What's wrong with Cryptographic API design

And what we can do to fix it

Matthew Green
Johns Hopkins University

This session

Is there a problem?

What can we do about it?

What are the <u>research</u> problems here?

History

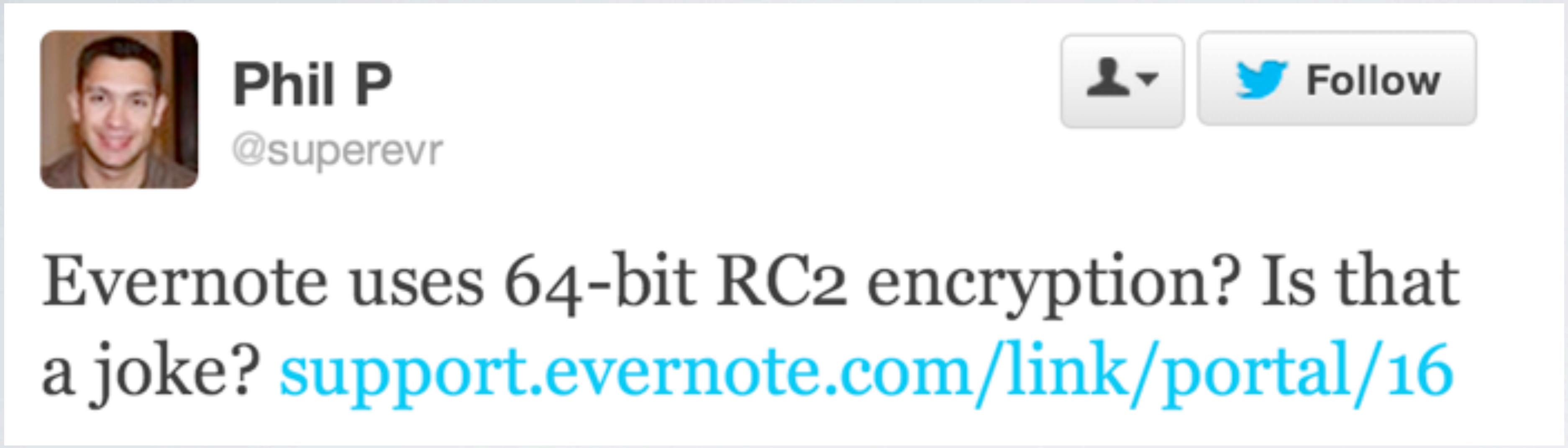
- In 1980s (1990s, 2000s) cryptography was a specialized area of security
  - Relatively few cryptographic tools
  - Written by 'cryptographic engineers', hobbyists, a few academic cryptographers
  - Focus was on developing security tools, protocols
  - ***Interfaces were poor***

Present Day

- For better or for worse: cryptography is becoming ubiquitous
  - Thousands of *non-security* applications use crypto
  - Written by non-expert software developers
  - This problem is set to worsen thanks to web cryptography and (soon) accessible Web Crypto APIs
  - ***Interfaces are still poor. It just matters more now.***

October 26th, 2012, 08:49 GMT · By Eduard Kovacs

Faulty SSL Certificate Validation Exposes Apps to MitM Attacks, Researchers Find


Phil P
@superevr

Evernote uses 64-bit RC2 encryption? Is that a joke? support.evernote.com/link/portal/16

How to check if your application is vulnerable to the ASP.NET Padding Oracle Vulnerability

LinkedIn password breach: How to tell if you're affected

stackoverflow

Is it insecure to pass initialization vector and salt along with ciphertext?

Not-so-surprising thesis

- **This is all our fault**
- The cryptography & security communities have largely abandoned practice
  - This is not an education problem!
  - We need tools and techniques -- and particularly APIs -- that will help the community in the future

Why cryptographic APIs?

- **For most developers this is the primary interface to cryptography**
- Properly designing these libraries and APIs can make a significant difference now
  - Facilitate proper usage
  - Permit auditing of code / machine auditing of code
  - Remove critical vulnerabilities before they happen

Two kinds of API

- **Developer APIs**: aid cryptographers in correct implementation
  - Ex: OpenSSL EVP, NaCl, MS Crypto API, W3C Web Crypto
- **Secure APIs**: enforce privilege separation, proper usage
  - User may be adversarial
  - Example: PKCS#11 Security Tokens

What's wrong with today's APIs?

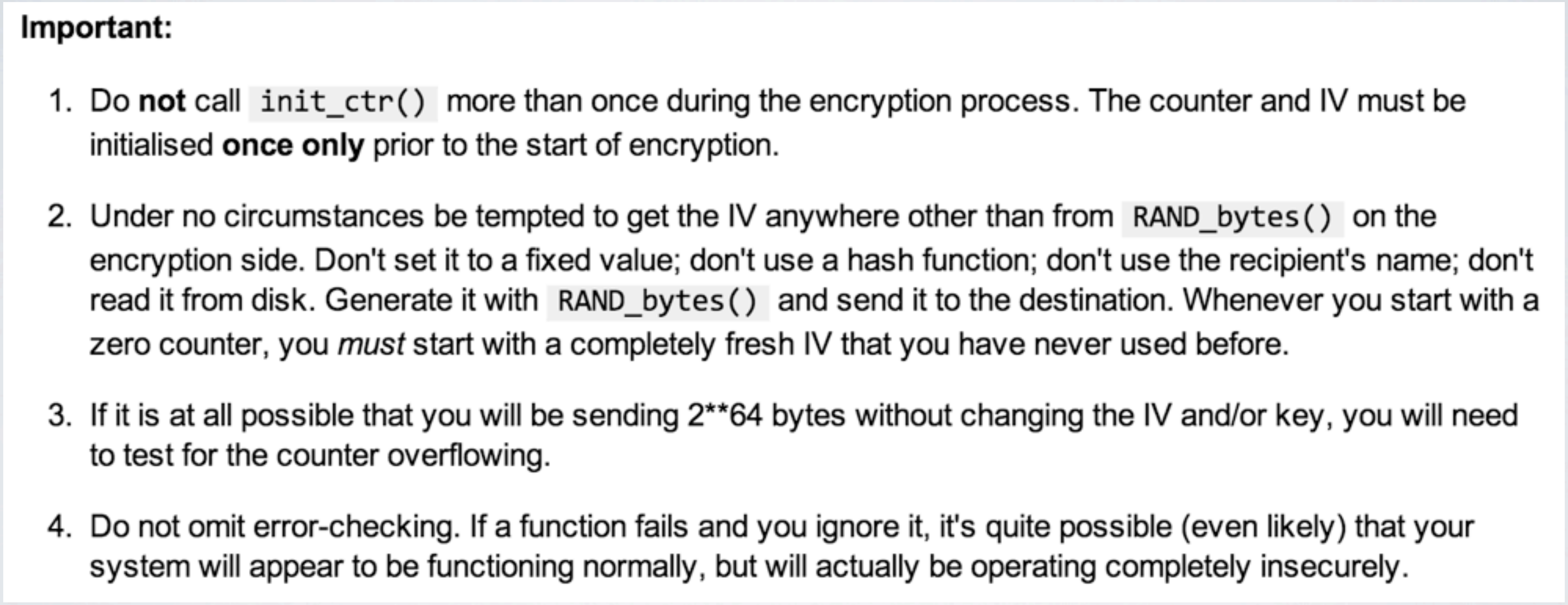
Problems

- Unnecessary complexity
- Algorithm choices & too many unsafe options
- Ambiguous specification
- Non-intuitive options
- Improper error codes
- Programming Language Interaction
- Key management?

Too much complexity 1/3

```c
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;
        goto out_free;
```

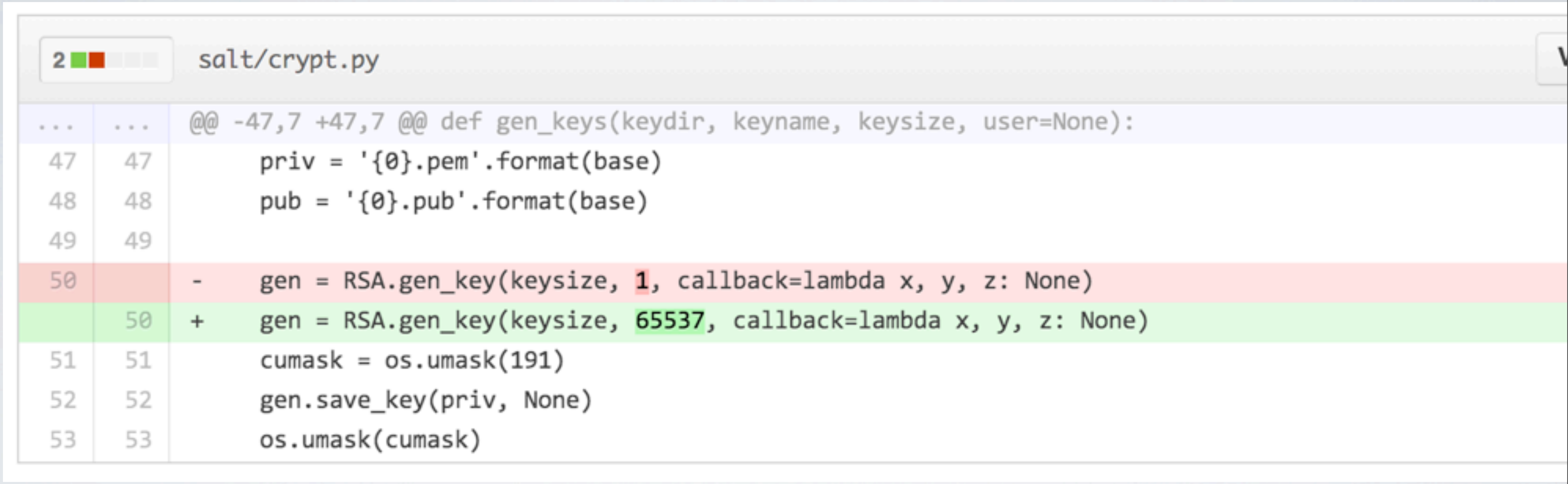
Too much complexity 2/3

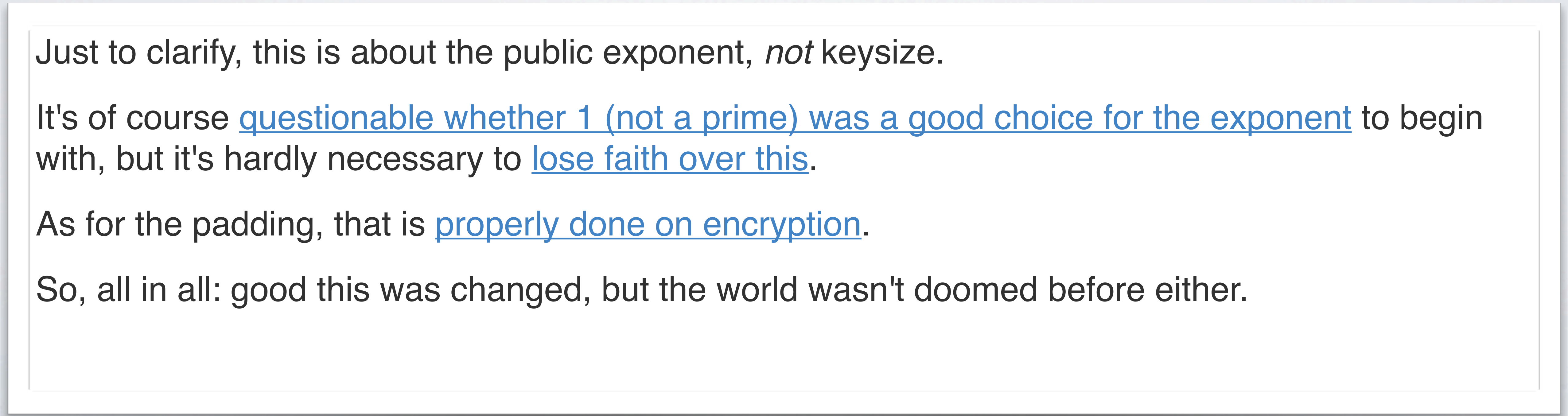


Important:

1. Do **not** call `init_ctr()` more than once during the encryption process. The counter and IV must be initialised **once only** prior to the start of encryption.
2. Under no circumstances be tempted to get the IV anywhere other than from `RAND_bytes()` on the encryption side. Don't set it to a fixed value; don't use a hash function; don't use the recipient's name; don't read it from disk. Generate it with `RAND_bytes()` and send it to the destination. Whenever you start with a zero counter, you *must* start with a completely fresh IV that you have never used before.
3. If it is at all possible that you will be sending 2**64 bytes without changing the IV and/or key, you will need to test for the counter overflowing.
4. Do not omit error-checking. If a function fails and you ignore it, it's quite possible (even likely) that your system will appear to be functioning normally, but will actually be operating completely insecurely.

Too much complexity 3/3


```
2 salt/crypt.py
@@ -47,7 +47,7 @@ def gen_keys(keydir, keyname, keysize, user=None):
       priv = '{0}.pem'.format(base)
       pub = '{0}.pub'.format(base)

-      gen = RSA.gen_key(keysize, 1, callback=lambda x, y, z: None)
+      gen = RSA.gen_key(keysize, 65537, callback=lambda x, y, z: None)
       cumask = os.umask(191)
       gen.save_key(priv, None)
       os.umask(cumask)
```


Just to clarify, this is about the public exponent, *not* keysize.

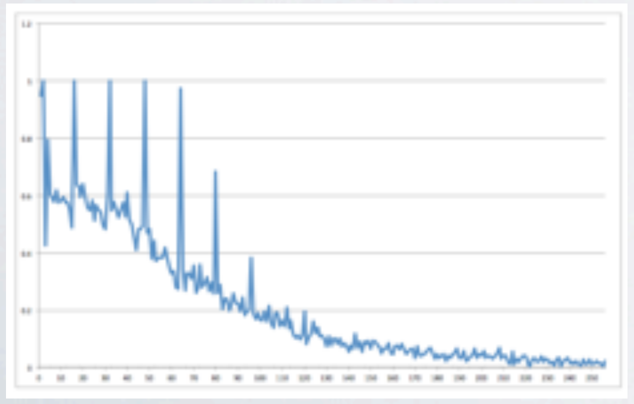
It's of course questionable whether 1 (not a prime) was a good choice for the exponent to begin with, but it's hardly necessary to lose faith over this.

