Authenticated Encryption in TLS

Kenny Paterson

DIAC 2013

Royal Holloway
University of London

Information Security Group

Outline

- TLS and the TLS Record Protocol
- Theory for TLS
- Attacks:
  - The BEAST
  - Padding oracles
  - Lucky 13
  - (More theory for TLS)
  - RC4 attack
  - CRIME/BREACH
- Discussion

TLS

- SSL = Secure Sockets Layer.
  - Developed by Netscape in mid 1990s.
  - SSLv3 still widely supported.

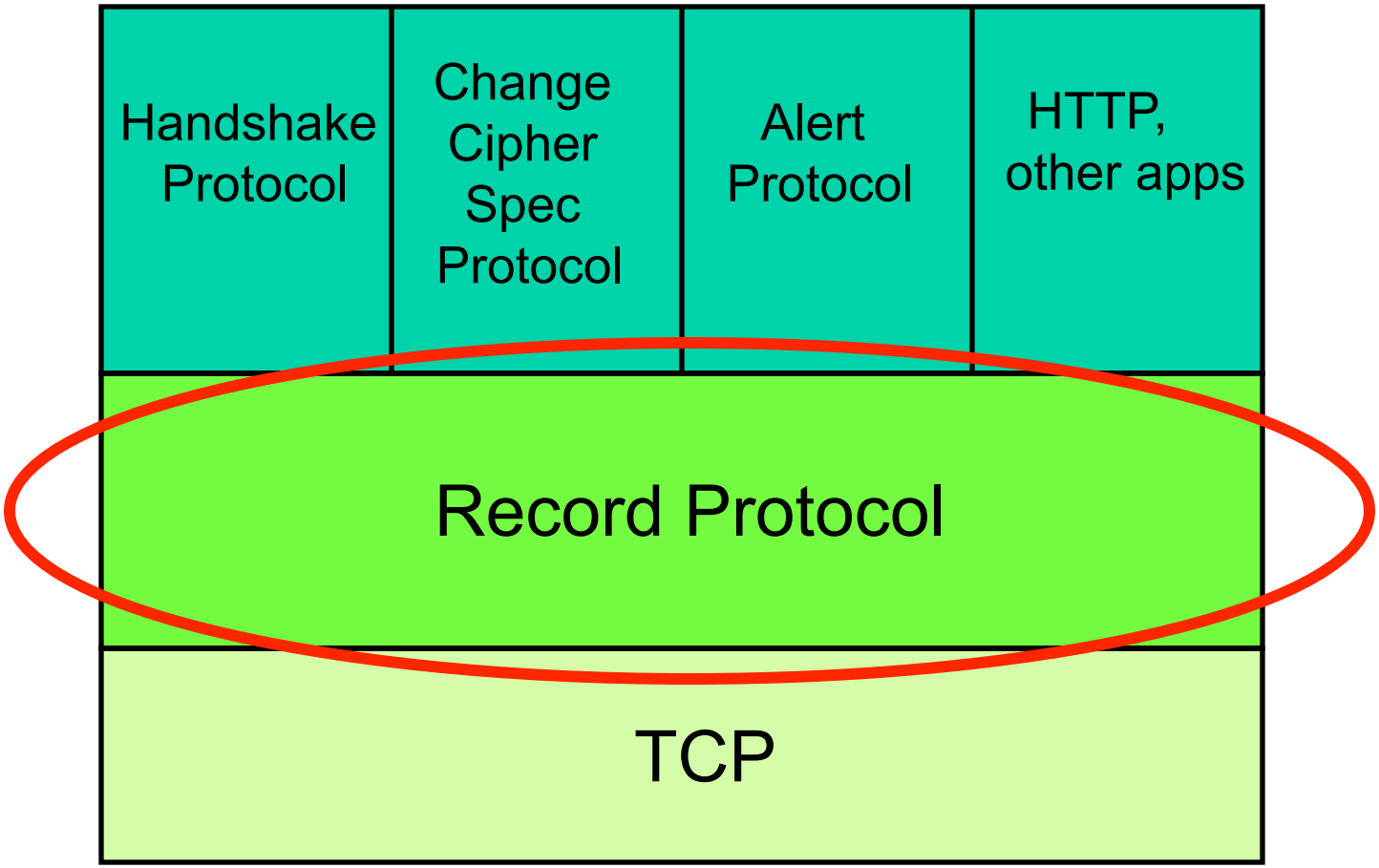
- TLS = Transport Layer Security.
  - IETF-standardised version of SSL.
  - TLS 1.0 in RFC 2246 (1999).
  - TLS 1.1 in RFC 4346 (2006).
  - TLS 1.2 in RFC 5246 (2008).

TLS – Why You Should Care

- Originally 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 the *de facto* secure protocol of choice.
  - Used by hundreds of millions of people and devices every day.
  - A serious attack could be catastrophic, both in real terms and in terms of perception/confidence.

TLS Protocol Architecture

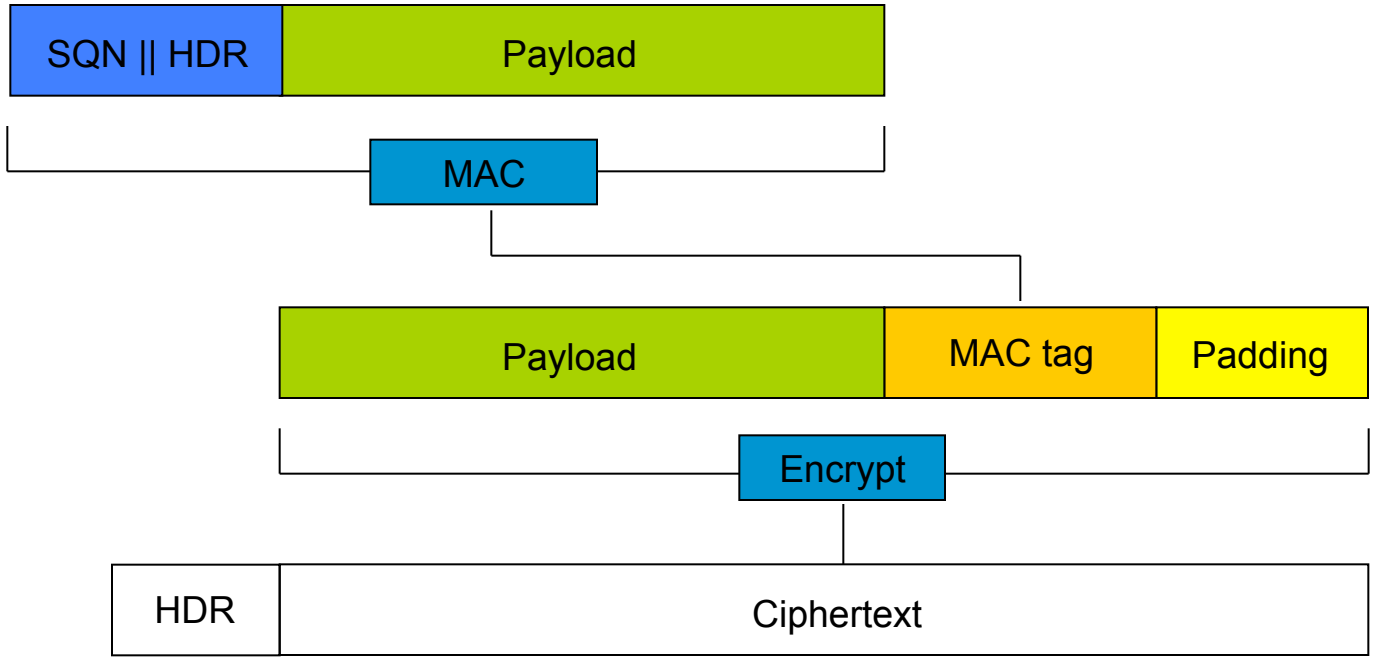


TLS Record Protocol

- TLS Record Protocol provides:
  - Data origin authentication, integrity using a MAC.
  - Confidentiality using a symmetric encryption algorithm.
  - Anti-replay using sequence numbers protected by the MAC.
  - Optional compression.
  - Fragmentation and reassembly of application layer messages.

- Keys for the algorithms are supplied by the TLS Handshake Protocol.

TLS Record Protocol: MAC-Encode-Encrypt



MAC	HMAC-MD5, HMAC-SHA1, HMAC-SHA256
Encrypt	CBC-AES128, CBC-AES256, CBC-3DES, RC4-128
Padding	“00” or “01 01” or “02 02 02” or …. or “FF FF….FF”

AE and TLS

- Dedicated Authenticated Encryption (AE) schemes are supported in TLS 1.2.
  - AES-GCM specified in RFC 5288.
  - AES-CCM specified in RFC 6655.

- But TLS 1.2 is not yet widely supported.
  - Most browsers support SSLv3 and TLS 1.0 only.
  - Currently, roughly 50/50 usage split between CBC-mode and RC4.
  - Less than 15% of servers support TLS 1.1 or higher (source: SSL Pulse).
  - [29% of servers still support SSLv2!]

- In the rest of the talk, we focus on CBC-mode and RC4.

TLS Record Protocol Features

- Sequence number is 64 bits in size and is incremented for each new message.
- Sequence number not transmitted as part of message.
  - Each end of connection maintains its own view of the current value of the sequence number.
  - TLS is reliant on TCP to deliver messages in order.
- Wrong sequence number leads failure of MAC verification
  - A fatal error leading to session teardown.
- Creates stateful encryption scheme.
  - Preventing replay, insertion, reordering attacks,…

Outline

- TLS and the TLS Record Protocol
- Theory for TLS
- Attacks:
  - The BEAST
  - Padding oracles
  - Lucky 13
  - (More theory for TLS)
  - RC4 attack
  - CRIME/BREACH
- Discussion

Theory for TLS

- TLS employs a (stateful) MAC-then-encrypt composition.

- This is known to be **not** generically secure, according to results of [BN00].
  - We can construct an IND-CPA secure encryption scheme and a SUF-CMA secure MAC scheme for which the MAC-then-encrypt composition fails to be IND-CCA secure.

Theory for TLS

- Building on results of Krawczyk [K01], the basic MAC-then-encrypt construction can be shown to be IND-CCA secure *for the special case of CBC mode encryption*.
  - Even better, for CBC mode encryption, the composition is both IND-CPA and INT-CTXT secure, i.e. AE-secure.
  - INT-CTXT: hard to forge a new, valid ciphertext, having seen many ciphertexts for chosen messages.
- This extends to the stateful setting, as formalised in [BKN03].
- AE security also holds for RC4 under the assumption that its output is pseudorandom.

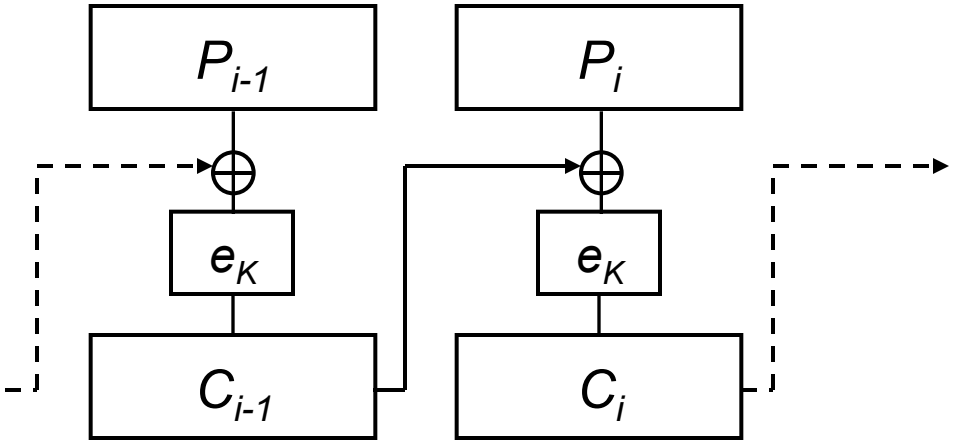
Theory for TLS – Caveats

- Krawczyk's analysis assumes random IVs for CBC mode.
  - SSL 3.0 and TLS 1.0 use *chained* IVs.
- TLS is really using MAC-Encode-Encrypt.
  - With a specific padding scheme for the Encode step.
  - But padding is not treated in the analysis of [K01].
  - Data is assumed to be block-aligned, and MAC size = block size.
- RC4 has known statistical weaknesses.
- Do these gaps between theory and reality matter?

