Cryptography is a systems problem

(or)

'Should we deploy TLS?'

Matthew Green
Johns Hopkins University

Why this presentation?

10 Things You Should Know
About Computer Security

5: Cryptography is a Solved Problem

Cryptography: The strongest link in the chain*

but not to others. Unfortunately, people concentrate too much on the cryptography of a system – which is the equivalent of strengthening the strongest link in a chain.

Why this presentation?

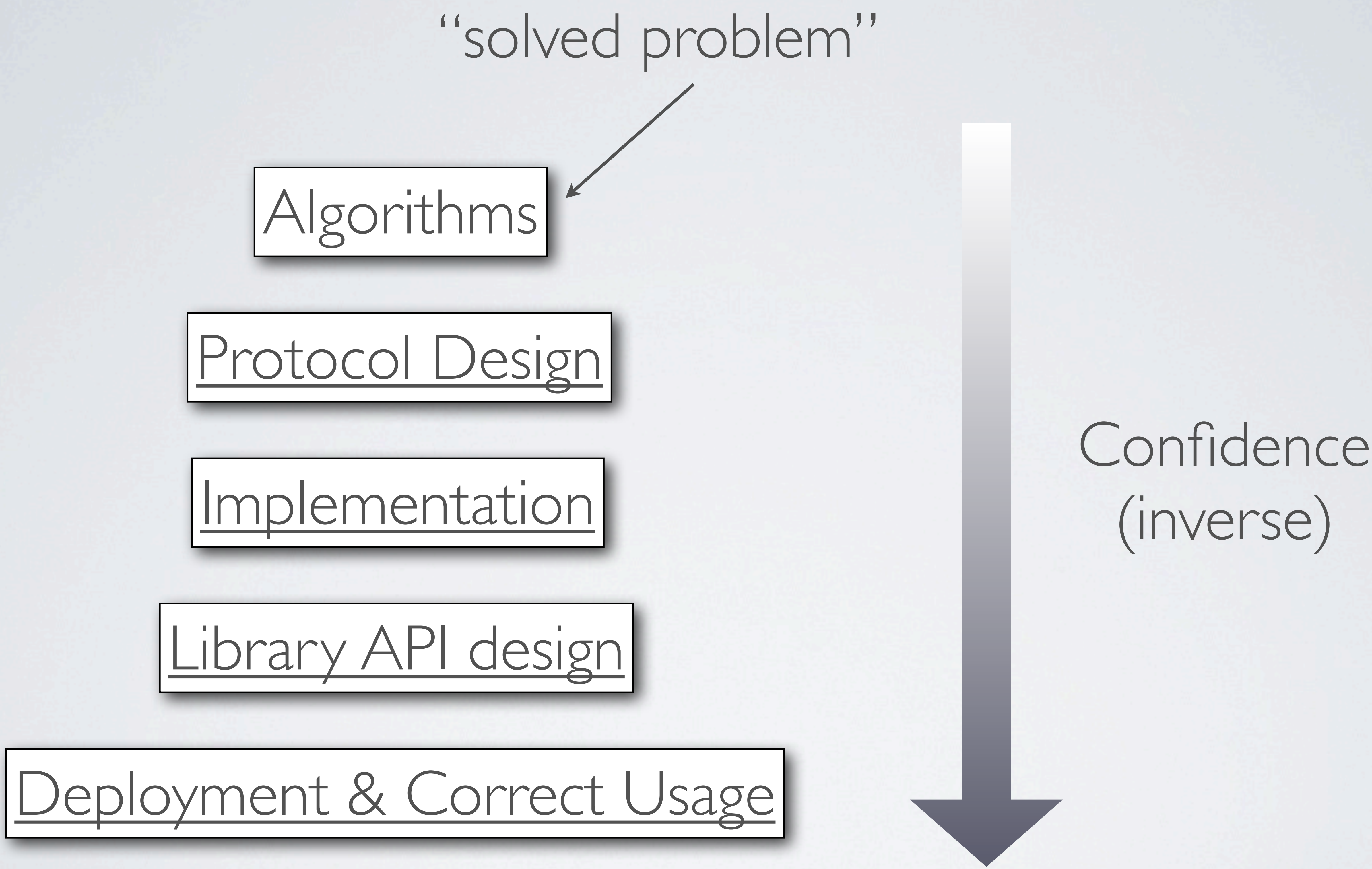
ings You Should Know