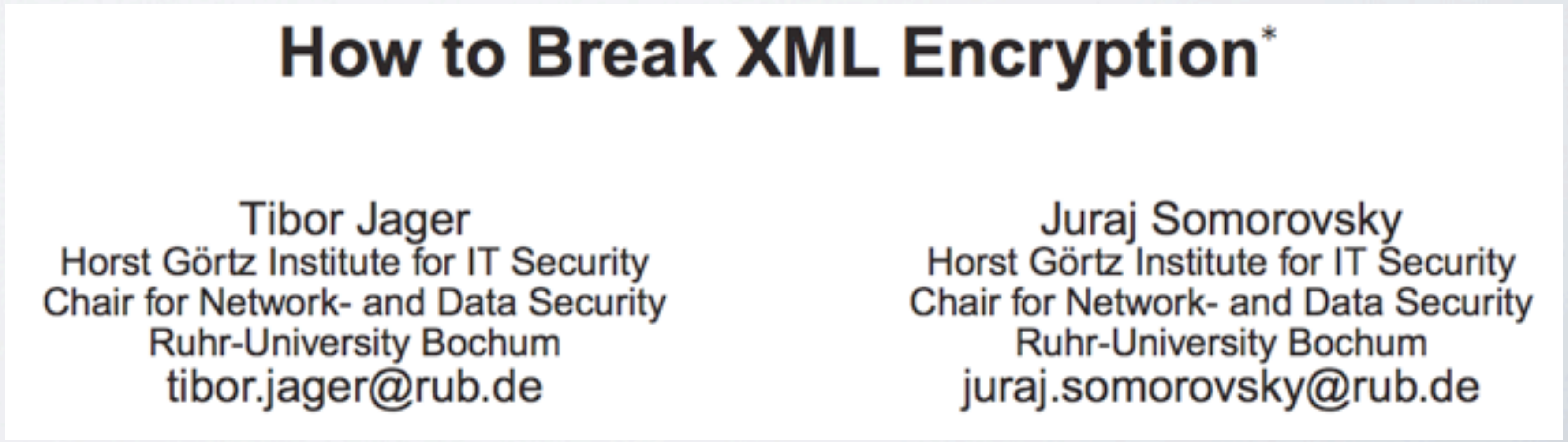
As for the padding, that is properly done on encryption.

So, all in all: good this was changed, but the world wasn't doomed before either.

Algorithm Choices 1/2

- Far too much developer responsibility for choosing and securely composing algorithms
  - Support for unauthenticated encryption (CBC/CTR)
  - RC4!
  - Generic composition of ciphers & MACs
  - Emphasis on legacy applications





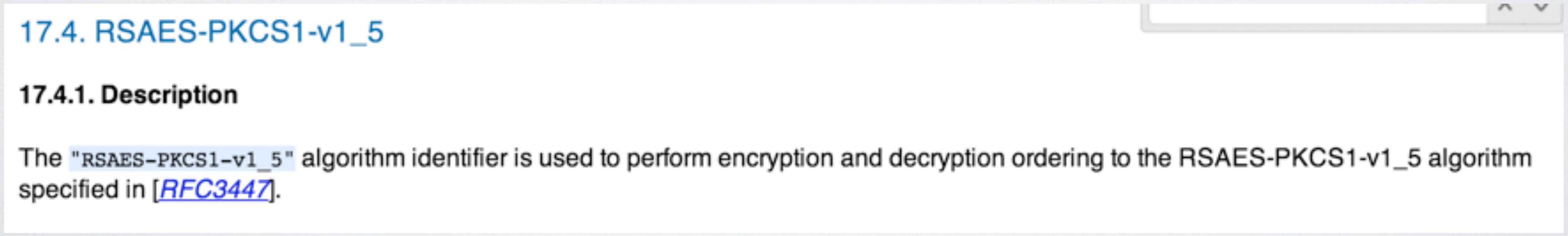
How to Break XML Encryption*

Tibor Jager
Horst Görtz Institute for IT Security
Chair for Network- and Data Security
Ruhr-University Bochum
tibor.jager@rub.de

Juraj Somorovsky
Horst Görtz Institute for IT Security
Chair for Network- and Data Security
Ruhr-University Bochum
juraj.somorovsky@rub.de

Algorithm Choices 2/2

- RSA with PKCS #1v1.5 encryption
  - Provided as the only mandatory padding scheme in many software devices (e.g., PKCS11 tokens)
  - **It is conceivably possible to encrypt some types of data securely with PKCS#1v1.5 padding**
  - **Almost nobody knows how to do it (even OpenSSL has active timing vulns.)**

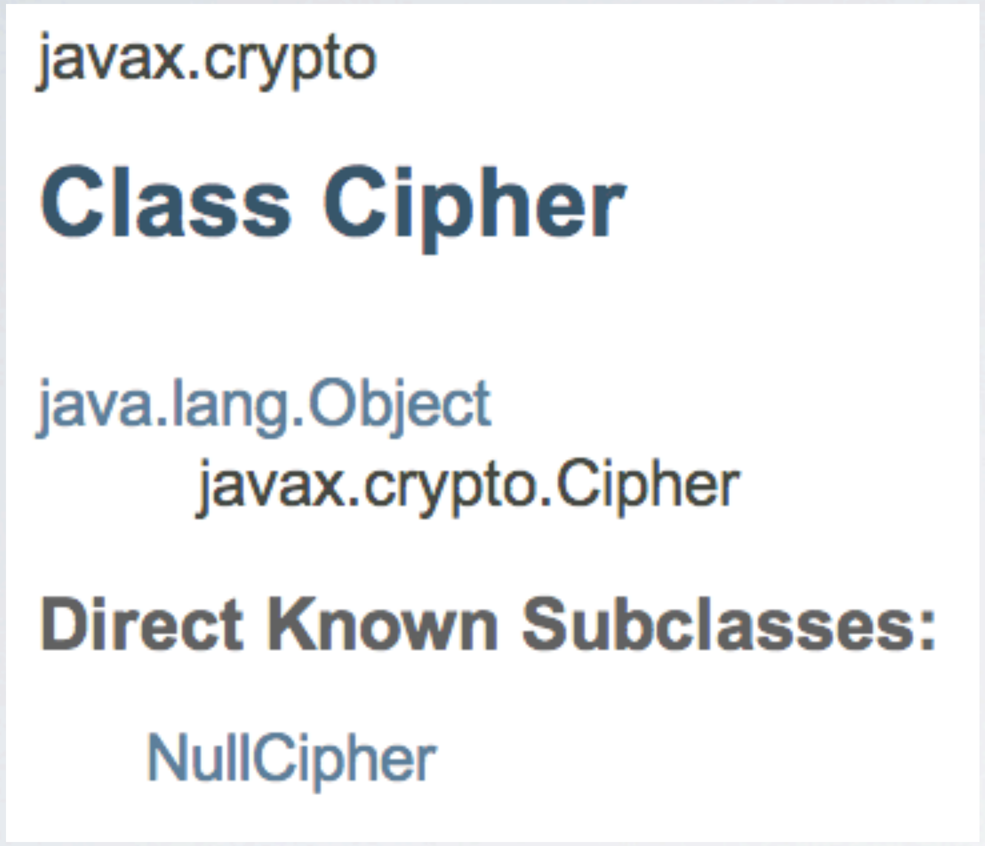


17.4. RSAES-PKCS1-v1_5

17.4.1. Description

The "RSAES-PKCS1-v1_5" algorithm identifier is used to perform encryption and decryption ordering to the RSAES-PKCS1-v1_5 algorithm specified in [RFC3447].

Ambiguous specification 1/2

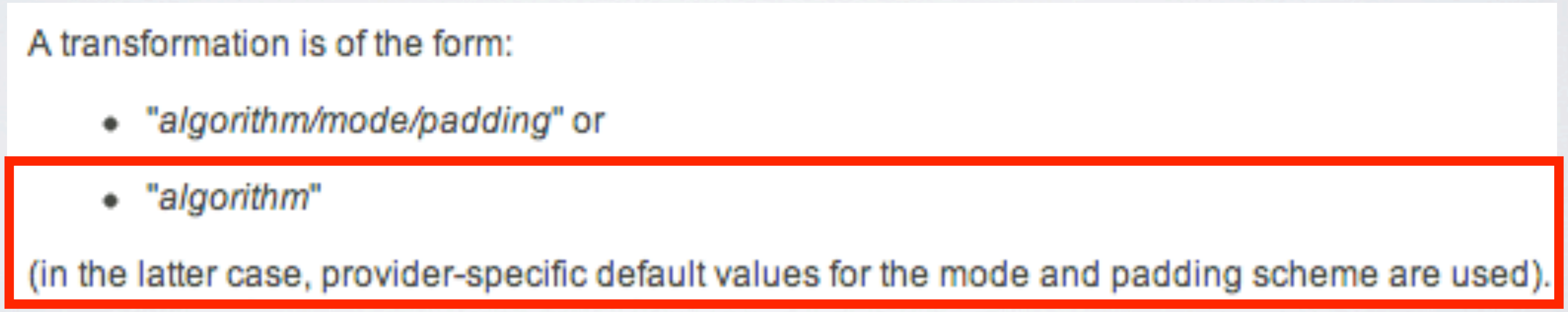

javax.crypto

Class Cipher

java.lang.Object
    javax.crypto.Cipher

Direct Known Subclasses:

NullCipher

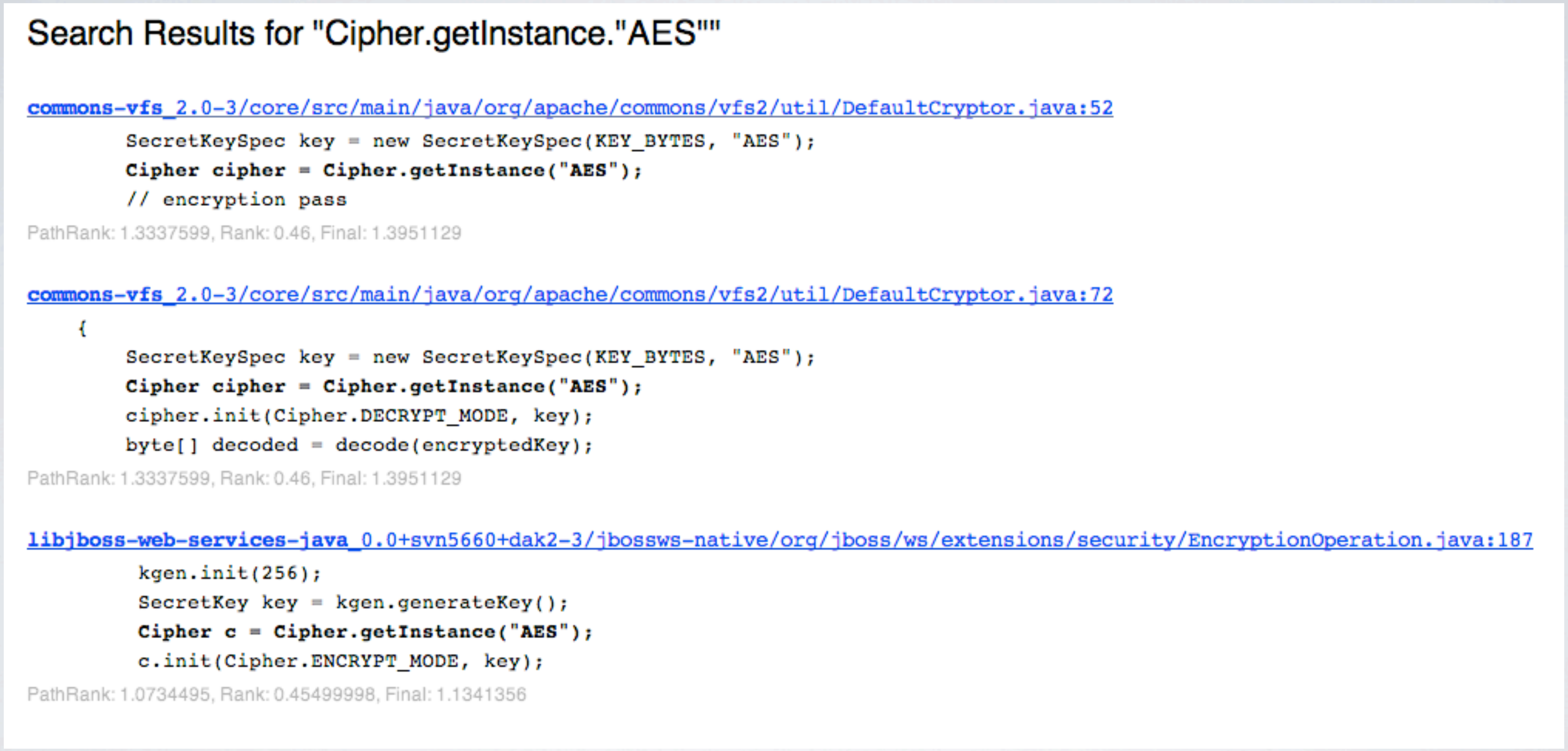

A transformation is of the form:

- "*algorithm/mode/padding*" or
- "*algorithm*"

(in the latter case, provider-specific default values for the mode and padding scheme are used).

