API design for cryptography

Frank Denis - @jedisct1

Who’s that creepy guy?

Frank Denis
@jedisct1

https://primulinus.com

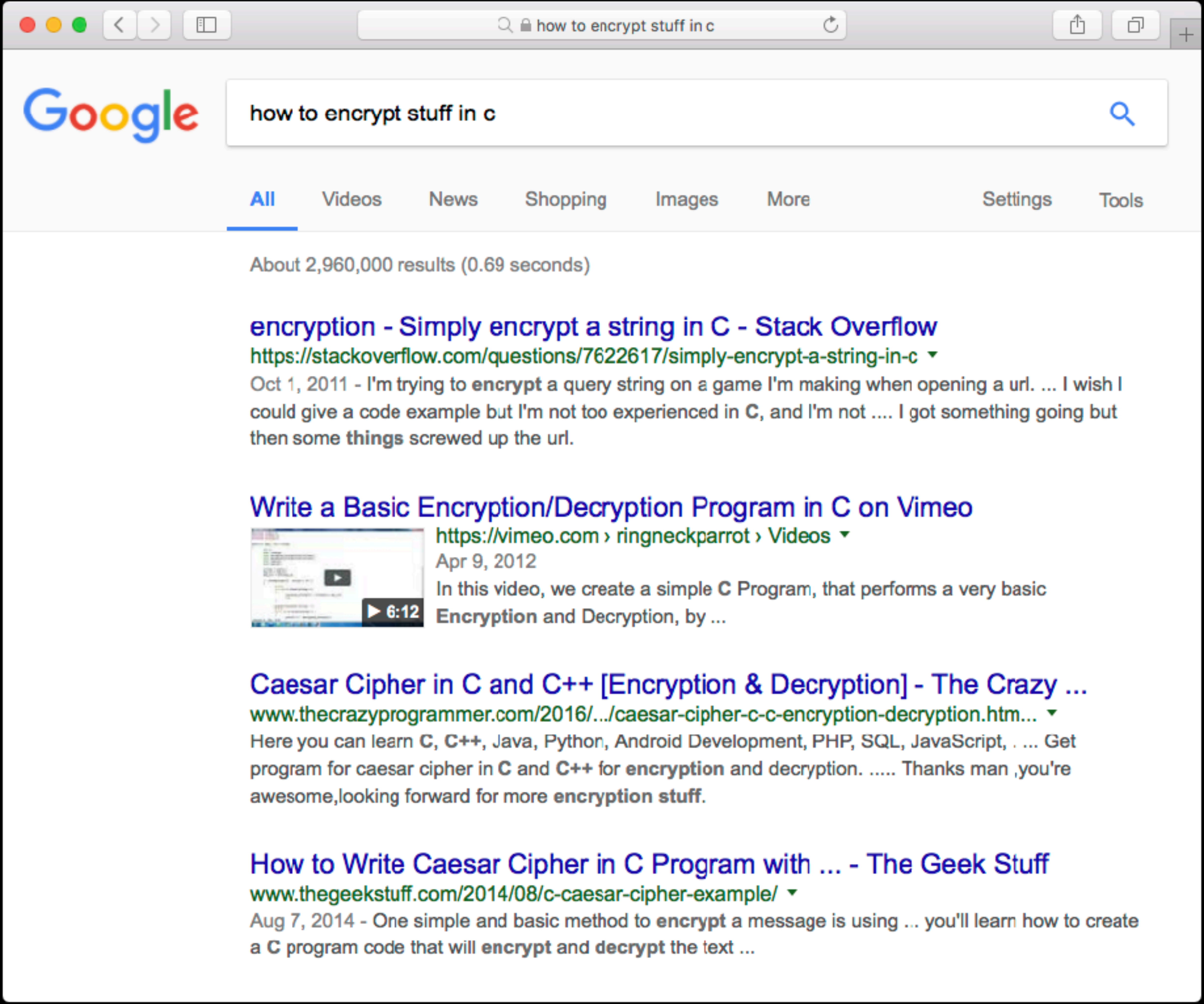
Application security, cryptography, malware analysis, protocol design, computer vision/digital image processing…

OSS zealot

Spends way too much time on Twitter

Crypto is everywhere

And its domain extends way beyond mere encryption.



how to encrypt stuff in c

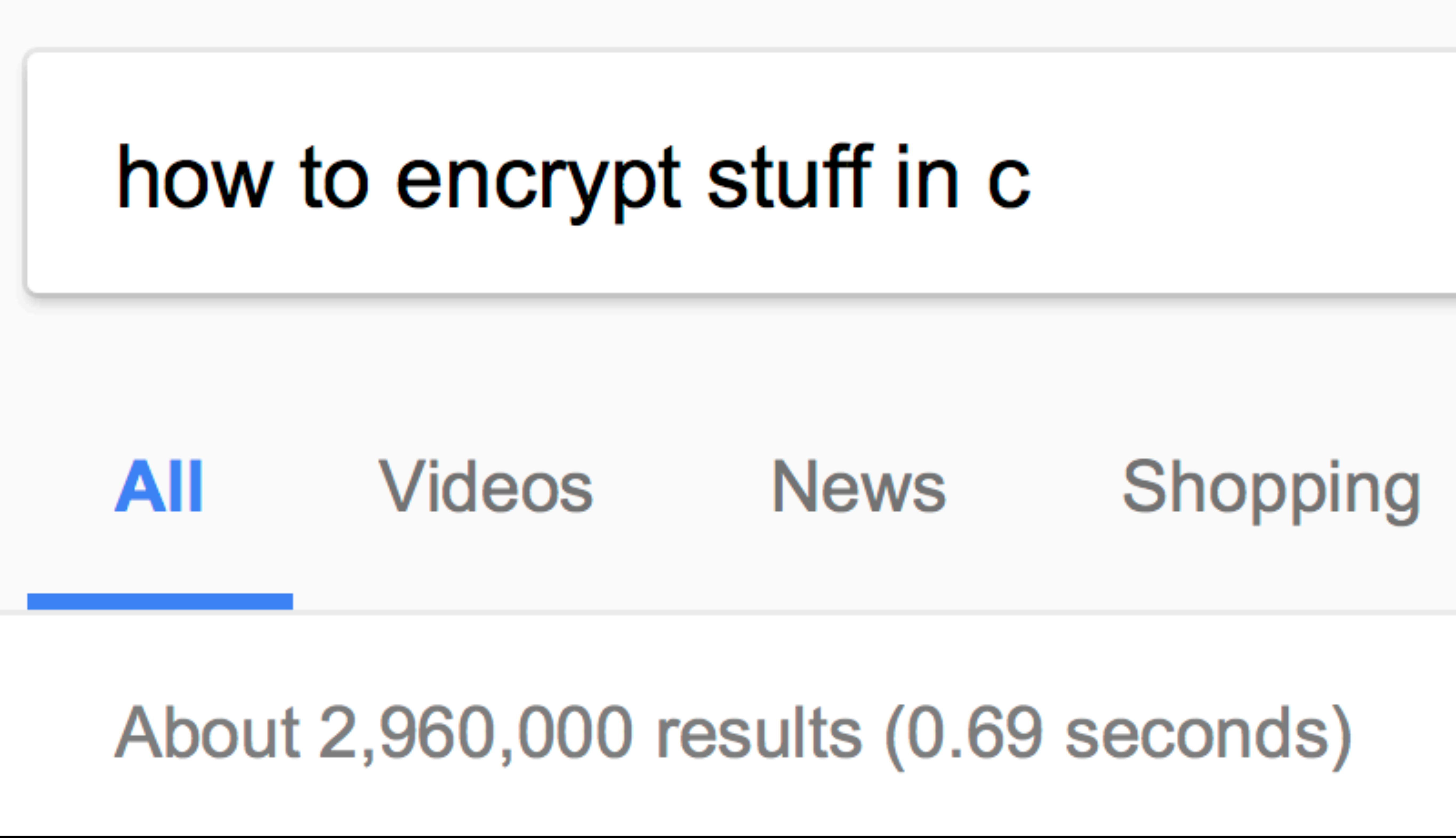
About 2,960,000 results (0.69 seconds)

encryption - Simply encrypt a string in C - Stack Overflow
https://stackoverflow.com/questions/7622617/simply-encrypt-a-string-in-c
Oct 1, 2011 - I'm trying to **encrypt** a query string on a game I'm making when opening a url. ... I wish I could give a code example but I'm not too experienced in **C**, and I'm not .... I got something going but then some **things** screwed up the url.

