo and Duong presented a **Compression Ratio Infoleak Made Easy (CRIME)** attack at the 2012 Ekoparty conference (cf. pp. 158 – 162)
- CRIME effectively turned the vulnerability found by Kelsey into a side-channel attack against the TLS protocol (with compression invoked at the TLS level)
- The side-channel is due to the message size (i.e., differently com-pressed messages have different sizes)

- The DEFLATE compression algorithm (cf. RFC 1951) is widely deployed on the Internet (also for HTTP)
- It combines LZ77 (Jacob Ziv and Abraham Lempel, 1977) and Huffman (David A. Huffman, 1952) encoding
- The CRIME attack targets LZ77
- For each character in the target string, the attack looks for the possibility that compresses most



- The CRIME attack is possible, because
  - the encryption does not hide the message length and
  - each character can be attacked individually
- The attack exploits the properties of LZ77
- Any other compression algorithm may make the attack more difficult or even impossible to mount
- This also applies to Huffman encoding (that is part of DEFLATE)
- The attack can be mitigated by disabling TLS-level compression (i.e., compression method `null`)
- But this does not solve the problem entirely
- If compression is done at the application level (e.g., HTTP), then the effects may be similar

- Early in 2013, Amichai Shulman and Tal Be'ery presented a variant of the CRIME attack named **Timing Info-leak Made Easy (TIME)**

- TIME targets HTTP-level compression and measures the timing of messages (instead of their respective sizes)
- Later in 2013, Neal Harris, Yoel Gluck, and Angelo Prado presented another CRIME variant named **Browser Reconnaissance and Exfiltration via Adaptive Compression of Hypertext (BREACH)**

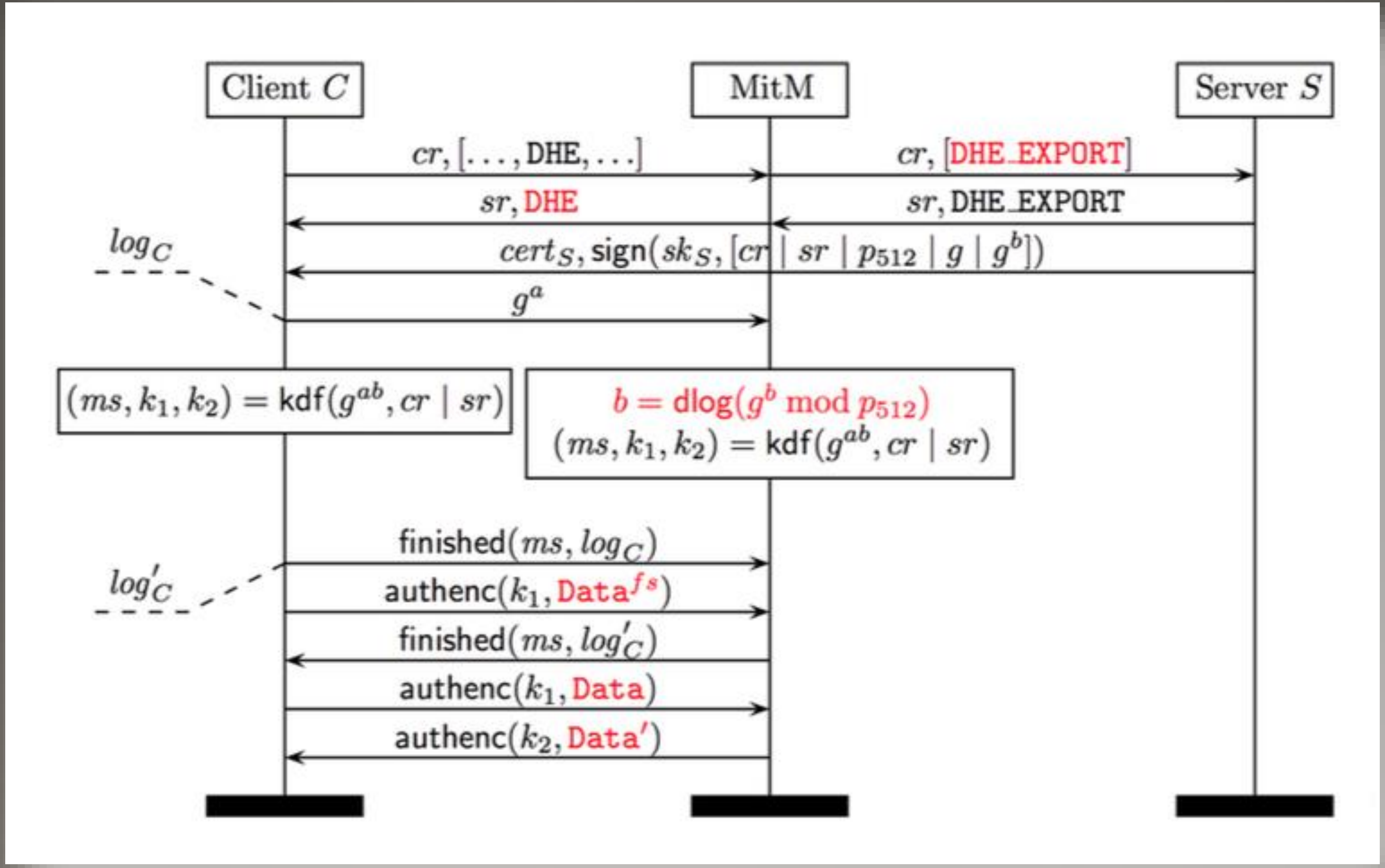
- BREACH targets HTTP-level compression (like TIME) and measures the message sizes (like CRIME)
- In contrast to TIME, BREACH has attracted a lot of media attention