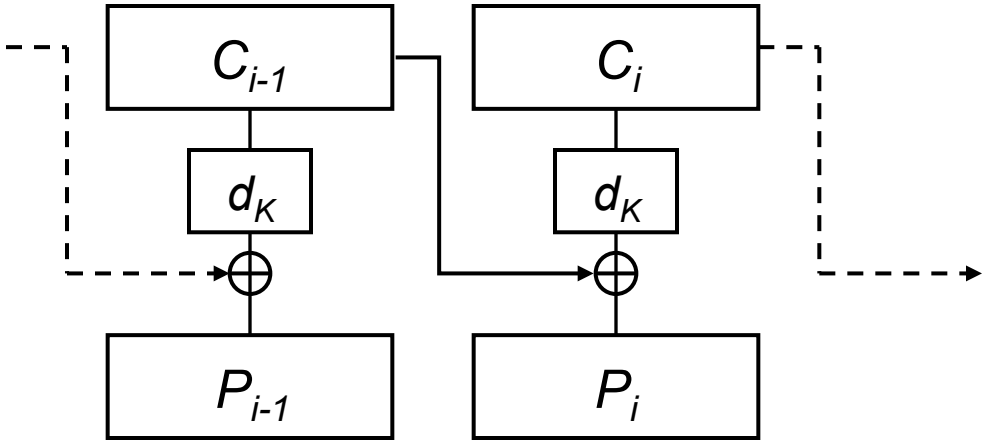
Outline

- TLS and the TLS Record Protocol
- Theory for TLS
- Attacks:
  - The BEAST
  - Padding oracles
  - Lucky 13
  - (More theory for TLS)
  - RC4 attack
  - CRIME/BREACH
- Discussion

CBC Mode in TLS



- SSLv3 and TLS 1.0 use a chained IV in CBC mode.
  - IV for current message is the last ciphertext block from the previous message.

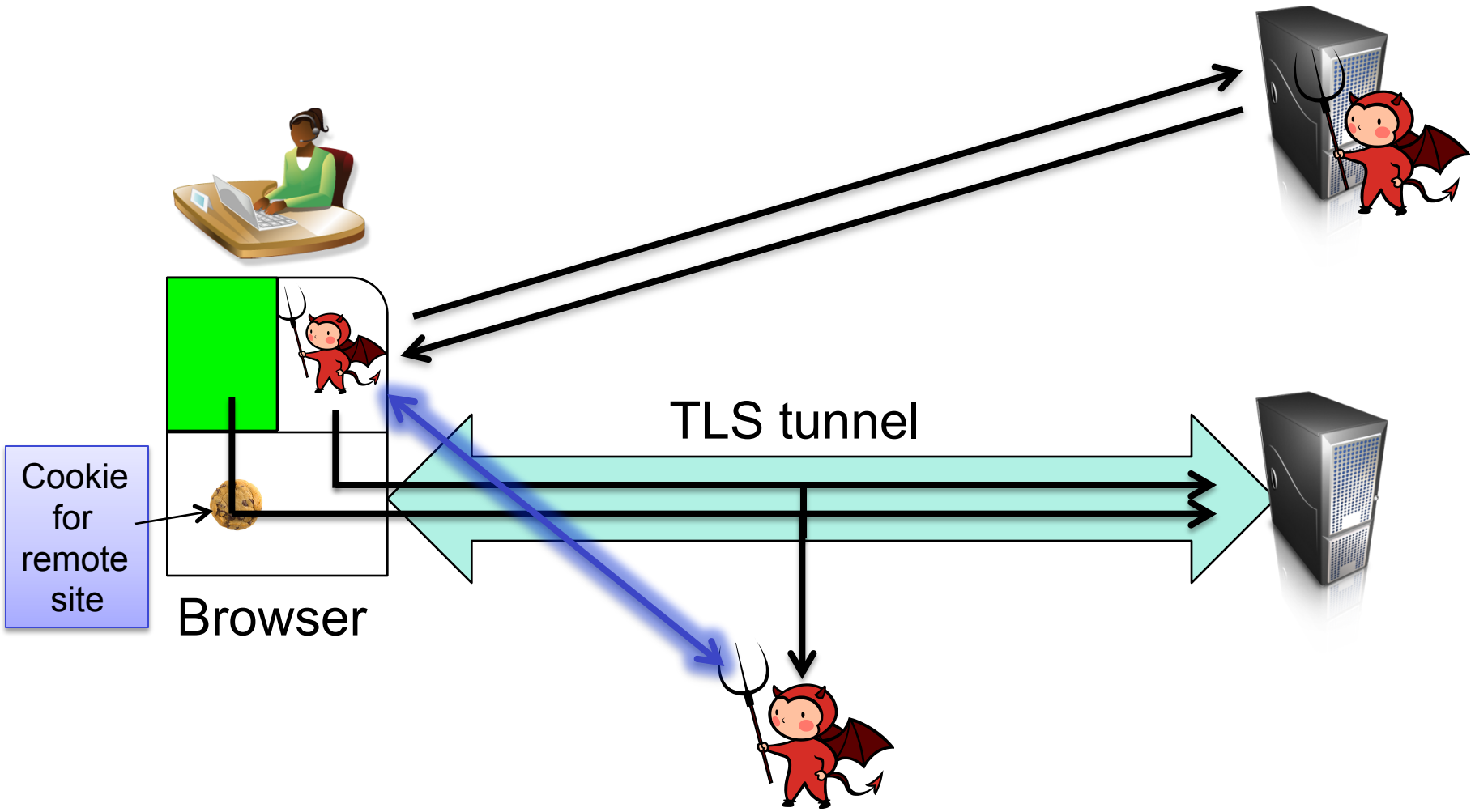


- Modified in TLS 1.1, 1.2.
  - TLS 1.2 now has explicit IV and recommends IV SHOULD be chosen at random for each message.

Attacking Predictable IVs

- IV chaining in SSLv3 and TLS 1.0 leads to a *chosen-plaintext distinguishing attack* against TLS.
  - First observed for CBC mode by Rogaway in 1995.
  - Then applied to TLS by Dai and Moeller in 2004.
  - Extended to theoretical plaintext recovery attack by Bard in 2004/2006.
  - Turned into a **practical** plaintext recovery attack by Duong and Rizzo in 2011.
  - The BEAST!

The BEAST and client-side malware



The BEAST

- Key points:
  - BEAST malware injected ahead of time into client browser.
  - Achieves **chosen-plaintext capability** for TLS encryption.
  - Uses HTTP padding to control positions of unknown bytes.
  - Communicates with MITM attacker.
  - It works.

- Malware can also initiate its own TLS sessions to remote host.
  - Browser will then automatically inject HTTP cookies into TLS session on behalf of malware.
  - Enables multi-session attacks targeting HTTP cookies.
  - More later!

The BEAST – Countermeasures

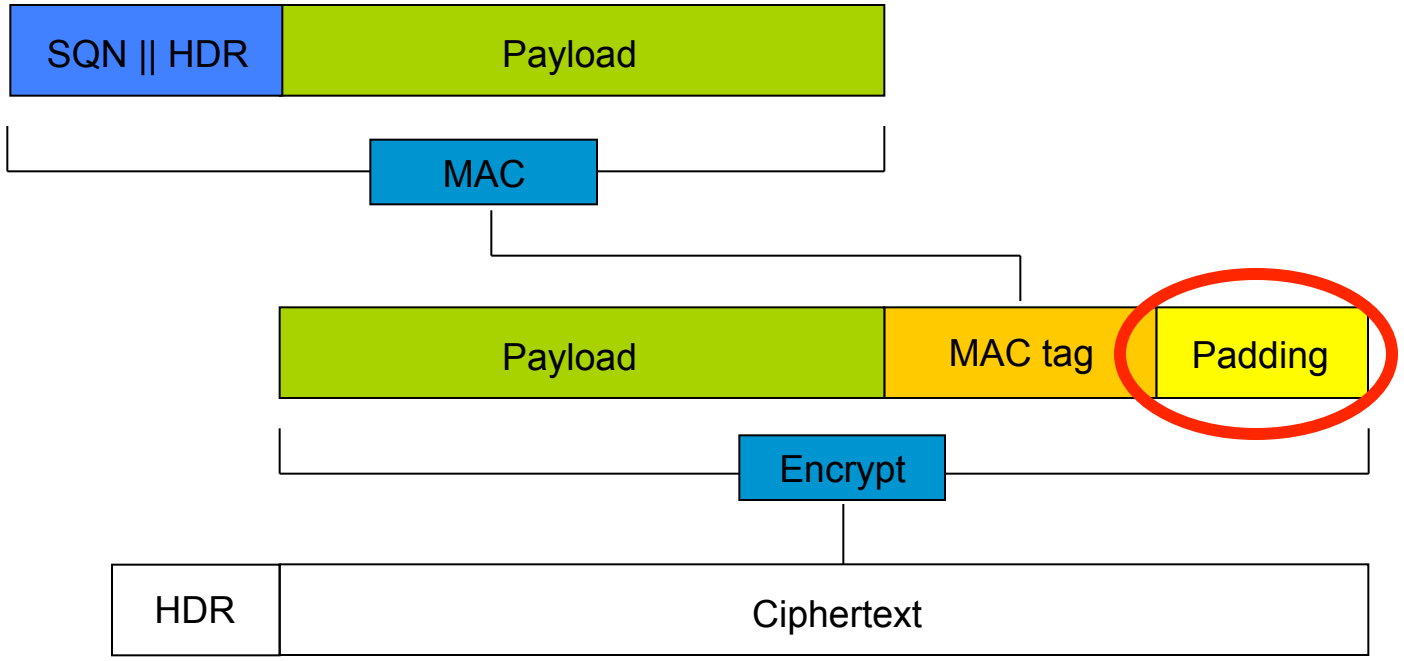
- Switch to using TLS 1.1 or 1.2.
  - Uses random IVs, so attack prevented.

- For TLS 1.0 users:
  - Use 1/n-1 record splitting.
    - Now implemented in most but not all browsers.
    - Safari (Apple): status unknown.
  - Send 0-length dummy record ahead of each real record.
    - Breaks some implementations.
  - Switch to using RC4.
    - As recommended by many expert commentators.

Outline

- TLS and the TLS Record Protocol
- Theory for TLS
- Attacks:
  - The BEAST
  - Padding oracles
  - Lucky 13
  - (More theory for TLS)
  - RC4 attack
  - CRIME/BREACH
- Discussion

TLS Record Protocol: MAC-Encode-Encrypt



MAC	HMAC-MD5, HMAC-SHA1, HMAC-SHA256
Encrypt	CBC-AES128, CBC-AES256, CBC-3DES, RC4-128
Padding	“00” or “01 01” or “02 02 02” or …. or “FF FF….FF”

TLS and Padding Oracles

[V02,CHVV03]:

- Specifics of TLS padding format can be exploited to mount a plaintext recovery attack.
- No chosen-plaintext requirement.
- The attack depends on being able to distinguish *good* from *bad* padding.
  - In practice, this is done via a timing side-channel.
  - The MAC is only checked if padding good, and the MAC is always bad in the attack.
  - Distinguish cases by timing TLS error messages.

TLS and Padding Oracles

[V02,CHVV03]:

- The attack is *multi-session*.
  - Each trial in the attack causes a fatal error and TLS session termination.
  - The attack still works if a *fixed* plaintext is repeated in a *fixed* location across many TLS sessions.
  - e.g. a password in an automated login.
  - Modern viewpoint: use BEAST-style malware and target HTTP cookies.
- Attack worked for OpenSSL.
  - Roughly 2ms difference for long messages.
  - Enabling recovery of TLS-protected Outlook passwords in about 3 hours.

Countermeasures?

- Redesign TLS:
  - Pad-MAC-Encrypt or Pad-Encrypt-MAC.
  - Too invasive, did not happen.
- Switch to RC4.
- Or add a fix to ensure *uniform errors*?
  - If attacker can't tell difference between MAC and pad errors, then maybe TLS's MEE construction is secure?
  - So how should TLS implementations ensure uniform errors?

Ensuring Uniform Errors

From the TLS 1.2 specification:

…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.

Compute the MAC on what though?

Ensuring Uniform Errors

For instance, if the pad appears to be incorrect, the implementation might assume a zero-length pad and then compute the MAC.

- This approach is adopted in many implementations, including OpenSSL, NSS (Chrome, Firefox), BouncyCastle, OpenJDK, …

- One alternative (GnuTLS and others) is to remove as many bytes as are indicated by the last byte of plaintext and compute the MAC on what's left.

