Metadata Protection Considerations for TLS Present and Future

Bryan Ford
Swiss Federal Institute of Technology (EPFL)
Lausanne, Switzerland

TRON Workshop – February 21, 2016

Outline

- Threat Models: Who is the Attacker?
- The Many Levels of Metadata Leakage
- Potential Countermeasures for TLS
  - Padding and Record Boundary Hiding
  - Encryption of Handshaking Metadata
  - Padding considerations for TLS implementations
- Conclusion

Threat Models

Philosophy: **Avoid Security Nihilism**

Must consider both “strong” and “weak” attackers

- Yes we should do what we can against the strong, all-seeing attacker
- But weaker, more limited attackers are probably far more numerous on the real Internet

Just because protection measure X doesn't stop all-powerful attacker doesn't mean X is useless!

Particular Threat Models of Interest

- **Passive Eavesdropper (EVE):** can monitor traffic but not inject packets. Ex: router taps
- **Man-On-The-Side (MOTS):** can monitor and inject but not block packets. Ex: WiFi snooper
- **Man-In-The-Middle (MITM):** can monitor, inject, and block legit packets. Ex: router
- **Man-On-The-Inside (MOTI):** can exert some control over *content* of encrypted traffic.
  Ex: via malicious JavaScript (CRIME attack)

What Does the Attacker Want?

Many possible objectives, e.g.,

- What website(s) or page(s) is this user visiting?
  - Bank? How many digits in balance?
- What user(s) are visiting this site?
  - Are these TLS flows from same or different users?
- What software, version(s) are endpoints using?
  - Pinpoint a version with a known bug we can exploit
- Tor de-anonymization via end-to-end correlation
  - Is flow X “going in” same as flow Y “going out”?

Outline

- Threat Models: Who is the Attacker?
- **The Many Levels of Metadata Leakage**
- Potential Countermeasures for TLS
  - Padding and Record Boundary Hiding
  - Encryption of Handshaking Metadata
  - Padding considerations for TLS implementations
- Conclusion

Sample of Relevant Background

Website fingerprinting: e.g.,

- Dyer et al, “Peek-a-Boo, I Still See You: Why Efficient Traffic Analysis Countermeasures Fail”, IEEE Security/Privacy 2012
- Cai et al, “Touching from a distance: Website fingerprinting attacks and defenses”, CCS '12
- Wang et al, “Effective attacks and provable defenses for website fingerprinting”, Sec '14

(and many others)

Many Levels of Metadata Leakage

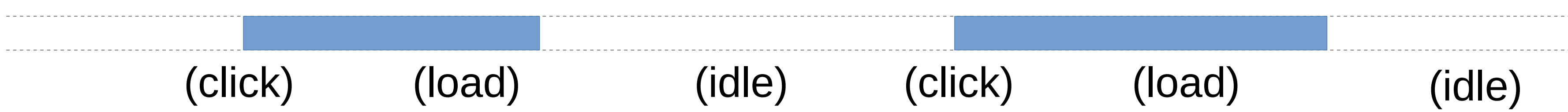
Leakage Level	**Who Can Mitigate?**
• Net activity bursts	• TLS impl, application
• Directional patterns	• TLS implementation
• TLS nego metadata	• **TLS spec**, impl
• TLS record metadata	• **TLS spec**, impl
• TCP metadata	• TLS impl, TCP stack
• Endpoints (IP, etc)	• WiFi, Proxy, VPN, Tor

Network Activity Bursts

Coarse-grained, macroscopic views of flows based on amount, timing of transmitted bytes

- Easy, efficient for eavesdropper to measure
- But results likely to be noisy, error-prone

Web browsing over TLS:



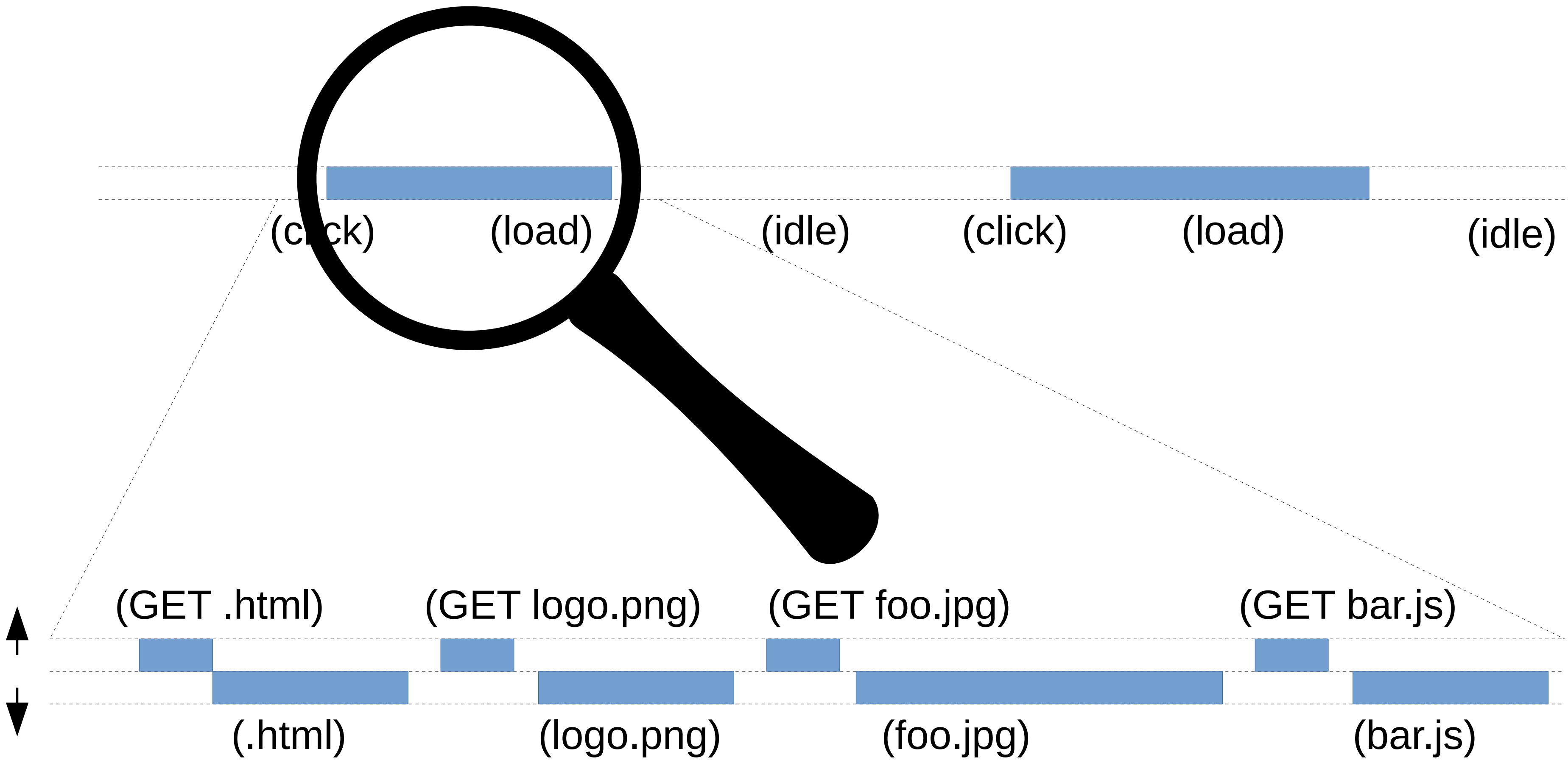
Audio stream over TLS:

Video stream TLS:

Directional flow & timing patterns

Attacker can use fine-grain upstream/downstream patterns *within* each burst of activity

- Much richer, more detailed, less error



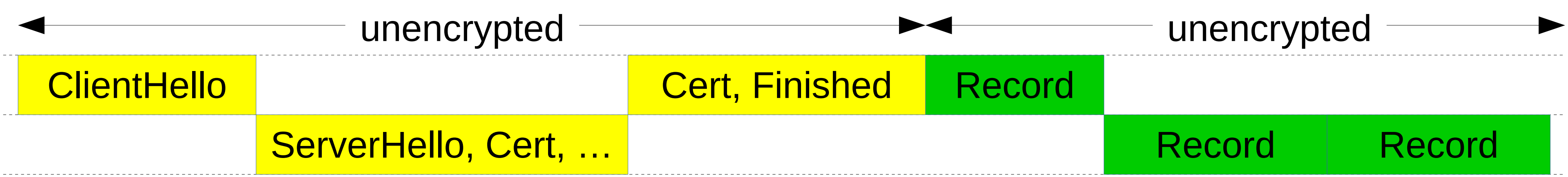
Exposed TLS Negotiation Metadata

Attacker can learn a lot just from the *unencrypted* negotiation metadata at beginning of TLS session

- Cyphersuites & groups supported, selected
- Server Name Indication (SNI)
- Reused “ephemeral” keys (link sessions)

Even “innocent” variation (e.g., ordering of fields) helps attacker fingerprint TLS impls, versions

- Useful for selective blocking, focusing attacks



Exposed TLS Record Metadata

Unencrypted 5-byte headers “give away” exact lengths, boundaries of each TLS record

Application write() boundaries often translate to readily-visible TLS record boundaries

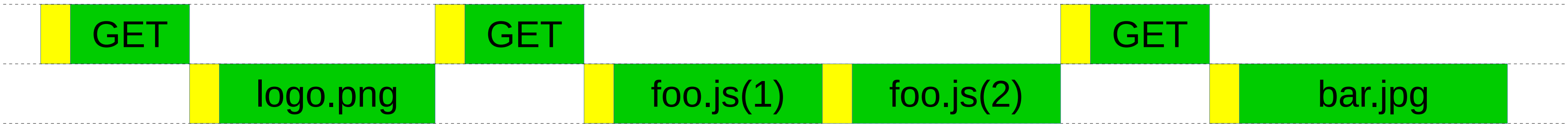
How important is this leak?