Source: Sun Java SE JDK 7, h/t Nick Mathewson, Nikolay Elenkov

Ambiguous specification 2/2



Search Results for "Cipher.getInstance."AES""

commons-vfs_2.0-3/core/src/main/java/org/apache/commons/vfs2/util/DefaultCryptor.java:52

```
        SecretKeySpec key = new SecretKeySpec(KEY_BYTES, "AES");
        Cipher cipher = Cipher.getInstance("AES");
        // encryption pass
```

PathRank: 1.3337599, Rank: 0.46, Final: 1.3951129

commons-vfs_2.0-3/core/src/main/java/org/apache/commons/vfs2/util/DefaultCryptor.java:72

```
    {
        SecretKeySpec key = new SecretKeySpec(KEY_BYTES, "AES");
        Cipher cipher = Cipher.getInstance("AES");
        cipher.init(Cipher.DECRYPT_MODE, key);
        byte[] decoded = decode(encryptedKey);
```

PathRank: 1.3337599, Rank: 0.46, Final: 1.3951129

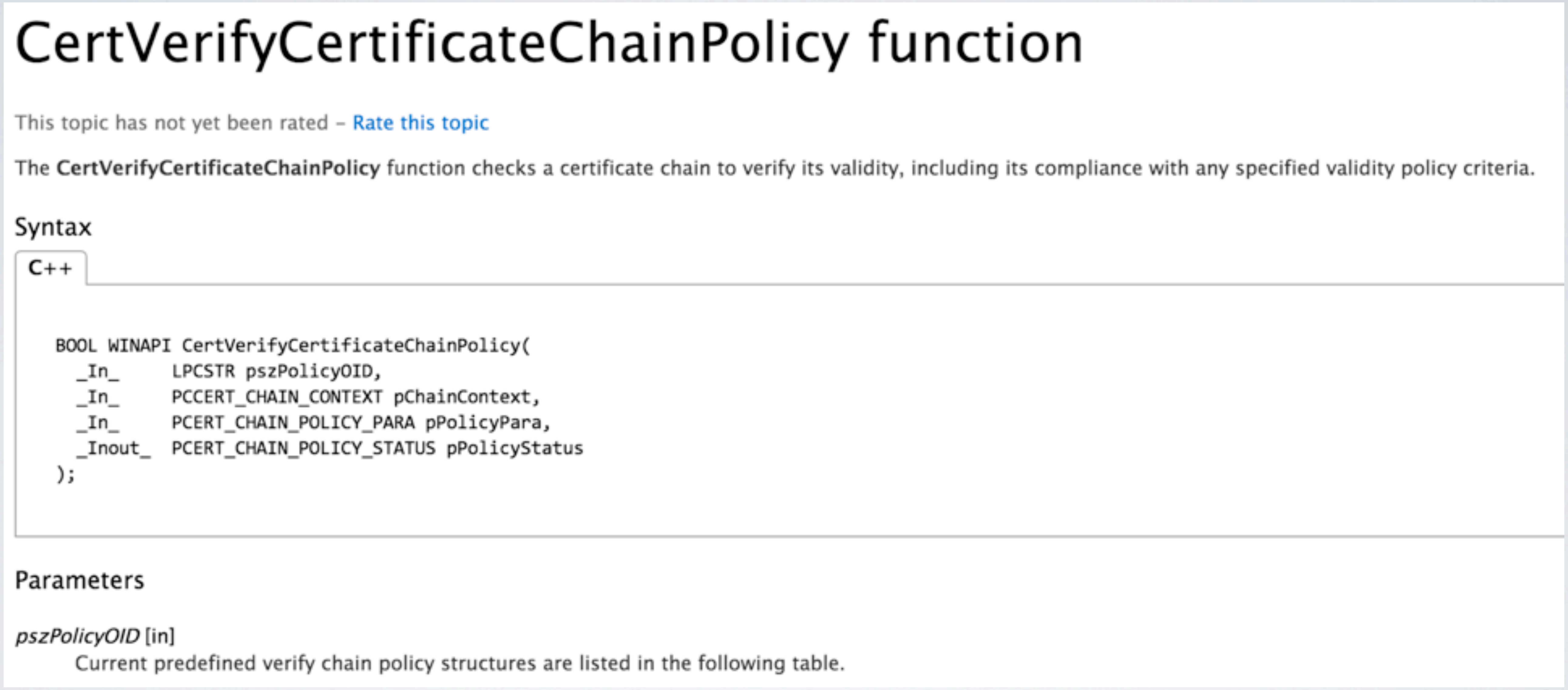
libjboss-web-services-java_0.0+svn5660+dak2-3/jbossws-native/org/jboss/ws/extensions/security/EncryptionOperation.java:187

```
         kgen.init(256);
         SecretKey key = kgen.generateKey();
         Cipher c = Cipher.getInstance("AES");
         c.init(Cipher.ENCRYPT_MODE, key);
```

PathRank: 1.0734495, Rank: 0.45499998, Final: 1.1341356

Source: Sun Java SE JDK 7

Non-intuitive interfaces 1/7



CertVerifyCertificateChainPolicy function

This topic has not yet been rated – Rate this topic

The **CertVerifyCertificateChainPolicy** function checks a certificate chain to verify its validity, including its compliance with any specified validity policy criteria.

Syntax

C++

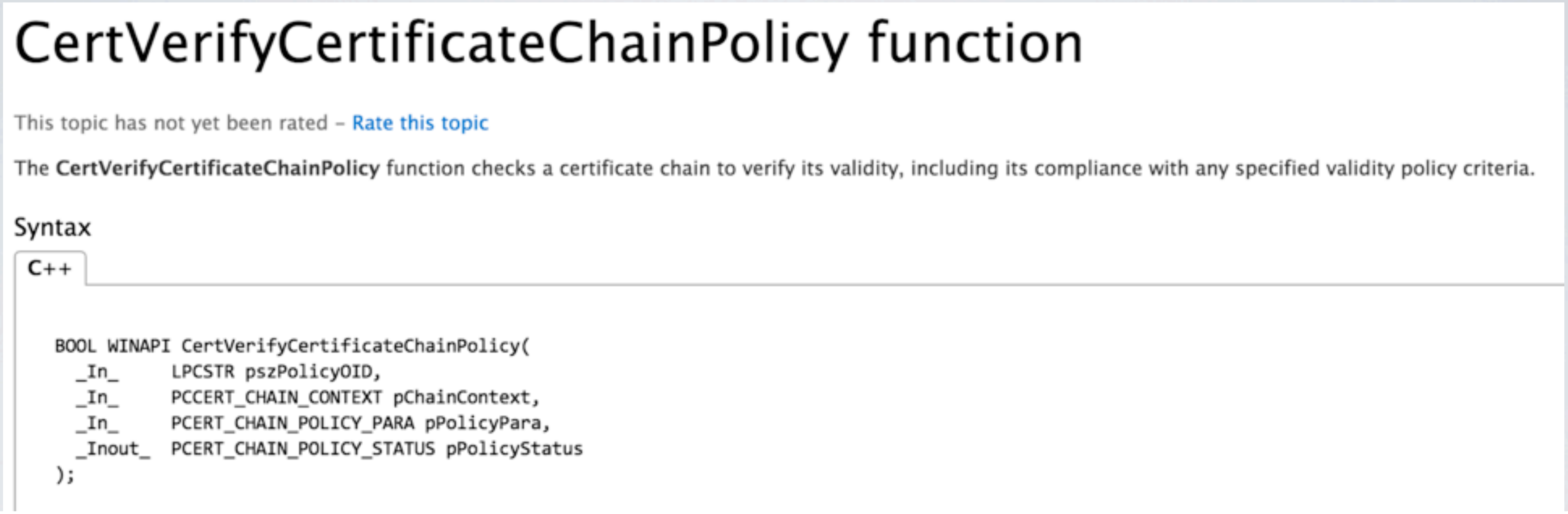
```
BOOL WINAPI CertVerifyCertificateChainPolicy(
  _In_     LPCSTR pszPolicyOID,
  _In_     PCCERT_CHAIN_CONTEXT pChainContext,
  _In_     PCERT_CHAIN_POLICY_PARA pPolicyPara,
  _Inout_  PCERT_CHAIN_POLICY_STATUS pPolicyStatus
);
```

Parameters

pszPolicyOID [in]
Current predefined verify chain policy structures are listed in the following table.

Source: MS Crypto API (current) h/t iarce

Non-intuitive interfaces 2/7



CertVerifyCertificateChainPolicy function

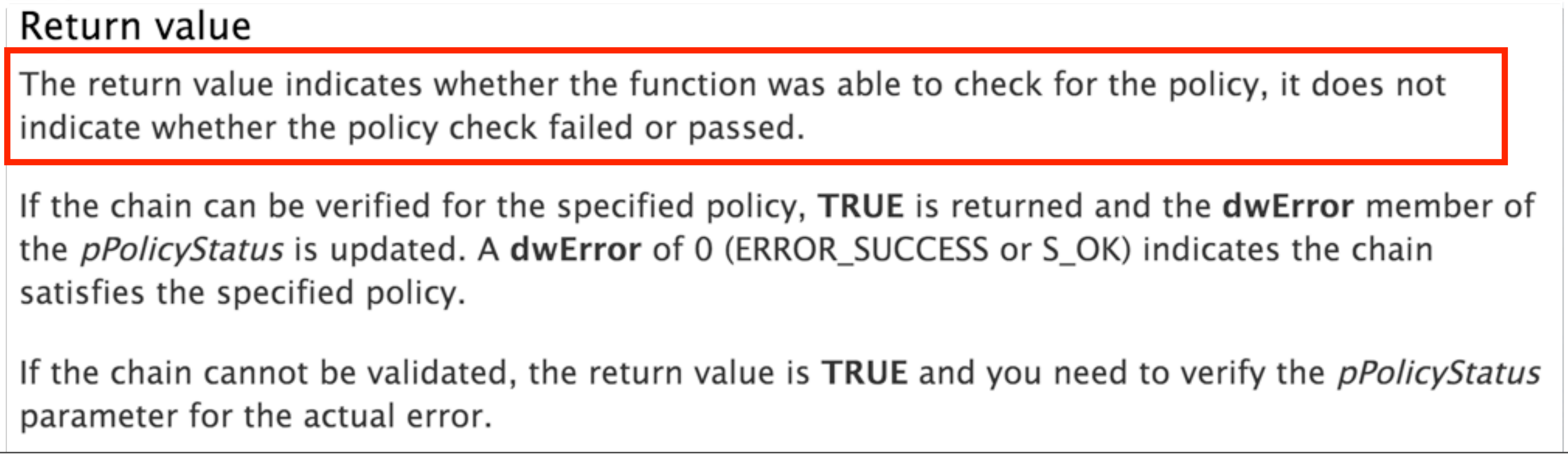
This topic has not yet been rated – Rate this topic

The **CertVerifyCertificateChainPolicy** function checks a certificate chain to verify its validity, including its compliance with any specified validity policy criteria.

Syntax

C++

```
BOOL WINAPI CertVerifyCertificateChainPolicy(
  _In_     LPCSTR pszPolicyOID,
  _In_     PCCERT_CHAIN_CONTEXT pChainContext,
  _In_     PCERT_CHAIN_POLICY_PARA pPolicyPara,
  _Inout_  PCERT_CHAIN_POLICY_STATUS pPolicyStatus
);
```



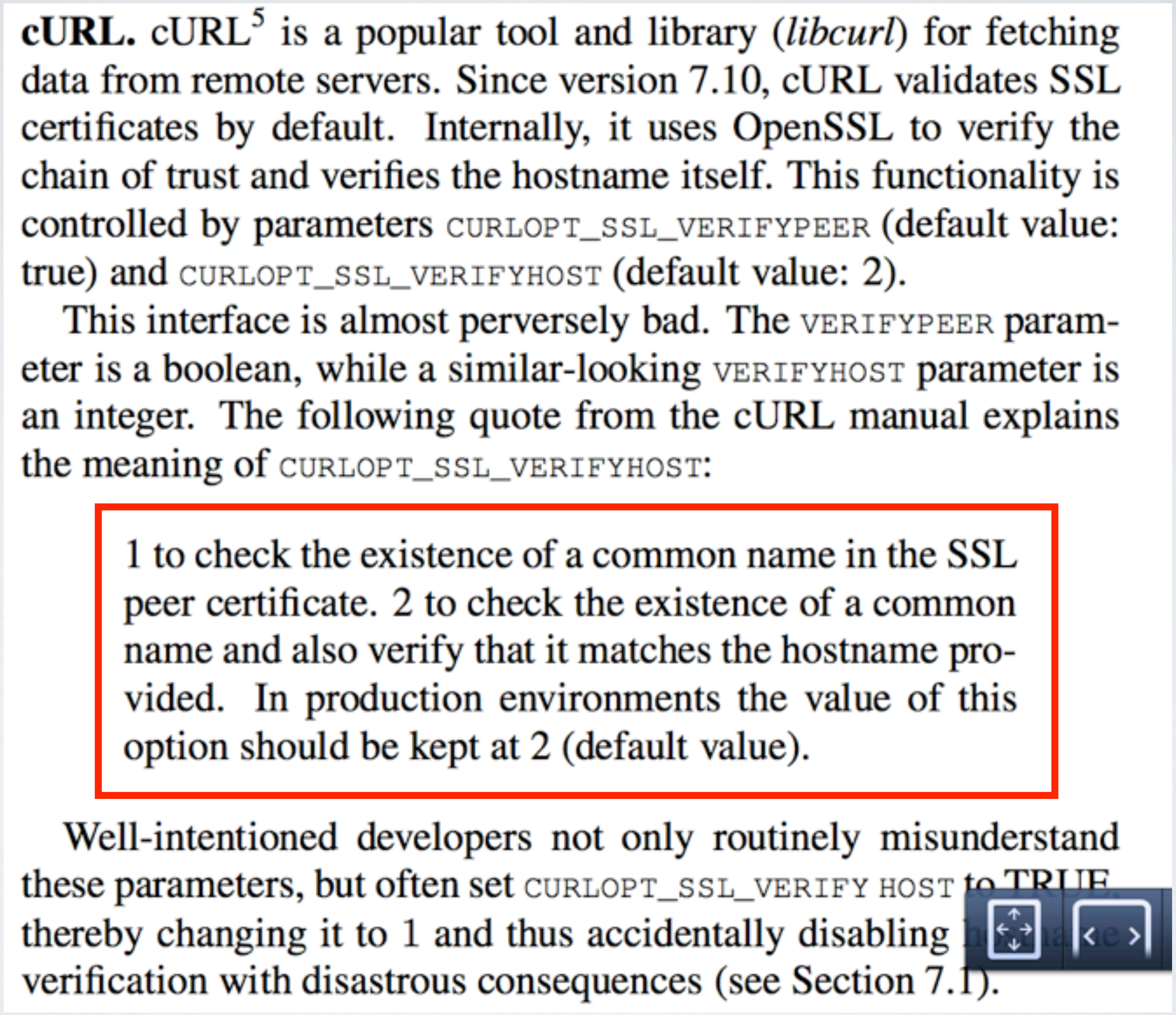
Return value

The return value indicates whether the function was able to check for the policy, it does not indicate whether the policy check failed or passed.