Write a Basic Encryption/Decryption Program in C on Vimeo
https://vimeo.com › ringneckparrot › Videos
Apr 9, 2012
In this video, we create a simple **C** Program, that performs a very basic **Encryption** and Decryption, by ...

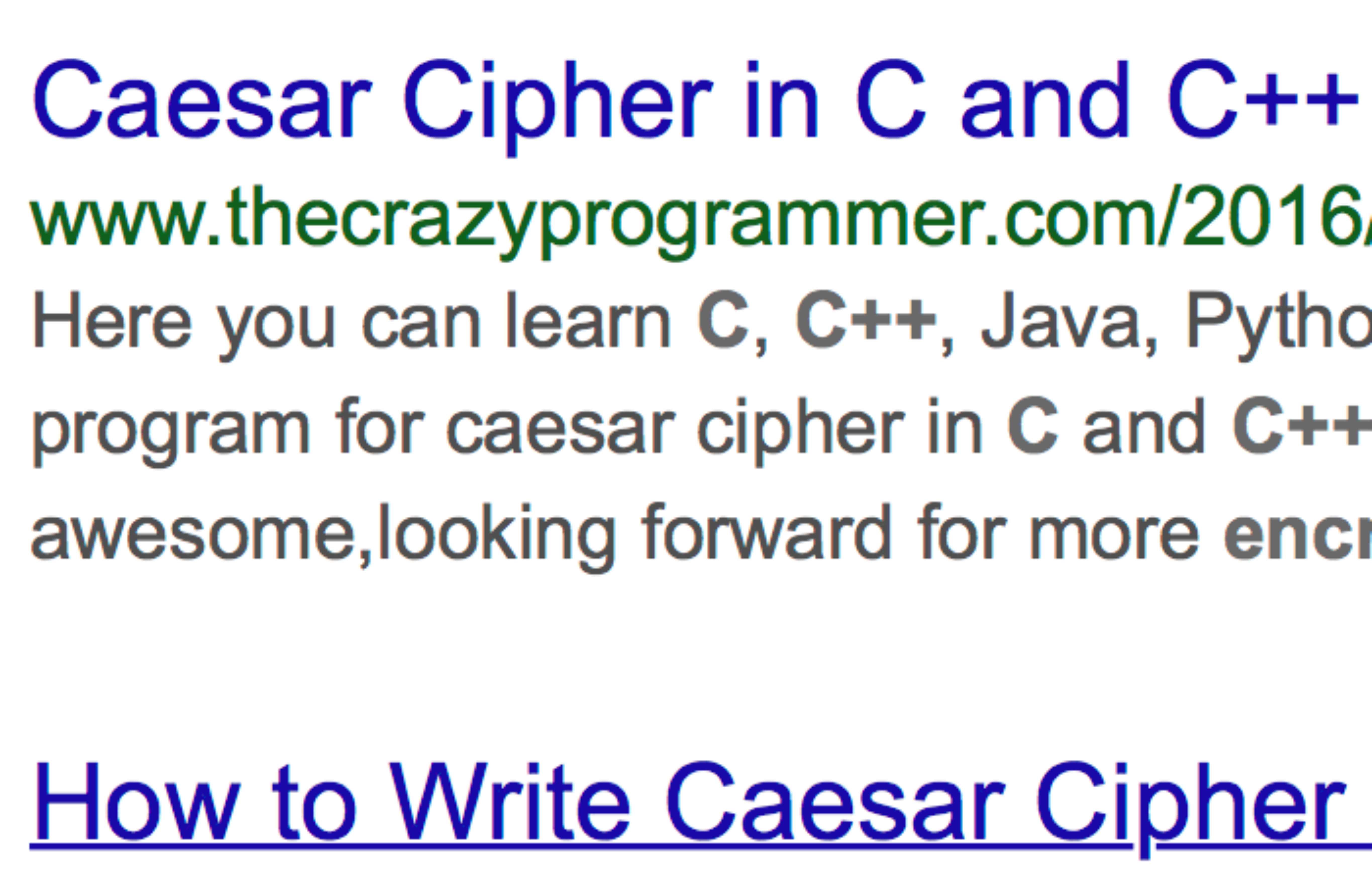
Caesar Cipher in C and C++ [Encryption & Decryption] - The Crazy ...
www.thecrazyprogrammer.com/2016/.../caesar-cipher-c-c-encryption-decryption.htm...
Here you can learn **C**, **C++**, Java, Python, Android Development, PHP, SQL, JavaScript, ... Get program for caesar cipher in **C** and **C++** for **encryption** and decryption. ..... Thanks man ,you're awesome,looking forward for more **encryption stuff**.

How to Write Caesar Cipher in C Program with ... - The Geek Stuff
www.thegeekstuff.com/2014/08/c-caesar-cipher-example/
Aug 7, 2014 - One simple and basic method to **encrypt** a message is using ... you'll learn how to create a **C** program code that will **encrypt** and **decrypt** the text ...



how to encrypt stuff in c

About 2,960,000 results (0.69 seconds)

