SAC Summer School 2016

Implementation and analysis of cryptographic protocols

Part 3: Attacks

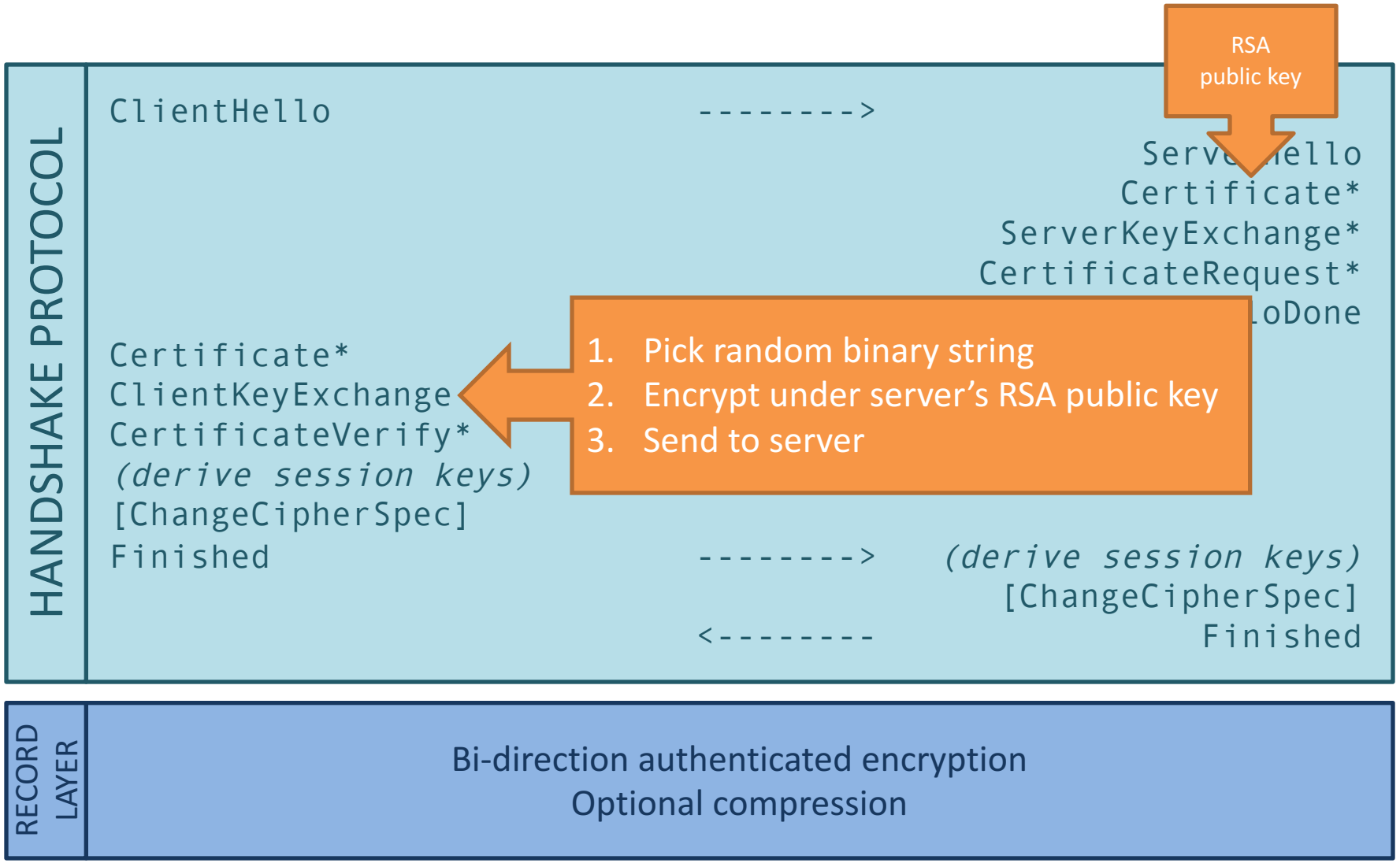
Dr. Douglas Stebila

McMaster University

https://www.douglas.stebila.ca/teaching/sac-2016

BLEICHENBACHER'S ATTACK ON RSA KEY TRANSPORT

RSA key transport



Textbook RSA public key encryption

- Key Generation:
  - Pick primes p, q
  - Compute n = pq
  - Compute $\phi(n) = (p-1)(q-1)$
  - Pick e = 3 or 65537 (for example)
  - Compute $d = e^{-1} \bmod \phi(n)$
  - Public key: (n, e)
  - Private key: (n, d)

Textbook RSA public key encryption

- Encrypt(m, pk = (n,e))
  - Represent m as an integer between 1 and n
  - Compute $c = m^e \bmod n$

Malleability

- Textbook RSA encryption is malleable:
  - c^2 mod n is the encryption of m^2 mod n
- Encryption isn't supposed to provide integrity, but this is still undesirable.
- One solution:
  - Add redundancy or encoding that would be hard to maintain after malleability

PKCS #1 v1.5 padding

- PKCS = Public Key Cryptography Standards
  - Originally created by the RSA company

- Let k be the length of n in bits

- M = 

  where |padding| = k - |msg| - 3
  and every byte of padding is non-zero

PKCS #1 v1.5 encryption/decryption

- Encrypt(m, pk = (n,e))
  - Encode m as M using PKCS#1 v1.5 padding
  - $c = M^e \bmod n$
- Decrypt(c, sk = (n,d))
  - Compute $M = c^d \bmod n$
  - If M is PKCS-conforming, parse and return m
  - Else output error

Bleichenbacher's attack

- Given c and an oracle for deciding if c is PKCS-conforming, find m.

1. Compute $c' = cs^e \bmod n$ for small s
2. If c' is PKCS-conforming, then first 2 bytes of ms mod n are 00 || 02
3. In other words, $2B \leq ms \bmod n < 3B$ where $B=2^{8(k-2)} \bmod n$
4. Repeat with many s to narrow down range of m
5. For 1024-bit N, about 2^{20} oracle queries suffice

PKCS-conformance oracle in SSLv3

Server processing of ClientKeyExchange message in RSA key transport:

1. Compute $m = c^d \bmod n$
2. If m not PKCS-conforming, reject
3. Else, do additional cryptographic operations
   - Includes verifying a MAC
4. If MAC fails, reject; else accept

A modified ciphertext will be rejected either way, but timing provides a way of deciding whether it was at step 2 or step 4.

Defending against Bleichenbacher's attack

1. Make server processing constant time
2. Don't support RSA key transport

Make server processing constant time

1. Generate random premaster secret
2. Receive ciphertext c
3. Decrypt using textbook RSA encryption
4. If PKCS conforming, continue as normal using plaintext
5. If not PKCS confirming, continue use previously generated random premaster secret
6. …
7. If MAC fails, reject; else accept.

Defending against Bleichenbacher's attack

1. Make server processing constant time
   - Hard to get right
   - Meyer et al. USENIX 2014:
     - Timing side channels in OpenSSL, JSSE, Cavium;
     - Error message side channel in JSSE
2. Don't support RSA key transport
   - Can still have problems if same key is used with older protocols that do support RSA key transport
     - Jager et al. CCS 2015: QUIC and TLS 1.3 exploitable with Bleichenbacher oracle from other protocols

BEAST ADAPTIVE CHOSEN PLAINTEXT ATTACK

CBC Mode

- Recall CBC mode encryption:
  - Divide message m into blocks $m_1 \mid m_2 \mid \ldots$
  - $c_1 = E_k(m_1 \text{ XOR } iv)$
  - $c_j = E_k(m_j \text{ XOR } c_{j-1})$

Pre-requisites for the attack

- In HTTPS, the same TLS connection is used for many requests
  - Main HTML page
  - Images
  - CSS
  - …
- In SSLv3 and TLSv1.0, the IV is derived from the master secret
  - Subsequent requests over the same TLS connection use the same IV

Oracle for testing plaintext block

- Adversary observes $c_1 \mid c_2 \mid \ldots \mid c_n$ for unknown plaintext $m_1 \mid m_2 \mid \ldots \mid m_n$
- Adversary wants to know if $m_j = m^*$
- Adversary directs user to send n+1st plaintext block as

  $m_{n+1} = c_{j-1}$ XOR c_n XOR m^*
- => $c_{n+1} = E_k(m_{n+1}$ XOR $c_n)$

  $= E_k(c_{j-1}$ XOR c_n XOR m^* XOR $c_n)$

  $= E_k(c_{j-1}$ XOR $m^*)$

  $= c_j$ iff $m^* = m_j$

Oracle for testing plaintext block

- Adversary can learn if $m_j = m^*$
- If block size is 128 bits, then can test one 128-bit guess with each chosen plaintext query
- Rizzo and Duong's BEAST attack makes this feasible

HTTP requests

- Suppose the adversary can make the victim make an HTTP request to a particular URL, and the cookie gets appended immediately after

- `GET /↵Cookie: s=123456789012345`6

- Corresponds to two 16-byte blocks of AES in CBC mode

First block of 16 bytes:
Entirely **known** to adversary

Second block of 16 bytes:
Entirely **unknown** to adversary

HTTP requests

- Adversary directs to client to request a different URL that has a different split across breaks

- `GET /abcdefghijklmno↵Cookie: s=1`
  `234567890123456`

First block of 16 bytes:
Entirely **known** to adversary

Second block of 16 bytes:
Adversary knows all except for one byte
=> <256 guesses required

HTTP requests

- Adversary directs to client to request a different URL that has a different split across breaks

- `GET /abcdefghijklmn↵Cookie: s=12345678901234`

First block of 16 bytes:
Entirely **known** to adversary

Second block of 16 bytes:
Adversary knows all except for one byte
=> <256 guesses required

Repeat until all target bytes become known

Defending against BEAST attack

- TLS v1.1 and above use explicit IVs, so a new IV is used with each request
- At the time of attack (2011), TLS v1.1 adoption was low
  - Recommended solution: switch to RC4
  - But then RC4 biases became problematic
- Countermeasure: 1/n-1 record splitting
  - Send only 1 byte (+ padding) in first block
  - Then n-1 bytes in next block
  - Has the effect of randomizing the IV in a backward compatible way

CRIME AND BREACH ATTACKS ON COMPRESSION

Symmetric key encryption

A *symmetric key encryption* scheme is a triple of algorithms:

- KeyGen() –> k
- $\text{Enc}_k(m)$ –> c
- $\text{Dec}_k(c)$ –> m

KeyGen and Enc can be probabilistic

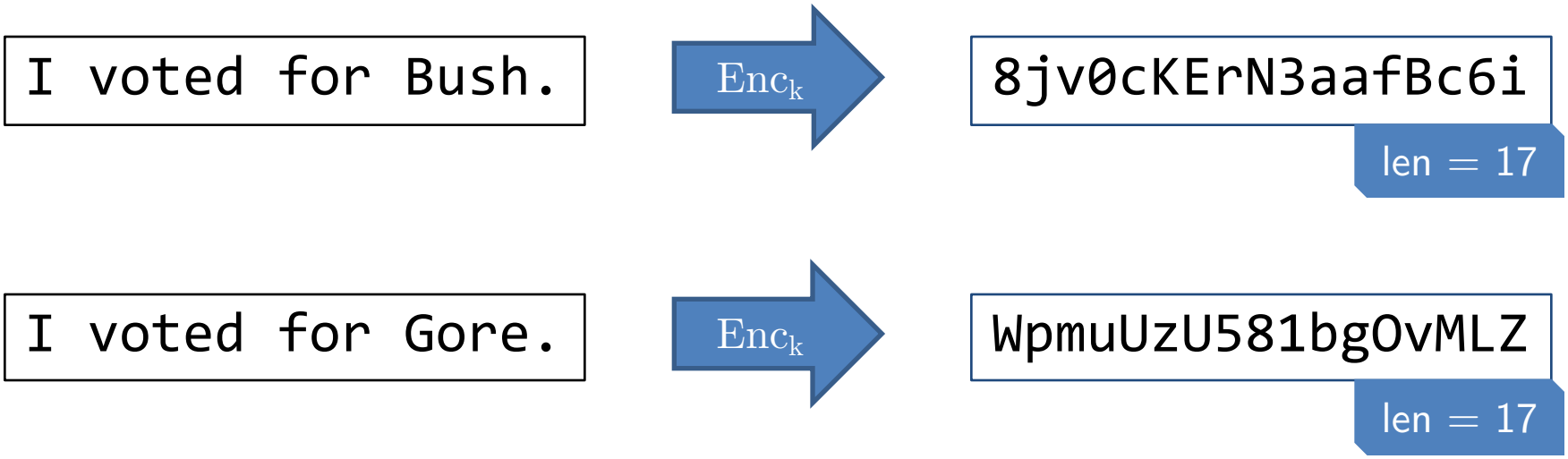
Main security goal:

- **indistinguishability**

Attacker cannot tell apart encryptions of two messages of the same length:

$\text{Enc}_k(m_0)$ looks like $\text{Enc}_k(m_1)$ when $|m_0|=|m_1|$

Symmetric key encryption



same length input => same length output

Compression

A *compression scheme* is a pair of algorithms:

- Comp(*m*) –> *o*
- Decomp(*o*) –> *m*

Comp may be probabilistic (but usually isn't)

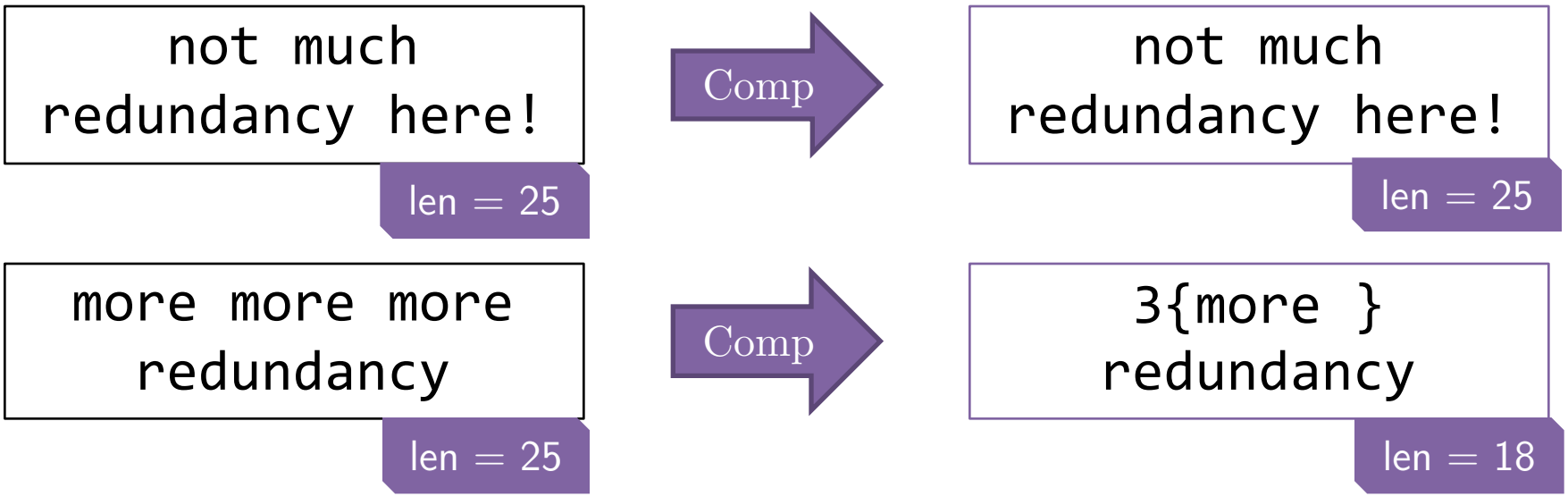
Main security goal:

- **none**

Main functionality goal:

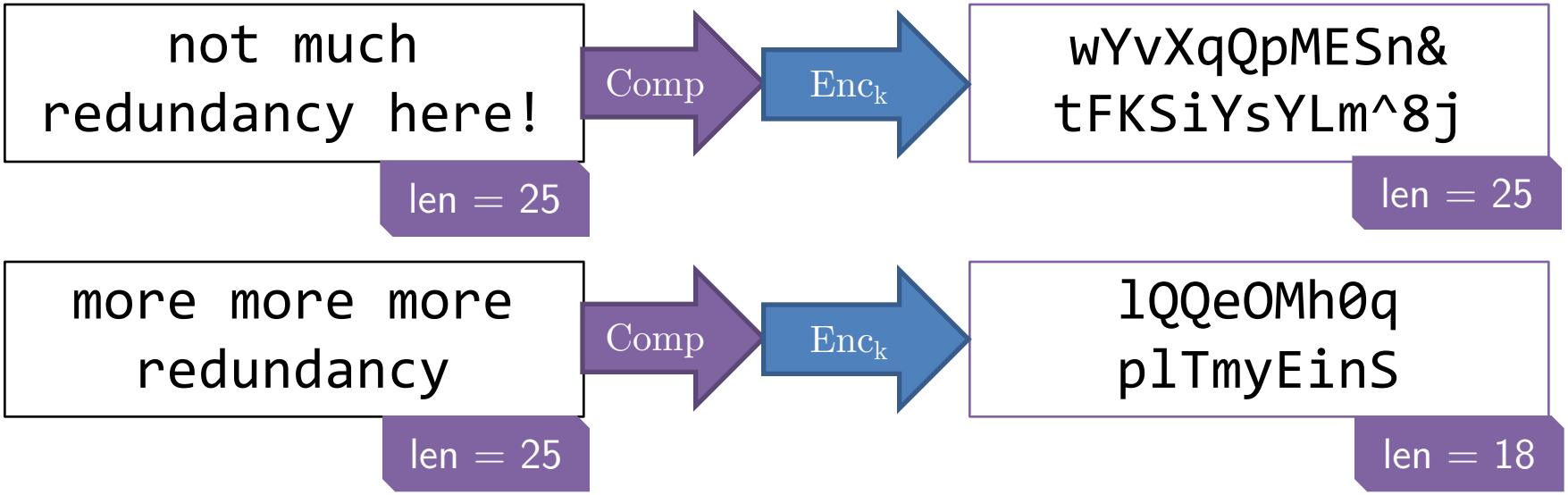
- |Comp(*m*)| << |*m*| for common distribution of *m*
- Can't be true for all *m* due to Shannon's theorem

Compression



same length input => possibly different length output

Compression then encryption

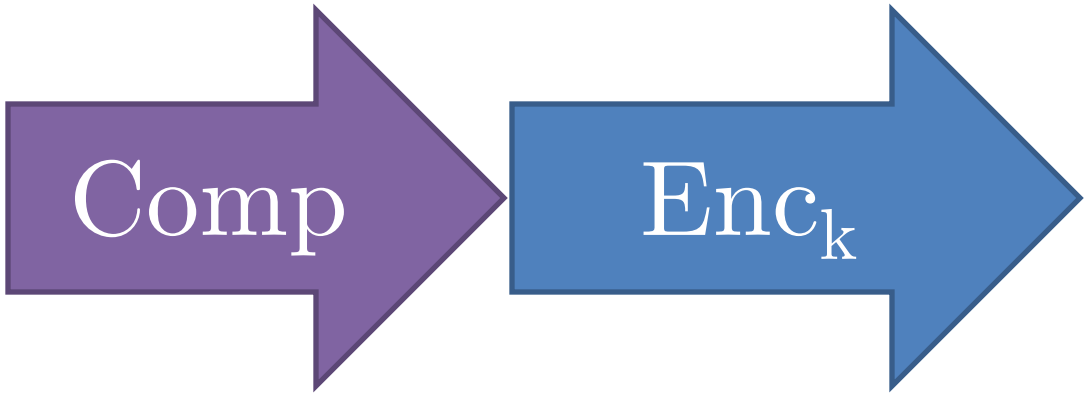


same length input => possibly different length output

A test

Man. U. 2005-2014
lost
lost
WON!
WON!
lost
WON!
lost
WON!
lost

Arsenal 2005-2014
lost
lost
lost
lost
lost
lost
lost
lost
lost
lost


Comp
Enc$_k$

Which ciphertext is for which message?

```
yI5pDrFhPk3
15Cmymr6xCb
LTVEAx
```

```
D1fAGUR1zqv
lhXdX3c8qd+
BYBwK6dAnoG
GQGCmvFIM9/
s6WJjgr2
```

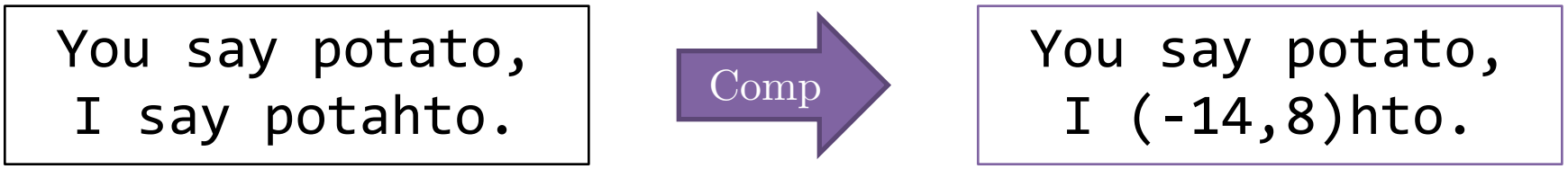
One message compresses more

Arsenal
2005-2014
10{lost }

Man. U.
2005-2014
2{lost }
2{WON! }
3{WON! lost }

Deflate (LZ77) compression algorithm

- Replaces repeated strings with back references (distance, length) to previous occurrence.