If the chain can be verified for the specified policy, **TRUE** is returned and the **dwError** member of the *pPolicyStatus* is updated. A **dwError** of 0 (ERROR_SUCCESS or S_OK) indicates the chain satisfies the specified policy.

If the chain cannot be validated, the return value is **TRUE** and you need to verify the *pPolicyStatus* parameter for the actual error.

Non-intuitive interfaces 3/7



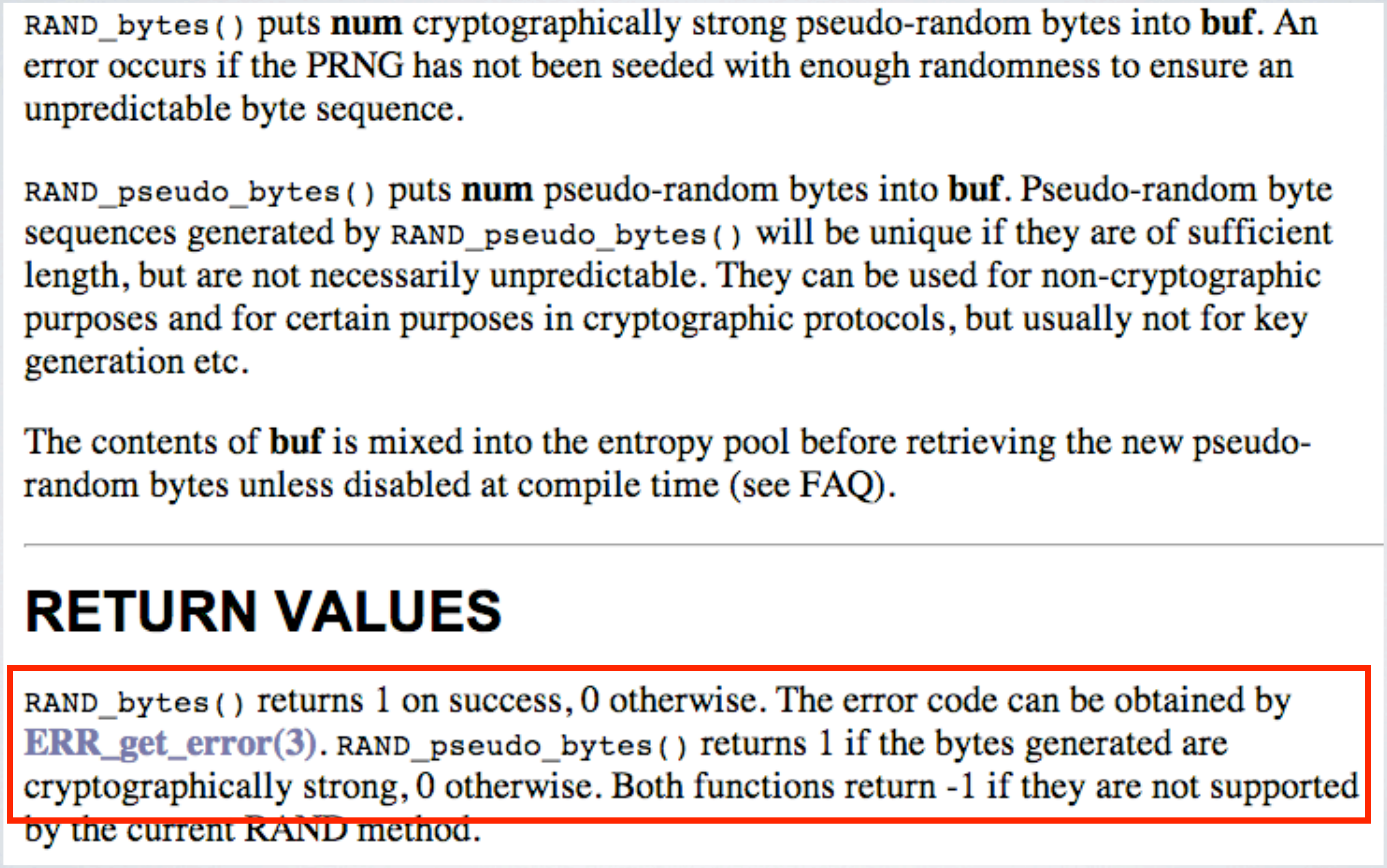
cURL. cURL[5] is a popular tool and library (*libcurl*) for fetching data from remote servers. Since version 7.10, cURL validates SSL certificates by default. Internally, it uses OpenSSL to verify the chain of trust and verifies the hostname itself. This functionality is controlled by parameters CURLOPT_SSL_VERIFYPEER (default value: true) and CURLOPT_SSL_VERIFYHOST (default value: 2).

This interface is almost perversely bad. The VERIFYPEER parameter is a boolean, while a similar-looking VERIFYHOST parameter is an integer. The following quote from the cURL manual explains the meaning of CURLOPT_SSL_VERIFYHOST:

> 1 to check the existence of a common name in the SSL peer certificate. 2 to check the existence of a common name and also verify that it matches the hostname provided. In production environments the value of this option should be kept at 2 (default value).

Well-intentioned developers not only routinely misunderstand these parameters, but often set CURLOPT_SSL_VERIFY HOST to TRUE, thereby changing it to 1 and thus accidentally disabling [illegible] verification with disastrous consequences (see Section 7.1).

Non-intuitive interfaces 4/7



RAND_bytes() puts **num** cryptographically strong pseudo-random bytes into **buf**. An error occurs if the PRNG has not been seeded with enough randomness to ensure an unpredictable byte sequence.

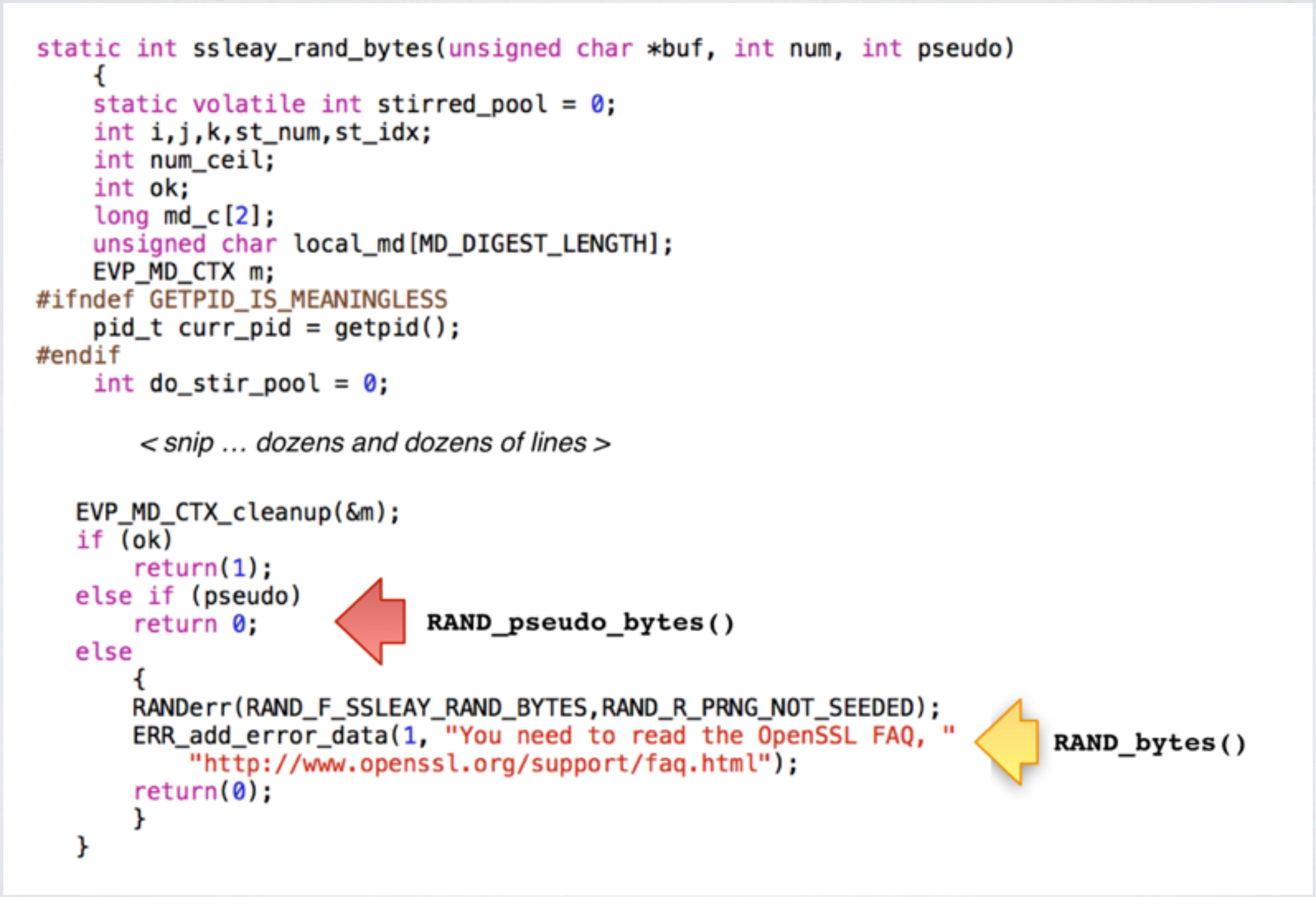
RAND_pseudo_bytes() puts **num** pseudo-random bytes into **buf**. Pseudo-random byte sequences generated by RAND_pseudo_bytes() will be unique if they are of sufficient length, but are not necessarily unpredictable. They can be used for non-cryptographic purposes and for certain purposes in cryptographic protocols, but usually not for key generation etc.

The contents of **buf** is mixed into the entropy pool before retrieving the new pseudo-random bytes unless disabled at compile time (see FAQ).

RETURN VALUES

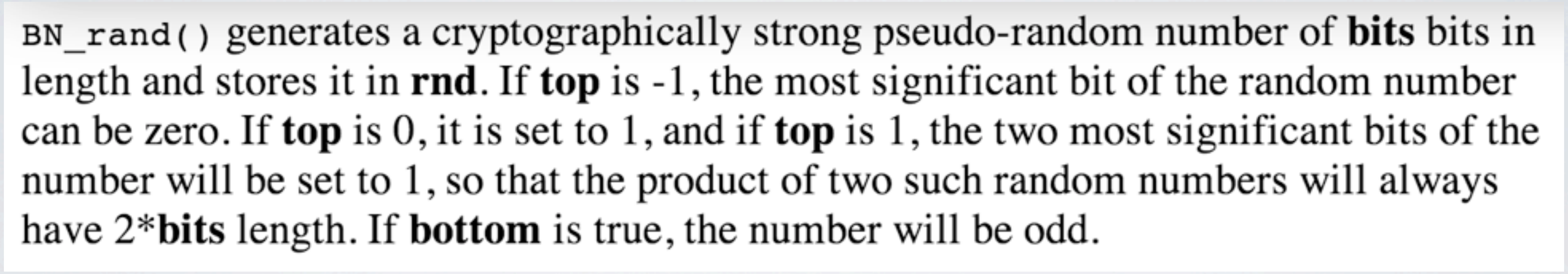
RAND_bytes() returns 1 on success, 0 otherwise. The error code can be obtained by ERR_get_error(3). RAND_pseudo_bytes() returns 1 if the bytes generated are cryptographically strong, 0 otherwise. Both functions return -1 if they are not supported by the current RAND method.

Non-intuitive interfaces 5/7



```
static int ssleay_rand_bytes(unsigned char *buf, int num, int pseudo)
    {
    static volatile int stirred_pool = 0;
    int i,j,k,st_num,st_idx;
    int num_ceil;
    int ok;
    long md_c[2];
    unsigned char local_md[MD_DIGEST_LENGTH];
    EVP_MD_CTX m;
#ifndef GETPID_IS_MEANINGLESS
    pid_t curr_pid = getpid();
#endif
    int do_stir_pool = 0;

        < snip ... dozens and dozens of lines >

   EVP_MD_CTX_cleanup(&m);
   if (ok)
       return(1);
   else if (pseudo)
       return 0;
   else
       {
       RANDerr(RAND_F_SSLEAY_RAND_BYTES,RAND_R_PRNG_NOT_SEEDED);
       ERR_add_error_data(1, "You need to read the OpenSSL FAQ, "
           "http://www.openssl.org/support/faq.html");
       return(0);
       }
   }
```