mputer Security

5: Cryptog Solved Problem

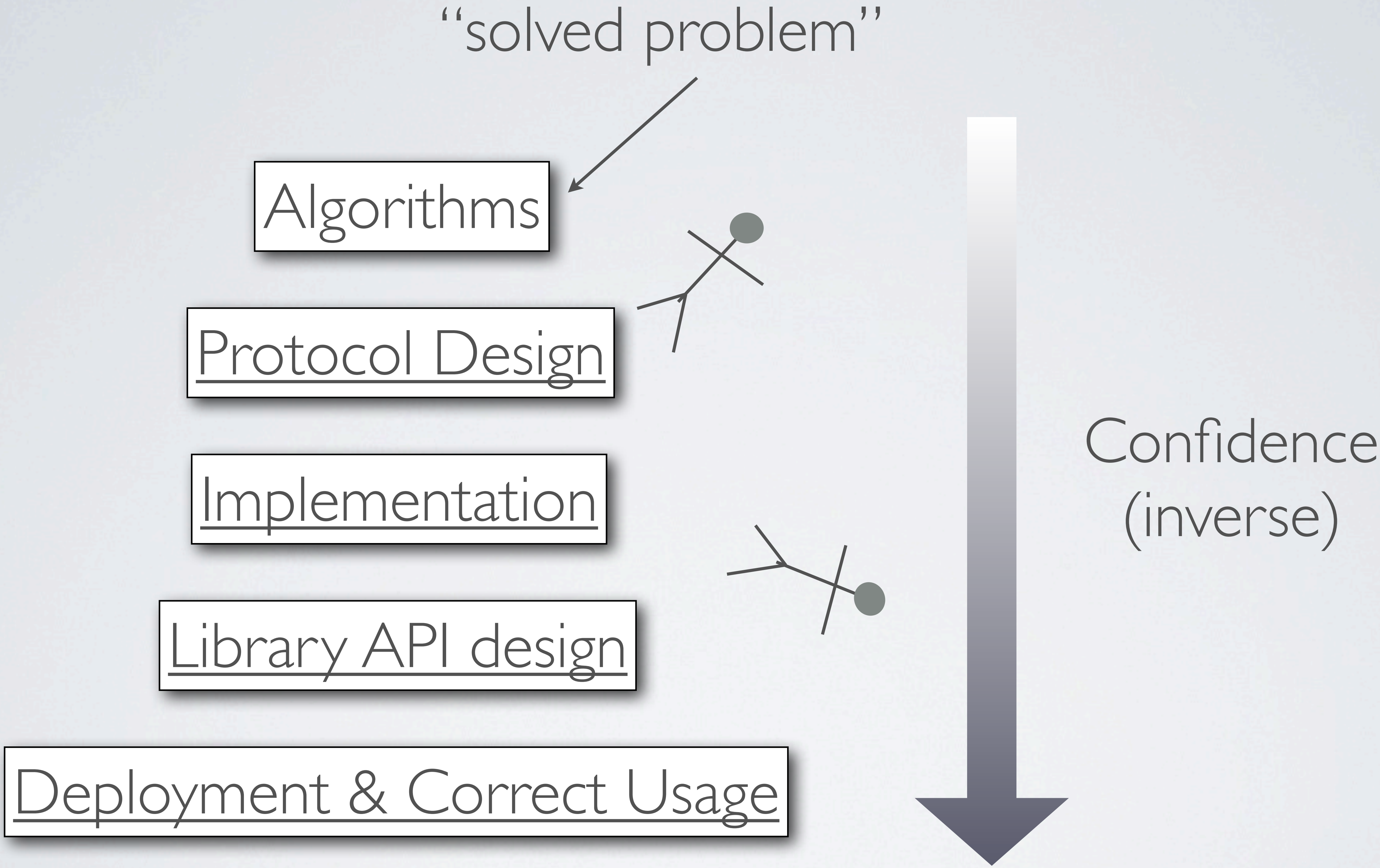
Cryptography: Th t link in the chain*

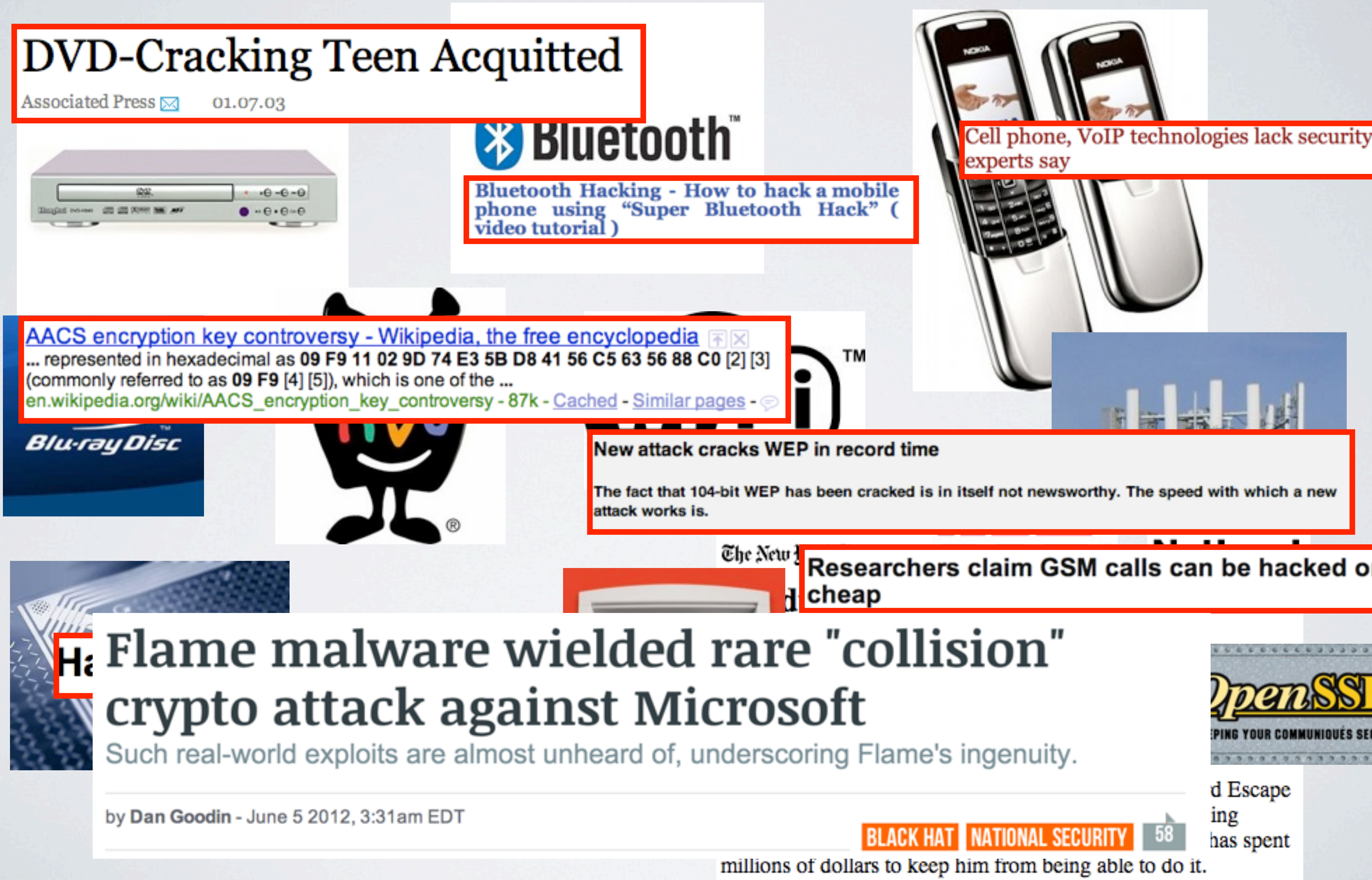
but not to others. Unfortunately, people concentrat on the
cryptography of a system – which is the equivalent of str
the strongest link in a chain.

These people are wrong



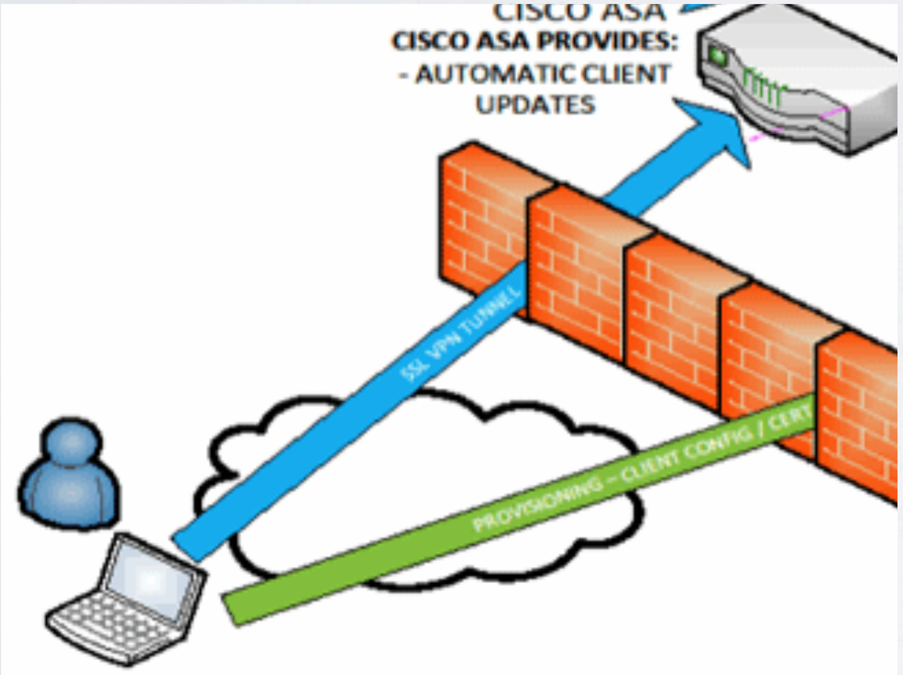
"solved problem"
Algorithms
Protocol Design
Implementation
Library API design
Deployment & Correct Usage
Confidence
(inverse)