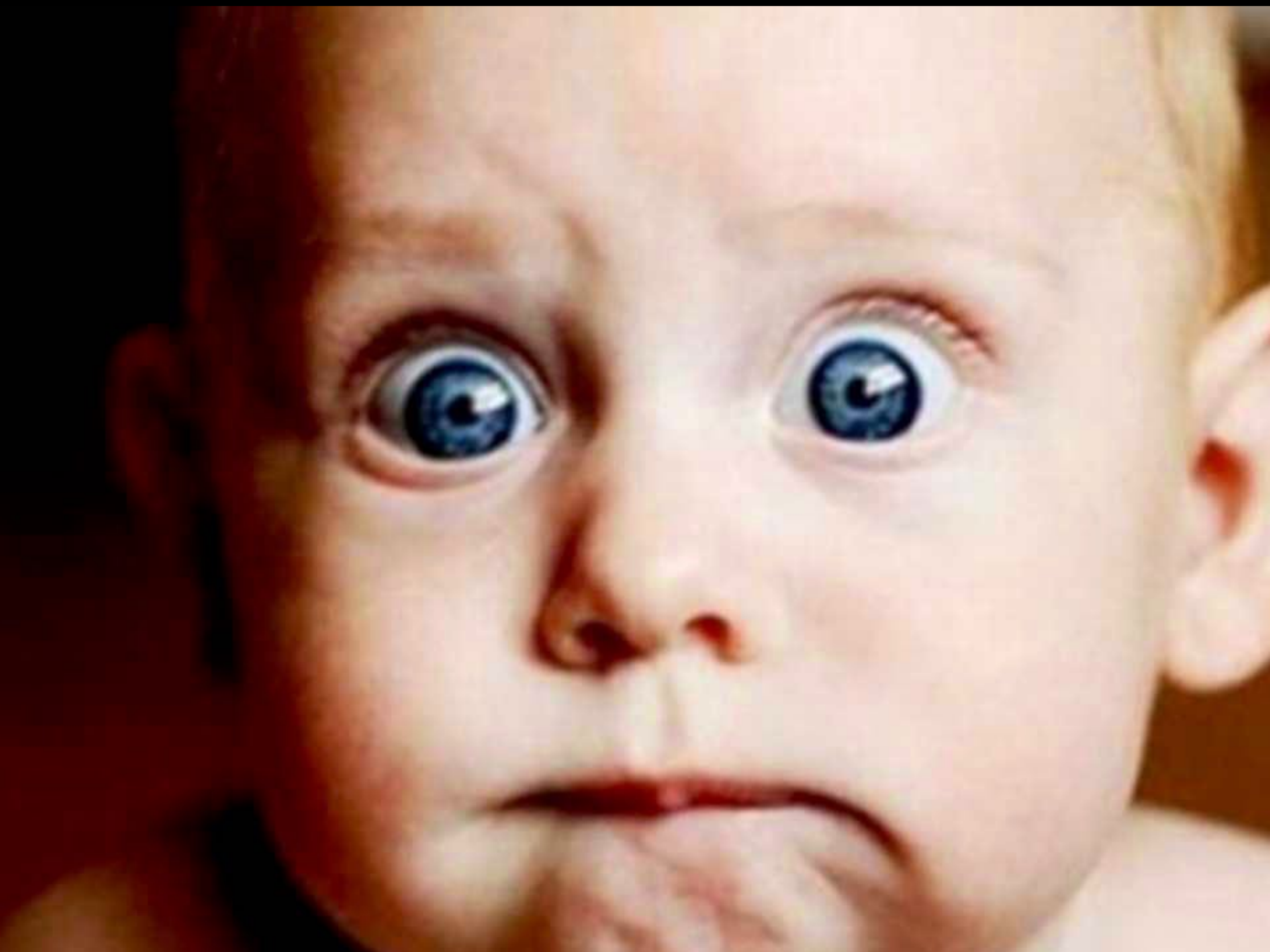


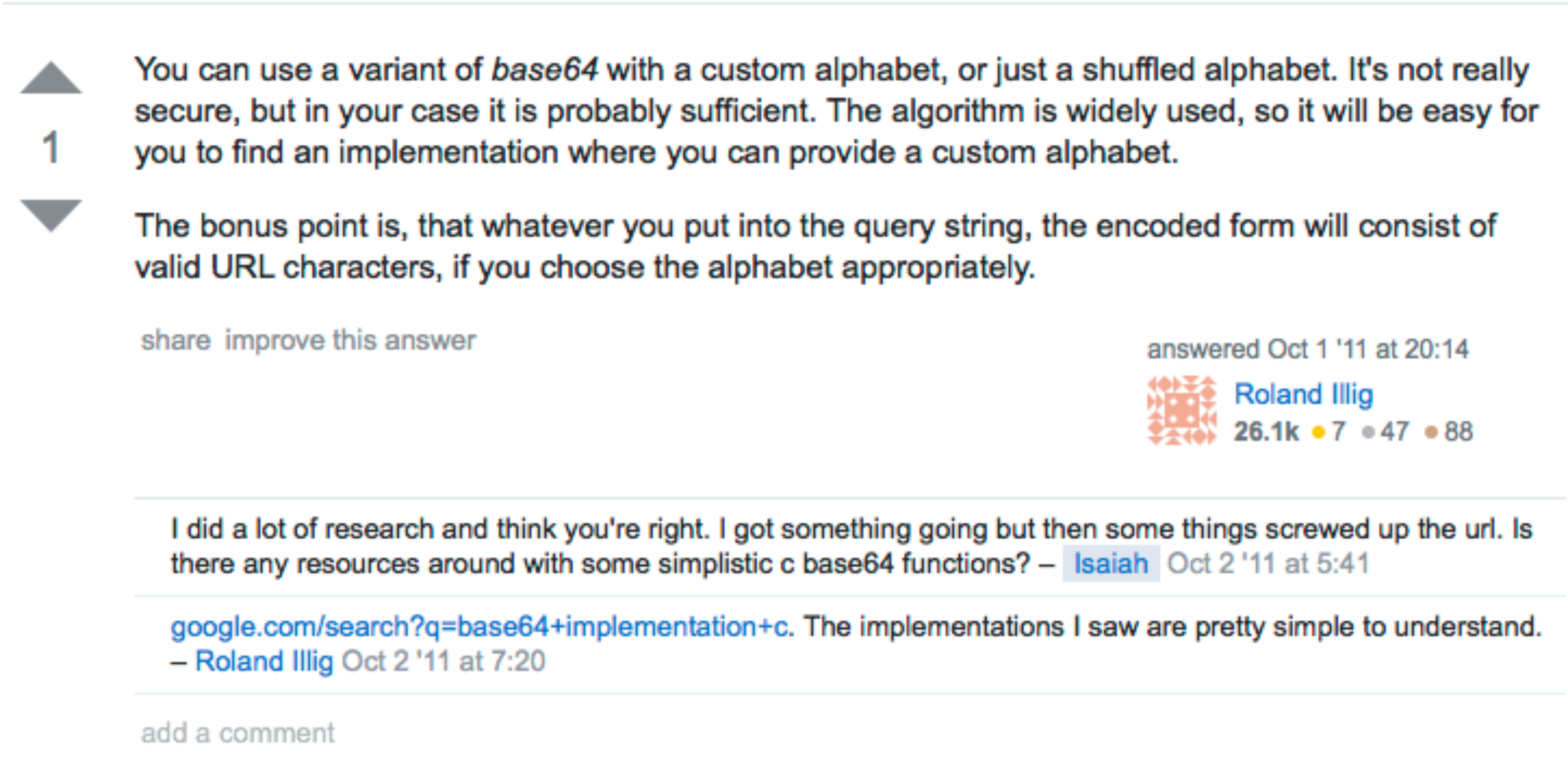
Caesar Cipher in C and C++

www.thecrazyprogrammer.com/2016/

Here you can learn **C**, **C++**, Java, Pytho
program for caesar cipher in **C** and **C++**
awesome,looking forward for more **encr**

How to Write Caesar Cipher





1

You can use a variant of *base64* with a custom alphabet, or just a shuffled alphabet. It's not really secure, but in your case it is probably sufficient. The algorithm is widely used, so it will be easy for you to find an implementation where you can provide a custom alphabet.

The bonus point is, that whatever you put into the query string, the encoded form will consist of valid URL characters, if you choose the alphabet appropriately.

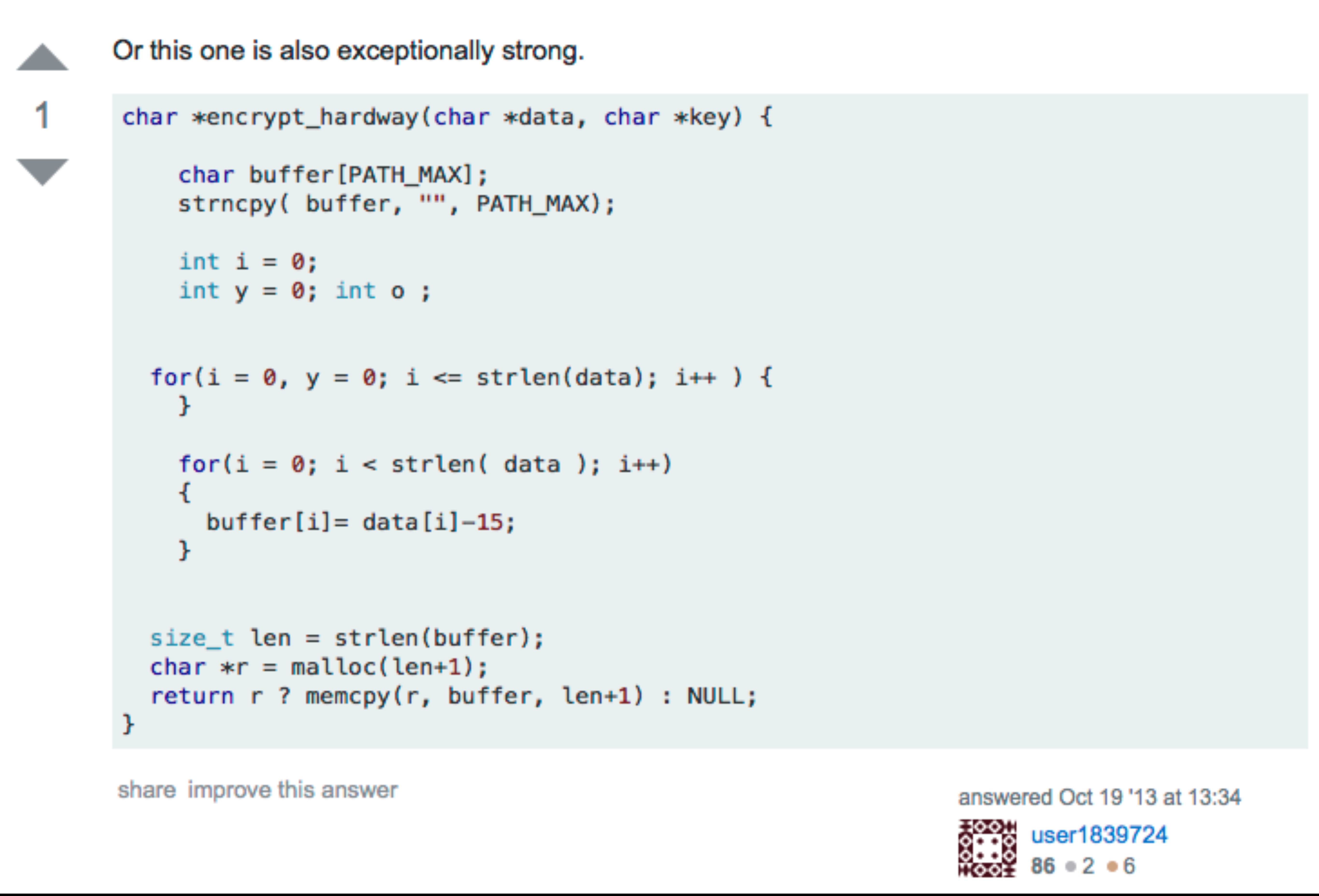
answered Oct 1 '11 at 20:14

Roland Illig

26.1k ● 7 ● 47 ● 88

I did a lot of research and think you're right. I got something going but then some things screwed up the url. Is there any resources around with some simplistic c base64 functions? – Isaiah Oct 2 '11 at 5:41

google.com/search?q=base64+implementation+c. The implementations I saw are pretty simple to understand. – Roland Illig Oct 2 '11 at 7:20



1

Or this one is also exceptionally strong.