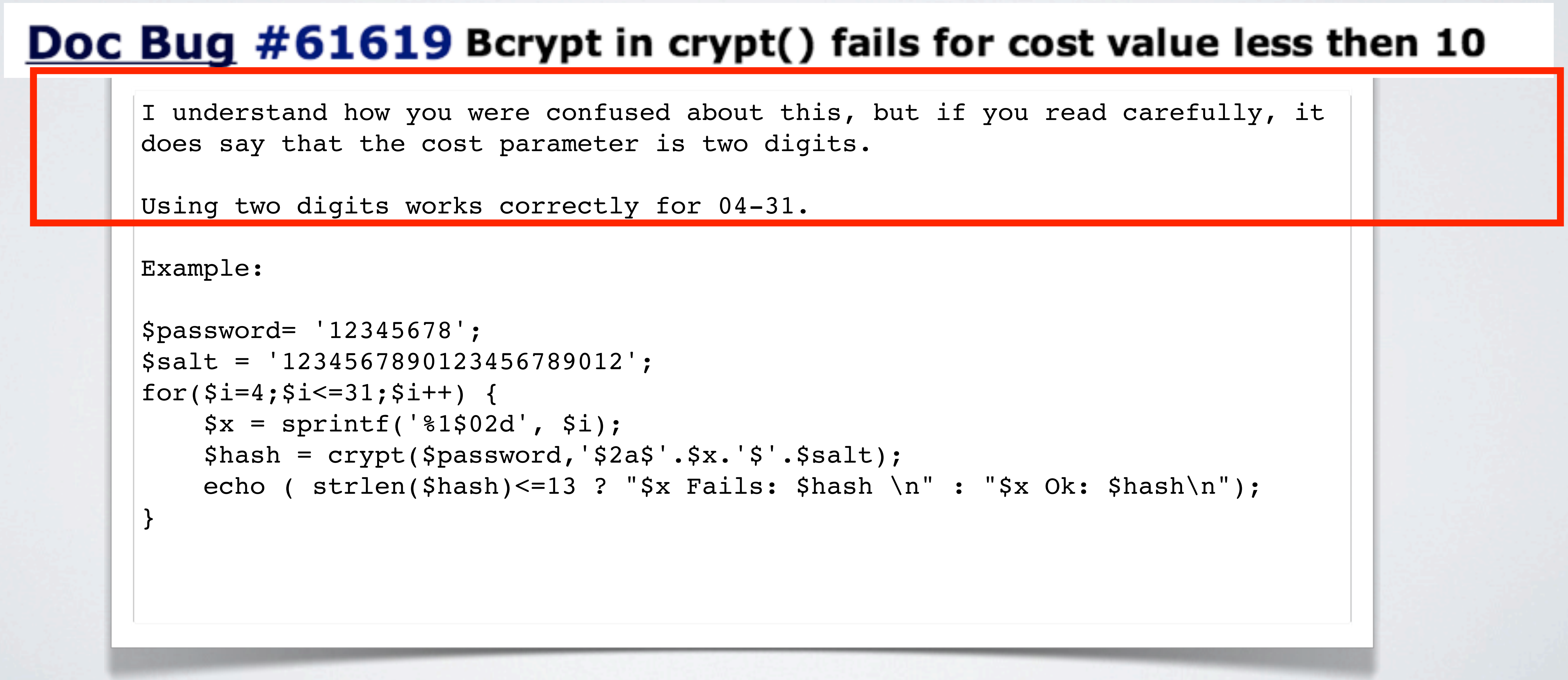
Non-intuitive interfaces 6/7



`BN_rand()` generates a cryptographically strong pseudo-random number of **bits** bits in length and stores it in **rnd**. If **top** is -1, the most significant bit of the random number can be zero. If **top** is 0, it is set to 1, and if **top** is 1, the two most significant bits of the number will be set to 1, so that the product of two such random numbers will always have 2***bits** length. If **bottom** is true, the number will be odd.

Non-intuitive interfaces 7/7

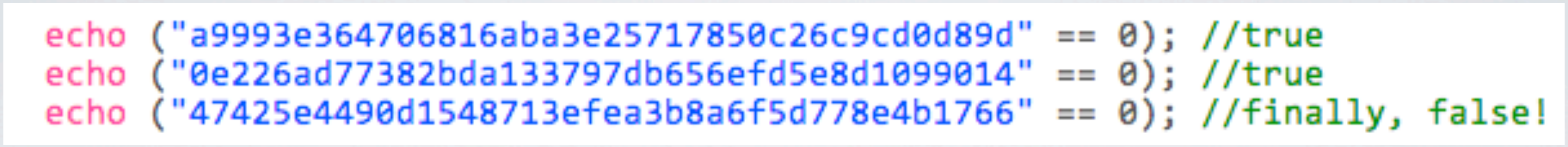
- Every single thing in PHP



Doc Bug #61619 **Bcrypt in crypt() fails for cost value less then 10**

```
I understand how you were confused about this, but if you read carefully, it
does say that the cost parameter is two digits.

Using two digits works correctly for 04-31.

Example:

$password= '12345678';
$salt = '1234567890123456789012';
for($i=4;$i<=31;$i++) {
    $x = sprintf('%1$02d', $i);
    $hash = crypt($password,'$2a$'.$x.'$'.$salt);
    echo ( strlen($hash)<=13 ? "$x Fails: $hash \n" : "$x Ok: $hash\n");
}
```

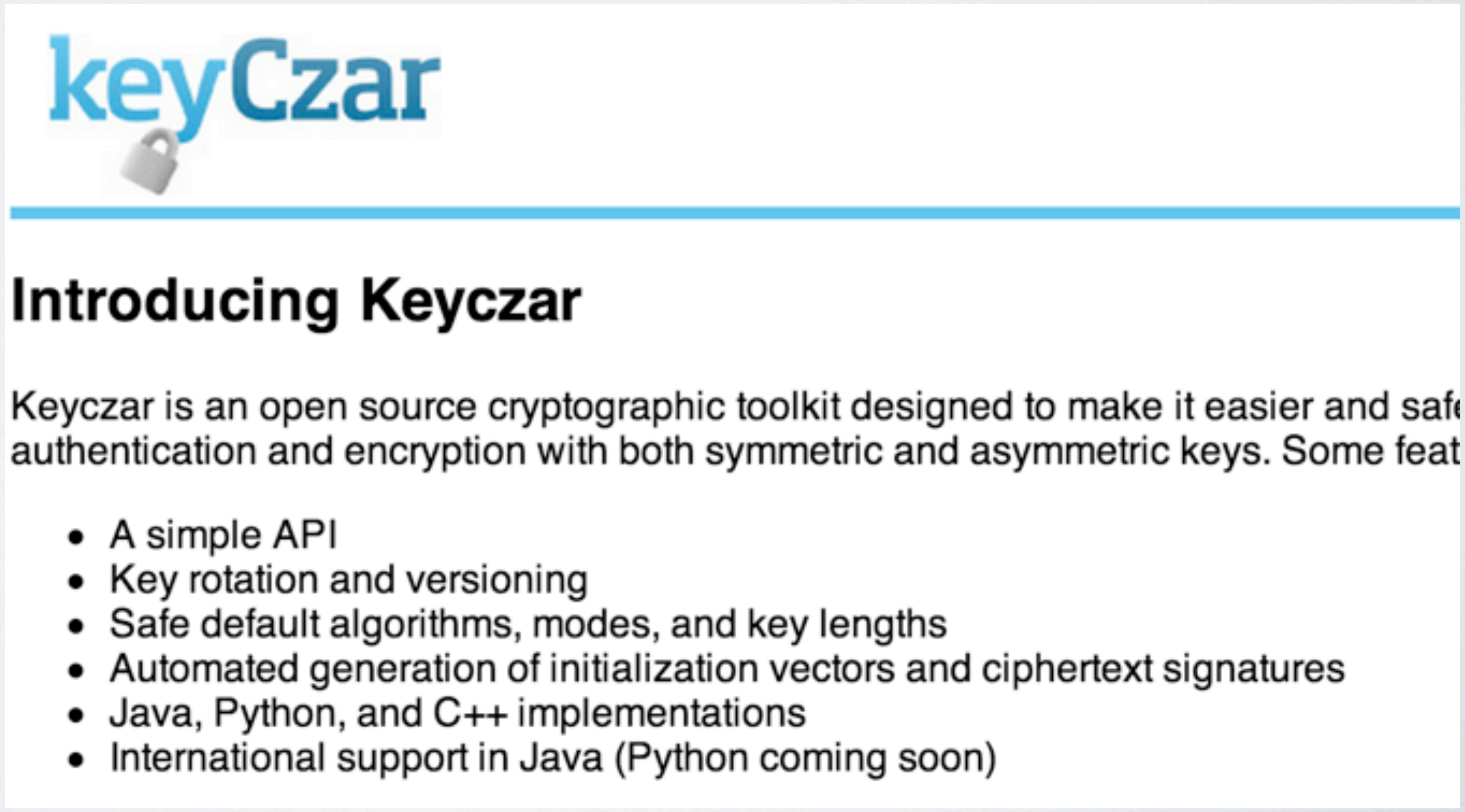
Language problems



```
echo ("a9993e364706816aba3e25717850c26c9cd0d89d" == 0); //true
echo ("0e226ad77382bda133797db656efd5e8d1099014" == 0); //true
echo ("47425e4490d1548713efea3b8a6f5d778e4b1766" == 0); //finally, false!
```

Key management

- Surprisingly few tools devoted to securely managing keys
    - Secure memory storage
    - Updating/revoking/distributing keys



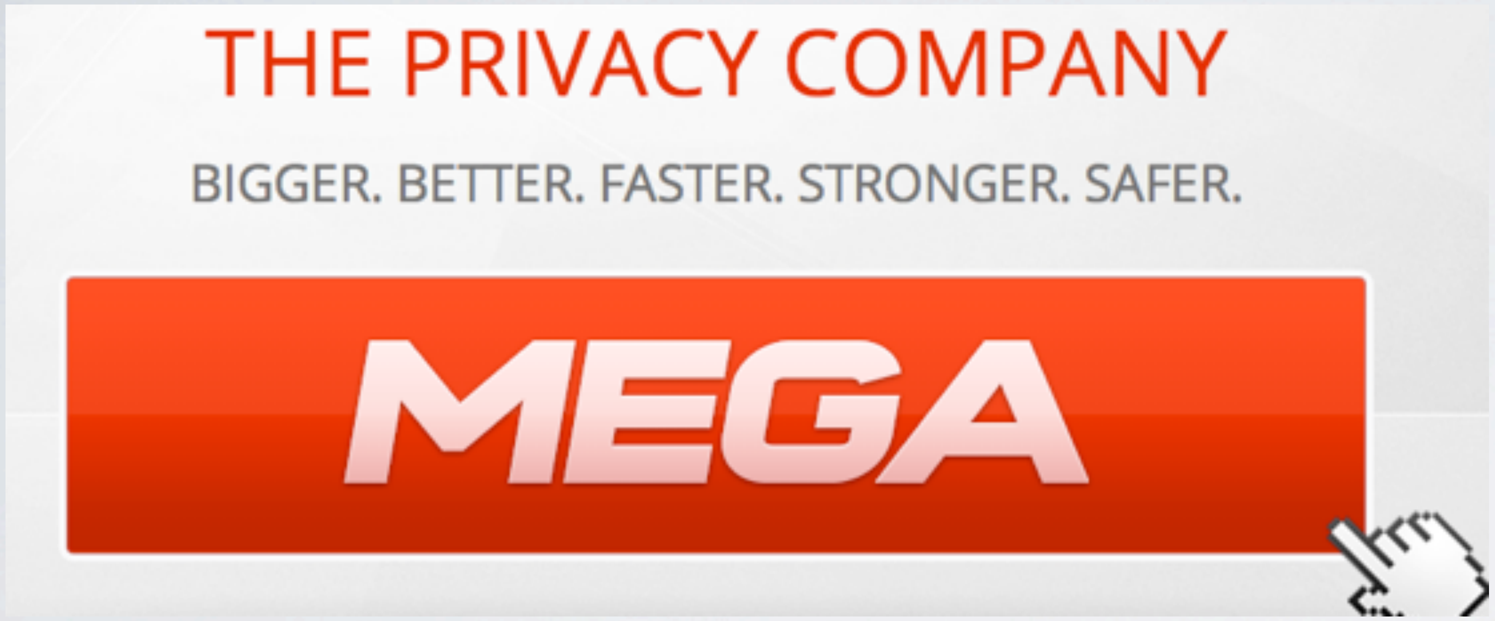
keyCzar

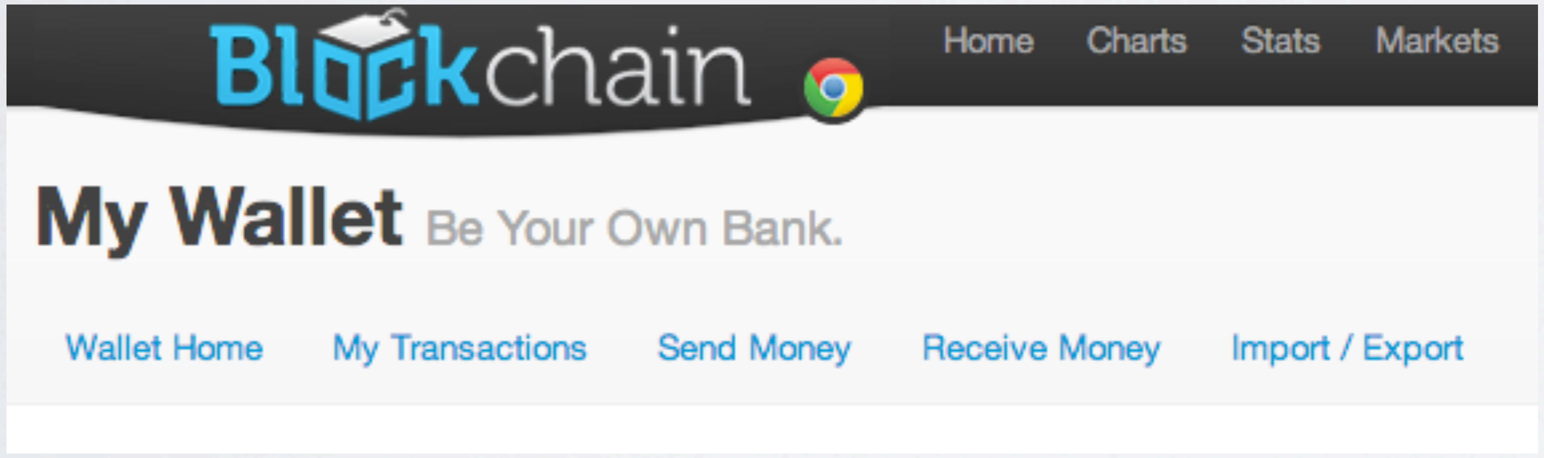
Introducing Keyczar

Keyczar is an open source cryptographic toolkit designed to make it easier and saf
authentication and encryption with both symmetric and asymmetric keys. Some feat