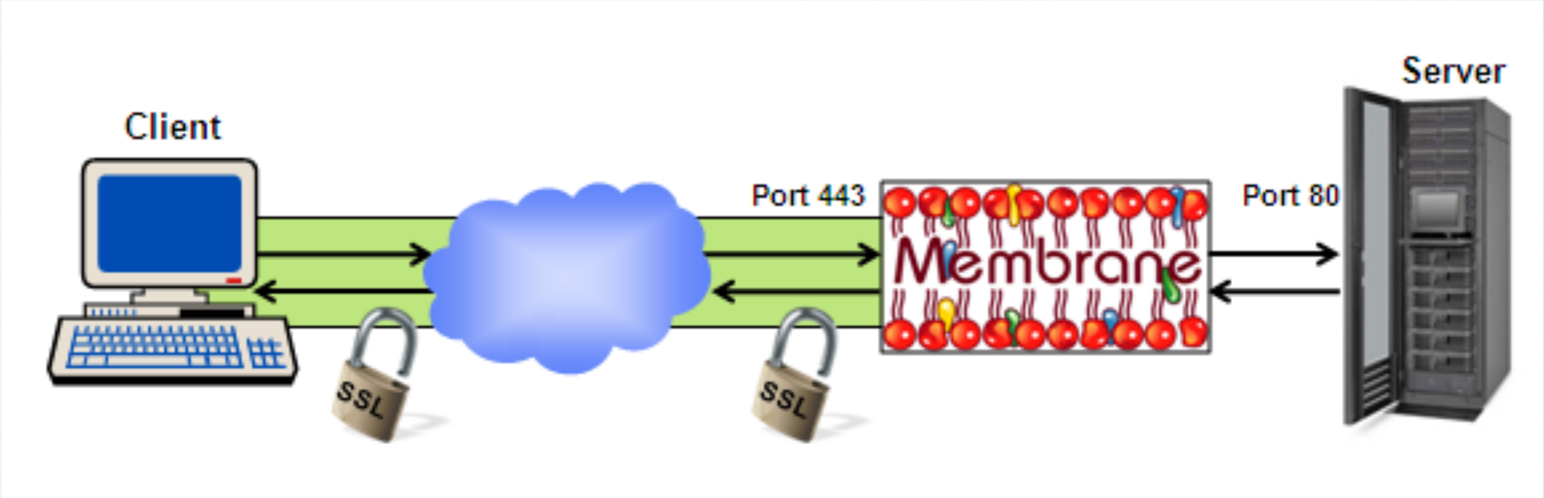

"solved problem"
Algorithms
Protocol Design
Implementation
Library API design
Deployment & Correct Usage
Confidence (inverse)


DVD-Cracking Teen Acquitted
Associated Press 01.07.03
Bluetooth™
Bluetooth Hacking - How to hack a mobile phone using "Super Bluetooth Hack" (video tutorial)
Cell phone, VoIP technologies lack security, experts say
AACS encryption key controversy - Wikipedia, the free encyclopedia
... represented in hexadecimal as 09 F9 11 02 9D 74 E3 5B D8 41 56 C5 63 56 88 C0 [2] [3] (commonly referred to as 09 F9 [4] [5]), which is one of the ...
en.wikipedia.org/wiki/AACS_encryption_key_controversy - 87k - Cached - Similar pages -
Blu-ray Disc
New attack cracks WEP in record time
The fact that 104-bit WEP has been cracked is in itself not newsworthy. The speed with which a new attack works is.
Researchers claim GSM calls can be hacked on cheap
Flame malware wielded rare "collision" crypto attack against Microsoft
Such real-world exploits are almost unheard of, underscoring Flame's ingenuity.
by Dan Goodin - June 5 2012, 3:31am EDT
BLACK HAT
NATIONAL SECURITY
58
d Escape
ing
has spent
millions of dollars to keep him from being able to do it.

Today's example: SSL/TLS

- Why SSL/TLS?
  - Because it's the most important security protocol in the world!





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 | +1 8 | Share 0 | Mail | Print

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

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

Google flips the switch on SSL encryption for Gmail

1

Filed in: Software , Wireless

By Colleen McColl, January 15, 2010 @ 9:47am

Tweet 0 | Submit | +1 0 | Like 3

Gmail by Google

-----BEGIN RSA PRIVATE KEY-----
MIICXQIBAAKBgQC+gu/eSRi5ThbY
w5kw+dSlOCU78jrGkbP4m0GyNN27
F5TsXqCJvqbwvTn+ExBxvBQyoYpdjl
AoGATkxUN2CFd8tfWmhKVHY/ioF

Google announced Wednesday that it will start encrypting all its Gmail traffic when using a remote wireless server.

This comes within days of Google's announcement that it might pull its offices from China after discovering concerted attempts to break into Gmail accounts of human rights activists.

Gmail users will now default to using HTTPS, the secure, encrypted method for communicating with a remote server. This is for the entire e-mail session, not just for log-in as was previously the case. Since 2008 this option

ENCRYPT THE WEB:
INSTALL HTTPS EVERYWHERE

Hacking in the Netherlands Took Aim at Internet Giants

By THE ASSOCIATED PRESS
Published: September 5, 2011

AMSTERDAM (AP) — Attackers who hacked into a Dutch Web security firm have issued hundreds of fraudulent security certificates for intelligence agency Web sites, including the C.I.A., as well as for Internet giants like Google, Microsoft and Twitter, the Dutch government said on Monday.