- Important parameter: **window size**
  - How far back does it go to search for occurrences?
  - a.k.a. dictionary size

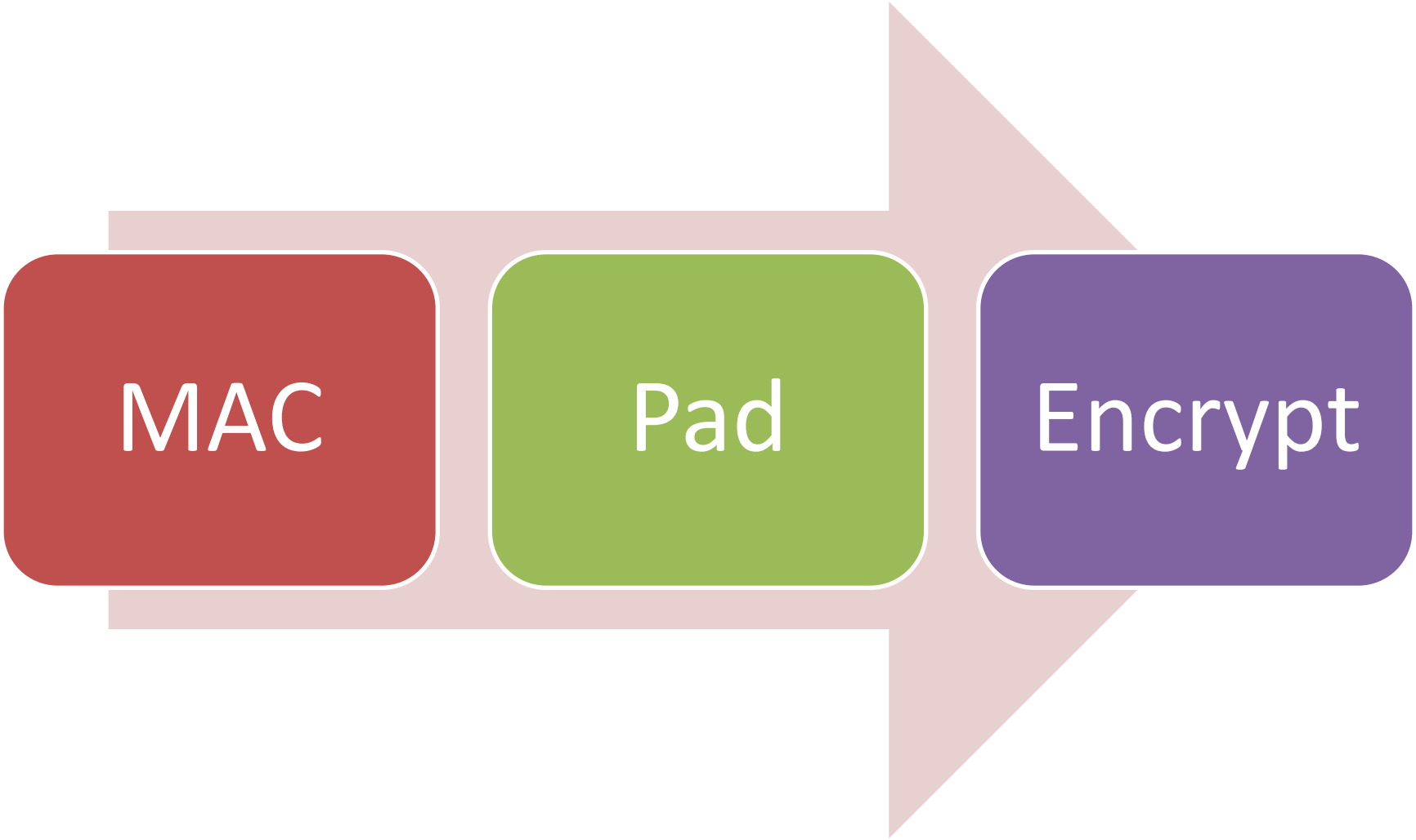
Combining user secrets + adversary input

- Suppose you have a secret
- and combine it with adversarial data
- then compress and encrypt

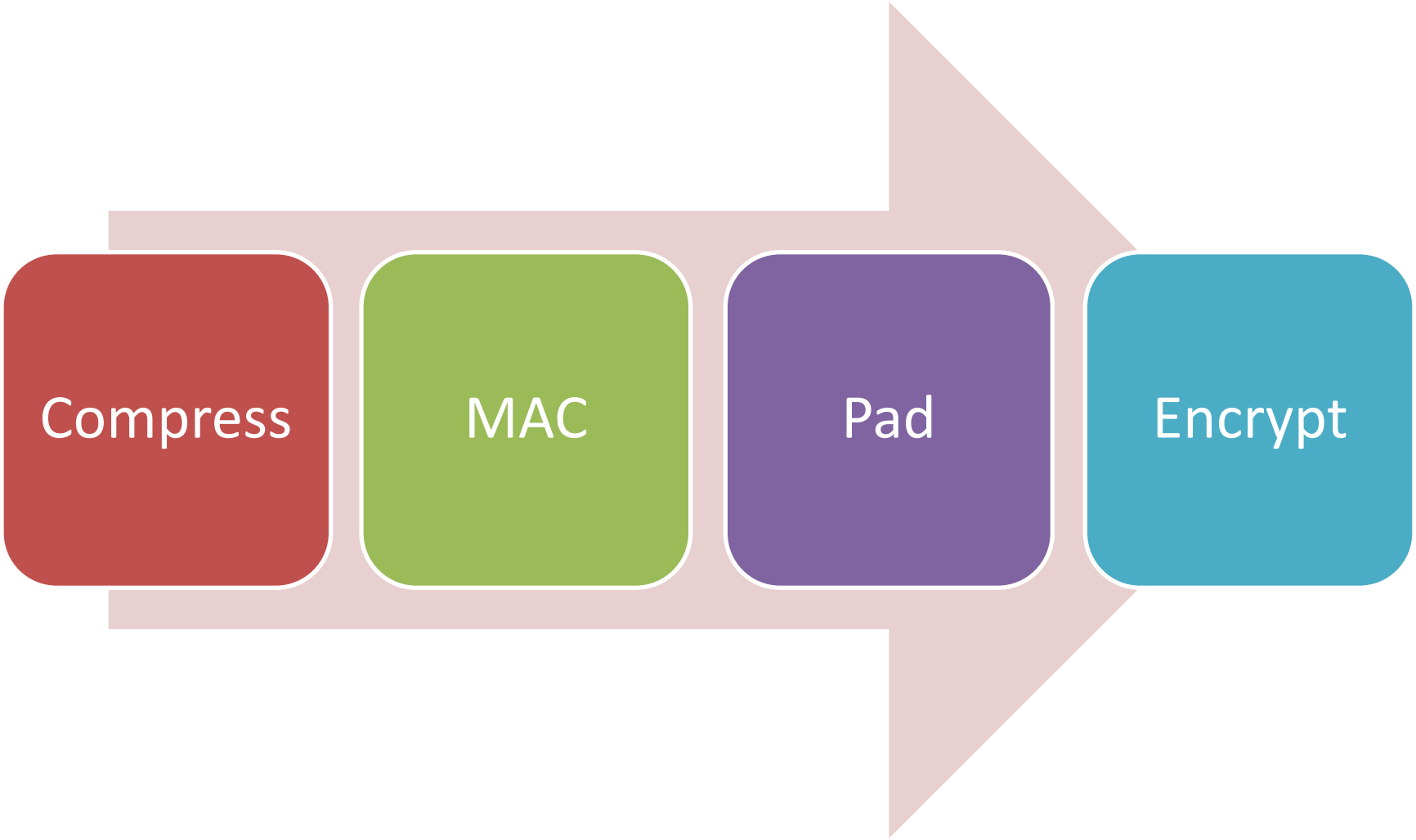
- **Adaptive** attacker can use this to learn your secret

CRIME ATTACK ON COMPRESSION IN TLS

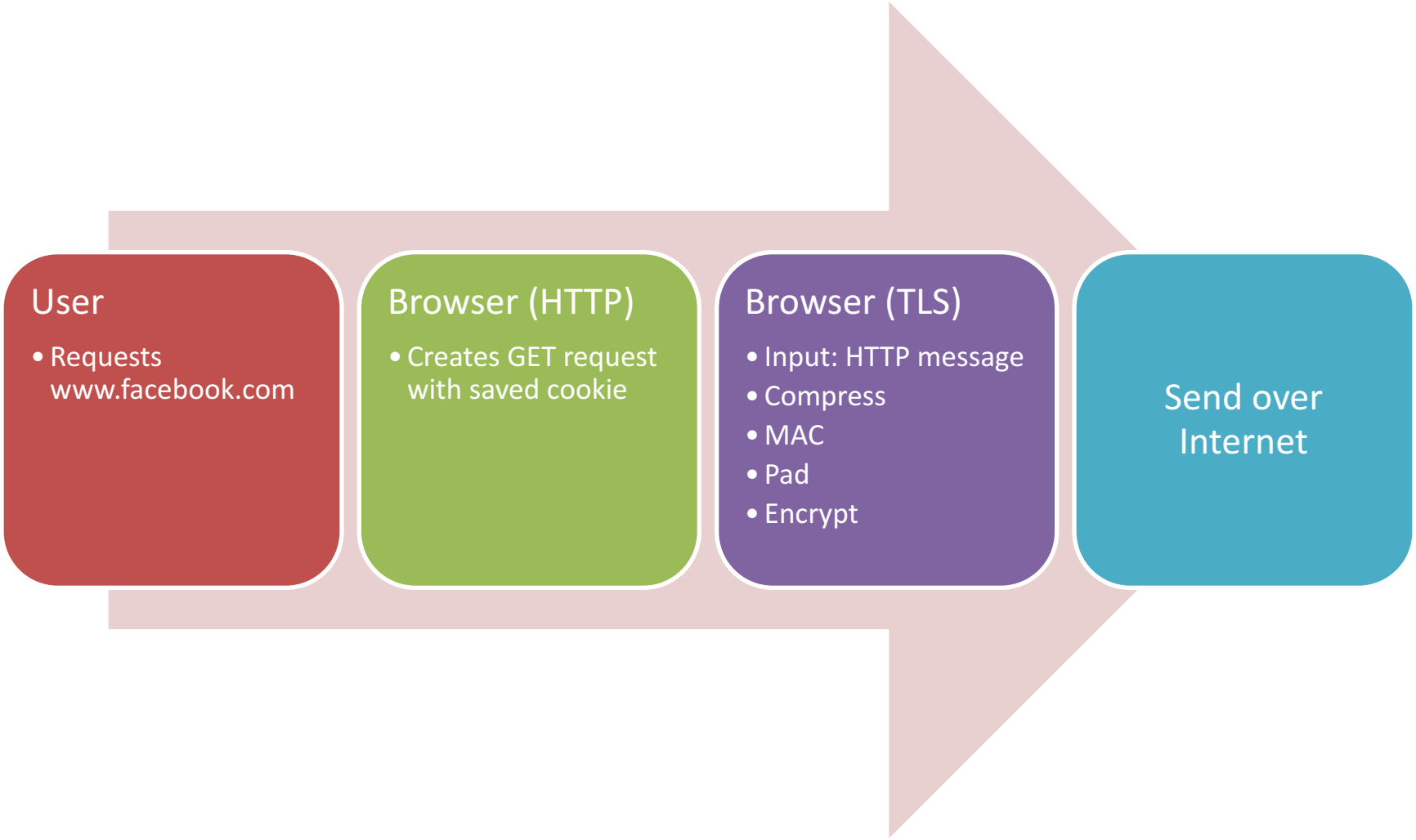
TLS record layer



Compression in TLS record layer



Transmitting an HTTP request



Secret values in HTTP documents

GE / The URL can be adversary-supplied data

Host: www.facebook.com

User-Agent: Mozilla/5.0 (Macintosh; Intel Mac OS X 10.10; rv:34.0) Gecko/20100101 Firefox/34.0