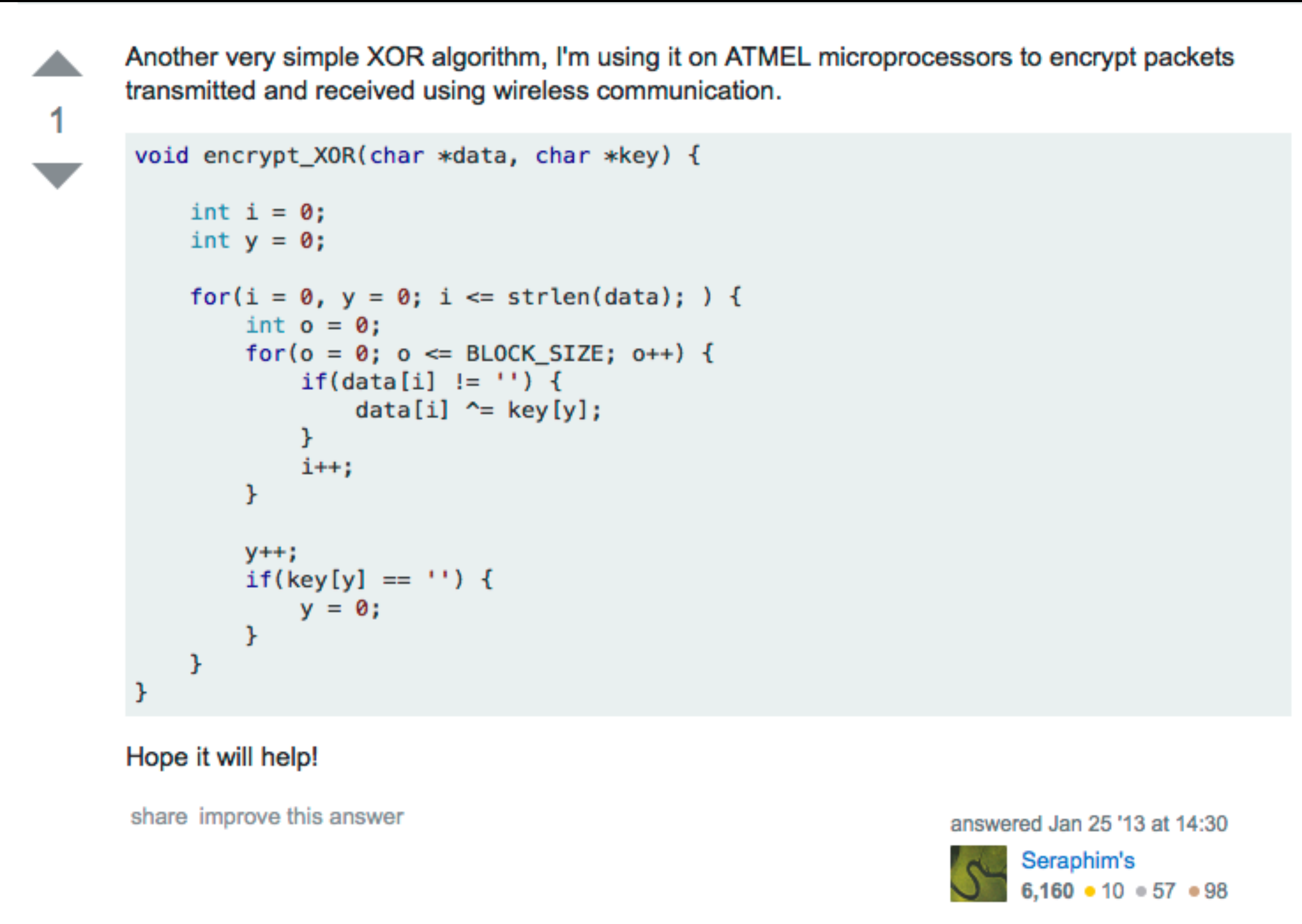
```
char *encrypt_hardway(char *data, char *key) {

    char buffer[PATH_MAX];
    strncpy( buffer, "", PATH_MAX);

    int i = 0;
    int y = 0; int o ;


  for(i = 0, y = 0; i <= strlen(data); i++ ) {
    }

    for(i = 0; i < strlen( data ); i++)
    {
      buffer[i]= data[i]-15;
    }


  size_t len = strlen(buffer);
  char *r = malloc(len+1);
  return r ? memcpy(r, buffer, len+1) : NULL;
}
```

share improve this answer

answered Oct 19 '13 at 13:34

user1839724

86 ● 2 ● 6



Another very simple XOR algorithm, I'm using it on ATMEL microprocessors to encrypt packets transmitted and received using wireless communication.

```
void encrypt_XOR(char *data, char *key) {

    int i = 0;
    int y = 0;

    for(i = 0, y = 0; i <= strlen(data); ) {
        int o = 0;
        for(o = 0; o <= BLOCK_SIZE; o++) {
            if(data[i] != '') {
                data[i] ^= key[y];
            }
            i++;
        }

        y++;
        if(key[y] == '') {
            y = 0;
        }
    }
}
```

Hope it will help!

