What's the matter with TLS?

Matthew Green
Johns Hopkins University

Why this presentation?

- 'Cause TLS is the most important protocol in the world! (after the null cipher)
- It's also:
  - badly designed
  - under-analyzed
  - poorly implemented
  - lousy with extensions

Why this presentation?

- 'Cause TLS is the most important protocol in the world! (after the null cipher)
- It's also:
  - badly designed
  - under-analyzed
  - poorly implemented
  - lousy with extensions
  - Yet... amazingly robust despite all that

But here's the great news

NEWS

Facebook Adopts Secure Web Pages By Default

Mathew J. Schwartz

See more from Mathew

Connect directly with Mathew: Bio | Contact

Facebook has finally started using HTTPS by default, following a 2010 FTC demand and in the distant footsteps of Google, Twitter, and Hotmail.

2 Comments | Tweet | 8 +1 | 0 Share | Mail | Print

Mathew J. Schwartz | November 19, 2012 11:11 AM

Facebook has begun making HTTPS, which provides SSL/TLS encryption, the default protocol for accessing all pages on its site.

This presentation

Question 1

What's the matter with TLS?
Designwise/
Analysis-wise/
Implementation-wise?

This presentation

Question II

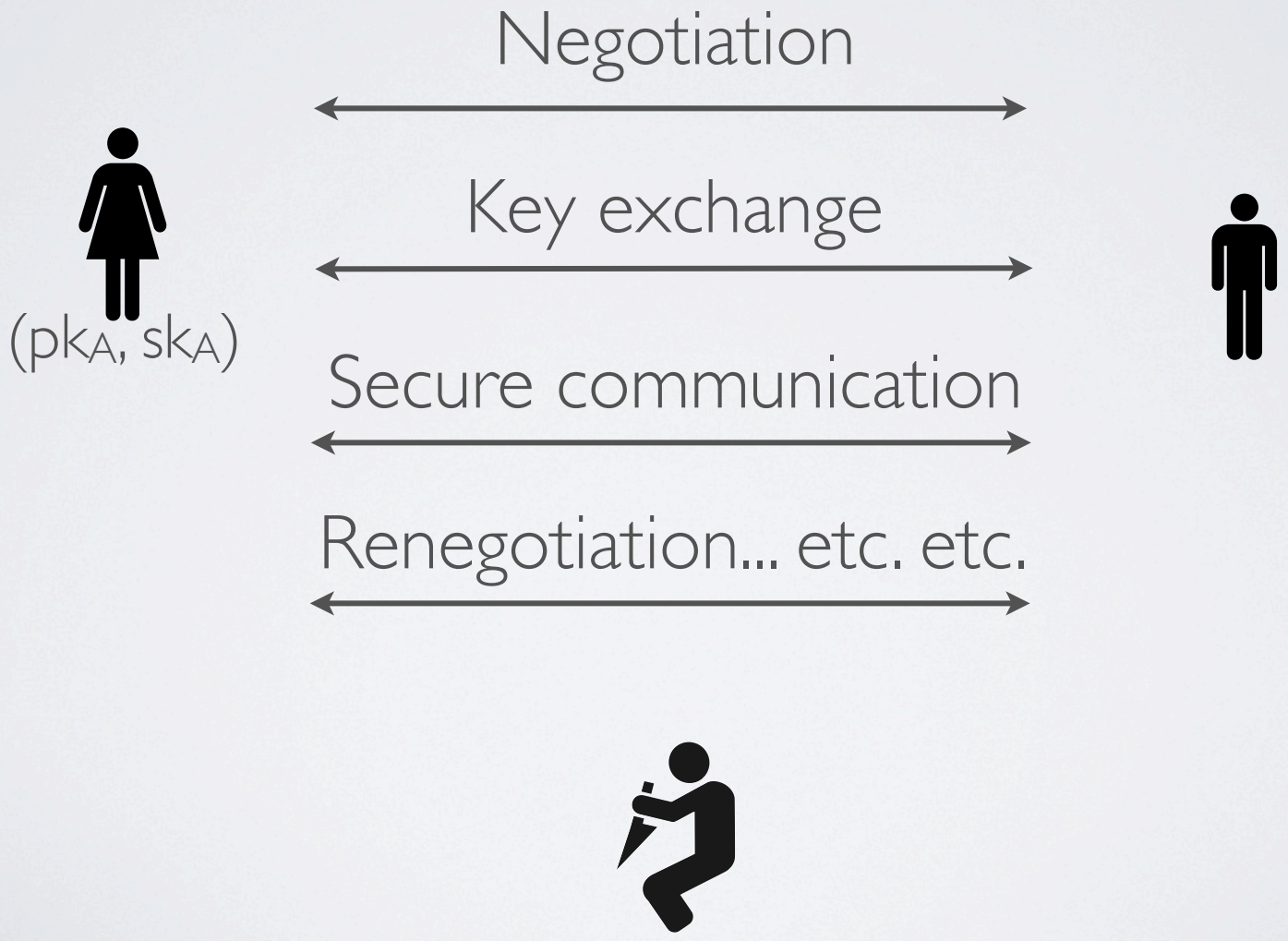
Which areas
[design, research, implementation]
should we be thinking about improving/replacing/analyzing?