Accept: text/html,a lication/xml;q=0.9,*/*;q=0.8

This secret cookie identifies my session to Facebook

Accept-Lang

Accept-Enco

DNT: 1

Cookie: datr=DzK9VBnObWDqfL7XLwGSSEsu;

reg_fb_ref=https%3A%2F%2Fwww.facebook.com%2F;

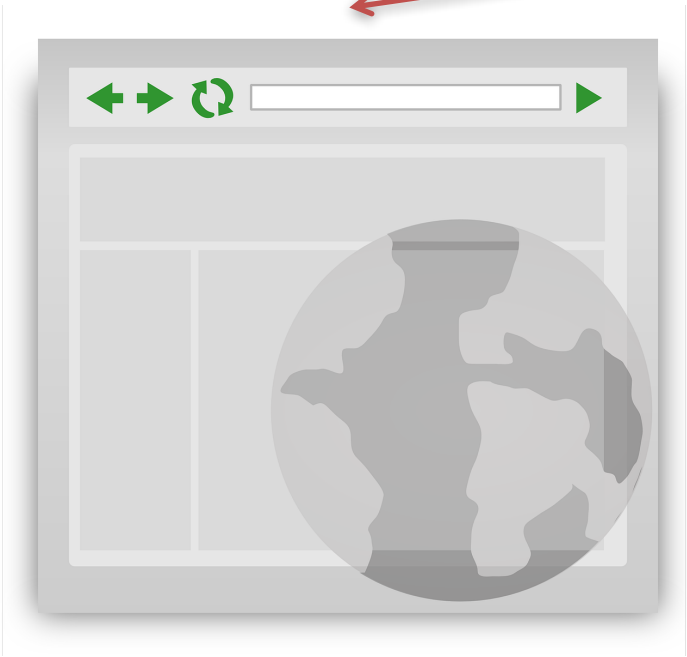
reg_fb_gate=https%3A%2F%2Fwww.facebook.com%2F; dpr=2

Connection: keep-alive

Cache-Control: max-age=0

Attack

Please send a GET request for
`https://www.facebook.com/?datr=A`



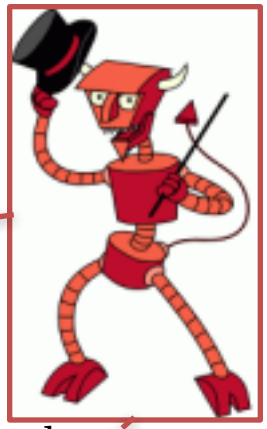
Attack

Please send a GET request for
`https://www.facebook.com/?datr=A`


```
GET /?datr=A
Host: www.facebook.com
Cookie: datr=DzK9VBnObW
DqfL7XLwGSSEsu
...
```

Attack

Please send a GET request for
`https://www.facebook.com/?datr=A`



Observes compressed
& encrypted request

```
VGytgpDn/1Ym5oCdB3Vh2
D5EmdjLRdkx7tEvKG43WJ
yD++cx8CJlBbetQejiXLX
+oQO9bnUMYQwtglOSf9bf
oyWJkYxHsKfqYNqWAfCIg
8U5BK92Ayvk858MJOnTuK
```

len = 204

Attack

Please send a GET request for
https://www.facebook.com/?datr=B


```
GET /?datr=B
Host: www.facebook.com
Cookie: datr=DzK9VBnObW
DqfL7XLwGSSEsu
...
```

Attack

Please send a GET request for
`https://www.facebook.com/?dat=B`

Observes compressed
& encrypted request


```
UQ5ItQ1Y4BVCy37Fhu5K4
hyre7l5P4pWwAYfvnzg9m
R5Qq250PF1yQpf83AFJ34
QS+9BPjUnBzVGENe15r29
rY9tRfIFAdE8ecEmVTFtl
zHy+8EIwxDK67rxM29clJ
```

len = 204

Attack

Please send a GET request for
`https://www.facebook.com/?dat=C`

Observes compressed
& encrypted request

```
Wdb42n0LeQbVweAoiCZxE
j9O0U+qaGPPbe9Sebz2Dx
GhYWj9U4X0cKYyBpTSpB4
4dOqd4DpCscHEsBdg0p6q
DXiSBJ+MLOKbpRvAAmPhy
9Sn9VPnsHgKyB4I1lgCKA
```

len = 204

Attack

Please send a GET request for
`https://www.facebook.com/?dat=D`

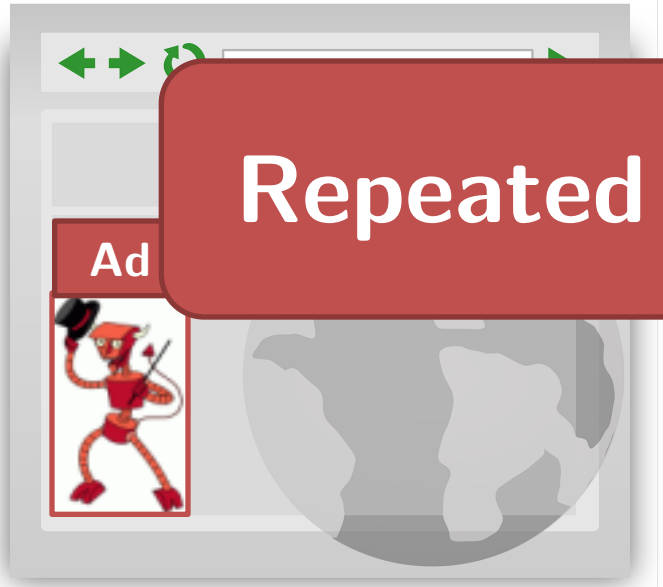
Observes compressed
& encrypted request

```
O8Gb8JwSuoNrcQ7190KSs
nM7n22lOtByzmvv555ZP+
+4lNW2wIuRrTF6KlKdjOB
425VVDUbKKdHNF9YaaxTy
lVWBVo1ApZ4PTSnB1J0pt
jAsecGXjRXOXTwye
```

len = 199

Attack

Please send a GET request for
https://www.facebook.com/ datr=D



Repeated text => compression

```
GET / datr=D
Host: www.facebook.com
Cookie: datr=DzK9VBnObW
DqfL7XLwGSSEsu
...
```

Attack

Please send a GET request for
`https://www.facebook.com/?dat=Da`

Observes compressed
& encrypted request

```
Ok3MV18blnYFIjz2tcucQ
x2mJ8MLULVqMSYO9Lo1r0
wxwjEG8pLwaPaVtrnf46l
ypdqbYQ22oJw63ixkS1HR
QVfz8UKs9tOhPvTAwUiwS
yukxrKq9x9I+3fO8lv8aU
```

len = 205

CRIME attack on TLS "Compression Ratio Info-leak Made Easy"

- Rizzo and Duong [ekoparty 2012]

- Victim visits adversary-controlled page
- Adversarial Javascript causes browser to make many requests
- Figure out 1st letter of cookie
- Figure out 2nd letter of cookie
- Figure out 3rd letter of cookie
- …

A few tricky bits to make it work in TLS:

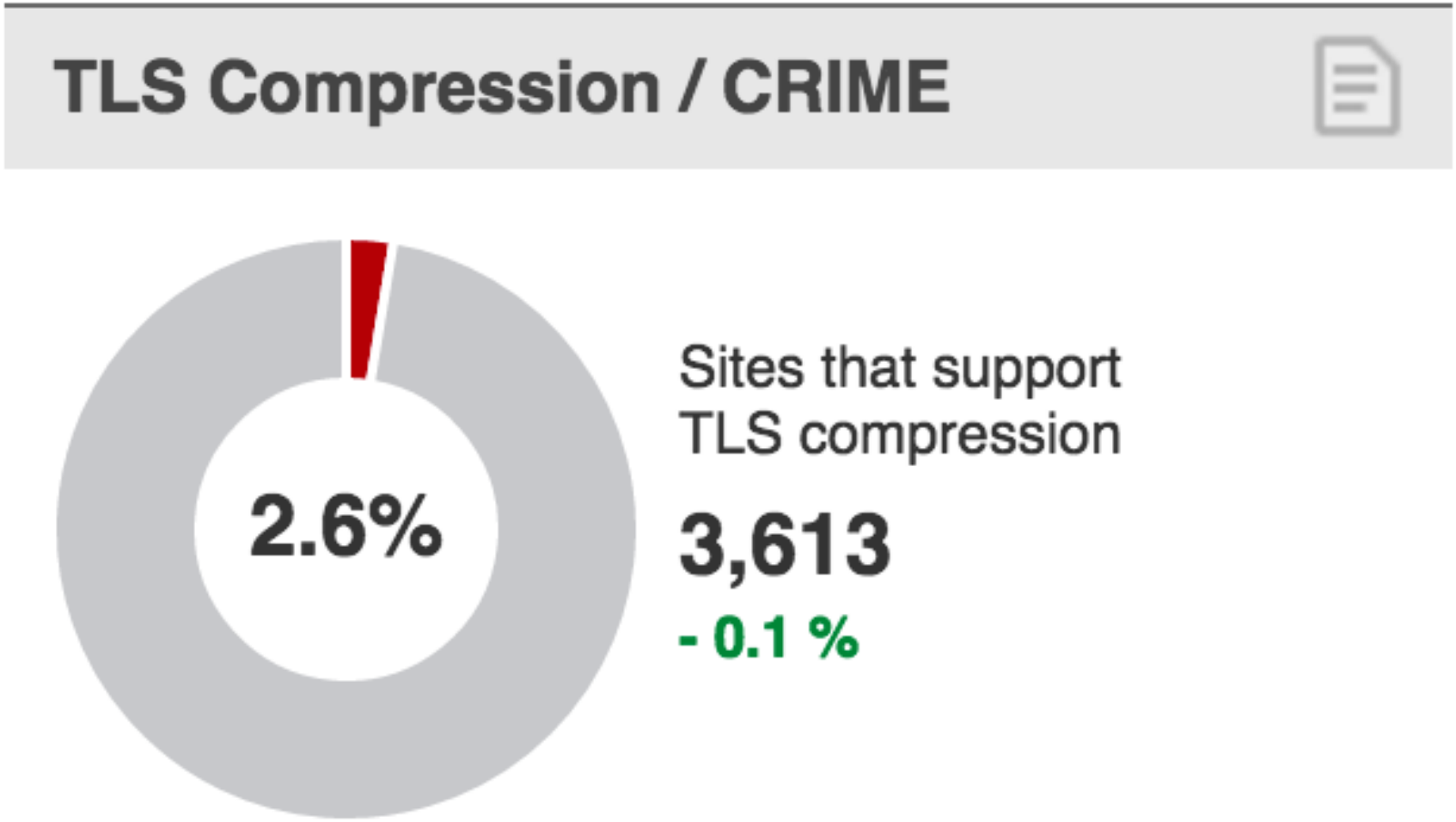
- TLS splits plaintext into 16K records then compresses and encrypts each record separately
- Need to ensure that you can observe length differences based on compression

- But it can be made to work!