answered Jan 25 '13 at 14:30

Seraphim's
6,160 ● 10 ● 57 ● 98

Main Entrance
EMERGENCY
Out Patient
Drop Off

RC4

SEED

AES

DES

GOST

Twofish

Camellia

Blowfish

RC6

CAST-128

RC5

IDEA

RC2

3DES

RC4

CCM

OCB

SEED

CFB

AES

DES

GOST

Twofish

EAX

Camellia

Blowfish

CBC

ECB

RC6

OFB

GCM

CAST-128

RC5

IDEA

RC2

CTR

3DES

XTS

RC4 CCM OCB SEED

CFB AES

DES

GOST

56 bits Twofish EAX

192 bits

Camellia

Blowfish

CBC

256 bits ECB

RC6 OFB

GCM CAST-128

RC5

IDEA

128 bits

RC2 CTR 3DES

XTS

RC4

CCM

OCB

SEED

MAC

CFB

AES

DES

GOST

Padding

56 bits

Twofish

EAX

192 bits

Camellia

Blowfish

CBC

256 bits

ECB

RC6

OFB

GCM

CAST-128

RC5

IDEA

128 bits

Yadi

Yada

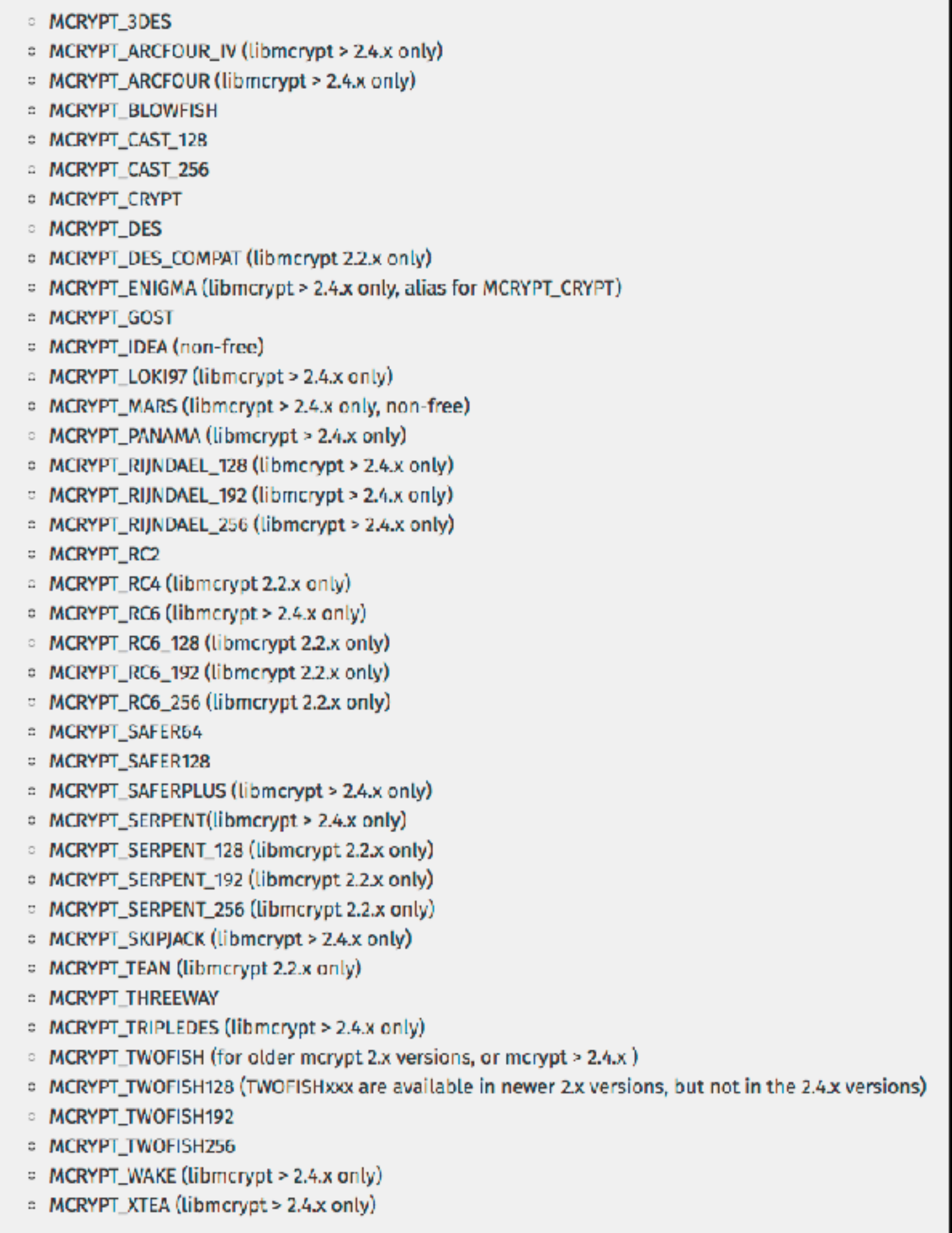
RC2

CTR

3DES

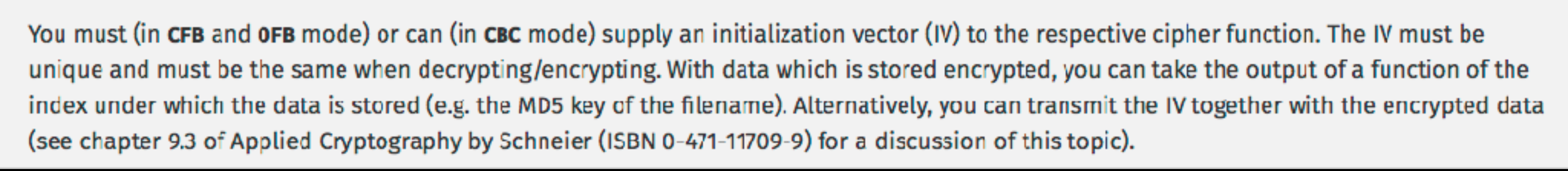
XTS

How to encrypt stuff in PHP?



- MCRYPT_3DES
- MCRYPT_ARCFOUR_IV (libmcrypt > 2.4.x only)
- MCRYPT_ARCFOUR (libmcrypt > 2.4.x only)
- MCRYPT_BLOWFISH
- MCRYPT_CAST_128
- MCRYPT_CAST_256
- MCRYPT_CRYPT
- MCRYPT_DES
- MCRYPT_DES_COMPAT (libmcrypt 2.2.x only)
- MCRYPT_ENIGMA (libmcrypt > 2.4.x only, alias for MCRYPT_CRYPT)
- MCRYPT_GOST
- MCRYPT_IDEA (non-free)
- MCRYPT_LOKI97 (libmcrypt > 2.4.x only)
- MCRYPT_MARS (libmcrypt > 2.4.x only, non-free)
- MCRYPT_PANAMA (libmcrypt > 2.4.x only)
- MCRYPT_RIJNDAEL_128 (libmcrypt > 2.4.x only)
- MCRYPT_RIJNDAEL_192 (libmcrypt > 2.4.x only)
- MCRYPT_RIJNDAEL_256 (libmcrypt > 2.4.x only)
- MCRYPT_RC2
- MCRYPT_RC4 (libmcrypt 2.2.x only)
- MCRYPT_RC6 (libmcrypt > 2.4.x only)
- MCRYPT_RC6_128 (libmcrypt 2.2.x only)
- MCRYPT_RC6_192 (libmcrypt 2.2.x only)
- MCRYPT_RC6_256 (libmcrypt 2.2.x only)
- MCRYPT_SAFER64
- MCRYPT_SAFER128
- MCRYPT_SAFERPLUS (libmcrypt > 2.4.x only)
- MCRYPT_SERPENT(libmcrypt > 2.4.x only)
- MCRYPT_SERPENT_128 (libmcrypt 2.2.x only)
- MCRYPT_SERPENT_192 (libmcrypt 2.2.x only)
- MCRYPT_SERPENT_256 (libmcrypt 2.2.x only)
- MCRYPT_SKIPJACK (libmcrypt > 2.4.x only)
- MCRYPT_TEAN (libmcrypt 2.2.x only)
- MCRYPT_THREEWAY
- MCRYPT_TRIPLEDES (libmcrypt > 2.4.x only)
- MCRYPT_TWOFISH (for older mcrypt 2.x versions, or mcrypt > 2.4.x)
- MCRYPT_TWOFISH128 (TWOFISHxxx are available in newer 2.x versions, but not in the 2.4.x versions)
- MCRYPT_TWOFISH192
- MCRYPT_TWOFISH256
- MCRYPT_WAKE (libmcrypt > 2.4.x only)
- MCRYPT_XTEA (libmcrypt > 2.4.x only)

Reference documentation



You must (in **CFB** and **OFB** mode) or can (in **CBC** mode) supply an initialization vector (IV) to the respective cipher function. The IV must be unique and must be the same when decrypting/encrypting. With data which is stored encrypted, you can take the output of a function of the index under which the data is stored (e.g. the MD5 key of the filename). Alternatively, you can transmit the IV together with the encrypted data (see chapter 9.3 of Applied Cryptography by Schneier (ISBN 0-471-11709-9) for a discussion of this topic).

Crypto is hard

USING crypto is hard, too

This leads to security disasters.

Developers are not to blame

Crypto is often a necessary, but tiny piece in an application

Developers expect things to just work.
Like all other pieces their application depends on.

Webcrypto API

Nooooo...