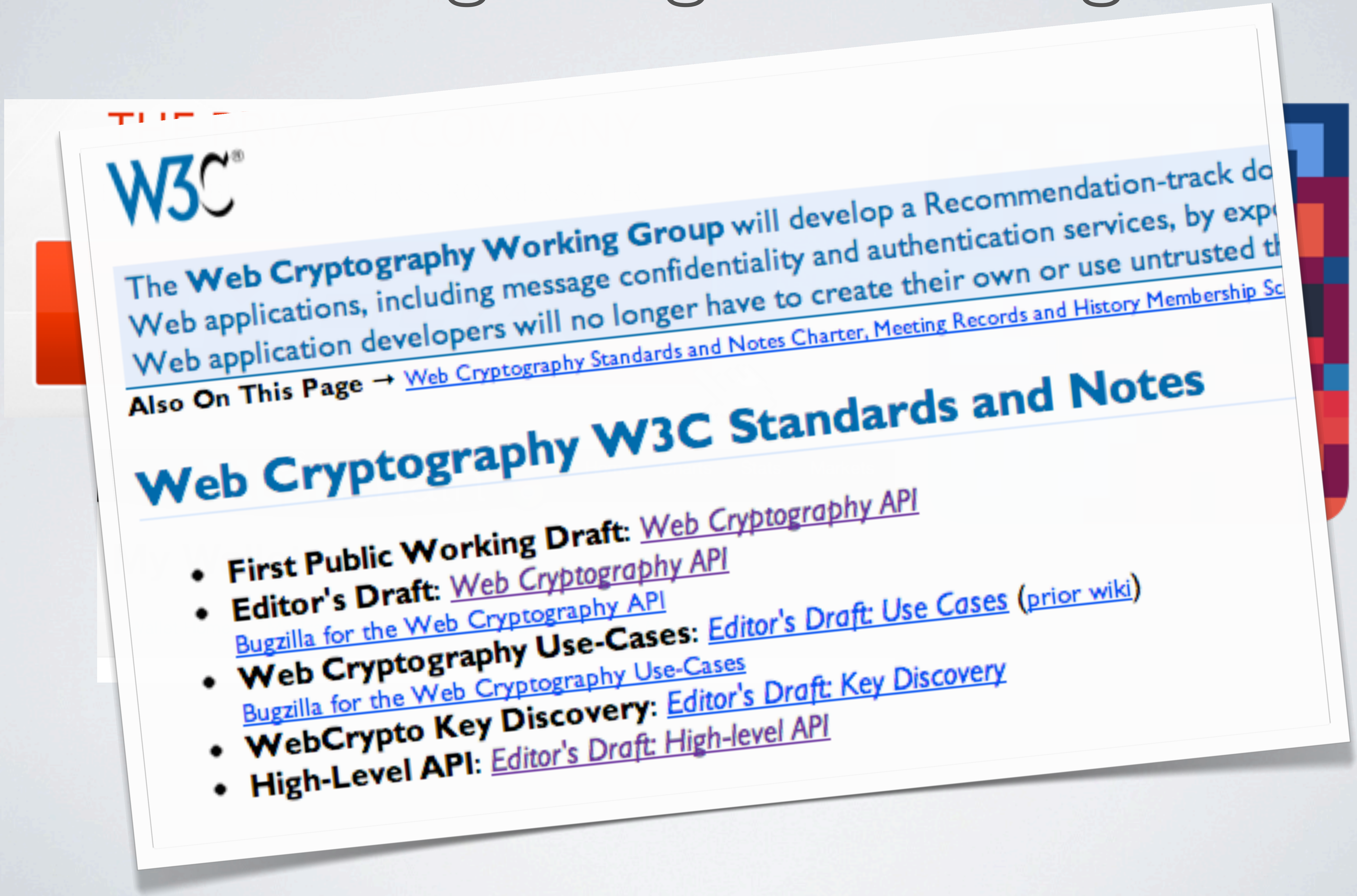
- A simple API
- Key rotation and versioning
- Safe default algorithms, modes, and key lengths
- Automated generation of initialization vectors and ciphertext signatures
- Java, Python, and C++ implementations
- International support in Java (Python coming soon)

But it's getting better. Right?





But it's getting better. Right?



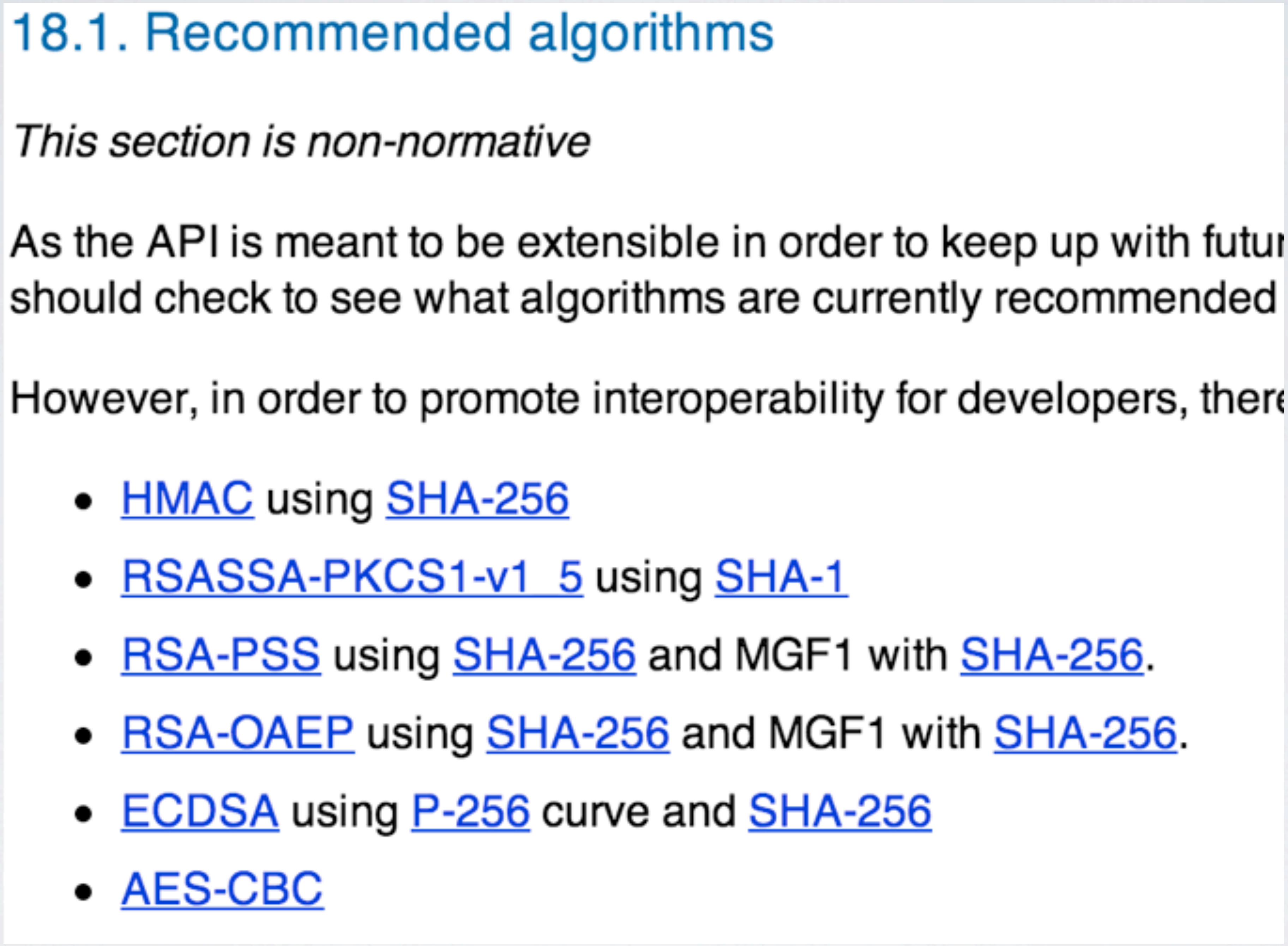
The **Web Cryptography Working Group** will develop a Recommendation-track do
Web applications, including message confidentiality and authentication services, by exp
Web application developers will no longer have to create their own or use untrusted th

Also On This Page → Web Cryptography Standards and Notes Charter, Meeting Records and History Membership Sc

Web Cryptography W3C Standards and Notes

- **First Public Working Draft:** Web Cryptography API
- **Editor's Draft:** Web Cryptography API
  Bugzilla for the Web Cryptography API
- **Web Cryptography Use-Cases:** Editor's Draft: Use Cases (prior wiki)
  Bugzilla for the Web Cryptography Use-Cases
- **WebCrypto Key Discovery:** Editor's Draft: Key Discovery
- **High-Level API:** Editor's Draft: High-level API

But it's getting better. Right?



18.1. Recommended algorithms

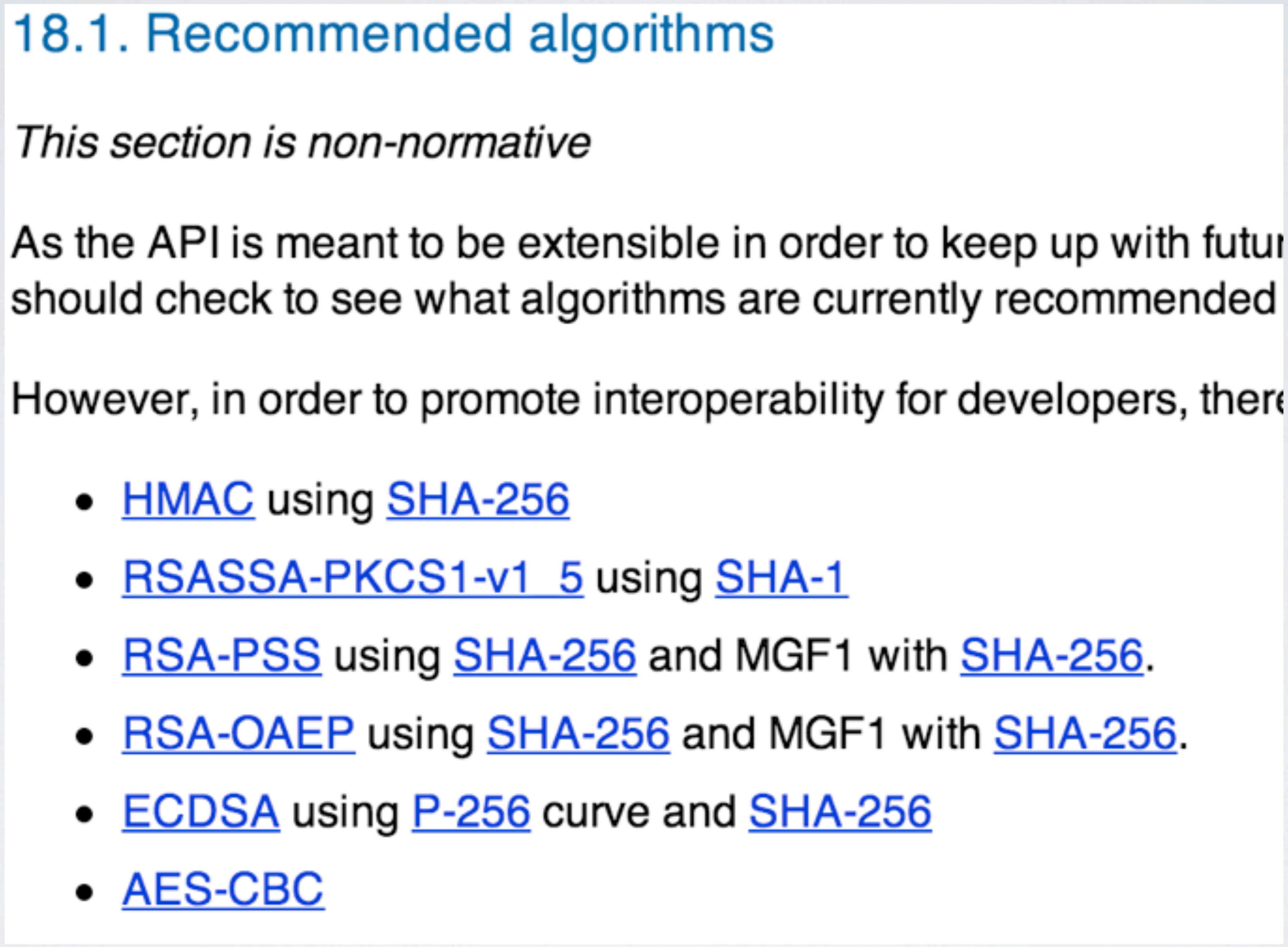
This section is non-normative

As the API is meant to be extensible in order to keep up with futu
should check to see what algorithms are currently recommended

However, in order to promote interoperability for developers, ther

- HMAC using SHA-256
- RSASSA-PKCS1-v1_5 using SHA-1
- RSA-PSS using SHA-256 and MGF1 with SHA-256.
- RSA-OAEP using SHA-256 and MGF1 with SHA-256.
- ECDSA using P-256 curve and SHA-256
- AES-CBC

But it's getting better. Right?