CRIME wasn’t new

- Kelsey [FSE 2002] theorized length-based attacks on compression-encryption with adversary-chosen prefix.

Impact of CRIME attack



August 3, 2016 ▪ https://www.trustworthyinternet.org/ssl-pulse/

But…

- Compression is present elsewhere on the Internet.

- HTTP allows gzip compression of the body

BREACH ATTACK ON COMPRESSION IN HTTP

BREACH attack

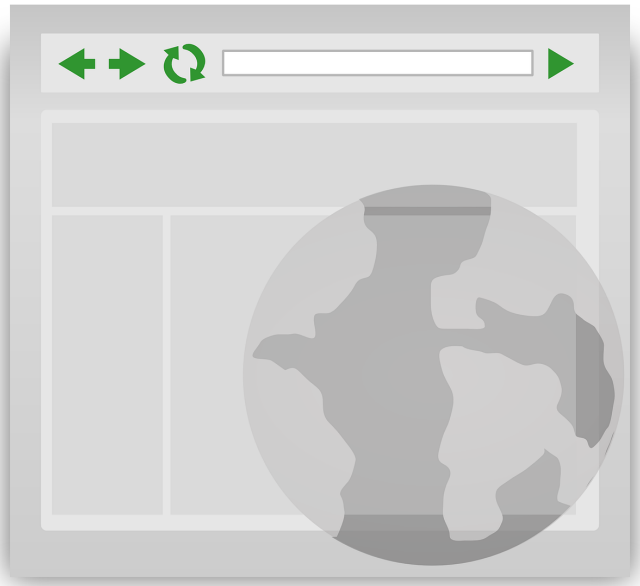
- Attack against HTTP compression hypothesized in CRIME presentation

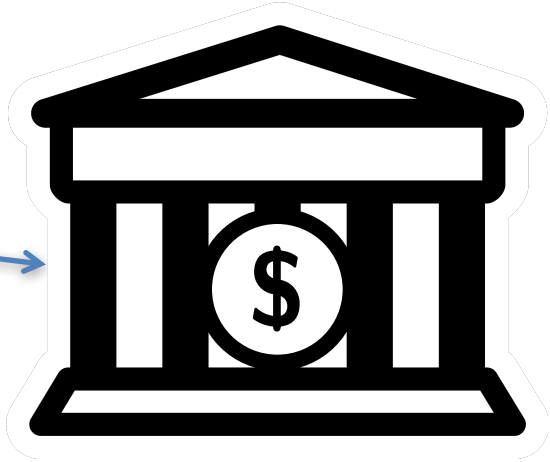
"Browser Reconnaissance and Exfiltration via Adaptive Compression of Hypertext"

- attack demonstrated against secrets in HTML
- Gluck, Harris, Prado [Black Hat 2013]

Cross-site request f y

Please send a GET request for
`https://www.bank.com/transfer`
`?to=Eve&amount=1000000`





```
GET /transfer?to=Eve
&amount=1000000
Host: www.bank.com
Cookie: account=Alice
...
```

Anti-CSRF tokens

Protection strategy: server hides a random token in each HTML form it creates and will only execute action if received response contains that token.

```
<form
action="/money_transfer"
method="post">
<input type="hidden"
name="csrftoken"

value="OWT4NmQlODE4ODRjN2Q1
NTlhMmZlYWE...">
...
</form>
```

BREACH Attack

Works against websites that echo user input in the same page as a valuable secret (e.g., anti-CSRF token).

- combining user secrets + adversary input then compressing

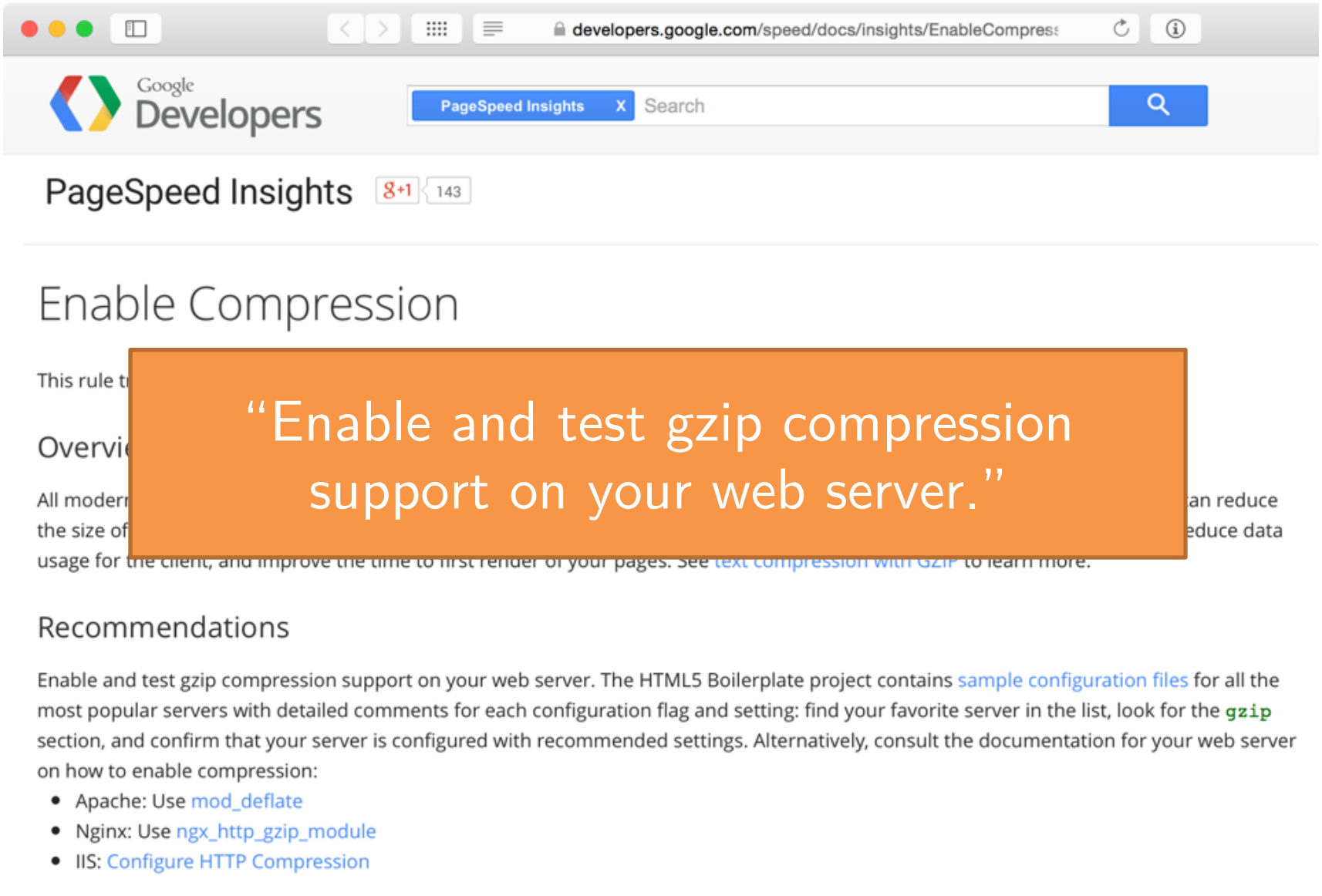
```
<p>Welcome,
<?=$_GET['username']?>.</p>
<form
action="/money_transfer"
method="post">
<input type="hidden"
name="csrftoken"

value="OWT4NmQlODE4ODRjN2Q1
NTlhMmZlYWE...">
...
</form>
```

Recommendations from BREACH attack

1. Disabling HTTP compression
2. Separating secrets from user input
3. Randomizing secrets per request
4. Masking secrets (effectively randomizing by XORing with a random nonce)
5. Length hiding (by adding a random number of bytes to the responses)
6. Rate-limiting the requests

Impact of BREACH attack



Compression in network protocols

HTTP/1.1

- supports compression
- BREACH attack
- still widely used

SPDY

- supports compression
- CRIME/BREACH work against early versions

HTTP/2

- separate compression of every headers
- uses special algorithm HPACK for header compression