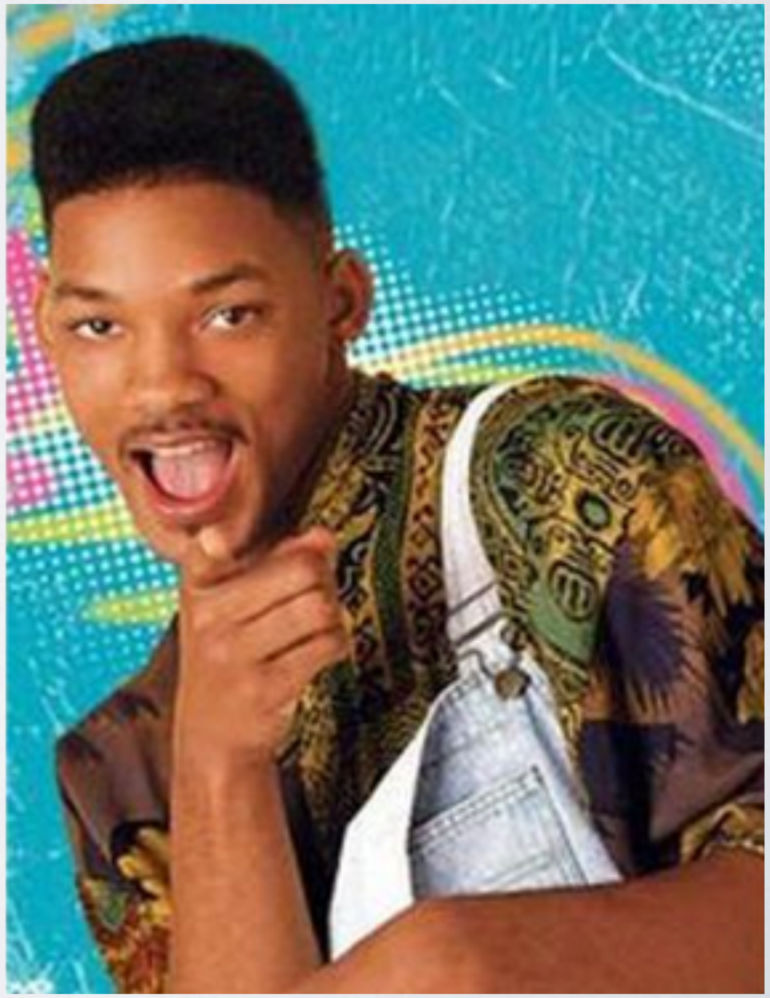
This presentation

What's the matter with TLS?

Designwise/
Analysis-wise/
Implementation-wise/
Usage-wise

A brief history

- SSLv1 born at Netscape. Never released. (~1995)
- SSLv2 released one year later
  - Serious protocol negotiation bugs.



A brief history

- SSLv1 born at Netscape (~1995)
- SSLv2 released one year later
  - Serious protocol negotiation bugs. Just awful.
- SSLv3
  - Slightly less serious issues [Schneier & Kelsey, others] padding oracles

> tocol. We conclude that, while there are still a few technical wrinkles to iron out, on the whole SSL 3.0 is a valuable contribution towards practical communications security.

A brief history

- TLSv1.0
  - Predictable IVs, renegotiation attacks and more!

Hackers break SSL encryption used by millions of sites

Beware of BEAST decrypting secret PayPal cookies

By **Dan Goodin in San Francisco** • **Get more from this author**

Posted in Security, 19th September 2011 21:10 GMT

Home > Security

News

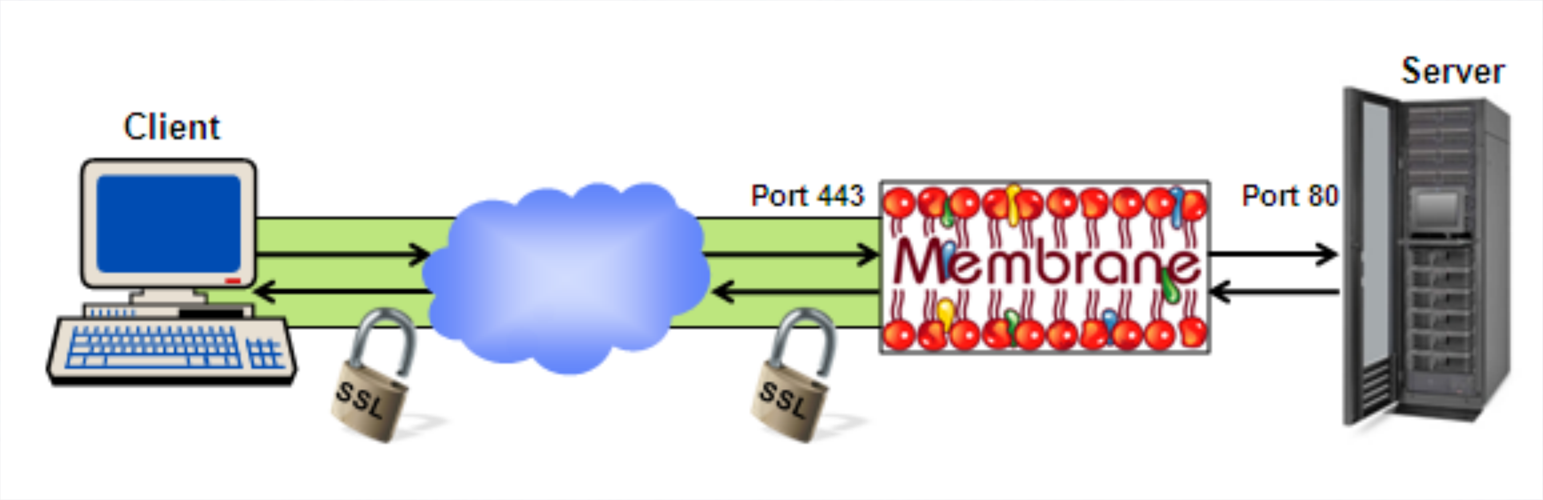
'CRIME' attack abuses SSL/TLS data compression feature to hijack HTTPS sessions

SSL/TLS data compression leaks information that can be used to decrypt HTTPS session cookies, researchers say

By Lucian Constantin

September 13, 2012 09:02 PM ET Add a comment

The SSL protocol



The SSL protocol

Protocol Design

It's bad

- Many problems result from TLS's use of "*pre-historic cryptography*" (- Eric Rescorla)
  - CBC with Mac-then-Encrypt, bad use of IVs
  - RSA-PKCS#1v1.5 encryption padding
  - RC4
  - Horrifying backwards compatibility requirements

MAC-then-pad-then-Encrypt

- TLS MACs the record, then pads (in CBC), then enciphers
  - Obvious problem: padding oracles (2002)

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

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

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 <u>remote</u> timing...