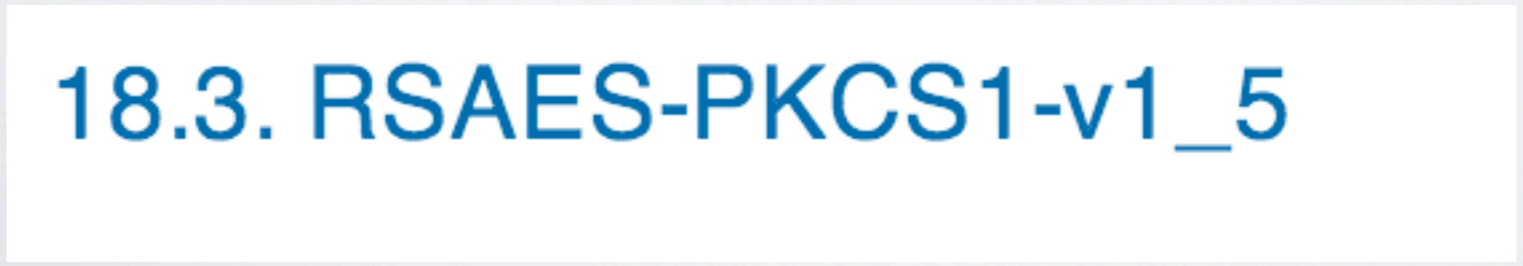
18.1. Recommended algorithms

This section is non-normative

As the API is meant to be extensible in order to keep up with futur
should check to see what algorithms are currently recommended

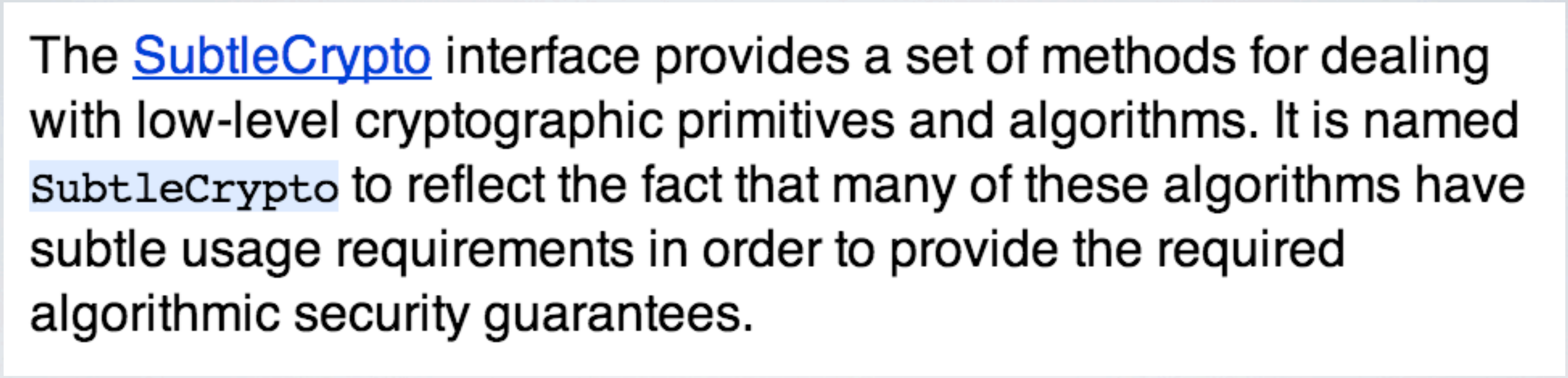
However, in order to promote interoperability for developers, there

- HMAC using SHA-256
- RSASSA-PKCS1-v1_5 using SHA-1
- RSA-PSS using SHA-256 and MGF1 with SHA-256.
- RSA-OAEP using SHA-256 and MGF1 with SHA-256.
- ECDSA using P-256 curve and SHA-256
- AES-CBC

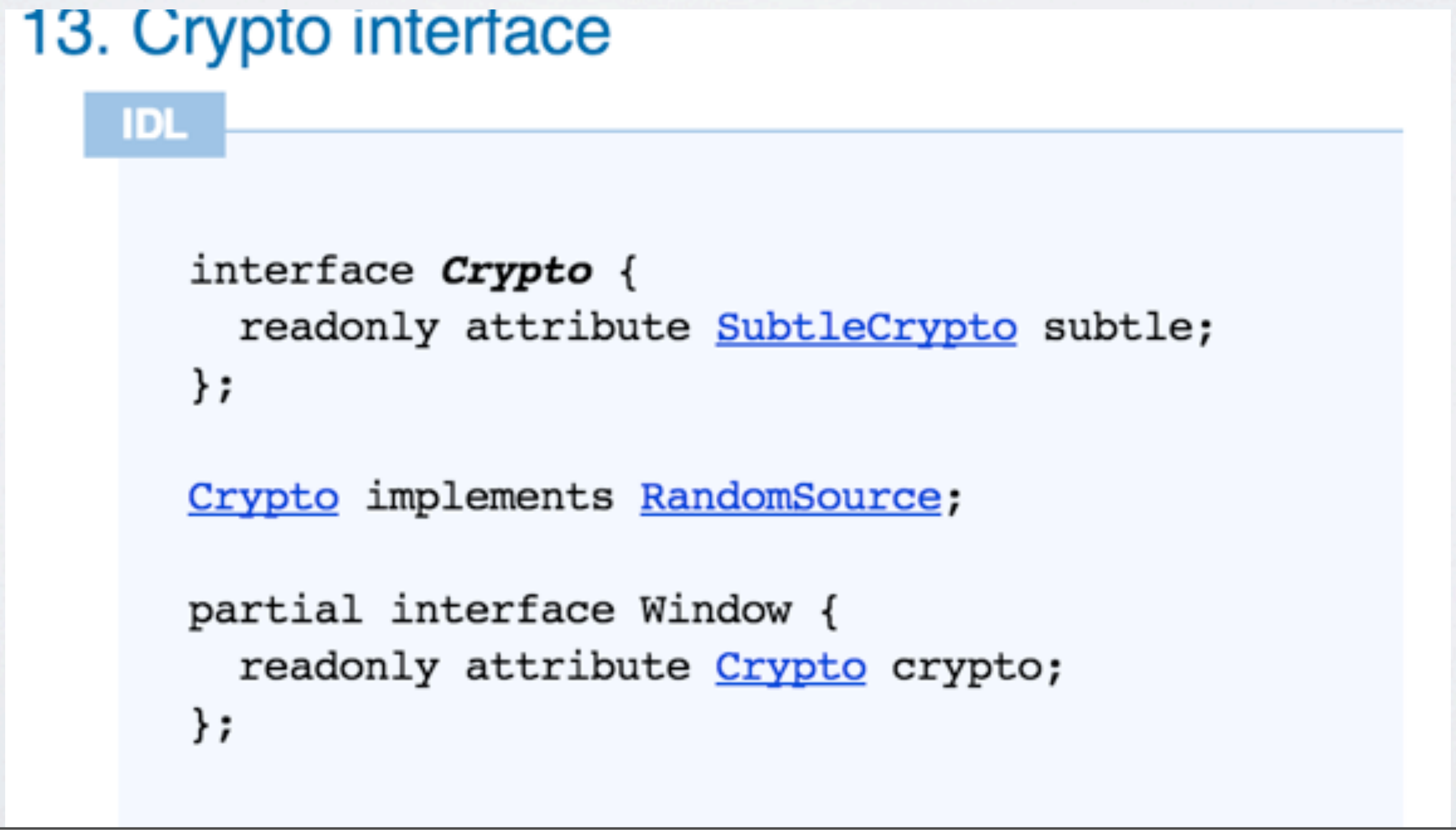


18.3. RSAES-PKCS1-v1_5

But it's getting better. Right?



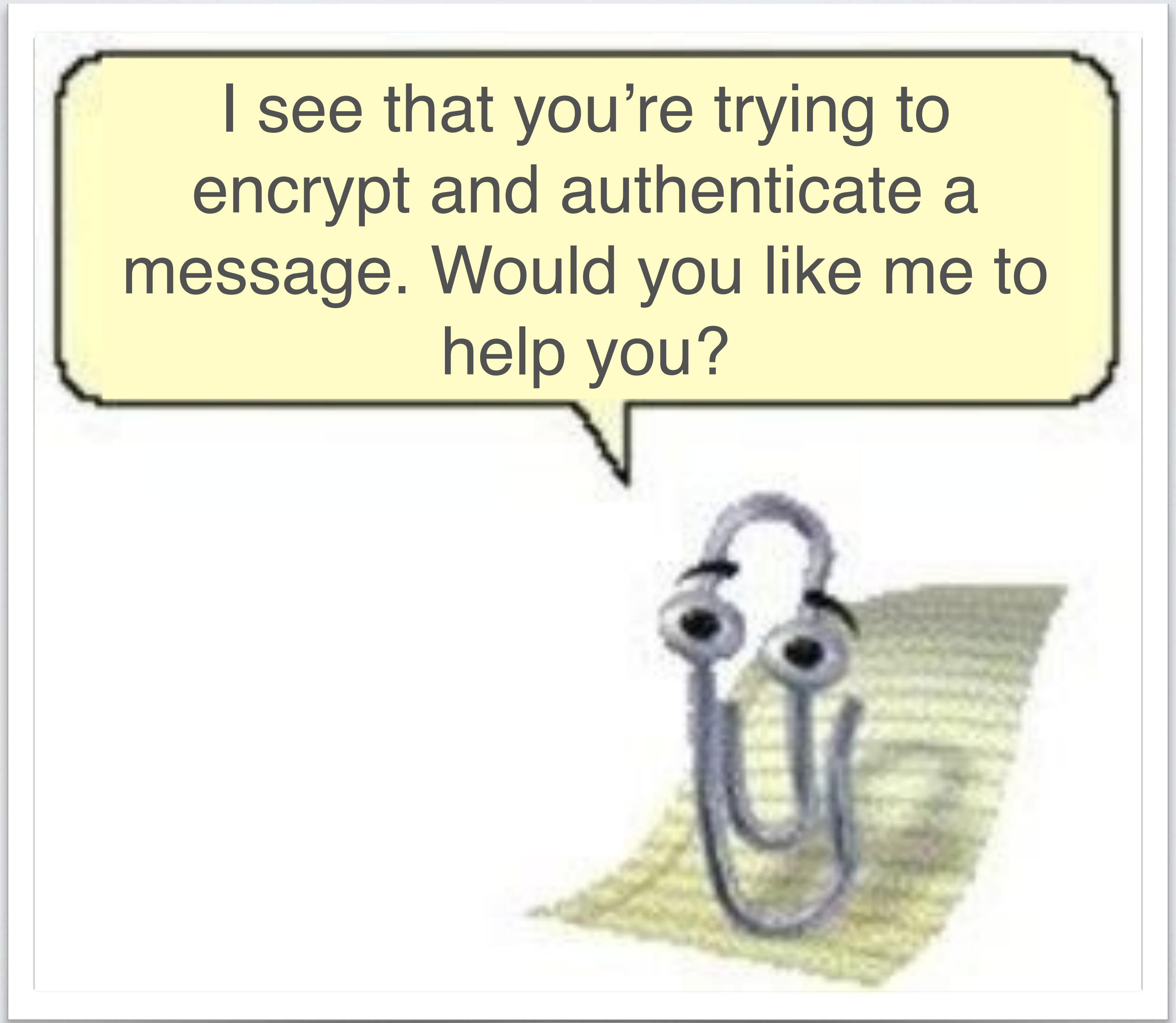
The SubtleCrypto interface provides a set of methods for dealing with low-level cryptographic primitives and algorithms. It is named `SubtleCrypto` to reflect the fact that many of these algorithms have subtle usage requirements in order to provide the required algorithmic security guarantees.



13. Crypto interface

```
interface Crypto {
  readonly attribute SubtleCrypto subtle;
};

Crypto implements RandomSource;

partial interface Window {
  readonly attribute Crypto crypto;
};
```

What to do about it?


I see that you're trying to encrypt and authenticate a message. Would you like me to help you?

Usage resilient cryptography

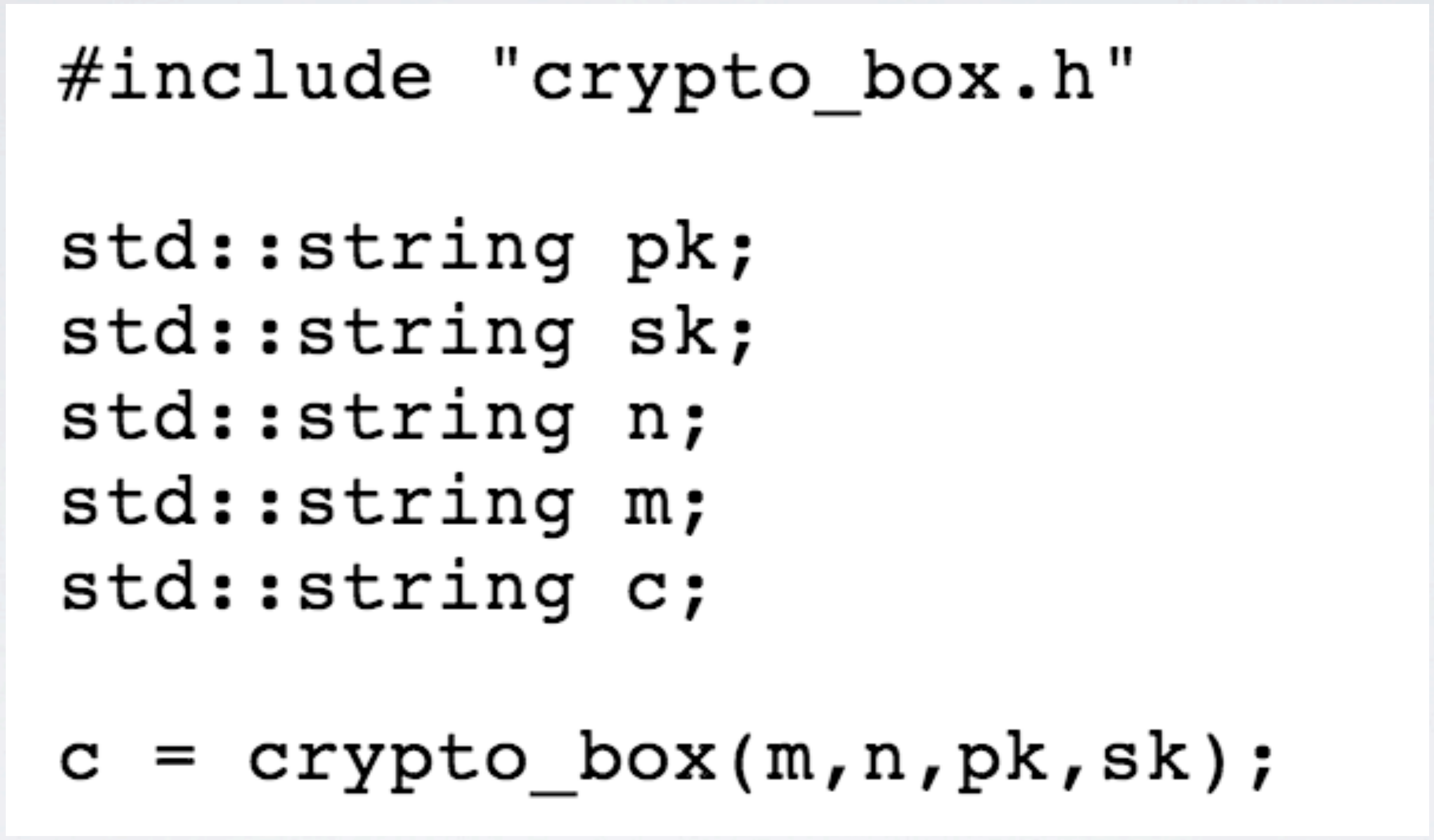
API simplicity 1/3

- Simplify the API
  - Most users don’t need legacy support
  - So remove the options and choose for them
  - Eliminate complex data structures
  - Simplify error cases