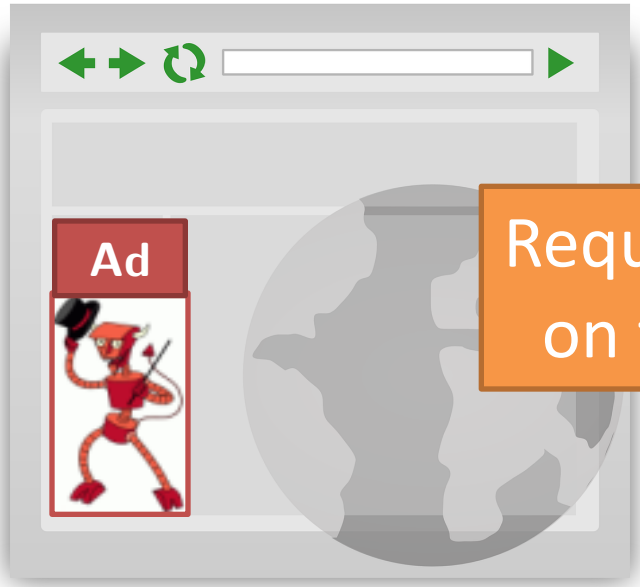
Others

- SSH
- PPTP
- OpenVPN
- XMPP
- IMAP
- SMTP

CRIME and BREACH Attacks

Please send a GET request for
`https://www.facebook.com/?datr=A`

Observes compressed & encrypted request



Requires attacker to be on the network path

```
VGytgpDn/1Ym5oCdB3Vh2
D5EmdjLRdkx7tEvKG43WJ
yD++cx8CJlBbetQejiXLX
+oQO9bnUMYQwtglOSf9bf
oyWJkYxHsKfqYNqWAfCIg
8U5BK92Ayvk858MJOnTuK
```

len = 204

The HEIST attack

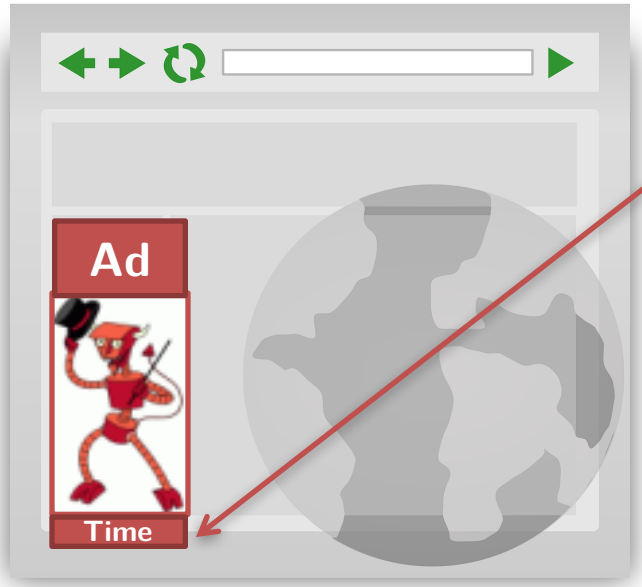
- “HTTP Encrypted Information can be Stolen through TCP windows”
- Just published at BlackHat 2016 last week

- https://www.blackhat.com/docs/us-16/materials/us-16-VanGoethem-HEIST-HTTP-Encrypted-Information-Can-Be-Stolen-Through-TCP-Windows-wp.pdf

HEIST attack

Please send a GET request for
`https://www.facebook.com/?datr=A`

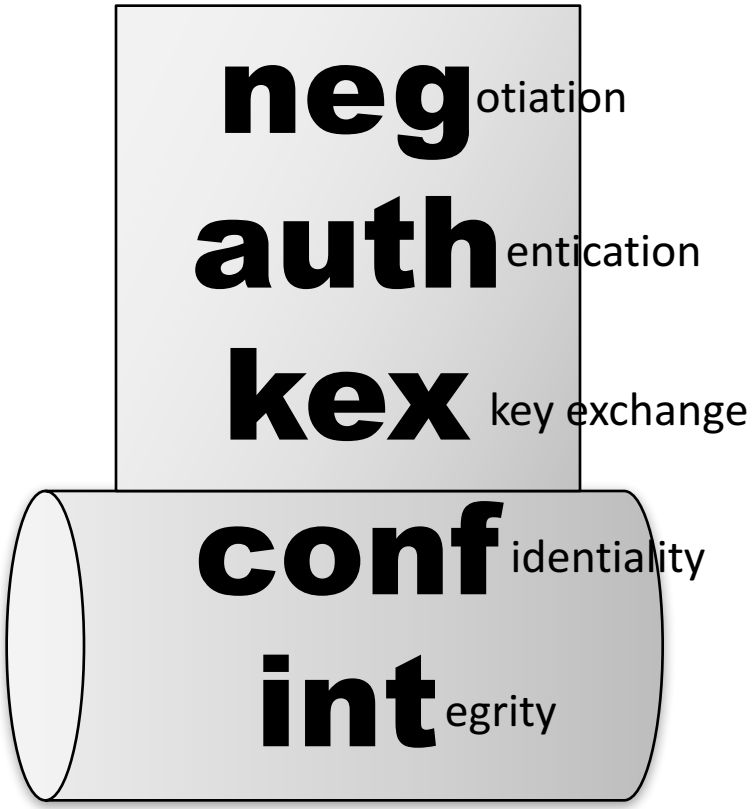
Tell me when you are done loading the request (Javascript Resource Timing API)



Doesn't require attacker to be on the network path

CROSS-CIPHERSUITE ATTACK

Security goals of TLS

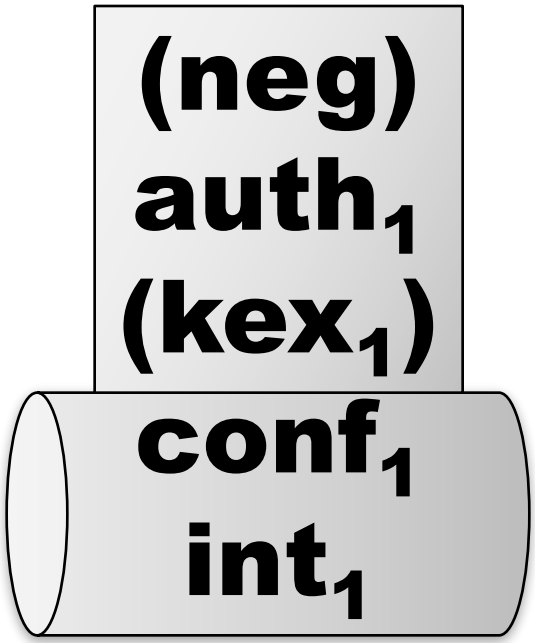


From an application perspective, TLS provides:

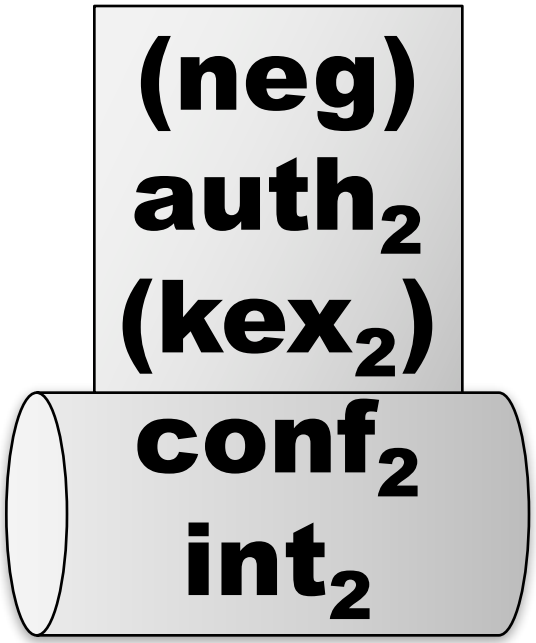
- negotiation of parameters
- **entity authentication**
- key exchange
- confidentiality and integrity of messages

How we'd like to analyze ciphersuites

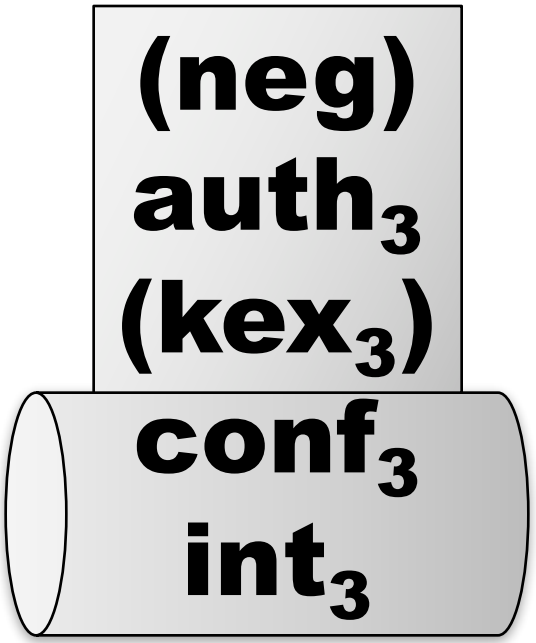
ciphersuite 1



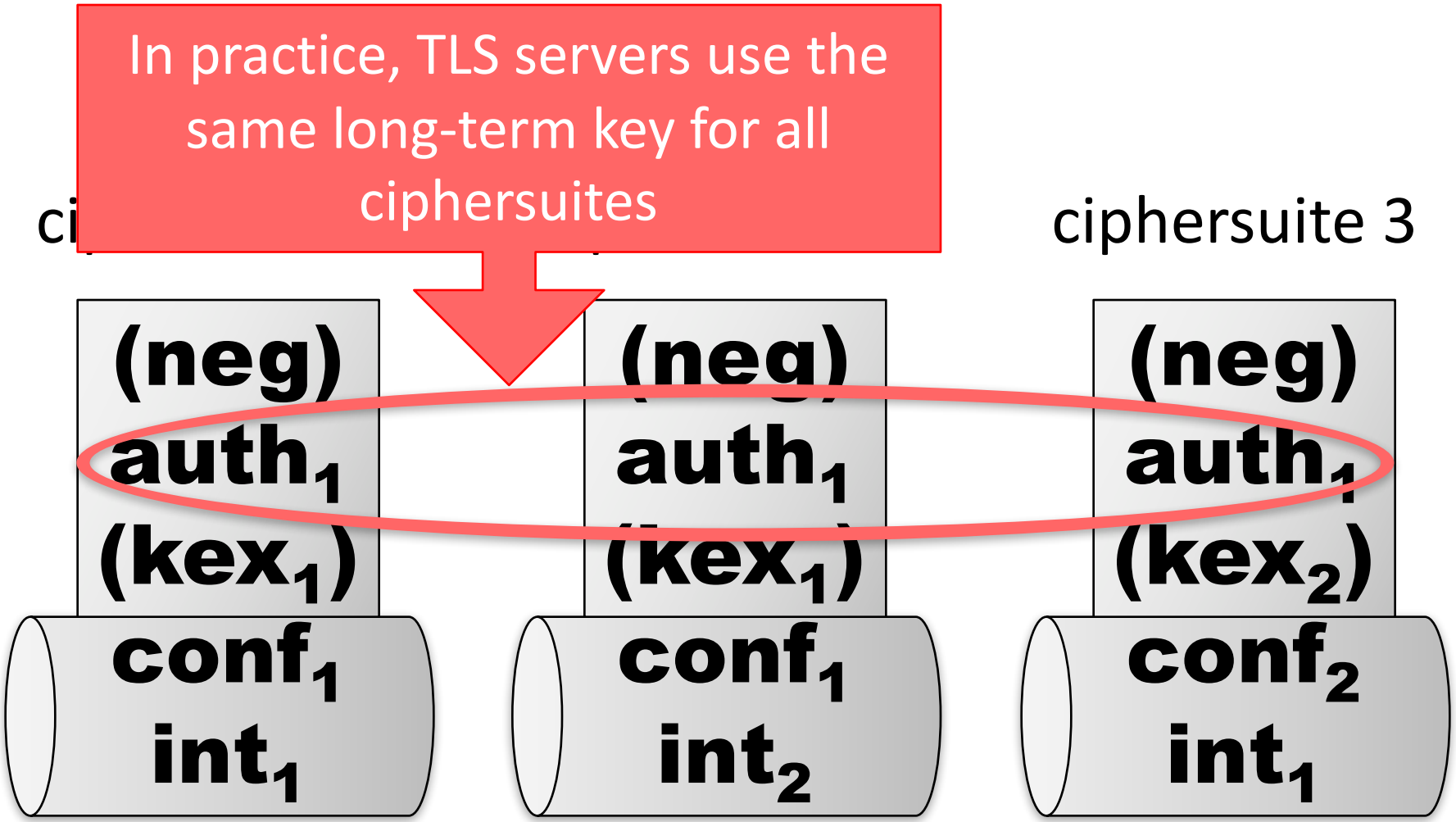
ciphersuite 2



ciphersuite 3



The reality of multi-ciphersuite usage



Long-term key reuse across ciphersuites

Is this secure?