(with applications to PhD students)

A brief history

- SSLv1 born at Netscape (~1995)
- SSLv2 released one year later
  - Serious protocol negotiation bugs. Don't even go here.
- SSLv3
  - Slightly less serious issues [Schneier & Kelsey, others] padding oracles
- TLSv1.0
  - Predictable IVs, renegotiation attacks and more!
- TLSv1.1, TLSv1.2 (lightly adopted)

The protocol



Design

It's bad

- Many problems result from TLS's use of "*pre-historic cryptography*" (- Eric Rescorla)
  - CBC with Mac-Pad-then-Encrypt
  - RSA-PKCS#1v1.5 encryption padding
  - RC4 with.....
  - Goofy stuff: ephemeral RSA key agreement, GOST ciphersuite, Snap Start, False Start (withdrawn)
  - Horrifying backwards compatibility requirements

MAC-then-pad-then-Encrypt

- TLS MACs the record, then pads (in CBC), then enciphers
  - Obvious problem: padding oracles

MAC-then-pad-then-Encrypt

- TLS MACs the record, then pads (in CBC), then enciphers
  - Obvious problem: padding oracles
  - Countermeasure(s):
    1. Do not distinguish padding/MAC failure

In theory, this works (if the MAC is larger than the block size)
-Paterson et al. '11

MAC-then-pad-then-Encrypt

- TLS MACs the record, then pads (in CBC), then enciphers
  - Obvious problem: padding oracles
  - Countermeasure(s):
    1. Do not distinguish padding/MAC failure
    2. "Constant-time" decryption

Probably good enough for remote timing...

MAC-then-pad-then-Encrypt

- TLS MACs the record, then pads (in CBC), then enciphers
  - Obvious problem: padding oracles
  - Countermeasure(s):
    1. Do not distinguish padding/MAC failure
    2. "Constant-time" decryption
    3. Kill session

Doesn't work against sessions with repeated plaintext (cookies)

BEAST

- Use of predictable IV (CBC residue bug)
  - Known since 2002, attack described by Bard in 2005 (*Bard was advised to focus on more interesting problems.*)
- Also apparently nobody read the spec (because there are crazy things in that spec)

> The following alternative procedure MAY be used; however, it has not been demonstrated to be as cryptographically strong as the above procedures. The sender prepends a fixed block F to the plaintext (or, alternatively, a block generated with a weak PRNG). He then encrypts as in (2), above, using the CBC residue from the previous block as the mask for the prepended block. Note that in this case the mask for the first record transmitted by the application (the Finished) MUST be generated using a cryptographically strong PRNG.