API simplicity 2/3

```c
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;
```

API simplicity 3/3

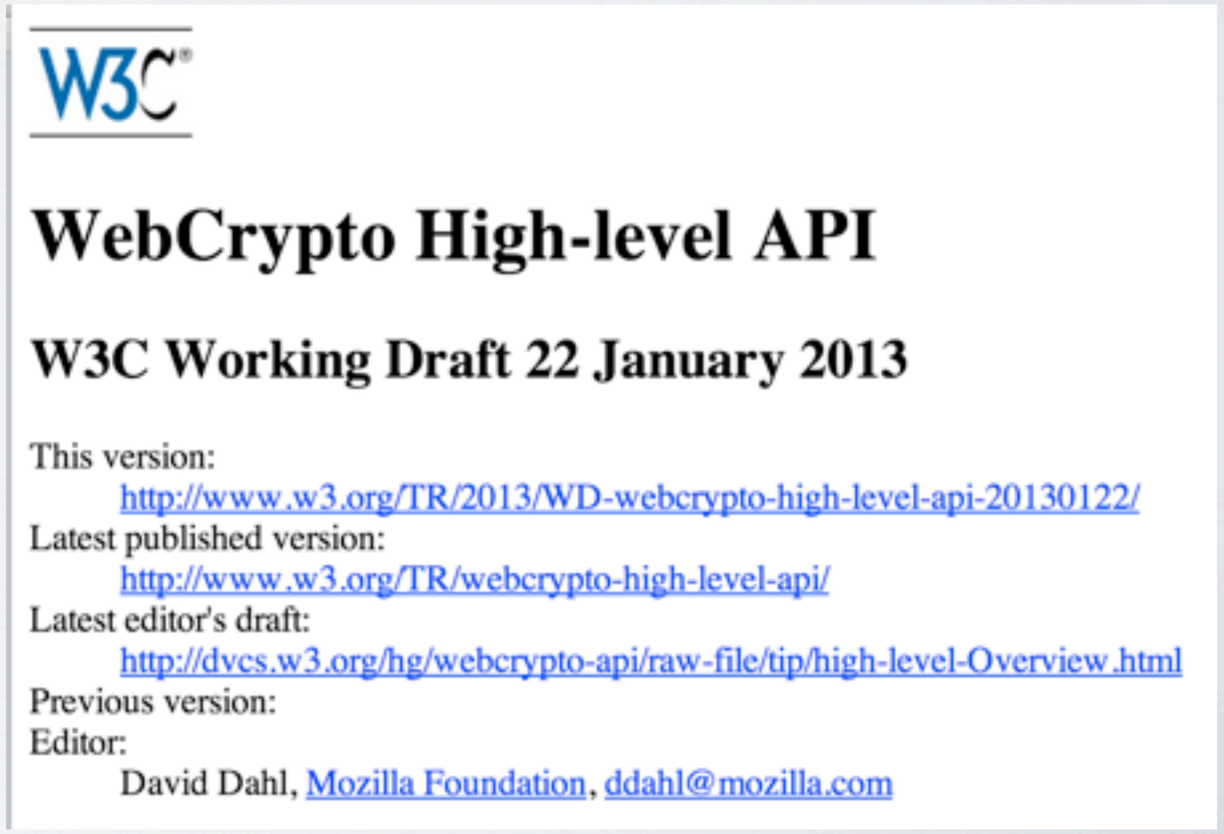


```
#include "crypto_box.h"

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

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

Dan Bernstein/Tanja Lange/Peter Schwabe: NaCl

API layering

- Separate API layers
  - Low-level APIs for 'expert' users (legacy support)
  - High level APIs for all others
  - Already present in the W3C Specification



W3C

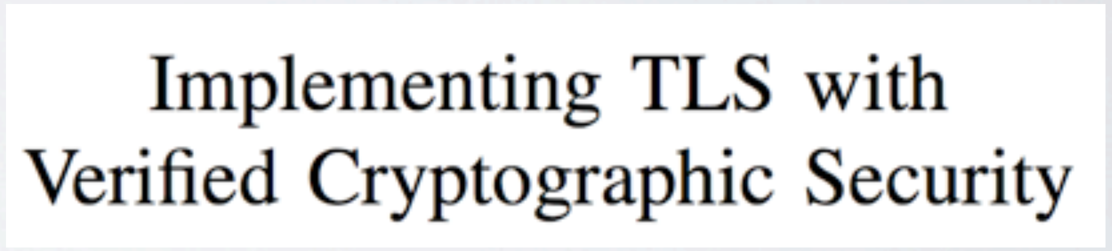
WebCrypto High-level API

W3C Working Draft 22 January 2013

This version:
http://www.w3.org/TR/2013/WD-webcrypto-high-level-api-20130122/
Latest published version:
http://www.w3.org/TR/webcrypto-high-level-api/
Latest editor's draft:
http://dvcs.w3.org/hg/webcrypto-api/raw-file/tip/high-level-Overview.html
Previous version:
Editor:
David Dahl, Mozilla Foundation, ddahl@mozilla.com

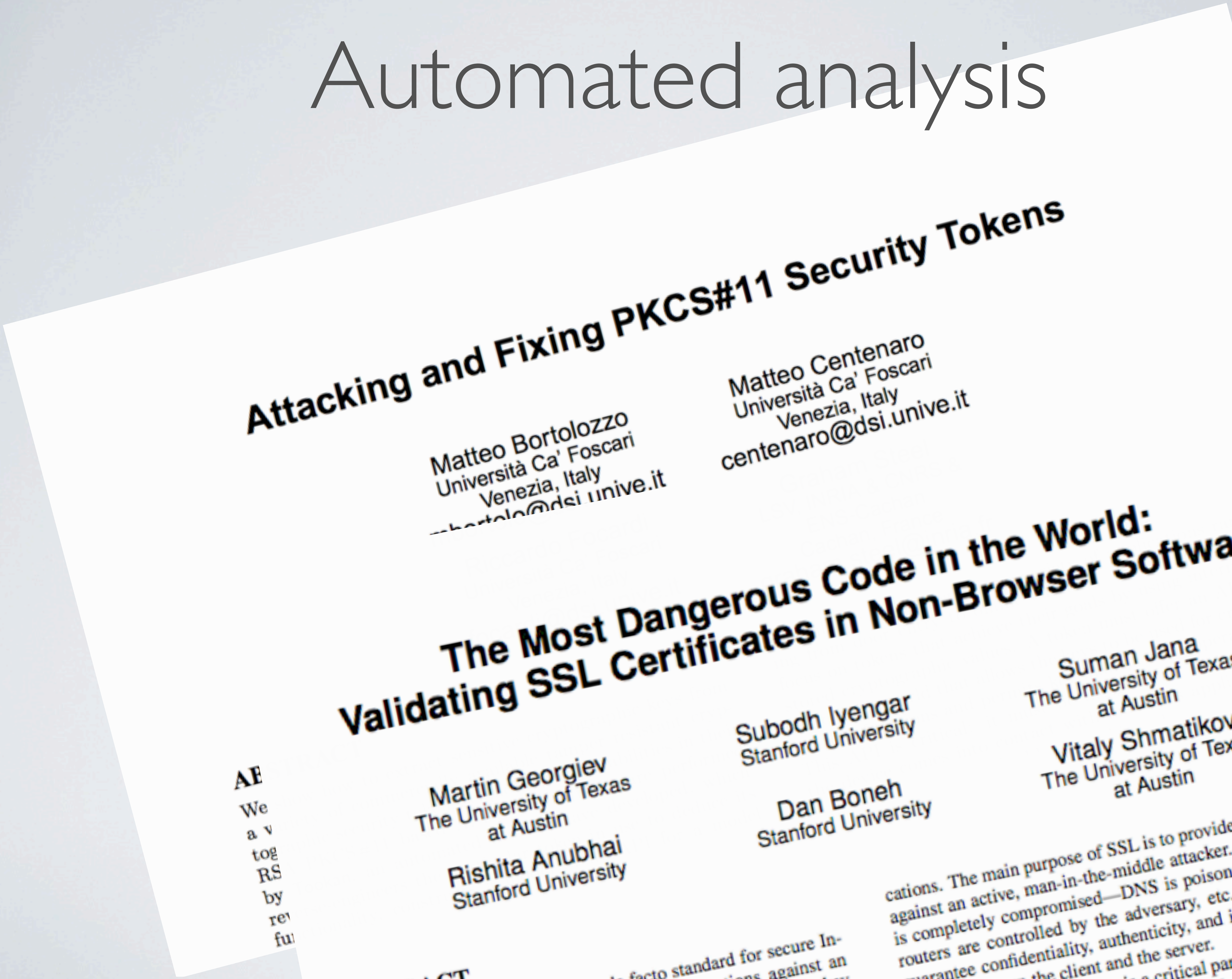
Formal methods/languages

- (Domain specific) languages for cryptography/security
  - Enforce security as a type safety problem
  - Ben Laurie et al. Haskell translator for symmetric crypto
  - Microsoft DKM/SSL library & F7 analysis
    - Very promising direction
    - But probability of developer adoption is low
    - For example: Microsoft's DKM was validated in F7, *then re-implemented in C#*



Implementing TLS with Verified Cryptographic Security

Automated analysis



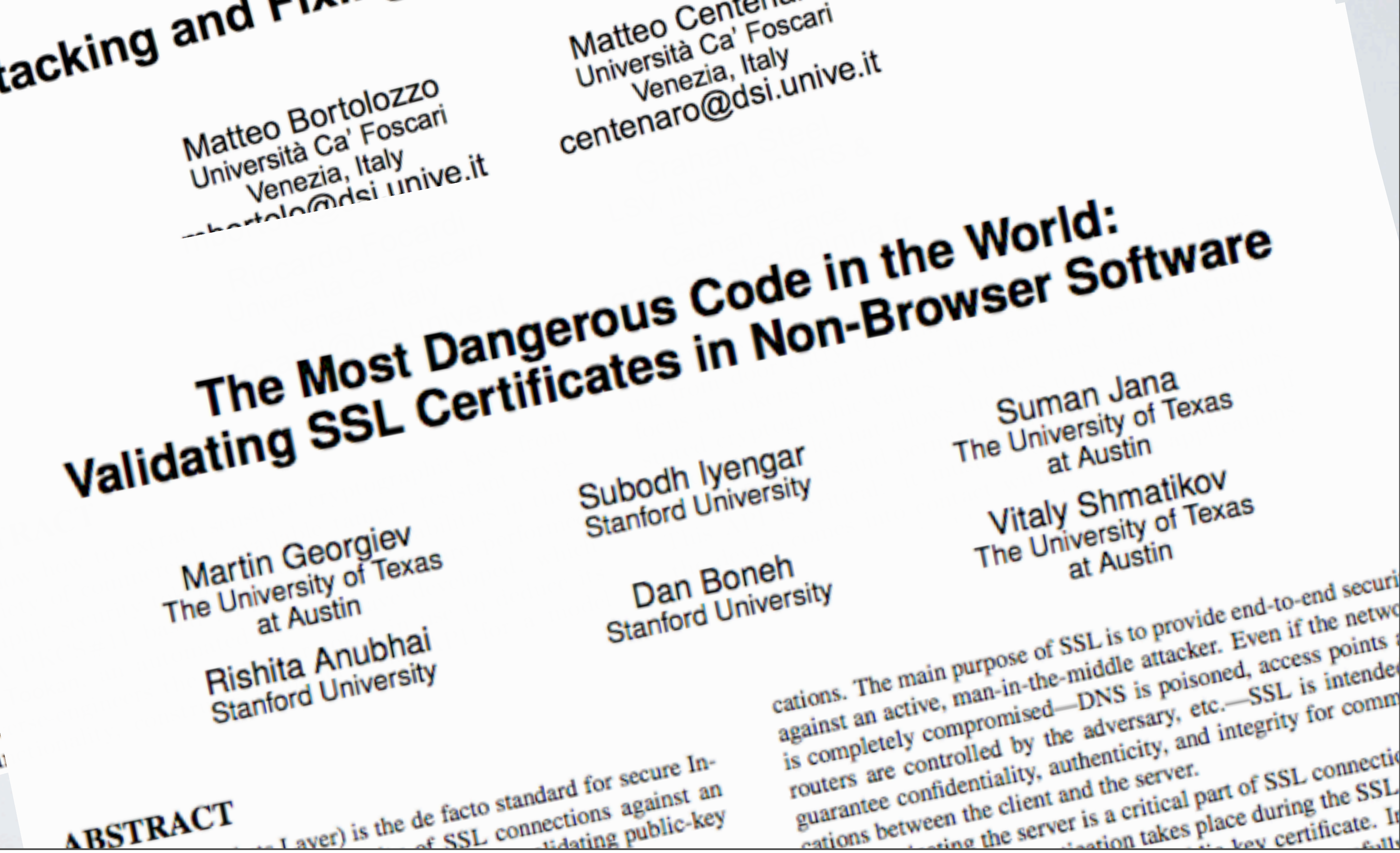
Attacking and Fixing PKCS#11 Security Tokens

Matteo Bortolozzo
Università Ca' Foscari
Venezia, Italy
unive.it

Matteo Centenaro
Università Ca' Foscari
Venezia, Italy
centenaro@dsi.unive.it

AB

We
a v
tog
RS
by
rev
fu



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

Rishita Anubhai
Stanford University

Dan Boneh
Stanford University

Vitaly Shmatikov
The University of Texas
at Austin

ABSTRACT

Layer) is the de facto standard for secure In-
of SSL connections against an
lidating public-key

cations. The main purpose of SSL is to provide end-to-end securi
against an active, man-in-the-middle attacker. Even if the netwo
is completely compromised—DNS is poisoned, access points a
routers are controlled by the adversary, etc.—SSL is intende
guarantee confidentiality, authenticity, and integrity for comm
cations between the client and the server.
ating the server is a critical part of SSL connecti
tion takes place during the SSL
key certificate. In

Wednesday, August 14, 13