Even if a ciphersuite is secure on its own, it may not be secure if the long-term key is shared between two ciphersuites.

Cross-ciphersuite attack

Mavrogiannopoulos et al. CCS 2012[MVVP12]

(built on observation of Wagner & Schneier 1996)

```
struct {
  select (KeyExchangeAlgorithm):
    case dhe_dss:
    case dhe_rsa:
      ServerDHParams params;
      digitally-signed struct {
        opaque client_random[32];
        opaque server_random[32];
        ServerDHParams params;
      } signed_params;
    case ec_diffie_hellman:
      ServerECDHParams params;
      digitally-signed struct {
        opaque client_random[32];
        opaque server_random[32];
        ServerECDHParams params;
      } signed_params;
} ServerKeyExchange
```

```
struct {
  opaque dh_p<1..2^16-1>;
  opaque dh_g<1..2^16-1>;
  opaque dh_Ys<1..2^16-1>;
} ServerDHParams;
```

```
struct {
  ECCurveType curve_type = explicit_prime(1);
  opaque      prime_p <1..2^8-1>;
  ECCurve     curve;
  ECPoint     base;
  opaque      order <1..2^8-1>;
  opaque      cofactor <1..2^8-1>;
  opaque      point <1..2^8-1>;
} ServerECDHParams;
```

1. No "type" information.

2. Some valid ServerECDHParams binary strings are also valid WEAK ServerDHParams binary strings.

[MVVP12] Cross-ciphersuite attack

(built on observation of Wagner & Schneier 1996)

=> TLS not secure with long-term key reuse.

=> Security of a ciphersuite in isolation does not imply security with long-term key reuse.

RENEGOTIATION ATTACK

Why renegotiate?

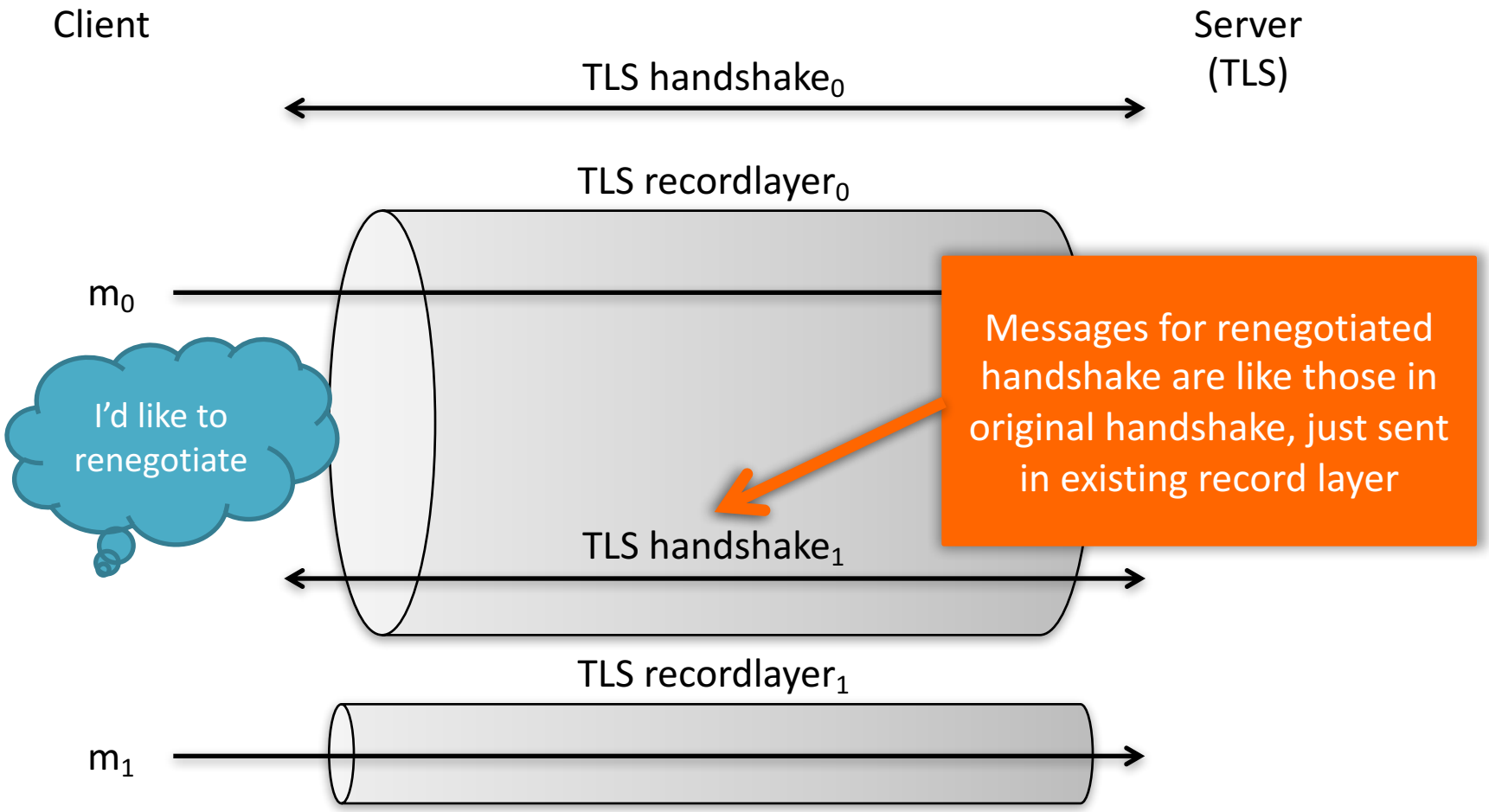
Renegotiation allows parties in an established TLS channel to create a new TLS channel that continues from the existing one.

Once you've established a TLS channel, why would you ever want to renegotiate it?

- Change cryptographic parameters
- Change authentication credentials
- Identity hiding for client
  - second handshake messages sent encrypted under first record layer
- Refresh encryption keys
  - more forward secrecy
  - record layer has maximum number of encryptions per session key

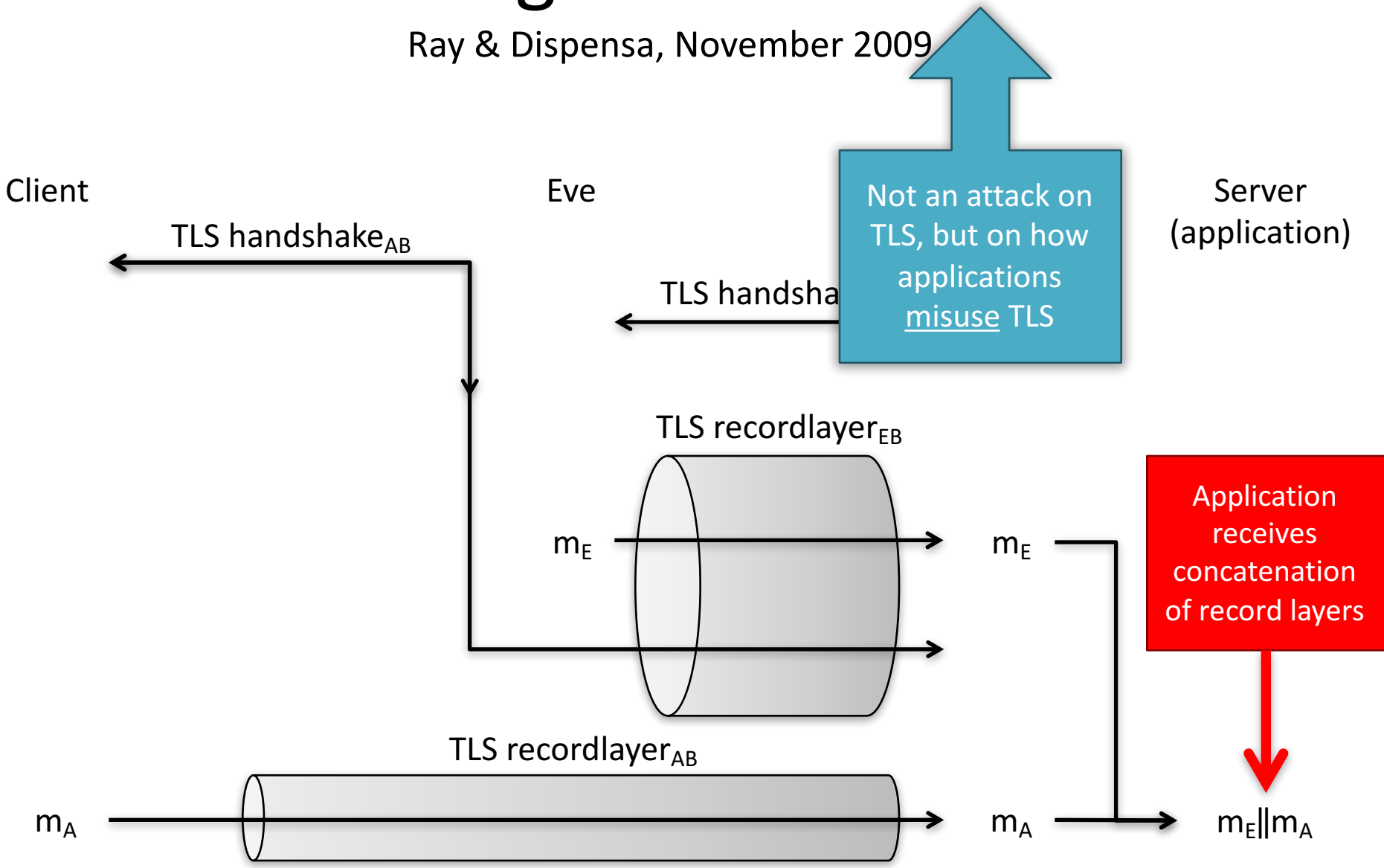
Renegotiation in TLS

(pre-November 2009)



TLS Renegotiation "Attack"

Ray & Dispensa, November 2009



Example: HTTP Injection

- Attacker sends
  - m_E = "GET /orderPizza?deliverTo=123-Fake-St↩
    X-Ignore-This: "
- Client sends
  - m_A = "GET /orderPizza?deliverTo=456-Real-St↩
    Cookie: Account=1A2B"
- Server's web server receives
  - $m_E \| m_A$ = "GET /orderPizza?deliverTo=123-Fake-St↩
    X-Ignore-This: GET /orderPizza?deliverTo=456-Real-St↩
    Cookie: Account=1A2B"

X-Ignore-This: is an invalid header, so the rest of that line gets ignored.