Unlucky for you: UK crypto-duo 'crack' HTTPS in Lucky 13 attack

OpenSSL patch to protect against TLS decryption boffinry

By **John Leyden** • **Get more from this author**

Posted in Security, 4th February 2013 16:58 GMT

BEAST

- Serious bug in TLS 1.0
- Allows complete (hollywood!) decryption of CBC ciphertexts
- Use of predictable Initialization Vector (CBC residue bug)
  - Known since 2002, attack described by Bard in 2005 (*Bard was advised to focus on more interesting problems*.)
  - Nobody cared or noticed until someone implemented it

Solution in practice: RC4

:-(

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

Compression (CRIME)

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

Analysis

TLS for cryptographers

Initiator I		**Responder R**
$r_I \xleftarrow{r} \{0,1\}^{p_1(k)}$	$\xrightarrow{\quad r_I \quad}$	
	$\xleftarrow{\quad r_R \quad}$	$r_R \xleftarrow{r} \{0,1\}^{p_2(k)}$

DHE	DHS	EKT
$\xleftarrow{g, g^x, \mathtt{SIG}_{sk_R}(r_I, r_R, g, g^x), R}$	$\xleftarrow{\quad R \quad}$	$\xleftarrow{\quad R \quad}$
$\xrightarrow{\quad g^y \quad}$	$\xrightarrow{\quad g^y \quad}$	$\xrightarrow{\mathtt{ENC}_{pk_R}(k_p)}$
$k_m \leftarrow \mathtt{PRF}_{g^{xy}}(l_1)$	$k_m \leftarrow \mathtt{PRF}_{g^{xy}}(l_1)$	$k_m \leftarrow \mathtt{PRF}_{k_p}(l_1)$

	$\xrightarrow{[\mathtt{SIG}_{sk_I}(\mathtt{trscrpt}), I]^*}$	parse $\mathtt{SIG}_{sk_I}()$ as σ IF $\mathtt{VER}_{vk_I}(\mathtt{trspt}, \sigma)$ accept ELSE abort
$(k_e^I, k_a^I, k_e^R, k_a^R) \leftarrow \mathtt{PRF}_{k_m}(l_2)$ $F_I \leftarrow \mathtt{PRF}_{k_m}(l_3)$		
	$\xrightarrow{\mathtt{E}_{k_e^I}(F_I \mid \mathtt{HMAC}_{k_a^I}(F_I))}$	parse $\mathtt{E}_{k_e^R}()$ as α $(k_e^I, k_a^I, k_e^R, k_a^R) \leftarrow \mathtt{PRF}_{k_m}(l_2)$ $(F_I \mid t_I) \leftarrow \mathtt{D}_{k_e^I}(\alpha)$ IF $F_I \leftarrow \mathtt{PRF}_{k_m}(l_3)$ AND $t_I \leftarrow \mathtt{HMAC}_{k_a^I}(F_I)$ accept $(k_e^I, k_a^I, k_e^R, k_a^R)$ ELSE abort $F_R \leftarrow \mathtt{PRF}_{k_m}(l_4)$
	$\mathtt{E}_{k_e^R}(F_R \mid \mathtt{HMAC}_{k_a^R}(F_R))$	

TLS for cryptographers

Initiator I | **Responder R**

$r_I \xleftarrow{r} \{0,1\}^{p_1(k)}$ $\xrightarrow{\quad r_I \quad}$

$\xleftarrow{\quad r_R \quad}$ $r_R \xleftarrow{r} \{0,1\}^{p_2(k)}$

DHE	DHS	EKT
$\xleftarrow{g, g^x, \mathtt{SIG}_{sk_R}(r_I, r_R, g, g^x), R}$	$\xleftarrow{\quad R \quad}$	$\xleftarrow{\quad R \quad}$
$\xrightarrow{\quad g^y \quad}$	$\xrightarrow{\quad g^y \quad}$	$\xrightarrow{\mathtt{ENC}_{pk_R}(k_p)}$
$k_m \leftarrow \mathtt{PRF}_{g^{xy}}(l_1)$	$k_m \leftarrow \mathtt{PRF}_{g^{xy}}(l_1)$	$k_m \leftarrow \mathtt{PRF}_{k_p}(l_1)$

$\xrightarrow{[\mathtt{SIG}_{sk_I}(\mathtt{trscrpt}), I]^*}$

parse $\mathtt{SIG}_{sk_I}()$ as σ
IF $\mathtt{VER}_{vk_I}(\mathtt{trspt}, \sigma)$ accept
ELSE abort

$(k_e^I, k_a^I, k_e^R, k_a^R) \leftarrow \mathtt{PRF}_{k_m}(l_2)$
$F_I \leftarrow \mathtt{PRF}_{k_m}(l_3)$

$\xrightarrow{\mathtt{E}_{k_e^I}(F_I | \mathtt{HMAC}_{k_a^I}(F_I))}$

parse $\mathtt{E}_{k_e^R}()$ as α
$(k_e^I, k_a^I, k_e^R, k_a^R) \leftarrow \mathtt{PRF}_{k_m}(l_2)$
$(F_I | t_I) \leftarrow \mathtt{D}_{k_e^I}(\alpha)$
IF $F_I \leftarrow \mathtt{PRF}_{k_m}(l_3)$
AND $t_I \leftarrow \mathtt{HMAC}_{k_a^I}(F_I)$
accept $(k_e^I, k_a^I, k_e^R, k_a^R)$
ELSE abort
$F_R \leftarrow \mathtt{PRF}_{k_m}(l_4)$

$\mathtt{E}_{k_e^R}(F_R | \mathtt{HMAC}_{k_a^R}(F_R))$

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

Surely we've analyzed TLS?

- Well -- not really.
  - In CRYPTO 2012 (!) we saw the first paper to successfully analyze TLS-DHE [Jager *et al.*]
  - To date: no published work that analyzes even the TLS-RSA handshake (in a realistic setting)
- Nobody has ever proven the /full/ TLS protocol to be secure

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
```

Code to the spec!

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
      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                (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
      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);
```

Thread locks

PKCS #1v1.5

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

(They have two.)

```
RF
                                          (1)
              M                           (2)
              length of message
else:
    pre_m
```

```
...ypt((int)n,p,p,rsa,RSA_PKCS1_PADDING);        (2)
...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);
```

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."