- Depends on how application protocol uses TLS

Example: HTTP/1.1 vs HTTP/2.0

HTTP/1.1 without pipelining or fixed-rate padding:

- Individual HTTP request size/pattern visible *either* via TLS records *or* via TCP-level bursts



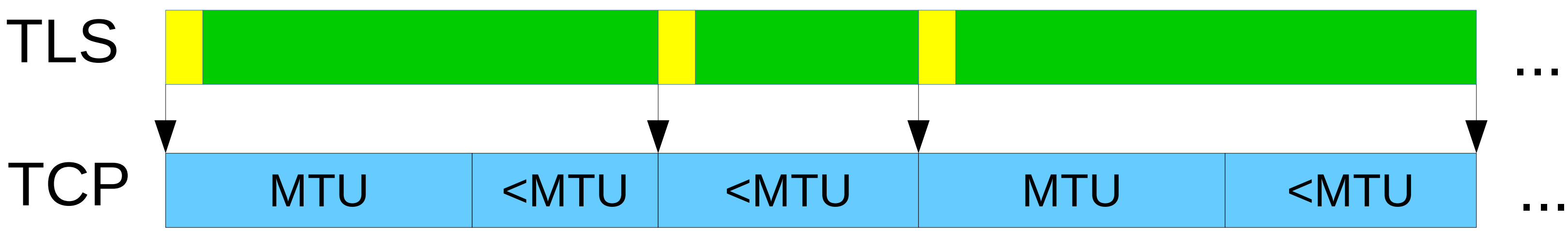
HTTP/2.0 with pipelining & multi-streaming:

- Concurrent bursts *could* obscure individual requests...
- Except that TLS record metadata still reveals them

TCP Segment Metadata

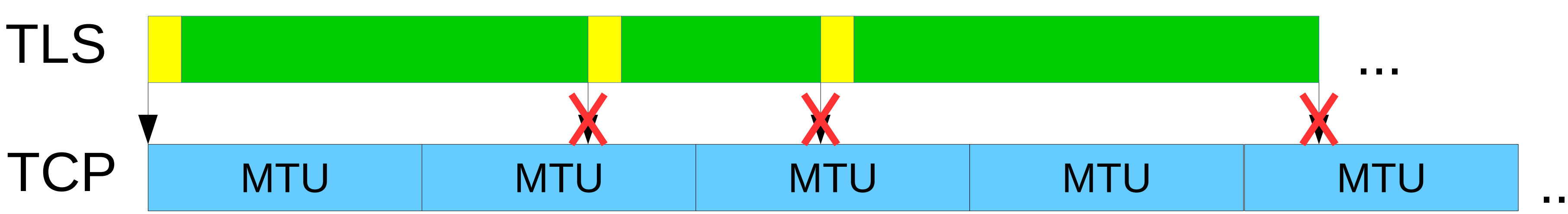
TCP segment boundaries *may* reveal TLS records

- If TLS write() translates to immediate TCP push



But also *may not*, as kernel forms MTU-len segs:

- Flow is congestion-limited, TX buffer nonempty
- If TCP_CORK or MSG_MORE options used



IP and Lower-Level Metadata

IP addresses, MAC addresses, HW fingerprints

Can be (partially) addressed via:

- WiFi encryption (if attacker isn't on same net)
- MAC address randomization
- HTTP proxies
- Corporate VPNs
- Tor

Not TLS's problem, or for TLS to solve.

Outline

- Threat Models: Who is the Attacker?
- The Many Levels of Metadata Leakage
- **Potential Countermeasures for TLS**
  - Padding and Record Boundary Hiding
  - Encryption of Handshaking Metadata
  - Padding considerations for TLS implementations
- Conclusion

Measures for TLS Implementations

Many countermeasures could be implemented without affecting basic TLS protocol spec

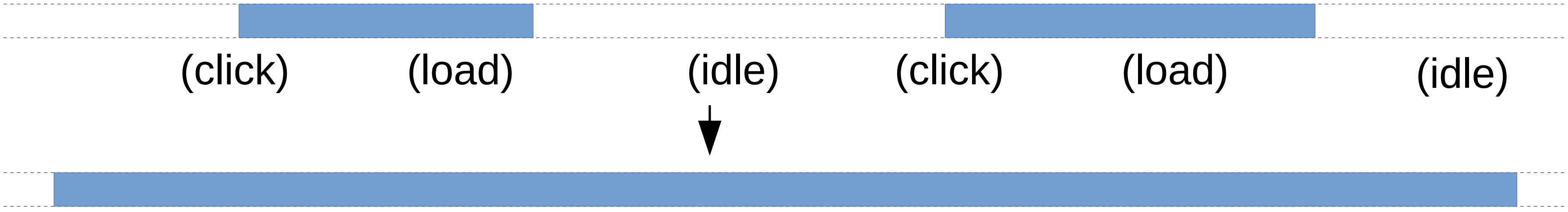
- Padding traffic to fixed-rate or maximum-rate
- Padding activity bursts until next idle period
- TCP segment MTU-size normalization