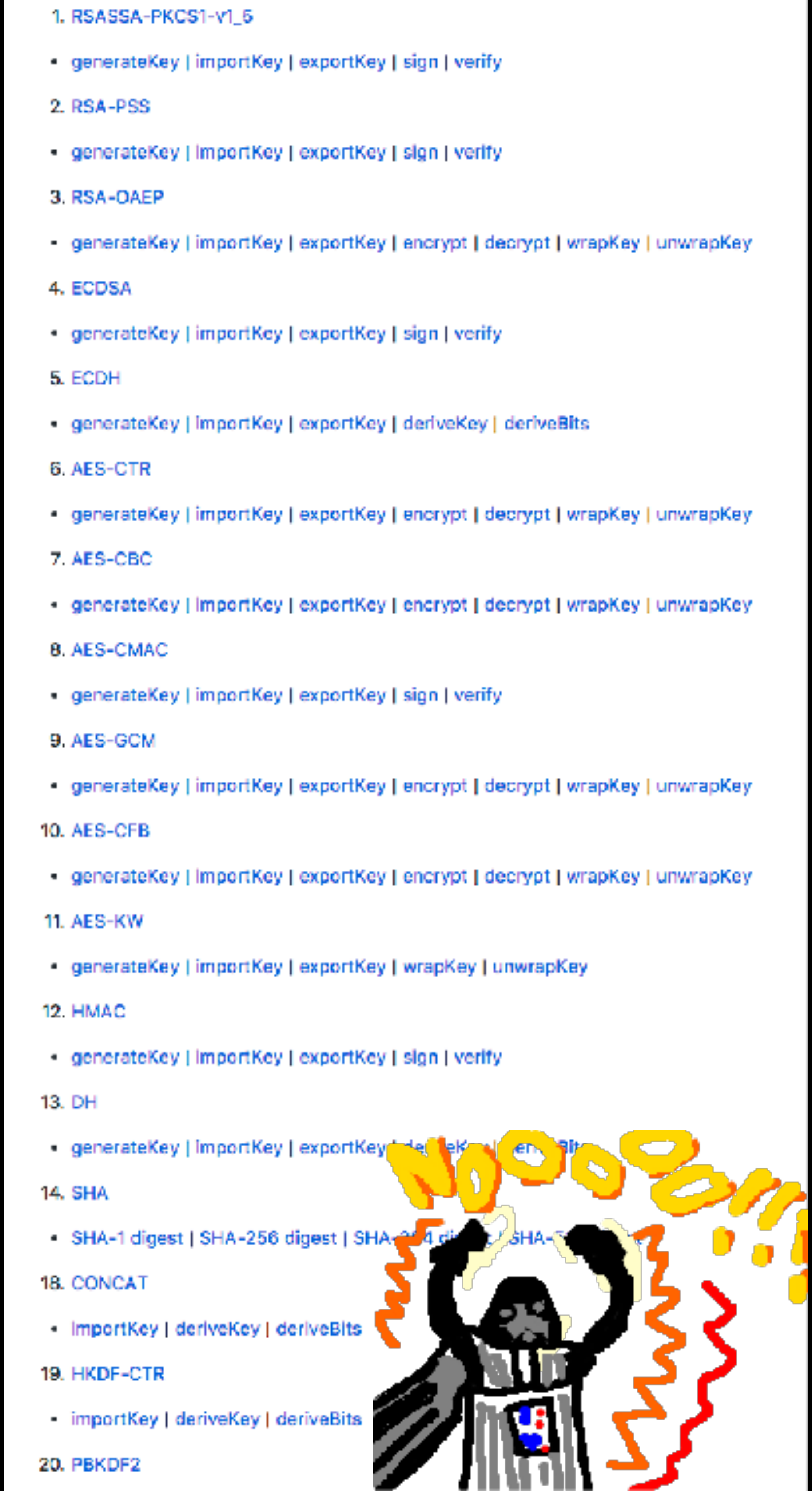
...ooo...

...ooo...

...ooo...

...ooo...

...ooo...



1. RSASSA-PKCS1-v1_5
- generateKey | importKey | exportKey | sign | verify

2. RSA-PSS
- generateKey | importKey | exportKey | sign | verify

3. RSA-OAEP
- generateKey | importKey | exportKey | encrypt | decrypt | wrapKey | unwrapKey

4. ECDSA
- generateKey | importKey | exportKey | sign | verify

5. ECDH
- generateKey | importKey | exportKey | deriveKey | deriveBits

6. AES-CTR
- generateKey | importKey | exportKey | encrypt | decrypt | wrapKey | unwrapKey

7. AES-CBC
- generateKey | importKey | exportKey | encrypt | decrypt | wrapKey | unwrapKey

8. AES-CMAC
- generateKey | importKey | exportKey | sign | verify

9. AES-GCM
- generateKey | importKey | exportKey | encrypt | decrypt | wrapKey | unwrapKey

10. AES-CFB
- generateKey | importKey | exportKey | encrypt | decrypt | wrapKey | unwrapKey

11. AES-KW
- generateKey | importKey | exportKey | wrapKey | unwrapKey

12. HMAC
- generateKey | importKey | exportKey | sign | verify

13. DH
- generateKey | importKey | exportKey | deriveKey | deriveBits

14. SHA
- SHA-1 digest | SHA-256 digest | SHA-384 digest | SHA-5

18. CONCAT
- importKey | deriveKey | deriveBits

19. HKDF-CTR
- importKey | deriveKey | deriveBits

20. PBKDF2

...ooo...

...ooo...

...ooo...

...ooo...

...ooo...

...ooooooo!

NaCl

Funded by the European Commission, released in 2010.

Focused on high-speed cryptography
and improving usability.

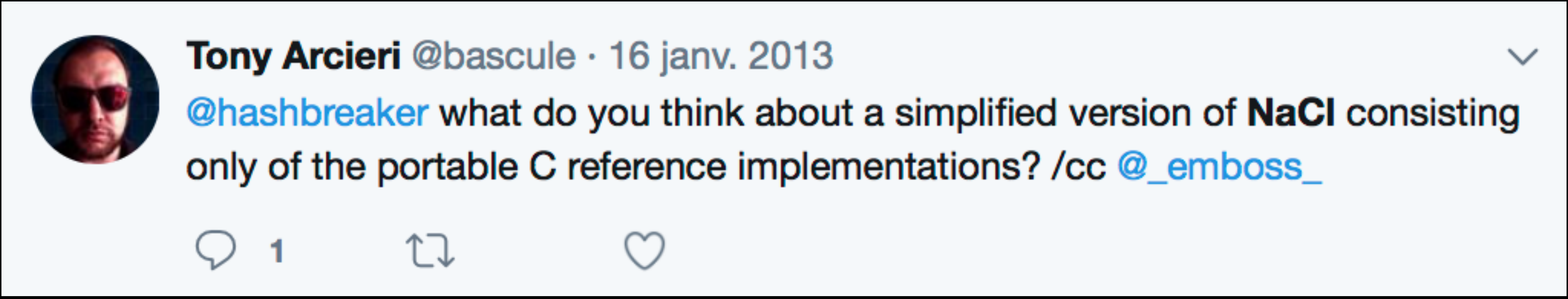
Restricted to a small set of primitives and parameters
chosen by experts

High-level APIs for common operations

Optimized for the host it was compiled on, using tricks of
the C language to save extra CPU cycles

State-of-the-start, simple, highly secure, high-speed cryptography!

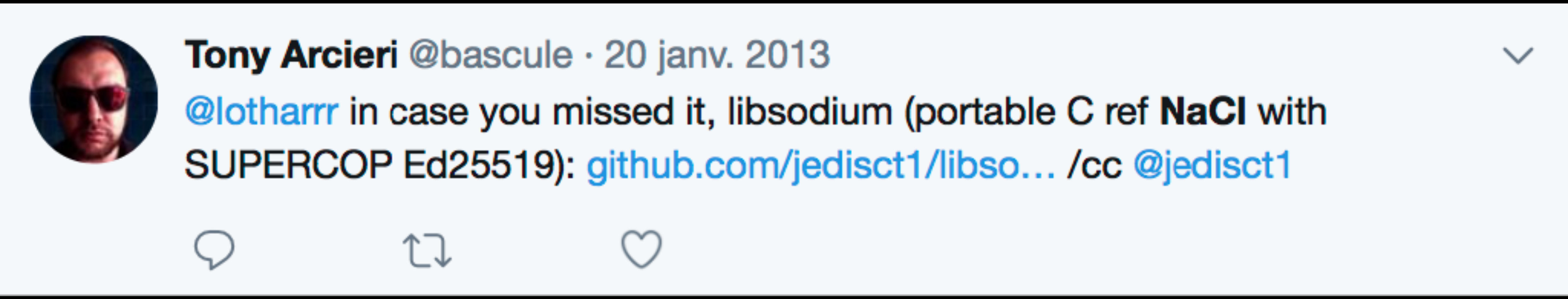
3 years later: adoption rate remains very low



Tony Arcieri @bascule · 16 janv. 2013

@hashbreaker what do you think about a simplified version of **NaCl** consisting only of the portable C reference implementations? /cc @_emboss_

2013: libsodium



Tony Arcieri @bascule · 20 janv. 2013

@lotharrr in case you missed it, libsodium (portable C ref **NaCl** with SUPERCOP Ed25519): github.com/jedisct1/libso… /cc @jedisct1

Warning: this is not a talk about libsodium

Libsodium just happens to be a good case to look at, because its API has evolved a lot over time.

Let’s see why, how,
and some takeaways from the past 4 years

Slow version of NaCl: Instant success!

Usability was the #1 problem to solve in cryptography

Not speed

Not security

¯_(ツ)_/¯

Cryptography makes devices communicate securely.

Cross-platform support is no more an option.

Today’s minimum expectations:

Linux
MacOS
iOS
Android
Windows (Visual Studio)
Embedded systems
Javascript / WebAssembly

Today’s applications are written using a combination of programming languages.

APIs designed for a specific language are problematic.

Macros and pointer arithmetic don’t play well with (not(C | C++))

Expose everything as a function

crypto_box_KEYBYTES -> crypto_box_keybytes()

Package maintainers are your best friends

How developers want to install dependencies today:

pkg_add, apt-get, brew, pacman, choco…

One pre-built, universal package.