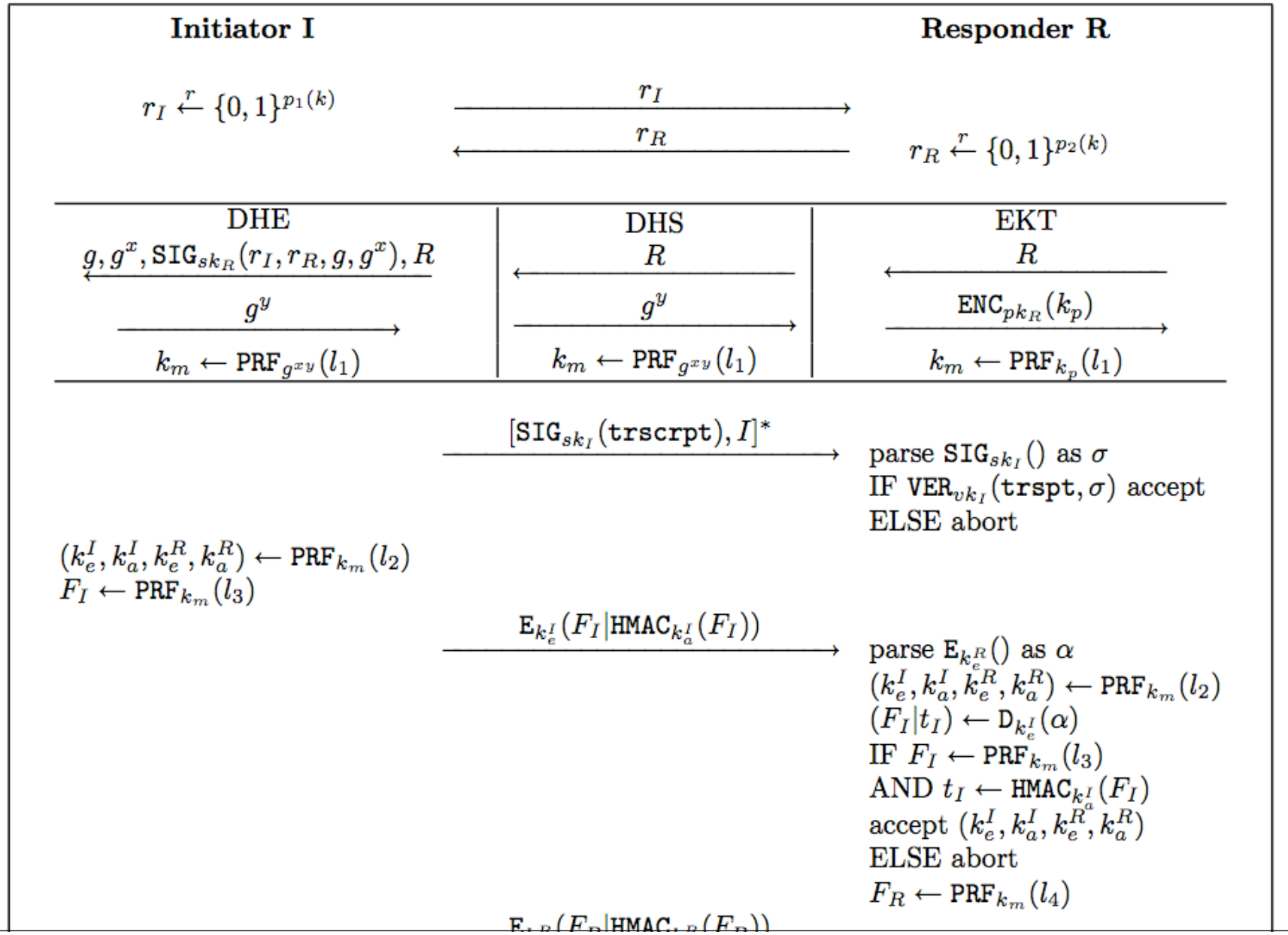
Solution in practice: RC4

:-(

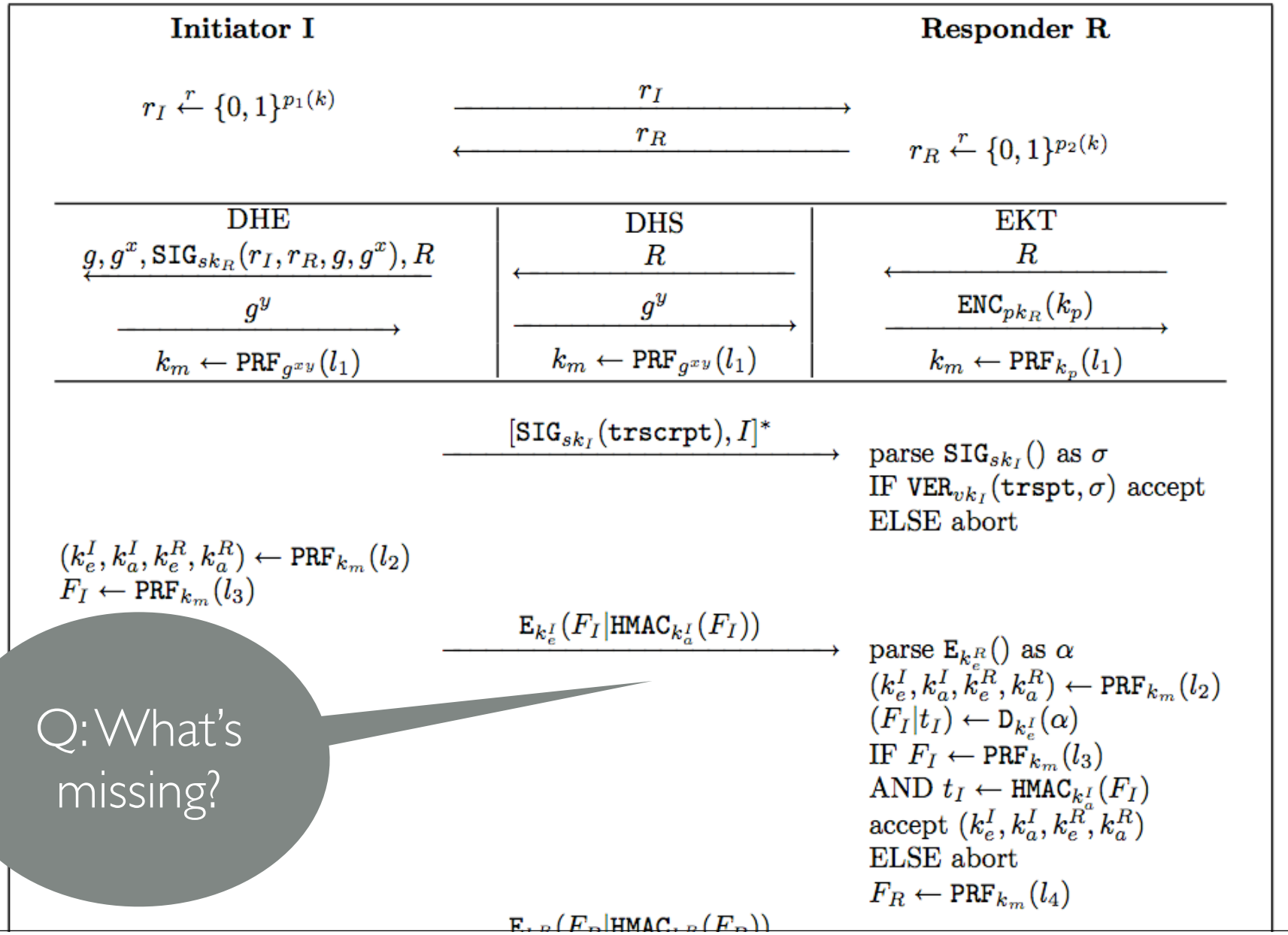
(When RC4 is your solution,
you need a better problem)