Recommendation: develop, standardize separate, follow-on “best practices” document for traffic analysis protection in TLS implementations

Example Padding Policies

1. Pad to fixed-rate or congestion-limited rate

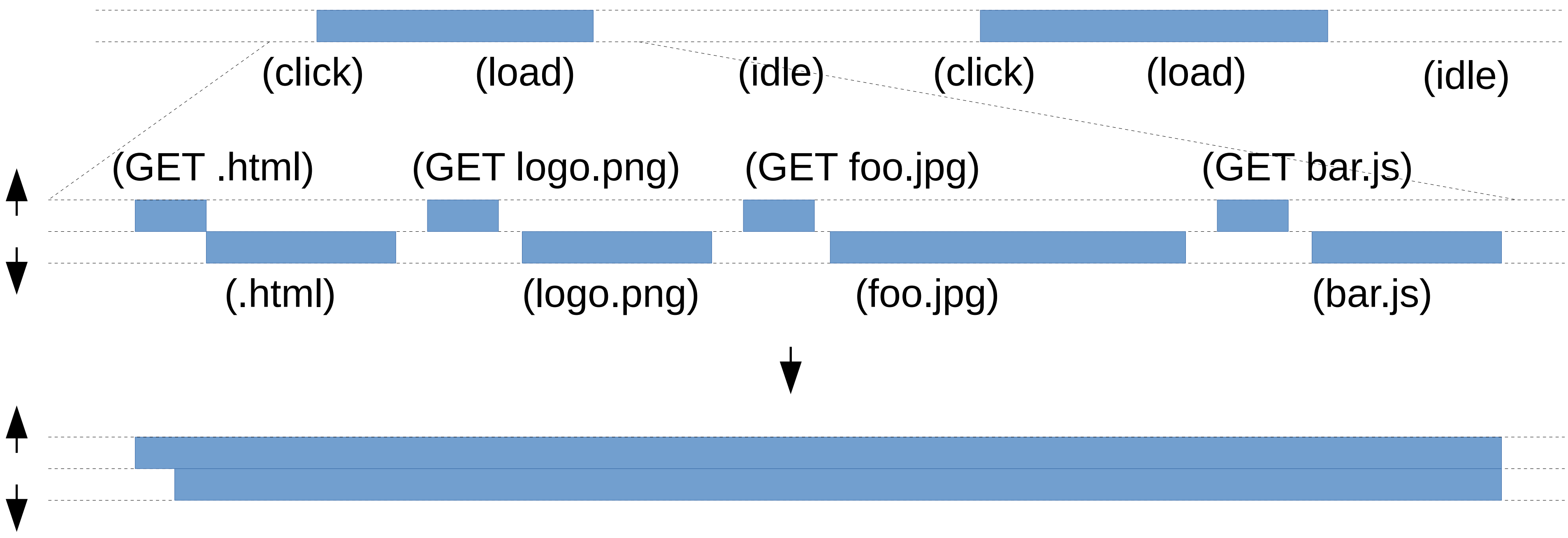
- Effective but probably too costly for most users
- May be practical client ↔ proxy or client ↔ VPN, but not client ↔ all-domains-a-page-depends-on



Example Padding Policies

2. Pad traffic only during “activity bursts”

- Costs probably more tolerable to many users
- But total size/length metrics can still leak info



Example Padding Policies

3. No special/costly padding measures

- Many users won't know or care enough to "pay" almost anything for padding
- Many TLS implementations won't implement

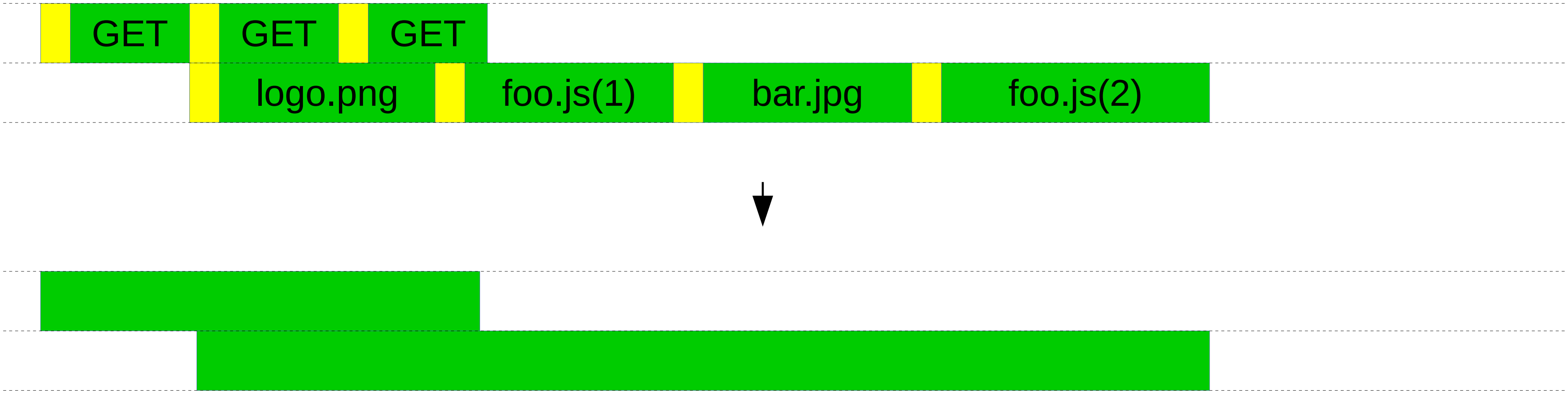
Can we still get *some* traffic analysis protection at low/no cost? (Repeat: avoid security nihilism!)