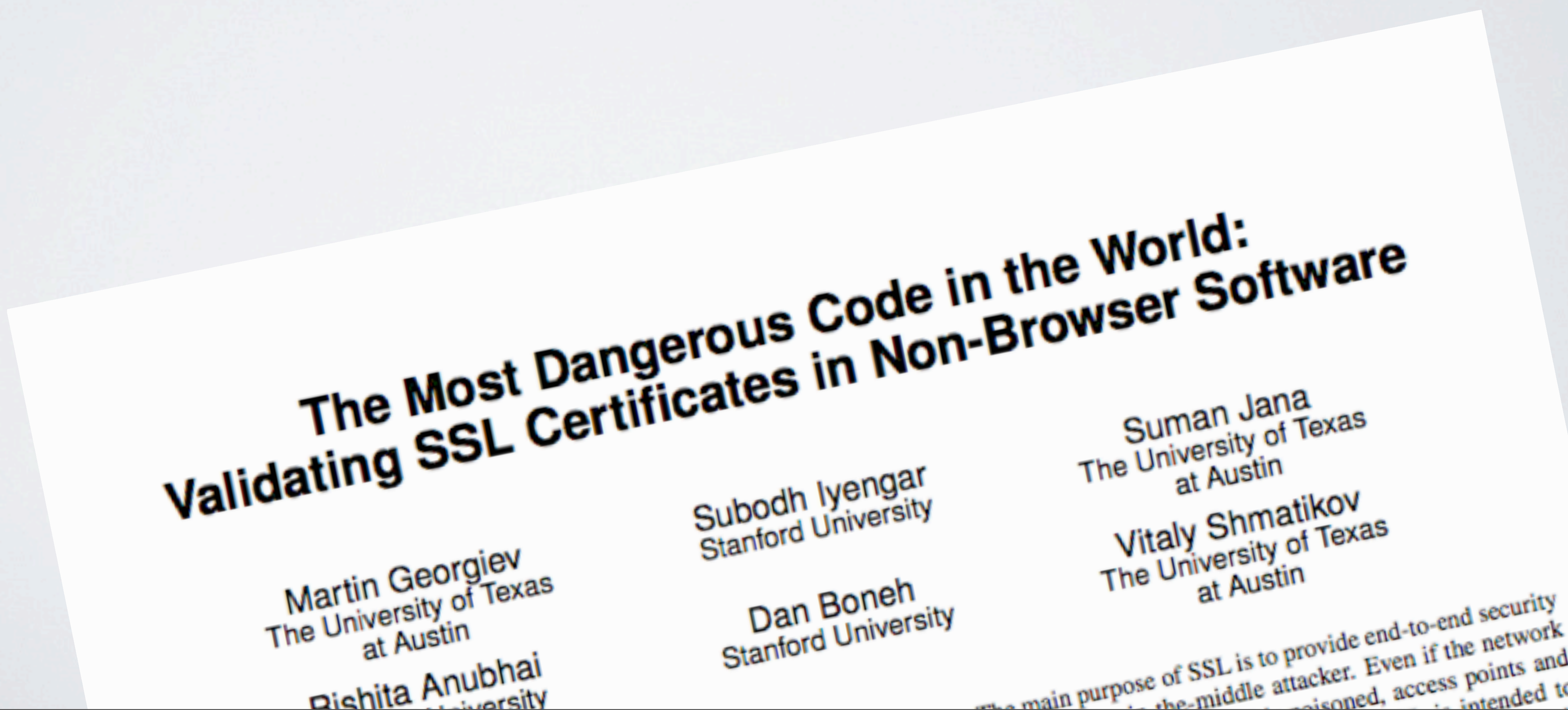
PKCS#1 recommendation

```
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

Rishita Anubhai

Subodh Iyengar
Stanford University

Dan Boneh
Stanford University

Suman Jana
The University of Texas
at Austin

Vitaly Shmatikov
The University of Texas
at Austin

main purpose of SSL is to provide end-to-end security
the-middle attacker. Even if the network
poisoned, access points and
intended to

APIs

The worst example from this paper is Curl's API.

Curl has an option, CURL_SSL_VERIFYHOST. When VERIFYHOST=0, Curl does what you'd expect: it effectively doesn't validate SSL certificates.

When VERIFYHOST=2, Curl does what you'd expect: it verifies SSL certificates, ensuring that one of the hosts attested by the certificate matches the host presenting it.

When VERIFYHOST=1, or, in some popular languages, when VERIFYHOST=TRUE, Curl does something very strange. It checks to see if the certificate attests to any hostnames, and then accepts the certificate *no matter who presents it*.

Developers reasonably assume parameters like "VERIFYHOST" are boolean; either we're verifying or we're not. So they routinely set VERIFYHOST to 1 or "true" (which can promote to 1). Because Curl has this weird in-between setting, which does not express any security policy I can figure out, they're effectively not verifying certificates.

Space shuttles vs. Elevators

```
int do_evp_seal(FILE *rsa_pkey_file, FILE *in_file, FILE *out_file)
{
    int retval = 0;
    RSA *rsa_pkey = NULL;
    EVP_PKEY *pkey = EVP_PKEY_new();
    EVP_CIPHER_CTX ctx;
    unsigned char buffer[4096];
    unsigned char buffer_out[4096 + EVP_MAX_IV_LENGTH];
    size_t len;
    int len_out;
    unsigned char *ek = NULL;
    int eklen;
    uint32_t eklen_n;
    unsigned char iv[EVP_MAX_IV_LENGTH];

    if (!PEM_read_RSA_PUBKEY(rsa_pkey_file, &rsa_pkey, NULL, NULL))
    {
        fprintf(stderr, "Error loading RSA Public Key File.\n");
        ERR_print_errors_fp(stderr);
        retval = 2;
        goto out;
    }

    if (!EVP_PKEY_assign_RSA(pkey, rsa_pkey))
    {
        fprintf(stderr, "EVP_PKEY_assign_RSA: failed.\n");
        retval = 3;
        goto out;
    }

    EVP_CIPHER_CTX_init(&ctx);
    ek = malloc(EVP_PKEY_size(pkey));

    if (!EVP_SealInit(&ctx, EVP_aes_128_cbc(), &ek, &eklen, iv, &pkey, 1))
    {
        fprintf(stderr, "EVP_SealInit: failed.\n");
        retval = 3;
```

Elevators vs. space shuttles

```
#include "crypto_box.h"

std::string pk;
std::string sk;
std::string n;
std::string m;
std::string c;

c = crypto_box(m,n,pk,sk);
```

Elevators vs. space shuttles

```
#include "crypto_box.h"

std::string pk;
std::string sk;
std::string n;
std::string m;
std::string c;