Ensuring Uniform Errors

… 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.

Ensuring Uniform Errors

… 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.

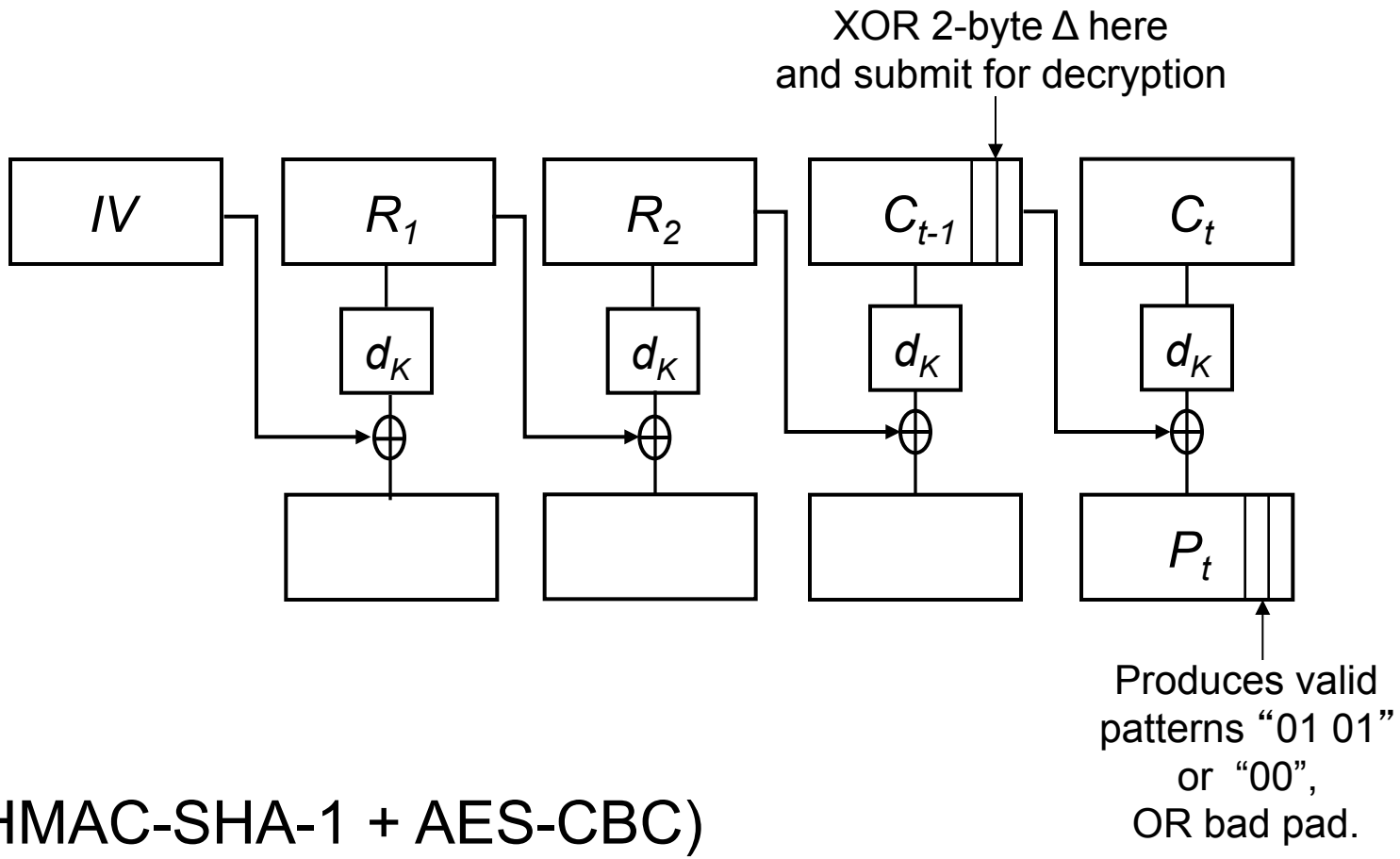
Outline

- TLS and the TLS Record Protocol
- Theory for TLS
- Attacks:
  - The BEAST
  - Padding oracles
  - Lucky 13
  - (More theory for TLS)
  - RC4 attack
  - CRIME/BREACH
- Discussion

Lucky 13 [AP13]

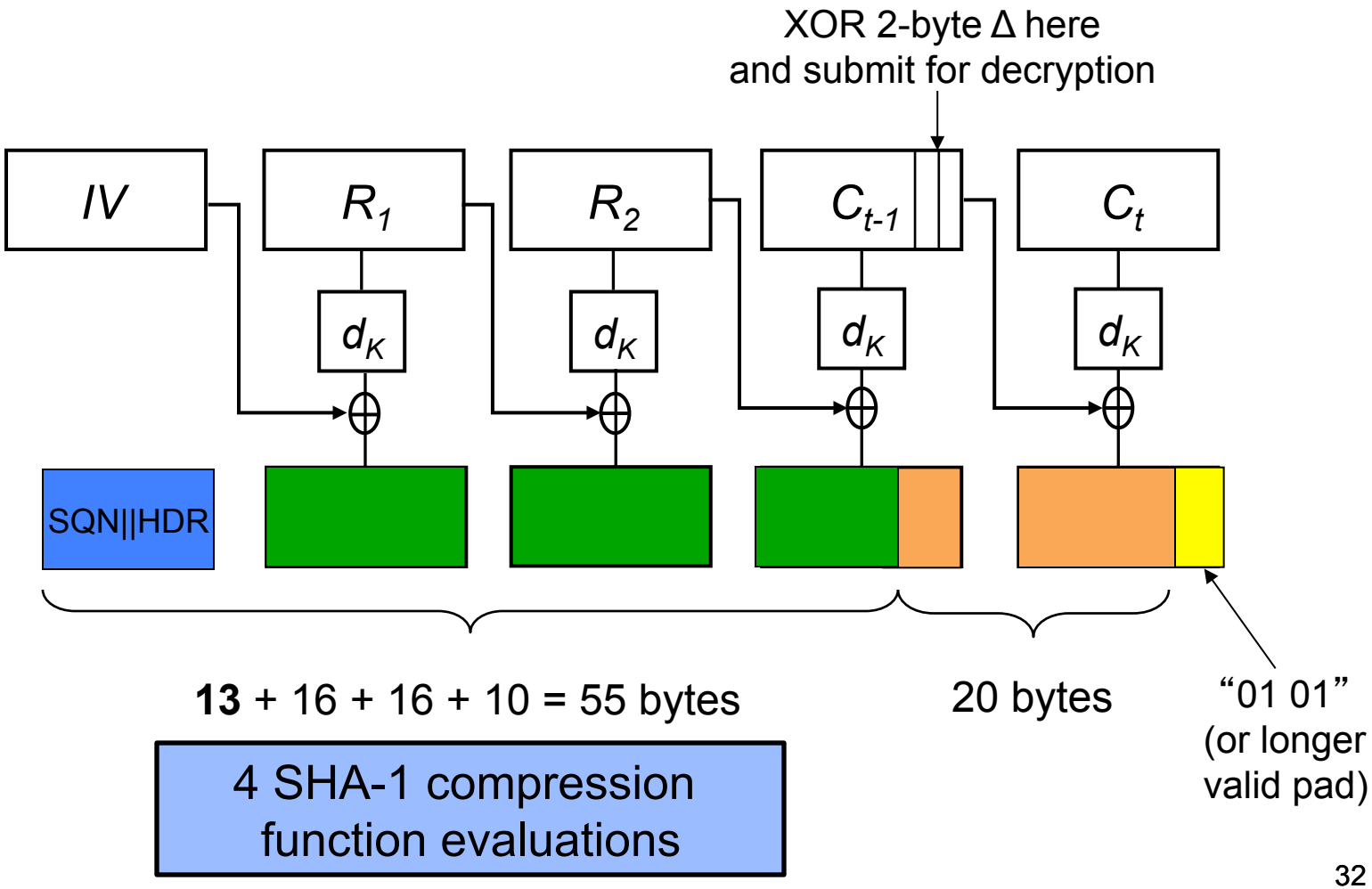
- Distinguishing attacks and full plaintext recovery attacks against TLS-CBC implementations following the advice in the TLS 1.2 spec.
  - And variant attacks against those that do not.
- Applies to all versions of SSL/TLS.
  - SSLv3.0, TLS 1.0, 1.1, 1.2.
  - And DTLS.
- Demonstrated in the lab against OpenSSL and GnuTLS.

Lucky 13 – Plaintext Recovery

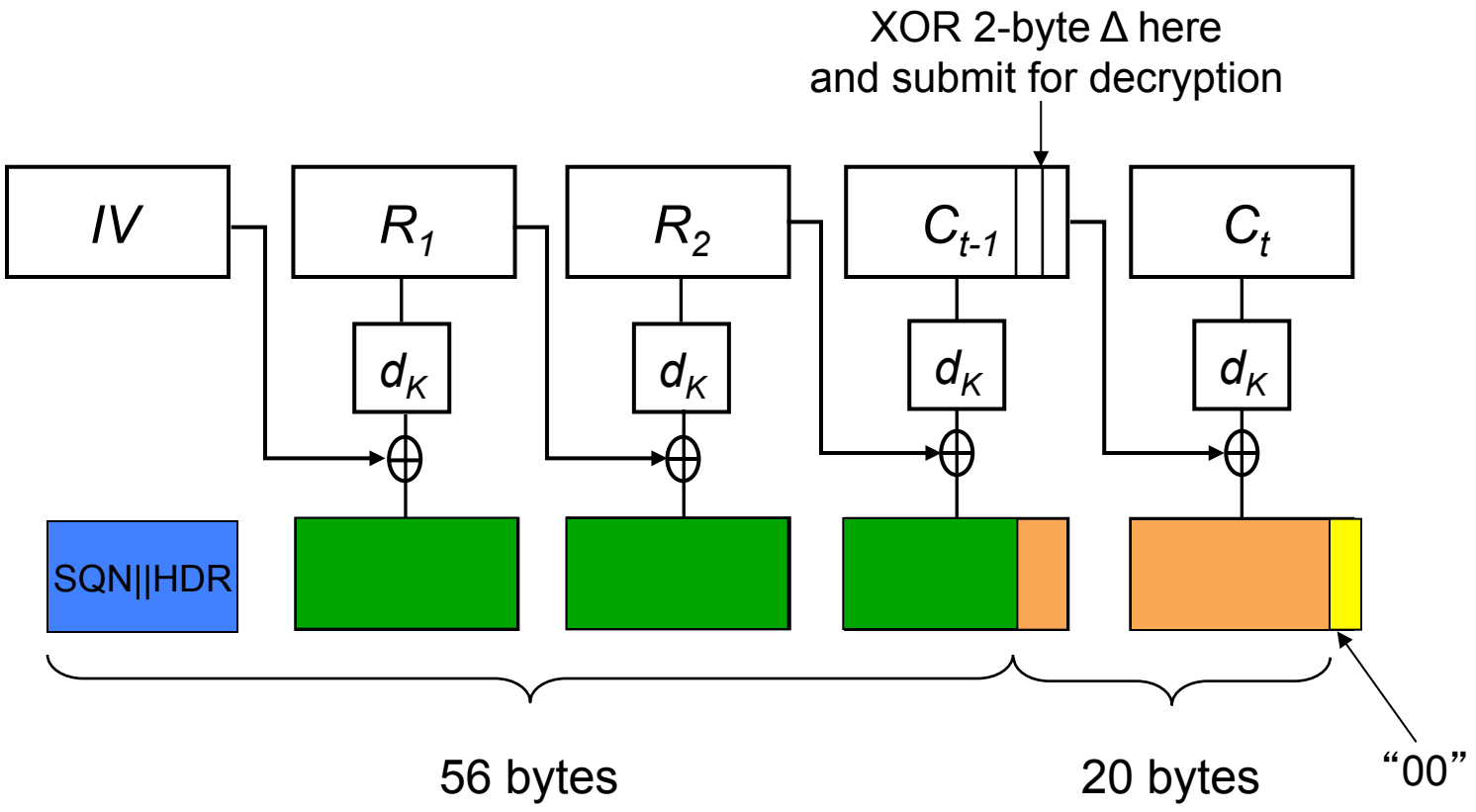


(HMAC-SHA-1 + AES-CBC)

Case: “01 01” (or longer valid pad)