Example Padding Policies

Can we still get *some* traffic analysis protection at low/no cost? (Repeat: avoid security nihilism!)

Yes: HTTP/2.0 will help, if TLS doesn't undermine

- Traffic analysis gets a lot harder/noisier if hard to distinguish individual requests/replies



Measures for TLS Specification

Two relevant potential countermeasures

- Hide **record boundaries**
- Hide **handshake metadata**

Ideal: “encrypt everything”

- All parts of stream look uniformly random to any eavesdropper without relevant keys

Too ambitious for TLS 1.3, but baby steps...

Outline

- Threat Models: Who is the Attacker?
- The Many Levels of Metadata Leakage
- Potential Countermeasures for TLS
  - **Padding and Record Boundary Hiding**
  - Encryption of Handshaking Metadata
  - Padding considerations for TLS implementations
- Conclusion

Hiding TLS Record Boundaries?

Feasible for TLS to hide its record boundaries?

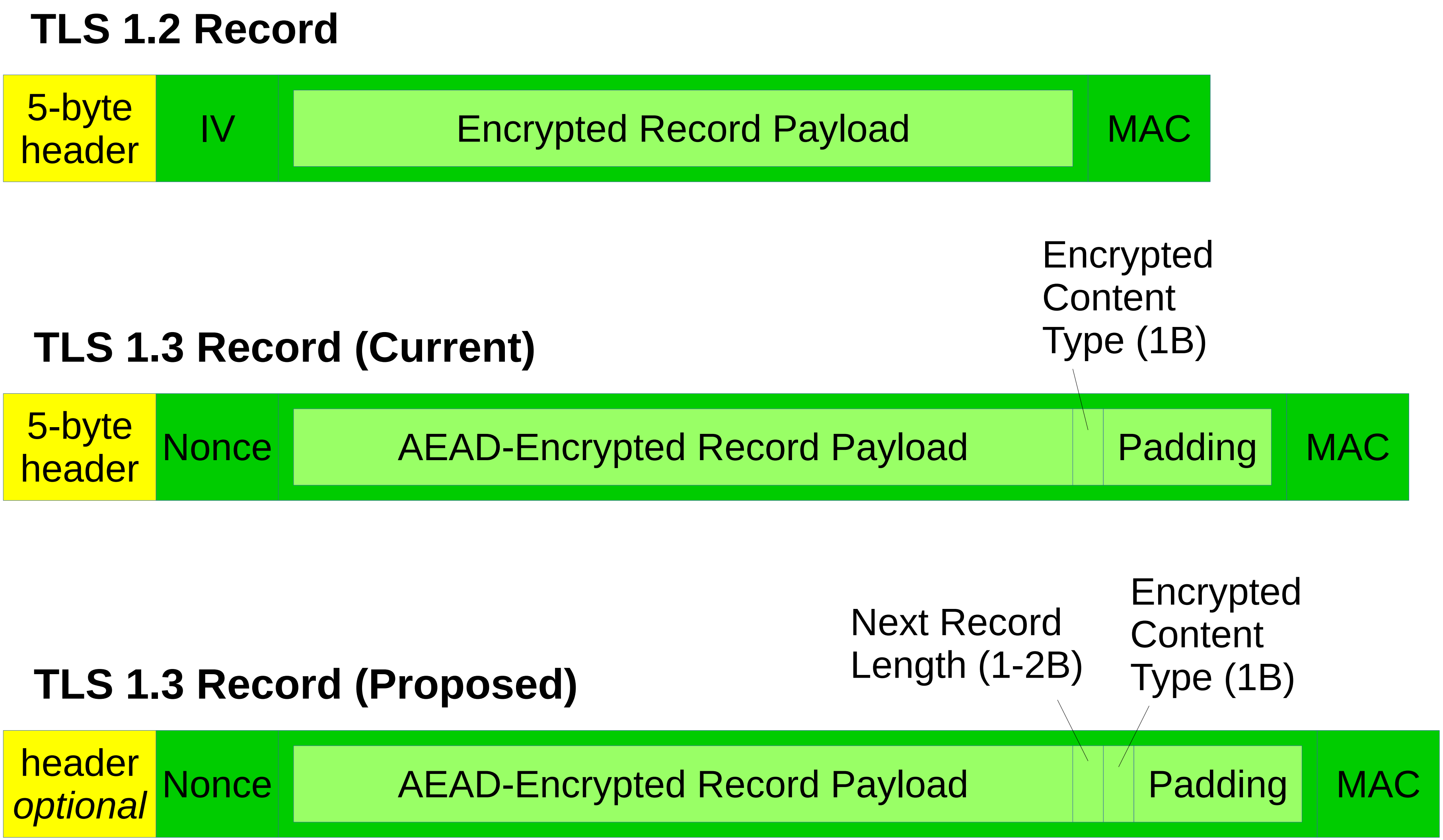
- Leave nothing unencrypted after handshake