- The bottom line is that compression-related attacks are feasible, and that the combined use of compression and encryption must be considered with care
- It is recommended (from a security perspective)
  - Not to use TLS-level compression at all
  - To use HTTP-level compression only where necessary and appropriate
- Due to the importance of compression in the field, it is possible and likely that many SSL/TLS-based applications are susceptible to compression-related attacks and that many such attacks will be found and published in the future
- Hence, securely combining compression and encryption (in this order) is an important research area

7. Key Exchange Dowgrade Attacks

- There are a few attacks that try to downgrade the key exchange mechanism that is negotiated in the SSL/TLS handshake protocol to something that is cryptographically as weak as possible
- The weakest key exchange mechanisms are the ones that have been used in the 1990s for export reasons (e.g., RSA_EXPORT, DHE_EXPORT, …)
- Consequently, a **key exchange downgrade attack** tries to enforce the use of such a key exchange mechanism in order to break it in the aftermath (cf. pp. 168 – 170)

- The FREAK attack (March 2015) targeted RSA_EXPORT and exploited an implementation bug found in some browsers
- The Logjam attack (May 2015) targeted a DHE_EXPORT (and did not exploit an implementation bug)
- In either case, the attack can be miti-gated by patching the browser and not supporting ex-port-grade crypto-graphy in the first place
- The mere support of such cipher suites has turned out to be dan-gerous


Client C
MitM
Server S
$cr, [\ldots, \text{DHE}, \ldots]$
$cr, [\text{DHE\_EXPORT}]$
sr, DHE
$sr, \text{DHE\_EXPORT}$
log_C
$cert_S, \text{sign}(sk_S, [cr \mid sr \mid p_{512} \mid g \mid g^b])$
g^a
$(ms, k_1, k_2) = \text{kdf}(g^{ab}, cr \mid sr)$
$b = \text{dlog}(g^b \bmod p_{512})$
$(ms, k_1, k_2) = \text{kdf}(g^{ab}, cr \mid sr)$
$\text{finished}(ms, log_C)$
log'_C
$\text{authenc}(k_1, \text{Data}^{fs})$
$\text{finished}(ms, log'_C)$
$\text{authenc}(k_1, \text{Data})$
$\text{authenc}(k_2, \text{Data}')$

8. Concluding Remarks

- Since the SSL/TLS protocols are ready for prime time, they are a popular target to attack (and hence heavily exposed to «cops and robbers» games)
- Many security researchers try to find shortcomings and vulnerabilities that can be exploited in specific attacks
- Many attacks are relevant, but only a few are devastating
- The security of the SSL/TLS protocols remains relative (to the attacks that have been found)
- Nevertheless, SSL/TLS is still a better choice than any new and home-brewed cryptographic protocol
- If such a protocol is designed, it is very likely that similar mistakes are made and that the resulting protcol has similar (if not worse) shortcomings and vulnerabilities

- In addition to the many patches and countermeasures, HTTP strict transport security (HSTS) and TLS 1.3 are going to have a deep impact on the overall security of SSL/TLS
- It is recommended to use (EC)DHE for key exchange mainly to provide forward secrecy
- If elliptic curve cryptography (ECC) is used, then the question what elliptic curve(s) to use is controversially discussed in the community
- If a block cipher is used, then it should be operated in CCM or GCM mode (instead of CBC mode)
- Alternatively, one may consider the use of ChaCha20-Poly1305
- Anyway, the «cops and robbers» game will continue …

Q&A

Thanks for your attention !