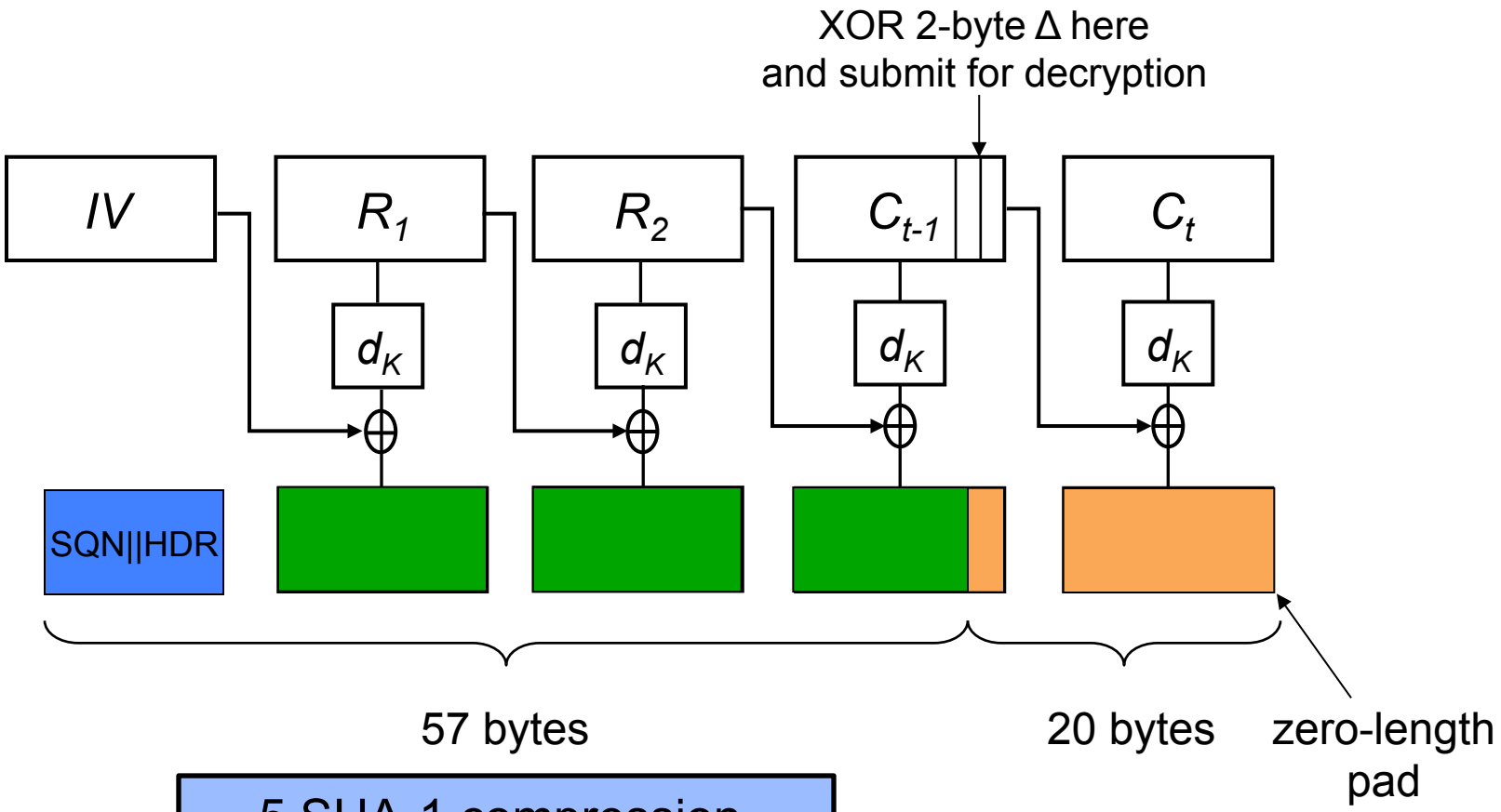


Case: “00”



5 SHA-1 compression function evaluations

Case: Bad padding



5 SHA-1 compression function evaluations

Lucky 13 – Plaintext Recovery

- The injected ciphertext causes bad padding and/or a bad MAC.
  - This leads to a TLS error message, which the attacker times.
- There is a timing *difference* between “01 01” case and the other 2 cases.
  - A single SHA-1 compression function evaluation.
  - Roughly 1000 clock cycles, 1μs range on typical processor.
  - Measurable difference on same host, LAN, or a few hops away.
- Detecting the “01 01” case allows last 2 plaintext bytes in the *target* block C_t to be recovered.
  - Using the standard CBC algebra.
  - Attack then extends easily to all bytes.

Lucky 13 – Attack Cost

- We need 2^{16} attempts to try all 2-byte Δ values.
- And we need around 2^{7} trials for each Δ value to reliably distinguish the different events.
  - Noise level depends on experimental set-up.
- Each trial kills the TLS session.
- Hence the headline attack cost is 2^{23} sessions, all encrypting the same plaintext.
- So what was all the fuss about?

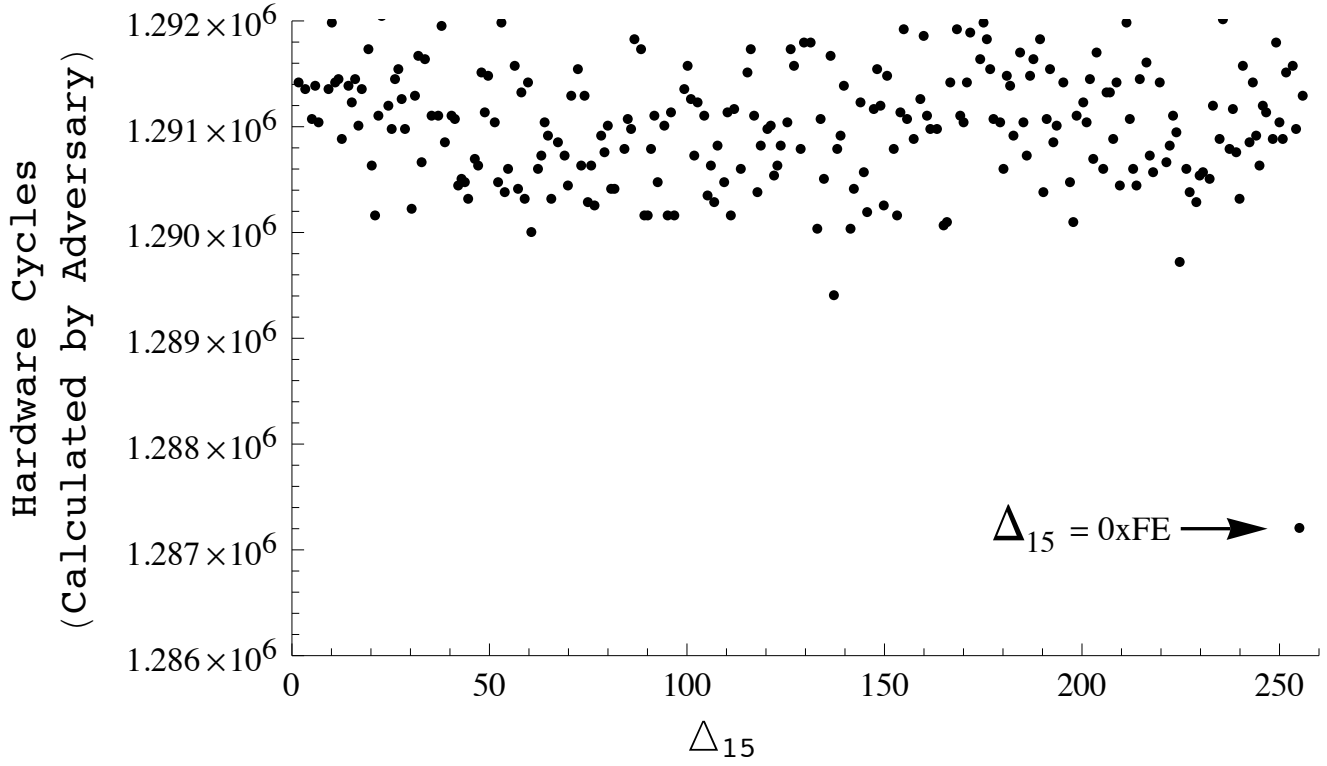
Lucky 13 – Improvements

- If 1-out-of-2 last bytes known, then we only need 2^8 attempts per byte.

- If the plaintext is `base64` encoded, then we only need 2^6 attempts per byte.
  - And 2^7 trials per attempt to de-noise, for a total of 2^{13}.

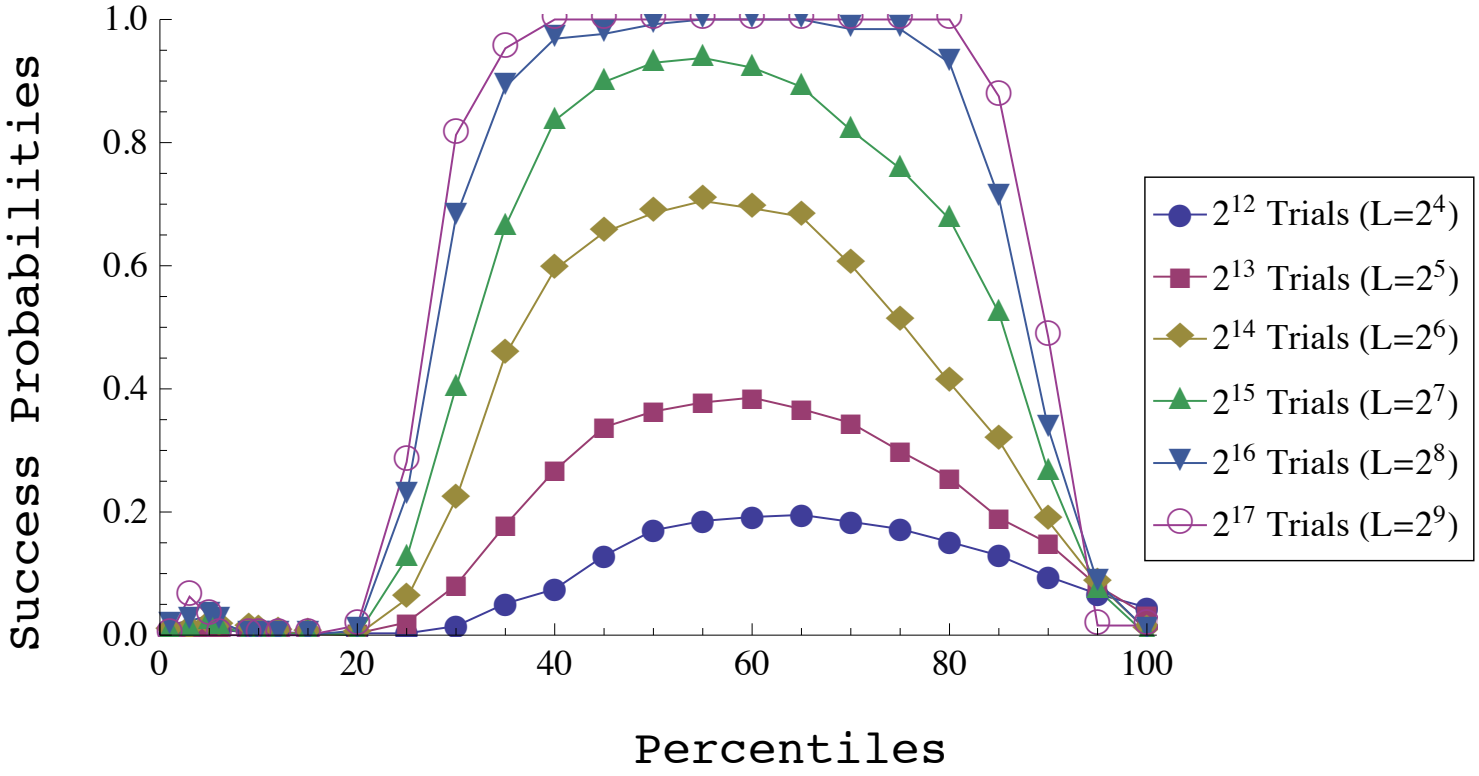
- BEAST-style attack targeting HTTP cookies.
  - Malicious client-side Javascript makes HTTP GET requests.
  - TLS sessions are automatically generated and HTTP cookies attached.
  - Pad GET requests so that 1-out-of-2 condition always holds.
  - Cost of attack is 2^{13} GET requests per byte of cookie.
  - Now a practical attack!