Main challenge: how receiver finds record length?

- Normally the only “important” part of header
- Need to separately/specially encrypt length?

Simpler alternative approach described in
TLS mail list messages Dec 1 and Dec 12

TLS Record Format Evolution



Optional Headers in TLS 1.3

Proposed header rules:

- First record *always* has usual 5-byte header
- If Next Record Length field == 0,
  following record also has usual 5-byte header
- If Next Record Length field != 0,
  following record has indicated length, *no header*

Upshot: sender gets to omit next record’s header,
but must decide next record’s length in advance

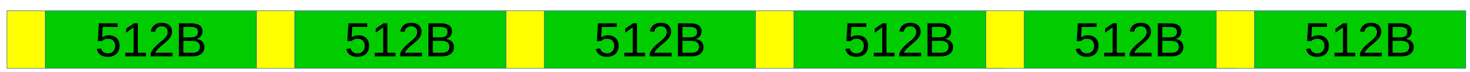
Design Advantages

- Minimal new receiver logic (1 state variable)
- Sender logic optional (can just set NextRec = 0)
- Sender logic trivial using fixed-length records
- Replace N L-byte records w/ N$\times$L-byte record
  - Reduce per-record compute, bandwidth costs
- Can disable if middleboxes really want headers
- Can save 3-4 bytes per record, FWIW

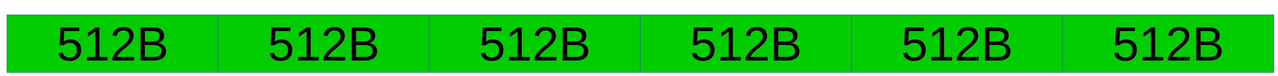
Transmission Example

Example: say we want to pad all records to 512 bytes

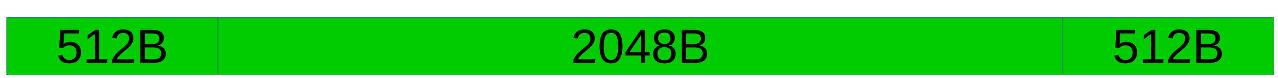
Current TLS 1.3 stream would look like this:



Proposed TLS 1.3 streams could instead look like this:



Or like this, *without* leaking anything to traffic analysis:



Prototype Implementation

Delta against NSS/NSPR available on GitHub

- https://github.com/bford/nss

Complexity metrics:

- TLS 1.2 → TLS 1.3 record format: 78-line delta
- TLS 1.3 → optional headers: 32-line delta

Further information: see Dec 12 mailing list post

- “[TLS] Prototype of TLS 1.3 records, padding, and optional headerless records”