Analysis

TLS for cryptographers



TLS for cryptographers



Q: What's missing?

Example: Negotiation

Each TLS protocol begins with a ciphersuite negotiation that determines which key agreement protocol (etc.) will be used.

Tolga Acar, Mira Belenkiy, Mihir Bellare, and David Cash, Cryptographic Agility and its Relation to Circular Encryption, in EUROCRYPT 2010

Example: Negotiation

The key agreement secures the negotiation (!)

Tolga Acar, Mira Belenkiy, Mihir Bellare, and David Cash, Cryptographic Agility and its Relation to Circular Encryption, in EUROCRYPT 2010

Example: Negotiation

Negotiation messages are protected by hashing them, then computing a MAC on the resulting hash *after* key agreement has completed.

Tolga Acar, Mira Belenkiy, Mihir Bellare, and David Cash, Cryptographic Agility and its Relation to Circular Encryption, in EUROCRYPT 2010

Surely we've analyzed this

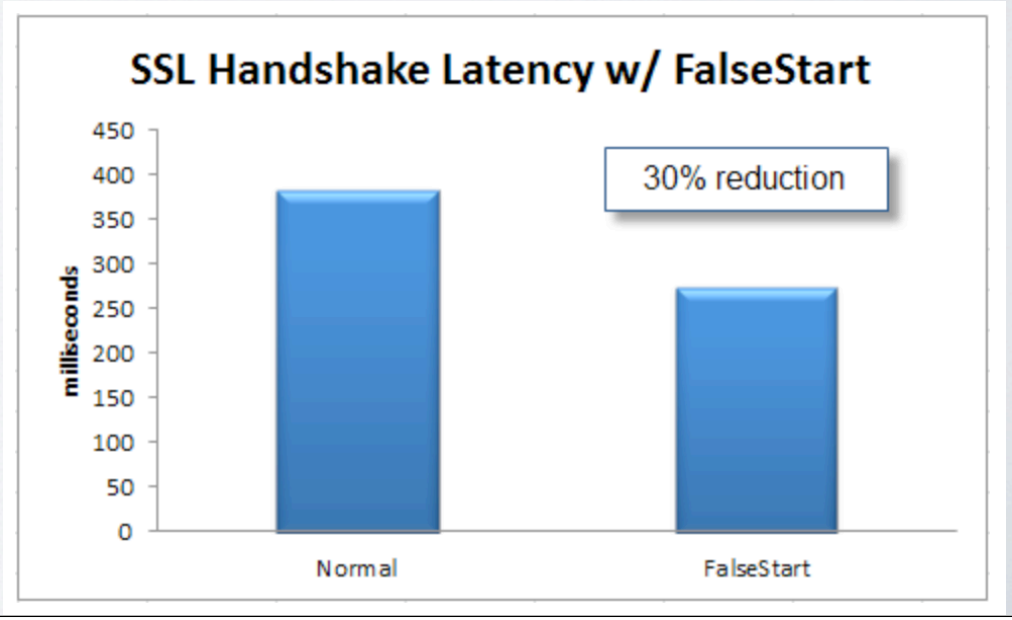
- Well -- not really.
  - In CRYPTO 2012 (!) we saw the first paper to successfully analyze TLS-DHE [Jager *et al.*]
  - It didn't consider any other protocols, or the negotiation
- To date: no published work that analyzes even the TLS-RSA handshake (in a realistic setting)
- Any research on negotiate/hash/exchange pattern?