Experimental Results



- Byte 14 of plaintext set to 01; byte 15 set to FF.
- OpenSSLv1.0.1 on server running at 1.87Ghz.
- 100 Mbit LAN.
- Median times (noise not shown).

Experimental Results



OpenSSL: recovering last byte in a block, using percentile test to extract correct byte value, no assumptions on plaintext.

Lucky 13 – Further Extensions

- The attack extends to other MAC algorithms.
  - Nice interplay between block-size, MAC tag size and 13-byte field SQN || HDR.
- The attack extends to other methods for dealing with bad padding.
  - e.g. as in GnuTLS, faster but partial plaintext recovery.
- [The attack can be applied to DTLS.
  - No error messages, but simulate these via DTLS Heartbeats.
  - Errors non-fatal, so can execute attack in a single session.
  - Cam amplify timing differences using AlFardan-Paterson techniques (NDSS 2012).]

Lucky 13 – Impact

- **OpenSSL** patched in versions 1.0.1d, 1.0.0k and 0.9.8y, released 05/02/2013.
- **NSS** (Firefox, Chrome) patched in version 3.14.3, released 15/02/2013.
- **Opera** patched in version 12.13, released 30/01/2013
- **Oracle** released a special critical patch update of JavaSE, 19/02/2013.
- **BouncyCastle** patched in version 1.48, 10/02/2013
- Also **GnuTLS**, **PolarSSL**, **CyaSSL**, **MatrixSSL**.
- **Microsoft** "determined that the issue had been adequately addressed in previous modifications to their TLS and DTLS implementation".
- **Apple:** status unknown.

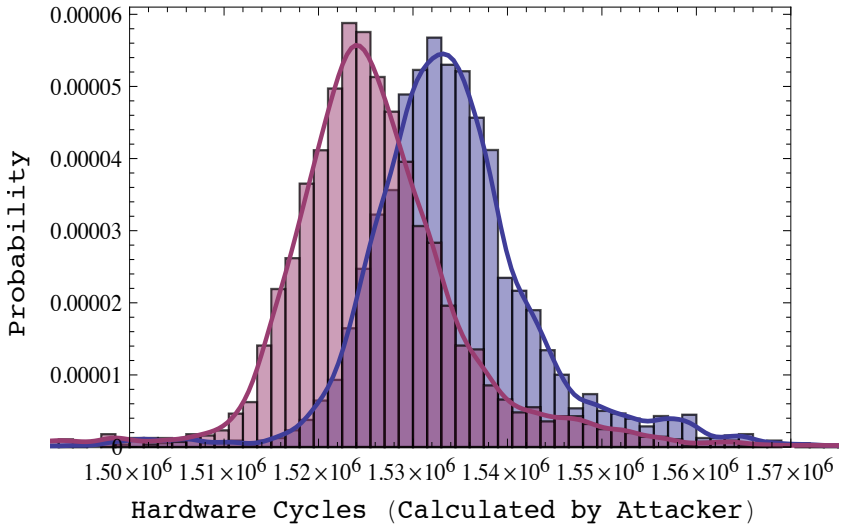
(Full details at: www.isg.rhul.ac.uk/tls/lucky13.html)

Lucky 13 – Countermeasures

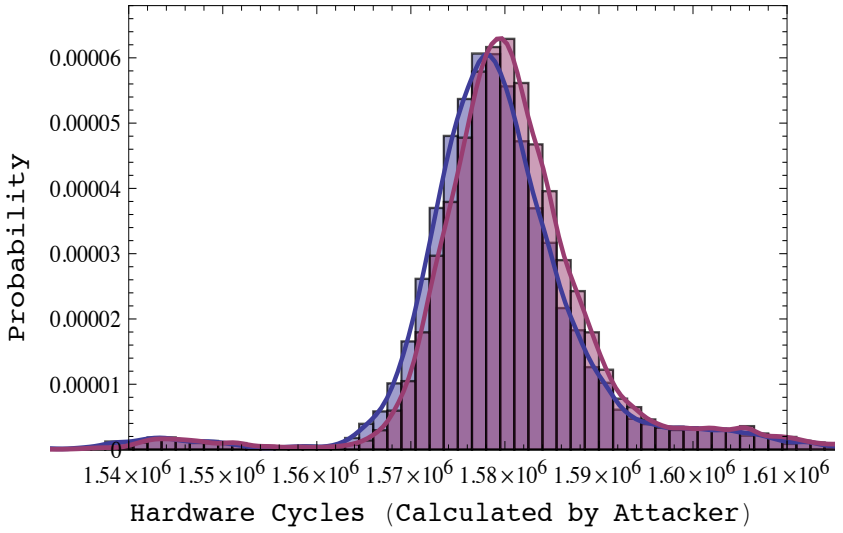
- We really need constant-time decryption for TLS-CBC.
- Add dummy hash compression function computations when padding is *good* to ensure total is the same as when padding is *bad*.
- Add dummy padding checks to ensure number of iterations done is independent of padding length and/or correctness of padding.
- Watch out for length sanity checks too.
  - Need to ensure there's enough space for *some* plaintext after removing padding and MAC, but without leaking any information about amount of padding removed.
- TL;DR:
  - It's a bit of a nightmare.

Performance of Countermeasures

Before



After



Outline

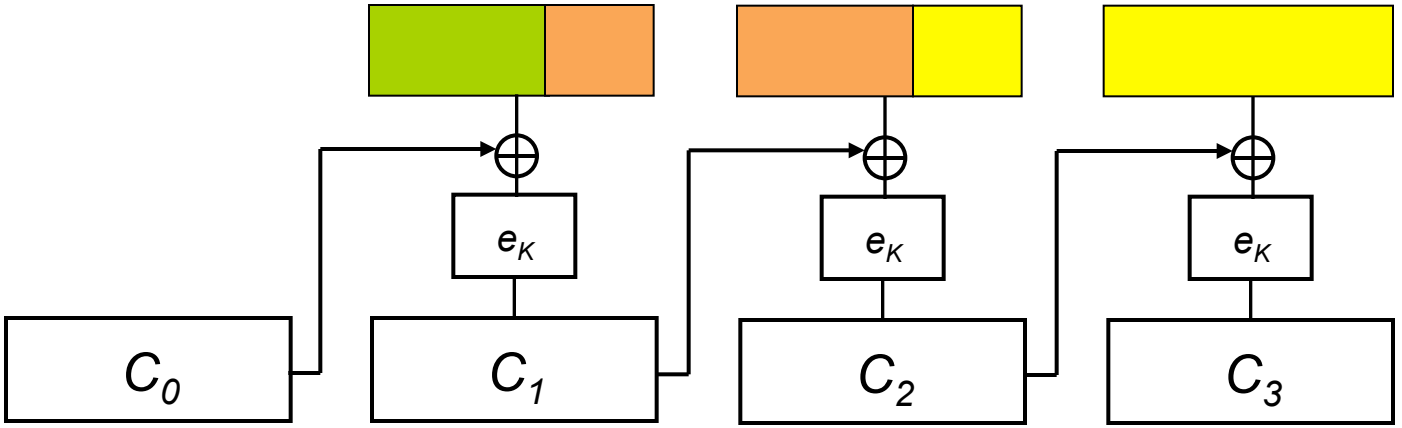
- TLS and the TLS Record Protocol
- Theory for TLS
- Attacks:
  - The BEAST
  - Padding oracles
  - Lucky 13
  - (More theory for TLS)
  - RC4 attack
  - CRIME/BREACH
- Discussion

More Theory for TLS

Theorem ([PRS11], informal statement)
Suppose E is a block cipher with block size n that is sprp-secure.
Suppose MAC has tag size t and is prf-secure.
Suppose that for all messages M queried by the adversary:

$$|M| + t \geq n.$$

Then MEE with CBC mode encryption, random IVs, TLS padding, and *uniform errors* is (LH)AE secure.



Other Lucky 13 Countermeasures?

- Introduce random delays during decryption.
  - Surprisingly *in*effective, analysis in [AP13].
- Redesign TLS:
  - Pad-MAC-Encrypt or Pad-Encrypt-MAC.
  - Currently, some discussion on TLS mailing lists.
  - No easy deployment route, seems unlikely to happen.
- Switch to TLS 1.2
  - Has support for AES-GCM and AES-CCM.
- Use RC4.

Outline

- TLS and the TLS Record Protocol
- Theory for TLS
- Attacks:
  - The BEAST
  - Padding oracles
  - Lucky 13
  - (More theory for TLS)
  - RC4 attack
  - CRIME/BREACH
- Discussion

RC4

- The RC4 stream cipher has long been known to have statistical weaknesses.
  - e.g. Mantin-Shamir bias, recent work of Maitra *et al.*, Isobe *et al.*
  - Most attention has been given to the initial few bytes of keystream.
  - The focus has been on finding and giving theoretical explanations for *individual* biases, and on key-recovery attacks.
- Usual countermeasure is to discard the initial bytes of keystream and use only “good” bytes.
- So what does RC4 in TLS do?

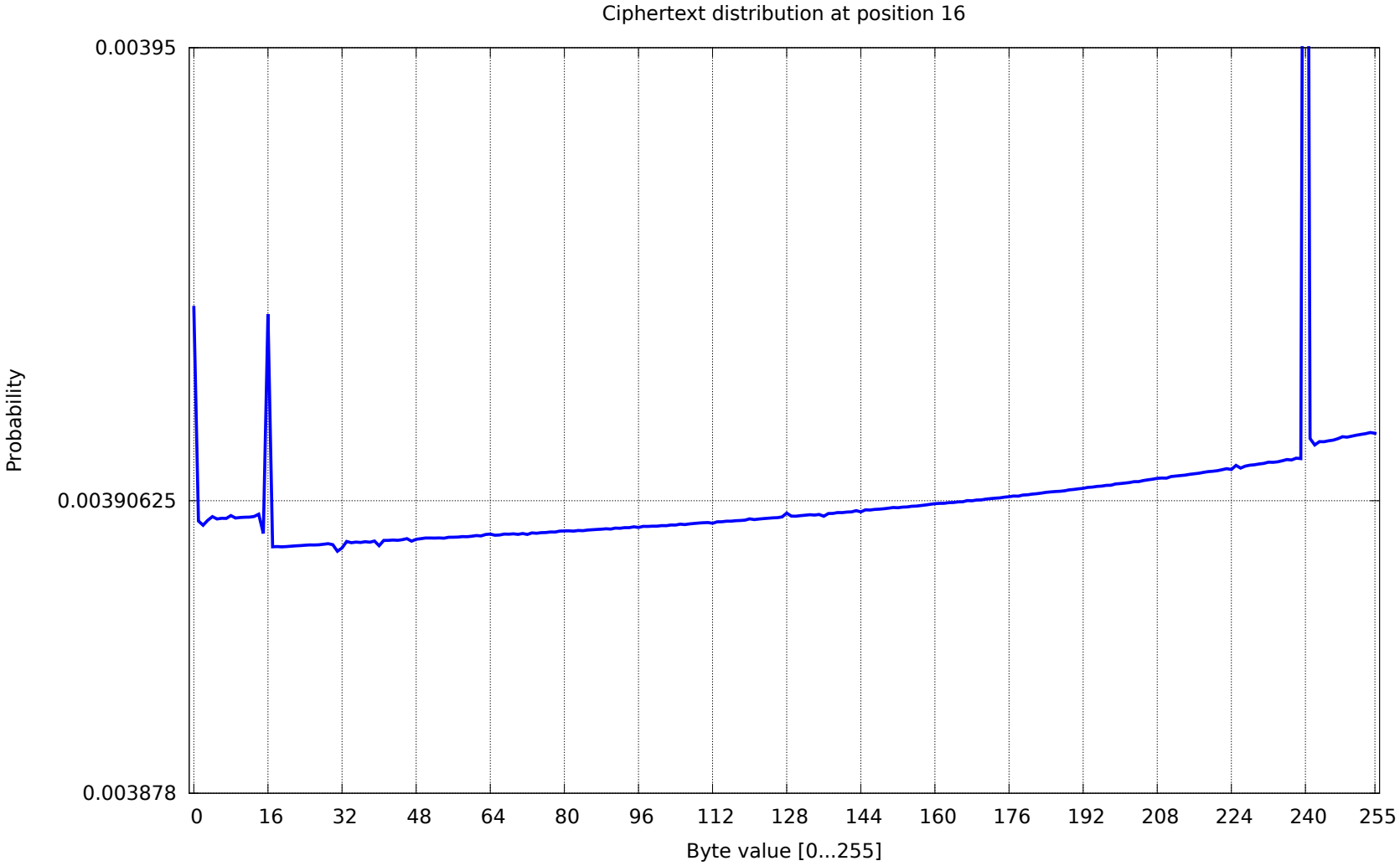
RC4 in TLS

- TLS does not discard bytes!
  - Because it hurts performance too much;
  - The biases are small anyway; and only exist in the first few bytes.