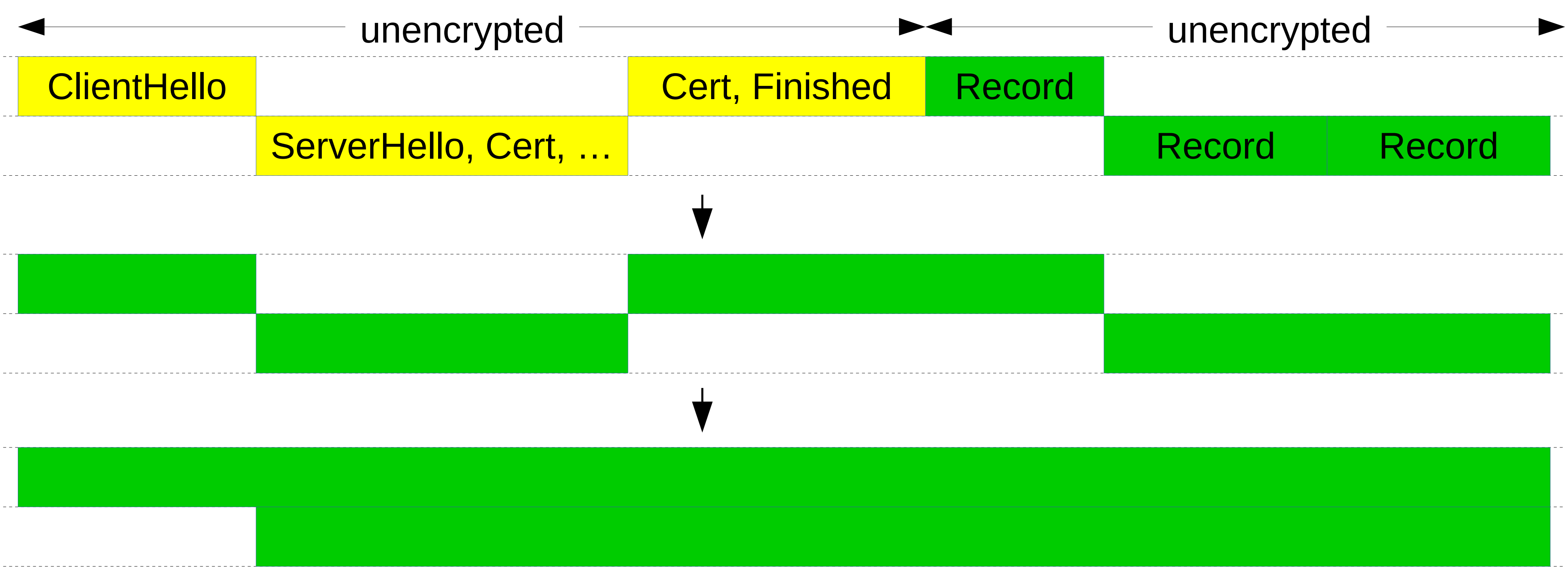
Outline

- Threat Models: Who is the Attacker?
- The Many Levels of Metadata Leakage
- Potential Countermeasures for TLS
  - Padding and Record Boundary Hiding
  - **Encryption of Handshaking Metadata**
  - Padding considerations for TLS implementations
- Conclusion

Towards Encrypted Handshaking

Could TLS **encrypt everything** from byte 0?

- Probably too ambitious for TLS 1.3,
  but worth considering for TLS 1.4 or 2.0?



Encrypted Handshaking: Feasible?

Key challenges:

- Client needs to have *some* cryptographic info (public keys) about server to start with
- Bootstrapping key agreement: e.g., making ephemeral DH keys uniformly random
- Negotiating multiple cyphersuites, groups, keys under encryption

Finding Server Public Keys

Client needs to have *some* cryptographic info (public keys) about server to start with.

At least two promising sources of this info:

- Cached information from previous sessions: same info clients need anyway for 0-RTT
  - Provide “enhanced TOFU” property:
    attacker who didn’t see first session doesn’t learn anything from subsequent handshakes
- Learn key(s) via DNSSEC/DANE lookups

Encrypted Key Agreement

Bootstrapping key agreement: e.g.,
making ephemeral DH keys uniformly random

- For RSA-based or DH-based key agreement, theoretically “straightforward”
- For ECDH-based key agreement, that’s what Elligator techniques are for

Ephemeral ECDH point	Symmetric-key encrypted data

Works as long as client “just knows” (or guesses) correct ciphersuite, group, etc to use.

Multi-Suite/Group/Key Handshaking

What if client “not sure” what crypto info to use?

- Has several possible server public keys, some may be obsolete, may have preferences

Simple solution: try each w/ separate TCP conn

Fancier solution: can build Elligator-style header decryptable via multiple suites, groups, keys