Compression (CRIME)

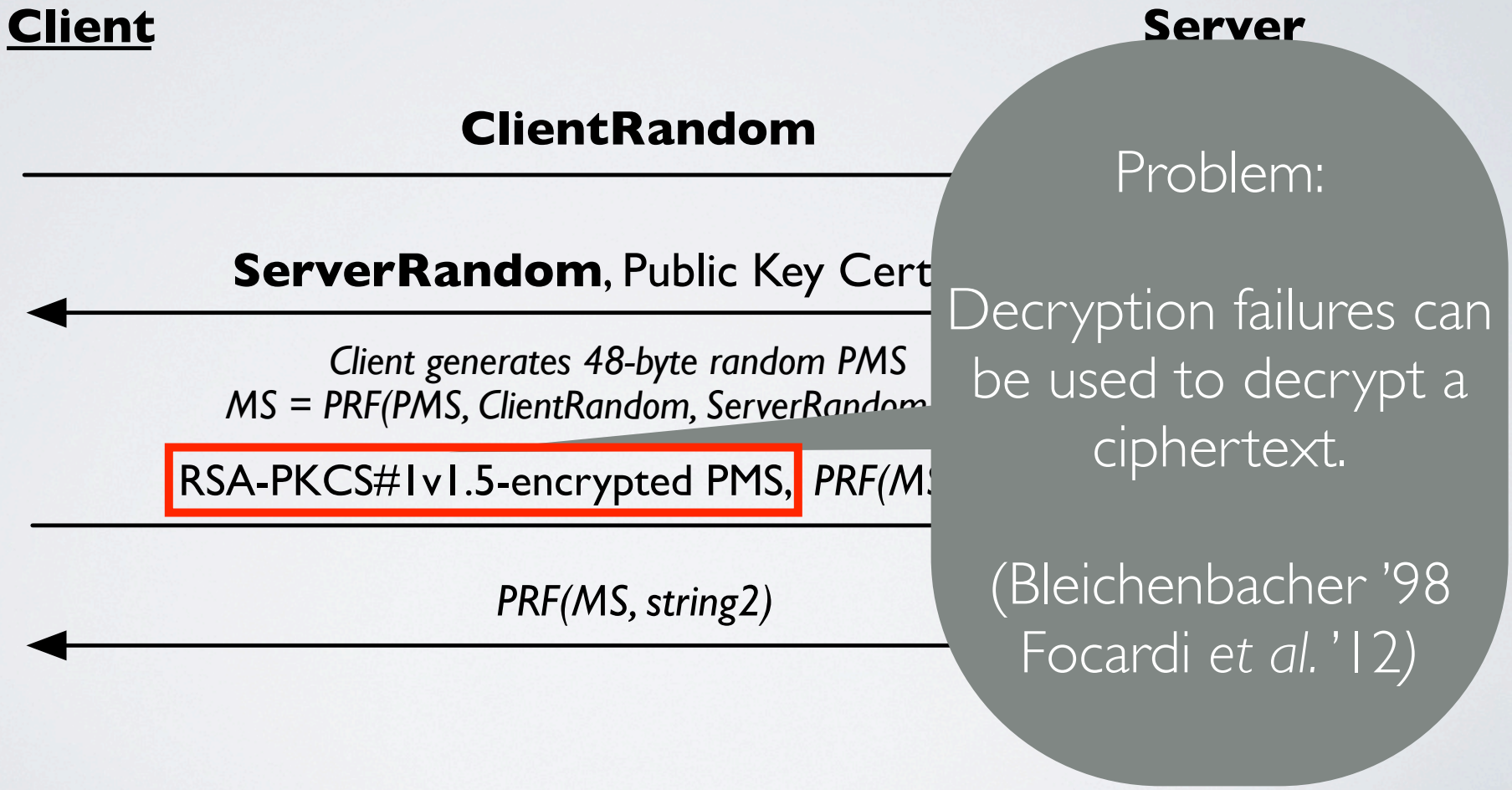
- Can't really blame the TLS designers for including it...
  - Blame cryptographers for not noticing it's still in use?
  - Blame cryptographers for pretending it would go away.
- We need a model for compression+encryption
  - Clearly this is weaker than semantic security
  - *But how much weaker? Can we quantify?*

Extensions

- TLS False Start
  - (Now withdrawn)
  - Completing the TLS handshake is expensive
  - So let's start transmitting before the final Exchange of MACs (FINISH)

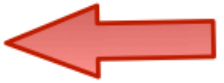


PKCS #1v1.5


Client
Server
ClientRandom
ServerRandom, Public Key Cert
Client generates 48-byte random PMS
MS = PRF(PMS, ClientRandom, ServerRandom
RSA-PKCS#1v1.5-encrypted PMS,
PRF(MS, string2)
Problem:
Decryption failures can be used to decrypt a ciphertext.
(Bleichenbacher '98
Focardi et al. '12)

PKCS #1v1.5

RFC 5246: TLS 1.2

1. Generate a string R of 46 random bytes  (1)

2. Decrypt the message to recover the plaintext M  (2)

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  (3)

   else:

   pre_master_secret = ClientHello.client_version || M[2..47]

This approach can be proven secure, but only under a goofy made-up assumption.

Implementation

Everything up 'til now was the good news.

OpenSSL, GnuTLS, NSS

- The problem with TLS is that we are cursed with implementations
  - OpenSSL being the chief offender
  - But followed closely...