The server's GET request is processed with the cookie supplied by the client.

Renegotiation security

Q: What property should a secure renegotiable protocol have?

A: Whenever two parties successfully renegotiate, they are assured they have the exact same view of everything that happened previously.

- Every time we accept, we have a matching conversation of previous handshakes and record layers.

TLS Renegotiation Countermeasures

Two related countermeasures standardized by IETF in RFC 5746:

1. Signalling Ciphersuite Value
2. Renegotiation Indication Extension

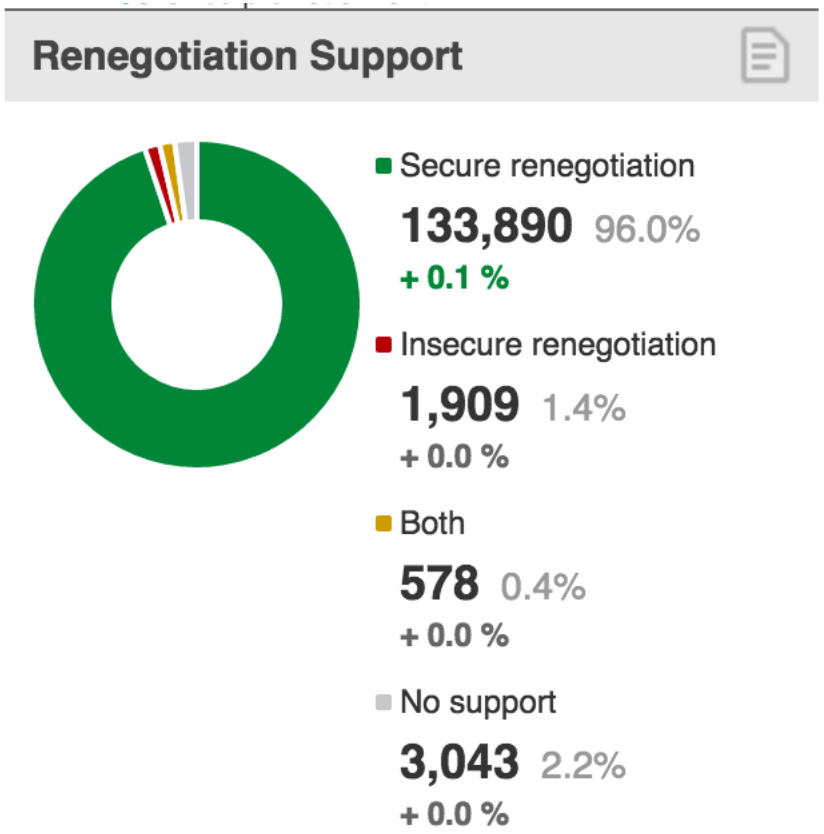
Basic idea: include fingerprint of previous handshake when renegotiating.

- Note: This is a "white-box" modification of TLS.

TLS Renegotiation Countermeasures

SCSV/RIE fairly quickly and widely adopted.

Currently 96% deployment
(SSL Pulse, August 3, 2016)



LOGJAM ATTACK

Export ciphersuites

- Early versions of SSL and TLS included **export** ciphersuites, which included weak (512-bit) RSA and Diffie-Hellman
- Recall: TLS ephemeral DH is signed Diffie-Hellman
  - But signature only on a subset of the request (nonces + server public key)
  - Transcript authentication comes from a MAC under the master secret derived from the DH shared secret

Logjam attack idea

1. MITM modifies client request to server to request export signed-DH ciphersuite
   - If adversary just relays this back, the client won't accept, because the transcripts won't match and the MAC will fail
2. MITM receives 2048-bit RSA signature on 512-bit finite field DH key
3. MITM computes discrete log on 512-bit public key
4. MITM computes DH shared secret
5. MITM computes MAC on transcript the client expects
6. MITM completes handshake with client

Export ciphersuites

- Most modern TLS clients and servers don't support export ciphersuites
- But around 3-8% of HTTPS servers did (2015)
- And some modern TLS clients would support small groups even in non-export ciphersuites

Logjam attack

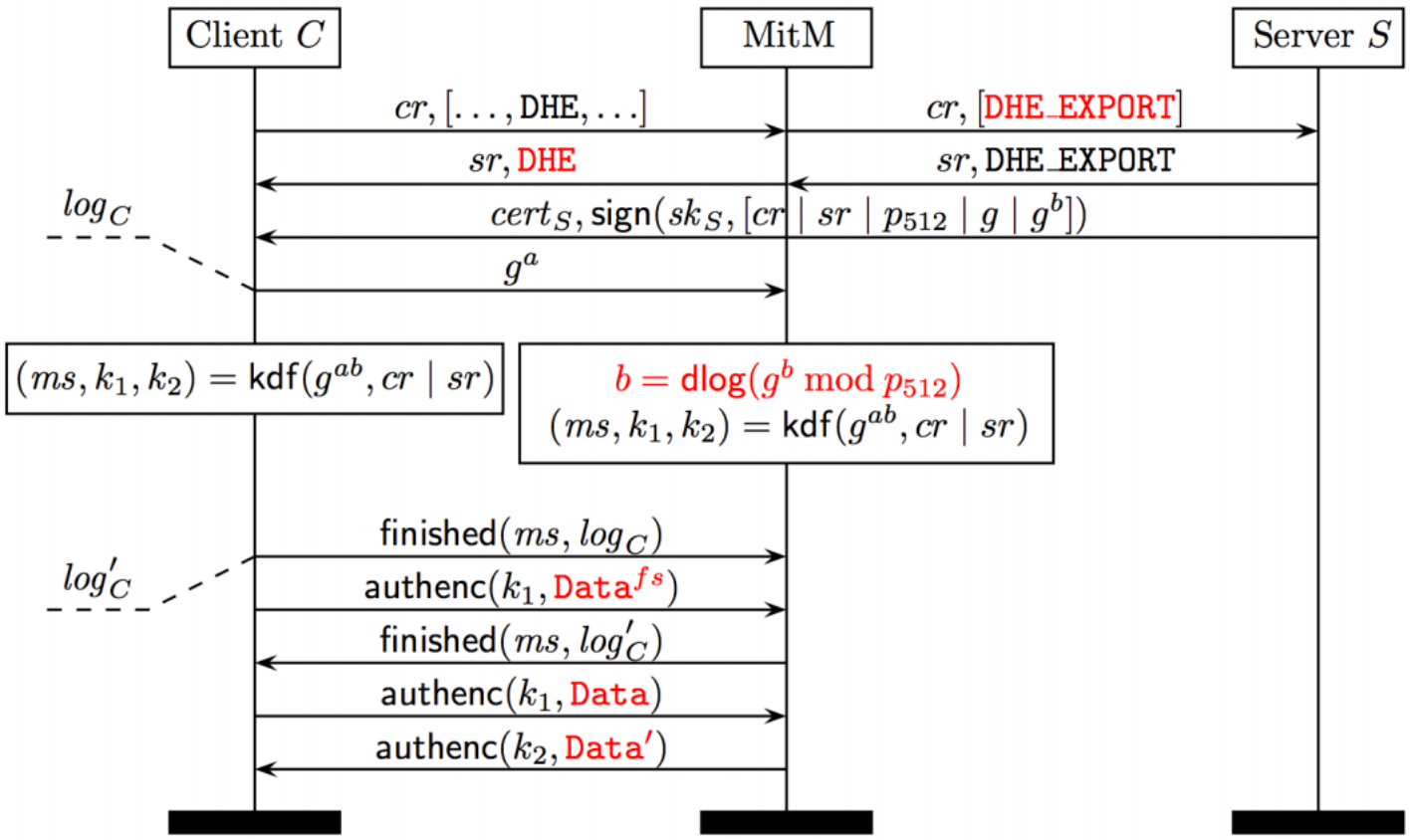


Figure from Adrian et al. CCS 2015.

IPsec

- Idea also applies to IPsec
- Many more IPsec servers support weak DH groups

How quickly can you compute discrete logarithms?

- 92% of vulnerable servers used one of two standardized 512-bit groups
- With one week of precomputation, can then compute individual discrete logs in about 1 minute

- Can you extend the technique to 768- or 1024-bit groups?

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-Béguelin[‖] Paul Zimmermann[†]

[*] INRIA Paris-Rocquencourt [†] INRIA Nancy-Grand Est, CNRS, and Université de Lorraine
[‖] Microsoft Research [‡] University of Pennsylvania [§] Johns Hopkins [¶] University of Michigan

4.2 Is NSA Breaking 1024-bit DH?

Our calculations suggest that it is plausibly within NSA's resources to have performed number field sieve precomputations for at least a small number of 1024-bit Diffie-Hellman groups. This would allow them to break any key exchanges made with those groups in close to real time. If true, this would answer one of the major cryptographic questions raised by the Edward Snowden leaks: How is NSA defeating the encryption for widely used VPN protocols?

Classified documents published by Der Spiegel [46] indicate that NSA is passively decrypting IPsec connections at significant scale. The documents do not describe the cryptanalytic techniques used, but they do provide an overview of the attack system architecture. After reviewing how IPsec key establishment works, we will use the published information to evaluate the hypothesis that the NSA is leveraging precomputation to calculate discrete logs at scale.

weakdh.org

CA BREACHES

Certificate authority breaches and errors

- DigiNotar in Jul. 2011
  - security breach, malicious certificates for many domains issued
  - went out of business
- TURKTRUST in Aug. 2011
  - issued intermediate CA with wildcard signing capabilities
  - later used for man-in-the-middle proxy filtering/scanning
  - no evidence for use in attack
  - detected only in Jan 2013
- Digicert Malaysia in Nov. 2011
  - 22 certificates with weak private keys or missing revocation details issued
- KPN/Getronics in Nov. 2011
  - suspended CA business after detecting infection on its web server no evidence of certificate malfeasance

- Web browsers trust 650+ certificate authorities which can issue certificates for any domain on the Internet
- Extended validation certificates don't solve the problem

LESSONS LEARNED

Lessons learned

- Be careful of protocol-level side channels
  - Bleichenbacher's attack
  - CRIME/BREACH compression
- Use standard cryptography correctly
  - IVs for CBC mode
  - MAC-then-encode-then-encrypt vs. encrypt-then-MAC
- Be careful of protocol logic
  - Renegotiation attack
- Sign everything
  - Downgrade attacks, Logjam