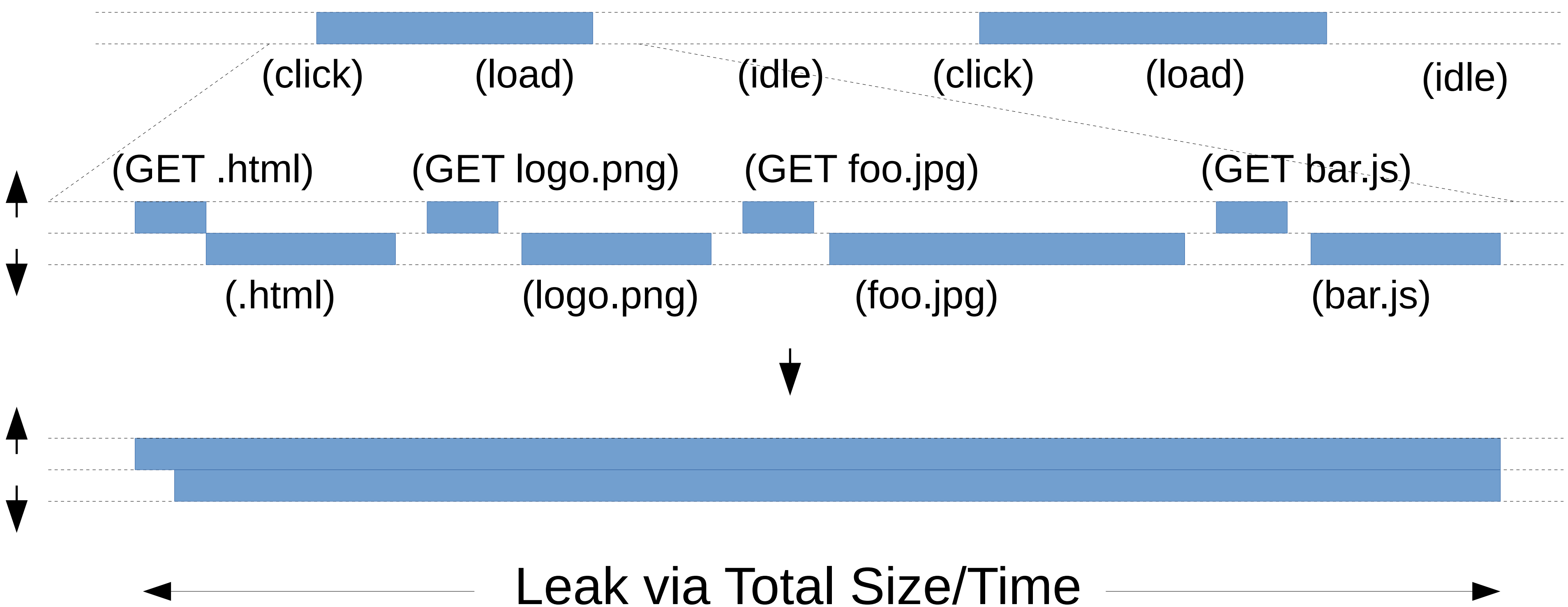
- Motivated by offline PGP-style encryption, but could be used in TLS handshaking too
- Further info: long, dense openpgp list E-mail

Outline

- Threat Models: Who is the Attacker?
- The Many Levels of Metadata Leakage
- Potential Countermeasures for TLS
  - Padding and Record Boundary Hiding
  - Encryption of Handshaking Metadata
  - **Padding considerations for TLS implementations**
- Conclusion

How to Pad Activity Bursts?

Balance cost in wasted bandwidth versus amount of information leaked by padded length



Burst Padding Policies, Revisited

Goal: minimize information leakage via length

- Can we formally bound Shannon entropy?

Simple approach: pad burst to next power-of-two

- Reduces leakage from O(log n) to O(log log n)

But:

- Incurs up to 2×, avg 1.5× bandwidth overhead
- Bad leak if attacker can force close-to-boundary

Reducing Bandwidth Waste

Allow lengths representable as floating-point with mantissa bit-length ≤ exponent bit-length

k-bit exponent	**k-bit mantissa**

- Still limits max leakage to O(log log N)
- But wastes max 11%, smaller for big bursts

Padded sizes vs padding waste

Length	Length bits	Leak bits	Length inc	Max waste
1	1	0	1	0.00%
2	2	1	1	0.00%
4	3	2	1	0.00%
8	4	2	2	11.11%
16	5	3	2	5.88%
32	6	3	4	9.09%
64	7	3	8	10.77%
128	8	3	16	11.63%
256	9	4	16	5.84%
512	10	4	32	6.04%
1024	11	4	64	6.15%
2048	12	4	128	6.20%
4096	13	4	256	6.22%
8192	14	4	512	6.24%
16384	15	4	1024	6.24%
32768	16	4	2048	6.25%
65536	17	5	2048	3.12%
131072	18	5	4096	3.12%
262144	19	5	8192	3.12%
524288	20	5	16384	3.12%
1048576	21	5	32768	3.12%
2097152	22	5	65536	3.12%
4194304	23	5	131072	3.12%
8388608	24	5	262144	3.12%

Example: 1-byte Next Record Len

4-bit exponent, 4-bit mantissa

- Compute actual length = mantissa << (exp - 4)
- Rep lengths up to 1.1111b×2^15 (> TLS max)

Randomized Internal Padding

Randomized padding: worthwhile?

- Weak by itself due to statistical leakage, but...

Add small random amount of padding *before* padding to next standardized burst length

- Reduces per-burst information leakage even if attacker can control internal layout, arrange for important info to be “on boundary”
- Stronger against “Man-On-The-Inside” attacks (e.g., malicious JavaScript, as used in CRIME)

Outline

- Threat Models: Who is the Attacker?
- The Many Levels of Metadata Leakage
- Potential Countermeasures for TLS
  - Padding and Record Boundary Hiding
  - Encryption of Handshaking Metadata
  - Padding considerations for TLS implementations
- **Conclusion**

Conclusion

Traffic analysis protection is a hard problem, but let’s avoid security nihilism and take baby steps

TLS record hiding: simple measure that can help

- With HTTP/2.0, obscure individual transactions
- Makes padding more efficient for multi-records

Longer-term goals to consider:

- Best-practices doc for traffic analysis protection
- Eventually: encrypt everything from byte 0?