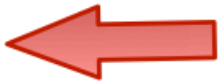
```
    if (bio == NULL)
        {
        if (PKCS7_is_detached(p7))
            bio=BIO_new(BIO_s_null());
        else if (os && os->length > 0)
            bio = BIO_new_mem_buf(os->data, os->length);
        if(bio == NULL)
            {
            bio=BIO_new(BIO_s_mem());
            if (bio == NULL)
                goto err;
            BIO_set_mem_eof_return(bio,0);
            }
        }
    if (out)
        BIO_push(out,bio);
    else
        out = bio;
    bio=NULL;
    if (0)
        {
err:
        if (out != NULL)
            BIO_free_all(out);
        if (btmp != NULL)
            BIO_free_all(btmp);
        out=NULL;
        }
    return(out);
    }
```

```
        {
        BIO_printf(b,"%ld bytes leaked in %d chunks\n",
                ml.bytes,ml.chunks);
#ifdef CRYPTO_MDEBUG_ABORT
        abort();
#endif
        }
    else
        {
        /* Make sure that, if we found no leaks, memory-leak debugging itself
         * does not introduce memory leaks (which might irritate
         * external debugging tools).
         * (When someone enables leak checking, but does not call
         * this function, we declare it to be their fault.)
         *
         * XXX    This should be in CRYPTO_mem_leaks_cb,
         * and CRYPTO_mem_leaks should be implemented by
         * using CRYPTO_mem_leaks_cb.
         * (Also there should be a variant of lh_doall_arg
         * that takes a function pointer instead of a void *;
         * this would obviate the ugly and illegal
         * void_fn_to_char kludge in CRYPTO_mem_leaks_cb.
         * Otherwise the code police will come and get us.)
         */
        int old_mh_mode;

        CRYPTO_w_lock(CRYPTO_LOCK_MALLOC);
```

```
        {
        goto end;
        }

    i=make_REQ(req,pkey,subj,multirdn,!x509, chtype);
    subj=NULL; /* done processing '-subj' option */
    if ((kludge > 0) && !sk_X509_ATTRIBUTE_num(req->req_info->attrib
        {
        sk_X509_ATTRIBUTE_free(req->req_info->attributes);
        req->req_info->attributes = NULL;
        }
    if (!i)
        {
        BIO_printf(bio_err,"problems making Certificate Request\n");
        goto end;
        }
    }
if (x509)
    {
    EVP_PKEY *tmppkey;
    X509V3_CTX ext_ctx;
    if ((x509ss=X509_new()) == NULL) goto end;

    /* Set version to V3 */
    if(extensions && !X509_set_version(x509ss, 2)) goto end;
    if (serial)
        {
        if (!X509_set_serialNumber(x509ss, serial)) goto end;
        }
    else
        {
```

PKCS #1v1.5

RFC 5246: TLS 1.2

1. Generate a string R of 46 random bytes  (1)

2. Decrypt the message to recover the plaintext M  (2)

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  (3)

   else:

   pre_master_secret = ClientHello.client_version || M[2..47]

PKCS #1v1.5

RFC 5246: TLS 1.2

1. Generate a string R of 46 random bytes (1)

2. Decrypt the message to recover the plaintext M (2)

3. If the PKCS#1 padding is not correct, or the length of message M is not exactly 48 bytes:
   pre_master_secre
   else If ClientHello
   number check is exp
   pre_master_secre
   else:
   pre_master_secre

OpenSSL 1.0.1c

```
i=RSA_private_decrypt((int)n,p,p,rsa,RSA_PKCS1_PADDING);   (2)

al = -1;

if (al != -1)
    {
    /* Some decryption failure -- use random value instead as countermeasure
     * against Bleichenbacher's attack on PKCS #1 v1.5 RSA padding
     * (see RFC 2246, section 7.4.7.1). */
    ERR_clear_error();
    i = SSL_MAX_MASTER_KEY_LENGTH;
    p[0] = s->client_version >> 8;
    p[1] = s->client_version & 0xff;
    if (RAND_pseudo_bytes(p+2, i-2) <= 0)                   (1)
    /* should be RAND_bytes, but we cannot work around a failure */
        goto err;
    }

s->session->master_key_length=
    s->method->ssl3_enc->generate_master_secret(s,          (3)
        s->session->master_key,
        p,i);
```

PKCS #1v1.5

RFC 5246: TLS 1.2

```
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_secre
   else If ClientHello
   number check is exp
      pre_master_secre
   else:
      pre_master_secre
```

(1)

(2)

OpenSSL 1.0.1c

```
i=RSA_private_decrypt((int)n,p,p,rsa,RSA_PKCS1_PADDING);

al = -1;



if (al != -1)
    {
    /* Some decryption failure -- use random value instead as countermeasure
     * against Bleichenbacher's attack on PKCS #1 v1.5 RSA padding
     * (see RFC 2246, section 7.4.7.1). */
    ERR_clear_error();
    i = SSL_MAX_MASTER_KEY_LENGTH;
    p[0] = s->client_version >> 8;
    p[1] = s->client_version & 0xff;
    if (RAND_pseudo_bytes(p+2, i-2) <= 0)
    /* should be RAND_bytes, but we cannot work around a failure */
        goto err;
    }

s->session->master_key_length=
    s->method->ssl3_enc->generate_master_secret(s,
        s->session->master_key,
        p,i);
```