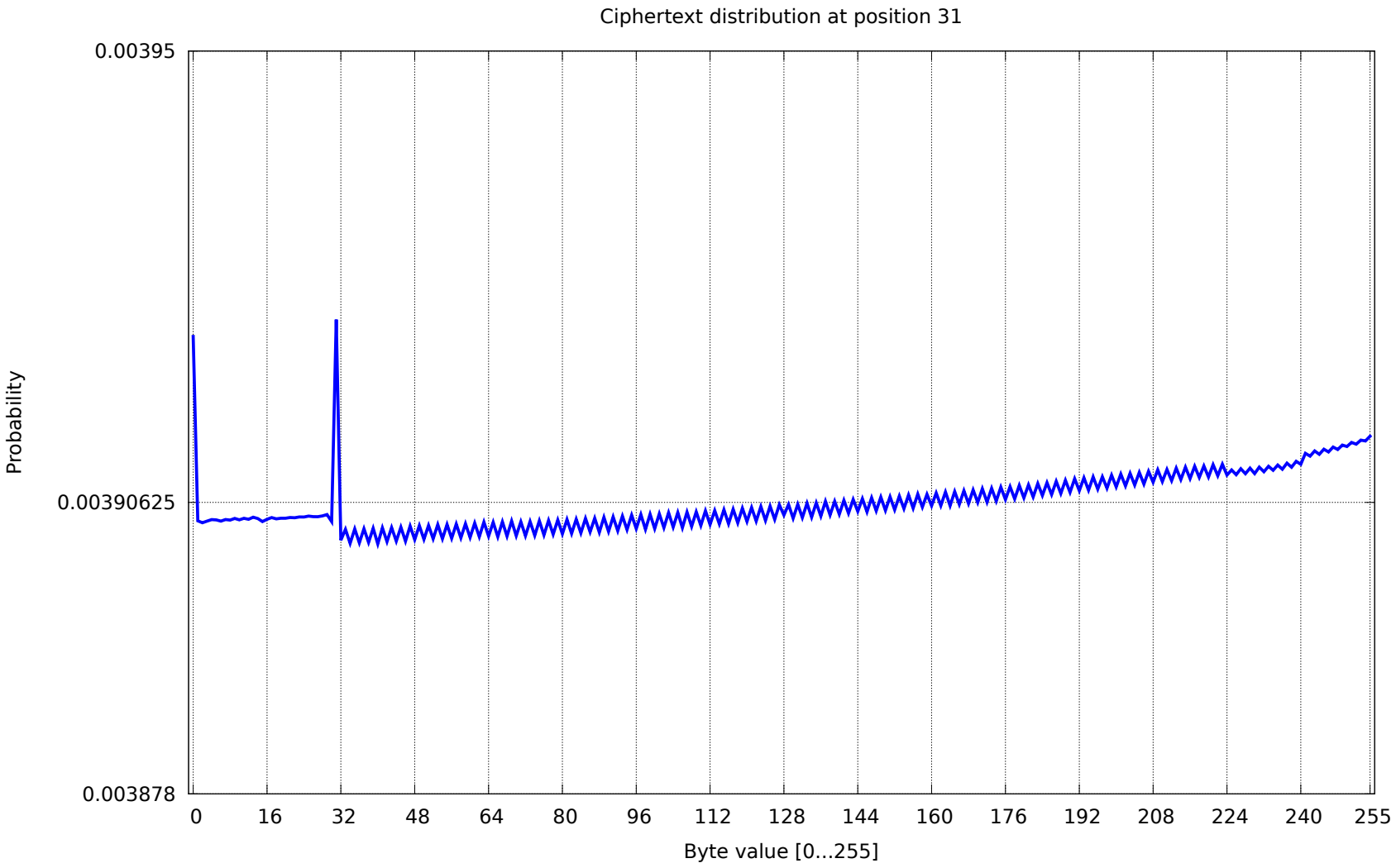
- [ABPPS13]: we estimated the biases in the first 256 output bytes by sampling RC4 keystreams for 2^{45} random 128-bit keys.

- We found many previously unreported biases of significant size…
  - www.isg.rhul.ac.uk/tls/biases.pdf

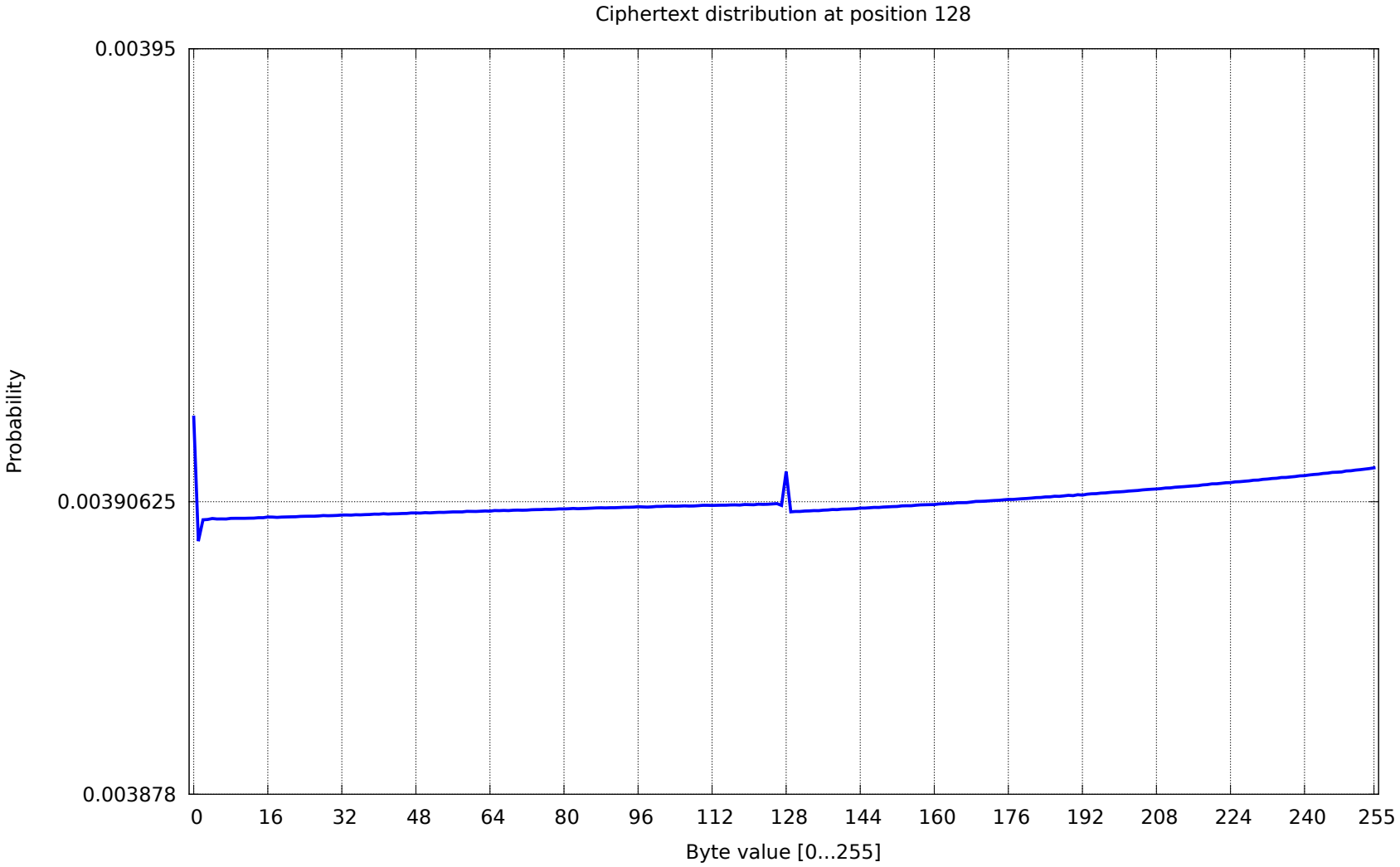
Biases in byte 16 of RC4 Output



Biases in byte 31 of RC4 Output



Biases in byte 128 of RC4 Output



TLS-RC4 Attack

- These biases are large enough to enable plaintext recovery attacks on first 256 bytes of TLS sessions.

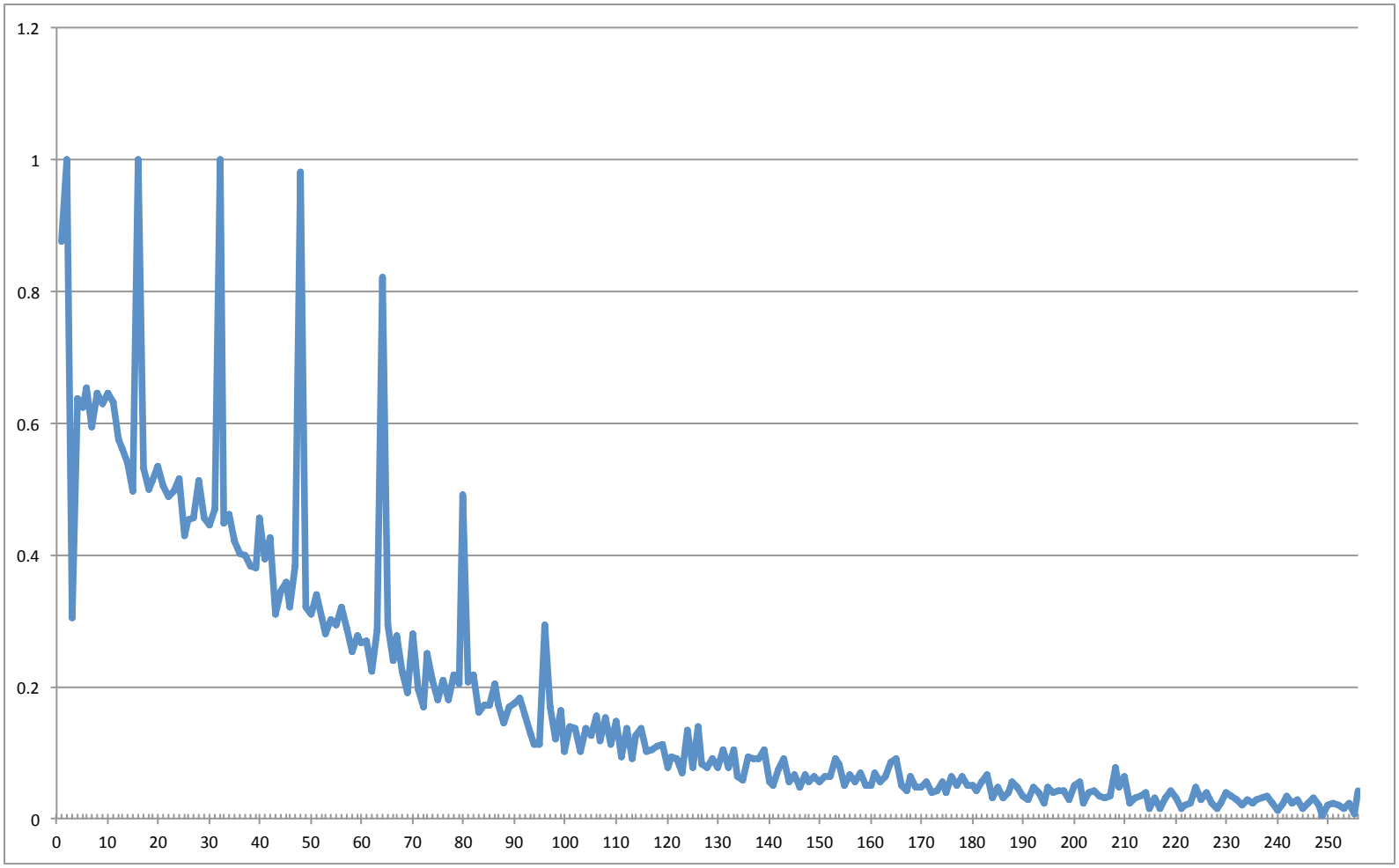
- Needs a multi-session attack.
  - BEAST-style malware as possible generation mechanism.