c = crypto_box(m,n,pk,sk);
```

Usage

HTTP->HTTPS

- Typical Banking Experience:
  - SSL URLs begin with https://
  - But users rarely type the prefix

User enters: americanexpress.com

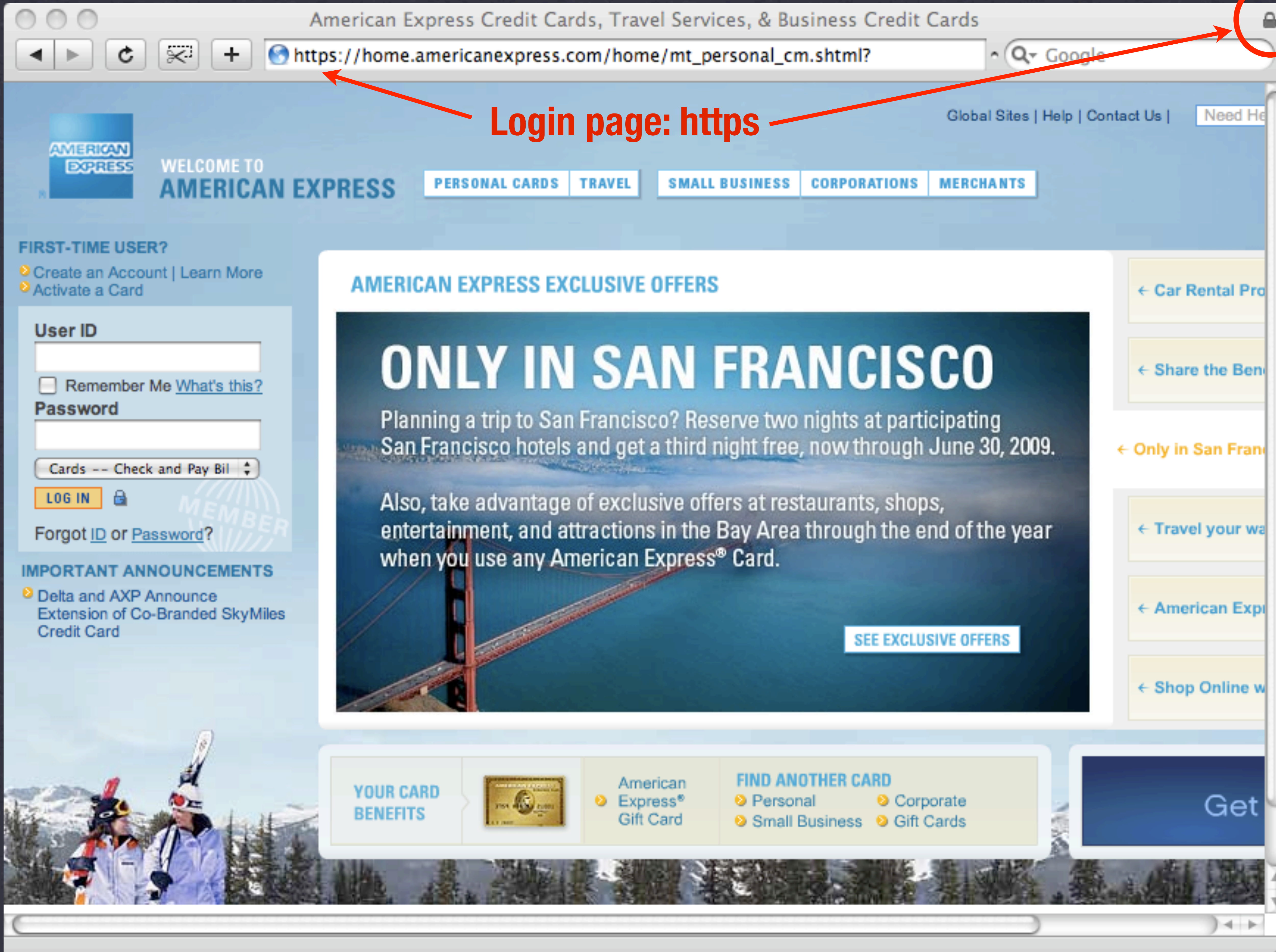
GET http://americanexpress.com

REDIRECT https://americanexpress.com

GET https://americanexpress.com

SSL secured web page

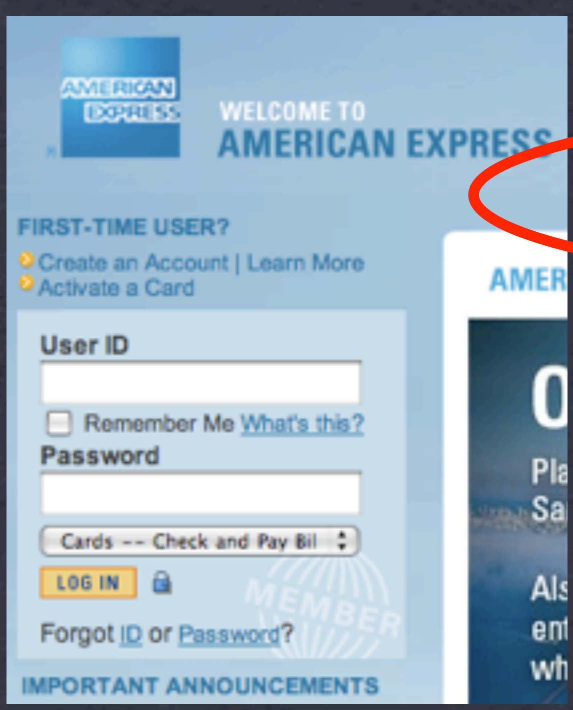
User login info


American Express Credit Cards, Travel Services, & Business Credit Cards
https://home.americanexpress.com/home/mt_personal_cm.shtml?
Google
Login page: https
Global Sites | Help | Contact Us |
AMERICAN EXPRESS
WELCOME TO
AMERICAN EXPRESS
PERSONAL CARDS
TRAVEL
SMALL BUSINESS
CORPORATIONS
MERCHANTS
FIRST-TIME USER?
Create an Account | Learn More
Activate a Card
User ID
Remember Me What's this?
Password
Cards -- Check and Pay Bil
LOG IN
Forgot ID or Password?
IMPORTANT ANNOUNCEMENTS
Delta and AXP Announce Extension of Co-Branded SkyMiles Credit Card
AMERICAN EXPRESS EXCLUSIVE OFFERS
ONLY IN SAN FRANCISCO
Planning a trip to San Francisco? Reserve two nights at participating San Francisco hotels and get a third night free, now through June 30, 2009.
Also, take advantage of exclusive offers at restaurants, shops, entertainment, and attractions in the Bay Area through the end of the year when you use any American Express® Card.
SEE EXCLUSIVE OFFERS
Only in San Fran
YOUR CARD BENEFITS
American Express® Gift Card
FIND ANOTHER CARD
Personal
Corporate
Small Business
Gift Cards
Get

HTTP->HTTPS

- If you can intercept the user's connection:
  - Don't redirect, or:
  - Redirect to malicious site, unsecured (http)

User enters: americanexpress.com



GET http://amex.com

REDIRECT http://secure.amex.com

GET https://secure.amex.com

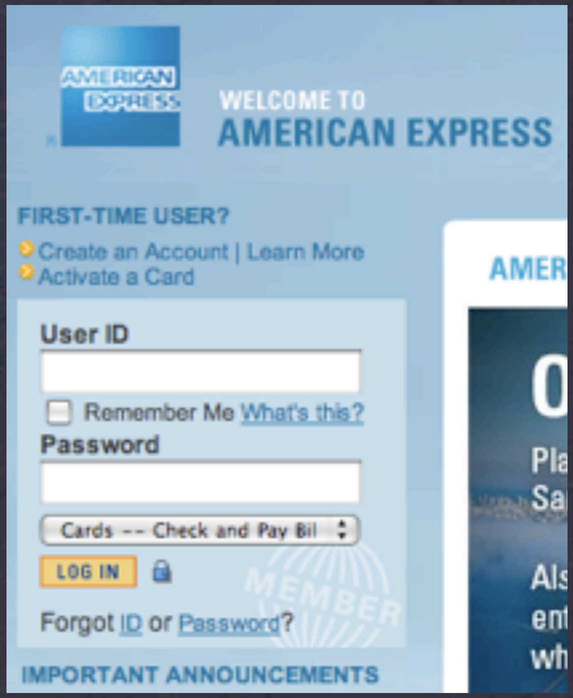
SSL secured web page

User login info

HTTP->HTTPS

- If you can intercept the user's connection:
  - Homograph site: paypaI.com (with a capital i), or:
  - Use clever IDN tricks e.g.,

    https://www.gmail.com/accounts/ServiceLogin!f.ijjk.cn



HTTP->HTTP->HTTPS

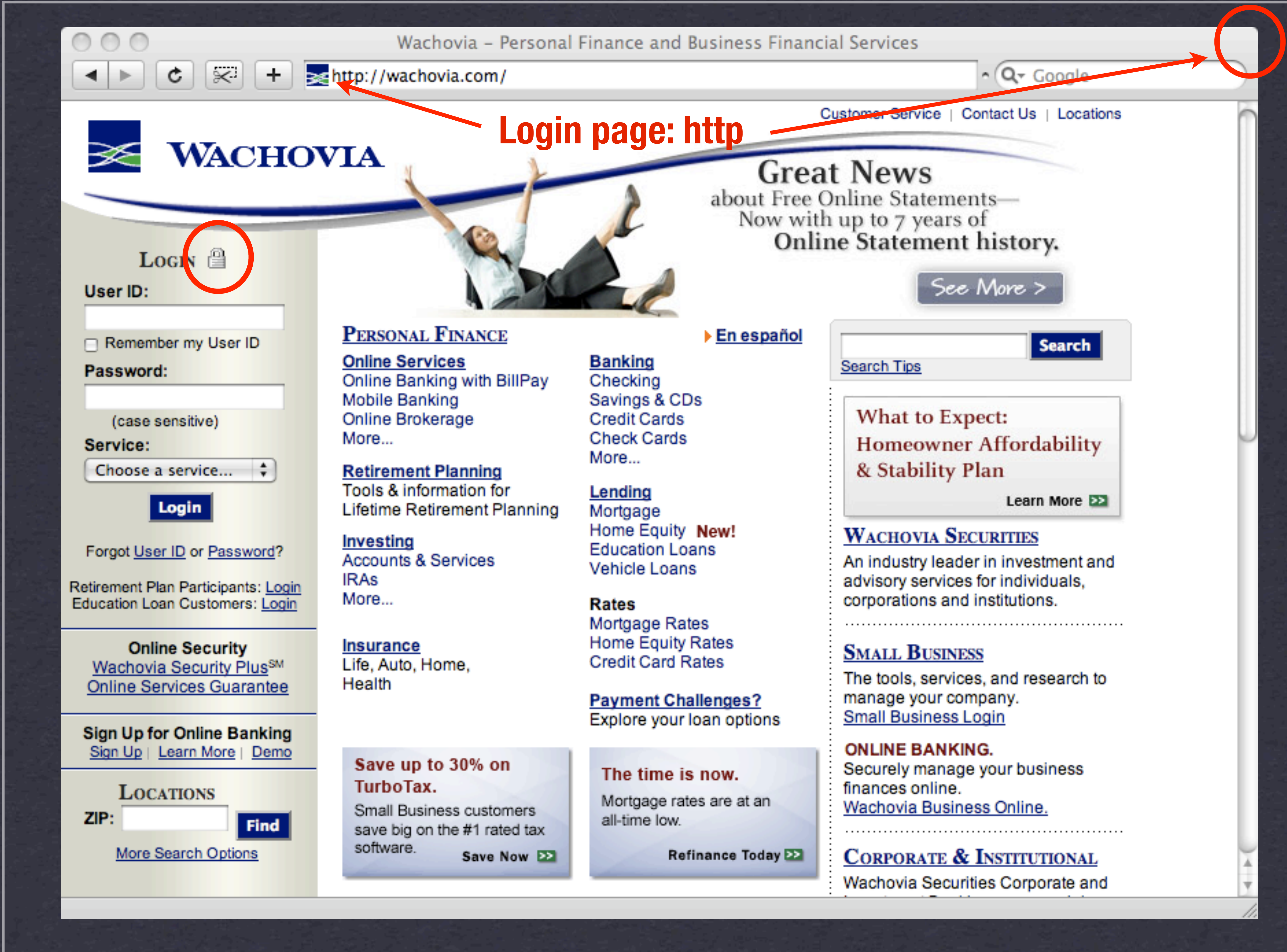
- It can be worse:
  - Some sites give an http page with a form that submits via https

User enters: americanexpress.com

GET http://americanexpress.com

Unsecured http web page

User login info


Wachovia – Personal Finance and Business Financial Services
http://wachovia.com/
Google
Login page: http
Customer Service | Contact Us | Locations
WACHOVIA
Great News
about Free Online Statements—
Now with up to 7 years of
Online Statement history.
See More >
Login
User ID:
Remember my User ID
Password:
(case sensitive)
Service:
Choose a service...
Login
Forgot User ID or Password?
Retirement Plan Participants: Login
Education Loan Customers: Login
Online Security
Wachovia Security Plus SM
Online Services Guarantee
Sign Up for Online Banking
Sign Up | Learn More | Demo
Locations
ZIP:
Find
More Search Options
Personal Finance
En español
Online Services
Online Banking with BillPay
Mobile Banking
Online Brokerage
More...
Retirement Planning
Tools & information for
Lifetime Retirement Planning
Investing
Accounts & Services
IRAs
More...
Insurance
Life, Auto, Home,
Health
Banking
Checking
Savings & CDs
Credit Cards
Check Cards
More...
Lending
Mortgage
Home Equity New!
Education Loans
Vehicle Loans
Rates
Mortgage Rates
Home Equity Rates
Credit Card Rates
Payment Challenges?
Explore your loan options
Save up to 30% on TurboTax.
Small Business customers save big on the #1 rated tax software.
Save Now
The time is now.
Mortgage rates are at an all-time low.
Refinance Today
Search
Search Tips
What to Expect:
Homeowner Affordability & Stability Plan
Learn More
Wachovia Securities
An industry leader in investment and advisory services for individuals, corporations and institutions.
Small Business
The tools, services, and research to manage your company.
Small Business Login
ONLINE BANKING.
Securely manage your business finances online.
Wachovia Business Online.
Corporate & Institutional
Wachovia Securities Corporate and

We're all gonna die

- This is not what I'm saying
- There is a lot of good news in here:
  - We are learning how to analyze TLS
  - We are learning how to implement & use TLS
- But there is still so much left to be done.

blog.cryptographyengineering.com