(2)

(1)

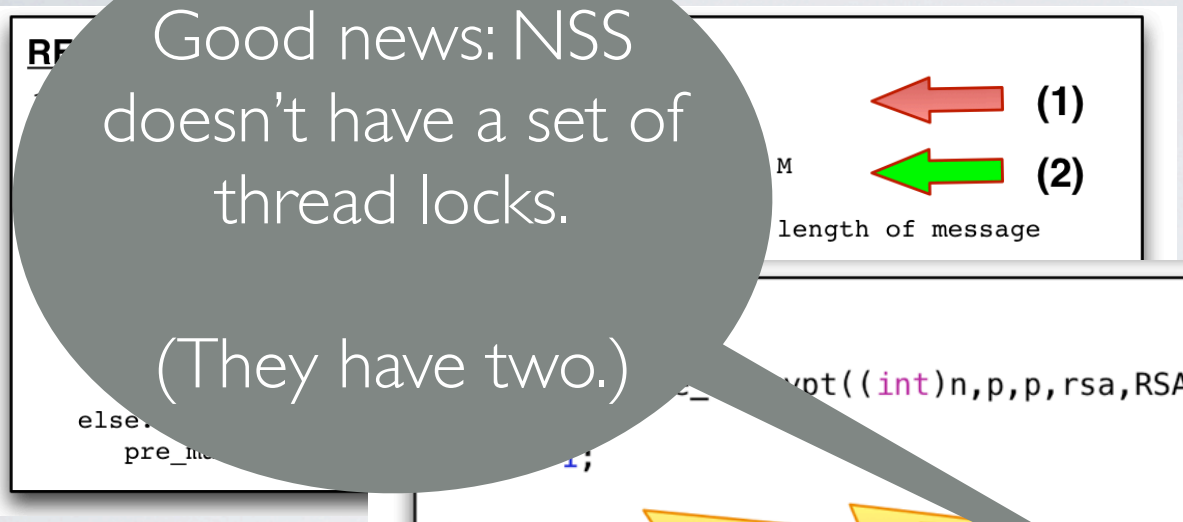
(3)

Thread locks

PKCS #1v1.5

Good news: NSS doesn't have a set of thread locks.

(They have two.)




```
                                                                    (2)
vpt((int)n,p,p,rsa,RSA_PKCS1_PADDING);

if (al != -1)
    {
    /* Some decryption failure -- use random value instead as countermeasure
     * against Bleichenbacher's attack on PKCS #1 v1.5 RSA padding
     * (see RFC 2246, section 7.4.7.1). */
    ERR_clear_error();
    i = SSL_MAX_MASTER_KEY_LENGTH;
    p[0] = s->client_version >> 8;
    p[1] = s->client_version & 0xff;
    if (RAND_pseudo_bytes(p+2, i-2) <= 0)                           (1)
    /* should be RAND_bytes, but we cannot work around a failure */
        goto err;
    }

s->session->master_key_length=
    s->method->ssl3_enc->generate_master_secret(s,                  (3)
        s->session->master_key,
        p,i);
```

Why do it the simple way?

3. *EMSA-PKCS1-v1_5 encoding*: Apply the EMSA-PKCS1-v1_5 encoding operation (Section 9.2) to the message M to produce a second encoded message EM' of length k octets:

$$EM' = \text{EMSA-PKCS1-v1\_5-ENCODE}\ (M, k)\ .$$

If the encoding operation outputs "message too long," output "message too long" and stop. If the encoding operation outputs "intended encoded message length too short," output "RSA modulus too short" and stop.

4. Compare the encoded message EM and the second encoded message EM'. If they are the same, output "valid signature"; otherwise, output "invalid signature."