Mainstream build systems suck. All of them.

But package maintainers know how to use them.

And adoption of your project depends on package maintainers.

Key idea behind NaCl/libsodium: expose high-level APIs for common operations

"I want to encrypt a message"

"I want to verify that a message hasn't been tampered with"

"I want to store a password"

(and stay cool if my company name ever ends up on haveibeenpwned.com)

Simple functions that keep the amount of user-supplied parameters down to a minimum

```
crypto_box_seal(c, “message”, 7, secret_key)
```

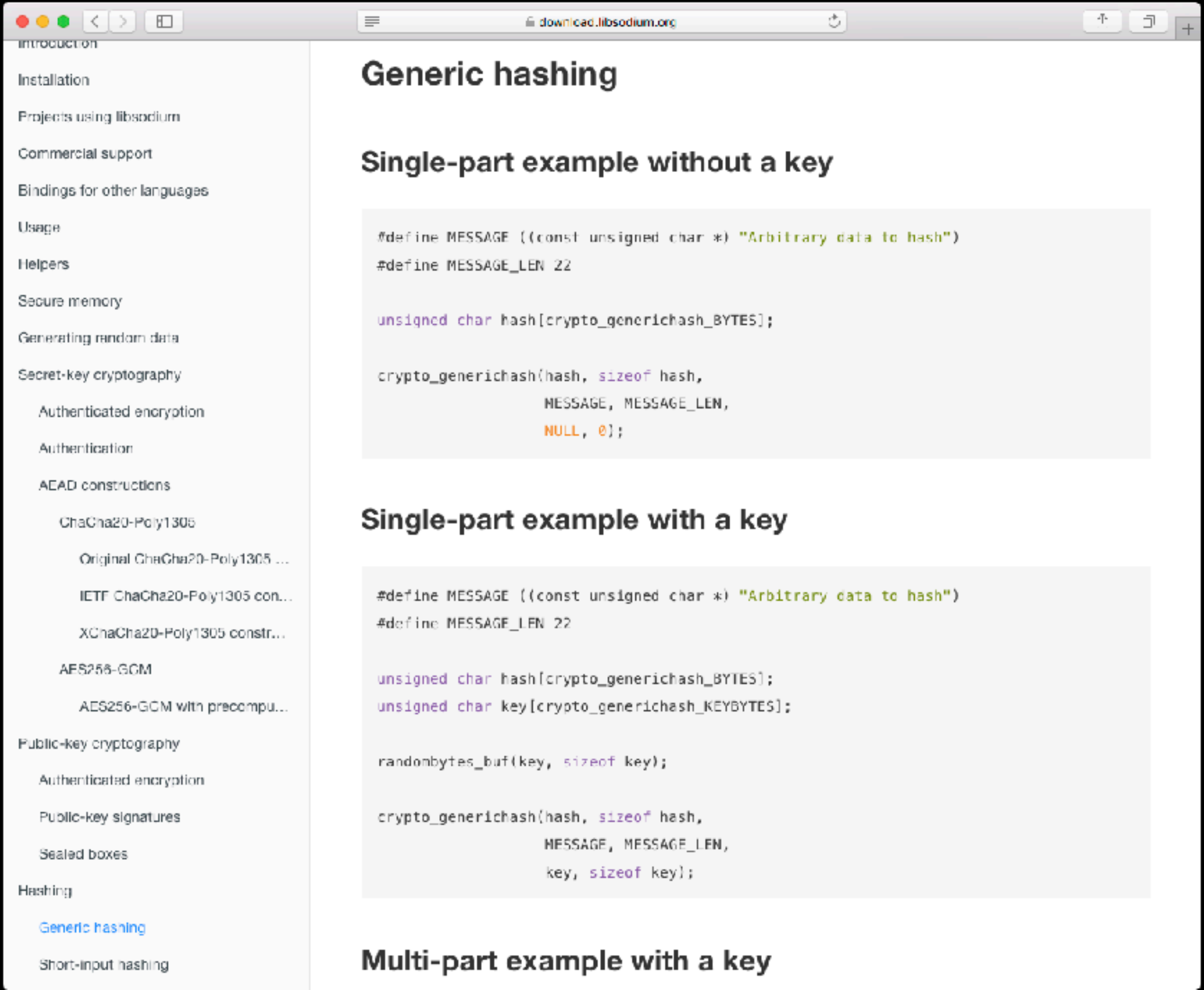
Nobody reads the f* documentation

What experts want: all the gory details about the chosen primitives, constructions and parameters

What everybody else want: example code, code snippets to copy/paste

Also keep in mind that for most people,
a “secret key” means “a password”

Provide examples, *then* explain:



Generic hashing

Single-part example without a key

```
#define MESSAGE ((const unsigned char *) "Arbitrary data to hash")
#define MESSAGE_LEN 22

unsigned char hash[crypto_generichash_BYTES];

crypto_generichash(hash, sizeof hash,
                   MESSAGE, MESSAGE_LEN,
                   NULL, 0);
```

Single-part example with a key

```
#define MESSAGE ((const unsigned char *) "Arbitrary data to hash")
#define MESSAGE_LEN 22

unsigned char hash[crypto_generichash_BYTES];
unsigned char key[crypto_generichash_KEYBYTES];

randombytes_buf(key, sizeof key);

crypto_generichash(hash, sizeof hash,
                   MESSAGE, MESSAGE_LEN,
                   key, sizeof key);
```

Multi-part example with a key

Watch how people use your APIs in their own projects

Watch yourself struggle when using that very API in your own projects

How libraries are used in real-world projects

crypto_box(): everybody writes wrappers.

crypto_sign(): everybody writes wrappers.
Vulnerability in early Golang bindings due to a misunderstanding of the API.

OpenSSL: libtls + a bazillion incompatible abstraction layers in all programming languages. Either close to the metal and dangerous, or completely different from the original API.

If people write wrappers,
your API could be improved

Watch what people are building with your APIs

Watch for recurring questions on Github, Stackoverflow, etc.

If something is not available out of the box, people will reinvent it.

So, implement it.

“It’s only 1 or 2 trivial lines of code, I’m not gonna add yet another set of APIs just for that [very common feature request]”

/me, not so long ago.

Reality check

- Adding a trivial function is not always bloat. It can be well worth it.
- It will improve code clarity, prevent bugs.
- It will save you from having to answer the same questions over and over again.
- It will make users aware that this operation is actually possible.

Libsodium examples

- `crypto_box_keygen()` to create a secret key.
- `crypto_box_seal()` to delete the secret key after encryption.
- `crypto_kdf()` for key derivation.
- `randombytes_deterministic()` for deterministic random numbers.

All of these are small and trivial functions, yet turned out to be welcome additions.

High-level APIs frustrate power users

Expose low-level APIs as well, with access to more parameters.

Documentation should remain focused on high-level APIs.

Do not expose specific implementations,
or you'll be screwed later.

Adding new primitives, new constructions:

Does it solve a common problem impossible to solve with the current APIs?

Adding new operations

Build a distinct project, maintained independently.
Experiment with new APIs. Wait for feedback. Watch how these APIs are being used.

Or if people use them at all.

Look at how people solved similar problems. Tweak the prototype. Use-it in your own apps. Tweak it again.

Eventually, port it to the main project (or not).

Example: blobcrypt

Again:

Watch how people use your APIs in their own projects

Watch yourself struggle when using that very API in your own projects

Nonces (IVs)

Supplement the secret key.

Must be unique for a given key.

The security of most nonce-based ciphers can be totally destroyed if not.

Shall a crypto API require nonces from applications?

Yes:

- Some protocols mandate specific nonces
- Nonces can be used to avoid replay attacks/associate questions with responses in non-pipelined protocols
- Come on, anyone can generate random data and maintain counters!

No:

- Users are too stupid to generate nonces (that’s what “misuse resistance” stands for, right?)

  — Not exactly.

Why "No" should be the answer today:

- Requires redundant code, that APIs could avoid.

- People don't have time to read documentation. Documentation can be misleading or incomplete.

- Maintaining counters is complicated in today's world where apps run in the cloud, in multiple containers sharing the same secret keys.

- Different ciphers have different requirements and security guarantees. Random nonces may not be secure. Ditto for counters. Protocols defining nonce constructions may be broken. APIs should hide these details and do the right thing instead of blaming users for "misuse".

- iOT/embedded systems: safely generating unique/random numbers may not be possible at all.