- Attack using simple Bayesian technique:
  - For each byte position i:

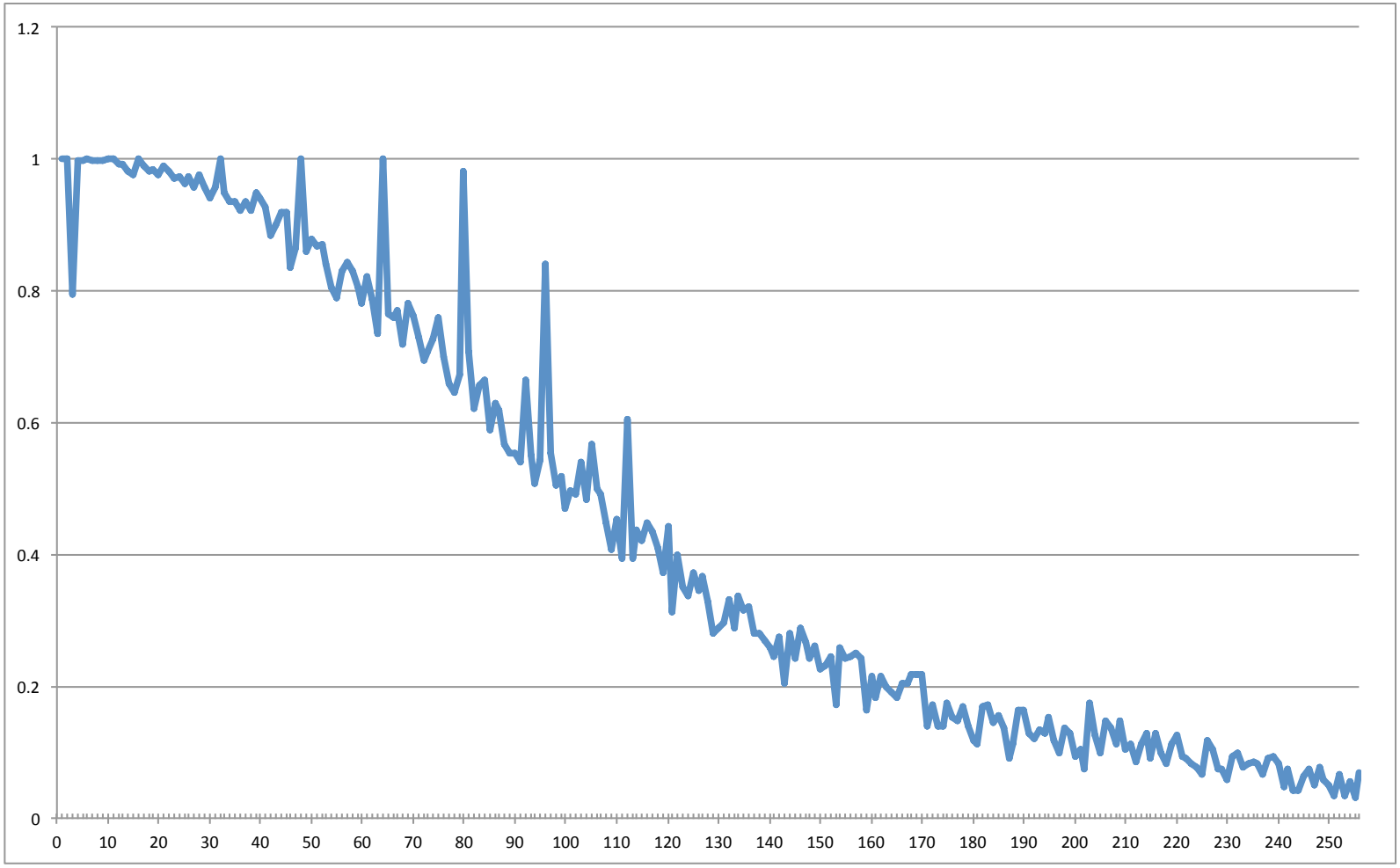
$$C_i = P_i \oplus K_i$$

  - Many samples of C_i, so each guess for P_i induces a distribution on K_i.
  - Estimate likelihood of induced distribution using measured distribution on K_i.
  - Select as correct plaintext byte the candidate P_i giving highest likelihood.