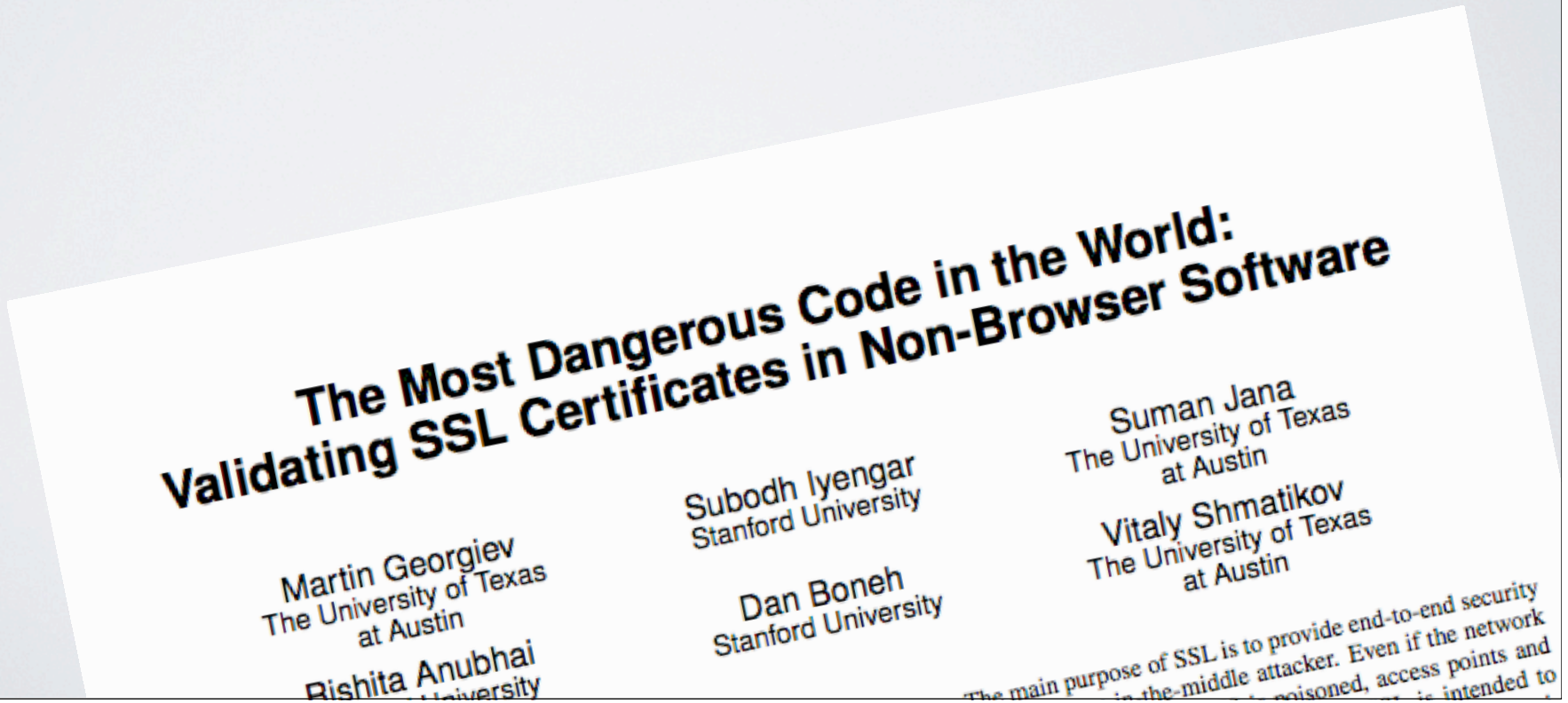
PKCS#1 recommendation

```
int RSA_padding_check_PKCS1_type_1(unsigned char *to, int tlen,
        const unsigned char *from, int flen, int num)
    {
    int i,j;
    const unsigned char *p;

    p=from;
    if ((num != (flen+1)) || (*(p++) != 01))
        {
        RSAerr(RSA_F_RSA_PADDING_CHECK_PKCS1_TYPE_1,RSA_R_BLOCK_TYPE_IS_NOT_01);
        return(-1);
        }

    /* scan over padding data */
    j=flen-1; /* one for type. */
    for (i=0; i<j; i++)
        {
        if (*p != 0xff) /* should decrypt to 0xff */
            {
            if (*p == 0)
                { p++; break; }
            else    {
                RSAerr(RSA_F_RSA_PADDING_CHECK_PKCS1_TYPE_1,RSA_R_BAD_FIXED_HEADER_DECRYPT);
                return(-1);
                }
            }
        p++;
        }

    if (i == j)
        {
        RSAerr(RSA_F_RSA_PADDING_CHECK_PKCS1_TYPE_1,RSA_R_NULL_BEFORE_BLOCK_MISSING);
        return(-1);
        }

    if (i < 8)
        {
        RSAerr(RSA_F_RSA_PADDING_CHECK_PKCS1_TYPE_1,RSA_R_BAD_PAD_BYTE_COUNT);
        return(-1);
```

OpenSSL v1.0.1c

APIs

just at least one guy who thinks "int enable" has only 2 values (not 3!) --- theGruqq



The Most Dangerous Code in the World: Validating SSL Certificates in Non-Browser Software

Martin Georgiev
The University of Texas at Austin

Subodh Iyengar
Stanford University

Suman Jana
The University of Texas at Austin

Dan Boneh
Stanford University

Vitaly Shmatikov
The University of Texas at Austin

Rishita Anubhai

The main purpose of SSL is to provide end-to-end security

We're all gonna die

- This is not at all what I'm saying
- There is a lot of good news in here:
  - We are learning how to analyze TLS
  - We may someday have a real proof of the protocol

We're all gonna die

- Moreover, the density of practical attacks on TLS is low
- We tend to fix them when they get announced
  - And most are active, so they won't be useful on old data
- That said, there's no excuse for neglecting it
  - This protocol & its implementations need to be a major focus of research if we're going to rely on them in the coming years