CVE-2017-13079

CVE-2017-13085

CVE-2017-13086

CVE-2017-13088

CVE-2017-13080

CVE-2017-13081

Krack

CVE-2017-13078

CVE-2017-13083

CVE-2017-13084

CVE-2017-13082

CVE-2017-13087

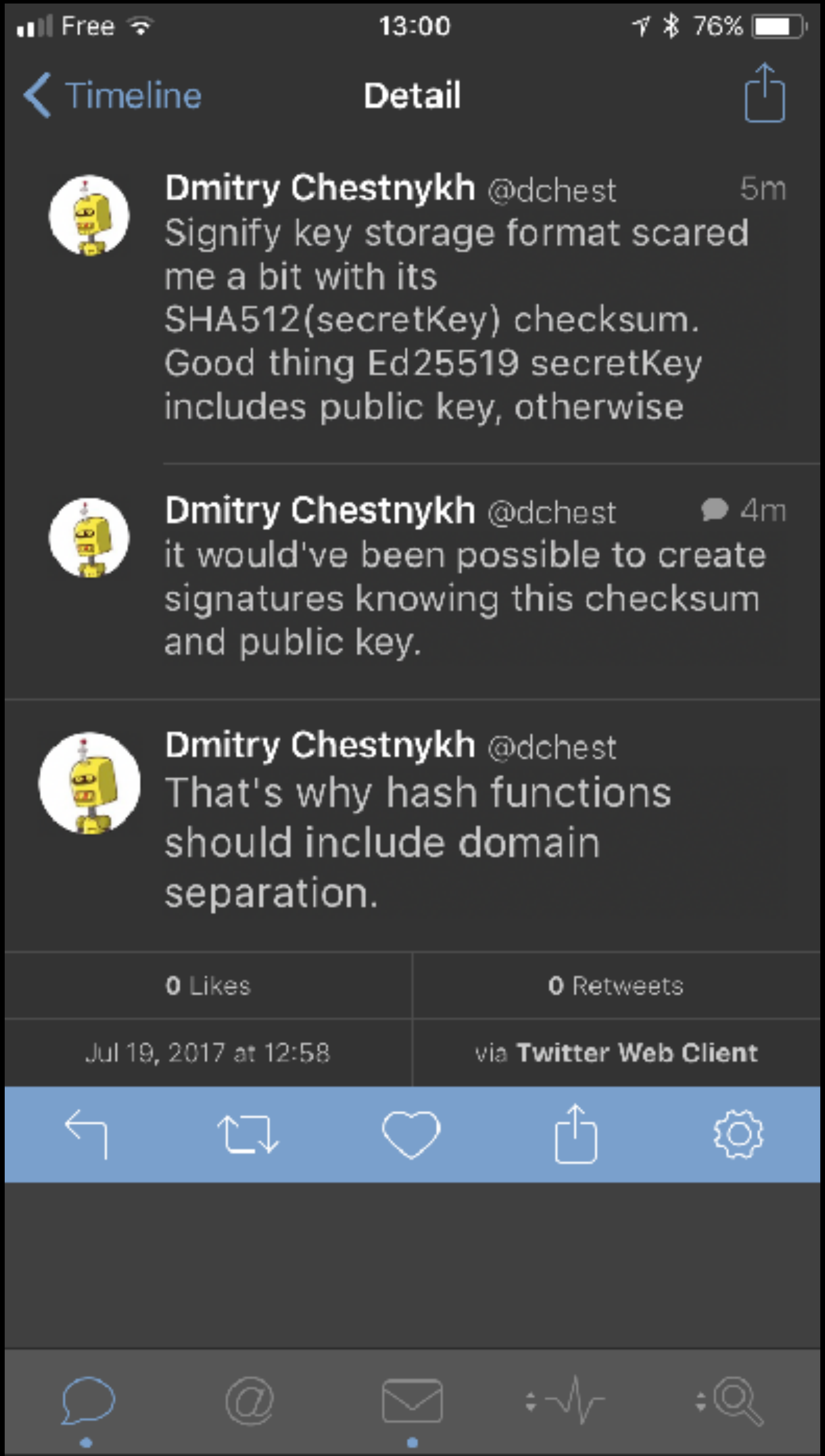
CVE-2017-13077

Context separation

Reusing a secret key for different purposes can have catastrophic implications.

Applications will not do that, right?

It may not be obvious at all:



Dmitry Chestnykh @dchest 5m
Signify key storage format scared me a bit with its SHA512(secretKey) checksum. Good thing Ed25519 secretKey includes public key, otherwise

Dmitry Chestnykh @dchest 4m
it would've been possible to create signatures knowing this checksum and public key.

Dmitry Chestnykh @dchest
That's why hash functions should include domain separation.

0 Likes | 0 Retweets
Jul 19, 2017 at 12:58 | via Twitter Web Client

Shall we blame the developers?

Or could APIs prevent that?

Modern crypto APIs should consider context separation.

As of today, no major library does.

Key exchange

Insufficient: provide a DH function.

Actually worse: provide a DH function + a lot of documentation about how to use it right.

Better in theory: use TLS.

Hell’s kitchen: reimplement a well-known AKE.

Playing with fire: invent a custom protocol.

Juggling with unlocked hand grenades blind-folded: reimplement TLS.

Limitations

No Practical Limitations

(from an API perspective)

Documentation make library developers feel guilt-free,
but doesn’t fix actual problems.

libhydrogen

Started as a lightweight crypto library for microcontrollers/constrained environments.

Also an opportunity to design new APIs based on lessons from the past, and current trends in cryptography.

Key concepts:

- Everything is built upon only two modern cryptographic building blocks: the Gimli permutation and the Curve25519 elliptic curve.
- Concise, consistent, easy-to-use, hard-to-misuse high-level API.
- One key size for all operations.
- Context (domain separation) required by virtually all APIs. One context size for all operations.
- Do not assume that a CSPRNG is available, or works as expected.
- Implement what applications frequently use in other libraries.

A single API for all your hashing needs

HMAC construction
Hash function for short messages
Hash function with 128 bit output
Hash function with 256 bit output
Hash function with 512 bit output
XOF or KDF + stream cipher

→ One generic hashing API

Initial libhydrogen prototype: siphash128 + blake2S + blake2SX
Today: one sponge function

Zero changes to the API

Encryption

Don’t ask applications for a nonce

Automatically attach a synthetic nonce to the ciphertext

“misuse” resistant

Encryption

Why do applications need explicit nonces/AD?

- Check that if we expect the 3rd message in sequence, what we just received actually is the 3rd message.

- Check a message id, to reorder fragmented, unordered messages (e.g. UDP datagrams).

- Check that a message is not older than a given timestamp.

- Check a protocol version.

Encryption

Why do applications need explicit nonces/AD?

- Check that a value attached to a message is the one we expect
- Check that a value attached to a message is the one we expect
- Check that a value attached to a message is the one we expect
- Check that a value attached to a message is the one we expect

From an API perspective: no AD, no nonce, but a 64 bit integer

Encryption

```
hydro_secretbox_keygen(key);

hydro_secretbox_encrypt(ciphertext,
    MESSAGE, MESSAGE_LEN, 1,
    CONTEXT, key);

hydro_secretbox_decrypt(decrypted,
    ciphertext, CIPHERTEXT_LEN, 1,
    CONTEXT, key)
```

Be consistent

HKDF parameters:
hash function, salt, key information.

Salt -> context
Key information -> 64 bit value

One vocabulary, same types used across all the APIs.

Even if the underlying primitives are more flexible, simplify their interface to what most real-world projects actually need.

Key exchange

Protocol independent

Transport independent

Can be extended

Hard to get wrong

Key exchange

Bob:

`hydro_kx_xx1()` -> packet1

Alice:

`hydro_kx_xx2(packet1)` -> packet2

Bob:

`hydro_kx_xx3(packet2)` -> packet3

(Optional) Alice:

`hydro_kx_xx4(packet3)` -> DONE!

Don’t reinvent the wheel

Noise

Noisesocket

Strobe

+ well-studied constructions

Improving security through better abstractions

From:

Many raw crypto primitives and combinators + high level APIs implementing specific protocols

To:

A translation of what primitives can do into what typical applications need. High-level building blocks with a simple, unified interface modeled after real-world use cases.

Requirements: no limitations, MR, domain separation.

Thanks!

Frank Denis
@jedisct1
frank@primulinus.com

https://libsodium.org
https://github.com/jedisct1/libhydrogen