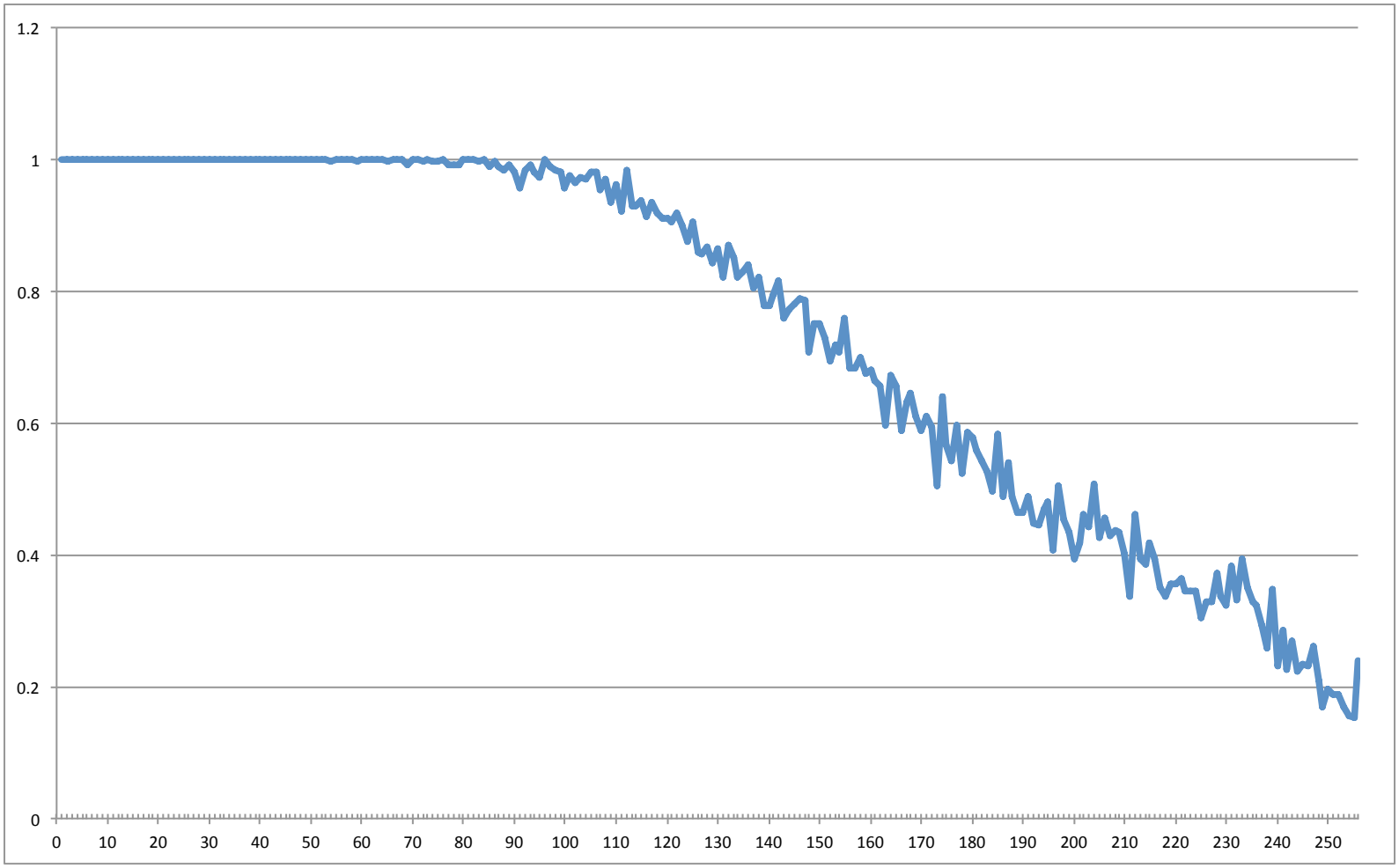
Success probability: 2^{24} sessions



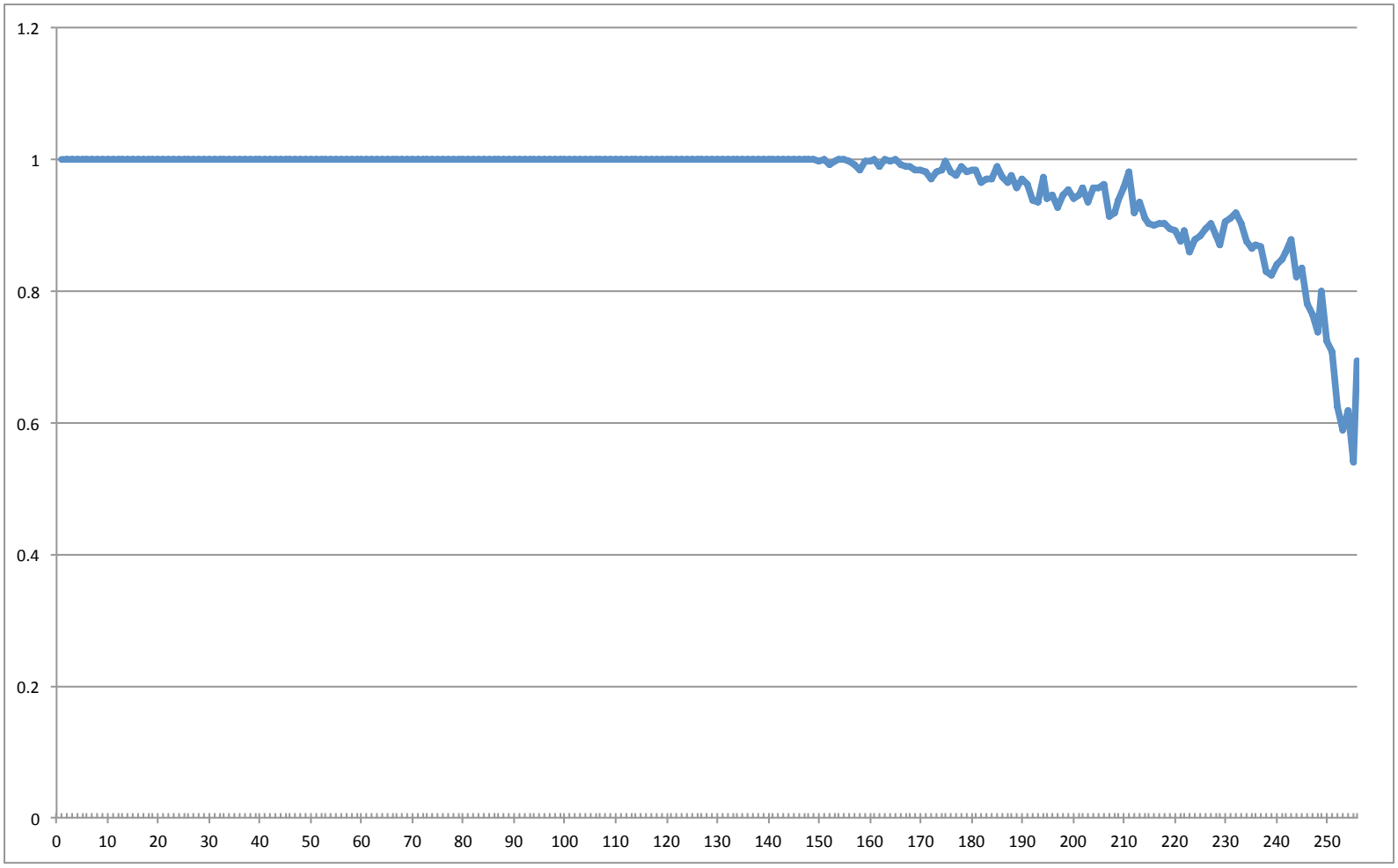
Success probability: 2^{26} sessions



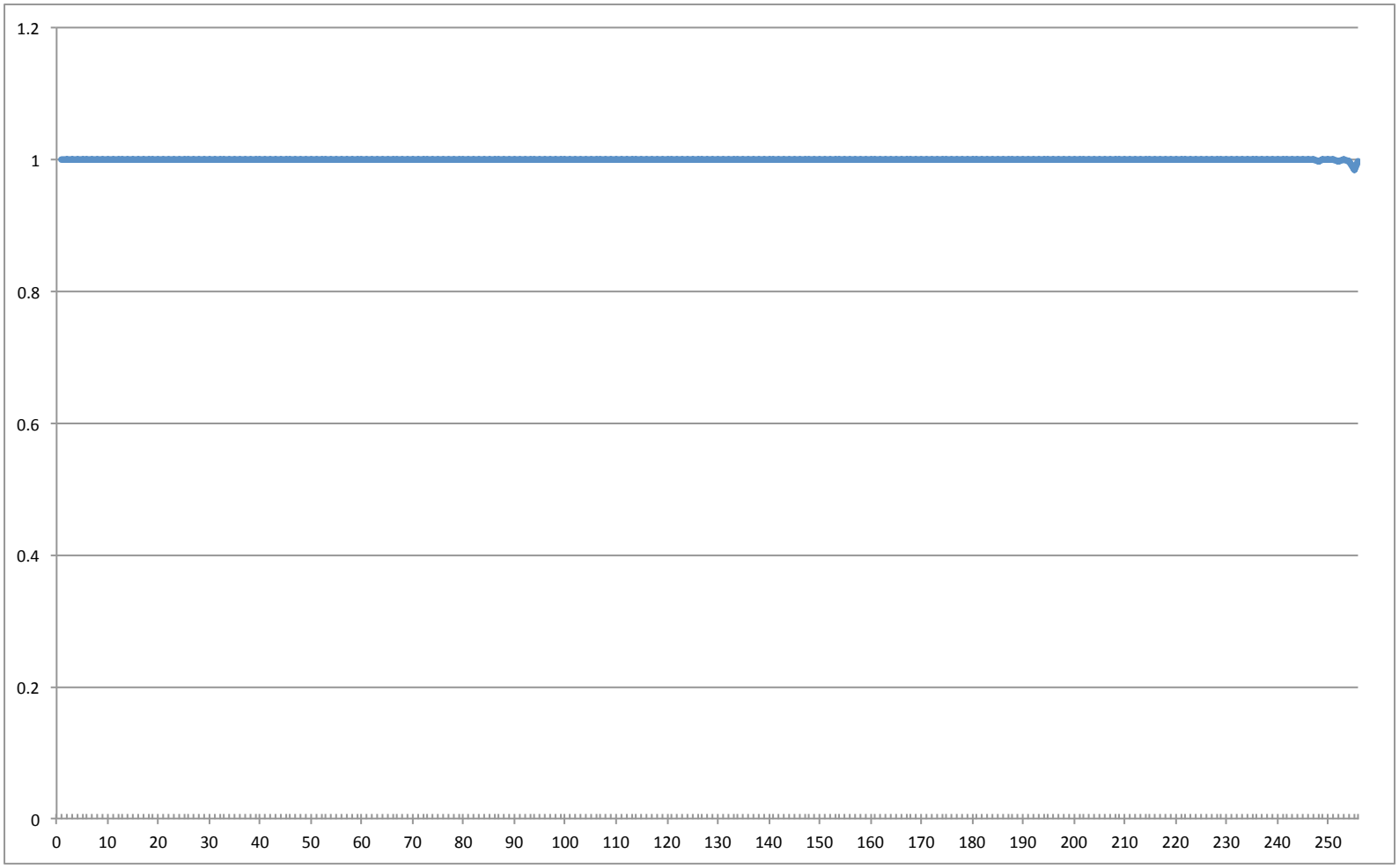
Success probability: 2^{28} sessions



Success probability: 2^{30} sessions



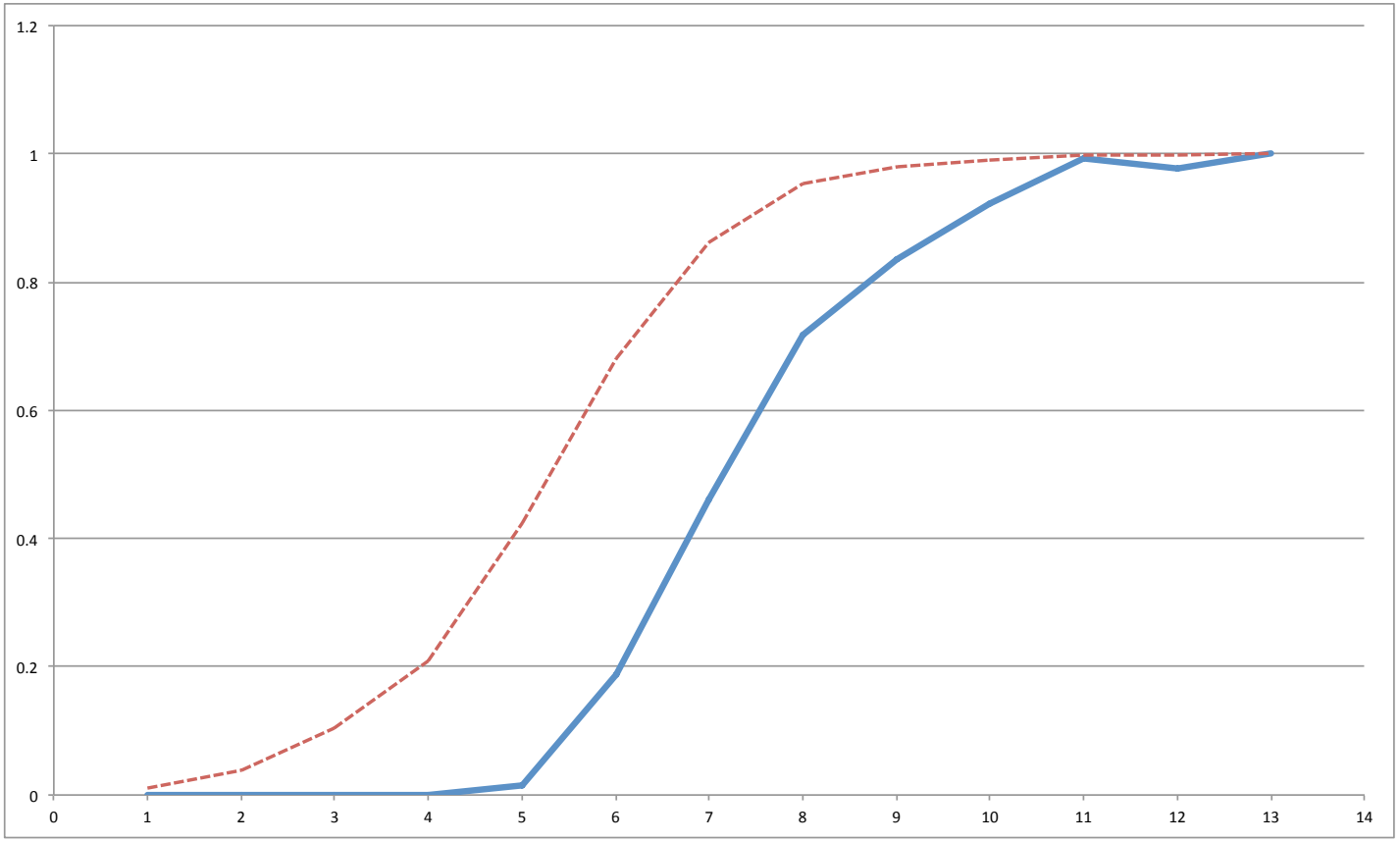
Success probability: 2^{32} sessions



Possible Attack Enhancements

- Many sessions are needed.
- The attack can only target the first 256 plaintext bytes.
  - Containing less interesting HTTP headers.
- In [ABPPS13], we solved both problems using 2-byte Fluhrer-McGrew biases.
  - Smaller biases, but persistent throughout keystream.
  - Arrange for HTTP session cookie to be repeatedly sent at predictable locations in keystream.
  - Use Viterbi-style algorithm to do ML estimation of plaintext bytes.
  - Roughly 2^{33} encryptions needed for reliable recovery.

Results for 2-byte Biases



x-axis: units of 2^{30} encryptions.

Blue line: success rate for 16-byte plaintext recovery.

Red line: success rate for individual byte recovery.

TLS-RC4 Attack – Countermeasures

- We can't just discard initial output bytes without updating all clients and servers simultaneously.
  - And this doesn't help against 2-byte attacks anyway.

- We had lots of discussion with vendors on *ad hoc* measures for HTTP.
  - Randomisation.
  - Burn-off initial bytes via short messages.
  - Put limits on number of times cookies can be sent.

TLS-RC4 Attack – Impact

- Fewer vendors have reacted publicly.
  - Google focussed on implementing TLS 1.2.
  - Microsoft disabled RC4 in Windows 8.1 Preview.
  - Opera has implemented cookie limit countermeasure.

- Further details at: www.isg.rhul.ac.uk/tls

Outline

- TLS and the TLS Record Protocol
- Theory for TLS
- Attacks:
  - The BEAST
  - Padding oracles
  - Lucky 13
  - (More theory for TLS)
  - RC4 attack
  - CRIME/BREACH
- Discussion

CRIME/BREACH

- Duong and Rizzo [DR12] found a way to exploit TLS's optional compression feature.
  - Similar to idea in 2002 paper by Kelsey.
- Compression algorithms are stateful.
  - Replace repeated strings by shorter references to previous occurrences.
- Degree of compression obtained for chosen plaintext reveals something about prior plaintexts!
- This small amount of leakage can be boosted to get plaintext recovery attack for HTTP cookies.
  - Using same chosen plaintext vector as for BEAST.
- Countermeasures: disable compression; use variable length padding.
- BREACH: similar ideas, applied to HTTP compression.

Outline

- TLS and the TLS Record Protocol
- Theory for TLS
- Attacks:
  - The BEAST
  - Padding oracles
  - Lucky 13
  - (More theory for TLS)
  - RC4 attack
  - CRIME/BREACH
- Discussion

Where Do We Stand Currently?

- Most TLS implementations now patched against BEAST.
- Many TLS implementations patched against Lucky 13.
- No simple TLS patch for RC4 attack.
  - Needs application-layer modifications.
- Disable TLS compression to prevent CRIME.
  - Still issues with compression at application layer (BREACH).

- We need TLS 1.2!
  - Use patched CBC-mode until we get it.
  - Other people advise differently.
    - Their logic is that BEAST-vulnerable browsers still exist, so less-broken RC4 is preferable.

Discussion

- TLS's *ad hoc* MAC-Encode-Encrypt construction is hard to implement securely and hard to prove positive security results about.
  - Long history of attacks and fixes.
  - Each fix was the "easiest option at the time".
  - Now reached point where a 500 line patch to OpenSSL was needed to fully eliminate the Lucky 13 attack.
  - Attacks show that small details matter.
  - The actual TLS-CBC construction was only fully analysed in 2011.

- RC4 was known to be weak for many years.
  - Actual exploitation of weaknesses in a TLS context went unexplored.
  - Needed multi-session mechanism (BEAST technology) to make the attack plausible.

Discussion

- Once a bad cryptographic choice is out there in implementations, it's very hard to undo.
  - Old versions of TLS hang around for a long time.
  - There is no TLS product recall programme!
  - Slow uptake of TLS 1.1, 1.2.

- TLS is coming under sustained pressure from attacks.
  - BEAST, Lucky 13 and RC4 attacks are providing incentives to move to TLS 1.2.
  - Attacks are "semi-practical" but we ignore such attacks at our peril.
  - Good vendor response to Lucky 13, less so to RC4 attack.
    - One is fixable, the other not (really).

Lessons for CAESAR

- Attacks really do improve with age.
  - BEAST (1995 – 2011), Lucky 13 (Feb. ’13 – Mar. ‘13).

- Design AE schemes for a broad set of use-cases and attack vectors.
  - Fixed-key, many-key (multi-session) attacks.
  - Chosen-plaintext, partially known-plaintext, …
  - Compression-based attacks?
  - Look carefully at error conditions and handling.

- Be realistic about timescale for adoption in already deployed systems.

Research Directions?

- TLS Record Protocol cryptography has now been heavily analysed.
  - Still some mileage in looking at AE implementations?
- Major recent progress in analysing TLS Handshake protocol.
  - [JKSS12], [KPW13].
- Can still expect implementation issues to emerge.
  - Check the “OpenSSL Fact” twitter feed regularly!
- Complex system of interacting protocols can still throw up surprises.
  - Alert Protocol desynchronisation attack [BFKPS13].
  - TLS Renegotiation attack [RD09].